

MapReduce Service

Application Development Guide

Issue 01
Date 2022-11-18



Copyright © Huawei Technologies Co., Ltd. 2022. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

Trademarks and Permissions



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

1 Description.....	1
2 Obtaining Sample Projects from Huawei Mirrors.....	3
3 Using Open-source JAR File Conflict Lists.....	7
3.1 HBase.....	7
3.2 HDFS.....	8
3.3 Kafka.....	8
3.4 Spark2x.....	9
4 Mapping Between Maven Repository JAR Versions and MRS Component Versions	10
5 Security Authentication.....	16
5.1 Security Authentication Principles and Mechanisms.....	16
5.2 Preparing a Developer Account.....	20
5.3 Handling an Authentication Failure.....	26
6 CQL Development Guide (Security Mode).....	28
6.1 Overview.....	28
6.1.1 Application Development Overview.....	28
6.1.2 Common Concepts.....	28
6.1.3 Development Process.....	29
6.2 Environment Preparation.....	30
6.2.1 Environment Preparation Overview.....	30
6.2.2 Preparing for Development and Operating Environment.....	31
6.2.3 Configuring and Importing Sample Projects.....	34
6.3 Developing an Application.....	38
6.3.1 Scenario.....	38
6.3.2 Example Codes.....	38
6.4 Running an Application.....	39
7 CQL Development Guide (Normal Mode).....	43
7.1 Overview.....	43
7.1.1 Application Development Overview.....	43
7.1.2 Common Concepts.....	43
7.1.3 Development Process.....	44

7.2 Environment Preparation.....	45
7.2.1 Environment Preparation Overview.....	46
7.2.2 Preparing for Development and Operating Environment.....	46
7.2.3 Configuring and Importing Sample Projects.....	49
7.3 Developing an Application.....	53
7.3.1 Scenario.....	54
7.3.2 Example Codes.....	54
7.4 Running an Application.....	55
7.4.1 Debugging, Compiling, and Running an Application in Windows.....	55
7.4.2 Debugging and Running an Application in Linux.....	58
7.4.3 Viewing Commissioning Results.....	58
7.5 FAQ.....	60
7.5.1 "Application 'example' already exist" Is Reported When a Sample Application Is Run in Windows...	60
8 ClickHouse Development Guide (Security Mode).....	62
8.1 Overview.....	62
8.1.1 Introduction to ClickHouse.....	62
8.1.2 Basic Concepts.....	63
8.1.3 Development Process.....	63
8.2 Environment Preparations.....	65
8.2.1 Preparing the Development and Operating Environment.....	65
8.2.2 Configuring and Importing a Sample Project.....	68
8.3 Application Development.....	70
8.3.1 Typical Application Scenario.....	70
8.3.2 Development Guideline.....	71
8.4 Sample Code.....	71
8.4.1 Setting Properties.....	71
8.4.2 Establishing a Connection.....	71
8.4.3 Creating a Database.....	72
8.4.4 Creating a Table.....	72
8.4.5 Inserting Data.....	72
8.4.6 Querying Data.....	72
8.4.7 Deleting a Table.....	73
8.5 Application Commissioning.....	73
8.5.1 Commissioning Applications on Windows.....	73
8.5.2 Commissioning Applications on Linux.....	76
9 ClickHouse Development Guide (Normal Mode).....	79
9.1 Overview.....	79
9.1.1 Introduction to ClickHouse.....	79
9.1.2 Basic Concepts.....	80
9.1.3 Development Process.....	80
9.2 Environment Preparations.....	82
9.2.1 Preparing the Development and Operating Environment.....	82

9.2.2 Configuring and Importing a Sample Project.....	85
9.3 Application Development.....	87
9.3.1 Typical Application Scenario.....	87
9.3.2 Development Guideline.....	88
9.4 Sample Code.....	88
9.4.1 Setting Properties.....	88
9.4.2 Establishing a Connection.....	88
9.4.3 Creating a Database.....	89
9.4.4 Creating a Table.....	89
9.4.5 Inserting Data.....	89
9.4.6 Querying Data.....	89
9.4.7 Deleting a Table.....	90
9.5 Application Commissioning.....	90
9.5.1 Commissioning Applications on Windows.....	90
9.5.2 Commissioning Applications on Linux.....	92
10 Flink Development Guide (Security Mode).....	96
10.1 Overview.....	96
10.1.1 Application Development.....	96
10.1.2 Basic Concepts.....	98
10.1.3 Development Process.....	99
10.2 Environment Preparation.....	101
10.2.1 Preparing for Development and Operating Environment.....	101
10.2.2 Configuring and Importing a Sample Project.....	104
10.2.3 Creating a Project (Optional).....	122
10.2.4 Preparing for Security Authentication.....	125
10.3 Developing an Application.....	131
10.3.1 DataStream Application.....	131
10.3.1.1 Scenarios.....	132
10.3.1.2 Java Sample Code.....	133
10.3.1.3 Scala Sample Code.....	135
10.3.2 Interconnecting with Kafka.....	137
10.3.2.1 Scenario Description.....	137
10.3.2.2 Java Sample Code.....	141
10.3.2.3 Scala Sample Code.....	143
10.3.3 Asynchronous Checkpoint Mechanism.....	145
10.3.3.1 Scenarios.....	145
10.3.3.2 Java Sample Code.....	146
10.3.3.3 Scala Sample Code.....	148
10.3.4 Job Pipeline Program.....	151
10.3.4.1 Scenario.....	151
10.3.4.2 Java Sample Code.....	153
10.3.4.3 Scala Sample Code.....	156

10.3.5 Stream SQL Join Program.....	158
10.3.5.1 Scenario.....	158
10.3.5.2 Java Sample Code.....	159
10.4 Debugging the Application.....	161
10.4.1 Compiling and Running the Application.....	161
10.4.2 Viewing the Debugging Result.....	169
10.5 More Information.....	174
10.5.1 Introduction to Common APIs.....	174
10.5.1.1 Java.....	174
10.5.1.2 Scala.....	190
10.5.2 Overview of RESTful APIs.....	204
10.5.3 Overview of Savepoints CLI.....	208
10.5.4 Introduction to Flink Client CLI.....	210
10.5.5 FAQ.....	211
10.5.5.1 Savepoints-related Problems.....	211
10.5.5.2 What If the Chrome Browser Cannot Display the Title.....	212
10.5.5.3 What If the Page Is Displayed Abnormally on Internet Explorer 10/11.....	213
10.5.5.4 What If Checkpoint Is Executed Slowly in RocksDBStateBackend Mode When the Data Amount Is Large.....	213
10.5.5.5 What If yarn-session Start Fails When blob.storage.directory Is Set to /home.....	216
10.5.5.6 Why Does Non-static KafkaPartitioner Class Object Fail to Construct FlinkKafkaProducer010?.....	217
10.5.5.7 When I Use a Newly Created Flink User to Submit Tasks, Why Does the Task Submission Fail and a Message Indicating Insufficient Permission on ZooKeeper Directory Is Displayed?.....	218
10.5.5.8 Why Cannot I Access the Apache Flink Dashboard?.....	218
10.5.5.9 How Do I View the Debugging Information Printed Using System.out.println or Export the Debugging Information to a Specified File?.....	219
10.5.5.10 Incorrect GLIBC Version.....	220
11 Flink Development Guide (Normal Mode).....	222
11.1 Overview.....	222
11.1.1 Application Development.....	222
11.1.2 Basic Concepts.....	224
11.1.3 Development Process.....	225
11.2 Environment Preparation.....	226
11.2.1 Preparing for Development and Operating Environment.....	226
11.2.2 Configuring and Importing a Sample Project.....	229
11.2.3 Creating a Project (Optional).....	249
11.3 Developing an Application.....	252
11.3.1 DataStream Application.....	252
11.3.1.1 Scenarios.....	252
11.3.1.2 Java Sample Code.....	254
11.3.1.3 Scala Sample Code.....	256
11.3.2 Interconnecting with Kafka.....	258
11.3.2.1 Scenarios.....	258

11.3.2.2 Java Sample Code.....	259
11.3.2.3 Scala Sample Code.....	260
11.3.3 Asynchronous Checkpoint Mechanism.....	262
11.3.3.1 Scenarios.....	262
11.3.3.2 Java Sample Code.....	262
11.3.3.3 Scala Sample Code.....	265
11.3.4 Job Pipeline Program.....	267
11.3.4.1 Scenario.....	267
11.3.4.2 Java Sample Code.....	270
11.3.4.3 Scala Sample Code.....	272
11.3.5 Stream SQL Join Program.....	274
11.3.5.1 Scenario.....	274
11.3.5.2 Java Sample Code.....	275
11.3.6 Interconnecting Flink with Cloud Search Service.....	277
11.3.6.1 Scenario Description.....	277
11.3.6.2 Java Sample Code.....	278
11.4 Debugging the Application.....	279
11.4.1 Compiling and Running the Application.....	279
11.4.2 Viewing the Debugging Result.....	286
11.5 More Information.....	290
11.5.1 Introduction to Common APIs.....	290
11.5.1.1 Java.....	290
11.5.1.2 Scala.....	307
11.5.2 Overview of RESTful APIs.....	321
11.5.3 Overview of Savepoints CLI.....	325
11.5.4 Introduction to Flink Client CLI.....	327
11.5.5 FAQ.....	328
11.5.5.1 Savepoints-related Problems.....	328
11.5.5.2 What If the Chrome Browser Cannot Display the Title.....	329
11.5.5.3 What If the Page Is Displayed Abnormally on Internet Explorer 10/11.....	330
11.5.5.4 What If Checkpoint Is Executed Slowly in RocksDBStateBackend Mode When the Data Amount Is Large.....	330
11.5.5.5 What If yarn-session Start Fails When blob.storage.directory Is Set to /home.....	333
11.5.5.6 Why Does Non-static KafkaPartitioner Class Object Fail to Construct FlinkKafkaProducer010?.....	334
11.5.5.7 When I Use a Newly Created Flink User to Submit Tasks, Why Does the Task Submission Fail and a Message Indicating Insufficient Permission on ZooKeeper Directory Is Displayed?.....	335
11.5.5.8 Why Cannot I Access the Apache Flink Dashboard?.....	335
11.5.5.9 How Do I View the Debugging Information Printed Using System.out.println or Export the Debugging Information to a Specified File?.....	336
11.5.5.10 Incorrect GLIBC Version.....	337
12 HBase Development Guide (Security Mode).....	339
12.1 Overview.....	339
12.1.1 Application Development Overview.....	339

12.1.2 Common Concepts.....	340
12.1.3 Development Process.....	340
12.2 Environment Preparation.....	342
12.2.1 Preparing for Development and Operating Environment.....	342
12.2.2 Configuring and Importing Sample Projects.....	346
12.2.3 Preparing for Security Authentication.....	355
12.2.3.1 Preparing Authentication Mechanism Code.....	355
12.2.3.2 Multi-Instance Authentication in Mutual Trust Scenarios.....	356
12.2.3.3 Authentication for accessing the HBase REST Service.....	358
12.2.3.4 Authentication for Accessing the ThriftServer Service.....	359
12.2.3.5 Authentication for Accessing Multiple ZooKeepers.....	361
12.3 Developing an Application.....	362
12.3.1 Typical Scenario Description.....	362
12.3.2 Development Idea.....	364
12.3.3 Example Code Description.....	364
12.3.3.1 Configuring Log4j Log Output.....	364
12.3.3.2 Creating Configuration.....	365
12.3.3.3 Creating Connection.....	365
12.3.3.4 Creating a Table.....	366
12.3.3.5 Deleting a Table.....	368
12.3.3.6 Modifying a Table.....	368
12.3.3.7 Inserting Data.....	369
12.3.3.8 Deleting Data.....	371
12.3.3.9 Reading Data Using Get.....	372
12.3.3.10 Reading Data Using Scan.....	373
12.3.3.11 Filtering Data.....	374
12.3.3.12 Creating a Secondary Index.....	375
12.3.3.13 Deleting an Index.....	377
12.3.3.14 Secondary Index-based Query.....	378
12.3.3.15 Multi-Point Region Division.....	381
12.3.3.16 Creating a Phoenix Table.....	382
12.3.3.17 Writing Data to the PhoenixTable.....	383
12.3.3.18 Reading the PhoenixTable.....	383
12.3.3.19 Accessing Multiple ZooKeepers.....	384
12.3.3.20 Querying Cluster Information Using REST.....	386
12.3.3.21 Obtaining All Tables Using REST.....	386
12.3.3.22 Operate Namespaces Using REST.....	386
12.3.3.23 Operate Tables Using REST.....	388
12.3.3.24 Accessing the ThriftServer Operation Table.....	389
12.3.3.25 Accessing ThriftServer to Write Data.....	390
12.3.3.26 Accessing ThriftServer to Read Data.....	391
12.3.3.27 Using HBase Dual-Read.....	393

12.4 Application Commissioning.....	395
12.4.1 Commissioning an Application in Windows.....	395
12.4.1.1 Compiling and Running an Application.....	395
12.4.1.2 Viewing Windows Commissioning Results.....	400
12.4.2 Commissioning an Application in Linux.....	401
12.4.2.1 Compiling and Running an Application When a Client Is Installed.....	401
12.4.2.2 Compiling and Running an Application When No Client Is Installed.....	404
12.4.2.3 Viewing Linux Commissioning Results.....	405
12.5 More Information.....	405
12.5.1 SQL Query.....	405
12.5.2 HBase Dual-Read Configuration Items.....	407
12.5.3 External Interfaces.....	411
12.5.3.1 Shell.....	411
12.5.3.2 Java API.....	412
12.5.3.3 SQLLine.....	416
12.5.3.4 JDBC APIs.....	422
12.5.3.5 WebUI.....	423
12.5.4 HBase Access Configuration on Windows Using EIPs.....	426
12.5.5 Phoenix Command Line.....	428
12.5.6 FAQs.....	429
12.5.6.1 How to Rectify the Fault When an Exception Occurs During the Running of an HBase-developed Application and "org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory" Is Displayed in the Error Information?.....	429
12.5.6.2 What Are the Application Scenarios of the Bulkload and put Data-loading Modes?.....	429
12.5.6.3 An Error Occurred When Building a JAR Package.....	430
13 HBase Development Guide (Normal Mode).....	432
13.1 Overview.....	432
13.1.1 Application Development Overview.....	432
13.1.2 Common Concepts.....	433
13.1.3 Development Process.....	433
13.2 Environment Preparation.....	435
13.2.1 Preparing for Development and Operating Environment.....	435
13.2.2 Configuring and Importing Sample Projects.....	438
13.3 Developing an Application.....	448
13.3.1 Typical Scenario Description.....	448
13.3.2 Development Idea.....	449
13.3.3 Example Code Description.....	450
13.3.3.1 Configuring Log4j Log Output.....	450
13.3.3.2 Creating Configuration.....	451
13.3.3.3 Creating Connection.....	451
13.3.3.4 Creating a Table.....	452
13.3.3.5 Deleting a Table.....	454
13.3.3.6 Modifying a Table.....	454

13.3.3.7 Inserting Data.....	455
13.3.3.8 Deleting Data.....	457
13.3.3.9 Reading Data Using Get.....	458
13.3.3.10 Reading Data Using Scan.....	458
13.3.3.11 Filtering Data.....	460
13.3.3.12 Creating a Secondary Index.....	461
13.3.3.13 Deleting an Index.....	463
13.3.3.14 Secondary Index-based Query.....	464
13.3.3.15 Multi-Point Region Division.....	467
13.3.3.16 Creating a Phoenix Table.....	468
13.3.3.17 Writing Data to the PhoenixTable.....	469
13.3.3.18 Reading the PhoenixTable.....	469
13.3.3.19 Accessing Multiple ZooKeepers.....	470
13.3.3.20 Querying Cluster Information Using REST.....	472
13.3.3.21 Obtaining All Tables Using REST.....	473
13.3.3.22 Operate Namespaces Using REST.....	473
13.3.3.23 Operate Tables Using REST.....	474
13.3.3.24 Accessing the ThriftServer Operation Table.....	475
13.3.3.25 Accessing ThriftServer to Write Data.....	478
13.3.3.26 Accessing ThriftServer to Read Data.....	479
13.3.3.27 Using HBase Dual-Read.....	480
13.4 Application Commissioning.....	482
13.4.1 Commissioning an Application in Windows.....	482
13.4.1.1 Compiling and Running an Application.....	483
13.4.1.2 Viewing Windows Commissioning Results.....	487
13.4.2 Commissioning an Application in Linux.....	488
13.4.2.1 Compiling and Running an Application When a Client Is Installed.....	488
13.4.2.2 Compiling and Running an Application When No Client Is Installed.....	491
13.4.2.3 Viewing Linux Commissioning Results.....	492
13.5 More Information.....	493
13.5.1 SQL Query.....	493
13.5.2 HBase Dual-Read Configuration Items.....	494
13.5.3 External Interfaces.....	498
13.5.3.1 Shell.....	498
13.5.3.2 Java APIs.....	499
13.5.3.3 SQLLine.....	503
13.5.3.4 JDBC APIs.....	510
13.5.3.5 WebUI.....	510
13.5.4 HBase Access Configuration on Windows Using EIPs.....	513
13.5.5 Phoenix Command Line.....	515
13.5.6 FAQs.....	516

13.5.6.1 How to Rectify the Fault When an Exception Occurs During the Running of an HBase-developed Application and "org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory" Is Displayed in the Error Information?.....	516
13.5.6.2 What Are the Application Scenarios of the bulkload and put Data-loading Modes?.....	516
13.5.6.3 An Error Occurred When Building a JAR Package.....	517
14 HDFS Development Guide (Security Mode).....	518
14.1 Overview.....	518
14.1.1 Introduction to HDFS.....	518
14.1.2 Basic Concepts.....	518
14.1.3 Development Process.....	520
14.2 Environment Preparation.....	521
14.2.1 Preparing Development and Operating Environment.....	521
14.2.2 Configuring and Importing Sample Projects.....	525
14.2.3 Preparing the Authentication Mechanism.....	531
14.3 Developing the Project.....	533
14.3.1 Scenario.....	533
14.3.2 Development Idea.....	534
14.3.3 Declare the Example Codes.....	534
14.3.3.1 Initializing the HDFS.....	534
14.3.3.2 Creating Directories.....	537
14.3.3.3 Writing Data into a File.....	537
14.3.3.4 Appending Data to a File.....	538
14.3.3.5 Reading Data from a File.....	539
14.3.3.6 Deleting a File.....	540
14.3.3.7 Deleting Directories.....	540
14.3.3.8 Multi-Thread Tasks.....	541
14.3.3.9 Setting Storage Policies.....	542
14.3.3.10 Colocation.....	543
14.4 Commissioning the Application.....	546
14.4.1 Commissioning an Application in the Windows Environment.....	546
14.4.1.1 Compiling and Running an Application.....	546
14.4.1.2 Checking the Commissioning Result.....	548
14.4.2 Commissioning an Application in the Linux Environment.....	549
14.4.2.1 Compiling and Running an Application With the Client Installed.....	549
14.4.2.2 Compiling and Running an Application With the Client Not Installed.....	550
14.4.2.3 Checking the Commissioning Result.....	552
14.5 More Information.....	553
14.5.1 Common API Introduction.....	553
14.5.1.1 Java API.....	553
14.5.1.2 C API.....	560
14.5.1.3 HTTP REST API.....	565
14.5.2 Shell Command Introduce.....	574
14.5.3 HDFS Access Configuration on Windows Using EIPs.....	576

15 HDFS Development Guide (Normal Mode)	580
15.1 Overview.....	580
15.1.1 Introduction to HDFS.....	580
15.1.2 Basic Concepts.....	580
15.1.3 Development Process.....	582
15.2 Environment Preparation.....	583
15.2.1 Development and Operating Environment.....	583
15.2.2 Configuring and Importing Sample Projects.....	586
15.3 Developing the Project.....	593
15.3.1 Scenario.....	593
15.3.2 Development Idea.....	594
15.3.3 Declare the Example Codes.....	594
15.3.3.1 Initializing the HDFS.....	594
15.3.3.2 Creating Directories.....	596
15.3.3.3 Writing Data into a File.....	597
15.3.3.4 Appending Data to a File.....	597
15.3.3.5 Reading Data from a File.....	598
15.3.3.6 Deleting a File.....	599
15.3.3.7 Deleting Directories.....	599
15.3.3.8 Multi-Thread Tasks.....	600
15.3.3.9 Setting Storage Policies.....	601
15.3.3.10 Colocation.....	602
15.4 Commissioning the Application.....	605
15.4.1 Commissioning an Application in the Windows Environment.....	605
15.4.1.1 Compiling and Running an Application.....	605
15.4.1.2 Checking the Commissioning Result.....	607
15.4.2 Commissioning an Application in the Linux Environment.....	608
15.4.2.1 Compiling and Running an Application with the Client Installed.....	608
15.4.2.2 Compiling and Running an Application with the Client Not Installed.....	609
15.4.2.3 Checking the Commissioning Result.....	611
15.5 More Information.....	612
15.5.1 Common API Introduction.....	612
15.5.1.1 Java API.....	612
15.5.1.2 C API.....	616
15.5.1.3 HTTP REST API.....	620
15.5.2 Shell Command Introduce.....	625
15.5.3 HDFS Access Configuration on Windows Using EIPs.....	627
16 Hive Development Guide (Security Mode)	631
16.1 Overview.....	631
16.1.1 Application Development Overview.....	631
16.1.2 Common Concepts.....	632
16.1.3 Required Permissions.....	632

16.1.4 Development Process.....	636
16.2 Preparing the Environment.....	637
16.2.1 Preparing Development and Operating Environment.....	637
16.2.2 Configuring the JDBC Sample Project.....	642
16.2.3 Configuring the Hcatalog Sample Project.....	645
16.2.4 Configuring the Python Sample Project.....	649
16.2.5 Configuring the Python3 Sample Project.....	650
16.3 Developing an Application.....	651
16.3.1 Typical Scenario Description.....	651
16.3.2 Example Codes.....	654
16.3.2.1 Creating a Table.....	654
16.3.2.2 Loading Data.....	655
16.3.2.3 Querying Data.....	656
16.3.2.4 UDF.....	657
16.3.2.5 Example Program Guide.....	659
16.3.2.6 Accessing Multiple ZooKeepers.....	664
16.4 Commissioning Applications.....	665
16.4.1 Running JDBC and Viewing Results.....	665
16.4.2 Running HCatalog and Viewing Results.....	667
16.4.3 Running Python and Viewing Results.....	669
16.4.4 Running Python3 and Viewing Results.....	670
16.5 More Information.....	670
16.5.1 Interface Reference.....	670
16.5.1.1 JDBC.....	670
16.5.1.2 Hive SQL.....	671
16.5.1.3 WebHCat.....	673
16.5.2 Hive Access Configuration on Windows Using EIPs.....	697
16.5.3 FAQ.....	699
16.5.3.1 A Message Is Displayed Stating "Unable to read HiveServer2 configs from ZooKeeper" During the Use of the Secondary Development Program.....	699
16.5.3.2 Problem performing GSS wrap Message Is Displayed Due to IBM JDK Exceptions.....	700
16.5.3.3 Hive SQL Is Incompatible with SQL2003 Standards.....	700
17 Hive Development Guide (Normal Mode).....	705
17.1 Overview.....	705
17.1.1 Application Development Overview.....	705
17.1.2 Common Concepts.....	706
17.1.3 Development Process.....	706
17.2 Preparing the Environment.....	707
17.2.1 Preparing Development and Operating Environment.....	707
17.2.2 Configuring the JDBC Sample Project.....	711
17.2.3 Configuring the Hcatalog Sample Project.....	715
17.2.4 Configuring the Python Sample Project.....	718

17.2.5 Configuring the Python3 Sample Project.....	719
17.3 Developing an Application.....	721
17.3.1 Typical Scenario Description.....	721
17.3.2 Example Codes.....	723
17.3.2.1 Creating a Table.....	723
17.3.2.2 Loading Data.....	725
17.3.2.3 Querying Data.....	726
17.3.2.4 UDF.....	726
17.3.2.5 Example Program Guide.....	728
17.3.2.6 Accessing Multiple ZooKeepers.....	733
17.4 Commissioning Applications.....	733
17.4.1 Running JDBC and Viewing Results.....	734
17.4.2 Running HCatalog and Viewing Results.....	737
17.4.3 Running Python and Viewing Results.....	740
17.4.4 Running Python3 and Viewing Results.....	741
17.5 More Information.....	742
17.5.1 Interface Reference.....	742
17.5.1.1 JDBC.....	742
17.5.1.2 Hive SQL.....	742
17.5.1.3 WebHCat.....	745
17.5.2 Hive Access Configuration on Windows Using EIPs.....	770
17.5.3 FAQ.....	772
17.5.3.1 Problem performing GSS wrap Message Is Displayed Due to IBM JDK Exceptions.....	772
18 Impala Development Guide (Security Mode).....	773
18.1 Overview.....	773
18.1.1 Application Development Overview.....	773
18.1.2 Basic Concepts.....	774
18.1.3 Application Development Process.....	774
18.2 Environment Preparation.....	776
18.2.1 Preparing Development and Operating Environment.....	776
18.3 Application Development.....	777
18.3.1 Typical Application Scenario.....	777
18.3.2 Creating a Table.....	779
18.3.3 Loading Data.....	781
18.3.4 Querying Data.....	782
18.3.5 User-defined Functions.....	782
18.3.6 Sample Program Guide.....	783
18.4 Application Commissioning.....	785
18.4.1 Commissioning Applications on Windows.....	785
18.4.2 Commissioning Applications on Linux.....	786
18.5 Impala APIs.....	787
18.5.1 JDBC.....	787

18.5.2 Impala SQL.....	787
18.6 Development Specifications.....	787
18.6.1 Rules.....	787
18.6.2 Suggestions.....	789
18.6.3 Examples.....	790
19 Impala Development Guide (Normal Mode).....	793
19.1 Overview.....	793
19.1.1 Application Development Overview.....	793
19.1.2 Basic Concepts.....	794
19.1.3 Application Development Process.....	794
19.2 Environment Preparation.....	796
19.2.1 Preparing Development and Operating Environment.....	796
19.2.2 Configuring and Importing a Sample Project.....	797
19.3 Application Development.....	798
19.3.1 Typical Application Scenario.....	799
19.3.2 Creating a Table.....	800
19.3.3 Loading Data.....	802
19.3.4 Querying Data.....	803
19.3.5 User-defined Functions.....	803
19.3.6 Sample Program Guide.....	805
19.4 Application Commissioning.....	806
19.4.1 Commissioning Applications on Windows.....	807
19.4.2 Commissioning Applications on Linux.....	808
19.5 Impala APIs.....	810
19.5.1 JDBC.....	811
19.5.2 Impala SQL.....	811
19.6 Development Specifications.....	811
19.6.1 Rules.....	811
19.6.2 Suggestions.....	813
19.6.3 Examples.....	813
20 Kafka Development Guide (Security Mode).....	816
20.1 Overview.....	816
20.1.1 Development Environment Preparation.....	816
20.1.2 Common Concepts.....	816
20.1.3 Development Process.....	817
20.2 Environment Preparation.....	819
20.2.1 Preparing for Development and Operating Environment.....	820
20.2.2 Configuring and Importing a Sample Project.....	823
20.2.3 Preparing for Security Authentication.....	827
20.2.3.1 Sasl Kerberos authentication.....	827
20.3 Developing an Application.....	828
20.3.1 Typical Scenario Description.....	828

20.3.2 Typical Scenario Sample Code Description.....	828
20.3.2.1 Producer API Usage Sample.....	828
20.3.2.2 Consumer API Usage Sample.....	829
20.3.2.3 Multi-thread Producer Sample.....	830
20.3.2.4 Multi-thread Consumer Sample.....	830
20.3.3 Kafka Streams Scenario Description.....	831
20.3.4 Kafka Streams Sample Code Description.....	832
20.3.4.1 High level KafkaStreams API Usage Sample.....	832
20.3.4.2 Low level KafkaStreams API Usage Sample.....	832
20.3.5 Kafka Token Authentication Mechanism Scenario.....	833
20.3.6 Sample Code of the Kafka Token Authentication Mechanism.....	834
20.4 Application Commissioning.....	835
20.4.1 Commissioning an Application in Windows.....	835
20.4.2 Commissioning an Application in Linux.....	837
20.4.3 Kafka Streams Sample Running Guide.....	840
20.4.3.1 High level Streams API Sample Usage Guide.....	840
20.4.3.2 Low level Streams API Sample Usage Guide.....	841
20.4.4 Running Guide of the Kafka Token Authentication Mechanism Sample Code.....	842
20.5 More Information.....	844
20.5.1 External Interfaces.....	844
20.5.1.1 Shell.....	844
20.5.1.2 Java API.....	845
20.5.1.3 Security Ports.....	847
20.5.1.4 SSL Encryption Function Used by a Client.....	848
20.5.2 Kafka Access Configuration on Windows Using EIPs.....	849
20.5.3 FAQ.....	850
20.5.3.1 Topic Authentication Fails During Sample Running and "example-metric1=TOPIC_AUTHORIZATION_FAILED" Is Displayed.....	850
20.5.3.2 Running the Producer.java Sample to Obtain Metadata Fails and "ERROR fetching topic metadata for topics..." Is Displayed, Even the Access Permission for the Related Topic.....	850
21 Kafka Development Guide (Normal Mode).....	852
21.1 Overview.....	852
21.1.1 Development Environment Preparation.....	852
21.1.2 Common Concepts.....	852
21.1.3 Development Process.....	853
21.2 Environment Preparation.....	854
21.2.1 Preparing for Development and Operating Environment.....	855
21.2.2 Configuring and Importing a Sample Project.....	857
21.3 Developing an Application.....	862
21.3.1 Typical Scenario Description.....	862
21.3.2 Example Code Description.....	863
21.3.2.1 Producer API Usage Sample.....	863
21.3.2.2 Consumer API Usage Sample.....	863

21.3.2.3 Multi-thread Producer Sample.....	864
21.3.2.4 Multi-thread Consumer Sample.....	865
21.3.3 Kafka Streams Sample Code Description.....	866
21.3.4 Kafka Streams Sample Code Description.....	866
21.3.4.1 High level KafkaStreams API Usage Sample.....	866
21.3.4.2 Low level KafkaStreams API Usage Sample.....	867
21.4 Application Commissioning.....	868
21.4.1 Commissioning an Application in Windows.....	868
21.4.2 Commissioning an Application in Linux.....	869
21.4.3 Kafka Streams Sample Running Guide.....	872
21.4.3.1 High Level Kafka Streams API Sample Usage Guide.....	872
21.4.3.2 Low level Kafka Streams API Sample Usage Guide.....	873
21.5 More Information.....	874
21.5.1 External Interfaces.....	874
21.5.1.1 Shell.....	874
21.5.1.2 Java API.....	875
21.5.2 Kafka Access Configuration on Windows Using EIPs.....	878
21.5.3 FAQ.....	879
21.5.3.1 Running the Producer.java Sample to Obtain Metadata Fails and "ERROR fetching topic metadata for topics..." Is Displayed, Even the Access Permission for the Related Topic.....	880
22 Kudu Development Guide (Security Mode).....	881
22.1 Overview.....	881
22.1.1 Introduction to Kudu.....	881
22.1.2 Basic Concepts.....	882
22.1.3 Development Process.....	883
22.2 Environment Preparation.....	884
22.2.1 Preparing the Development and Running Environment.....	884
22.2.2 Preparing for Security Authentication.....	886
22.3 Developing an Application.....	887
22.3.1 Typical Application Scenario.....	887
22.3.2 Development Idea.....	887
22.3.3 Sample Code Description.....	887
22.3.3.1 Establish Connections.....	887
22.3.3.2 Creating a Table.....	888
22.3.3.3 Opening a Table.....	889
22.3.3.4 Modifying a Table.....	889
22.3.3.5 Writing Data.....	889
22.3.3.6 Read Data.....	890
22.3.3.7 Deleting a Table.....	891
22.4 Commissioning the Application.....	891
22.5 More Information.....	893
22.5.1 Common APIs.....	893

22.5.1.1 Java API.....	893
23 Kudu Development Guide (Normal Mode).....	894
23.1 Overview.....	894
23.1.1 Introduction to Kudu.....	894
23.1.2 Basic Concepts.....	895
23.1.3 Development Process.....	896
23.2 Environment Preparation.....	897
23.2.1 Preparing the Development and Running Environment.....	897
23.3 Developing an Application.....	899
23.3.1 Typical Application Scenario.....	899
23.3.2 Development Idea.....	899
23.3.3 Sample Code Description.....	900
23.3.3.1 Establish Connections.....	900
23.3.3.2 Creating a Table.....	900
23.3.3.3 Opening a Table.....	901
23.3.3.4 Modifying a Table.....	901
23.3.3.5 Writing Data.....	901
23.3.3.6 Read Data.....	902
23.3.3.7 Deleting a Table.....	903
23.4 Commissioning the Application.....	903
23.5 More Information.....	905
23.5.1 Common APIs.....	905
23.5.1.1 Java API.....	905
24 MapReduce Development Guide (Security Mode).....	906
24.1 Overview.....	906
24.1.1 MapReduce Overview.....	906
24.1.2 Basic Concepts.....	906
24.1.3 Development Process.....	908
24.2 Environment Preparation.....	909
24.2.1 Preparing for Development and Operating Environment.....	909
24.2.2 Configuring and Importing Sample Projects.....	912
24.2.3 Creating a New Project (Optional).....	915
24.2.4 Preparing the Authentication Mechanism.....	918
24.3 Developing the Project.....	919
24.3.1 MapReduce Statistics Sample Project.....	919
24.3.1.1 Typical Scenarios.....	919
24.3.1.2 Example Code.....	920
24.3.2 MapReduce Accessing Multi-Component Example Project.....	923
24.3.2.1 Instance.....	924
24.3.2.2 Example Code.....	925
24.4 Commissioning the Application.....	930
24.4.1 Commissioning the Application in the Windows Environment.....	931

24.4.1.1 Compiling and Running the Application.....	931
24.4.1.2 Checking the Commissioning Result.....	932
24.4.2 Commissioning an Application in the Linux Environment.....	934
24.4.2.1 Compiling and Running the Application.....	934
24.4.2.2 Checking the Commissioning Result.....	936
24.5 More Information.....	937
24.5.1 Common APIs.....	938
24.5.1.1 Java API.....	938
24.5.1.2 REST API.....	940
24.5.2 FAQ.....	942
24.5.2.1 No Response from the Client When Submitting the MapReduce Application.....	942
24.5.2.2 When an Application Is Run, An Abnormality Occurs Due to Network Faults.....	943
24.5.2.3 How to Perform Remote Debugging During MapReduce Secondary Development?.....	943
25 MapReduce Development Guide (Normal Mode).....	947
25.1 Overview.....	947
25.1.1 MapReduce Overview.....	947
25.1.2 Basic Concepts.....	947
25.1.3 Development Process.....	948
25.2 Environment Preparation.....	950
25.2.1 Preparing Development and Operating Environment.....	950
25.2.2 Configuring and Importing Sample Projects.....	953
25.2.3 Creating a New Project (Optional).....	955
25.3 Developing the Project.....	958
25.3.1 MapReduce Statistics Sample Project.....	958
25.3.1.1 Typical Scenarios.....	958
25.3.1.2 Example Codes.....	959
25.3.2 MapReduce Accessing Multi-Component Example Project.....	962
25.3.2.1 Instance.....	962
25.3.2.2 Example Code.....	963
25.4 Commissioning the Application.....	968
25.4.1 Commissioning the Application in the Windows Environment.....	968
25.4.1.1 Compiling and Running the Application.....	968
25.4.1.2 Checking the Commissioning Result.....	969
25.4.2 Commissioning the Application in the Linux Environment.....	971
25.4.2.1 Compiling and Running the Application.....	971
25.4.2.2 Checking the Commissioning Result.....	972
25.5 More Information.....	975
25.5.1 Common APIs.....	975
25.5.1.1 Java API.....	975
25.5.1.2 REST API.....	978
25.5.2 FAQ.....	979
25.5.2.1 No Response from the Client When Submitting the MapReduce Application.....	980

25.5.2.2 How to Perform Remote Debugging During MapReduce Secondary Development?.....980

26 Oozie Development Guide (Security Mode)..... 983

26.1 Overview..... 983

26.1.1 Application Development Overview..... 983

26.1.2 Common Concepts..... 984

26.1.3 Development Process..... 984

26.2 Environment Preparation..... 986

26.2.1 Preparing Development and Operating Environment..... 986

26.2.2 Downloading and Importing Sample Projects..... 987

26.2.3 Preparing Authentication Mechanism Code..... 989

26.3 Developing the Project..... 990

26.3.1 Development of Configuration Files..... 990

26.3.1.1 Description..... 990

26.3.1.2 Development Procedure..... 991

26.3.2 Example Codes..... 993

26.3.2.1 job.properties..... 993

26.3.2.2 workflow.xml..... 994

26.3.2.3 Start Action..... 995

26.3.2.4 End Action..... 996

26.3.2.5 Kill Action..... 996

26.3.2.6 FS Action..... 997

26.3.2.7 MapReduce Action..... 998

26.3.2.8 coordinator.xml..... 999

26.3.3 Development of Java..... 1000

26.3.3.1 Description..... 1000

26.3.3.2 Sample Code..... 1000

26.3.4 Scheduling Spark2x to Access HBase and Hive Using Oozie..... 1002

26.4 Commissioning the Application..... 1005

26.4.1 Commissioning an Application in the Windows Environment..... 1005

26.4.1.1 Compiling and Running Applications..... 1005

26.4.1.2 Checking the Commissioning Result..... 1006

26.5 More Information..... 1007

26.5.1 Common API Introduce..... 1007

26.5.1.1 Shell..... 1007

26.5.1.2 Java..... 1008

26.5.1.3 REST..... 1008

27 Oozie Development Guide (Normal Mode)..... 1009

27.1 Overview..... 1009

27.1.1 Application Development Overview..... 1009

27.1.2 Common Concepts..... 1010

27.1.3 Development Process..... 1010

27.2 Environment Preparation..... 1012

27.2.1 Preparing for Development and Operating Environment.....	1012
27.2.2 Downloading and Importing Sample Projects.....	1013
27.3 Developing the Project.....	1015
27.3.1 Development of Configuration Files.....	1015
27.3.1.1 Description.....	1015
27.3.1.2 Development Procedure.....	1016
27.3.2 Example Codes.....	1018
27.3.2.1 job.properties.....	1018
27.3.2.2 workflow.xml.....	1019
27.3.2.3 Start Action.....	1020
27.3.2.4 End Action.....	1020
27.3.2.5 Kill Action.....	1021
27.3.2.6 FS Action.....	1021
27.3.2.7 MapReduce Action.....	1022
27.3.2.8 coordinator.xml.....	1024
27.3.3 Development of Java.....	1025
27.3.3.1 Description.....	1025
27.3.3.2 Sample Code.....	1025
27.3.4 Scheduling Spark2x to Access HBase and Hive Using Oozie.....	1026
27.4 Commissioning the Application.....	1029
27.4.1 Commissioning an Application in the Windows Environment.....	1029
27.4.1.1 Compiling and Running Applications.....	1029
27.4.1.2 Checking the Commissioning Result.....	1030
27.5 More Information.....	1031
27.5.1 Common API Introduce.....	1031
27.5.1.1 Shell.....	1031
27.5.1.2 Java.....	1032
27.5.1.3 REST.....	1032
28 Presto Development Guide (Security Mode).....	1033
28.1 Overview.....	1033
28.1.1 Application Development Overview.....	1033
28.1.2 Basic Concepts.....	1033
28.1.3 Application Development Process.....	1034
28.2 Environment Preparation.....	1035
28.2.1 Preparing Development and Operating Environment.....	1035
28.2.2 Configuring and Importing the JDBC API Sample Project.....	1038
28.2.3 Configuring and Importing the HCatalog API Sample Project.....	1039
28.3 Developing the Project.....	1040
28.3.1 Typical Application Scenario.....	1040
28.3.2 Sample Code Description.....	1042
28.4 Commissioning the Application.....	1043
28.5 Presto APIs.....	1044

28.6 FAQs.....	1044
28.6.1 No Certificate Is Available When PrestoJDBCExample Run on a Node Outside the Cluster.....	1044
28.6.2 When a Node Outside a Cluster Is Connected to a Cluster with Kerberos Authentication Enabled, HTTP Cannot Find the Corresponding Record in the Kerberos Database.....	1047
29 Presto Development Guide (Normal Mode).....	1051
29.1 Overview.....	1051
29.1.1 Application Development Overview.....	1051
29.1.2 Basic Concepts.....	1051
29.1.3 Application Development Process.....	1052
29.2 Environment Preparation.....	1053
29.2.1 Preparing Development and Operating Environment.....	1053
29.2.2 Configuring and Importing the JDBC API Sample Project.....	1056
29.2.3 Configuring and Importing the HCatalog API Sample Project.....	1057
29.3 Developing the Project.....	1058
29.3.1 Typical Application Scenario.....	1058
29.3.2 Sample Code Description.....	1060
29.4 Commissioning the Application.....	1061
29.5 Presto APIs.....	1061
30 Spark2x Development Guide (Security Mode).....	1062
30.1 Overview.....	1062
30.1.1 Application Development Overview.....	1062
30.1.2 Basic Concepts.....	1063
30.1.3 Development Process.....	1070
30.2 Preparing for the Environment.....	1073
30.2.1 Preparing for Development and Operating Environment.....	1073
30.2.2 Configuring and Importing Sample Projects.....	1077
30.2.3 Creating a New Project (Optional).....	1094
30.2.4 Preparing for Security Authentication.....	1095
30.2.5 Configuring the Python3 Sample Project.....	1099
30.3 Developing the Project.....	1100
30.3.1 Spark Core Project.....	1100
30.3.1.1 Overview.....	1100
30.3.1.2 Java Sample Code.....	1102
30.3.1.3 Scala Sample Code.....	1104
30.3.1.4 Python Sample Code.....	1104
30.3.2 Spark SQL Project.....	1105
30.3.2.1 Overview.....	1105
30.3.2.2 Java Sample Code.....	1107
30.3.2.3 Scala Sample Code.....	1108
30.3.2.4 Python Sample Code.....	1109
30.3.3 Accessing the Spark SQL Through JDBC.....	1110
30.3.3.1 Overview.....	1110

30.3.3.2 Java Sample Code.....	1111
30.3.3.3 Scala Sample Code.....	1113
30.3.4 Spark on HBase.....	1115
30.3.4.1 Performing Operations on Data in Avro Format.....	1115
30.3.4.2 Performing Operations on the HBase Data Source.....	1119
30.3.4.3 Using the BulkPut Interface.....	1122
30.3.4.4 Using the BulkGet Interface.....	1125
30.3.4.5 Using the BulkDelete Interface.....	1129
30.3.4.6 Using the BulkLoad Interface.....	1132
30.3.4.7 Using the foreachPartition Interface.....	1135
30.3.4.8 Distributedly Scanning HBase Tables.....	1139
30.3.4.9 Using the mapPartition Interface.....	1141
30.3.4.10 Writing Data to HBase Tables In Batches Using SparkStreaming.....	1145
30.3.5 Reading Data from HBase and Write It Back to HBase.....	1149
30.3.5.1 Overview.....	1149
30.3.5.2 Java Example Code.....	1151
30.3.5.3 Scala Example Code.....	1154
30.3.5.4 Python Example Code.....	1156
30.3.6 Reading Data from Hive and Write It to HBase.....	1156
30.3.6.1 Overview.....	1156
30.3.6.2 Java Example Code.....	1159
30.3.6.3 Scala Example Code.....	1161
30.3.6.4 Python Example Code.....	1163
30.3.7 Streaming Connecting to Kafka0-10.....	1164
30.3.7.1 Overview.....	1164
30.3.7.2 Java Example Code.....	1167
30.3.7.3 Scala Example Code.....	1169
30.3.8 Structured Streaming Project.....	1172
30.3.8.1 Overview.....	1172
30.3.8.2 Java Sample Code.....	1174
30.3.8.3 Scala Sample Code.....	1175
30.3.8.4 Python Sample Code.....	1176
30.3.9 Structured Streaming Stream-Stream Join.....	1177
30.3.9.1 Overview.....	1177
30.3.9.2 Scala Example Code.....	1181
30.3.10 Structured Streaming Status Operation.....	1183
30.3.10.1 Overview.....	1184
30.3.10.2 Scala Sample Code.....	1186
30.3.11 Concurrent Access from Spark to HBase in Two Clusters.....	1188
30.3.11.1 Overview.....	1188
30.3.11.2 Scala Sample Code.....	1188
30.3.12 Synchronizing HBase Data from Spark to CarbonData.....	1189

30.3.12.1 Overview.....	1189
30.3.12.2 Java Example Code.....	1191
30.3.13 Using Spark to Perform Basic Hudi Operations.....	1192
30.3.13.1 Overview.....	1192
30.3.13.2 Java Example Code.....	1193
30.3.13.3 Scala Example Code.....	1194
30.3.13.4 Python Example Code.....	1195
30.3.14 Compiling User-defined Configuration Items for Hudi.....	1197
30.3.14.1 HoodieDeltaStreamer.....	1197
30.3.14.2 User-defined Partitioner.....	1198
30.4 Commissioning the Application.....	1199
30.4.1 Commissioning Applications on Windows.....	1199
30.4.1.1 Spark Access Configuration on Windows Using EIPs.....	1199
30.4.1.2 Compiling and Running Applications.....	1201
30.4.1.3 View Debugging Results.....	1203
30.4.2 Commissioning an Application in Linux.....	1204
30.4.2.1 Compiling and Running the Application.....	1204
30.4.2.2 Checking the Commissioning Result.....	1207
30.5 More Information.....	1208
30.5.1 Common APIs.....	1208
30.5.1.1 Java.....	1208
30.5.1.2 Scala.....	1213
30.5.1.3 Python.....	1219
30.5.1.4 REST API.....	1223
30.5.2 Common CLIs.....	1231
30.5.3 JDBCServer Interface.....	1232
30.5.4 Structured Streaming Functions and Reliability.....	1234
30.5.5 FAQ.....	1239
30.5.5.1 How to Add a User-Defined Library.....	1239
30.5.5.2 How to Automatically Load Jars Packages?.....	1242
30.5.5.3 Why the "Class Does not Exist" Error Is Reported While the SparkStreamingKafka Project Is Running?.....	1242
30.5.5.4 Privilege Control Mechanism of SparkSQL UDF Feature.....	1243
30.5.5.5 Why Does Kafka Fail to Receive the Data Written Back by SLog in to the node where the client is installed as the client installation user.park Streaming?.....	1243
30.5.5.6 Why a Spark Core Application Is Suspended Instead of Being Exited When Driver Memory Is Insufficient to Store Collected Intensive Data?.....	1244
30.5.5.7 Why the Name of the Spark Application Submitted in Yarn-Cluster Mode Does not Take Effect?.....	1246
30.5.5.8 How to Perform Remote Debugging Using IDEA?.....	1246
30.5.5.9 How to Submit the Spark Application Using Java Commands?.....	1249
30.5.5.10 A Message Stating "Problem performing GSS wrap" Is Displayed When IBM JDK Is Used.....	1251
30.5.5.11 Application Fails When ApplicationManager Is Terminated During Data Processing in the Cluster Mode of Structured Streaming.....	1252

30.5.5.12 Restrictions on Restoring the Spark Application from the checkpoint.....	1252
30.5.5.13 Support for Third-party JAR Packages on x86 and TaiShan Platforms.....	1253
30.5.5.14 What Should I Do If a Large Number of Directories Whose Names Start with blockmgr- or spark- Exist in the /tmp Directory on the Client Installation Node?.....	1255
30.5.5.15 Error Code 139 Reported When Python Pipeline Runs in the ARM Environment.....	1256
30.5.5.16 What Should I Do If the Structured Streaming Task Submission Way Is Changed?.....	1257
30.5.5.17 Common JAR File Conflicts.....	1258
31 Spark2x Development Guide (Normal Mode).....	1261
31.1 Overview.....	1261
31.1.1 Application Development Overview.....	1261
31.1.2 Basic Concepts.....	1262
31.1.3 Development Process.....	1269
31.2 Preparing for the Environment.....	1271
31.2.1 Development and Operating Environment.....	1271
31.2.2 Configuring and Importing Sample Projects.....	1276
31.2.3 Creating a New Project (Optional).....	1291
31.2.4 Configuring the Python3 Sample Project.....	1293
31.3 Developing the Project.....	1294
31.3.1 Spark Core Project.....	1294
31.3.1.1 Overview.....	1294
31.3.1.2 Java Example Code.....	1296
31.3.1.3 Scala Sample Code.....	1297
31.3.1.4 Python Example Code.....	1298
31.3.2 Spark SQL Project.....	1299
31.3.2.1 Overview.....	1299
31.3.2.2 Java Sample Code.....	1301
31.3.2.3 Scala Sample Code.....	1302
31.3.2.4 Python Sample Code.....	1302
31.3.3 Accessing the Spark SQL Through JDBC.....	1303
31.3.3.1 Overview.....	1303
31.3.3.2 Java Example Code.....	1304
31.3.3.3 Scala Example Code.....	1306
31.3.4 Spark on HBase.....	1308
31.3.4.1 Performing Operation on Data in Avro Format.....	1308
31.3.4.2 Performing Operations on the HBase Data Source.....	1311
31.3.4.3 Using the BulkPut Interface.....	1314
31.3.4.4 Using the BulkGet Interface.....	1317
31.3.4.5 Using the BulkDelete Interface.....	1320
31.3.4.6 Using the BulkLoad Interface.....	1323
31.3.4.7 Using the foreachPartition Interface.....	1326
31.3.4.8 Distributedly Scanning HBase Tables.....	1329
31.3.4.9 Using the mapPartition Interface.....	1331

31.3.4.10 Writing Data to HBase Tables In Batches Using SparkStreaming.....	1334
31.3.5 Reading Data from HBase and Write It Back to HBase.....	1338
31.3.5.1 Overview.....	1338
31.3.5.2 Java Example Code.....	1340
31.3.5.3 Scala Example Code.....	1342
31.3.5.4 Python Example Code.....	1344
31.3.6 Reading Data from Hive and Write It to HBase.....	1345
31.3.6.1 Overview.....	1345
31.3.6.2 Java Example Code.....	1347
31.3.6.3 Scala Example Code.....	1349
31.3.6.4 Python Example Code.....	1351
31.3.7 Streaming Connecting to Kafka0-10.....	1351
31.3.7.1 Overview.....	1352
31.3.7.2 Java Example Code.....	1354
31.3.7.3 Scala Example Code.....	1356
31.3.8 Structured Streaming Project.....	1359
31.3.8.1 Overview.....	1359
31.3.8.2 Java Sample Code.....	1361
31.3.8.3 Scala Sample Code.....	1362
31.3.8.4 Python Sample Code.....	1363
31.3.9 Structured Streaming Stream-Stream Join.....	1363
31.3.9.1 Overview.....	1364
31.3.9.2 Scala Example Code.....	1367
31.3.10 Structured Streaming Status Operation.....	1369
31.3.10.1 Overview.....	1369
31.3.10.2 Scala Sample Code.....	1371
31.3.11 Synchronizing HBase Data from Spark to CarbonData.....	1374
31.3.11.1 Overview.....	1374
31.3.11.2 Java Example Code.....	1376
31.3.12 Using Spark to Perform Basic Hudi Operations.....	1376
31.3.12.1 Overview.....	1377
31.3.12.2 Java Example Code.....	1377
31.3.12.3 Scala Example Code.....	1378
31.3.12.4 Python Sample Code.....	1380
31.3.13 Compiling User-defined Configuration Items for Hudi.....	1382
31.3.13.1 HoodieDeltaStreamer.....	1382
31.3.13.2 User-defined Partitioner.....	1383
31.4 Commissioning the Application.....	1383
31.4.1 Commissioning Applications on Windows.....	1384
31.4.1.1 Spark Access Configuration on Windows Using EIPs.....	1384
31.4.1.2 Compiling and Running Applications.....	1386
31.4.1.3 View Debugging Results.....	1387

31.4.2 Commissioning an Application in Linux.....	1387
31.4.2.1 Compiling and Running the Application.....	1388
31.4.2.2 Checking the Commissioning Result.....	1392
31.5 More Information.....	1392
31.5.1 Common APIs.....	1392
31.5.1.1 Java.....	1392
31.5.1.2 Scala.....	1398
31.5.1.3 Python.....	1403
31.5.2 Common CLIs.....	1407
31.5.3 JDBCServer Interface.....	1408
31.5.4 Structured Streaming Functions and Reliability.....	1410
31.5.5 FAQ.....	1415
31.5.5.1 How to Add a User-Defined Library.....	1415
31.5.5.2 How to Automatically Load Jars Packages?.....	1418
31.5.5.3 Why the "Class Does not Exist" Error Is Reported While the SparkStreamingKafka Project Is Running?.....	1418
31.5.5.4 Why Does Kafka Fail to Receive the Data Written Back by Spark Streaming?.....	1419
31.5.5.5 Why a Spark Core Application Is Suspended Instead of Being Exited When Driver Memory Is Insufficient to Store Collected Intensive Data?.....	1420
31.5.5.6 Why the Name of the Spark Application Submitted in Yarn-Cluster Mode Does not Take Effect?.....	1422
31.5.5.7 How to Perform Remote Debugging Using IDEA?.....	1422
31.5.5.8 How to Submit the Spark Application Using Java Commands?.....	1425
31.5.5.9 A Message Stating "Problem performing GSS wrap" Is Displayed When IBM JDK Is Used.....	1427
31.5.5.10 Application Fails When ApplicationManager Is Terminated During Data Processing in the Cluster Mode of Structured Streaming.....	1428
31.5.5.11 Restrictions on Restoring the Spark Application from the checkpoint.....	1428
31.5.5.12 Support for Third-party JAR Packages on x86 and TaiShan Platforms.....	1429
31.5.5.13 What Should I Do If a Large Number of Directories Whose Names Start with blockmgr- or spark- Exist in the /tmp Directory on the Client Installation Node?.....	1431
31.5.5.14 Error Code 139 Reported When Python Pipeline Runs in the ARM Environment.....	1432
31.5.5.15 What Should I Do If the Structured Streaming Task Submission Way Is Changed?.....	1433
31.5.5.16 Common JAR File Conflicts.....	1434
32 Storm Development Guide (Security Mode).....	1437
32.1 Overview.....	1437
32.1.1 Application Development Overview.....	1437
32.1.2 Common Concepts.....	1438
32.1.3 Development Process.....	1438
32.2 Environment Preparation.....	1440
32.2.1 Environment Preparation Overview.....	1441
32.2.2 Preparing for Development and Operating Environment.....	1441
32.2.3 Configuring and Importing Sample Projects.....	1444
32.3 Developing an Application.....	1451

32.3.1 Typical Scenario Description.....	1451
32.3.2 Development Idea.....	1451
32.3.3 Example Code Description.....	1452
32.3.3.1 Creating a Spout.....	1452
32.3.3.2 Creating a Bolt.....	1452
32.3.3.3 Creating a Topology.....	1453
32.4 Running an Application.....	1455
32.4.1 Packaging IntelliJ IDEA Code.....	1455
32.4.2 Packaging Services.....	1458
32.4.2.1 Overview.....	1458
32.4.2.2 Packaging Services on a Linux OS.....	1458
32.4.2.3 Packaging Services on a Windows OS.....	1459
32.4.3 Submitting a Topology.....	1459
32.4.3.1 Submitting a Topology When a Client Is Installed on a Linux OS.....	1460
32.4.3.2 Submitting a Topology When No Client Is Installed on a Linux OS.....	1461
32.4.3.3 Submitting a Topology in IntelliJ IDEA Remotely.....	1462
32.4.4 Viewing Results.....	1464
32.5 More Information.....	1465
32.5.1 Storm-Kafka Development Guideline.....	1466
32.5.2 Storm-JDBC Development Guideline.....	1469
32.5.3 Storm-HDFS Development Guideline.....	1472
32.5.4 Storm-HBase Development Guideline.....	1475
32.5.5 Flux Development Guideline.....	1478
32.5.6 External Interfaces.....	1484
32.5.7 FAQ.....	1484
32.5.7.1 How Do I Use IDEA to Remotely Debug Services?.....	1484
32.5.7.2 How Do I Handle the Error "Command line is too long" Reported When Main Is Executed for Remote Topology Submission in IntelliJ IDEA.....	1488
33 Storm Development Guide (Normal Mode).....	1489
33.1 Overview.....	1489
33.1.1 Application Development Overview.....	1489
33.1.2 Common Concepts.....	1490
33.1.3 Development Process.....	1490
33.2 Environment Preparation.....	1492
33.2.1 Environment Preparation Overview.....	1493
33.2.2 Preparing for Development and Operating Environment.....	1493
33.2.3 Configuring and Importing Sample Projects.....	1496
33.3 Developing an Application.....	1503
33.3.1 Typical Scenario Description.....	1503
33.3.2 Development Idea.....	1503
33.3.3 Example Code Description.....	1504
33.3.3.1 Creating a Spout.....	1504

33.3.3.2 Creating a Bolt.....	1504
33.3.3.3 Creating a Topology.....	1505
33.4 Running an Application.....	1507
33.4.1 Packaging IntelliJ IDEA Code.....	1507
33.4.2 Packaging Services.....	1510
33.4.2.1 Overview.....	1510
33.4.2.2 Packaging Services on a Linux OS.....	1510
33.4.2.3 Packaging Services on a Windows OS.....	1511
33.4.3 Submitting a Topology.....	1511
33.4.3.1 Submitting a Topology When a Client Is Installed on a Linux OS.....	1512
33.4.3.2 Submitting a Topology When No Client Is Installed on a Linux OS.....	1512
33.4.3.3 Submitting a Topology in IntelliJ IDEA Remotely.....	1514
33.4.4 Viewing Results.....	1515
33.5 More Information.....	1516
33.5.1 Storm-Kafka Development Guideline.....	1517
33.5.2 Storm-JDBC Development Guideline.....	1520
33.5.3 Storm-HDFS Development Guideline.....	1522
33.5.4 Storm-HBase Development Guideline.....	1525
33.5.5 Flux Development Guideline.....	1527
33.5.6 External Interfaces.....	1533
33.5.7 FAQ.....	1533
33.5.7.1 How Do I Use IDEA to Remotely Debug Services?.....	1533
33.5.7.2 How Do I Set the Offset Correctly when Using the Old Plug-in storm-kafka?.....	1536
33.5.7.3 How Do I Handle the Error "Command line is too long" Reported When Main Is Executed for Remote Topology Submission in IntelliJ IDEA.....	1537
34 YARN Development Guide (Security Mode).....	1539
34.1 Overview.....	1539
34.2 Interfaces.....	1540
34.2.1 Command.....	1540
34.2.2 Java API.....	1547
34.2.3 REST API.....	1549
34.2.4 REST APIs of Superior Scheduler.....	1553
35 YARN Development Guide (Normal Mode).....	1569
35.1 Overview.....	1569
35.2 Interfaces.....	1570
35.2.1 Command.....	1570
35.2.2 Java API.....	1577
35.2.3 REST API.....	1579
35.2.4 REST APIs of Superior Scheduler.....	1583
36 Development Specifications.....	1599
36.1 Development Environment Construction.....	1599
36.1.1 Rules.....	1599

36.2 Security Authentication.....	1599
36.2.1 Rules.....	1600
36.2.2 Suggestions.....	1600
36.3 ClickHouse.....	1600
36.3.1 Rules.....	1600
36.3.2 Suggestions.....	1602
36.4 Flink.....	1605
36.4.1 Applicable Scenarios.....	1605
36.4.2 Rules.....	1605
36.4.3 Suggestions.....	1605
36.5 GraphBase.....	1605
36.5.1 Rules.....	1606
36.5.2 Recommendations.....	1607
36.6 HBase.....	1607
36.6.1 Application Scenarios.....	1608
36.6.2 Rules.....	1608
36.6.3 Suggestions.....	1613
36.6.4 Examples.....	1615
36.6.5 Appendix.....	1621
36.7 HDFS.....	1623
36.7.1 Application Scenarios.....	1623
36.7.2 Rules.....	1623
36.7.3 Suggestions.....	1627
36.8 Hive.....	1628
36.8.1 Application Scenarios.....	1628
36.8.2 Rules.....	1629
36.8.3 Suggestions.....	1633
36.8.4 Examples.....	1634
36.9 Kafka.....	1643
36.9.1 Application Scenarios.....	1643
36.9.2 Rules.....	1644
36.9.3 Suggestions.....	1644
36.10 Mapreduce.....	1645
36.10.1 Application Scenarios.....	1645
36.10.2 Rules.....	1645
36.10.3 Suggestions.....	1647
36.10.4 Examples.....	1647
36.11 Oozie.....	1649
36.11.1 Application Scenarios.....	1649
36.11.2 Rules.....	1650
36.11.3 Suggestions.....	1651
36.12 Spark2x.....	1653

36.12.1 Application Scenarios.....	1653
36.12.2 Rules.....	1653
36.12.3 Suggestions.....	1656
36.13 Yarn.....	1659
36.13.1 Application Scenarios.....	1659
36.13.2 Rules.....	1659
37 Manager Management Development Guide.....	1661
37.1 Overview.....	1661
37.1.1 Application Development Overview.....	1661
37.1.2 Common Concepts.....	1661
37.1.3 Development Process.....	1662
37.2 Environment Preparation.....	1664
37.2.1 Preparing Development and Running Environments.....	1664
37.2.2 Configuring and Importing a Sample Project.....	1665
37.3 Developing an Application.....	1668
37.3.1 Typical Scenario Description.....	1668
37.3.2 Development Guideline.....	1669
37.3.3 Example Code Description.....	1669
37.3.3.1 Login Authentication.....	1669
37.3.3.2 Adding Users.....	1670
37.3.3.3 Searching for Users.....	1670
37.3.3.4 Modifying Users.....	1670
37.3.3.5 Deleting Users.....	1671
37.3.3.6 Exporting a User List.....	1671
37.4 Application Commissioning.....	1671
37.4.1 Commissioning an Application in the Windows OS.....	1672
37.4.1.1 Compiling and Running an Application.....	1672
37.4.1.2 Viewing Windows Commissioning Results.....	1672
37.5 More Information.....	1676
37.5.1 External Interfaces.....	1676
37.5.1.1 Java API.....	1676
37.5.2 FAQ.....	1678
37.5.2.1 JDK1.6 Fails to Connect to the FusionInsight System Using JDK1.8.....	1678
37.5.2.2 An Operation Fails and "authorize failed" Is Displayed in Logs.....	1679
37.5.2.3 An Operation Fails and "log4j:WARN No appenders could be found for logger(basicAuth.Main)" Is Displayed in Logs.....	1680
37.5.2.4 An Operation Fails and "illegal character in path at index 57" Is Displayed in Logs.....	1680
37.5.2.5 Run the curl Command to Access REST APIs.....	1681

1 Description

Overview

This document describes the application development process, sample projects, and code of components in the big data cluster. It also provides FAQs and development specifications for developers with Java development experience to develop applications.

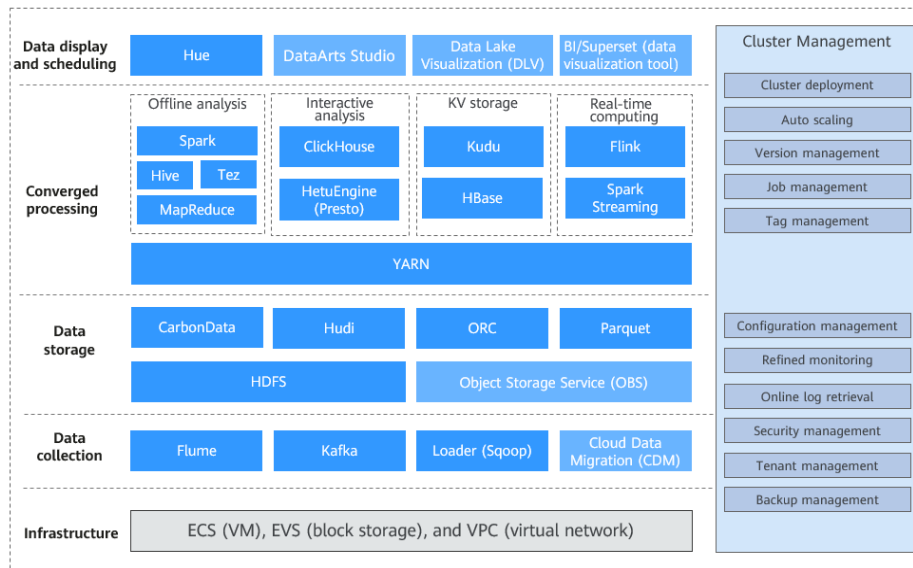
Preparations

- The cluster has been installed and is running properly.
- You have a basic understanding of components in the big data cluster.
- You have a basic understanding of Java.
- You have learned about MRS development components and how to use the Elastic Cloud Server (ECS) on HUAWEI CLOUD.
- You have a basic understanding about how to use Maven.

Obtaining Sample Projects

Log in to GitHub to obtain the files related to the component sample projects. Download the dependency JAR files of the sample projects from Huawei Mirrors, and download the rest open source dependency JAR files from the Maven central repository.

Figure 1-1 MRS architecture



The MRS sample code library provides sample projects of basic functions of each component for users. You can import a sample project for compilation and running based on the Kerberos authentication mode of the cluster.

2 Obtaining Sample Projects from Huawei Mirrors

Procedure

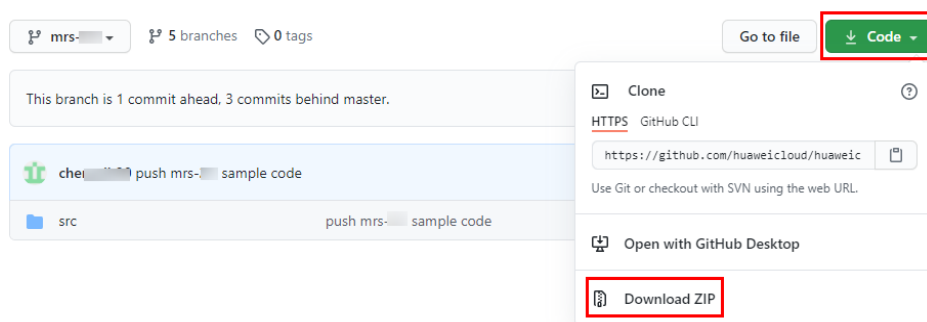
Building a sample project includes the following operations:

1. Download the Maven project source code and configuration files of the sample project. For details, see [Obtaining Sample Projects](#).
2. Configure the Maven mirror repository of the SDK in Huawei Mirrors. For details, see [Configuring Huawei Open-Source Mirrors](#).
3. Build a complete Maven project based on user requirements.

Obtaining Sample Projects

- For MRS 1.8.x, you can download sample projects at <https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-1.8>.
- For MRS 1.9.x, you can download sample projects at <https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-1.9>.
- For MRS 2.0.x and 2.1.x, you can download sample projects at <https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-2.0>.
- For MRS 3.0.2, you can download sample projects at <https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-3.0.2>.
- For MRS 3.1.0, you can download sample projects at <https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-3.1.0>.

Figure 2-1 Downloading sample code



Download the package to the local host and decompress it to obtain the sample code of the components.

Configuring Huawei Open-Source Mirrors

Huawei provides Huawei Mirrors for you to download all dependency JAR files of sample projects. However, you need to download the rest dependency open-source JAR files from the Maven central repository or other custom repository address.

NOTE

Before using a development tool to download the dependency JAR files in the local environment, ensure that the following conditions are met:

- The local network is normal.
Uses a browser and visit Huawei Mirrors to check whether the website can be accessed. If the access is abnormal, connect the local network.
- The proxy is disabled for the development tool.
Take the IntelliJ IDEA development tool of version 2020.2 as an example. Choose **File > Settings > Appearance & Behavior > System Settings > HTTP Proxy**, select **No proxy**, and click **OK** to save the configuration.

Perform the following steps to configure the open-source mirror warehouse.

Step 1 Check that JDK 1.8 or later and Maven 3.0 or later have been installed.

Step 2 Configure the Maven configuration file.

- To overwrite the Maven configuration file, choose **Huawei SDK > HuaweiCloud SDK** at Huawei Mirrors, download the **settings.xml** file, and overwrite the `<Maven installation directory>/conf/settings.xml` file with the downloaded file.
- (Optional) If you do not want to overwrite the Maven configuration file, manually modify the **settings.xml** configuration file or the **pom.xml** file of the component sample project to configure the mirror repository address. The configuration methods are as follows:
 - **Configuration method 1:** Modify the **settings.xml** configuration file.

Add the following open-source mirror repository address to **mirrors** in the **settings.xml** configuration file.

```
<mirror>
  <id>repo2</id>
  <mirrorOf>central</mirrorOf>
  <url>https://repo1.maven.org/maven2/</url>
</mirror>
```

Add the following mirror repository address to **profiles** in the **settings.xml** configuration file.

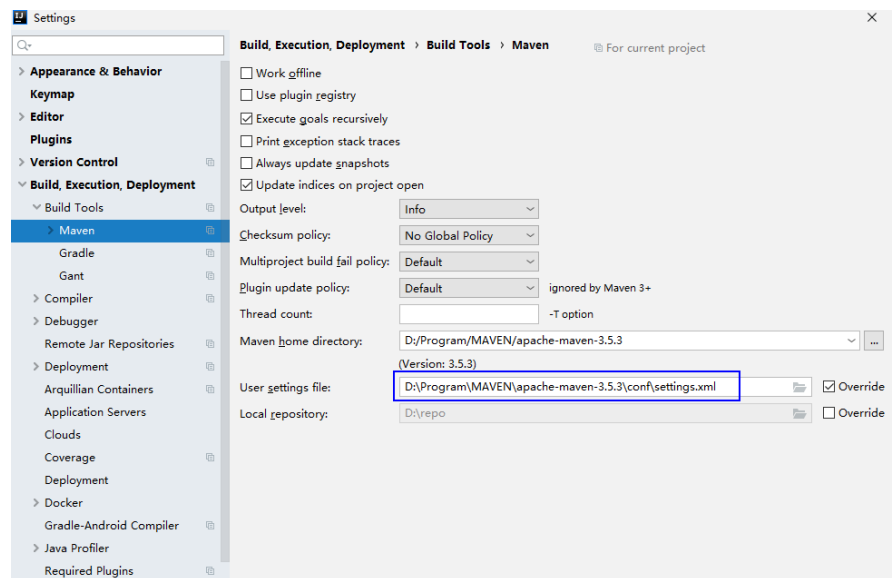
```
<profile>
  <id>huaweicloudsdk</id>
  <repositories>
    <repository>
      <id>huaweicloudsdk</id>
      <url>https://repo.huaweicloud.com/repository/maven/huaweicloudsdk/</url>
      <releases><enabled>true</enabled></releases>
      <snapshots><enabled>true</enabled></snapshots>
    </repository>
  </repositories>
</profile>
```

Add the following mirror repository address to the **activeProfiles** node in the **settings.xml** file.

```
<activeProfile>huaweicloudsdk</activeProfile>
```

NOTE

- Huawei Mirrors does not provide third-party open-source JAR files. After configuring Huawei open-source mirrors, you need to separately configure third-party Maven mirror repository address.
- When using the IntelliJ IDEA development tool, you can choose **File > Settings > Build, Execution, Deployment > Build Tools > Maven** to view the directory where the **settings.xml** file is stored.



- **Configuration method 2:** Modify the **pom.xml** configuration file.

Add the following mirror repository address directly to the **pom.xml** file in the secondary development sample project.

```
<repositories>
  <repository>
    <id>huaweicloudsdk</id>
    <url>https://mirrors.huaweicloud.com/repository/maven/huaweicloudsdk</url>
    <releases><enabled>true</enabled></releases>
    <snapshots><enabled>true</enabled></snapshots>
  </repository>
  <repository>
    <id>central</id>
    <name>Maven Central</name>
    <url>https://repo1.maven.org/maven2</url>
  </repository>
</repositories>
```

Step 3 Configure the default Maven code and JDK. Add the following information to **profiles** in the **settings.xml** configuration file:

```
<profile>
<id>JDK1.8</id>
<activation>
<activeByDefault>true</activeByDefault>
<jdk>1.8</jdk>
</activation>
<properties>
```

```
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
<maven.compiler.encoding>UTF-8</maven.compiler.encoding>  
<maven.compiler.source>1.8</maven.compiler.source>  
<maven.compiler.target>1.8</maven.compiler.target>  
<maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>  
</properties>  
</profile>
```

----End

3 Using Open-source JAR File Conflict Lists

3.1 HBase

HBase JAR File Conflict List

JAR File	Description
hbase-client-2.2.3-*.jar	JAR file required for connecting to the HBase service.
zookeeper-*.jar	JAR file required for connecting to the ZooKeeper service.

Solution

Use the ZooKeeper JAR file **Zookeeper*.jar** provided by the MRS cluster.

Use **exclusions** to exclude ZooKeeper from the HBase client.

3.2 HDFS

HDFS JAR File Conflict List

JAR File	Description	Solution
hadoop-plugins-*.jar	<p>HDFS can directly use the open-source Hadoop JAR file of the same version to run the sample code. However, in versions later than MRS 3.x, the default active/standby switchover class is dfs.client.failover.proxy.provider.hacluster=org.apache.hadoop.hdfs.server.namenode.ha.AdaptiveFailoverProxyProvider.</p> <p>The default HDFS LZC compression format class is io.compression.codec.lzc.class=com.huawei.hadoop.datasinght.io.compress.lzc.ZCodec.</p>	<ul style="list-style-type: none"> Method 1: Add the following configuration to the pom.xml file in the sample code: <pre><properties> <hadoop.ext.version>8.0.2-302002</hadoop.ext.version> </properties> ... <dependency> <groupId>com.huawei.mrs</groupId> <artifactId>hadoop-plugins</artifactId> <version>\${hadoop.ext.version}</version> </dependency></pre> Method 2: <ol style="list-style-type: none"> Change the value of the dfs.client.failover.proxy.provider.hacluster parameter in the hdfs-site.xml configuration file to org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider, which is used in the open source community. Do not use the LZC compression format.

3.3 Kafka

Kafka JAR File Conflict List

JAR File	Description
kafka_2.11-*.jar	JAR file required for connecting to the Kafka service.

JAR File	Description
kafka-clients-*.jar	JAR file required for connecting to the Kafka service.

Solution

The open-source Kafka JAR file is not recommended.

3.4 Spark2x

Spark2x JAR File Conflict List

JAR File	Description	Solution
spark-core_2.1.1-*.jar	Core JAR file of Spark tasks.	Spark can directly use the open-source Spark JAR file of the same version to run the sample code. However, the serialization IDs of spark-core JAR files of various versions may be different. You are advised to use the JAR file provided by the cluster.
jackson-*.jar	The following error message is reported when the Spark program is executed: com.fasterxml.jackson.databind.JsonMappingException: Scala module 2.11.4 requires Jackson Databind version >= 2.11.0 and < 2.12.0	The version of the Jackson file imported during program execution is inconsistent with that of the package file by the cluster. You are advised to use the Jackson JAR file provided by the cluster in <i>Client installation directory/Spark2x/spark/jars</i> .

NOTE

For details about how to handle Spark2x JAR file conflicts, see [Common JAR File Conflicts](#).

4 Mapping Between Maven Repository JAR Versions and MRS Component Versions

Mapping Between Maven Repository JAR Versions and MRS Component Versions in an MRS 3.1.0 Cluster

Table 4-1 Mapping between Maven repository JAR versions and MRS component versions in an MRS 3.1.0 cluster

Component	Component Version	JAR Version
Flink	1.12.0	1.12.0-hw-ei-310003
Hive	3.1.0	3.1.0-hw-ei-310003
Tez	0.9.2	0.9.1.0101-hw-ei-12
Spark2x	2.4.5	2.4.5-hw-ei-310003
CarbonData	2.0.1	-
Hadoop	3.1.1	3.1.1-hw-ei-310003
HBase	2.2.3	2.2.3-hw-ei-310003
ZooKeeper	3.5.6	3.5.6-hw-ei-310003
Hue	4.7.0	-
Oozie	5.1.0	5.1.0-hw-ei-310003
Flume	1.9.0	-
Kafka	2.4.0	2.4.0-hw-ei-310003
Ranger	2.0.0	-
Phoenix	5.0.0	5.0.0-HBase-2.0-hw-ei-310003
ClickHouse	21.3.4.25	0.3.0

Component	Component Version	JAR Version
Presto	316	316-hw-ei-310003
scala	2.11	-

Mapping Between Maven Repository JAR Versions and MRS Component Versions in an MRS 3.0.x Cluster

Table 4-2 Mapping between Maven repository JAR versions and MRS component versions in an MRS 3.0.x cluster

Component	Component Version	JAR Version
Flink	1.10.0	1.10.0-hw-ei-302002
Hive	3.1.0	3.1.0-hw-ei-302002
Tez	0.9.2	0.9.1.0101-hw-ei-12
Spark	2.4.5	2.4.5-hw-ei-302002
CarbonData	2.0.0	-
Hadoop	3.1.1	3.1.1-hw-ei-302002
HBase	2.2.3	2.2.3-hw-ei-302002
ZooKeeper	3.5.6	3.5.6-hw-ei-302002
Hue	4.7.0	-
Oozie	5.1.0	5.1.0-hw-ei-302002
Flume	1.9.0	-
Kafka	2.4.0	2.4.0-hw-ei-302002
Ranger	2.0.0	-
GeoMesa	2.4.0	-
Storm	1.2.0	1.2.1-hw-ei-302002
Phoenix	5.0.0	5.0.0-HBase-2.0-hw-ei-302002
Presto	316	316-hw-ei-302002
scala	2.11	-

Mapping Between Maven Repository JAR Versions and MRS Component Versions in an MRS 2.1.x Cluster

Table 4-3 Mapping between Maven repository JAR versions and MRS component versions in an MRS 2.1.x cluster

Component	Component Version	JAR Version
ZooKeeper	3.5.1	3.5.1-mrs-2.1
Hadoop	3.1.1	3.1.1-mrs-2.1
HBase	2.1.1	2.1.1-mrs-2.1
Tez	0.9.1	0.9.1.0101-hw-ei-12
Hive	3.1.0	3.1.0-mrs-2.1
Hive_Spark	1.2.1	1.2.1.spark_2.3.2-mrs-2.1
Spark	2.3.2	2.3.2-mrs-2.1
Carbon	1.5.1	-
Presto	308	-
Kafka	1.1.0	1.1.0-mrs-2.1
KafkaManager	1.3.3.18	-
Flink	1.7.0	1.7.0-mrs-2.1
Storm	1.2.1	1.2.1-mrs-2.1
Flume	1.6.0	1.6.0-mrs-2.1
Hue	3.11.0	-
Loader(Sqoop)	1.99.7	1.99.7-mrs-2.1
scala	2.11	-

Mapping Between Maven Repository JAR Versions and MRS Component Versions in an MRS 2.0.x Cluster

Table 4-4 Mapping between Maven repository JAR versions and MRS component versions in an MRS 2.0.x cluster

Component	Component Version	JAR Version
ZooKeeper	3.5.1	3.5.1-mrs-2.0
Hadoop	3.1.1	3.1.1-mrs-2.0
HBase	2.1.1	2.1.1-mrs-2.0

Component	Component Version	JAR Version
Tez	0.9.1	0.9.1.0101-hw-ei-12
Hive	3.1.0	3.1.0-mrs-2.0
Hive_Spark	1.2.1	1.2.1.spark_2.3.2-mrs-2.0
Spark	2.3.2	2.3.2-mrs-2.0
Carbon	1.5.1	-
Presto	308	-
Kafka	1.1.0	1.1.0-mrs-2.0
KafkaManager	1.3.3.18	-
Storm	1.2.1	1.2.1-mrs-2.0
Flume	1.6.0	1.6.0-mrs-2.0
Hue	3.11.0	-
Loader(Sqoop)	1.99.7	1.99.7-mrs-2.0
scala	2.11	-

Mapping Between Maven Repository JAR Versions and MRS Component Versions in an MRS 1.9.x Cluster

Table 4-5 Mapping between Maven repository JAR versions and MRS component versions in an MRS 1.9.x cluster

Component	Component Version	JAR Version
ZooKeeper	3.5.1	3.5.1-mrs-1.9.0
Hadoop	2.8.3	2.8.3-mrs-1.9.0
HBase	1.3.1	1.3.1-mrs-1.9.0
OpenTSDB	2.3.0	-
Tez	0.9.1	0.9.1.0101-hw-ei-12
Hive	2.3.3	2.3.3-mrs-1.9.0
Hive_Spark	1.2.1	1.2.1.spark_2.2.1-mrs-1.9.0
Spark	2.2.2	2.2.2-mrs-1.9.0
Carbon	1.6.1	-
Presto	0.216	0.216-mrs-1.9

Component	Component Version	JAR Version
Kafka	1.1.0	1.1.0-mrs-1.9.0
KafkaManager	1.3.3.1	-
Flink	1.7.0	1.7.0-mrs-1.9.0
Storm	1.2.1	1.2.1-mrs-1.9.0
Flume	1.6.0	1.6.0-mrs-1.9.0
Hue	3.11.0	-
Loader(Sqoop)	1.99.7	1.99.7-mrs-1.9.0
scala	2.11	-

Mapping Between Maven Repository JAR Versions and MRS Component Versions in an MRS 1.8.x Cluster

Table 4-6 Mapping between Maven repository JAR versions and MRS component versions in an MRS 1.8.x cluster

Component	Component Version	JAR Version
ZooKeeper	3.5.1	3.5.1-mrs-1.8.0
Hadoop	2.8.3	2.8.3-mrs-1.8.0
HBase	1.3.1	1.3.1-mrs-1.8.0
OpenTSDB	2.3.0	-
Hive	1.3.0	1.3.0-mrs-1.8.0
Hive_Spark	1.2.1	1.2.1.spark_2.2.1-mrs-1.8.0
Spark	2.2.1	2.2.1-mrs-1.8.0
Carbon	1.6.1	-
Presto	0.215	0.215-mrs-1.8.0
Kafka	1.1.0	1.1.0-mrs-1.8.0
KafkaManager	1.3.3.18	-
Flink	1.7.0	1.7.0-mrs-1.8.0
Storm	1.2.1	1.2.1-mrs-1.8.0
Flume	1.6.0	1.6.0-mrs-1.8.0
Hue	3.11.0	-

Component	Component Version	JAR Version
Loader(Sqoop)	1.99.7	1.99.7-mrs-1.8.0
scala	2.11	-

5 Security Authentication

5.1 Security Authentication Principles and Mechanisms

Function

For clusters in security mode with Kerberos authentication enabled, security authentication is required during application development.

Kerberos, named after the ferocious three-headed guard dog of Hades from Greek mythology, is now used to a concept in authentication. The Kerberos protocol adopts a client-server model and cryptographic algorithms such as AES (Advanced Encryption Standard). It provides mutual authentication, that is, both the client and the server can verify each other's identity. Kerberos is used to prevent interception and replay attacks and protect data integrity. It is a system that manages keys by using a symmetric key mechanism.

Architecture

Kerberos architecture is shown in [Figure 5-1](#) and module description in [Table 5-1](#).

Figure 5-1 Kerberos architecture

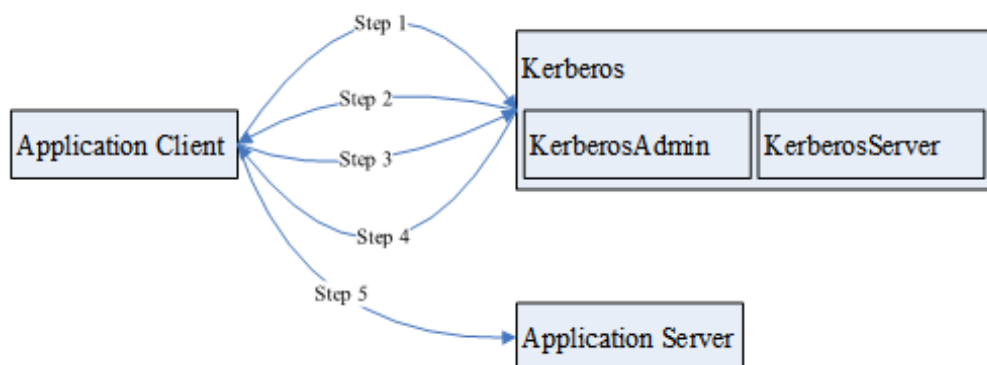


Table 5-1 Module description

Module	Description
Application Client	An application client, which is usually an application that submits tasks or jobs
Application Server	An application server, which is usually an application that an application client accesses
Kerberos	A service that provides security authentication
KerberosAdmin	A process that provides authentication user management
KerberosServer	A process that provides authentication ticket distribution

The process and principle are described as follows:

An application client can be a service in a cluster or a secondary development application of the customer. An application client can submit tasks or jobs to an application service.

1. Before submitting a task or job, the application client needs to apply for a ticket granting ticket (TGT) from the Kerberos service to establish a secure session with the Kerberos server.
2. After receiving the TGT request, the Kerberos service resolves parameters in the request to generate a TGT, and uses the key of the username specified by the client to encrypt the response.
3. After receiving the TGT response, the application client (based on the underlying RPC) resolves the response and obtains the TGT, and then applies for a server ticket (ST) of the application server from the Kerberos service.
4. After receiving the ST request, the Kerberos service verifies the TGT validity in the request and generates an ST of the application service, and then uses the application service key to encrypt the response.
5. After receiving the ST response, the application client packages the ST into a request and sends the request to the application server.
6. After receiving the request, the application server uses its local application service key to resolve the ST. After successful verification, the request becomes valid.

Basic Concepts

The following concepts can help users learn the Kerberos architecture quickly and understand the Kerberos service better. The following uses security authentication for HDFS as an example.

TGT

A TGT is generated by the Kerberos service and used to establish a secure session between an application and the Kerberos server. The validity period of a TGT is 24 hours. After 24 hours, the TGT expires automatically.

The following describes how to apply for a TGT (HDFS is used as an example):

1. Obtain a TGT through an API provided by HDFS.

```
/**
 * login Kerberos to get TGT, if the cluster is in security mode
 * @throws IOException if login is failed
 */
private void login() throws IOException {
    // not security mode, just return
    if (!"kerberos".equalsIgnoreCase(conf.get("hadoop.security.authentication"))) {
        return;
    }

    //security mode
    System.setProperty("java.security.krb5.conf", PATH_TO_KRB5_CONF);

    UserGroupInformation.setConfiguration(conf);
    UserGroupInformation.loginUserFromKeytab(PRINCIPAL_NAME, PATH_TO_KEYTAB);
}
```

2. Run shell commands on the client in kinit mode.

ST

An ST is generated by the Kerberos service and used to establish a secure session between an application and application service. An ST is valid only once.

In FusionInsight products, the generation of an ST is based on the Hadoop-RPC communication. The underlying RPC submits a request to the Kerberos server and the Kerberos server generates an ST.

Sample Authentication Code

```
package com.huaweixxx.bigdata.hdfs.examples;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.security.UserGroupInformation;

public class KerberosTest {
    private static String PATH_TO_HDFS_SITE_XML = KerberosTest.class.getClassLoader().getResource("hdfs-site.xml")
        .getPath();
    private static String PATH_TO_CORE_SITE_XML = KerberosTest.class.getClassLoader().getResource("core-site.xml")
        .getPath();
    private static String PATH_TO_KEYTAB =
        KerberosTest.class.getClassLoader().getResource("user.keytab").getPath();
    private static String PATH_TO_KRB5_CONF =
        KerberosTest.class.getClassLoader().getResource("krb5.conf").getPath();
    private static String PRINCIPAL_NAME = "develop";
    private FileSystem fs;
    private Configuration conf;

    /**
     * initialize Configuration
     */
    private void initConf() {
        conf = new Configuration();
```

```
// add configuration files
conf.addResource(new Path(PATH_TO_HDFS_SITE_XML));
conf.addResource(new Path(PATH_TO_CORE_SITE_XML));
}

/**
 * login Kerberos to get TGT, if the cluster is in security mode
 * @throws IOException if login is failed
 */
private void login() throws IOException {
    // not security mode, just return
    if (!"kerberos".equalsIgnoreCase(conf.get("hadoop.security.authentication"))) {
        return;
    }

    //security mode
    System.setProperty("java.security.krb5.conf", PATH_TO_KRB5_CONF);

    UserGroupInformation.setConfiguration(conf);
    UserGroupInformation.loginUserFromKeytab(PRINCIPAL_NAME, PATH_TO_KEYTAB);
}

/**
 * initialize FileSystem, and get ST from Kerberos
 * @throws IOException
 */
private void initFileSystem() throws IOException {
    fs = FileSystem.get(conf);
}

/**
 * An example to access the HDFS
 * @throws IOException
 */
private void doSth() throws IOException {
    Path path = new Path("/tmp");
    FileStatus fStatus = fs.getFileStatus(path);
    System.out.println("Status of " + path + " is " + fStatus);
    //other thing
}

public static void main(String[] args) throws Exception {
    KerberosTest test = new KerberosTest();
    test.initConf();
    test.login();
    test.initFileSystem();
    test.doSth();
}
}
```

NOTE

1. During Kerberos authentication, you need to configure the file parameters required for configuring the Kerberos authentication, including the keytab path, Kerberos authentication username, and the **krb5.conf** configuration file of the client for Kerberos authentication.
2. Method **login()** indicates calling the Hadoop API to perform Kerberos authentication and generating a TGT.
3. Method **doSth** indicates calling the Hadoop API to access the file system. In this situation, the underlying RPC automatically carries the TGT to Kerberos for verification and then an ST is generated.
4. The preceding code can be used to create **KerberosTest.java** in the HDFS secondary development sample project in security mode and run and view the commissioning result. For details, see [HDFS Development Guide \(Security Mode\)](#).

5.2 Preparing a Developer Account

Scenario

A developer account is used to run the sample project. When developing components for different services, you need to assign different user permissions.

Procedure

Step 1 Log in to FusionInsight Manager.

Step 2 Choose **System > Permission > Role > Create Role**.

1. Enter a role name, for example, **developrole**.
2. Check whether Ranger authentication is enabled. For details, see [How Do I Determine Whether the Ranger Authentication Is Used for a Service?](#)
 - If yes, go to [Step 3](#).
 - If no, edit the role to add the permissions required for service development based on the permission control type of the service. For details, see [Table 5-2](#).

Table 5-2 List of permissions

Service	Permissions to Be Granted
HDFS	In Configure Resource Permission , choose <i>Name of the desired cluster</i> > HDFS > File System , select Read , Write , and Execute for hdfs://hacluster , and click OK .

Service	Permissions to Be Granted
MapReduce/ Yarn	<p>1. In Configure Resource Permission, choose <i>Name of the desired cluster</i> > HDFS > File System > hdfs://hacluster/, and select Read, Write, and Execute for the user. Choose <i>Name of the desired cluster</i> > HDFS > File System > hdfs://hacluster/ > user, select Read, Write, and Execute for mapred.</p> <p>To execute multiple component cases, perform the following operations:</p> <p>Choose <i>Name of the desired cluster</i> > HBase > HBase Scope > global and select the default option create.</p> <p>Choose <i>Name of the desired cluster</i> > HBase > HBase Scope > global > hbase, select hbase:meta, and click Execute.</p> <p>Choose <i>Name of the desired cluster</i> > HDFS > File System > hdfs://hacluster/ > user, and select Read, Write, and Execute for Hive.</p> <p>Choose <i>Name of the desired cluster</i> > HDFS > File System > hdfs://hacluster/ > user > hive, and select Read, Write, and Execute for warehouse.</p> <p>Choose <i>Name of the desired cluster</i> > HDFS > File System > hdfs://hacluster/ > tmp, and select Read, Write and Execute for hive-scratch. If examples exist, select Read, Write, Execute, and recursion for example.</p> <p>Choose <i>Name of the desired cluster</i> > Hive > Hive Read Write Privileges and select Query, Insert, Create and recursion for default. Click OK.</p> <p>2. Edit the role. In Configure Resource Permission, choose <i>Name of the desired cluster</i> > Yarn > Scheduler Queue > root, select the default option Submit, and click OK.</p>
HBase	<p>In Configure Resource Permission, choose <i>Name of the desired cluster</i> > HBase > HBase Scope > global, select the admin, create, read, write, and execute permissions, and click OK.</p>

Service	Permissions to Be Granted
Spark2x	<ol style="list-style-type: none"> 1. (Configure this parameter if HBase is installed.) In Configure Resource Permission, choose <i>Name of the desired cluster</i> > HBase > HBase Scope > global, select the default option create, and click OK. 2. (Configure this parameter if HBase is installed.) Edit the role. In Configure Resource Permission, choose <i>Name of the desired cluster</i> > HBase > HBase Scope > global > hbase, select execute for hbase:meta, and click OK. 3. Edit the role. In Configure Resource Permission, choose <i>Name of the desired cluster</i> > HDFS > File System > hdfs://hacluster/ > user, select Execute for hive, and click OK. 4. Edit the role. In Configure Resource Permission, choose <i>Name of the desired cluster</i> > HDFS > File System > hdfs://hacluster/ > user > hive, select Read, Write, and Execute for warehouse, and click OK. 5. Edit the role. In Configure Resource Permission, choose <i>Name of the desired cluster</i> > Hive > Hive Read Write Privileges, select the default option Create, and click OK. 6. Edit the role. In Configure Resource Permission, choose <i>Name of the desired cluster</i> > Yarn > Scheduler Queue > root, select the default option Submit, and click OK.
Hive	<p>In Configure Resource Permission, choose <i>Name of the desired cluster</i> > Yarn > Scheduler Queue > root, select the default option Submit and Admin, and click OK.</p> <p>NOTE Extra operation permissions required for Hive application development must be obtained from the system administrator.</p>
ClickHouse	<p>In Configure Resource Permission, choose <i>Name of the desired cluster</i> > ClickHouse > ClickHouse Scope and select Create Privilege for the target database Click the database name, select the read and write permissions of the corresponding table based on the task scenario, and click OK.</p>

Service	Permissions to Be Granted
Flink	<ol style="list-style-type: none"> In the Configure Resource Permission table, choose <i>Name of the desired cluster</i> > HDFS > File System > hdfs://hacluster/ > flink, select Read, Write, and Execute, and click Service in the Configure Resource Permission table to return. In Configure Resource Permission, choose <i>Name of the desired cluster</i> > Yarn > Scheduler Queue > root, select the default option Submit, and click OK. <p>NOTE If state backend is set to a path on HDFS, for example, hdfs://hacluster/flink-checkpoint, configure the read, write, and execute permissions on the hdfs://hacluster/flink-checkpoint directory.</p>
GraphBase	-
Kafka	-
Impala	-
Storm/CQL	-
Oozie	<ol style="list-style-type: none"> In Configure Resource Permission, choose <i>Name of the desired cluster</i> > Oozie > Common User Privileges, and click OK. Edit the role. In Configure Resource Permission, choose <i>Name of the desired cluster</i> > HDFS > File System, select Read, Write, and Execute for hdfs://hacluster, and click OK. Edit the role. In Configure Resource Permission, choose <i>Name of the desired cluster</i> > Yarn, select Cluster Admin Operations, and click OK.

Step 3 Choose **System** > **Permission** > **User Group** > **Create User Group** to create a user group for the sample project, for example, **developgroup**.

Step 4 Choose **System** > **Permission** > **User** > **Create** to create a user for the sample project.

Step 5 Enter a username, for example, **developuser**, select a user type and user group to which the user is to be added according to [Table 5-3](#), bind the role **developrole** obtain permissions, and click **OK**.

Table 5-3 User type and user group list

Service	User Type	User Group
HDFS	Machine-Machine	Join the developgroup and supergroup groups. Set the primary group to supergroup .
MapReduce/Yarn	Machine-Machine	Join the developgroup group.
HBase	Machine-Machine	Join the hadoop group.
Spark2x	Machine-Machine/ Human-Machine	Join the developgroup group. If the user needs to interconnect with Kafka, add the Kafkaadmin user group.
Hive	Machine-Machine/ Human-Machine	Join the hive group.

Service	User Type	User Group
Kafka	Machine	Join the kafkaadmin group.
Impala	Machine	Join the impala and supergroup group. Set the primary group to supergroup .
Storm/CQL	Human	Join the storm group.
ClickHouse	Human	Join the developgroup and supergroup (primary) groups and add a role with the ClickHouse permission.
Oozie	Human	Join the hadoop , supergroup , and hive groups If the multi-instance function is enabled for Hive, the user must belong to a specific Hive instance group, for example, hive3 .
GraphBase	Human	Join the graphbaseadmin , graphbasedeveloper , or graphbaseoperator group.

Service	User Type	User Group
Flink	Human-Machine	<p>Join the developgroup and hadoop groups. Set the primary group to developgroup.</p> <p>NOTE If a user wants to interconnect with Kafka, a hybrid cluster with Flink and Kafka components is required, or cross-cluster mutual trust needs to be configured for the cluster with Flink and the cluster with Kafka components. Additionally, the created Flink user is added to the kafkaadmin user group.</p>

Step 6 If Ranger authentication is enabled for the service, in addition to the permissions of the default user group and role, grant required permissions to the user or its role or user group on the Ranger web UI after the user is created. For details, see [Configuring Component Permission Policies](#).

Step 7 On the homepage of FusionInsight Manager, choose **System > Permission > User**. Select **developuser** from the user list and click **More > Download Authentication Credential** to download authentication credentials. Save the downloaded package and decompress the file to obtain **user.keytab** and **krb5.conf** files. These files are used for security authentication during the sample project. For details, see the corresponding service development guide.

 **NOTE**

If the user type is human-machine, you need to change the initial password before downloading the authentication credential file. Otherwise, **Password has expired - change password to reset** is displayed when you use the authentication credential file. As a result, security authentication fails.

----End

5.3 Handling an Authentication Failure

Symptom

An authentication failure occurs during the commissioning and running of an example project.

Troubleshooting Process

There are many reasons that will cause an authentication failure. You are advised to perform the following steps for troubleshooting in different scenarios:

Step 1 Check whether the network connection between the device where this application runs and the cluster is normal, and check whether the TCP and UDP ports required by Kerberos authentication can be accessed.

Step 2 Check whether each configuration file is correctly read and stored in a correct directory.

- Step 3** Ensure that the username and keytab file are obtained according to the operation guide.
 - Step 4** Check whether the configuration information is properly set before initiating an authentication.
 - Step 5** Check whether multiple authentication requests are initiated in the same process. That is, check whether the **login()** method is called repeatedly.
 - Step 6** If the problem persists, contact Huawei engineers for further analysis.
- End

Authentication Failure Examples

"clock skew too great" is displayed during authentication.

- Step 1** Check the cluster time.
 - Step 2** Check the time of the machine where the development environment is located. The difference between the machine time and the cluster time must be less than 5 minutes.
- End

"(Receive time out) can not connect to kdc server" is displayed during authentication.

- Step 1** Check whether the content of the **krb5.conf** file is correct. That is, check whether the file content is the same as the service IP address configuration of KerberoServer in the cluster.
 - Step 2** Check whether the Kerberos service is running properly.
 - Step 3** Check whether the firewall is disabled.
- End

An exception occurs when a client application submits a task to the Hadoop cluster, and the message "Failed to find any Kerberos tgt" or "No valid credentials provided" is displayed.

- Step 1** Check whether the **kinit** command is executed. If no, perform kinit authentication before submitting the task.
- Step 2** In the multi-thread scenario, when the process starts, call the `loginfromkeytab` function provided by Hadoop to log in to KDC to obtain TGT. Before submitting tasks, call the `reloginFromKeytab` function to update the TGT.

```
// After the first login succeeds, set user group information.  
UserGroupInformation.loginUserFromKeytab(this.userPrincipal,this.keytabFile);  
// Before the thread submits the task:  
UserGroupInformation.getLoginUser().reloginFromKeytab();
```

- Step 3** If multiple scripts use the **kinit** command to authenticate the same user, run the **export KRB5CCNAME=keytab_path** command before running the **kinit** command in each script to ensure that the path specified by **KRB5CCNAME** in each script process is inconsistent.
- End

6 CQL Development Guide (Security Mode)

6.1 Overview

6.1.1 Application Development Overview

Continuous Query Language (CQL) is a query language used for data streams. Compared with standard SQL, CQL introduces the concept of window. Data is stored in memory so that in-memory computing can be quickly implemented. CQL query results are computing results at a specific moment of data streams.

CQL has the following features:

- **Simple usage:** Similar to standard SQL syntax, CQL syntax supports SQL-based quick development.
- **Rich functions:** Except various standard basic SQL expressions, CQL provides window, pre-window filtering, post-window filtering, and concurrency setting functions for stream processing scenarios to meet requirements in multiple real-time service processing scenarios.
- **Easy to scale:** CQL provides an extension interface to supporting increasingly complex service scenarios. Users can customize the input, output, serialization, and deserialization to meet specific service requirements.
- **Easy to debug:** CQL provides detailed explanation of error codes, facilitating users to rectify faults.

6.1.2 Common Concepts

Window

Window: A way for processing unbounded and streaming events. A window sets an event stream to a static view at a moment for various query operations on database tables.

Expression

An expression is a set of symbols and operators. The CQL parsing engine processes an expression to obtain single values. A simple expression can be a constant, variable, or function. Two or more simple expressions can be combined as a complex expression by using operators.

keytab file

A key file for storing user information. Applications use the key file for API authentication on FusionInsight MRS.

6.1.3 Development Process

This document describes how to develop CQL applications based on Java APIs.

Figure 6-1 and **Table 6-1** describe each phase of the development process.

Figure 6-1 CQL application development process

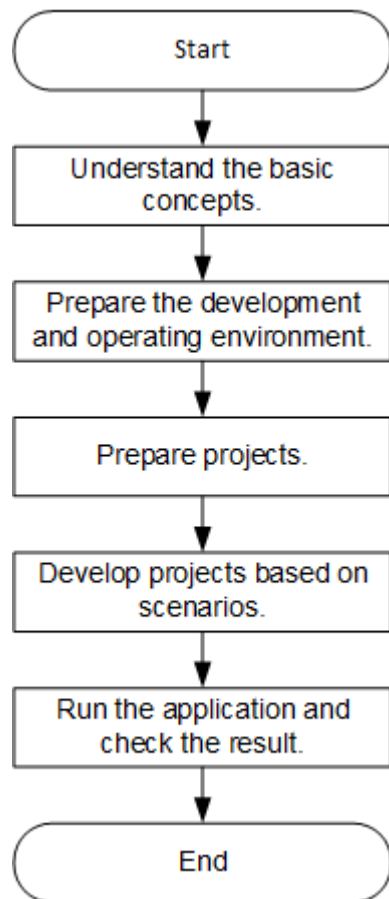


Table 6-1 CQL application development process

Phase	Description	Reference Document
Understand the basic concepts.	Before application development, learn basic concepts of CQL and understand the scenario requirements.	Common Concepts
Prepare the development and operating environment.	You can use the IntelliJ IDEA tool to invoke Java APIs to develop CQL applications. The CQL operating environment is a Storm client. Install and configure the client according to the guide.	Preparing for Development and Operating Environment
Prepare projects.	CQL provides sample projects for different scenarios. You can import a sample project to learn the application.	Configuring and Importing Sample Projects
Develop projects based on scenarios.	Sample projects are provided, including projects about how to generate random numbers and how to collect statistics by condition.	Developing an Application
Run the application and check the result.	Storm sample programs are running in the Linux OS. IntelliJ IDEA code needs to be packaged into a JAR package.	Running an Application

6.2 Environment Preparation

6.2.1 Environment Preparation Overview

This development guide describes CQL-based IntelliJ IDEA sample projects and Linux command line interfaces of the Storm component so that developers can quickly be familiar with CQL.

Prepare clients for developing and submitting applications. The client for developing applications provides modes for submitting applications from an IntelliJ IDEA client and instance programs used by CQL Java APIs. Generally, applications are developed in Windows and submitted in Linux.

6.2.2 Preparing for Development and Operating Environment

Preparing Development Environment

[Table 6-2](#) describes the environment required for application development.

Table 6-2 Development environment

Item	Description
OS	<ul style="list-style-type: none"> Development environment: Windows 7 or later version is supported. Running environment: Windows or Linux If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> Oracle JDK versions: 1.8 IBM JDK versions: 1.8.5.11 For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> OpenJDK: 1.8.0_272
IntelliJ IDEA installation and configuration	This tool is used for application development. The version of the tool must be 2019.1 or other compatible versions.
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.
User development preparation	See Preparing a Developer Account for configuration.
7-zip	It is a tool used to decompress .zip and .rar packages. The 7-Zip 16.04 is supported.

Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.

- a. **Log in to the FusionInsight Manager portal** and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is

FusionInsight_Cluster_1_Services_Client.tar, decompress it to obtain **FusionInsight_Cluster_1_Services_ClientConfig.tar file**. Then, decompress **FusionInsight_Cluster_1_Services_ClientConfig.tar file** to the **D:\FusionInsight_Cluster_1_Services_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight_Cluster_1_Services_ClientConfig\Storm\config**. Obtain the CQL-related configuration file.

Table 6-3 describes the main configuration files.

Table 6-3 Configuration file

Document Name	Function
streaming-site.xml	Sets parameters required for the CQL sample code
user.keytab	Provides user information for Kerberos security authentication.
krb5.conf	Kerberos server configuration information

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.

- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client on the node. For example, the client installation directory is `/opt/client`.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. [Log in to the FusionInsight Manager portal](#). Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user `root`. Go to the decompression path of the cluster client and copy all configuration files `streaming-site.xml` in the `FusionInsight_Cluster_1_Services_ClientConfig/Storm/config` directory to the `conf` directory where the compiled JAR package is stored for subsequent commissioning, for example, `/opt/client/conf`.

For example, if the client software package is `FusionInsight_Cluster_1_Services_Client.tar` and the download path is `/tmp/FusionInsight-Client` on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Storm/config/streaming-site.xml root@IP address of the client
node:/opt/client/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 6-4](#) describes the main configuration files.

Table 6-4 Configuration file

Document Name	Function
streaming-site.xml	Sets parameters required for the CQL sample code
user.keytab	Provides user information for Kerberos security authentication.
krb5.conf	Kerberos server configuration information

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the `hosts` file on the client node. You are advised to check whether the `/etc/hosts` file contains the host names of the nodes in the cluster. If no,

manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

6.2.3 Configuring and Importing Sample Projects

Scenario

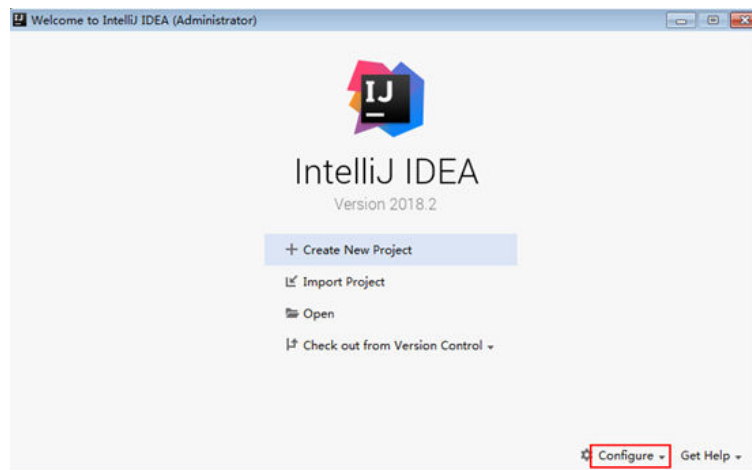
To execute sample codes of CQL, perform the following operations:

Application programs can be submitted through the Windows or Linux environment during application development.

Windows Environment

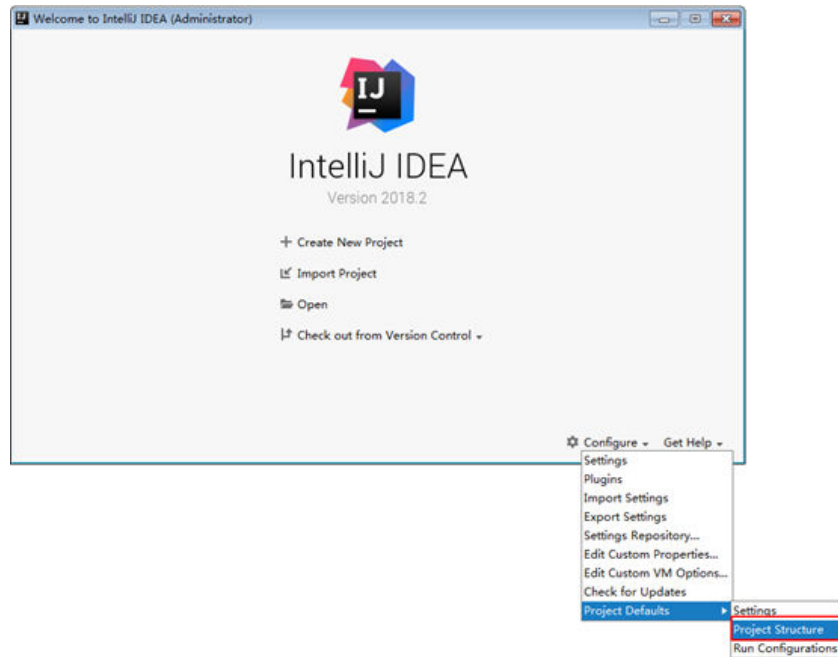
- Step 1** Obtain the sample project folder **cql-examples** in the `src\storm-examples` directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the **user.keytab** and **krb5.conf** files obtained in the [Preparing a Developer Account](#) section and the cluster configuration file obtained in section [Preparing an Operating Environment](#) to the `cql-examples\src\main\resources` directory of the sample project.
- Step 3** After installing the IntelliJ IDEA and JDK, configure the JDK in the IntelliJ IDEA.
 1. Start IntelliJ IDEA and choose **Configure**.

Figure 6-2 Quick Start



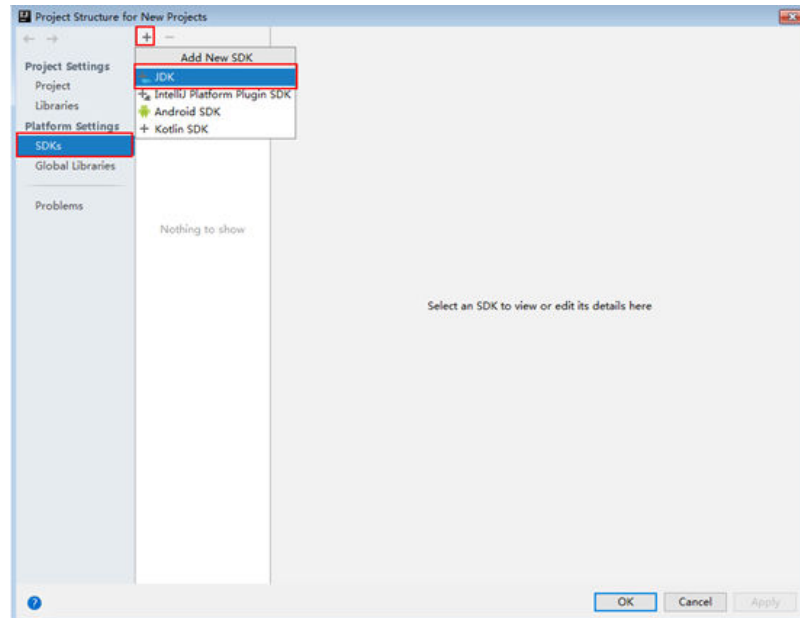
2. Choose **Project Defaults > Project Structure** from the drop-down list box.

Figure 6-3 Configure



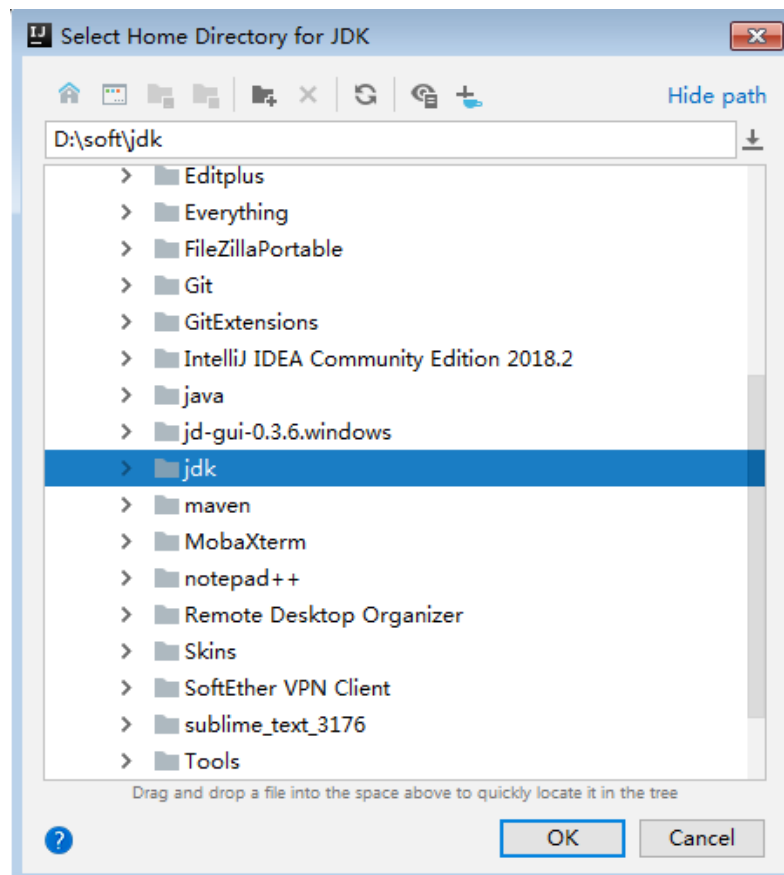
3. On the **Project Structure for New Projects** page that is displayed, select **SDKs** and click the plus sign (+) to add JDK.

Figure 6-4 Project Structure for New Projects



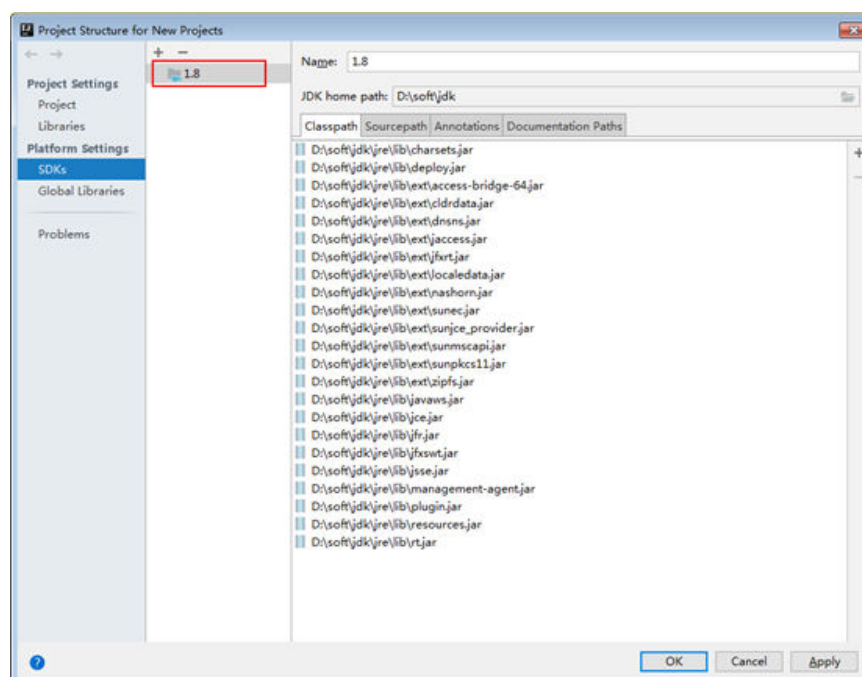
4. In the displayed **Select Home Directory for JDK** window, select the JDK directory, and click **OK**.

Figure 6-5 Select Home Directory for JDK



5. After selecting the JDK, click **OK** to complete the configuration.

Figure 6-6 Completing the JDK configuration



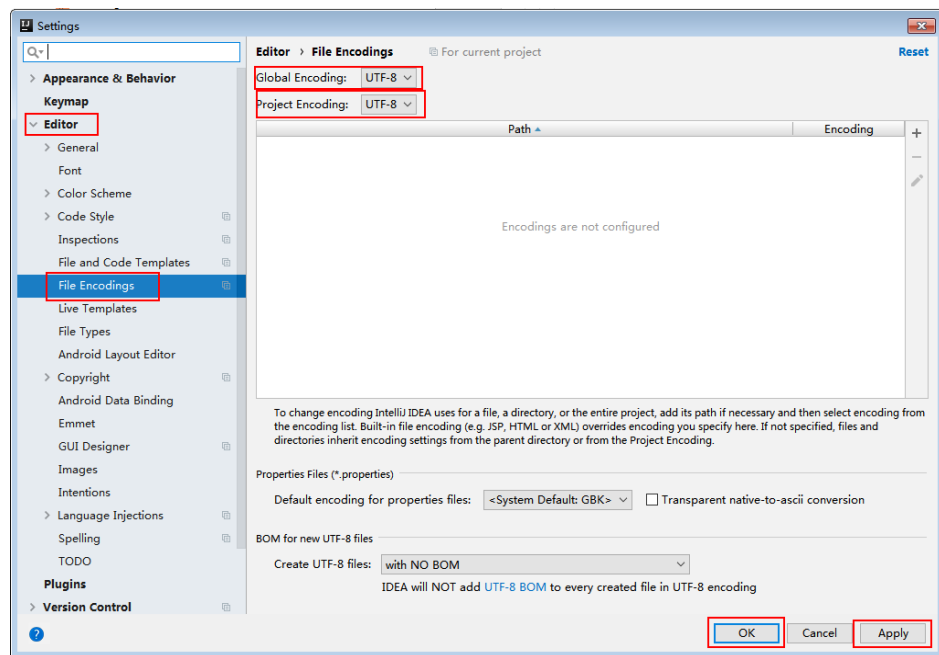
Step 4 Import the example project into the IntelliJ IDEA development environment.

1. Choose **Open**.
The dialog box for browsing directories is displayed.
2. Select the sample project directory and click **OK**.

Step 5 Set the IntelliJ IDEA text file coding format to prevent invalid characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.
The **Settings** window is displayed.
2. Choose **Editor > File Encodings** in the navigation tree. In the **Project Encoding** area and **Global Encoding** area, set the parameter to **UTF-8**, click **Apply**, and then click **OK**, as shown in [Figure 6-7](#).

Figure 6-7 Setting the IntelliJ IDEA coding format



Step 6 Modify the parameter names of **userPrincipal** and **keytabPath** in the **setSecurityConfs** method in the **CQLExample** class of the **com.huawei.streaming.example.cql** package of the sample program.

```
String userPrincipal = "developuser@HADOOP.COM";  
String keytabPath = confDir + "user.keytab";
```

----End

Linux Environment

Step 1 Initialize client environment variables.

Access the installation directory, and run the following command to import environment variables:

```
source bigdata_env
```

Step 2 Use the development user created in [Preparing a Developer Account](#) section to execute **kinit** for secure login.

kinit *user name*

For example, run the following command:

kinit developuser

Enter a password as prompted. If no error message appears, kerberos authentication is completed for the user.

Step 3 Access **Storm/streaming-cql-1.0/bin** in the installation directory and run the following command:

./cql

If information about the currently running application is found and no error occurs after you run the following command, the commands are successfully run.

```
[root@host1 bin]# ./cql
End CQL with ';' and end client with 'exit;'
Streaming>show applications;
applicationName  Status  Num_workers  Uptime_secs
Streaming>
```

----End

6.3 Developing an Application

6.3.1 Scenario

Scenario

A user develops an application to filter out all data whose ID is greater than 5, group the data by field type, and calculates the amount of data of each type in every 20 seconds.

Development Idea

Compile CQL statements and submit the statements by using Java APIs on IntelliJ IDEA or by using CQL CLI on a Linux client.

For details about certain code, see [Example Codes](#). For details about all Java API code, see the cql-examples sample project.

6.3.2 Example Codes

Function Description

Random data is generated by following the rule that one piece of data is generated per second, and the data is sent to other operators for processing. All data whose ID is greater than 5 is filtered out, data is grouped by field type, the number of types is calculated every 20 seconds, and results are recorded in application log files.

Code Sample

The following provides CQL statement examples. Such CQL statements are in the **example.cql** file in the **src/main/resources** directory of the sample project.

```
/*
 * CQL sample project, used for generating random data in the operator for generating random data.
 * Record processed data on a console (in log files).
 */

--Generate one piece of random data per second.
CREATE INPUT STREAM s
(id INT, name STRING, type INT)
SOURCE randomgen
  PROPERTIES
    ("timeUnit" = "SECONDS", "period" = "1",
     "eventNumPerPeriod" = "1", "isSchedule" = "true");

--Record calculation results on a console (in log files).
CREATE OUTPUT STREAM rs
(type INT, cc INT)
SINK ConsoleOutput;

--Calculate the data whose ID is greater than 5 every 20 seconds and group the data by field type.
--Time jump window, generating calculation results every 20 seconds.
INSERT INTO STREAM rs SELECT type, count(id) as cc
FROM s[RANGE 20 SECONDS BATCH] WHERE id > 5 GROUP BY type;
--Submit an application.
SUBMIT APPLICATION example;
```

Precautions


If the security service is enabled for a cluster, verify that you have logged in in security mode before submitting an application.

If the CQL statement contains Chinese characters, change the encoding format of the Linux SSH client to UTF-8.

6.4 Running an Application

CQL programs can be submitted on IntelliJ IDEA or CQL CLI in Linux.

Procedure for Submitting CQL Statements on a Windows Client

Right-click the CQLExample class on IntelliJ IDEA and choose  from the shortcut menu to submit a task.

Procedure for Submitting CQL Statements on a Linux Client

Submit CQL statements in CQL CLI on a Linux client, as shown in [Figure 6-8](#).

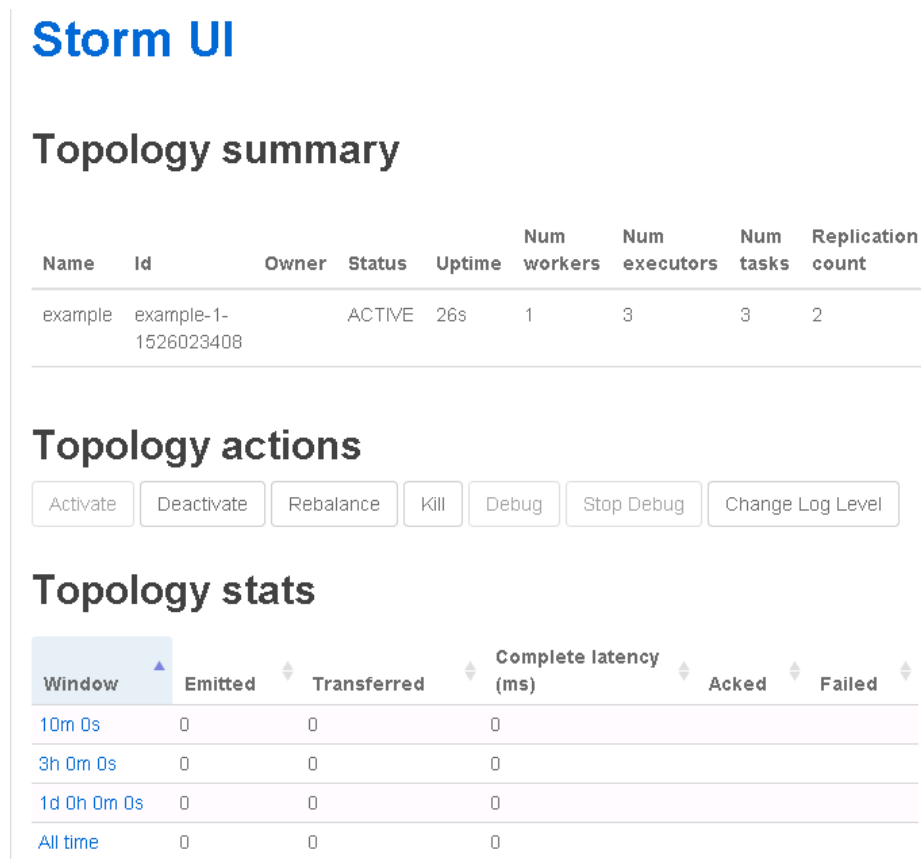
Figure 6-8 Submitting CQL statements in CLI

```
>CREATE INPUT STREAM s
>  (id INT, name STRING, type INT)
> SOURCE randomgen
>   PROPERTIES
>     ( "timeUnit" = "SECONDS", "period" = "1",
>       "eventNumPerPeriod" = "1", "isSchedule" = "true");
Streaming>
>CREATE OUTPUT STREAM rs
>(type INT, cc INT)
> SINK ConsoleOutput;
Streaming>
>INSERT INTO STREAM rs SELECT type, count(id) as cc FROM s[RANGE 2
0 SECONDS BATCH] WHERE id > 5 GROUP BY type;
Streaming>
>SUBMIT APPLICATION example;
Streaming>show applications;
applicationName      Status      Num_workers  Uptime_secs
example              ACTIVE      1             7
Streaming>
```

Viewing Results

- Step 1** Log in to FusionInsight Manager.
- Step 2** Choose **Cluster** > *Name of the desired cluster* > **Service** > **Storm** and click the link next to **Storm WebUI** to enter Storm UI.
- Step 3** Click an example application in Storm UI and view application running status, as shown in [Figure 6-9](#).

Figure 6-9 Storm application execution page



Topology stats collects statistics about the total volume of data sent between operators in different time segments.

Spouts collects statistics about the total number of messages sent in the period from the time when the spout operator starts to the current time. **Bolts** collects statistics about the total number of messages sent by the Count and split operators, as shown in [Figure 6-10](#).

Figure 6-10 Total volume of data sent by the Storm application operator

Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked
RandomGen_001	1	1	100	100	0.000	0

Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execu latenc (ms)
Aggregator_004	1	1	180	180	0.000	0.000
ConsoleOutput_002	1	1	0	0	0.000	0.200

----End

7 CQL Development Guide (Normal Mode)

7.1 Overview

7.1.1 Application Development Overview

Continuous Query Language (CQL) is a query language used for data streams. Compared with standard SQL, CQL introduces the concept of window. Data is stored in memory so that in-memory computing can be quickly implemented. CQL query results are computing results at a specific moment of data streams.

CQL has the following features:

- **Simple usage:** Similar to standard SQL syntax, CQL syntax supports SQL-based quick development.
- **Rich functions:** Except various standard basic SQL expressions, CQL provides window, pre-window filtering, post-window filtering, and concurrency setting functions for stream processing scenarios to meet requirements in multiple real-time service processing scenarios.
- **Easy to scale:** CQL provides an extension interface to supporting increasingly complex service scenarios. Users can customize the input, output, serialization, and deserialization to meet specific service requirements.
- **Easy to debug:** CQL provides detailed explanation of error codes, facilitating users to rectify faults.

7.1.2 Common Concepts

Window

Window: A way for processing unbounded and streaming events. A window sets an event stream to a static view at a moment for various query operations on database tables.

Expression

An expression is a set of symbols and operators. The CQL parsing engine processes an expression to obtain single values. A simple expression can be a constant, variable, or function. Two or more simple expressions can be combined as a complex expression by using operators.

7.1.3 Development Process

This document describes how to develop CQL applications based on Java APIs.

Figure 7-1 and **Table 7-1** describe the phases in the development process.

Figure 7-1 CQL application development process

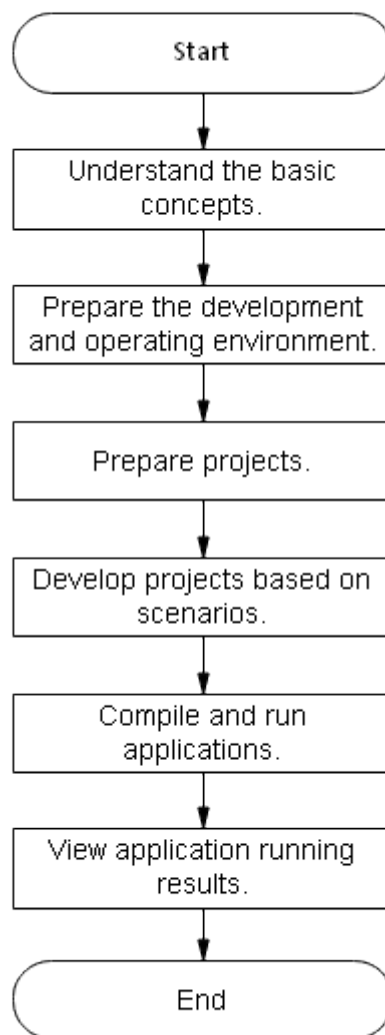


Table 7-1 CQL application development process

Phase	Description	Reference Document
Understand the basic concepts.	Before application development, learn basic concepts of CQL and understand the scenario requirements.	Common Concepts
Prepare the development and operating environment.	Use the IntelliJ IDEA tool to call Java APIs to develop CQL applications. The CQL operating environment is a Storm client. Install and configure the client according to the guide.	Preparing for Development and Operating Environment
Prepare projects.	CQL provides sample projects for different scenarios. You can import a sample project to learn applications.	Configuring and Importing Sample Projects
Develop projects based on scenarios.	Sample projects are provided, including projects about how to generate random numbers and how to collect statistics by condition.	Developing an Application
Compile and run applications.	Storm sample applications are running in the Linux OS. IntelliJ IDEA code needs to be packaged into a JAR package.	Running an Application
View application running results.	Application running results are exported to a path you specify. You can also view the application running status on the UI.	Viewing Commissioning Results

7.2 Environment Preparation

7.2.1 Environment Preparation Overview

This development guide describes CQL-based IntelliJ IDEA sample projects and Linux command line interfaces of the Storm component so that developers can quickly be familiar with CQL.

Prepare clients for developing and submitting applications. The client for developing applications provides modes for submitting applications from an IntelliJ IDEA client and instance programs used by CQL Java APIs. Generally, applications are developed in Windows and submitted in Linux.

7.2.2 Preparing for Development and Operating Environment

Preparing Development Environment

[Table 7-2](#) describes the environment required for application development.

Table 7-2 Development environment

Item	Description
OS	<ul style="list-style-type: none"> Development environment: Windows 7 or later version is supported. Operating environment: Windows or Linux If the application needs to be commissioned locally, the operating environment must be able to communicate with network on the cluster service plane.
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK 1.8.0_272. Other JDKs cannot be used.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> Oracle JDK versions: 1.8 IBM JDK versions: 1.8.5.11 TaiShan client: OpenJDK 1.8.0_272
IntelliJ IDEA installation and configuration	<p>This tool is used for application development. The version of the tool must be 2019.1 or other compatible versions.</p>
Installation of Maven	<p>Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.</p>

Item	Description
7-zip	It is a tool used to decompress *.zip and *.rar files. The 7-Zip 16.04 is supported.

Preparing an Operating Environment

During application development, prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning application; configure the network connection, and commission the application in Windows.
 - a. **Log in to the FusionInsight Manager portal** and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is

FusionInsight_Cluster_1_Services_Client.tar, decompress it to obtain **FusionInsight_Cluster_1_Services_ClientConfig.tar**. Then, decompress **FusionInsight_Cluster_1_Services_ClientConfig.tar** file to the **D:\FusionInsight_Cluster_1_Services_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight_Cluster_1_Services_ClientConfig\Storm\config**. Obtain the CQL-related configuration file.

Table 7-3 describes the main configuration files.

Table 7-3 Configuration file

Document Name	Function
streaming-site.xml	Sets parameters required for the CQL sample code.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, C: \WINDOWS\system32\drivers\etc\hosts.
- If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client on the node. For example, the client installation directory is /opt/client/.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. [Log in to the FusionInsight Manager portal](#). Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy the **streaming-site.xml** configuration file in the **FusionInsight_Cluster_1_Services_ClientConfig/Storm/config** directory to the client node for subsequent commissioning, for example, in the **/opt/client/conf** directory.

For example, if the client software package is **FusionInsight_Cluster_1_Services_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Storm/config/streaming-site.xml root@IP address of the client node:/opt/client/conf
```

[Table 7-4](#) describes the main configuration files.

Table 7-4 Configuration file

Document Name	Function
streaming-site.xml	Sets parameters required for the CQL sample code.

- c. Check the network connection of the client node.
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no,

manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

7.2.3 Configuring and Importing Sample Projects

Scenario

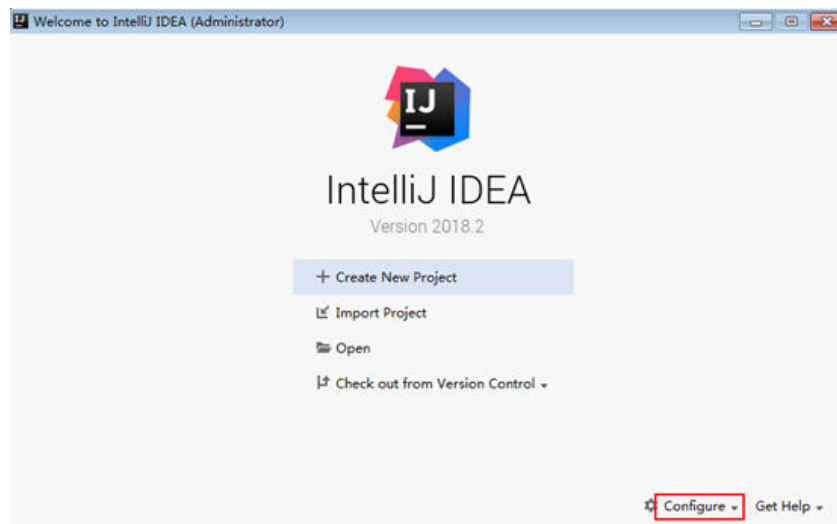
To execute sample codes of CQL, perform the following operations:

Applications can be submitted through the Windows or Linux environment during application development.

Windows Environment

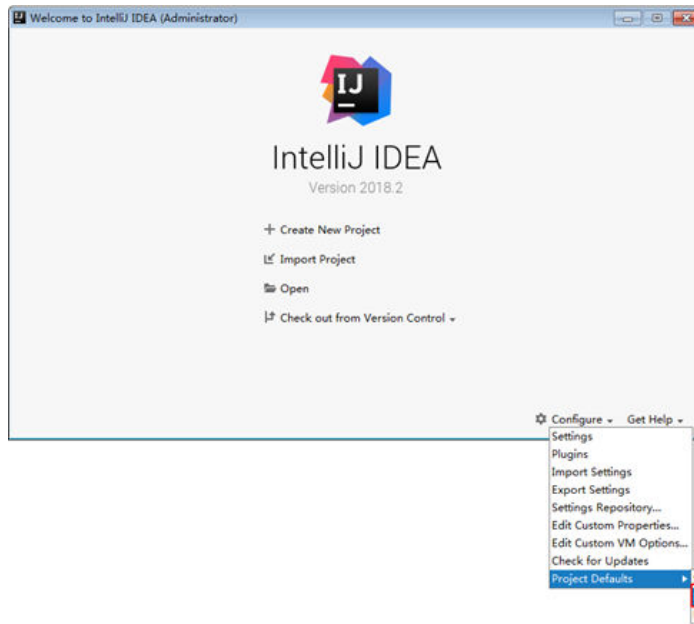
- Step 1** Obtain the sample project folder **cql-examples** in the `src\storm-examples` directory where the sample code is decompressed. For details, see [Configuring Huawei Open-Source Mirrors](#).
- Step 2** Copy the cluster configuration file obtained in section [Preparing an Operating Environment](#) to the `cql-examples\src\main\resources` directory of the sample project.
- Step 3** After installing the IntelliJ IDEA and JDK, configure the JDK in the IntelliJ IDEA.
 1. Start IntelliJ IDEA and choose **Configure**.

Figure 7-2 Quick Start



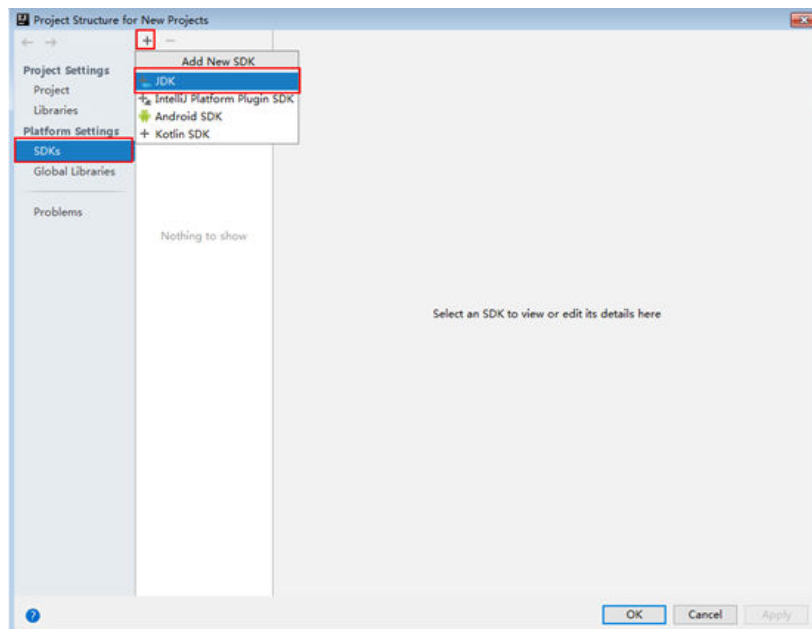
2. Choose **Project Defaults > Project Structure** from the drop-down list box.

Figure 7-3 Configure



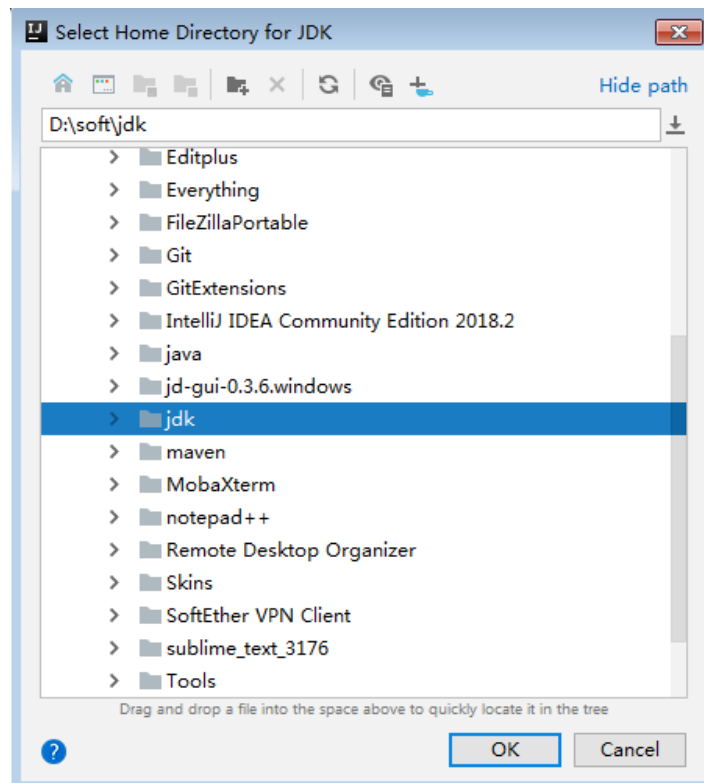
3. On the **Project Structure for New Projects** page that is displayed, select **SDKs** and click the plus sign (+) to add JDK.

Figure 7-4 Project Structure for New Projects



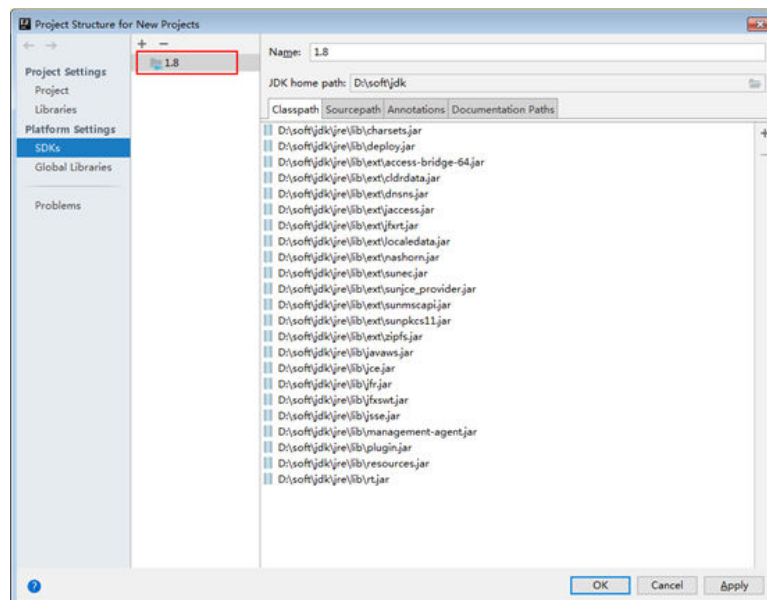
4. In the displayed **Select Home Directory for JDK** window, select the JDK directory, and click **OK**.

Figure 7-5 Select Home Directory for JDK



5. After selecting the JDK, click **OK** to complete the configuration.

Figure 7-6 Completing the JDK configuration



Step 4 Import the sample project into the IntelliJ IDEA development environment.

1. Choose **Open**.
2. The dialog box for browsing directories is displayed.
3. Select the sample project directory and click **OK**.

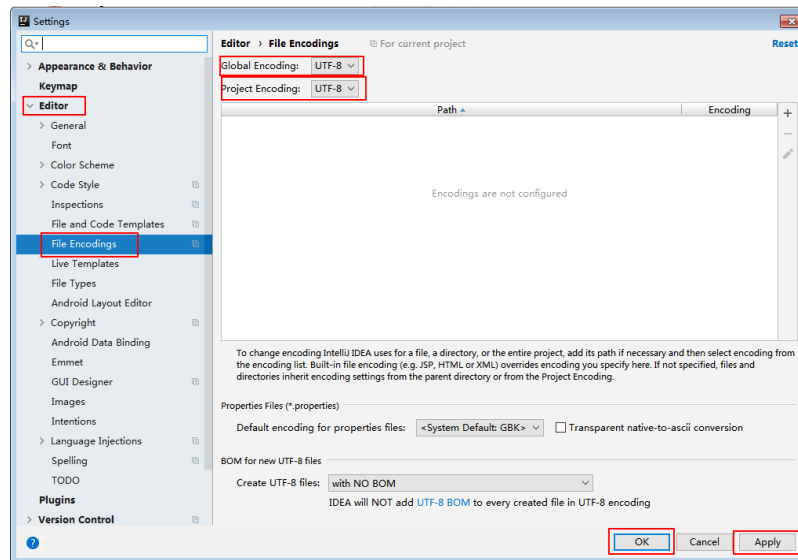
Step 5 Set the IntelliJ IDEA text file coding format to prevent invalid characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.

The **Settings** window is displayed.

2. Choose **Editor > File Encodings** in the navigation tree. In the **Project Encoding** area and **Global Encoding** area, set the parameter to **UTF-8**, click **Apply**, and then click **OK**, as shown in [Figure 7-7](#).

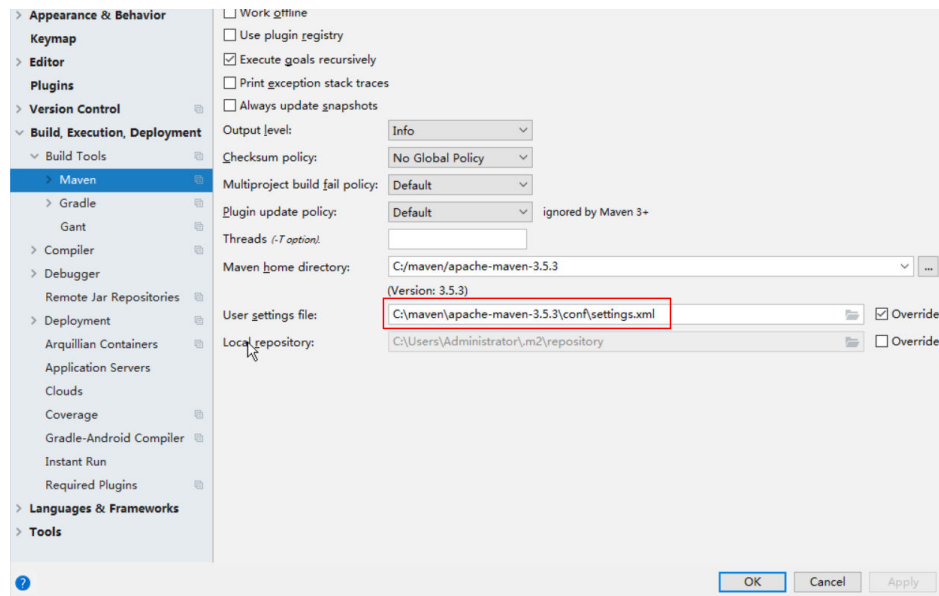
Figure 7-7 Setting the IntelliJ IDEA coding format



Step 6 Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).

When using the IntelliJ IDEA development tool, you can choose **File > Settings > Build, Execution, Deployment > Build Tools > Maven** to view the directory where the **settings.xml** file is stored. Ensure that the directory is **<Local Maven installation directory>\conf\settings.xml**.

Figure 7-8 Directory for storing the settings.xml file



----End

Linux Environment

Step 1 Initialize client environment variables.

Access the installation directory `/opt/client/`, and run the following command to import environment variables:

```
source bigdata_env
```

Step 2 Access `Storm/streaming-cql-*/bin` in the installation directory and run the following command:

```
./cql
```

If information about the currently running application is found and no error occurs after you run the following command, the commands are successfully run.

```
[root@host1 bin]# ./cql
End CQL with ';' and end client with 'exit;'
Streaming>show applications;
applicationName  Status  Num_workers  Uptime_secs
Streaming>
```

----End

7.3 Developing an Application

7.3.1 Scenario

Scenario

A user develops an application to filter out all data whose ID is greater than 5, group the data by field type, and calculates the amount of data of each type in every 20 seconds.

Development Idea

Compile CQL statements and submit the statements by using Java APIs on IntelliJ IDEA or by using CQL CLI on a Linux client.

For details about certain code, see [Example Codes](#). For details about all Java API code, see the `cql-examples` sample project.

7.3.2 Example Codes

Function Description

Random data is generated by following the rule that one piece of data is generated per second, and the data is sent to other operators for processing. All data whose ID is greater than 5 is filtered out, data is grouped by field type, the number of types is calculated every 20 seconds, and results are recorded in application log files.

Code Sample

The following provides CQL statement examples. Such CQL statements are in the `example.cql` file in the `src/main/resources` directory of the sample project.

```
/*
 * CQL sample project, used for generating random data in the operator for generating random data.
 * Record processed data on a console (in log files).
 */

--Generate one piece of random data per second.
CREATE INPUT STREAM s
(id INT, name STRING, type INT)
SOURCE randomgen
  PROPERTIES
    ("timeUnit" = "SECONDS", "period" = "1",
     "eventNumPerPeriod" = "1", "isSchedule" = "true");

--Record calculation results on a console (in log files).
CREATE OUTPUT STREAM rs
(type INT, cc INT)
SINK ConsoleOutput;

--Calculate the data whose ID is greater than 5 every 20 seconds and group the data by field type.
--Time jump window, generating calculation results every 20 seconds.
INSERT INTO STREAM rs SELECT type, count(id) as cc
FROM s[RANGE 20 SECONDS BATCH] WHERE id > 5 GROUP BY type;

--Submit an application.
SUBMIT APPLICATION example;
```

Precautions

If the security service is enabled for a cluster, verify that you have logged in in security mode before submitting an application.

If the CQL statement contains Chinese characters, change the encoding format of the Linux SSH client to UTF-8.

7.4 Running an Application

7.4.1 Debugging, Compiling, and Running an Application in Windows

Scenario

Run applications in a Windows development environment after application code is developed. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host.

Procedure


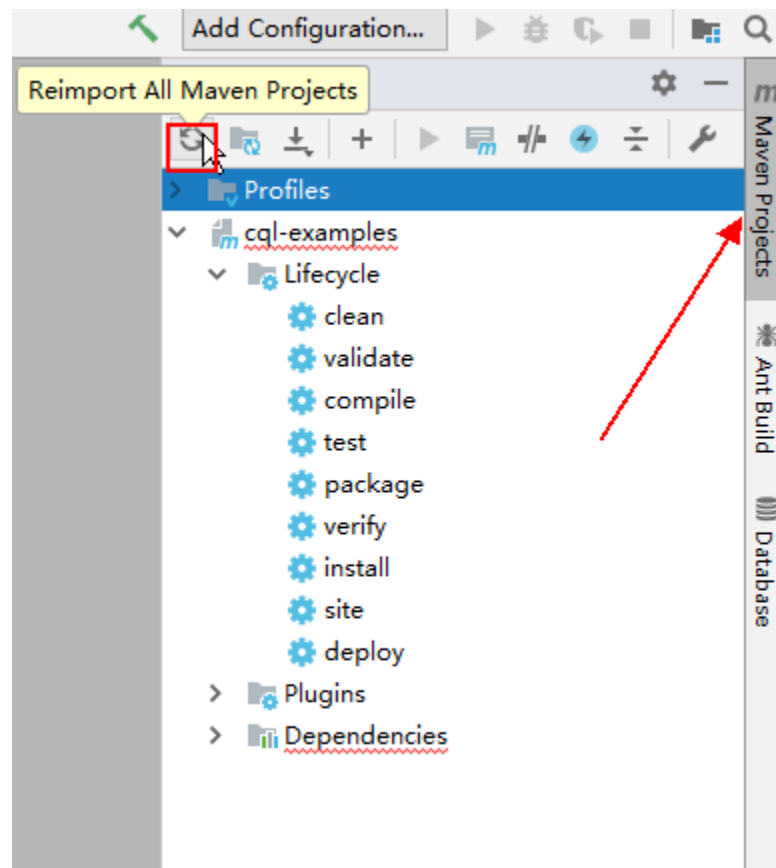
- Step 1** Click  (Reimport All Maven Projects) in the **Maven Projects** window in IntelliJ IDEA to import the Maven project dependency.

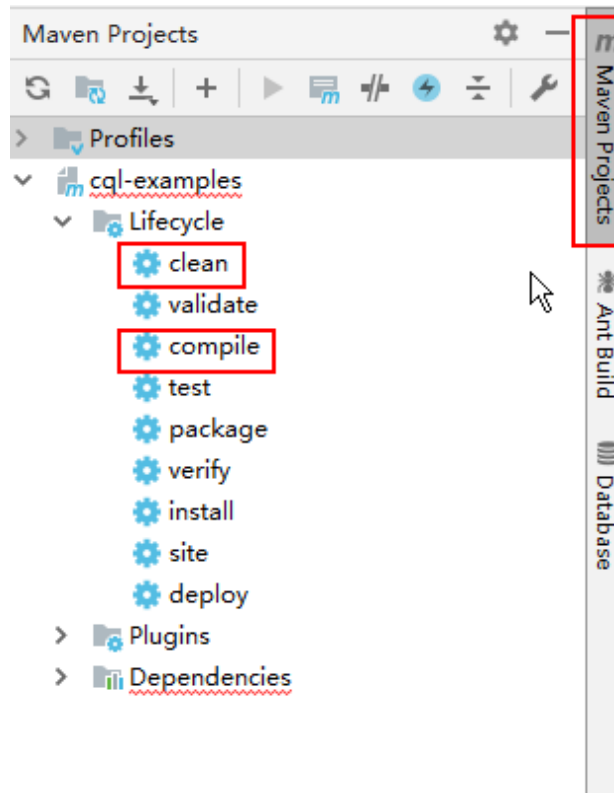
Figure 7-9 Reimport All Maven Projects



Step 2 Compile and run an application.

1. Compile and run the application.
 - Choose **Maven Projects** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.
 - Choose **Maven Projects** > *Sample project name* > **Lifecycle** > **compile** and double-click **compile** to run the **compile** command of Maven.

Figure 7-10 clean and compile tools of Maven



An example of the compilation result is as follows:

```
"C:\Program Files (x86)\Java\jdk1.8.0_73\bin\java.exe" -Dmaven.multiModuleProjectDirectory=C:\Users\
Administrator\Desktop\huaweicloud-mrs-example-mrs-3.0.2\src\storm-examples\cql-examples "-
Dmaven.home=C:\Program Files\JetBrains\IntelliJ IDEA 2018.2.8\plugins\maven\lib\maven3" "-
Dclassworlds.conf=C:\Program Files\JetBrains\IntelliJ IDEA 2018.2.8\plugins\maven\lib\maven3\bin
\m2.conf" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2018.2.8\lib\idea_rt.jar=61287:C:
\Program Files\JetBrains\IntelliJ IDEA 2018.2.8\bin" -Dfile.encoding=UTF-8 -classpath "C:\Program Files
\JetBrains\IntelliJ IDEA 2018.2.8\plugins\maven\lib\maven3\boot\plexus-classworlds-2.5.2.jar"
org.codehaus.classworlds.Launcher -Didea.version=2018.2.8 -s C:\maven\apache-maven-3.5.3\conf
\settings.xml compile
[INFO] Scanning for projects...
[INFO] -----
[INFO] Building cql-examples 8.0.2
[INFO] -----
[WARNING] The POM for ring:ring-core:jar:1.2.2 is missing, no dependency information available
[WARNING] The POM for org.jboss.netty:netty:jar:3.10.4.Final is missing, no dependency information
available
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ cql-examples ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 2 resources
[INFO] ---
[INFO] --- maven-compiler-plugin:2.0.2:compile (default-compile) @ cql-examples ---
```

```
[INFO] Nothing to compile - all classes are up to date
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.171 s
[INFO] Finished at: 2021-04-15T14:52:50+08:00
[INFO] Final Memory: 12M/29M
[INFO] -----
```

Process finished with exit code 0

2. Run the application.


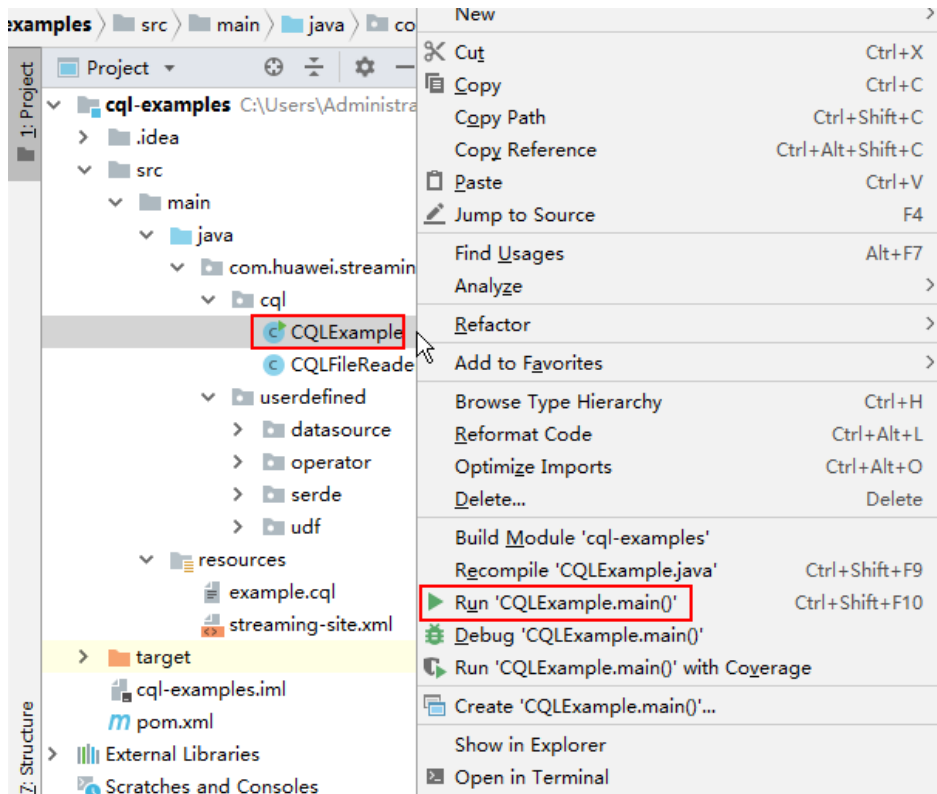
Right-click the **CQLExample** class in IntelliJ IDEA and click  in the shortcut menu to run the application, as shown in [Figure 7-11](#).

Figure 7-11 Running the application



The application running success result may contain the following information:

```
"C:\Program Files (x86)\Java\jdk1.8.0_73\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ
IDEA 2018.2.8\lib\idea_rt.jar=59898:C:\Program Files\JetBrains\IntelliJ IDEA 2018.2.8\bin" -
Dfile.encoding=UTF-8 -classpath "
...
SLF4J: Found binding in [jar:file:/C:/Users/Administrator/.m2/repository/org/apache/logging/log4j/
log4j-slf4j-impl/2.13.3/log4j-slf4j-impl-2.13.3.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/C:/Users/Administrator/.m2/repository/org/slf4j/slf4j-log4j12/1.7.30/
slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Security framework of XStream not initialized, XStream is probably vulnerable.
Security framework of XStream not initialized, XStream is probably vulnerable.
Security framework of XStream not initialized, XStream is probably vulnerable.
Security framework of XStream not initialized, XStream is probably vulnerable.
1609 [main] INFO c.h.s.c.s.p.ApplicationParser - start to parse cql : CREATE INPUT STREAM s (id INT,
name STRING, type INT) SOURCE randomgen PROPERTIES ("timeUnit" = "SECONDS",
"period" = "1", "eventNumPerPeriod" = "1", "isSchedule" = "true")
```



```

1797 [main] INFO c.h.s.c.s.p.ApplicationParser - Parse Completed
1828 [main] INFO c.h.s.c.t.LazyTask - start to execute CREATE INPUT STREAM s (id INT, name
STRING, type INT) SOURCE randomgen PROPERTIES ( 'timeUnit' = 'SECONDS', 'period' = '1',
'eventNumPerPeriod' = '1', 'isSchedule' = 'true' )
1828 [main] INFO c.h.s.c.s.p.ApplicationParser - start to parse cql : CREATE OUTPUT STREAM rs(type
INT, cc INT) SINK ConsoleOutput
1828 [main] INFO c.h.s.c.s.p.ApplicationParser - Parse Completed
1836 [main] INFO c.h.s.c.t.LazyTask - start to execute CREATE OUTPUT STREAM rs (type INT, cc INT)
SINK ConsoleOutput
1836 [main] INFO c.h.s.c.s.p.ApplicationParser - start to parse cql : INSERT INTO STREAM rs SELECT
type, count(id) as cc FROM s[RANGE 20 SECONDS BATCH] WHERE id > 5 GROUP BY type
1844 [main] INFO c.h.s.c.s.p.ApplicationParser - Parse Completed
Security framework of XStream not initialized, XStream is probably vulnerable.
Security framework of XStream not initialized, XStream is probably vulnerable.
1875 [main] INFO c.h.s.c.t.LazyTask - start to execute INSERT INTO STREAM rs SELECT type,
count(id) AS cc FROM s[RANGE 20 SECONDS BATCH ] WHERE id > 5 GROUP BY type
1875 [main] INFO c.h.s.c.s.p.ApplicationParser - start to parse cql : SUBMIT APPLICATION example
1875 [main] INFO c.h.s.c.s.p.ApplicationParser - Parse Completed
...
Process finished with exit code 0

```

----End

7.4.2 Debugging and Running an Application in Linux

After the preparation in [Linux Environment](#) is complete, enter the following CQL statements in the CQL command line interface of the Linux client:

```

>CREATE INPUT STREAM s
> (id INT, name STRING, type INT)
> SOURCE randomgen
> PROPERTIES
>   ( "timeUnit" = "SECONDS", "period" = "1",
>     "eventNumPerPeriod" = "1", "isSchedule" = "true");
Streaming>
>CREATE OUTPUT STREAM rs
>(type INT, cc INT)
> SINK ConsoleOutput;
Streaming>
>INSERT INTO STREAM rs SELECT type, count(id) as cc FROM s[RANGE 20 SECONDS BATCH] WHERE id >
5 GROUP BY type;
Streaming>
>SUBMIT APPLICATION example;
Streaming>show applications;

```

[Figure 7-12](#) shows the running results.

Figure 7-12 Running result

```

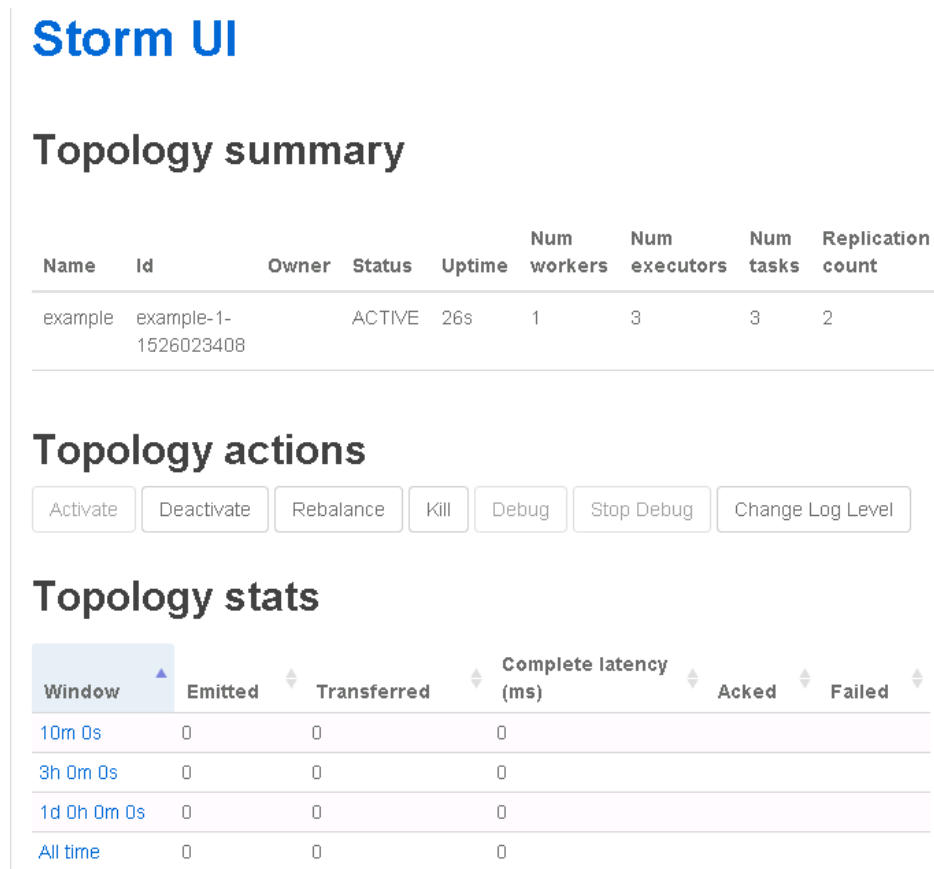
Streaming>show applications;
applicationName      Status      Num_workers  Uptime_secs
example              ACTIVE      1             7

```

7.4.3 Viewing Commissioning Results

- Step 1** Log in to FusionInsight Manager.
- Step 2** Log in to FusionInsight Manager, choose **Cluster** > *Name of the desired cluster* > **Service** > **Storm**, and click the link of Storm UI.
- Step 3** Click the **example** application and view application running status. See [Figure 7-13](#). In **Topology stats**, the total volume of data transferred between operators in different periods of time is displayed.

Figure 7-13 Storm application page



Spouts (All time) displays the total number of messages sent by the **Spout** operator since its startup. **Bolts (All time)** displays the total number of messages sent by the **Count** and **Split** operators, as shown in [Figure 7-14](#).

Figure 7-14 Total volume of data sent by the Storm application operators

Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked
RandomGen_001	1	1	100	100	0.000	0

Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execu latenc (ms)
Aggregator_004	1	1	180	180	0.000	0.000
ConsoleOutput_002	1	1	0	0	0.000	0.200

----End

7.5 FAQ

7.5.1 "Application 'example' already exist" Is Reported When a Sample Application Is Run in Windows

Symptom

When a sample application is run in Windows, an error message similar to the following is displayed in the run log:

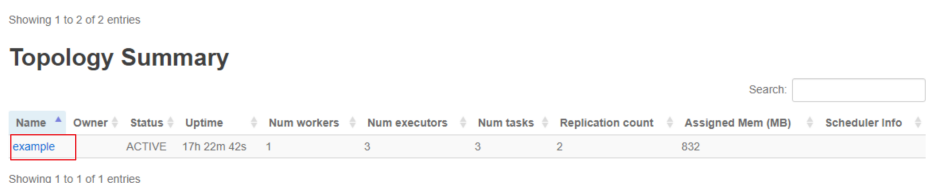
```
Exception in thread "main" com.huawei.streaming.cql.exception.ExecutorException: ERROR  
CQL-03001: Application 'example' already exist.  
    at  
com.huawei.streaming.cql.exception.ExecutorException.wrapStreamingException(ExecutorException.java:87)  
    at com.huawei.streaming.cql.executor.PhysicalPlanExecutor.execute(PhysicalPlanExecutor.java:142)  
    at com.huawei.streaming.cql.tasks.SubmitTask.submitApplication(SubmitTask.java:195)  
    at com.huawei.streaming.cql.tasks.SubmitTask.execute(SubmitTask.java:104)  
    at com.huawei.streaming.cql.Driver.executeTask(Driver.java:217)  
    at com.huawei.streaming.cql.Driver.run(Driver.java:144)  
    at com.huawei.streaming.example.cql.CQLExample.executeCQL(CQLExample.java:55)  
    at com.huawei.streaming.example.cql.CQLExample.execute(CQLExample.java:41)  
    at com.huawei.streaming.example.cql.CQLExample.main(CQLExample.java:21)
```

Possible Causes

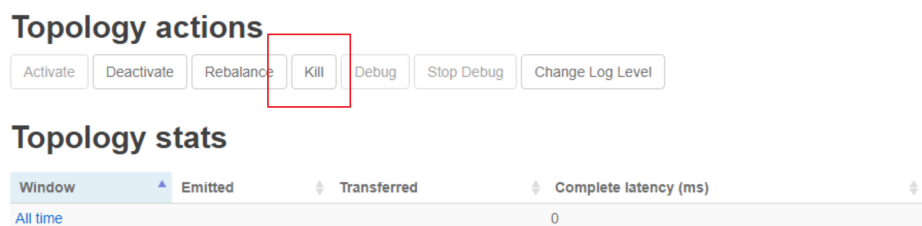
The **example** task has been submitted. If you run the sample application again, the task will be repeatedly submitted. To prevent repeated task submission, this error is reported.

Procedure

- Step 1** Log in to FusionInsight Manager.
- Step 2** Log in to FusionInsight Manager, choose **Cluster** > *Name of the desired cluster* > **Service** > **Storm**, and click the link of Storm UI.
- Step 3** On Storm UI, click the example application.



- Step 4** Click **Kill** under **Topology actions**, click **OK** as prompted, and wait until the application stops running.



Step 5 After the **example** task is complete, wait for about 5 minutes and run the sample application in Windows again.

----End

8 ClickHouse Development Guide (Security Mode)

8.1 Overview

8.1.1 Introduction to ClickHouse

ClickHouse

ClickHouse is a column-based database oriented to online analysis and processing. It supports SQL query and provides good query performance. The aggregation analysis and query performance based on large and wide tables is excellent, which is one order of magnitude faster than other analytical databases.

Advantages for ClickHouse:

- High data compression ratio
- Multi-core parallel computing
- Vectorized computing engine
- Supporting nested data structure
- Supporting sparse indexes
- Supporting INSERT and UPDATE

ClickHouse application scenarios:

- Real-time data warehouse
The streaming computing engine (such as Flink) is used to write real-time data to ClickHouse. With the excellent query performance of ClickHouse, Multi-dimensional and multi-mode real-time query and analysis requests can be responded within subseconds.
- Offline query
Large-scale service data is imported to ClickHouse and constructs a large wide table with hundreds of millions to tens of billions of records and hundreds of dimensions. It supports personalized statistics collection and continuous

exploratory query and analysis at any time to assist business decision-making and provides excellent query experience.

Introduction to the ClickHouse Development Interface

ClickHouse is developed using C++ and positioned as a DBMS. It supports HTTP and Native TCP network interface protocols and multiple driver modes such as JDBC and ODBC. You are advised to use [clickhouse-jdbc](#) of the community version for application development.

8.1.2 Basic Concepts

Concepts

- **cluster**
Cluster: Cluster is a logical concept in ClickHouse. It can be defined by users as required, which is different from the general understanding of cluster. Multiple ClickHouse nodes are loosely coupled and exist independently.
- **shards**
Shard: A shard is a horizontal division of a cluster. A cluster can consist of multiple shards.
- **replicas**
Replica: One shard can contain multiple replicas.
- **partition**
Partition: A partition is used for local replicas and can be considered as a vertical division.
- **MergeTree**
ClickHouse has a huge table engine system. As the basic table engine of the family system, MergeTree provides functions such as data partitioning, primary indexes, and secondary indexes. When creating a table, you need to specify the table engine. Different table engines determine the final character of a data table, for example, the features of a data table, the form how data is stored, and how data is loaded.

8.1.3 Development Process

[Figure 8-1](#) and [Table 8-1](#) describe the phases in the development process.

Figure 8-1 ClickHouse application development process

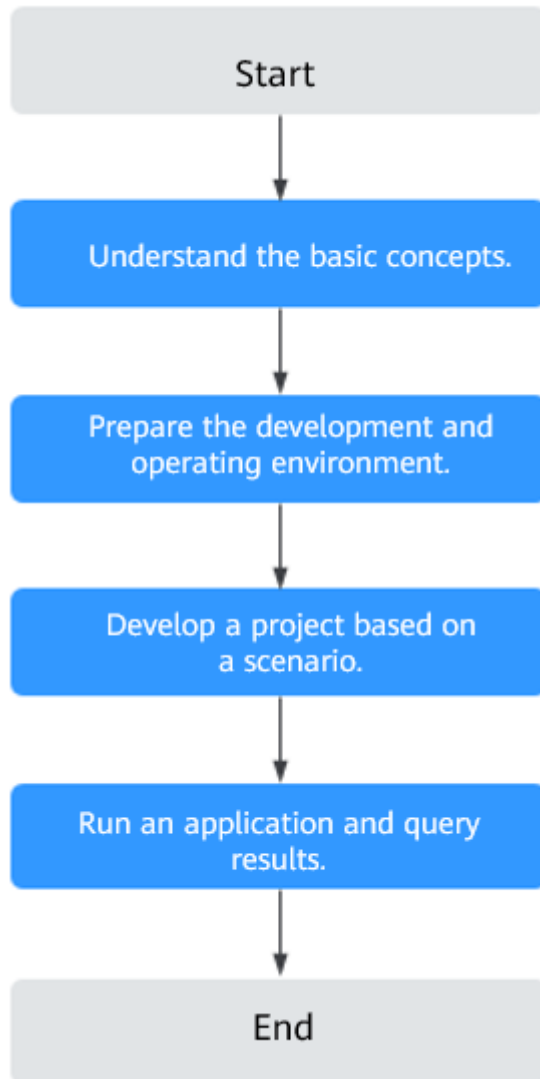


Table 8-1 Description of the ClickHouse application development process

Phase	Description	Reference
Understand the basic concepts.	Before application development, you need to understand basic concepts of ClickHouse.	Concepts
Prepare the development and operating environment.	ClickHouse applications can be developed in multiple languages, mainly Java. The IntelliJ IDEA tool is recommended. Configure the development environment based on the guide.	Preparing the Development and Operating Environment

Phase	Description	Reference
Develop a project based on a scenario.	Sample projects are provided to help you quickly understand APIs of ClickHouse components.	Configuring and Importing a Sample Project
Run application and query results.	You can check the application running status from the running results.	Commissioning Applications on Windows Commissioning Applications on Linux

8.2 Environment Preparations

8.2.1 Preparing the Development and Operating Environment

Preparing the Development Environment

[Table 8-2](#) describes the environment required for application development.

Table 8-2 Development environment

Item	Description
Operating System (OS)	<ul style="list-style-type: none"> Development environment: Windows 7 or later. Operating environment: Linux <p>If the program needs to be commissioned locally, the operating environment must be able to communicate with network on the cluster service plane.</p>

Item	Description
JDK installation	<p>Basic configurations of the development and operating environments. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK 1.8.0_272.</p> <p>Customers' applications that need to reference the JAR files of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> • x86 client: Oracle JDK 1.8; IBM JDK 1.8.5.11 • TaiShan client: OpenJDK 1.8.0_272 <p>NOTE</p> <p>For security purposes, the server supports only TLS V1.2 or later.</p> <p>By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter com.ibm.jsse2.overrideDefaultTLS to true. After the setting, the IBM JDK supports TLS V1.0, TLS V1.1, and TLS V1.2. For details, see https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls.</p>
Installing and configuring IntelliJ IDEA	<p>Basic configuration of the development environment. You are advised to use version 2019.1 or other compatible versions.</p> <p>NOTE</p> <ul style="list-style-type: none"> • If you use IBM JDK, ensure that the JDK configured in IntelliJ IDEA is IBM JDK. • If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK. • If you use Open JDK, ensure that the JDK configured in IntelliJ IDEA is Open JDK. • Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.
Apache Maven installation	<p>Basic configuration of the development environment. This operation is used for project management throughout the lifecycle of software development.</p>
Developer account preparation	<p>Prepare a cluster user for application development and grant permissions to the user. For details, see Preparing a Developer Account.</p>
7-zip	<p>This tool is used to decompress *.zip and *.rar files. 7-Zip 16.04 is supported.</p>

Preparing the Operating Environment

During application development, you need to prepare the environment for code running and commissioning to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning application; configure the network connection, and commission the application in Windows.
 - a. **Logging In to FusionInsight Manager** and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Configuration Files Only**. Select the platform type based on the type of the node where the client is to be installed (select **x86_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is

FusionInsight_Cluster_1_Services_Client.tar, decompress it to obtain **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar**. Then, decompress **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar** to the **D:\FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles** directory on the local PC. The directory name cannot contain spaces.

- b. Copy all items from the hosts file in the decompression directory to the hosts file on the node where the client is installed. Ensure that the network communication between the local PC and hosts listed in the hosts file in the decompression directory is normal.

NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
 - a. Install the client in a directory, for example, **/opt/client**, on the node. Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use a client on the master or core nodes inside a cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to use a client outside a cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).
 - b. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides. Ensure

that the local host can communicate with each host in the cluster on the network.

8.2.2 Configuring and Importing a Sample Project

Context

Obtain the ClickHouse development sample project and import the project to IntelliJ IDEA to learn the sample project.

Prerequisites

Ensure that the difference between the local PC time and the cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the time of the cluster in the lower-right corner on the FusionInsight Manager page.

Scenarios

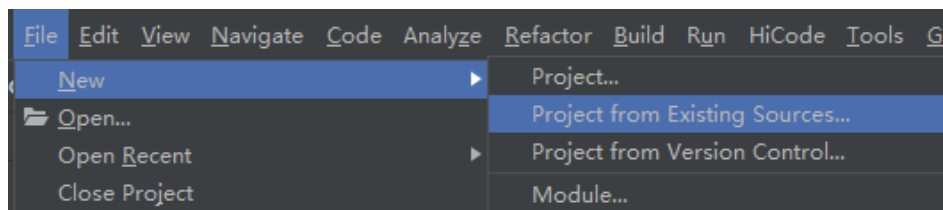
ClickHouse provides sample projects for multiple scenarios to help you quickly learn ClickHouse projects.

Procedure

Step 1 Obtain the sample project folder **clickhouse-examples** and Maven configurations in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

Step 2 In the application development environment, import the sample project to the IntelliJ IDEA development environment.

1. On the IDEA page, choose **File > New > Project from Existing Sources**.



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **clickhouse-examples** folder and click **OK**.
3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.
4. Select the recommended JDK version and click **Finish**.

Step 3 After the project is imported, modify the **clickhouse-example.properties** file in the **conf** directory of the sample project based on the actual environment information.

```
loadBalancerIPList=  
sslUsed=false  
loadBalancerHttpPort=21425  
loadBalancerHttpsPort=21426  
CLICKHOUSE_SECURITY_ENABLED=true  
user=  
password=
```

```
clusterName=default_cluster
databaseName=testdb
tableName=testtb
batchRows=10000
batchNum=10
```

Table 8-3 Configuration description

Parameter	Default Value	Definition
loadBalancerIPList	-	Mandatory. Set this parameter to the IP address list of LoadBalance. Log in to FusionInsight Manager and choose Cluster > Services > ClickHouse . Click Instance and check the service IP addresses of all ClickHouseBalancer instances. Use commas (,) to separate multiple IP addresses, for example, 10.10.10.100,10.10.10.101 .
sslUsed	false	Whether to enable SSL encryption. You are advised to set this parameter to true for clusters in security mode.
loadBalancerHttpPort	21425	HTTP port number of LoadBalance. If sslUsed is set to false , this parameter cannot be left blank. Log in to FusionInsight Manager and choose Cluster > Services > ClickHouse . Click Instance and click the corresponding ClickHouseBalancer instance. Click Instance Configurations then All Configurations , search for lb_http_port , and obtain the parameter value. The default value is 21425 .
loadBalancerHttpsPort	21426	HTTPS port number of LoadBalance. If sslUsed is set to true , this parameter cannot be left blank. Log in to FusionInsight Manager and choose Cluster > Services > ClickHouse . Click Instance , click the corresponding ClickHouseBalancer instance, choose Instance Configurations > All Configurations , search for lb_https_port , and obtain the parameter value. The default value is 21426 .
CLICKHOUSE_SECURITY_ENABLED	true	Whether to enable the security mode for ClickHouse. This parameter has a fixed value of true for a cluster in security mode.
user	N/A	Developer user name prepared in Table 8-2 .

Parameter	Default Value	Definition
password	N/A	Password of the development user NOTE If the user is created on FusionInsight Manager, you need to change the initial password upon the first login.
clusterName	default_cluster	ClickHouse logical cluster name. Retain the default value.
databaseName	testdb	Name of the database to be created in the sample code project. You can change the database name based on the site requirements.
tableName	testtb	Name of the table to be created in the sample code project. You can change the table name based on the site requirements.
batchRows	10000	Number of data records written in a batch.
batchNum	10	Total number of batches in which data is written.

 **NOTE**

ClickHouse uses the LoadBalance-based deployment architecture to automatically distribute user access traffic to multiple backend nodes, expanding service capabilities to external systems and improving fault tolerance. When a client application requests a cluster, Nginx-based ClickHouseBalancer is used to distribute traffic. In this way, data read/write load and high availability of application access are guaranteed.

----End

8.3 Application Development

8.3.1 Typical Application Scenario

You can quickly learn and master the ClickHouse development process and know key interface functions in a typical application scenario.

Scenario

ClickHouse enables you to perform common service operations by executing SQL statements. The SQL operations involved in sample codes include creating a database, creating a table, inserting data into a table, querying table data, and deleting a table.

Sample codes are described in the following sequence:

1. Setting properties

2. Establishing a connection
3. Creating a database
4. Creating a table
5. Inserting data into a table
6. Querying data
7. Deleting a table

8.3.2 Development Guideline

Development Guideline

As an independent DBMS system, ClickHouse allows you to use the SQL language to perform common operations. In the development program example, the `clickhouse-jdbc` API is used for description. The development process consists of the following parts:

- **Setting properties:** Sets the parameters for connecting to a ClickHouse service instance.
- **Establishing a connection:** Establishes a connection to the ClickHouse service instance.
- **Creating a database:** Creates a ClickHouse database.
- **Creating a table:** Creates a table in the ClickHouse database.
- **Inserting data:** Inserts data into a ClickHouse table.
- **Querying data:** Queries data in ClickHouse tables.
- **Deleting a table:** Deletes a ClickHouse table.

8.4 Sample Code

8.4.1 Setting Properties

Set the connection properties. In the following sample code, the socket timeout interval is set to 60s.

```
ClickHouseProperties clickHouseProperties = new ClickHouseProperties();  
clickHouseProperties.setSocketTimeout(60000);
```

If `sslUsed` in the `clickhouse-example.properties` file in [Configuring and Importing a Sample Project](#) is set to `true`, set the following connection properties:

```
clickHouseProperties.setSsl(true);  
clickHouseProperties.setSslMode("none");
```

8.4.2 Establishing a Connection

During connection creation, the user and password configured in [Table 8-3](#) are used as authentication credentials. ClickHouse performs security authentication on the server with the user and password.

```
ClickHouseDataSource clickHouseDataSource = new ClickHouseDataSource(JDBC_PREFIX +  
serverList.get(tries - 1), clickHouseProperties);  
connection = clickHouseDataSource.getConnection(user, password);
```

8.4.3 Creating a Database

Run the **on cluster** statement to create a database whose name is the value of **databaseName** in [Table 8-3](#) in the cluster.

```
private void createDatabase(String databaseName, String clusterName) throws Exception {
    String createDbSql = "create database if not exists " + databaseName + " on cluster " + clusterName;
    util.exeSql(createDbSql);
}
```

8.4.4 Creating a Table

Run the **on cluster** statement to create the ReplicatedMerge and Distributed tables whose names are the same as the values of **tableName** in [Table 8-3](#).

```
private void createTable(String databaseName, String tableName, String clusterName) throws Exception {
    String createSql = "create table " + databaseName + "." + tableName + " on cluster " + clusterName +
        " (name String, age UInt8, date Date)engine=ReplicatedMergeTree('/clickhouse/tables/{shard}/" +
        databaseName + "." + tableName + "," + "{replica}") partition by toYYYYMM(date) order by age";
    String createDisSql = "create table " + databaseName + "." + tableName + "_all" + " on cluster " +
        clusterName + " as " + databaseName + "." + tableName + " ENGINE = Distributed(default_cluster," +
        databaseName + "." + tableName + ", rand());";
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(createSql);
    sqlList.add(createDisSql);
    util.exeSql(sqlList);
}
```

8.4.5 Inserting Data

The table created in [Creating a Table](#) has three fields of the String, UInt8, and Date types.

```
String insertSql = "insert into " + databaseName + "." + tableName + " values (?, ?, ?)";
PreparedStatement preparedStatement = connection.prepareStatement(insertSql);
long allBatchBegin = System.currentTimeMillis();
for (int j = 0; j < batchNum; j++) {
    for (int i = 0; i < batchRows; i++) {
        preparedStatement.setString(1, "huawei_" + (i + j * 10));
        preparedStatement.setInt(2, ((int) (Math.random() * 100)));
        preparedStatement.setDate(3, generateRandomDate("2018-01-01", "2021-12-31"));
        preparedStatement.addBatch();
    }
    long begin = System.currentTimeMillis();
    preparedStatement.executeBatch();
    long end = System.currentTimeMillis();
    log.info("Inert batch time is {} ms", end - begin);
}
long allBatchEnd = System.currentTimeMillis();
log.info("Inert all batch time is {} ms", allBatchEnd - allBatchBegin);
```

8.4.6 Querying Data

Query statement 1 **querySql1** queries any 10 records in the **tableName** table created in [Creating a Table](#). Query statement 2 **querySql2** uses a built-in function to combine the year and month fields in the **tableName** table created in [Creating a Table](#).

```
private void queryData(String databaseName, String tableName) throws Exception {
    String querySql1 = "select * from " + databaseName + "." + tableName + "_all" + " order by age limit 10";
    String querySql2 = "select toYYYYMM(date),count(1) from " + databaseName + "." + tableName + "_all" +
        " group by toYYYYMM(date) order by count(1) DESC limit 10";
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(querySql1);
    sqlList.add(querySql2);
}
```

```

ArrayList<ArrayList<ArrayList<String>>> result = util.exeSql(sqlList);
for (ArrayList<ArrayList<String>> singleResult : result) {
    for (ArrayList<String> strings : singleResult) {
        StringBuilder stringBuilder = new StringBuilder();
        for (String string : strings) {
            stringBuilder.append(string).append("\t");
        }
        log.info(stringBuilder.toString());
    }
}
}

```

8.4.7 Deleting a Table

Delete the replica table and distributed table created in [Creating a Table](#).

```

private void dropTable(String databaseName, String tableName, String clusterName) throws Exception {
    String dropLocalTableSql = "drop table if exists " + databaseName + "." + tableName + " on cluster " +
        clusterName;
    String dropDisTableSql = "drop table if exists " + databaseName + "." + tableName + "_all" + " on cluster
" + clusterName;
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(dropLocalTableSql);
    sqlList.add(dropDisTableSql);
    util.exeSql(sqlList);
}

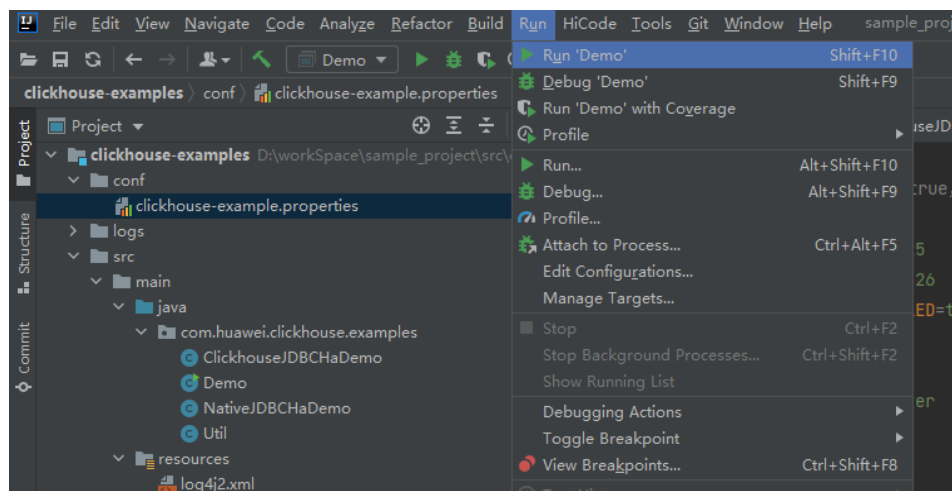
```

8.5 Application Commissioning

8.5.1 Commissioning Applications on Windows

Compile and run applications.

You can run applications in the Windows environment after application code development is complete. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host. In the IntelliJ IDEA project **clickhouse-examples** of the development environment, click **Run'Demo'** to run the application project.



Viewing Commissioning Results

After a ClickHouse application is run, you can use one of the following methods to view the running result:

- Viewing the command output
- View the **clickhouse-example.log** file in the **logs** directory to obtain the application running status.

After the complete sample of the **clickhouse-examples** is run, the following information is displayed on the console:

```
Connected to the target VM, address: '127.0.0.1:53321', transport: 'socket'
2021-04-21 21:02:16,900 | INFO | main | loadBalancerIPList is 10.120.147.36, loadBalancerHttpPort is 21425, user is xxx, clusterName is default_cluster, isSec is true, password is xxx. |
com.huawei.clickhouse.examples.Demo.main(Demo.java:40)
2021-04-21 21:02:16,904 | INFO | main | ckLbServerList current member is 0, ClickhouseBalancer is 10.120.147.36:21425 | com.huawei.clickhouse.examples.Demo.getCkLbServerList(Demo.java:96)
2021-04-21 21:02:16,914 | INFO | main | Driver registered |
ru.yandex.clickhouse.ClickHouseDriver.<clinit>(ClickHouseDriver.java:49)
2021-04-21 21:02:16,918 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:17,517 | INFO | main | Execute query:drop table if exists testdb.testtb on cluster default_cluster | com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:17,656 | INFO | main | Execute time is 139 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:17,657 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:17,701 | INFO | main | Execute query:drop table if exists testdb.testtb_all on cluster default_cluster | com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:17,851 | INFO | main | Execute time is 148 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:17,851 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:17,895 | INFO | main | Execute query:create database if not exists testdb on cluster default_cluster | com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:18,043 | INFO | main | Execute time is 148 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:18,044 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:18,088 | INFO | main | Execute query:create table testdb.testtb on cluster default_cluster (name String, age UInt8, date Date)engine=ReplicatedMergeTree('/clickhouse/tables/{shard}/testdb.testtb',{replica}') partition by toYYYYMM(date) order by age |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:18,233 | INFO | main | Execute time is 144 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:18,233 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:18,278 | INFO | main | Execute query:create table testdb.testtb_all on cluster default_cluster as testdb.testtb ENGINE = Distributed(default_cluster,testdb,testtb, rand()); |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:18,422 | INFO | main | Execute time is 144 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:18,423 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.insertData(Util.java:128)
2021-04-21 21:02:19,380 | INFO | main | Inert batch time is 720 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:19,927 | INFO | main | Inert batch time is 492 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:20,456 | INFO | main | Inert batch time is 504 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:20,894 | INFO | main | Inert batch time is 410 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:21,348 | INFO | main | Inert batch time is 431 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:21,813 | INFO | main | Inert batch time is 442 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:22,273 | INFO | main | Inert batch time is 434 ms |
```

```
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:22,730 | INFO | main | Inert batch time is 435 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:23,212 | INFO | main | Inert batch time is 459 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:23,689 | INFO | main | Inert batch time is 452 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:23,689 | INFO | main | Inert all batch time is 5216 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:148)
2021-04-21 21:02:23,689 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:23,732 | INFO | main | Execute query:select * from testdb.testtb_all order by age limit 10 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:23,803 | INFO | main | Execute time is 71 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:23,804 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:23,848 | INFO | main | Execute query:select toYYYYMM(date),count(1) from
testdb.testtb_all group by toYYYYMM(date) order by count(1) DESC limit 10 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:23,895 | INFO | main | Execute time is 47 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:23,896 | INFO | main | name age date |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_4077 0 2021-12-21 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_183 0 2021-12-10 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_5407 0 2021-12-13 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_1072 0 2021-12-03 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_4667 0 2021-12-22 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_1767 0 2021-12-03 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_8001 0 2021-12-22 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_1822 0 2021-12-04 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_5095 0 2021-12-23 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_7133 0 2021-12-26 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | toYYYYMM(date) count() |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 202101 2184 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 202105 2176 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 201810 2173 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 201907 2162 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 201803 2159 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 201805 2153 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 202110 2145 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 201801 2144 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 201908 2143 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 202005 2133 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
Disconnected from the target VM, address: '127.0.0.1:53321', transport: 'socket'
Process finished with exit code 0
```

8.5.2 Commissioning Applications on Linux

ClickHouse applications can run in a Linux environment. After the application code is developed, you can upload the JAR file to the prepared Linux environment.

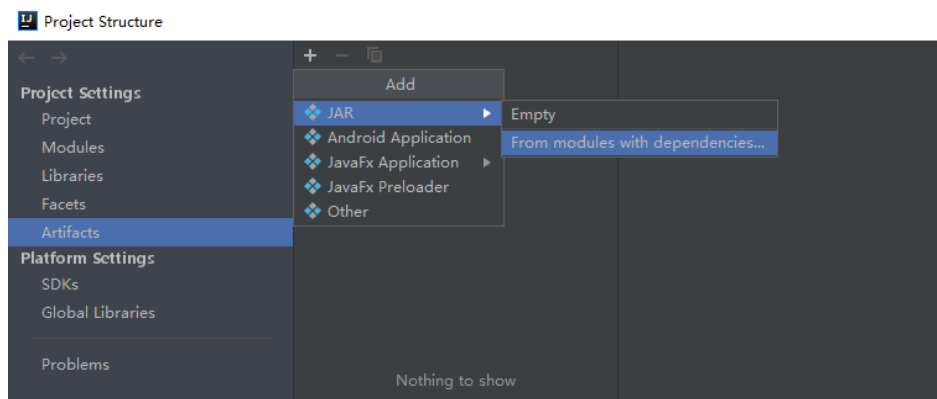
Prerequisites

JDK has been installed on Linux. The version must be the same as JDK version of the JAR file exported from IntelliJ IDEA. Java environment variables have been set.

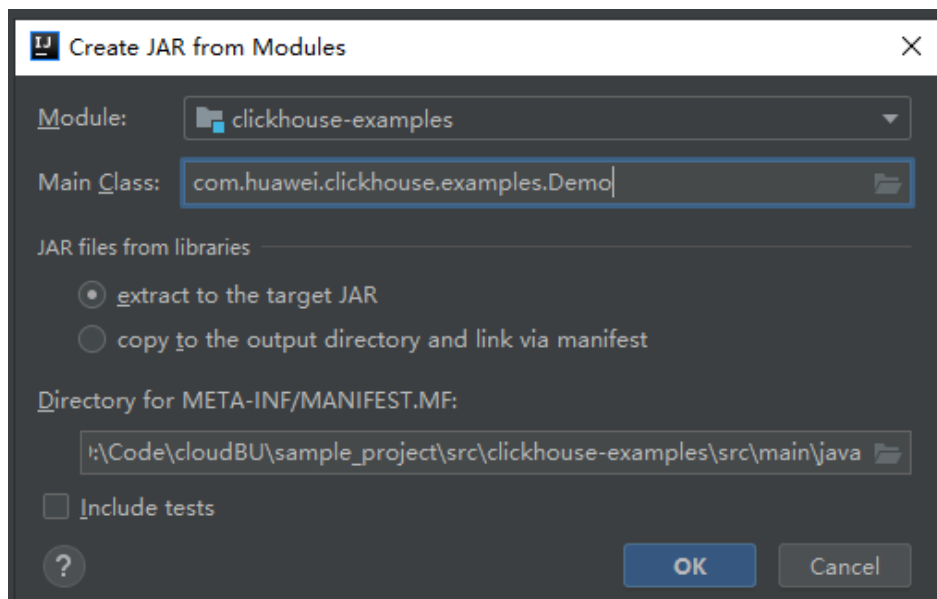
Compile and run applications.

Step 1 Export the JAR file.

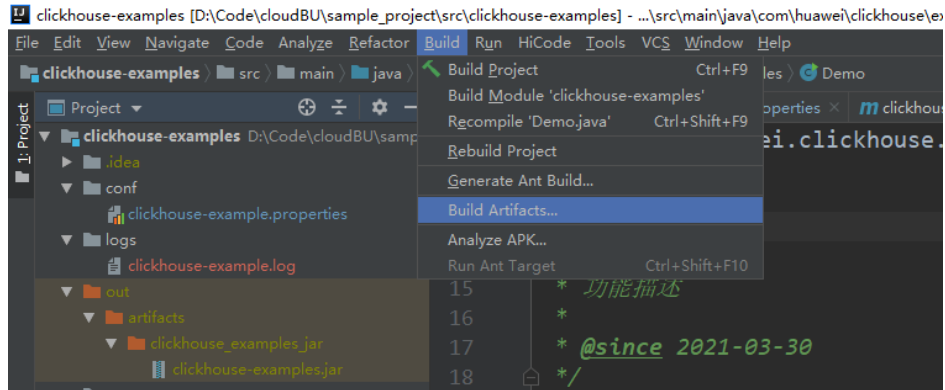
1. Log in to IntelliJ IDEA and choose **File > Project Structure > Artifacts**.
2. Click the plus sign (+) and choose **JAR > From modules with dependencies**.



3. Choose **com.huawei.clickhouse.examples.Demo** from the **Main Class** drop-down list and click **OK**.



4. Choose **Build > Build Artifacts**. After the compilation is successful, view and obtain the **clickhouse_examples.jar** file in the **clickhouse-examples\out\artifacts\clickhouse_examples.jar** directory.



Step 2 Copy the **clickhouse_examples.jar** file and the **conf** folder in the **clickhouse-examples** directory to the ClickHouse client installation directory, for example, **/opt/client**.

Step 3 Log in to the client node, go to the directory where the JAR file is uploaded, and change the file permission to 700.

```
cd /opt/client
```

```
chmod 700 clickhouse-examples.jar
```

Step 4 In the client directory where **clickhouse_examples.jar** is stored, run the following commands to run the JAR file:

```
source bigdata_env
```

```
java -cp ./*:conf/clickhouse-example.properties  
com.huawei.clickhouse.examples.Demo
```

```
----End
```

Viewing Commissioning Results

After a ClickHouse application is run, you can use one of the following methods to view the running result:

- Viewing the command output
- Viewing ClickHouse logs

That is, view the **logs/clickhouse-example.log** file in the directory where the current JAR file is located, for example, **/opt/client/logs/clickhouse-example.log**.

The execution result of the JAR file is as follows:

```
2021-06-10 20:53:56,028 | INFO | main | Current load balancer is 10.112.17.150:21426 |  
com.huawei.clickhouse.examples.Util.insertData(Util.java:128)  
2021-06-10 20:53:58,247 | INFO | main | Inert batch time is 1442 ms |  
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)  
2021-06-10 20:53:59,649 | INFO | main | Inert batch time is 1313 ms |  
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)  
2021-06-10 20:54:05,872 | INFO | main | Inert batch time is 6132 ms |  
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)  
2021-06-10 20:54:10,223 | INFO | main | Inert batch time is 4272 ms |  
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)  
2021-06-10 20:54:11,614 | INFO | main | Inert batch time is 1300 ms |  
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)  
2021-06-10 20:54:12,871 | INFO | main | Inert batch time is 1200 ms |
```

```
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:14,589 | INFO | main | Inert batch time is 1663 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:16,141 | INFO | main | Inert batch time is 1500 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:17,690 | INFO | main | Inert batch time is 1498 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:19,206 | INFO | main | Inert batch time is 1468 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:19,207 | INFO | main | Inert all batch time is 22626 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:148)
2021-06-10 20:54:19,208 | INFO | main | Current load balancer is 10.112.17.150:21426 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-06-10 20:54:20,231 | INFO | main | Execute query:select * from mutong1.testtb_all order by age limit
10 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-06-10 20:54:21,266 | INFO | main | Execute time is 1035 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-06-10 20:54:21,267 | INFO | main | Current load balancer is 10.112.17.150:21426 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-06-10 20:54:21,815 | INFO | main | Execute query:select toYYYYMM(date),count(1) from
mutong1.testtb_all group by toYYYYMM(date) order by count(1) DESC limit 10 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-06-10 20:54:22,897 | INFO | main | Execute time is 1082 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-06-10 20:54:22,898 | INFO | main | name age date |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,898 | INFO | main | huawei_266 0 2021-12-19 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_2500 0 2021-12-29 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_8980 0 2021-12-16 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_671 0 2021-12-29 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_2225 0 2021-12-12 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_6040 0 2021-12-14 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_7294 0 2021-12-10 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_1133 0 2021-12-25 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | huawei_3161 0 2021-12-21 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | huawei_3992 0 2021-11-25 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | toYYYYMM(date) count() |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | 201910 2247 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | 202105 2213 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | 201801 2208 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | 201803 2204 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201810 2167 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201805 2166 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201901 2164 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201908 2145 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201912 2143 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 202107 2137 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
```

9 ClickHouse Development Guide (Normal Mode)

9.1 Overview

9.1.1 Introduction to ClickHouse

ClickHouse

ClickHouse is a column-based database oriented to online analysis and processing. It supports SQL query and provides good query performance. The aggregation analysis and query performance based on large and wide tables is excellent, which is one order of magnitude faster than other analytical databases.

Advantages for ClickHouse:

- High data compression ratio
- Multi-core parallel computing
- Vectorized computing engine
- Supporting nested data structure
- Supporting sparse indexes
- Supporting INSERT and UPDATE

ClickHouse application scenarios:

- Real-time data warehouse
The streaming computing engine (such as Flink) is used to write real-time data to ClickHouse. With the excellent query performance of ClickHouse, Multi-dimensional and multi-mode real-time query and analysis requests can be responded within subseconds.
- Offline query
Large-scale service data is imported to ClickHouse and constructs a large wide table with hundreds of millions to tens of billions of records and hundreds of dimensions. It supports personalized statistics collection and continuous

exploratory query and analysis at any time to assist business decision-making and provides excellent query experience.

Introduction to the ClickHouse Development Interface

ClickHouse is developed using C++ and positioned as a DBMS. It supports HTTP and Native TCP network interface protocols and multiple driver modes such as JDBC and ODBC. You are advised to use [clickhouse-jdbc](#) of the community version for application development.

9.1.2 Basic Concepts

Concepts

- **cluster**
Cluster: Cluster is a logical concept in ClickHouse. It can be defined by users as required, which is different from the general understanding of cluster. Multiple ClickHouse nodes are loosely coupled and exist independently.
- **shards**
Shard: A shard is a horizontal division of a cluster. A cluster can consist of multiple shards.
- **replicas**
Replica: One shard can contain multiple replicas.
- **partition**
Partition: A partition is used for local replicas and can be considered as a vertical division.
- **MergeTree**
ClickHouse has a huge table engine system. As the basic table engine of the family system, MergeTree provides functions such as data partitioning, primary indexes, and secondary indexes. When creating a table, you need to specify the table engine. Different table engines determine the final character of a data table, for example, the features of a data table, the form how data is stored, and how data is loaded.

9.1.3 Development Process

[Figure 9-1](#) and [Table 9-1](#) describe the phases in the development process.

Figure 9-1 ClickHouse application development process

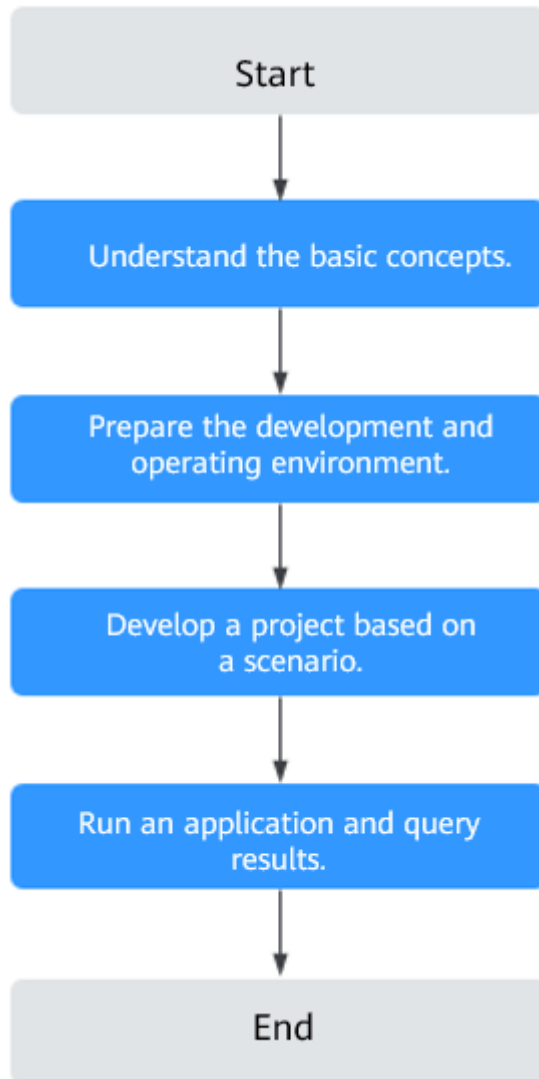


Table 9-1 Description of the ClickHouse application development process

Phase	Description	Reference
Understand the basic concepts.	Before application development, you need to understand basic concepts of ClickHouse.	Basic Concepts
Prepare the development and operating environment.	ClickHouse applications can be developed in multiple languages, mainly Java. The IntelliJ IDEA tool is recommended. Configure the development environment based on the guide.	Preparing the Development and Operating Environment

Phase	Description	Reference
Develop a project based on a scenario.	Sample projects are provided to help you quickly understand APIs of ClickHouse components.	Configuring and Importing a Sample Project
Run application and query results.	You can check the application running status from the running results.	Commissioning Applications on Windows Commissioning Applications on Linux

9.2 Environment Preparations

9.2.1 Preparing the Development and Operating Environment

Preparing the Development Environment

[Table 9-2](#) describes the environment required for application development.

Table 9-2 Development environment

Item	Description
Operating System (OS)	<ul style="list-style-type: none"> Development environment: Windows 7 or later. Operating environment: Linux <p>If the program needs to be commissioned locally, the operating environment must be able to communicate with network on the cluster service plane.</p>

Item	Description
JDK installation	<p>Basic configurations of the development and operating environments. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK 1.8.0_272.</p> <p>Customers' applications that need to reference the JAR files of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> • x86 client: Oracle JDK 1.8; IBM JDK 1.8.5.11 • TaiShan client: OpenJDK 1.8.0_272 <p>NOTE</p> <p>For security purposes, the server supports only TLS V1.2 or later.</p> <p>By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter com.ibm.jsse2.overrideDefaultTLS to true. After the setting, the IBM JDK supports TLS V1.0, TLS V1.1, and TLS V1.2. For details, see https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetts-oracle#matchsslcontext_tls.</p>
Installing and configuring IntelliJ IDEA	<p>Basic configuration of the development environment. You are advised to use version 2019.1 or other compatible versions.</p> <p>NOTE</p> <ul style="list-style-type: none"> • If you use IBM JDK, ensure that the JDK configured in IntelliJ IDEA is IBM JDK. • If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK. • If you use Open JDK, ensure that the JDK configured in IntelliJ IDEA is Open JDK. • Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.
Apache Maven installation	<p>Basic configuration of the development environment. This operation is used for project management throughout the lifecycle of software development.</p>
Developer account preparation	<p>Prepare a cluster user for application development and grant permissions to the user by referring to Preparing a Developer Account.</p>
7-zip	<p>This tool is used to decompress *.zip and *.rar files. 7-Zip 16.04 is supported.</p>

Preparing the Operating Environment

During application development, you need to prepare the environment for code running and commissioning to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning application; configure the network connection, and commission the application in Windows.
 - a. **Logging In to FusionInsight Manager** and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Configuration Files Only**. Select the platform type based on the type of the node where the client is to be installed (select **x86_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is

FusionInsight_Cluster_1_Services_Client.tar, decompress it to obtain **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar**. Then, decompress **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar** to the **D:\FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles** directory on the local PC. The directory name cannot contain spaces.

- b. Copy all items from the hosts file in the decompression directory to the hosts file on the node where the client is installed. Ensure that the network communication between the local PC and hosts listed in the hosts file in the decompression directory is normal.

NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
 - a. Install the client in a directory, for example, **/opt/client**, on the node. Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use a client on the master or core nodes inside a cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to use a client outside a cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).
 - b. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides. Ensure

that the local host can communicate with each host in the cluster on the network.

9.2.2 Configuring and Importing a Sample Project

Context

Obtain the ClickHouse development sample project and import the project to IntelliJ IDEA to learn the sample project.

Prerequisites

Ensure that the difference between the local PC time and the cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the time of the cluster in the lower-right corner on the FusionInsight Manager page.

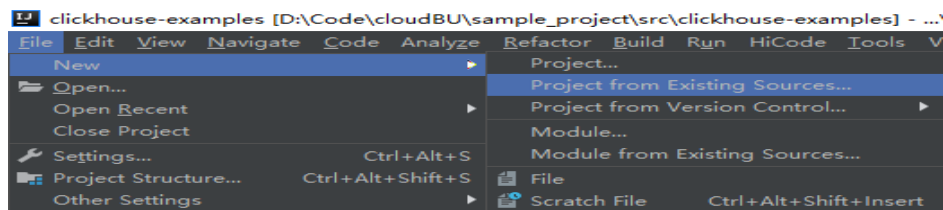
Scenarios

ClickHouse provides sample projects for multiple scenarios to help you quickly learn ClickHouse projects.

Procedure

- Step 1** Obtain the sample project folder **clickhouse-examples** and Maven configurations in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.

1. On the IDEA page, choose **File > New > Project from Existing Sources**.



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **clickhouse-examples** folder and click **OK**.
3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.
4. Select the recommended JDK version and click **Finish**.

- Step 3** After the project is imported, modify the **clickhouse-example.properties** file in the **conf** directory of the sample project based on the actual environment information.

```
loadBalancerIPList=  
sslUsed=false  
loadBalancerHttpPort=21425  
loadBalancerHttpsPort=21426  
CLICKHOUSE_SECURITY_ENABLED=true  
user=  
password=
```

```
clusterName=default_cluster
databaseName=testdb
tableName=testtb
batchRows=10000
batchNum=10
```

Table 9-3 Configuration description

Parameter	Default Value	Definition
loadBalancerIPList	-	Mandatory. Set this parameter to the IP address list of LoadBalance. Log in to FusionInsight Manager and choose Cluster > Services > ClickHouse . Click Instance and check the service IP addresses of all ClickHouseBalancer instances. Use commas (,) to separate multiple IP addresses, for example, 10.10.10.100,10.10.10.101 .
sslUsed	false	Whether to enable SSL encryption. You are advised to set this parameter to false for clusters in common mode.
loadBalancerHttpPort	21425	HTTP port number of LoadBalance. If sslUsed is set to false , this parameter cannot be left blank. Log in to FusionInsight Manager and choose Cluster > Services > ClickHouse . Click Instance , click the corresponding ClickHouseBalancer instance, choose Instance Configurations > All Configurations , search for lb_http_port , and obtain the parameter value. The default value is 21425 .
loadBalancerHttpsPort	21426	HTTPS port number of LoadBalance. If sslUsed is set to true , this parameter cannot be empty. Log in to FusionInsight Manager and choose Cluster > Services > ClickHouse . Click Instance , click the corresponding ClickHouseBalancer instance, choose Instance Configurations > All Configurations , search for lb_https_port , and obtain the parameter value. The default value is 21426 .
CLICKHOUSE_SECURITY_ENABLED	true	Whether to enable the security mode. Set this parameter to false for a cluster in normal mode.
user	No default value	Developer user created in Table 9-2 .

Parameter	Default Value	Definition
password	No default value	Password of the development user NOTE If the user is created on FusionInsight Manager, you need to change the initial password upon the first login.
clusterName	default_cluster	ClickHouse logical cluster name. Retain the default value.
databaseName	testdb	Name of the database to be created in the sample code project. You can change the database name based on the site requirements.
tableName	testtb	Name of the table to be created in the sample code project. You can change the table name based on the site requirements.
batchRows	10000	Number of data records written in a batch.
batchNum	10	Total number of batches in which data is written.

 **NOTE**

ClickHouse uses the LoadBalance-based deployment architecture to automatically distribute user access traffic to multiple backend nodes, expanding service capabilities to external systems and improving fault tolerance. When a client application requests a cluster, Nginx-based ClickHouseBalancer is used to distribute traffic. In this way, data read/write load and high availability of application access are guaranteed.

----End

9.3 Application Development

9.3.1 Typical Application Scenario

You can quickly learn and master the ClickHouse development process and know key interface functions in a typical application scenario.

Scenario

ClickHouse enables you to perform common service operations by executing SQL statements. The SQL operations involved in sample codes include creating a database, creating a table, inserting data into a table, querying table data, and deleting a table.

Sample codes are described in the following sequence:

1. Setting properties
2. Establishing a connection
3. Creating a database
4. Creating a table
5. Inserting data into a table
6. Querying data
7. Deleting a table

9.3.2 Development Guideline

Development Guideline

As an independent DBMS system, ClickHouse allows you to use the SQL language to perform common operations. In the development program example, the `clickhouse-jdbc` API is used for description. The development process consists of the following parts:

- **Setting properties:** Sets the parameters for connecting to a ClickHouse service instance.
- **Establishing a connection:** Establishes a connection to the ClickHouse service instance.
- **Creating a database:** Creates a ClickHouse database.
- **Creating a table:** Creates a table in the ClickHouse database.
- **Inserting data:** Inserts data into a ClickHouse table.
- **Querying data:** Queries data in ClickHouse tables.
- **Deleting a table:** Deletes a ClickHouse table.

9.4 Sample Code

9.4.1 Setting Properties

Set the connection properties. In the following sample code, the socket timeout interval is set to 60s.

```
ClickHouseProperties clickHouseProperties = new ClickHouseProperties();  
clickHouseProperties.setSocketTimeout(60000);
```

If `sslUsed` in the `clickhouse-example.properties` file in [Configuring and Importing a Sample Project](#) is set to `true`, set the following connection properties:

```
clickHouseProperties.setSsl(true);  
clickHouseProperties.setSslMode("none");
```

9.4.2 Establishing a Connection

During connection creation, the user and password configured in [Table 9-3](#) are used as authentication credentials. ClickHouse performs security authentication on the server with the user and password.

```
ClickHouseDataSource clickHouseDataSource =new ClickHouseDataSource(JDBC_PREFIX +
serverList.get(tries - 1), clickHouseProperties);
connection = clickHouseDataSource.getConnection(user, password);
```

9.4.3 Creating a Database

Run the **on cluster** statement to create a database whose name is the value of **databaseName** in [Table 9-3](#) in the cluster.

```
private void createDatabase(String databaseName, String clusterName) throws Exception {
    String createDbSql = "create database if not exists " + databaseName + " on cluster " + clusterName;
    util.exeSql(createDbSql);
}
```

9.4.4 Creating a Table

Run the **on cluster** statement to create the ReplicatedMerge and Distributed tables whose names are the same as the values of **tableName** in [Table 9-3](#).

```
private void createTable(String databaseName, String tableName, String clusterName) throws Exception {
    String createSql = "create table " + databaseName + "." + tableName + " on cluster " + clusterName +
" (name String, age UInt8, date Date)engine=ReplicatedMergeTree('/clickhouse/tables/{shard}"/ +
databaseName + "." + tableName + "," + "{replica}') partition by toYYYYMM(date) order by age";
    String createDisSql = "create table " + databaseName + "." + tableName + "_all" + " on cluster " +
clusterName + " as " + databaseName + "." + tableName + " ENGINE = Distributed(default_cluster," +
databaseName + "," + tableName + ", rand());";
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(createSql);
    sqlList.add(createDisSql);
    util.exeSql(sqlList);
}
```

9.4.5 Inserting Data

The table created in [Creating a Table](#) has three fields of the String, UInt8, and Date types.

```
String insertSql = "insert into " + databaseName + "." + tableName + " values (?,?,?)";
PreparedStatement preparedStatement = connection.prepareStatement(insertSql);
long allBatchBegin = System.currentTimeMillis();
for (int j = 0; j < batchNum; j++) {
    for (int i = 0; i < batchRows; i++) {
        preparedStatement.setString(1, "huawei_" + (i + j * 10));
        preparedStatement.setInt(2, ((int) (Math.random() * 100)));
        preparedStatement.setDate(3, generateRandomDate("2018-01-01", "2021-12-31"));
        preparedStatement.addBatch();
    }
    long begin = System.currentTimeMillis();
    preparedStatement.executeBatch();
    long end = System.currentTimeMillis();
    log.info("Inert batch time is {} ms", end - begin);
}
long allBatchEnd = System.currentTimeMillis();
log.info("Inert all batch time is {} ms", allBatchEnd - allBatchBegin);
```

9.4.6 Querying Data

Query statement 1 **querySql1** queries any 10 records in the **tableName** table created in [Creating a Table](#). Query statement 2 **querySql2** uses a built-in function to combine the year and month fields in the **tableName** table created in [Creating a Table](#).

```
private void queryData(String databaseName, String tableName) throws Exception {
    String querySql1 = "select * from " + databaseName + "." + tableName + "_all" + " order by age limit 10";
    String querySql2 = "select toYYYYMM(date),count(1) from " + databaseName + "." + tableName + "_all"
```



```
+ " group by toYYYYMM(date) order by count(1) DESC limit 10";
ArrayList<String> sqlList = new ArrayList<String>();
sqlList.add(querySql1);
sqlList.add(querySql2);
ArrayList<ArrayList<ArrayList<String>>> result = util.exeSql(sqlList);
for (ArrayList<ArrayList<String>> singleResult : result) {
    for (ArrayList<String> strings : singleResult) {
        StringBuilder stringBuilder = new StringBuilder();
        for (String string : strings) {
            stringBuilder.append(string).append("\t");
        }
        log.info(stringBuilder.toString());
    }
}
}
```

9.4.7 Deleting a Table

Delete the replica table and distributed table created in [Creating a Table](#).

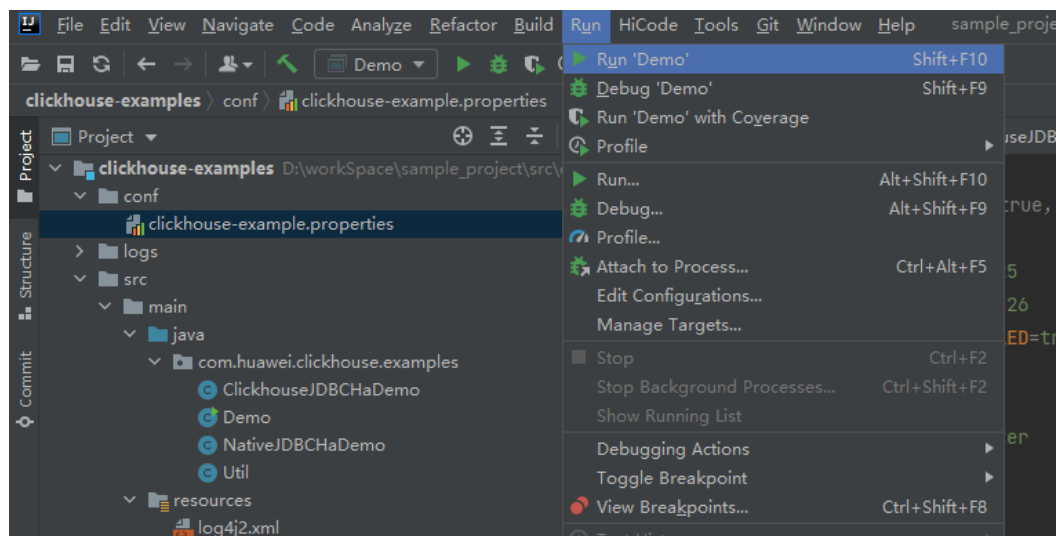
```
private void dropTable(String databaseName, String tableName, String clusterName) throws Exception {
    String dropLocalTableSql = "drop table if exists " + databaseName + "." + tableName + " on cluster " +
clusterName;
    String dropDisTableSql = "drop table if exists " + databaseName + "." + tableName + "_all" + " on cluster
" + clusterName;
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(dropLocalTableSql);
    sqlList.add(dropDisTableSql);
    util.exeSql(sqlList);
}
```

9.5 Application Commissioning

9.5.1 Commissioning Applications on Windows

Compile and run applications.

You can run applications in the Windows environment after application code development is complete. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host. In the IntelliJ IDEA project **clickhouse-examples** of the development environment, click **Run'Demo'** to run the application project.



Viewing Commissioning Results

After a ClickHouse application is run, you can use one of the following methods to view the running result:

- Viewing the command output
- View the **clickhouse-example.log** file in the **logs** directory to obtain the application running status.

After the complete sample of the **clickhouse-examples** is run, the following information is displayed on the console:

```
Connected to the target VM, address: '127.0.0.1:53321', transport: 'socket'
2021-04-21 21:02:16,914 | INFO | main | Driver registered |
ru.yandex.clickhouse.ClickHouseDriver.<clinit>(ClickHouseDriver.java:49)
2021-04-21 21:02:16,918 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:17,517 | INFO | main | Execute query:drop table if exists testdb.testtb on cluster
default_cluster | com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:17,656 | INFO | main | Execute time is 139 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:17,657 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:17,701 | INFO | main | Execute query:drop table if exists testdb.testtb_all on cluster
default_cluster | com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:17,851 | INFO | main | Execute time is 148 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:17,851 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:17,895 | INFO | main | Execute query:create database if not exists testdb on cluster
default_cluster | com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:18,043 | INFO | main | Execute time is 148 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:18,044 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:18,088 | INFO | main | Execute query:create table testdb.testtb on cluster default_cluster
(name String, age UInt8, date Date)engine=ReplicatedMergeTree('/clickhouse/tables/{shard}/
testdb.testtb',{replica}') partition by toYYYYMM(date) order by age |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:18,233 | INFO | main | Execute time is 144 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:18,233 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:18,278 | INFO | main | Execute query:create table testdb.testtb_all on cluster
default_cluster as testdb.testtb ENGINE = Distributed(default_cluster,testdb,testtb, rand()); |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:18,422 | INFO | main | Execute time is 144 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:18,423 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.insertData(Util.java:128)
2021-04-21 21:02:19,380 | INFO | main | Inert batch time is 720 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:19,927 | INFO | main | Inert batch time is 492 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:20,456 | INFO | main | Inert batch time is 504 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:20,894 | INFO | main | Inert batch time is 410 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:21,348 | INFO | main | Inert batch time is 431 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:21,813 | INFO | main | Inert batch time is 442 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:22,273 | INFO | main | Inert batch time is 434 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:22,730 | INFO | main | Inert batch time is 435 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:23,212 | INFO | main | Inert batch time is 459 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
```

```

2021-04-21 21:02:23,689 | INFO | main | Inert batch time is 452 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-04-21 21:02:23,689 | INFO | main | Inert all batch time is 5216 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:148)
2021-04-21 21:02:23,689 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:23,732 | INFO | main | Execute query:select * from testdb.testtb_all order by age limit 10 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:23,803 | INFO | main | Execute time is 71 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:23,804 | INFO | main | Current load balancer is 10.120.147.36:21425 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-04-21 21:02:23,848 | INFO | main | Execute query:select toYYYYMM(date),count(1) from
testdb.testtb_all group by toYYYYMM(date) order by count(1) DESC limit 10 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-04-21 21:02:23,895 | INFO | main | Execute time is 47 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-04-21 21:02:23,896 | INFO | main | name age date |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_4077 0 2021-12-21 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_183 0 2021-12-10 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_5407 0 2021-12-13 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_1072 0 2021-12-03 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_4667 0 2021-12-22 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_1767 0 2021-12-03 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_8001 0 2021-12-22 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_1822 0 2021-12-04 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_5095 0 2021-12-23 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,896 | INFO | main | huawei_7133 0 2021-12-26 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | toYYYYMM(date) count() |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 202101 2184 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 202105 2176 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 201810 2173 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 201907 2162 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 201803 2159 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 201805 2153 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 202110 2145 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 201801 2144 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 201908 2143 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-04-21 21:02:23,897 | INFO | main | 202005 2133 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
Disconnected from the target VM, address: '127.0.0.1:53321', transport: 'socket'
Process finished with exit code 0

```

9.5.2 Commissioning Applications on Linux

ClickHouse applications can run in a Linux environment. After the application code is developed, you can upload the JAR file to the prepared Linux environment.

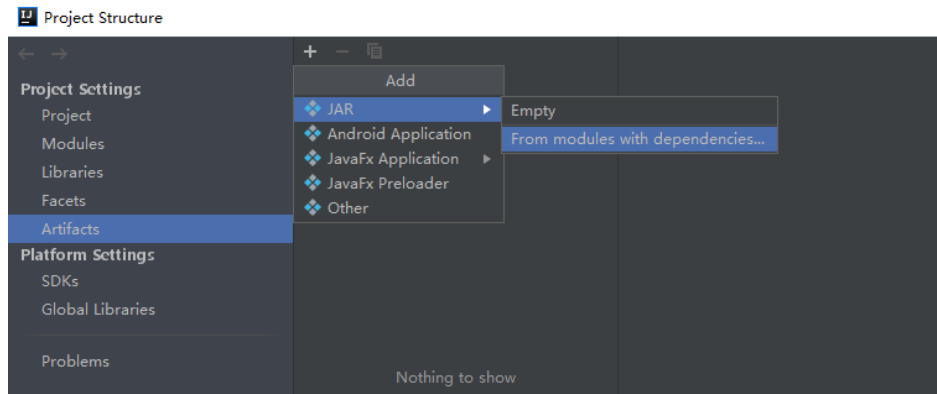
Prerequisites

JDK has been installed on Linux. The version must be the same as JDK version of the JAR file exported from IntelliJ IDEA. Java environment variables have been set.

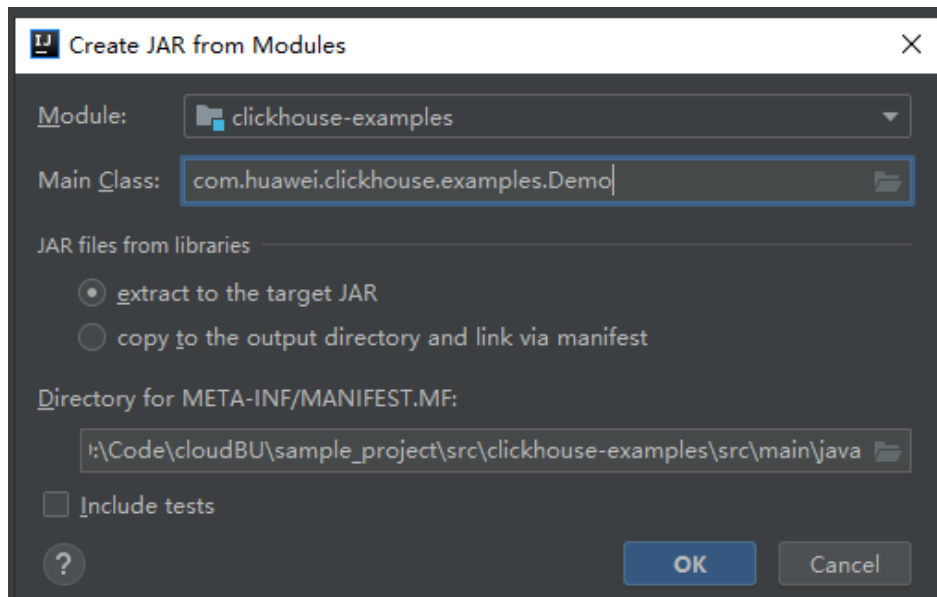
Compile and run applications.

Step 1 Export the JAR file.

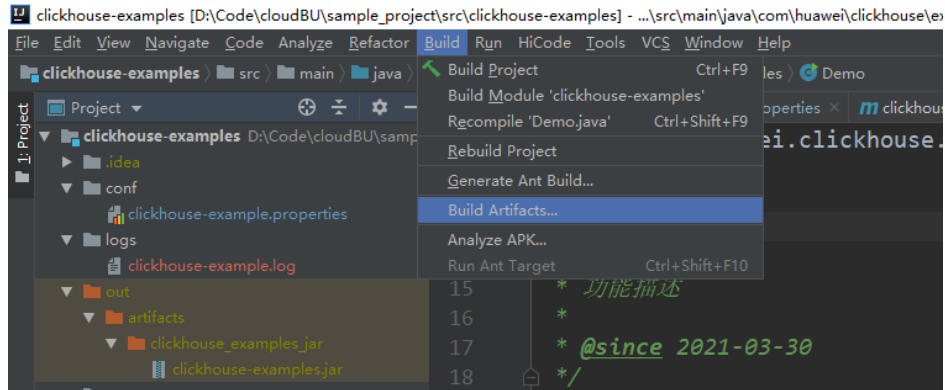
1. Log in to IntelliJ IDEA and choose **File > Project Structure > Artifacts**.
2. Click the plus sign (+) and choose **JAR > From modules with dependencies**.



3. Choose **com.huawei.clickhouse.examples.Demo** from the **Main Class** dropdown list and click **OK**.



4. Choose **Build > Build Artifacts...**. After the compilation is successful, view and obtain the **clickhouse_examples.jar** file in the **clickhouse-examples\out\artifacts** directory.



Step 2 Copy the **clickhouse_examples.jar** file and the **conf** folder in the **clickhouse-examples** directory to the ClickHouse client installation directory, for example, **/opt/client**.

Step 3 Log in to the client node, go to the directory where the JAR file is uploaded, and change the file permission to 700.

```
cd /opt/client
```

```
chmod 700 clickhouse-examples.jar
```

Step 4 In the client directory where **clickhouse_examples.jar** is stored, run the following commands to run the JAR file:

```
source bigdata_env
```

```
java -cp ./*:conf/clickhouse-example.properties  
com.huawei.clickhouse.examples.Demo
```

```
----End
```

Viewing Commissioning Results

After a ClickHouse application is run, you can use one of the following methods to view the running result:

- Viewing the command output
- Viewing ClickHouse logs

That is, view the **logs/clickhouse-example.log** file in the directory where the current JAR file is located, for example, **/opt/client/logs/clickhouse-example.log**.

The execution result of the JAR file is as follows:

```
2021-06-10 20:53:56,028 | INFO | main | Current load balancer is 10.112.17.150:21426 |  
com.huawei.clickhouse.examples.Util.insertData(Util.java:128)  
2021-06-10 20:53:58,247 | INFO | main | Inert batch time is 1442 ms |  
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)  
2021-06-10 20:53:59,649 | INFO | main | Inert batch time is 1313 ms |  
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)  
2021-06-10 20:54:05,872 | INFO | main | Inert batch time is 6132 ms |  
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)  
2021-06-10 20:54:10,223 | INFO | main | Inert batch time is 4272 ms |  
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)  
2021-06-10 20:54:11,614 | INFO | main | Inert batch time is 1300 ms |  
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)  
2021-06-10 20:54:12,871 | INFO | main | Inert batch time is 1200 ms |  
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
```

```
2021-06-10 20:54:14,589 | INFO | main | Inert batch time is 1663 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:16,141 | INFO | main | Inert batch time is 1500 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:17,690 | INFO | main | Inert batch time is 1498 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:19,206 | INFO | main | Inert batch time is 1468 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:19,207 | INFO | main | Inert all batch time is 22626 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:148)
2021-06-10 20:54:19,208 | INFO | main | Current load balancer is 10.112.17.150:21426 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-06-10 20:54:20,231 | INFO | main | Execute query:select * from mutong1.testtb_all order by age
limit 10 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-06-10 20:54:21,266 | INFO | main | Execute time is 1035 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-06-10 20:54:21,267 | INFO | main | Current load balancer is 10.112.17.150:21426 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-06-10 20:54:21,815 | INFO | main | Execute query:select toYYYYMM(date),count(1) from
mutong1.testtb_all group by toYYYYMM(date) order by count(1) DESC limit 10 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-06-10 20:54:22,897 | INFO | main | Execute time is 1082 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-06-10 20:54:22,898 | INFO | main | name age date |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,898 | INFO | main | huawei_266 0 2021-12-19 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_2500 0 2021-12-29 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_8980 0 2021-12-16 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_671 0 2021-12-29 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_2225 0 2021-12-12 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_6040 0 2021-12-14 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_7294 0 2021-12-10 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_1133 0 2021-12-25 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | huawei_3161 0 2021-12-21 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | huawei_3992 0 2021-11-25 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | toYYYYMM(date) count() |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | 201910 2247 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | 202105 2213 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | 201801 2208 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | 201803 2204 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201810 2167 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201805 2166 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201901 2164 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201908 2145 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201912 2143 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 202107 2137 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
```

10 Flink Development Guide (Security Mode)

10.1 Overview

10.1.1 Application Development

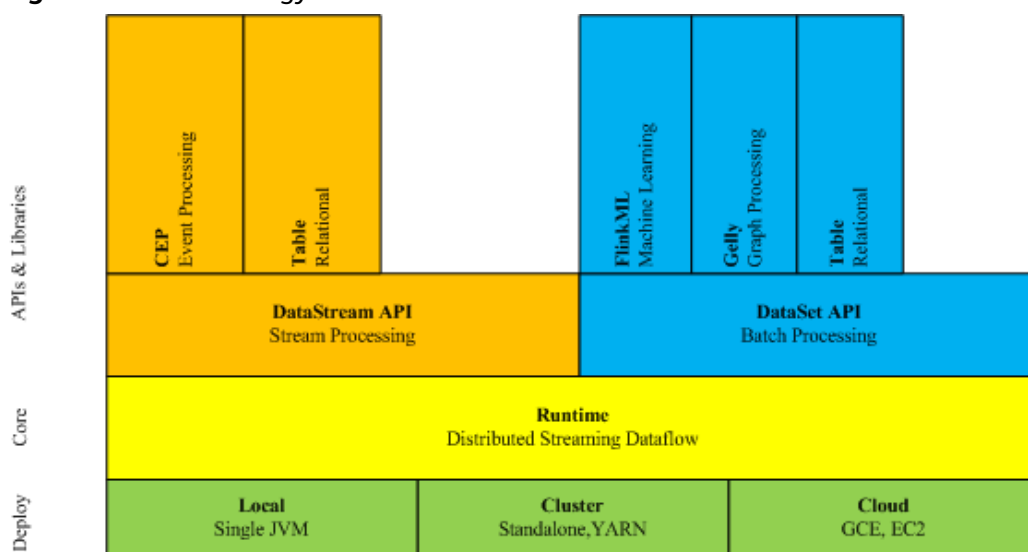
Overview

Flink is a unified computing framework that supports both batch processing and stream processing. It provides a stream data processing engine that supports data distribution and parallel computing. Flink features stream processing and is a top open-source stream processing engine in the industry.

Flink provides high-concurrency pipeline data processing, millisecond-level latency, and high reliability, making it suitable for low-latency data processing.

Figure 10-1 shows the technology stack of Flink.

Figure 10-1 Technology stack of Flink



The following lists the key features of Flink in the current version:

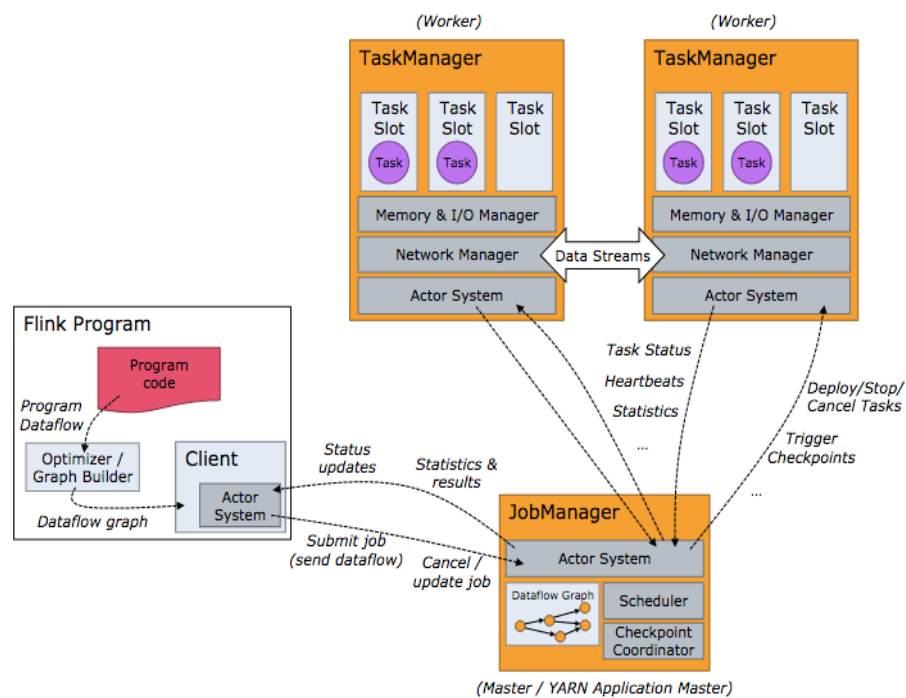
- DataStream
- Checkpoint
- Window
- Job Pipeline
- Configuration Table

For details about other Flink features, see <https://ci.apache.org/projects/flink/flink-docs-release-1.15>.

Architecture

Figure 10-2 shows the architecture of Flink.

Figure 10-2 Flink architecture



As shown in Figure 10-2, the entire Flink system consists of three parts:

- Client
Flink client is used to submit jobs (streaming jobs) to Flink.
- TaskManager
TaskManager (also called worker) is a service execution node of Flink. It executes specific tasks. A Flink system could have multiple TaskManagers. These TaskManagers are equivalent to each other.
- JobManager
JobManager (also called master) is a management node of Flink. It manages all TaskManagers and schedules tasks submitted by users to specific

TaskManagers. In high-availability (HA) mode, multiple JobManagers are deployed. Among these JobManagers, one of which is selected as the leader, and the others are standby.

Flink provides the following features:

- Low latency
Millisecond-level processing capability.
- Exactly once
Asynchronous snapshot mechanism, ensuring that all data is processed only once.
- High availability
Leader/Standby JobManagers, preventing single point of failure (SPOF).
- Scale out
Manual scale out supported by TaskManagers.

Flink Development APIs

Flink DataStream API can be developed using Scala and Java languages, as shown in [Table 10-1](#).

Table 10-1 Flink DataStream API

Function	Description
Scala API	API in Scala, which can be used for data processing, such as filtering, joining, windowing, and aggregation.
Java API	API in Java, which can be used for data processing, such as filtering, joining, windowing, and aggregation.

10.1.2 Basic Concepts

Basic Concepts

- **DataStream**
DataStream is the minimum unit of Flink processing and is one of core concepts of Flink. DataStreams are initially imported from external systems in formats of socket, Kafka, and files. After being processed by Flink, DataStreams are exported to external systems in formats of socket, Kafka, and files.
- **Data Transformation**
A data transformation is a data processing unit that transforms one or multiple DataStreams into a new DataStream.
Data transformation can be classified as follows:
 - One-to-one transformation, for example, map.
 - One-to-zero, one-to-one, or one-to-multiple transformation, for example, flatMap.

- One-to-zero or one-to-one transformation, for example, filter.
- Multiple-to-one transformation, for example, union.
- Transformation of multiple aggregations, for example, window and keyby.
- **Checkpoint**
Checkpoint is the most important Flink mechanism to ensure reliable data processing. Checkpoints ensure that all application statuses can be recovered from a checkpoint in case of failure occurs and data is processed exactly once.
- **Savepoint**
Savepoints are externally stored checkpoints that you can use to stop-and-resume or update your Flink programs. After the upgrade, you can set the task status to the savepoint storage status and start the restoration, ensuring data continuity.

10.1.3 Development Process

Development Process of a Flink Application

[Figure 10-3](#) shows the Flink development process:

Figure 10-3 Development process of a Flink application

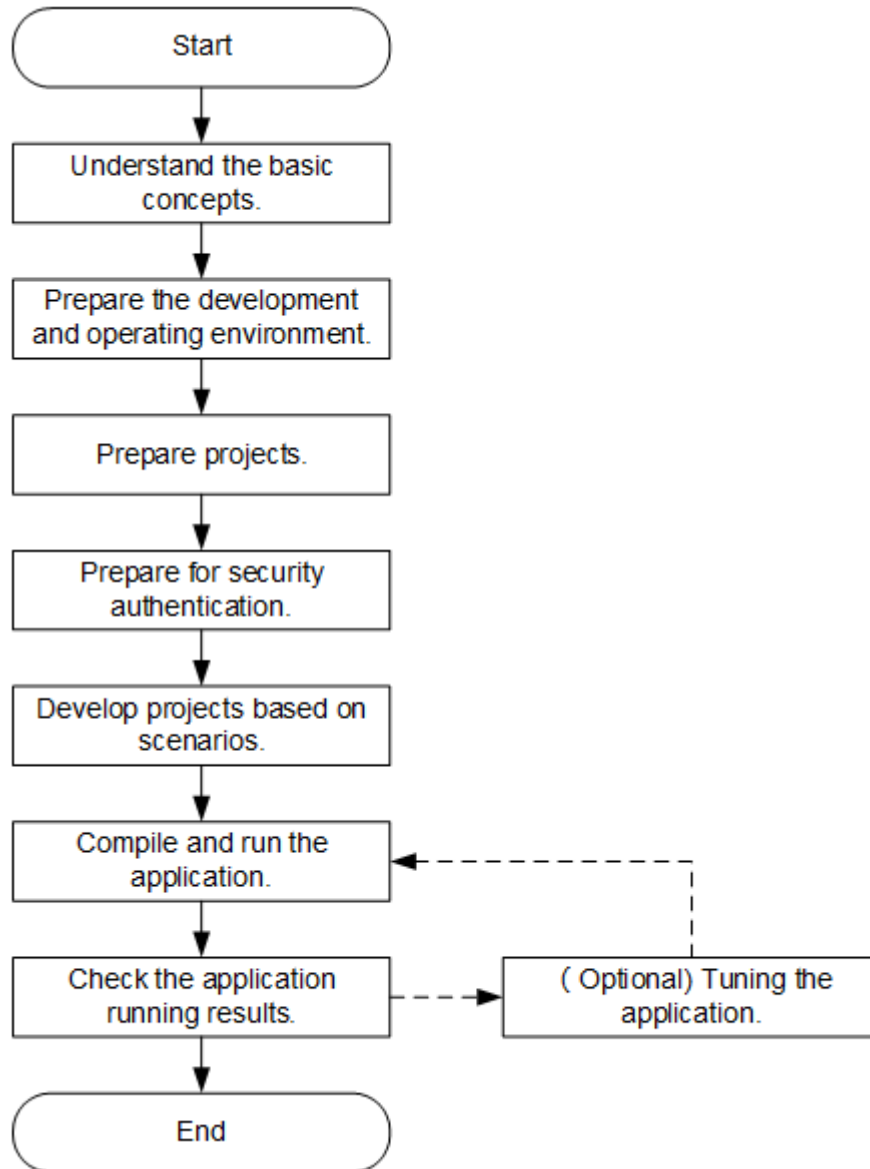


Table 10-2 Description of the development process

Phase	Description	Reference
Understand the basic concepts.	Before the development process, you are advised to gain a basic understanding of Flink.	Basic Concepts

Phase	Description	Reference
Prepare the development and operating environment.	Flink applications can be developed using Scala or Java. You are advised to use IntelliJ IDEA tool to configure the development environment as instructed, based on your development language. The running environment of Flink is the Flink client. Install and configure the client as instructed.	Preparing for Development and Operating Environment
Prepare projects.	Flink provides sample projects for you to import and learn. Alternatively, you can create a Flink project as instructed.	Configuring and Importing a Sample Project Creating a Project (Optional)
Prepare for security authentication.	The security authentication is mandatory if security clusters are used.	Preparing for Security Authentication
Develop the project.	Sample projects in Scala and Java are provided to help you quickly understand programming interfaces of Flink components.	Developing an Application
Compile and run the application.	Guidance is provided for you to compile and run a developed application.	Compiling and Running the Application
Check running results.	Application running results are stored in a path specified by you. You can also view application running status through Apache Flink Dashboard.	Viewing the Debugging Result
Tune the application.	Tune the application to meet certain service requirements. After the tuning is complete, the application needs to be compiled and run again.	"Flink Performance Tuning" in the Component Operation Guide

10.2 Environment Preparation

10.2.1 Preparing for Development and Operating Environment

Preparing Development Environment

[Table 10-3](#) describes the environment required for application development.

Table 10-3 Development environment

Item	Description
OS	<ul style="list-style-type: none"> Development environment: Windows OS. Windows 7 or later is supported. Operating environment: Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> Oracle JDK versions: 1.8 IBM JDK versions: 1.8.5.11 For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> OpenJDK: 1.8.0_272
IntelliJ IDEA installation and configuration	IntelliJ IDEA is a tool used to develop Flink applications. The version must be 2019.1 or other compatible version.
Scala installation	Install Scala is the basic configuration for the Scala development environment. The required version is 2.11.7.
Scala plug-in installation	Installing Scala plug-ins is the basic configuration for the Scala development environment. The required version is 1.5.4.
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
Developer account preparation	See Preparing a Developer Account for configuration.
7-zip	It is a tool used to decompress .zip and .rar packages. The 7-Zip 16.04 is supported.

Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
 - a. Install the client on the node. For example, the client installation directory is **/opt/client**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

 NOTE

- Verify that the configuration options in the Flink client configuration file **flink-conf.yaml** are correctly configured. For details, see [Preparing for Security Authentication](#).
 - In security mode, appended the service IP address of the node where the client is installed and floating IP address of Manager to the `jobmanager.web.allow-access-address` configuration item in the `flink-conf.yaml` file. Use commas (,) to separate IP addresses.
- b. Log in to the FusionInsight Manager portal. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight_Cluster_1_Services_ClientConfig\Flink\config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/client/conf**.

For example, if the client software package is **FusionInsight_Cluster_1_Services_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Flink/config/* root@IP address of the client node:/opt/client/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 10-4](#) describes the main configuration files.

Table 10-4 Configuration file

Document Name	Function
core-site.xml	Configures Flink parameters.
hdfs-site.xml	Configures hdfs parameters.
yarn-site.xml	Configures yarn parameters.

Document Name	Function
flink-conf.yaml	Configures Flink parameters.
user.keytab	Provides user information for Kerberos security authentication.
krb5.conf	Kerberos server configuration information

- c. Check the network connection of the client node.
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

10.2.2 Configuring and Importing a Sample Project

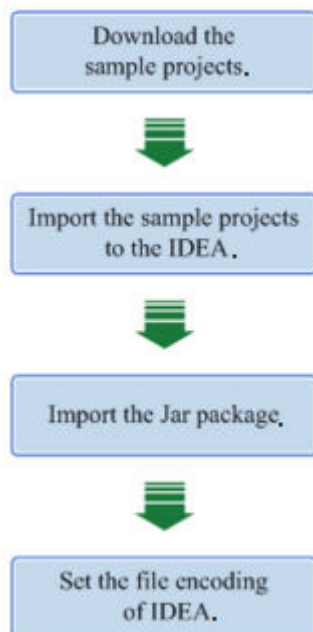
Scenarios

Flink provides sample projects for multiple scenarios, including Java and Scala sample projects, to help you quickly learn Flink projects.

Methods to import Java and Scala projects are the same.

The following example describes how to import Java sample code. [Figure 10-4](#) shows the operation process.

Figure 10-4 Process of importing sample projects



Procedure

Step 1 Obtain the sample project folder **flink-examples-security** in the **src\flink-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

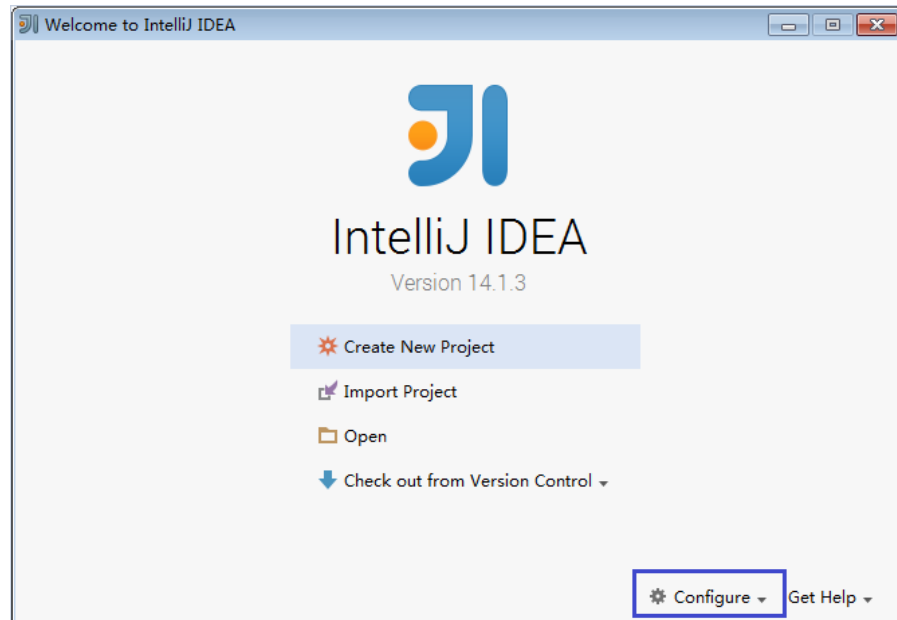
 **NOTE**

- In security mode, obtain the sample project **flink-examples-security** from the **src\flink-examples** folder.
- In normal mode, obtain the sample project **flink-examples-normal** from the **src\flink-examples** folder.

Step 2 Before importing the sample project, configure JDK for IntelliJ IDEA.

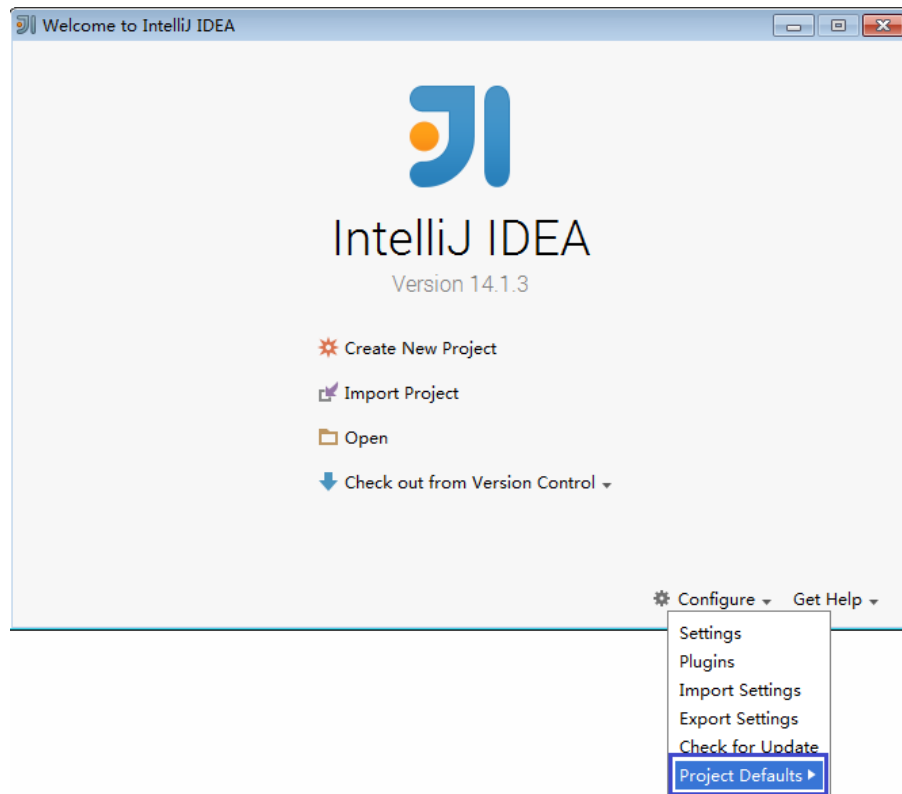
1. Start IntelliJ IDEA and click **Configure**.

Figure 10-5 Choosing Configure



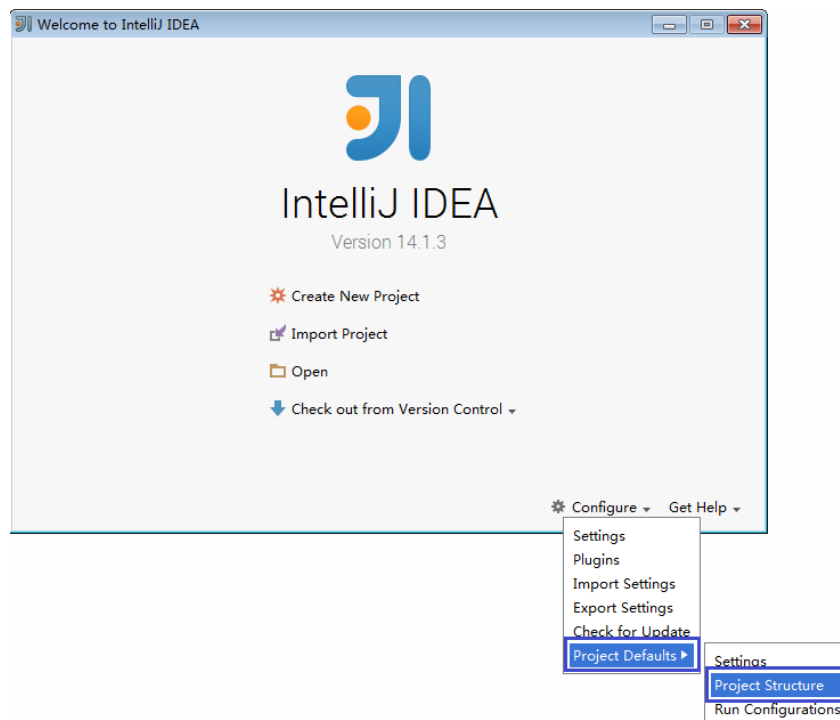
2. Choose **Project Defaults** from the **Configure** drop-down list.

Figure 10-6 Choosing Project Defaults



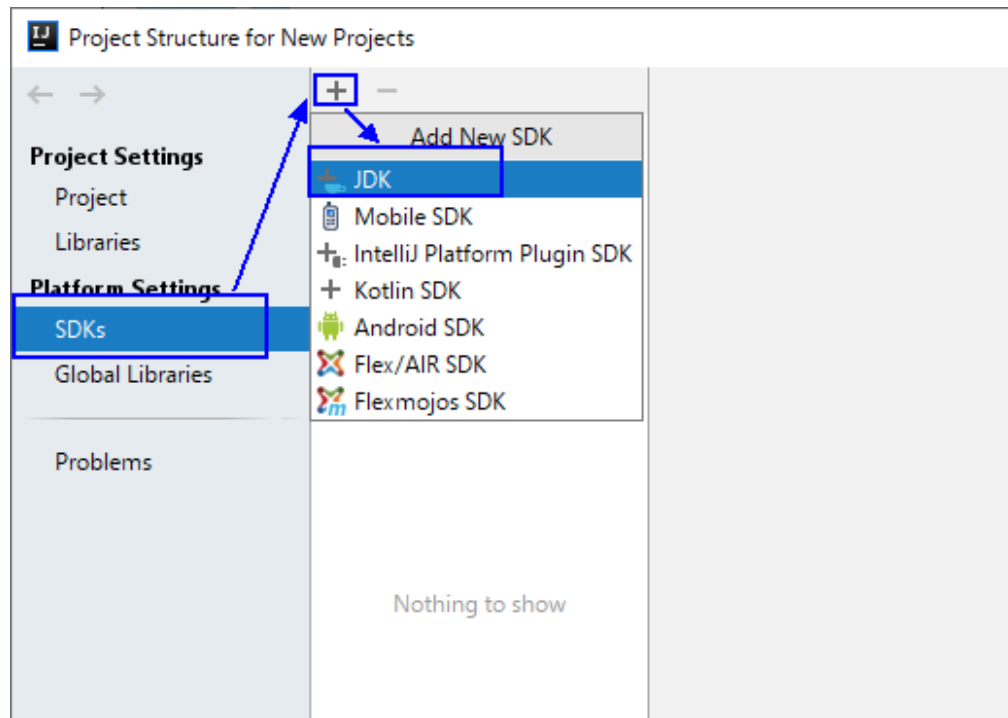
3. Choose **Project Structure** from the **Project Defaults** submenu.

Figure 10-7 Project Defaults



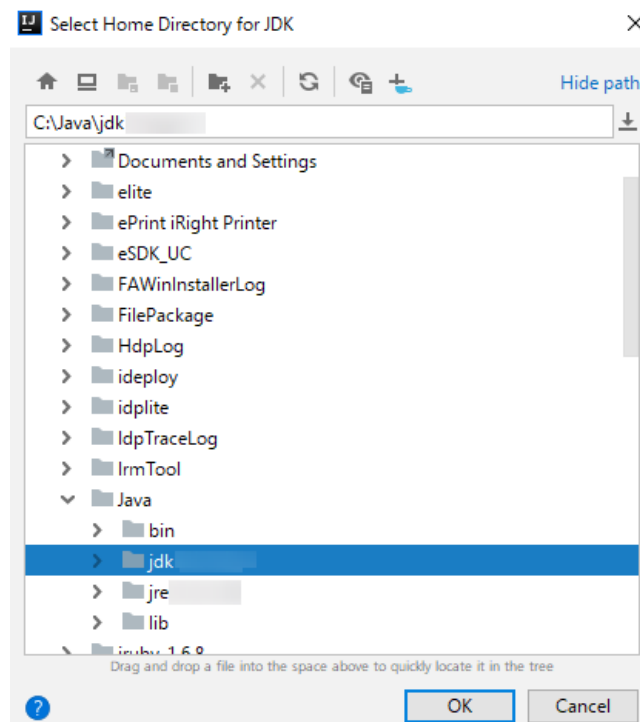
4. On the **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 10-8 Adding the JDK



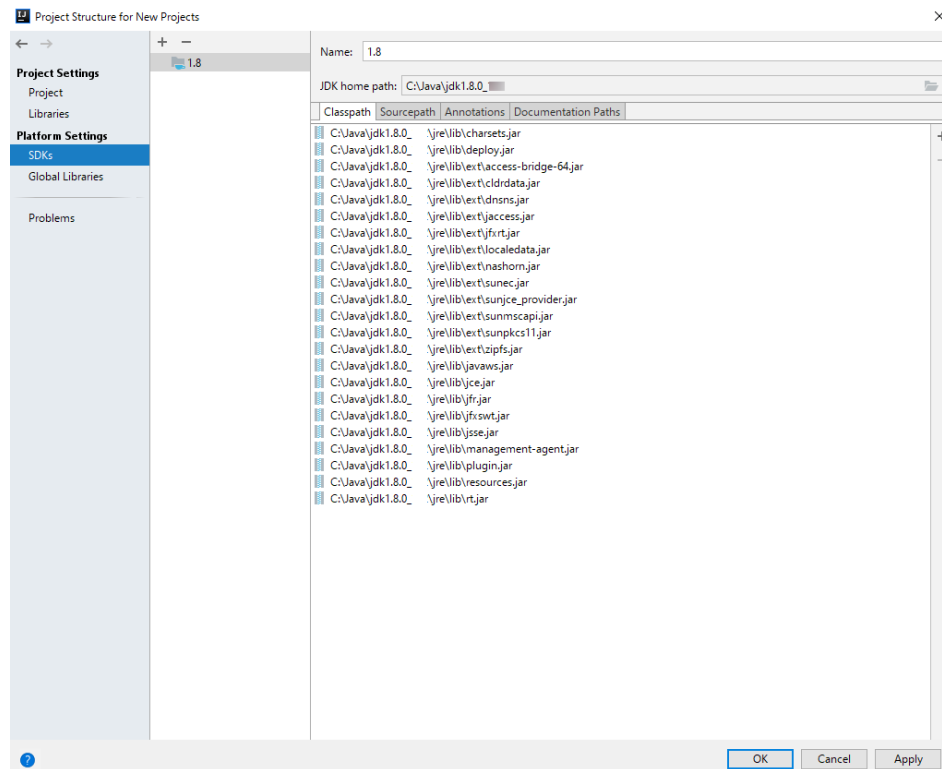
5. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 10-9 Selecting the JDK directory



6. After the JDK is selected, click **OK** to complete the configuration.

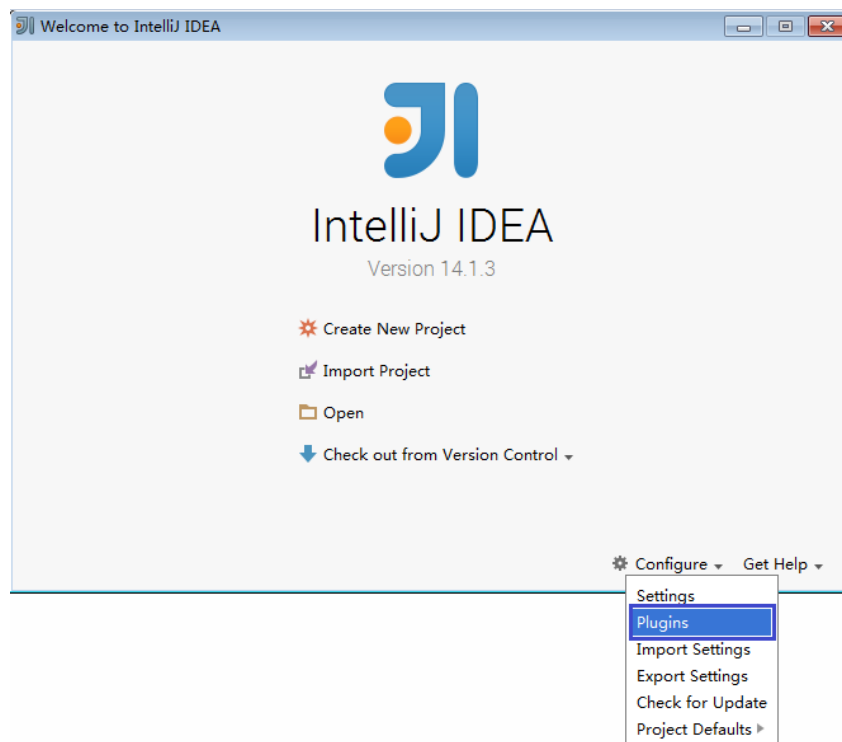
Figure 10-10 Completing the JDK configuration



Step 3 (Optional) If a Scala sample project is imported, install Scala plug-ins in IntelliJ IDEA.

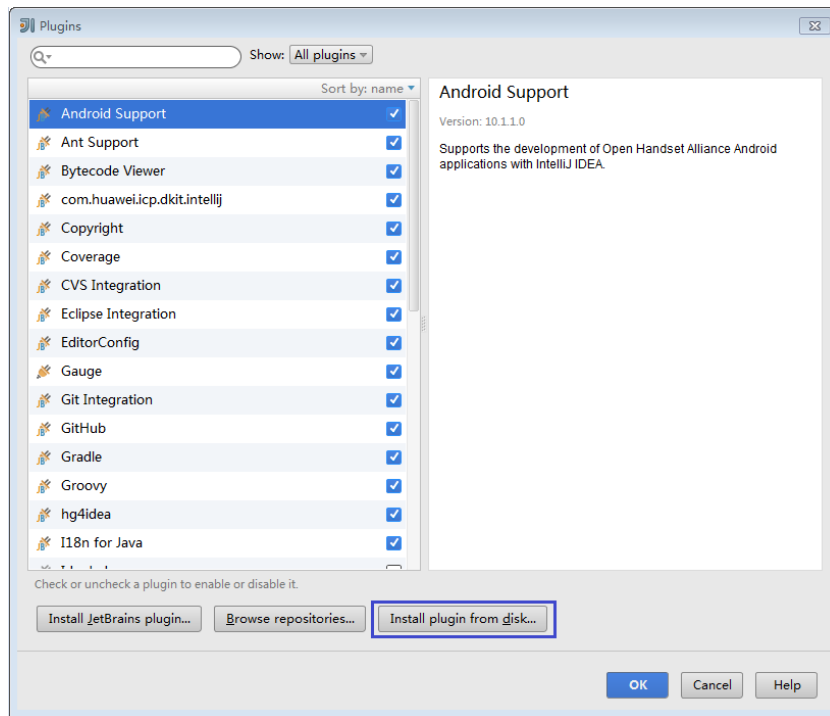
1. Choose **Plugins** from the **Configure** drop-down list.

Figure 10-11 Plugins



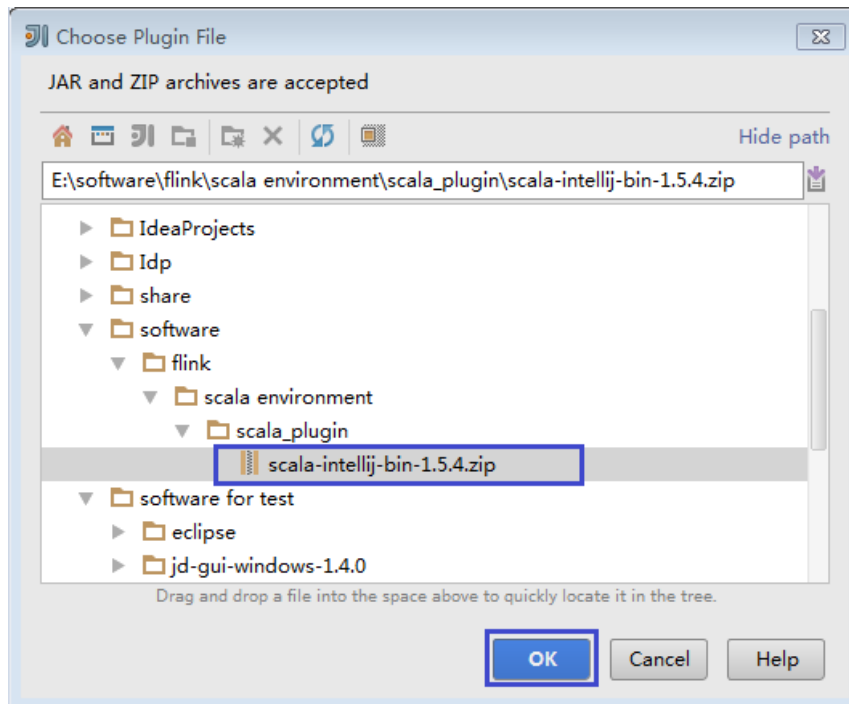
2. On the **Plugins** page, click **Install plugin from disk**.

Figure 10-12 Install plugin from disk



3. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.

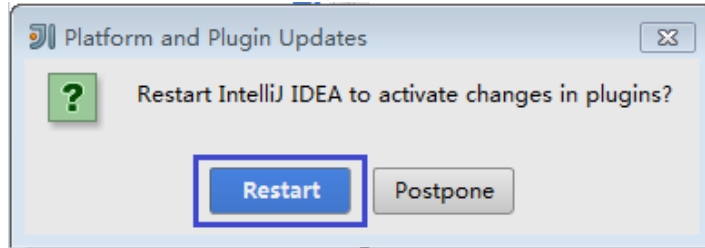
Figure 10-13 Choose Plugin File



4. On the **Plugins** page, click **Apply** to install the Scala plugins.

5. On the **Platform and Plugin Updates** page that is displayed, click **Restart** to make the configurations take effect.

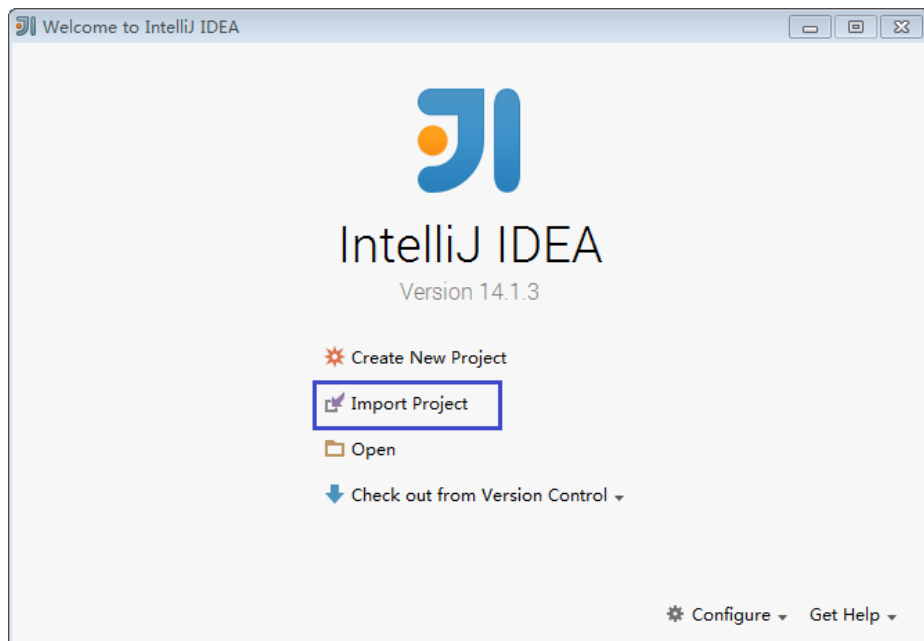
Figure 10-14 Platform and Plugin Updates



Step 4 Import the Java sample project to IDEA.

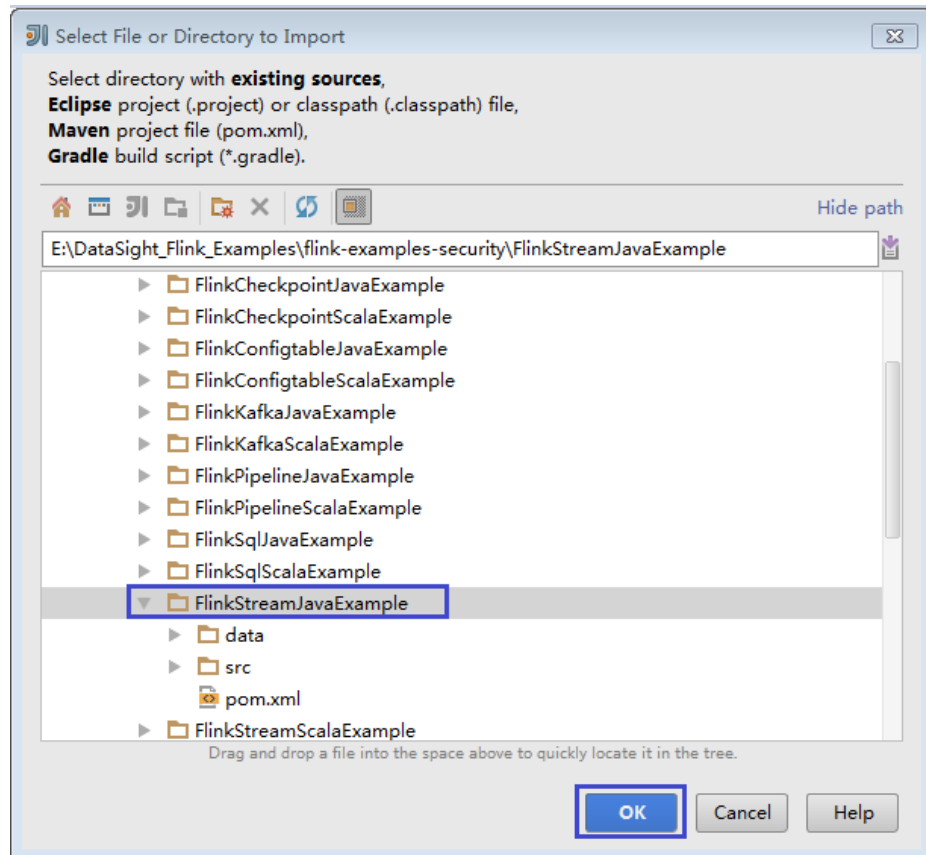
1. Start IntelliJ IDEA. On the **Quick Start** page, select **Import Project**.
Alternatively, for the used IDEA tool, add the project directly from the IDEA home page. Choose **File > Import project...** to import a project.

Figure 10-15 Importing a project (on the Quick Start page)



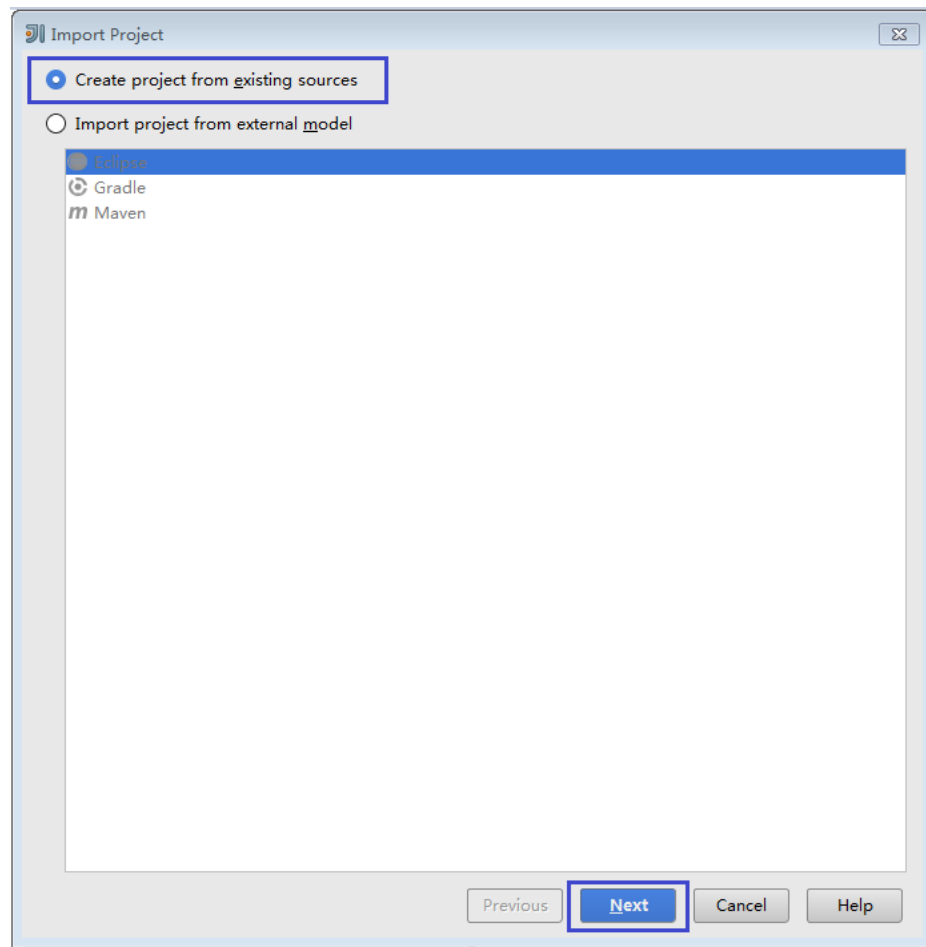
2. Select the directory for storing the imported projects and click **OK**.

Figure 10-16 Select File or Directory to Import



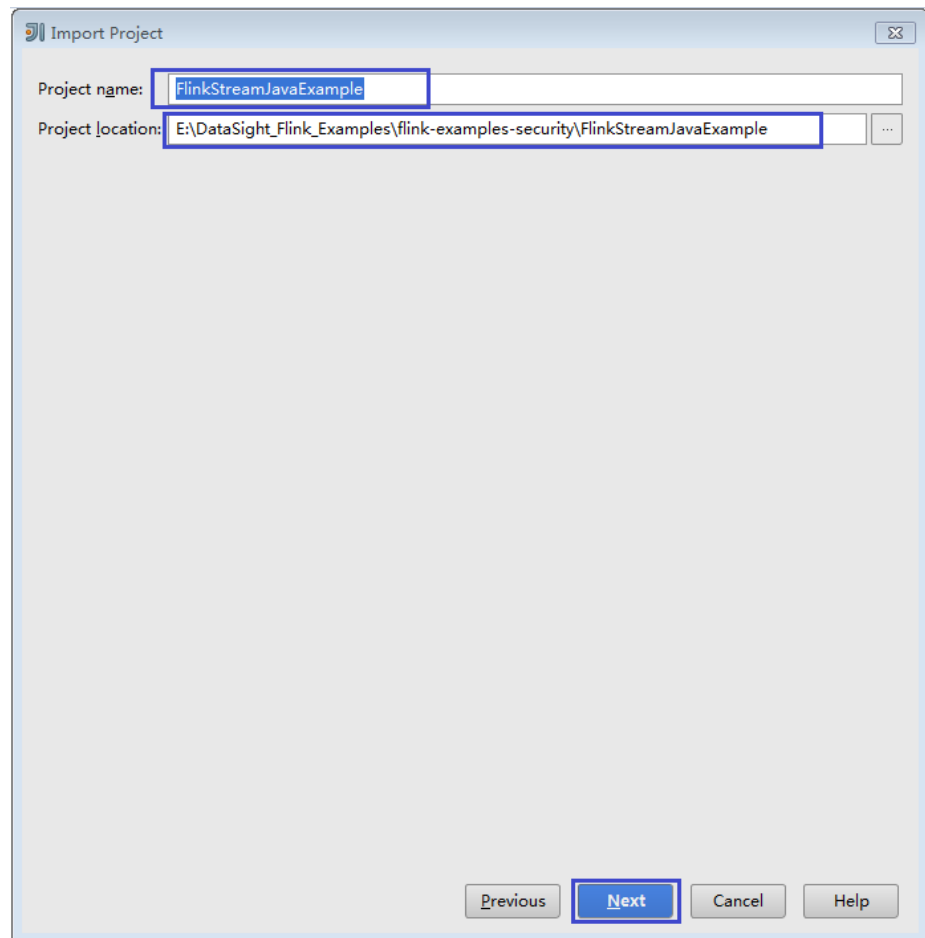
3. Select **Create project from existing sources** and click **Next**.

Figure 10-17 Create project from existing sources



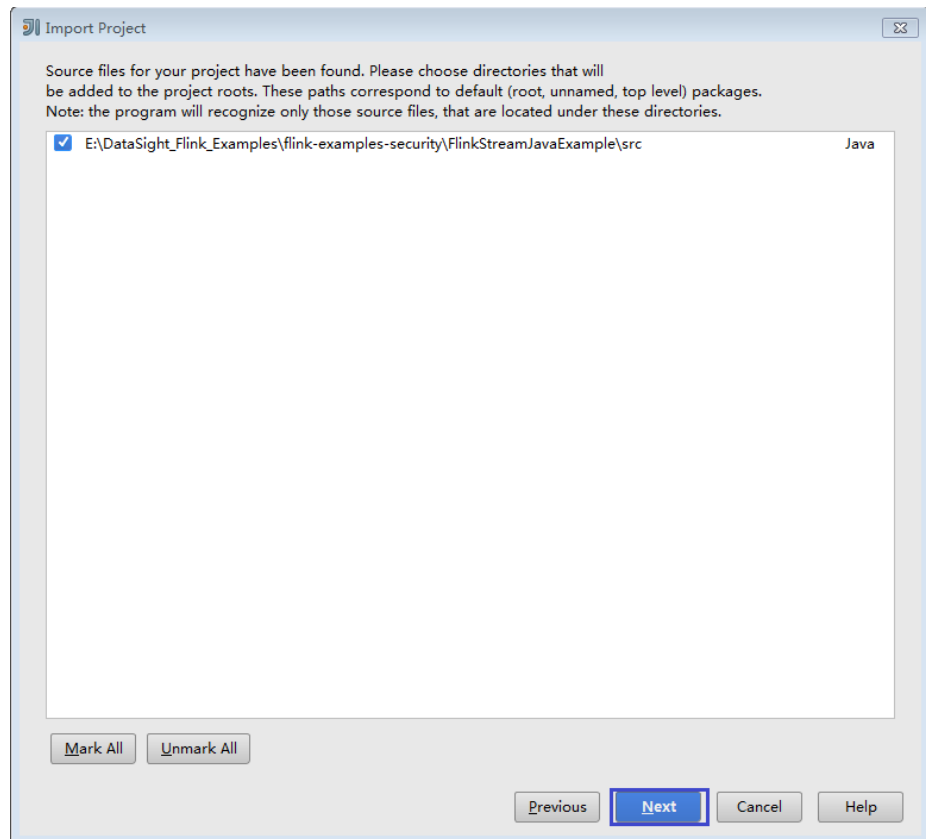
4. Confirm the project location and project name, and click **Next**.

Figure 10-18 Import Project



5. Retain the default value of the **root** directory for the project to be imported and click **Next**.

Figure 10-19 Import Project



6. Retain the default dependency library that is automatically recognized by IDEA and the suggested module structure, and click **Next**.
7. Confirm the JDK to be used by the project and click **Next**.
8. After the import is complete, click **Finish**. The imported sample project is displayed on the IDEA home page.

Figure 10-20 Completing the import

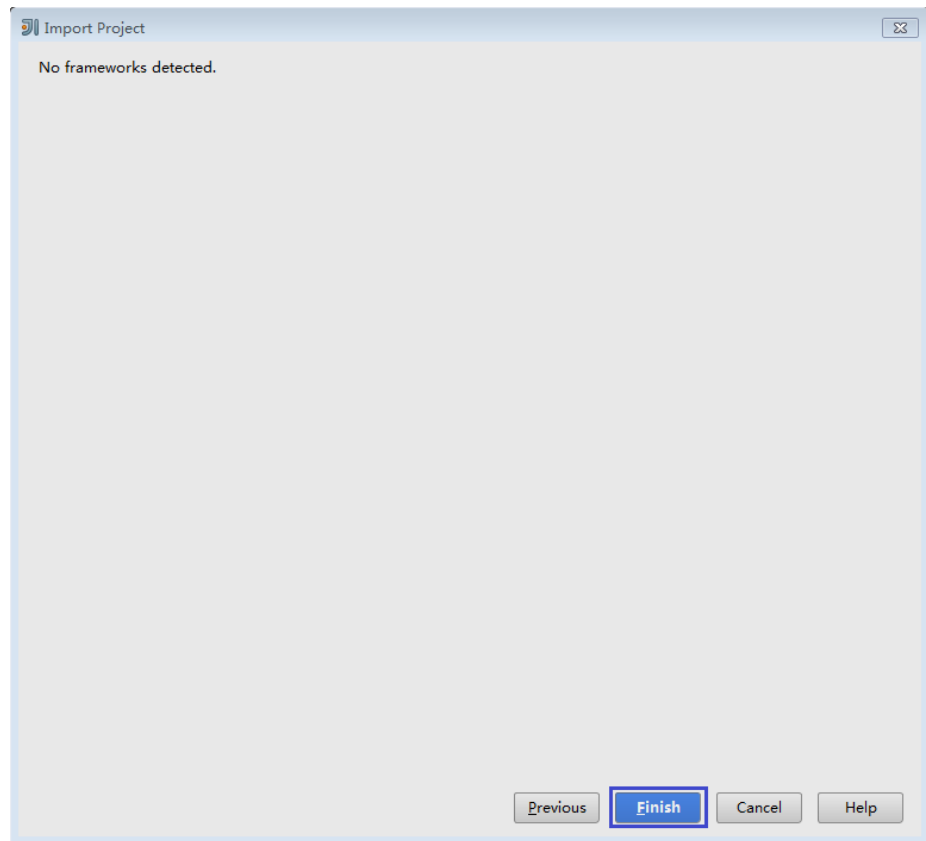
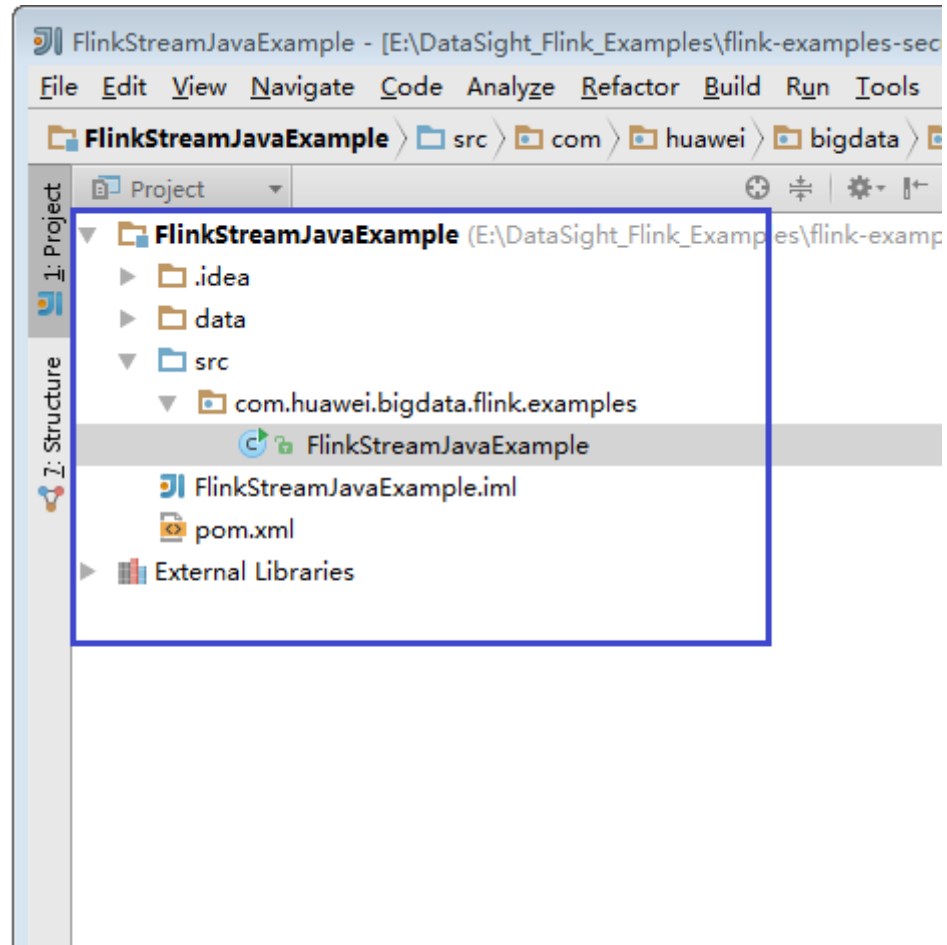


Figure 10-21 Imported project



Step 5 Import the dependency JAR file for the sample project.

If the sample project code is obtained from an open-source image site, dependency JAR files are automatically downloaded after Maven is configured.

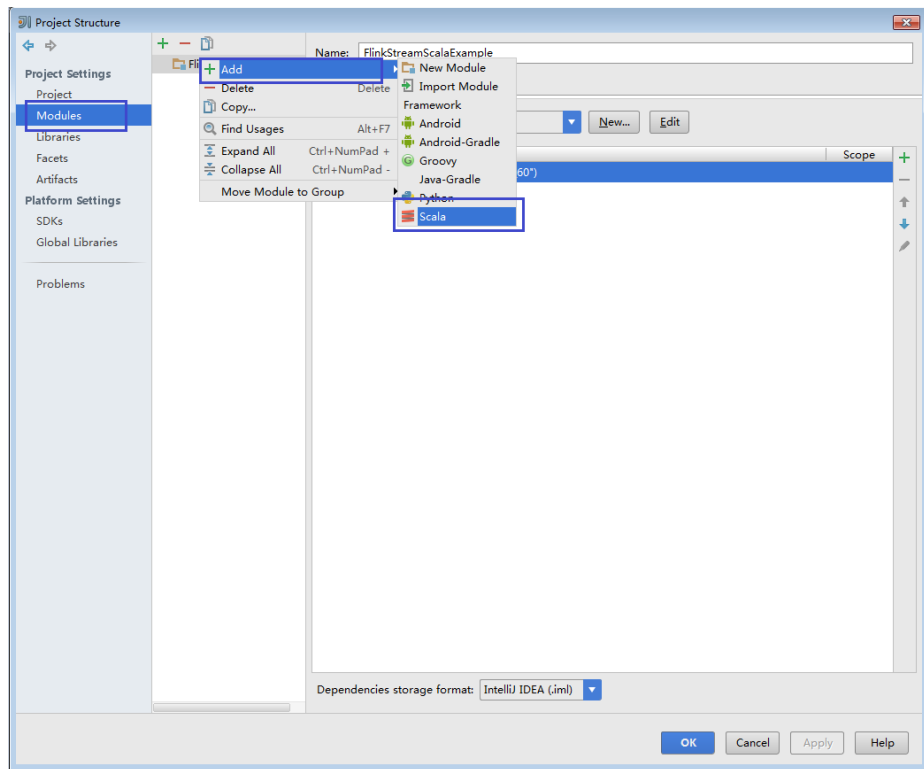
NOTE

If other FusionInsight components such as Kafka are used in the sample code, obtain them from the installation directory of these FusionInsight components. For details about dependency packages of sample projects, see [Reference Information](#).

Step 6 (Optional) If a Scala sample application is imported, configure a language for the project.

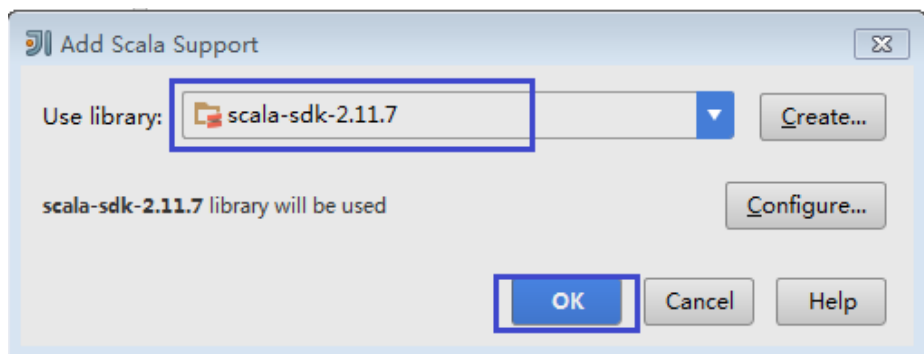
1. On the IDEA home page, choose **File > Project Structures...** to go to the **Project Structure** page.
2. Choose **Modules**, right-click a project name, and choose **Add > Scala**.

Figure 10-22 Selecting Scala



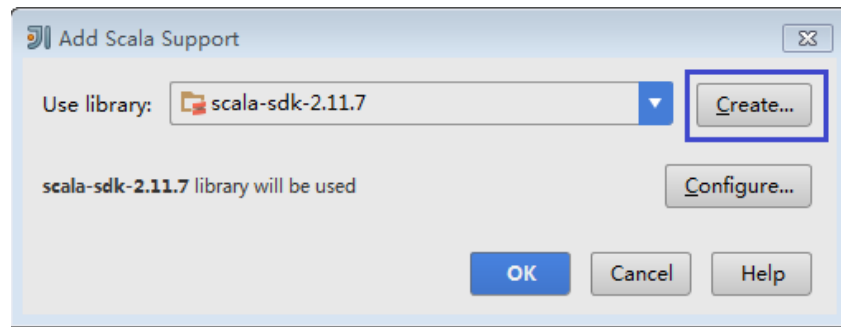
3. Wait until IDEA identifies Scala SDK, select the dependency JAR files in the **Add Scala Support** dialog box, and then click **OK**.

Figure 10-23 Add Scala Support



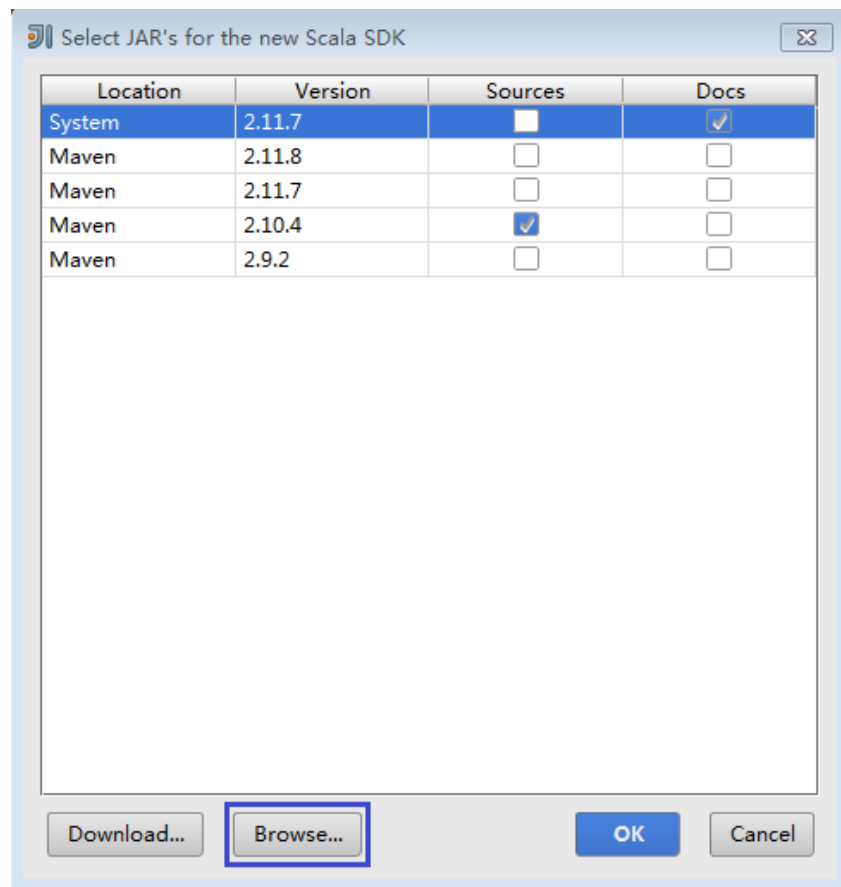
4. If IDEA fails to identify Scala SDK, create a Scala SDK.
 - a. Click **Create....**

Figure 10-24 Create...



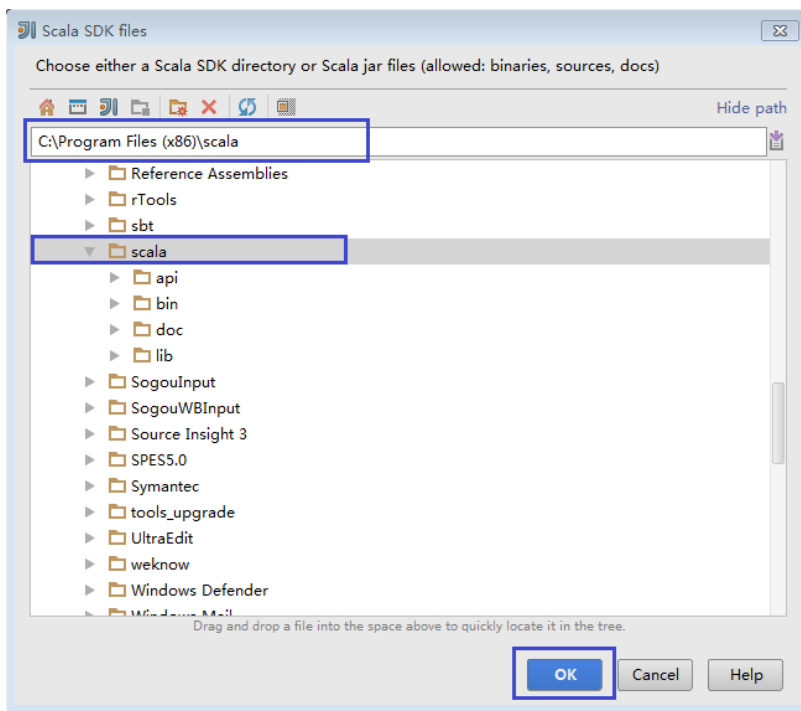
- b. On the **Select JAR's for the new Scala SDK** page, click **Browse...**

Figure 10-25 Select JAR's for the new Scala SDK



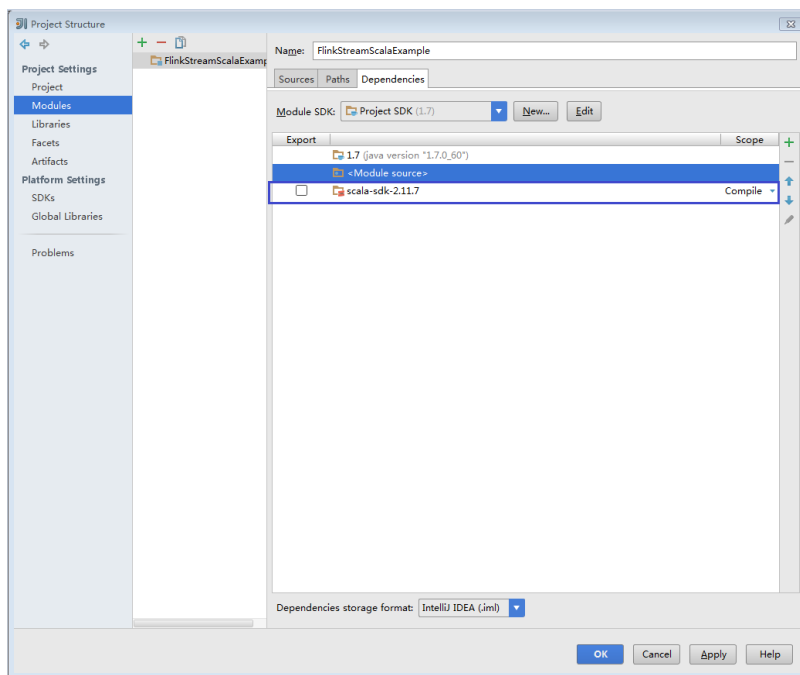
- c. On the **Scala SDK files** page, select the **scala sdk** directory and click **OK**.

Figure 10-26 Scala SDK files



- 5. Click **OK**.

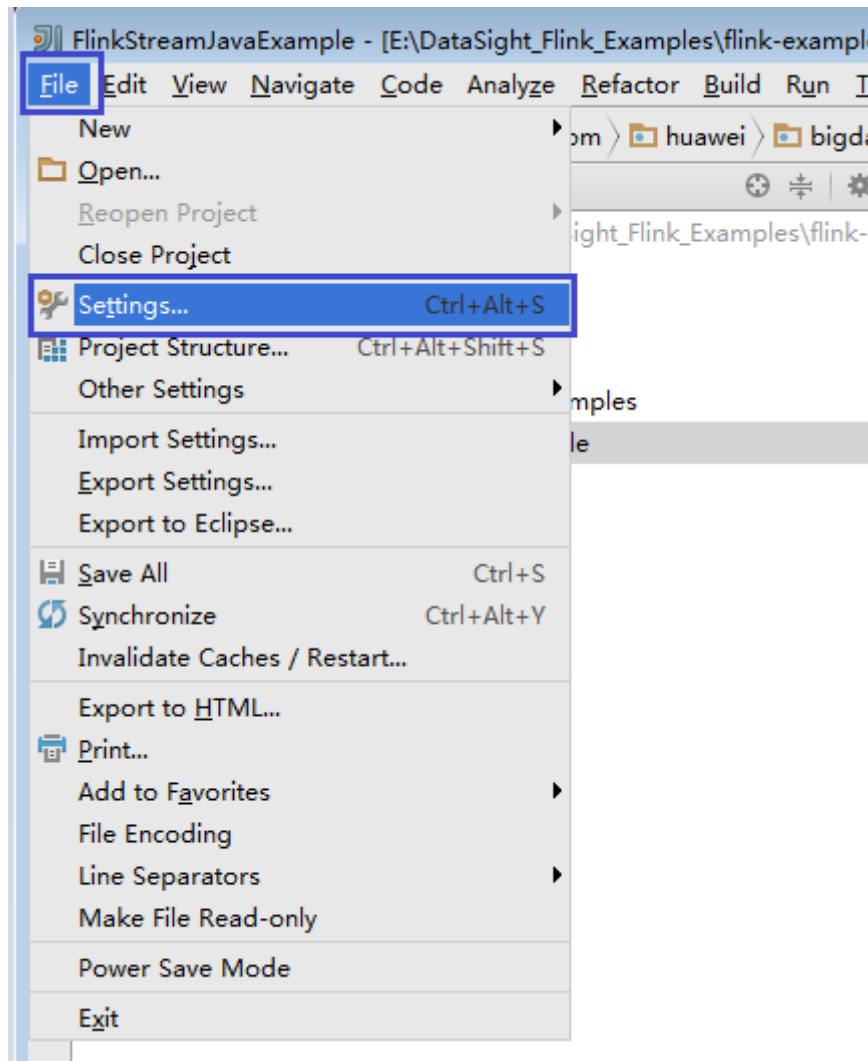
Figure 10-27 The configuration is successful.



Step 7 Configure the text file encoding format of IDEA to prevent garbled characters.

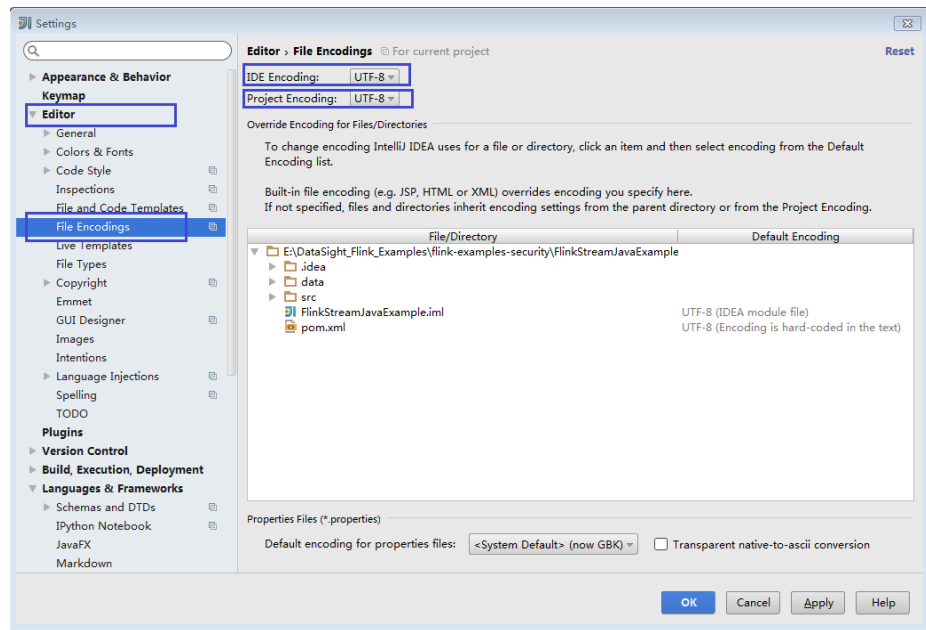
- 1. On the IDEA home page, choose **File > Settings...**

Figure 10-28 Choosing **Settings**



2. Configure encoding.
 - a. On the **Settings** page, unfold **Editor**, and choose **File Encodings**.
 - b. Select **UTF-8** from the **IDE Encoding** and **Project Encoding** drop-down lists on the right.
 - c. Click **Apply**.
 - d. Click **OK** to complete the encoding settings.

Figure 10-29 Settings



NOTE

- Obtain related dependency packages from the Flink server installation directory.
- Obtain Kafka dependency packages from the Kafka environment.
- For details about the dependency packages, see [Reference Information](#).

----End

Reference Information

The **lib** and **opt** directories of the Flink client contain Flink JAR files. By default, the **lib** directory contains the Flink core JAR file, and the **opt** directory contains the JAR file (for example, **flink-connector-kafka*.jar**) for connecting to external components. If it is required during application development, manually copy the related JAR files to the **lib** directory.

The dependency packages of the sample projects provided by Flink are as follows:

- Sample project of the DataStream application (in Java or Scala)
 - flink-dist_*.jar

NOTE

flink-dist_*.jar can be obtained from **lib** in the installation directory of the Flink client or server.

- Sample projects of the application for producing and consuming data in Kafka (in Java or Scala)
 - kafka-clients-*.jar
 - flink-connector-kafka_*.jar

 NOTE

- **flink-connector-kafka_*.jar** can be obtained from **opt** in the installation directory of the Flink client or server.
- **kafka-clients-*.jar** is provided by Kafka and can be obtained from the **lib** directory in the installation directory of the Kafka client or server.
- Sample project of the asynchronous checkpoint mechanism application (in Java or Scala)
 - `flink-dist_*.jar`
- Sample projects of pipeline (in Java or Scala)
 - `flink-connector-netty_*.jar`
 - `flink-dist_*.jar`
 - `curator-client-2.12.0.jar`
 - `curator-framework-2.12.0.jar`

 NOTE

flink-connector-netty_*.jar, **curator-client-2.12.0.jar**, and **curator-framework-2.12.0.jar** can be obtained from the **lib** folder generated after the secondary development sample code is compiled.

- Sample projects of Stream SQL JOIN (in Java)
 - `kafka-clients-*.jar`
 - `flink-connector-kafka_2.11*.jar`
 - `flink-dist_2.11*.jar`
 - `flink-table_2.11*.jar`

10.2.3 Creating a Project (Optional)

Scenarios

IntelliJ IDEA can be used to create a Flink project. A Java project is used as an example to illustrate how to create a Flink project.

Procedure

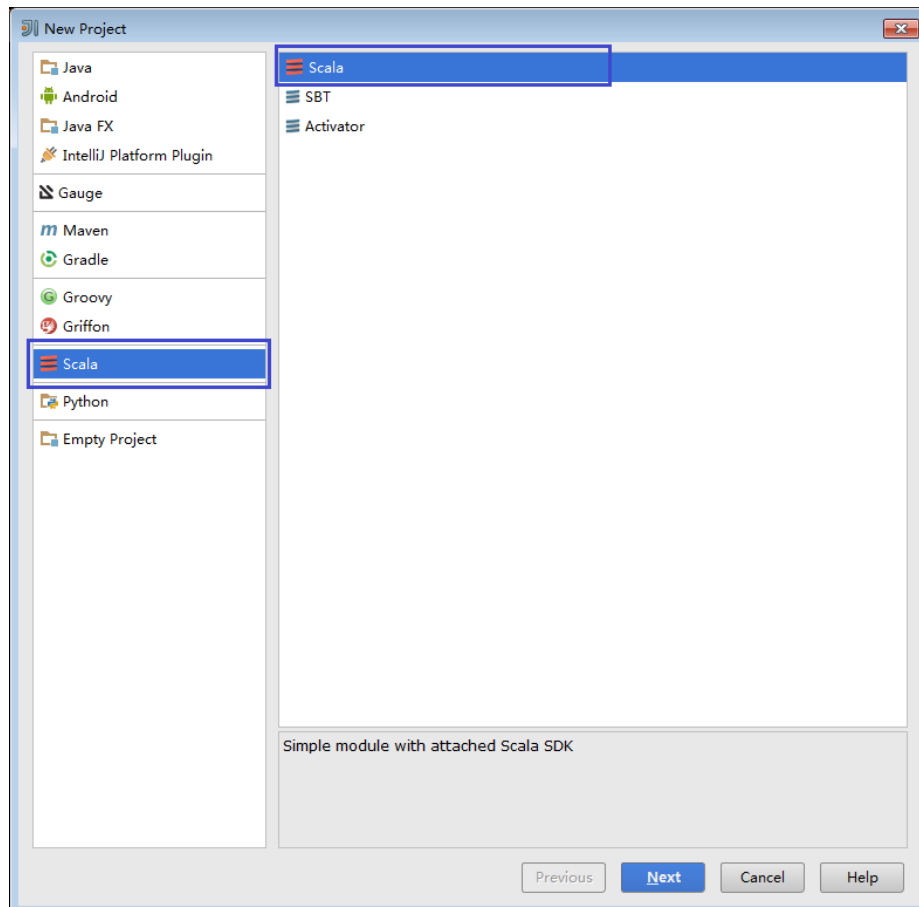
- Step 1** Start the IDEA and choose **Create New Project**.

Figure 10-30 Creating a project



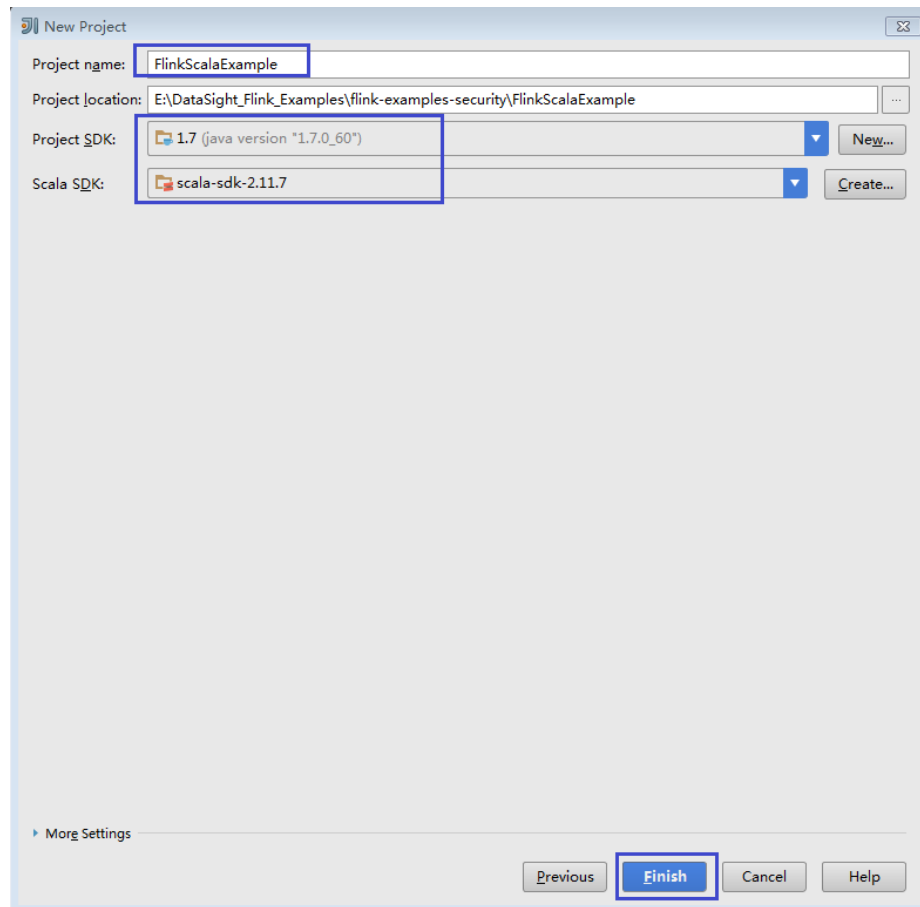
Step 2 On the **New Project** page, choose a Scala development environment, choose **Scala Module**, and then click **Next**. If you want to create a Java project, choose corresponding parameters of Java.

Figure 10-31 Selecting a development environment



Step 3 On the New Project page, enter the project name and project location, select the JDK version and Scala SDK, and then click **Finish**.

Figure 10-32 Filling in project information



----End

10.2.4 Preparing for Security Authentication

Scenarios

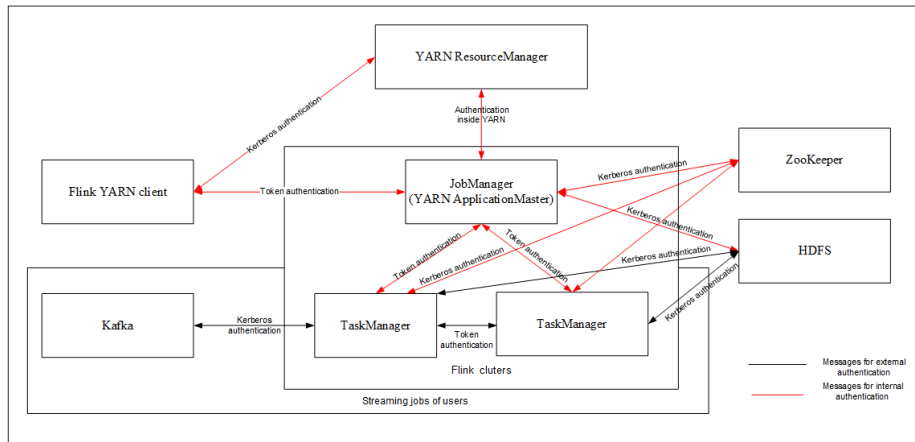
In secure cluster mode, all clusters must authenticate each other before communicating.

When you submit Flink applications, Flink applications need to communicate with components such as YARN and HDFS. Therefore, security authentication must be configured for Flink.

Flink supports authentication and encrypted transmission. This section describes preparations required for authentication and encrypted transmission.

Authentication

Figure 10-33 Authentication modes of Flink



Flink uses following authentication modes:

- **Kerberos authentication:** Kerberos authentication is used between Flink YARN client and YARN ResourceManager, JobManager and ZooKeeper, JobManager and HDFS, TaskManager and HDFS, Kafka and TaskManager, as well as TaskManager and ZooKeeper.
- **Security cookie authentication:** Security cookie authentication is used between Flink YARN client and JobManager, JobManager and TaskManager, as well as TaskManager and TaskManager.
- **Authentication inside YARN:** The Internal authentication mechanism of YARN is used between YARN ResourceManager and ApplicationMaster (AM).

NOTE

- Flink JobManager and YARN ApplicationMaster are in the same process.
- If you want to use the security mode, the kerberos authentication and security cookie authentication are mandatory.

Table 10-5 Authentication modes

Authentication Mode	Configuration Method
<p>Kerberos authentication Currently, only keytab authentication mode is supported.</p>	<ol style="list-style-type: none"> 1. Download user keytab from FusionInsight Manager, and place the keytab to a directory on the host of Flink client. 2. Configure following parameters in the flink-conf.yaml file: <ol style="list-style-type: none"> a. Keytab path. security.kerberos.login.keytab: /home/flinkuser/keytab/flinkuser.keytab NOTE /home/flinkuser/keytab/ indicates the directory for storing keytab. b. Principal name. security.kerberos.login.principal: flinkuser c. In HA mode, if Zookeeper is configured, the Kerberos authentication configuration items must be configured as follows: zookeeper.sasl.disable: false security.kerberos.login.contexts: Client d. If you want to perform Kerberos authentication between Kafka client and Kafka broker, set the value as follows: security.kerberos.login.contexts: Client,KafkaClient

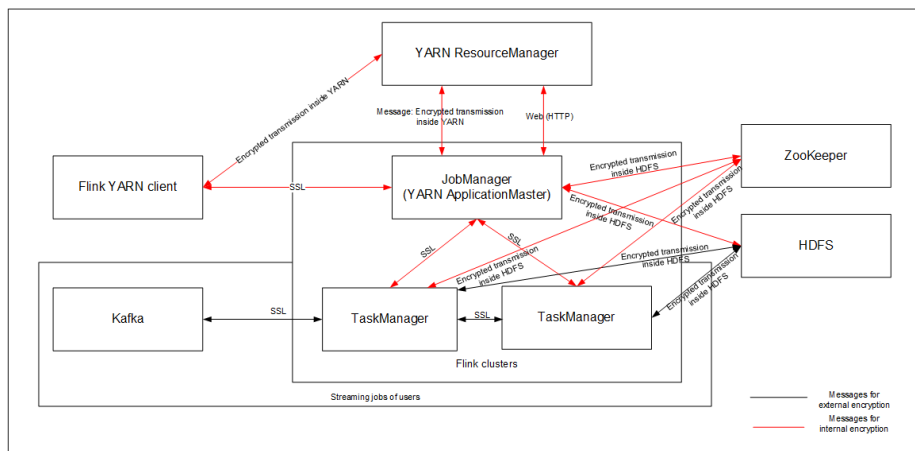
Authentication Mode	Configuration Method
Security cookie authentication	<ol style="list-style-type: none"> Obtain the SSL certificate and save it to the Flink client. For details, see Authentication and Encryption. Assign values to the following configuration items in the flink-conf.yaml file in the conf directory of the Flink client: <ul style="list-style-type: none"> Set the security.ssl.key-password, security.ssl.keystore-password, and security.ssl.truststore-password to <i><password></i>. Set security.ssl.keystore to the relative path of the keystore file, that is, <i>ssl/flink.keystore</i>. Set security.ssl.truststore to the relative path of the truststore file, that is, <i>ssl/flink.truststore</i>. Set security.cookie to a random password string. <p>NOTE The generated flink.keystore, flink.truststore, and security.cookie items are automatically filled in the corresponding configuration items in flink-conf.yaml as shown in Authentication and Encryption.</p> <p>The values of security.ssl.key-password, security.ssl.keystore-password, and security.ssl.truststore-password need to be obtained using the Manager plaintext encryption API by running the curl -k -i -u username:password -X POST -HContent-type:application/json -d '{"plainText": "password"}' 'https://x.x.x.x:28443/web/api/v2/tools/encrypt'. In the preceding command, <i>Username:Password</i> indicates the user name and password for logging in to the system. The password if "plainText" indicates the one used to call the generate_keystore.sh script. <i>x.x.x.x</i> indicates the floating IP address of Manager.</p> Set security.enable: true to true and configure security cookie. Example: <pre>security.cookie: ae70acc9-9795-4c48-ad35-8b5adc8071744f605d1d-2726-432e-88ae-dd39bfec40a9</pre>
Authentication inside YARN	The authentication mode does not need to be configured.

 **NOTE**

One Flink cluster supports only one user. One user can create multiple Flink clusters.

Encrypted Transmission

Figure 10-34 Encrypted transmission of Flink



Flink uses following encrypted transmission modes:

- Encrypted transmission inside YARN: encrypted transmission is used between Flink YARN client and YARN ResourceManager, as well as YARN ResourceManager and JobManager.
- SSL transmission: SSL transmission is used between Flink YARN client and JobManager, JobManager and TaskManager, as well as TaskManager and TaskManager.
- Encrypted transmission inside Hadoop: The internal encrypted transmission mode of Hadoop used between JobManager and HDFS, TaskManager and HDFS, JobManager and ZooKeeper, as well as TaskManager and ZooKeeper.

NOTE

Configuration about SSL encrypted transmission is mandatory while configuration about encryption of YARN and Hadoop is not required.

In the **flink-conf.yaml** file on the client, configure following parameters to configure the SSL transmission.

1. Enable SSL and configure the SSL encryption algorithm. [Table 10-6](#) lists the parameters. Modify the parameter value as required.

Table 10-6 Parameter Description

Parameter	Example Parameter Value	Description
security.ssl.enabled	true	Enable SSL
akka.ssl.enabled	true	Enable Akka SSL
blob.service.ssl.enabled	true	Enable SSL for blob channel

Parameter	Example Parameter Value	Description
taskmanager.data.ssl.enabled	true	Enable SSL for TaskManagers
security.ssl.algorithms	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Configure the SSL encryption algorithm

 NOTE

Enabling SSL transmission between TaskManagers may pose great impact on system performance.

2. In the bin directory on the Flink client, run the *sh generate_keystore.sh <password>* command.see [Authentication and Encryption](#).The configuration items in [Table 10-7](#) are set by default. You can also configure them manually.

Table 10-7 Parameter Description

Parameter	Example Parameter Value	Description
security.ssl.keystore	\${path}/flink.keystore	Path for storing keystore. flink.keystore indicates the name of the keystore file generated by the generate_keystore.sh* tool.
security.ssl.keystore-password	-	Password of keystore. The indicates the user-defined password.
security.ssl.key-password	-	Password of SSL key. The indicates the user-defined password.
security.ssl.truststore	\${path}/flink.truststore	Path for storing the truststore. flink.truststore indicates the name of the truststore file generated by the generate_keystore.sh* tool.

Parameter	Example Parameter Value	Description
security.ssl.truststore-password	-	Password of truststore, The indicates the user-defined password.

 NOTE

The **path** directory is used to store SSL configuration files of SSL keystore and truststore. This directory is user-defined. Commands for absolute path and relative path are different. For details, see [3](#) and [4](#).

- If the keystore or truststore file path is a relative path, either of the following method can be used to transmit the keystore and truststore file:

- Add **-t** option to the CLI `yarn-session.sh` command. For example:

```
cd /opt/client/Flink/flink  
./bin/yarn-session.sh -t ssl/
```

- Add **-yt** option to the `flink run` command. For example:

```
./bin/flink run -yt ssl/ -ys 3 -m yarn-cluster -c  
com.huawei.SocketWindowWordCount ../lib/flink-eg-1.0.jar --  
hostname r3-d3 --port 9000
```

 NOTE

- The `ssl/` directory is used to store SSL configuration files of SSL keystore and truststore.
- The relative path `ssl/` of current path where the Flink Client command is run must be accessible.

- If the keystore or truststore file path is an absolute path, the keystore or truststore file must exist in the absolute path on Flink Client and all the YARN nodes.

Either of the following methods can be used to run applications. The **-t** or **-yt** option does not need to be added to transmit the keystore and truststore file.

- Run the CLI `yarn-session.sh` command of Flink to execute applications.

```
./bin/yarn-session.sh
```

- Run the `flink run` command to execute applications.

```
./bin/flink run -ys 3 -m yarn-cluster -c  
com.huawei.SocketWindowWordCount ../lib/flink-eg-1.0.jar --  
hostname r3-d3 --port 9000
```

10.3 Developing an Application

10.3.1 DataStream Application

10.3.1.1 Scenarios

Scenarios

Develop a DataStream application of Flink to perform the following operations on logs about dwell durations of netizens for shopping online.

NOTE

The DataStream application can run in both the Windows environment and the Linux environment.

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours in total in a real time manner.
- The first column in the log file records names, the second column records genders, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

log1.txt: logs collected on Saturday. The log file can be obtained from the **data** directory of the sample project.

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

log2.txt: logs collected on Sunday. The log file can be obtained from the **data** directory of the sample project.

```
LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
FangBo,female,50
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
CaiXuyu,female,50
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
FangBo,female,50
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

Data Planning

Data of DataStream sample project is stored in a **.txt** file.

Place the **log1.txt** and **log2.txt** in two directories, for example, **/opt/log1.txt** and **/opt/log2.txt**.

 NOTE

- If the data file is stored in the local file system, the data file must be stored in the specified directory on all nodes where Yarn NodeManager is deployed, and the running user access permission must be set.
- Alternatively, store the data file on HDFS and set the file read path in the program to the HDFS path, for example, **hdfs://hacluster/path/to/file**.

Development Approach

Collect the information about female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

1. Read text data, generate DataStreams, and parse data to generate UserRecord information.
2. Filter the data about the time that female netizens spend online.
3. Perform keyby operation based on the name and gender, and collect the time that female netizens spend online within a time window.
4. Filter data about users whose consecutive online duration exceeds the threshold, and obtain the result.

10.3.1.2 Java Sample Code

Function Description

Collect the information about female netizens who continuously spend more than 2 hour in online shopping and print the result.

DataStream FlinkStreamJavaExampleSample Code

The following code segment is an example. For details, see [com.huawei.bigdata.flink.examples.FlinkStreamJavaExample](#).

```
//Parameter description:
//<filePath> indicates paths where text is read. Paths are separated by commas (,).
//<windowTime> indicates the period covered by statistics window. The unit is minute.
public class FlinkStreamJavaExample {
    public static void main(String[] args) throws Exception {
        //Print reference command for executing flink run.
        System.out.println("use command as: ");
        System.out.println("./bin/flink run --class
com.huawei.bigdata.flink.examples.FlinkStreamJavaExample /opt/test.jar --filePath /opt/log1.txt,/opt/
log2.txt --windowTime 2");
        System.out.println("*****");
        System.out.println("<filePath> is for text file to read data, use comma to separate");
        System.out.println("<windowTime> is the width of the window, time as minutes");
        System.out.println("*****");

        //Paths where text is read. Paths are separated by commas (,)
        final String[] filePaths = ParameterTool.fromArgs(args).get("filePath", "/opt/log1.txt,/opt/
log2.txt").split(",");
        assert filePaths.length > 0;

        //windowTime specifies the size of the time window. By default, a window with 2 minutes as the size
        can read all data in a text file.
        final int windowTime = ParameterTool.fromArgs(args).getInt("windowTime", 2);
```

```
//Build the execution environment and run eventTime to process window data.
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
env.setParallelism(1);

//Read DataStream of the text.
DataStream<String> unionStream = env.readTextFile(filePaths[0]);
if (filePaths.length > 1) {
    for (int i = 1; i < filePaths.length; i++) {
        unionStream = unionStream.union(env.readTextFile(filePaths[i]));
    }
}

//Convert data, build the logic for the entire data process, calculate, and print results.
unionStream.map(new MapFunction<String, UserRecord>() {
    @Override
    public UserRecord map(String value) throws Exception {
        return getRecord(value);
    }
}).assignTimestampsAndWatermarks(
    new Record2TimestampExtractor()
).filter(new FilterFunction<UserRecord>() {
    @Override
    public boolean filter(UserRecord value) throws Exception {
        return value.sexy.equals("female");
    }
}).keyBy(
    new UserRecordSelector()
).window(
    TumblingEventTimeWindows.of(Time.minutes(windowTime))
).reduce(new ReduceFunction<UserRecord>() {
    @Override
    public UserRecord reduce(UserRecord value1, UserRecord value2)
        throws Exception {
        value1.shoppingTime += value2.shoppingTime;
        return value1;
    }
}).filter(new FilterFunction<UserRecord>() {
    @Override
    public boolean filter(UserRecord value) throws Exception {
        return value.shoppingTime > 120;
    }
}).print();

//Call execute to trigger the execution.
env.execute("FemaleInfoCollectionPrint java");
}

//Build the keyword of keyBy as the grouping basis.
private static class UserRecordSelector implements KeySelector<UserRecord, Tuple2<String, String>> {
    @Override
    public Tuple2<String, String> getKey(UserRecord value) throws Exception {
        return Tuple2.of(value.name, value.sexy);
    }
}

//Parse the row data of the text and build UserRecord data structure.
private static UserRecord getRecord(String line) {
    String[] elems = line.split(",");
    assert elems.length == 3;
    return new UserRecord(elems[0], elems[1], Integer.parseInt(elems[2]));
}

//Defines UserRecord data structure and rewrite toString printing method.
public static class UserRecord {
    private String name;
    private String sexy;
    private int shoppingTime;
}
```

```

public UserRecord(String n, String s, int t) {
    name = n;
    sexy = s;
    shoppingTime = t;
}

public String toString() {
    return "name: " + name + " sexy: " + sexy + " shoppingTime: " + shoppingTime;
}
}

//Build class that inherits AssignerWithPunctuatedWatermarks to configure eventTime and waterMark.
private static class Record2TimestampExtractor implements
AssignerWithPunctuatedWatermarks<UserRecord> {

    //add tag in the data of datastream elements
    @Override
    public long extractTimestamp(UserRecord element, long previousTimestamp) {
        return System.currentTimeMillis();
    }

    //give the watermark to trigger the window to execute, and use the value to check if the window
elements are ready
    @Override
    public Watermark checkAndGetNextWatermark(UserRecord element, long extractedTimestamp) {
        return new Watermark(extractedTimestamp - 1);
    }
}
}

```

The following is the command output:

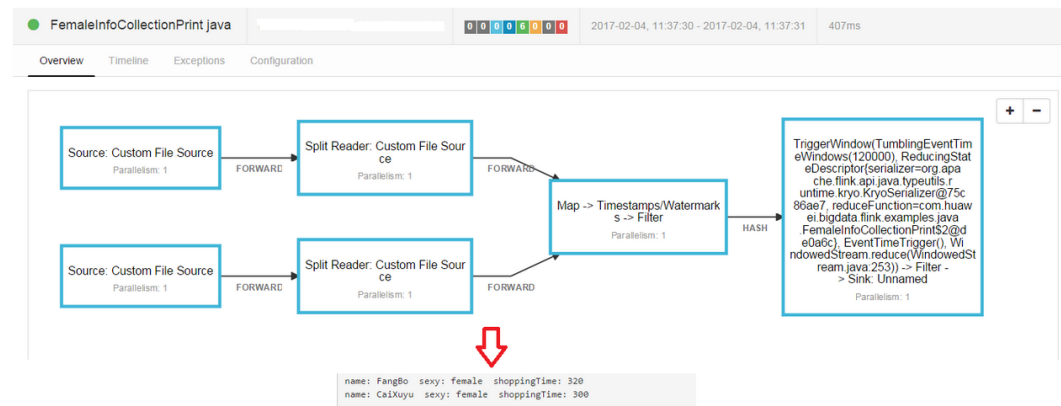
```

name: FangBo sexy: female shoppingTime: 320
name: CaiXuyu sexy: female shoppingTime: 300

```

Figure 10-35 shows the process of execution.

Figure 10-35 Process of execution



10.3.1.3 Scala Sample Code

Function Description

Collect the information about female netizens who continuously spend more than 2 hour in online shopping and print the result.

DataStream FlinkStreamScalaExampleSample Code

The following code is an example. For details, see `com.huawei.bigdata.flink.examples.FlinkStreamScalaExample`.

```
//Parameter description
//filePath indicates paths where text is read. Paths are separated by commas (.).
//windowTime indicates the period covered by statistics window. The unit is minute.
object FlinkStreamScalaExample {
def main(args: Array[String]) {
//Print the reference command used to execute flink run.
System.out.println("use command as: ")
System.out.println("./bin/flink run --class
com.huawei.bigdata.flink.examples.FlinkStreamScalaExample /opt/test.jar --filePath /opt/log1.txt,/opt/
log2.txt --windowTime 2")
System.out.println("*****")
System.out.println("<filePath> is for text file to read data, use comma to separate")
System.out.println("<windowTime> is the width of the window, time as minutes")
System.out.println("*****")

//Paths where text is read. Paths are separated by commas (.)
val filePaths = ParameterTool.fromArgs(args).get("filePath",
"/opt/log1.txt,/opt/log2.txt").split(",").map(_.trim)
assert(filePaths.length > 0)

//windowTime specifies the size of the time window. By default, a window with 2 minutes as the size can
read all data in a text file.
val windowTime = ParameterTool.fromArgs(args).getInt("windowTime", 2)

//Build the execution environment and run eventTime to process window data.
val env = StreamExecutionEnvironment.getExecutionEnvironment
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
env.setParallelism(1)
//Read DataStream of the text.
val unionStream = if (filePaths.length > 1) {
val firstStream = env.readTextFile(filePaths.apply(0))
firstStream.union(filePaths.drop(1).map(it => env.readTextFile(it)): _*)
} else {
env.readTextFile(filePaths.apply(0))
}

//Convert data, build the logic for the entire data process, calculate, and print results.
unionStream.map(getRecord(_))
.assignTimestampsAndWatermarks(new Record2TimestampExtractor)
.filter(_._sexy == "female")
.keyBy("name", "sexy")
.window(TumblingEventTimeWindows.of(Time.minutes(windowTime)))
.reduce((e1, e2) => UserRecord(e1.name, e1.sexy, e1.shoppingTime + e2.shoppingTime))
.filter(_._shoppingTime > 120).print()

//Call execute to trigger the execution
env.execute("FemaleInfoCollectionPrint scala")
}

//Parse the row data of the text and build UserRecord data structure.
def getRecord(line: String): UserRecord = {
val elems = line.split(",")
assert(elems.length == 3)
val name = elems(0)
val sexy = elems(1)
val time = elems(2).toInt
UserRecord(name, sexy, time)
}

//Defines UserRecord data structure.
case class UserRecord(name: String, sexy: String, shoppingTime: Int)

//Build class that inherits AssignerWithPunctuatedWatermarks to configure eventTime and waterMark.
```

```
private class Record2TimestampExtractor extends AssignerWithPunctuatedWatermarks[UserRecord] {
    //add tag in the data of datastream elements
    override def extractTimestamp(element: UserRecord, previousTimestamp: Long): Long = {
        System.currentTimeMillis()
    }

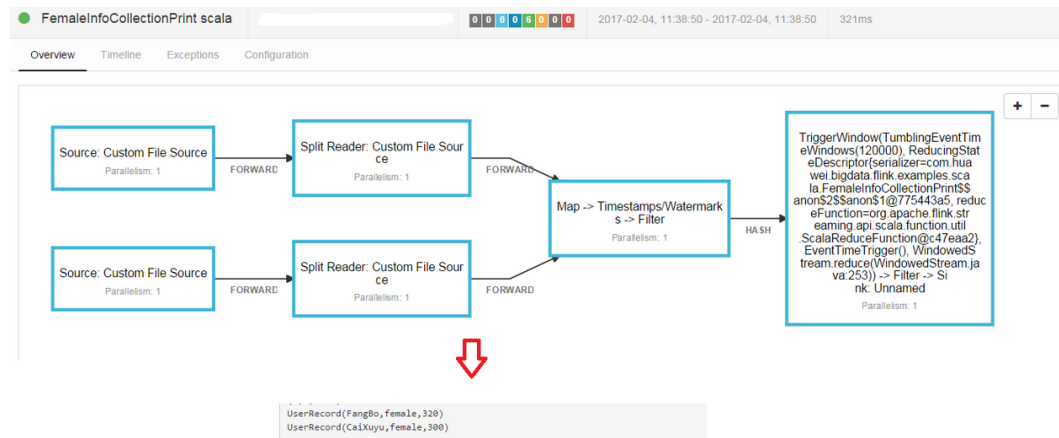
    //give the watermark to trigger the window to execute, and use the value to check if the window
    elements are ready
    def checkAndGetNextWatermark(lastElement: UserRecord,
        extractedTimestamp: Long): Watermark = {
        new Watermark(extractedTimestamp - 1)
    }
}
}
```

The following is the command output:

```
UserRecord(FangBo,female,320)
UserRecord(CaiXuyu,female,300)
```

Figure 10-36 shows the process of execution.

Figure 10-36 Process of execution



10.3.2 Interconnecting with Kafka

10.3.2.1 Scenario Description

Scenario

Assume that a Flink service receives one message record every second.

A Flink application is developed to output prefixed message content in real time based on service requirements.

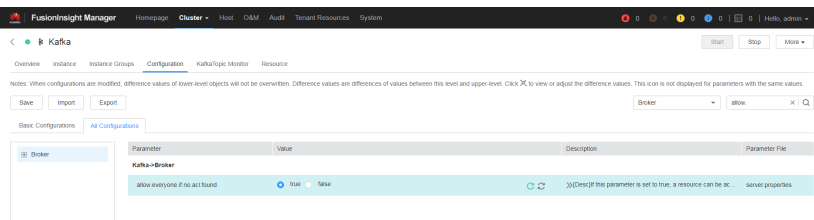
Data Preparation

Flink's sample project data is stored in Kafka. A user with Kafka permission can send data to Kafka and receive data from it.

1. Ensure that clusters have been installed, including HDFS, Yarn, Flink, and Kafka.

2. Create a topic.
 - a. Configure the user permission for creating topics on the server.
Change the value of the **allow.everyone.if.no.acl.found** parameter of Kafka Broker to **true**. See [Figure 10-37](#). After the configuration is complete, restart the Kafka service.

Figure 10-37 Setting the user permission for creating topics



- b. Run a Linux command line to create a topic. Before running a command, run the **kinit** command, for example, **kinit flinkuser**, to authenticate the human-machine account.

NOTE

To create a Flink user, you need to have the permission to create Kafka topics. For details, see [Preparing a Developer Account](#).

The command for creating a Kafka topic is as follows:

```
bin/kafka-topics.sh --create --zookeeper {zkQuorum}/kafka --partitions {partitionNum} --replication-factor {replicationNum} --topic {Topic}
```

Table 10-8 Parameters

Parameter	Description
{zkQuorum}	ZooKeeper cluster information in the <i>IP:Port</i> format
{partitionNum}	Number of topic partitions
{replicationNum}	Number of data replicas of each partition in a topic
{Topic}	Topic name

For example, run the following command on the Kafka client. Assume that the values of IP address:port number for the ZooKeeper cluster are **10.96.101.32:2181**, **10.96.101.251:2181**, **10.96.101.177:2181**, **10.91.8.160:2181**, and the topic name is **topic1**.

```
bin/kafka-topics.sh --create --zookeeper 10.96.101.32:2181,10.96.101.251:2181,10.96.101.177:2181,10.91.8.160:2181/kafka --partitions 5 --replication-factor 1 --topic topic1
```

3. Perform security authentication.
You can use Kerberos authentication, SSL encryption authentication, or Kerberos + SSL authentication.

– **Kerberos authentication**

i. Configure the client.

In Flink configuration file **flink-conf.yaml**, add configurations about Kerberos authentication. For example, add **KafkaClient** in **contexts** as follows:

```
security.kerberos.login.keytab: /home/demo/flink/release/flink-x.x.x/keytab/user.keytab
security.kerberos.login.principal: flinkuser
security.kerberos.login.contexts: Client,KafkaClient
security.kerberos.login.use-ticket-cache: false
```

ii. Run parameters.

Running parameters about the **SASL_PLAINTEXT** protocol are as follows:

```
--topic topic1 --bootstrap.servers 10.96.101.32:21007 --security.protocol SASL_PLAINTEXT
--sasl.kerberos.service.name kafka --kerberos.domain.name hadoop.System domain name.com
```

 **NOTE**

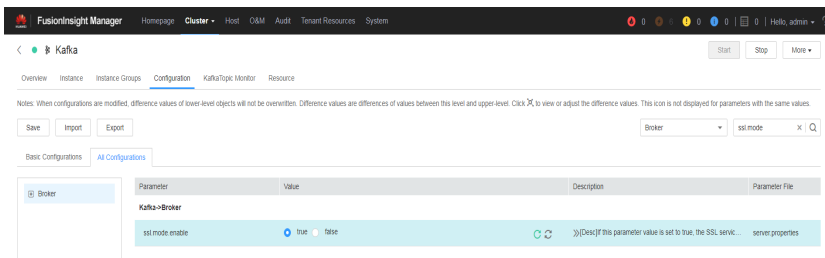
- *10.96.101.32:21007*: IP address and port number of the Kafka server.
- *System domain name*: You can log in to FusionInsight Manager, choose **System > Permission > Domain and Mutual Trust**, and check the value of **Local Domain**.

– **SSL encryption**

■ Configure the server.

Set **ssl.mode.enable** to **true**. See [Figure 10-38](#).

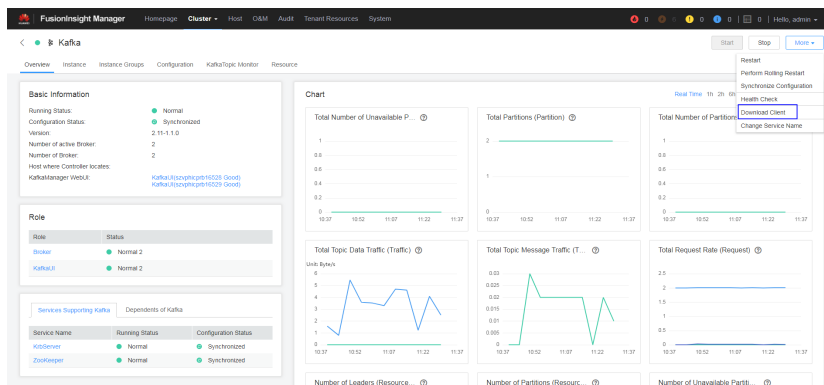
Figure 10-38 Configuring the server



■ Configure the client.

- 1) Log in to FusionInsight Manager and choose **Cluster > Name of the desired cluster > Services > Kafka**. On the page that is displayed, choose **More > Download Client** to download the Kafka client. See [Figure 10-39](#).

Figure 10-39 Configuring the client



- 2) Use the **ca.crt** certificate file in the client root directory to generate the **truststore** file for the client.

Run the following command:

```
keytool -noprompt -import -alias myservcert -file ca.crt -keystore truststore.jks
```

The command output is as follows:

```
drwx-----, 5 zgd users 4096 Feb 4 16:22 .
drwxr-xr-x, 10 zgd users 4096 Jan 22 17:38 ..
-rwx-----, 1 zgd users 135 Jan 22 17:31 application.properties
-rwx-----, 1 zgd users 790 Jan 22 17:31 bigdata_env.sample
-rw-----, 1 zgd users 1322 Jan 22 17:31 ca.crt
-rwx-----, 1 zgd users 4508 Jan 22 17:31 conf.py
-rw-----, 1 zgd users 120 Jan 22 17:31 hosts
-rwx-----, 1 zgd users 745 Jan 22 17:31 install.bat
-rwx-----, 1 zgd users 15082 Jan 22 17:31 install.sh
drwx-----, 2 zgd users 4096 Jan 22 17:38 JDK
-rwx-----, 1 zgd users 37021723 Jan 22 17:31 jython-standalone-2.7.0.jar
drwx-----, 5 zgd users 4096 Jan 22 17:38 Kafka
drwx-----, 3 zgd users 4096 Jan 22 17:38 KrbClient
-rwx-----, 1 zgd users 473 Jan 22 17:31 log4j.properties
-rwx-----, 1 zgd users 2107 Jan 22 17:31 README
-rwx-----, 1 zgd users 6949 Jan 22 17:31 refreshConfig.sh
-rwx-----, 1 zgd users 1736 Jan 22 17:31 switchuser.py
-rw-r--r--, 1 root root 1004 Feb 4 16:22 truststore.jks
```

- 3) Run parameters.

The value of **ssl.truststore.password** must be the same as the password you entered when creating **truststore**. Run the following command to run parameters:

```
--topic topic1 --bootstrap.servers 10.96.101.32:9093 --security.protocol SSL --
ssl.truststore.location /home/zgd/software/FusionInsight_XXX_Kafka_ClientConfig/
truststore.jks --ssl.truststore.password huawei //10.96.101.32:9093 indicates the
IP:Port of the Kafka server. XXX indicates the FusionInsight version.
```

– **Kerberos+SSL encryption**

After completing preceding configurations of the client and server of Kerberos and SSL, modify the port number and protocol type in running parameters to enable the Kerberos+SSL encryption mode.

```
--topic topic1 --bootstrap.servers 10.96.101.32:21009 --security.protocol SASL_SSL --
sasl.kerberos.service.name kafka --ssl.truststore.location --kerberos.domain.name
hadoop.System domain name.com /home/zgd/software/FusionInsight_XXX_Kafka_ClientConfig/
truststore.jks --ssl.truststore.password huawei //10.96.101.32:21009 indicates the IP
address.Port number of the Kafka server, and XXX indicates the FusionInsight version number.
```

Development Guideline

1. Start Flink Kafka Producer to send data to Kafka.
2. Start Flink Kafka Consumer to receive data from Kafka. Ensure that Flink Kafka Consumer and Flink Kafka Producer have the same topics.

3. Add prefixes to data content and print the results.

10.3.2.2 Java Sample Code

Function Description

In a Flink application, call API of the **flink-connector-kafka** module to produce and consume data.

Sample Code

If you want to use FusionInsight in security mode, ensure that the **kafka-clients-*.jar** is obtained from the FusionInsight client directory.

Following is the main logic code of Kafka Consumer and Kafka Producer.

For the complete code, see `com.huawei.bigdata.flink.examples.WriteIntoKafka` and `com.huawei.bigdata.flink.examples.ReadFromKafka`.

```
//producer code
public class WriteIntoKafka {

    public static void main(String[] args) throws Exception {
        //Print the reference command of flink run.

        System.out.println("use command as: ");

        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:9092");

        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21007 --security.protocol
SASL_PLAINTEXT --sas.l.kerberos.service.name kafka");

        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:9093 --security.protocol SSL --
ssl.truststore.location /home/truststore.jks --ssl.truststore.password huawei");

        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21009 --security.protocol SASL_SSL --
sas.l.kerberos.service.name kafka --ssl.truststore.location /home/truststore.jks --ssl.truststore.password
huawei");

        System.out.println
("*****");

        System.out.println("<topic> is the kafka topic name");

        System.out.println("<bootstrap.servers> is the ip:port list of brokers");

        System.out.println
("*****");

        //Build the execution environment.
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        //Configure the parallelism.
        env.setParallelism(1);
        //Parse the execution parameter.
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        //Build the StreamGraph and write data generated by customized source into Kafka.
        DataStream<String> messageStream = env.addSource(new SimpleStringGenerator());
    }
}
```

```

messageStream.addSink(new FlinkKafkaProducer<>(paraTool.get("topic"),
    new SimpleStringSchema(),
    paraTool.getProperties()));
//Call execute to trigger the execution.
env.execute();
}

//Customize source to continuously generate messages every one second.
public static class SimpleStringGenerator implements SourceFunction<String> {

    private static final long serialVersionUID = 2174904787118597072L;

    boolean running = true;

    long i = 0;

    @Override
    public void run(SourceContext<String> ctx) throws Exception {

        while (running) {

            ctx.collect("element-" + (i++));

            Thread.sleep(1000);

        }

    }

    @Override
    public void cancel() {

        running = false;

    }

}

//consumer code
public class ReadFromKafka {

    public static void main(String[] args) throws Exception {
        //Print the reference command of flink run.
        System.out.println("use command as: ");

        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:9092");

        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21007 --security.protocol
SASL_PLAINTEXT --sasL.kerberos.service.name kafka");

        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka" +

```

```

    " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:9093 --security.protocol SSL --
ssl.truststore.location /home/truststore.jks --ssl.truststore.password huawei");

    System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka" +

    " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21009 --security.protocol SASL_SSL --
sasl.kerberos.service.name kafka --ssl.truststore.location /home/truststore.jks --ssl.truststore.password
huawei");

    System.out.println
("*****");

    System.out.println("<topic> is the kafka topic name");

    System.out.println("<bootstrap.servers> is the ip:port list of brokers");

    System.out.println
("*****");
    //Build the execution environment.
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    //Configure the parallelism.
    env.setParallelism(1);
    //Parse the execution parameter.
    ParameterTool paraTool = ParameterTool.fromArgs(args);
    //Build the StreamGraph, read data from Kafka and print the result in another row.
    DataStream<String> messageStream = env.addSource(new
FlinkKafkaConsumer<>(paraTool.get("topic"),

        new SimpleStringSchema(),

        paraTool.getProperties()));

    messageStream.rebalance().map(new MapFunction<String, String>() {

        @Override

        public String map(String s) throws Exception {

            return "Flink says " + s + System.getProperty("line.separator");

        }

    }).print();
    //Call execute to trigger the execution.
    env.execute();

}
}

```

10.3.2.3 Scala Sample Code

Function Description

In a Flink application, call API of the **flink-connector-kafka** module to produce and consume data.

Sample Code

If you want to use FusionInsight in security mode, ensure that the **kafka-clients-*.jar** is obtained from the FusionInsight client directory.

Following is the main logic code of Kafka Consumer and Kafka Producer.

For the complete code, see `com.huawei.bigdata.flink.examples.WriteIntoKafka` and `com.huawei.bigdata.flink.examples.ReadFromKafka`.

```
//Code of producer
object WriteIntoKafka {

  def main(args: Array[String]) {
    //Print the reference command of flink run.
    System.out.println("use command as: ")

    System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
      " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:9092")

    System.out.println
    ("*****")

    System.out.println("<topic> is the kafka topic name")

    System.out.println("<bootstrap.servers> is the ip:port list of brokers")

    System.out.println
    ("*****")
    //Build the execution environment.
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    //Configure the parallelism.
    env.setParallelism(1)
    //Parse the execution parameter.
    val paraTool = ParameterTool.fromArgs(args)
    //Build the StreamGraph and wirte data generated by customized source into Kafka
    val messageStream: DataStream[String] = env.addSource(new SimpleStringGenerator)

    messageStream.addSink(new FlinkKafkaProducer(
      paraTool.get("topic"), new SimpleStringSchema, paraTool.getProperties))
    //Call execute to trigger the execution.
    env.execute
  }
}

//Customize source to continuously generate messages every one second.
class SimpleStringGenerator extends SourceFunction[String] {

  var running = true

  var i = 0

  override def run(ctx: SourceContext[String]) {

    while (running) {

      ctx.collect("element-" + i)

      i += 1

      Thread.sleep(1000)

    }

  }

  override def cancel() {

    running = false

  }

}
```

```

    }
}

//consumer code
object ReadFromKafka {

    def main(args: Array[String]) {
        //Print the reference command of flink run.
        System.out.println("use command as: ")

        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:9092")

        System.out.println
        ("*****")

        System.out.println("<topic> is the kafka topic name")

        System.out.println("<bootstrap.servers> is the ip:port list of brokers")

        System.out.println
        ("*****")

        //Build the execution environment
        val env = StreamExecutionEnvironment.getExecutionEnvironment
        //Configure the parallelism.
        env.setParallelism(1)
        //Parse the execution parameter.
        val paraTool = ParameterTool.fromArgs(args)
        //Build the StreamGraph, read data from Kafka and print the result in another row.
        val messageStream = env.addSource(new FlinkKafkaConsumer(
            paraTool.get("topic"), new SimpleStringSchema, paraTool.getProperties))

        messageStream

            .map(s => "Flink says " + s + System.getProperty("line.separator")).print()
        //Call execute to trigger the execution
        env.execute()
    }
}

```

10.3.3 Asynchronous Checkpoint Mechanism

10.3.3.1 Scenarios

Scenarios

Assume that you want to collect data volume in the window covering preceding 4 seconds at the interval of one second, and achieve strict consistency of status. In this case, when the application is recovered from failure, all operator statuses are the same.

Data Planning

1. Customized operators generate about 10000 pieces of data per second.
2. Generated data is of four tuples (Long, String, String, Integer).

3. Statistic results are printed on the devices.
4. Printed data is of the long type.

Development Approach

1. The source operator sends 10000 pieces of data and injects the data to the window operator every second.
2. The window operator calculates the data volume of preceding 4 seconds at the interval of one second.
3. The statistics is printed to the device at the interval of one second. Please refer to the specific [Viewing the Debugging Result](#)
4. The checkpoint is triggered at the interval of 6 seconds and the checkpoint result is stored in HDFS.

10.3.3.2 Java Sample Code

Function Description

Assume that you want to collect data volume in the window covering preceding 4 seconds at the interval of one second, and achieve strict consistency of status.

Sample Code

1. Snapshot data

The snapshot data is used to store number of data pieces recorded by operators during creation of snapshots.

```
import java.io.Serializable;

//The class is a part of the snapshot and is used to store user-defined statuses.
public class UDFState implements Serializable {
    private long count;

    //Initialize user-defined statuses.
    public UDFState() {
        count = 0L;
    }

    //Configure self-defined statuses.
    public void setState(long count) {
        this.count = count;
    }

    //Obtain user-defined statuses.
    public long geState() {
        return this.count;
    }
}
```

2. Data source with checkpoints

Code of the source operator. The code can be used to send 10000 pieces after each pause of one second. When a snapshot is created, number of sent data pieces is recorded in UDFState. When the snapshot is used for restoration, the number of sent data pieces recorded in UDFState is read and assigned to the count variable.

```
import org.apache.flink.api.java.tuple.Tuple4;
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed;
import org.apache.flink.streaming.api.functions.source.RichSourceFunction;
```

```

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

//The class is the source operator with checkpoint.
public class SEventSourceWithChk extends RichSourceFunction<Tuple4<Long, String, String, Integer>>
implements ListCheckpointed<UDFState> {
    private Long count = 0L;
    private boolean isRunning = true;
    private String alphabet =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyz
xyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321";

    //The main logic of the operator is to inject 10000 tuples to the StreamGraph.
    public void run(SourceContext<Tuple4<Long, String, String, Integer>> ctx) throws Exception {
        Random random = new Random();
        while(isRunning) {
            for (int i = 0; i < 10000; i++) {
                ctx.collect(Tuple4.of(random.nextLong(), "hello-" + count, alphabet, 1))
                count++;
            }
            Thread.sleep(1000);
        }
    }

    //Call this when the task is canceled.
    public void cancel() {
        isRunning = false;
    }

    //Customize a snapshot.
    public List<UDFState> snapshotState(long l, long ll) throws Exception {
        UDFState udfState = new UDFState();
        List<UDFState> listState = new ArrayList<UDFState>();
        udfState.setState(count);
        listState.add(udfState);
        return listState;
    }

    //Restore data from customized snapshots.
    public void restoreState(List<UDFState> list) throws Exception {
        UDFState udfState = list.get(0);
        count = udfState.getState();
    }
}

```

3. Definition of window with checkpoint.

This code is about the window operator and is used to calculate number or tuples in the window.

```

import org.apache.flink.api.java.tuple.Tuple;
import org.apache.flink.api.java.tuple.Tuple4;
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed;
import org.apache.flink.streaming.api.functions.windowing.WindowFunction;
import org.apache.flink.streaming.api.windowing.windows.TimeWindow;
import org.apache.flink.util.Collector;

import java.util.ArrayList;
import java.util.List;

//The class is the window operator with checkpoint.
public class WindowStatisticWithChk implements WindowFunction<Tuple4<Long, String, String,
Integer>, Long, Tuple, TimeWindow>, ListCheckpointed<UDFState> {
    private Long total = 0L;

    //The implementation logic of the window operator, which is used to calculate the number of
tuples in a window.
    void apply(Tuple key, TimeWindow window, Iterable<Tuple4<Long, String, String, Integer>> input,
Collector<Long> out) throws Exception {

```

```
    long count = 0L;
    for (Tuple4<Long, String, String, Integer> event : input) {
        count++;
    }
    total += count;
    out.collect(count);
}

//Customize snapshot.
public List<UDFState> snapshotState(Long l, Long ll) {
    List<UDFState> listState = new ArrayList<UDFState>();
    UDFState udfState = new UDFState();
    udfState.setState(total);
    listState.add(udfState);
    return listState;
}

//Restore data from customized snapshots.
public void restoreState(List<UDFState> list) throws Exception {
    UDFState udfState = list.get(0);
    total = udfState.getState();
}
}
```

4. Application code

The code is about the definition of StreamGraph and is used to implement services. The proctime is used as the timestamp for triggering the window.

```
import org.apache.flink.runtime.state.filesystem.FsStateBackend;
import org.apache.flink.streaming.api.CheckpointingMode;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.windowing.assigners.SlidingProcessingTimeWindows;
import org.apache.flink.streaming.api.windowing.time.Time;

public class FlinkProcessingTimeAPIChekMain {
    public static void main(String[] args) throws Exception{

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        //Set configurations and enable checkpoint.
        env.setStateBackend(new FsStateBackend("hdfs://hacluster/flink/checkpoint/"));
        env.getCheckpointConfig.setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE);
        env.getCheckpointConfig.setCheckpointInterval(6000);

        //Application logic.
        env.addSource(new SEventSourceWithChk())
            .keyBy(0)
            .window(SlidingProcessingTimeWindows.of(Time.seconds(4), Time.seconds(1)))
            .apply(new WindowStatisticWithChk())
            .print()

        env.execute();
    }
}
```

10.3.3.3 Scala Sample Code

Function Description

Assume that you want to collect data volume in the window covering preceding 4 seconds at the interval of one second, and achieve strict consistency of status.

Sample Code

1. Formats of sent data.
case class SEvent(id: Long, name: String, info: String, count: Int)

2. Snapshot data

The snapshot data is used to store number of data pieces recorded by operators during creation of snapshots.

```
//User-defined statuses.

class UDFState extends Serializable{
  private var count = 0L

  //Configure user-defined statuses.
  def setState(s: Long) = count = s

  //Obtain user-defined statuses.
  def getState = count
}
```

3. Data source with checkpoints

Code of the source operator. The code can be used to send 10000 pieces after each pause of one second. When a snapshot is created, number of sent data pieces is recorded in UDFState. When the snapshot is used for restoration, the number of sent data pieces recorded in UDFState is read and assigned to the *count* variable.

```
import java.util
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed
import org.apache.flink.streaming.api.functions.source.RichSourceFunction
import org.apache.flink.streaming.api.functions.source.SourceFunction.SourceContext

//The class is the source operator with checkpoint.
class SEventSourceWithChk extends RichSourceFunction[SEvent] with ListCheckpointed[UDFState]{
  private var count = 0L
  private var isRunning = true
  private val alphabet =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstu
vwxyzABCDEFGHIJKLMNPOQRSTUVWXYZ0987654321"

  //The logic of the operator is to inject 10000 tuples to the StreamGraph.
  override def run(sourceContext: SourceContext[SEvent]): Unit = {
    while(isRunning) {
      for (i <- 0 until 10000) {
        sourceContext.collect(SEvent(1, "hello-"+count, alphabet,1))
        count += 1L
      }
      Thread.sleep(1000)
    }
  }

  //Call this when the task is canceled.
  override def cancel(): Unit = {
    isRunning = false;
  }

  override def close(): Unit = super.close()

  //Create a snapshot.
  override def snapshotState(l: Long, l1: Long): util.List[UDFState] = {
    val udfList: util.ArrayList[UDFState] = new util.ArrayList[UDFState]
    val udfState = new UDFState
    udfState.setState(count)
    udfList.add(udfState)
    udfList
  }

  //Obtain status from the snapshot.
  override def restoreState(list: util.List[UDFState]): Unit = {
    val udfState = list.get(0)
    count = udfState.getState
  }
}
```

```
}  
}
```

4. Definition of window with checkpoint.

This code is about the window operator and is used to calculate number or tuples in the window.

```
import java.util  
import org.apache.flink.api.java.tuple.Tuple  
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed  
import org.apache.flink.streaming.api.scala.function.WindowFunction  
import org.apache.flink.streaming.api.windowing.windows.TimeWindow  
import org.apache.flink.util.Collector  
  
//The class is the window operator with checkpoint.  
class WindowStatisticWithChk extends WindowFunction[SEvent, Long, Tuple, TimeWindow] with  
ListCheckpointed[UDFState]{  
  private var total = 0L  
  
  //The implementation logic of the window operator, which is used to calculate the number of  
  tuples in a window.  
  override def apply(key: Tuple, window: TimeWindow, input: Iterable[SEvent], out: Collector[Long]):  
  Unit = {  
    var count = 0L  
    for (event <- input) {  
      count += 1L  
    }  
    total += count  
    out.collect(count)  
  }  
  
  //Customize a snapshot.  
  override def snapshotState(l: Long, l1: Long): util.List[UDFState] = {  
    val udfList: util.ArrayList[UDFState] = new util.ArrayList[UDFState]  
    val udfState = new UDFState  
    udfState.setState(total)  
    udfList.add(udfState)  
    udfList  
  }  
  
  //Restore data from customized snapshots.  
  override def restoreState(list: util.List[UDFState]): Unit = {  
    val udfState = list.get(0)  
    total = udfState.getState  
  }  
}
```

5. Application code

The code is about the definition of StreamGraph and is used to implement services. The **event time** is used as the timestamp for triggering the window.

```
import com.huawei.rt.flink.core.{SEvent, SEventSourceWithChk, WindowStatisticWithChk}  
import org.apache.flink.contrib.streaming.state.RocksDBStateBackend  
import org.apache.flink.streaming.api.functions.AssignerWithPeriodicWatermarks  
import org.apache.flink.streaming.api.{CheckpointingMode, TimeCharacteristic}  
import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment  
import org.apache.flink.streaming.api.watermark.Watermark  
import org.apache.flink.streaming.api.windowing.assigners.SlidingEventTimeWindows  
import org.apache.flink.streaming.api.windowing.time.Time  
import org.apache.flink.api.scala._  
import org.apache.flink.runtime.state.filesystem.FsStateBackend  
import org.apache.flink.streaming.api.environment.CheckpointConfig.ExternalizedCheckpointCleanup  
  
object FlinkEventTimeAPIChkMain {  
  def main(args: Array[String]): Unit = {  
    val env = StreamExecutionEnvironment.getExecutionEnvironment  
    env.setStateBackend(new FsStateBackend("hdfs://hacluster/flink/checkpoint/"))  
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)  
    env.getConfig.setAutoWatermarkInterval(2000)  
    env.getCheckpointConfig.setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE)
```

```
env.getCheckpointConfig.setCheckpointInterval(6000)

//Application logic.
env.addSource(new SEventSourceWithChk)
  .assignTimestampsAndWatermarks(new AssignerWithPeriodicWatermarks[SEvent] {
    //Configure watermark.
    override def getCurrentWatermark: Watermark = {
      new Watermark(System.currentTimeMillis())
    }
    //Add timestamp for each tuple.
    override def extractTimestamp(t: SEvent, l: Long): Long = {
      System.currentTimeMillis()
    }
  })
  .keyBy(0)
  .window(SlidingEventTimeWindows.of(Time.seconds(4), Time.seconds(1)))
  .apply(new WindowStatisticWithChk)
  .print()
env.execute()
}
```

10.3.4 Job Pipeline Program

10.3.4.1 Scenario

Scenario

Assume that there are three jobs, a publisher and two subscribers. The publisher generates 10000 pieces of data each second. Data is sent by the NettySink operator from the publisher job to downstream subscribers which subscribe a same piece of data.

Data Planning

1. The publisher uses customized operators to generate about 10000 pieces of data per second.
2. The data contains two attributes, which are of integer and string types.
3. Configuration file
 - `nettyconnector.registerserver.topic.storage`: (Mandatory) Configures the path (on a third-party server) to information about IP address, port numbers, and concurrency of NettySink. For example:
`nettyconnector.registerserver.topic.storage: /flink/nettyconnector`
 - `nettyconnector.sinkserver.port.range`: (Mandatory) Configures the range of port numbers of NettySink. For example:
`nettyconnector.sinkserver.port.range: 28444-28943`
 - `nettyconnector.ssl.enabled`: Configures whether to enable SSL encryption between NettySink and NettySource. The default value is false. For example:
`nettyconnector.ssl.enabled: true`
 - `nettyconnector.sinkserver.subnet`: Configure the network domain. For example:
`nettyconnector.sinkserver.subnet: 10.162.0.0/16`
4. Security authentication configuration:

- SASL authentication of ZooKeeper depends on HA-related configurations in **flink-conf.yaml**. For details, see section [Flink Configuration Management](#).
- SSL configurations such as keystore, truststore, keystore password, truststore password, and password inherit from **flink-conf.yaml**. For details, see [Encrypted Transmission](#).

5. Description of APIs

- RegisterServer API

RegisterServerHandler stores information such as IP address, port number, and concurrency of NettySink for the connection with NettySource. Following APIs are provided for users:

```
public interface RegisterServerHandler {
    /**
     * Start the RegisterServer
     *
     * @param The configuration indicates the configuration type of Flink.
     */
    void start(Configuration configuration) throws Exception;
    /**
     * Create a topic node (directory) on the RegisterServer
     *
     * @param The topic indicates the name of the topic node
     */
    void createTopicNode(String topic) throws Exception;
    /**
     * Register the information to a topic node (directory)
     *
     * @param topic @param The topic indicates the directory to be registered with
     * @param The registerRecord indicates the information to be registered
     */
    void register(String topic, RegisterRecord registerRecord) throws Exception;
    /**
     * Delete the topic node.
     *
     * @param The topic indicates the topic to be deleted
     */
    void deleteTopicNode(String topic) throws Exception;
    /**
     * Deregister the registration inDeregister the registration information formation
     * @param The topic indicates the topic where the registration information locates.
     * @param The recordId indicates the ID of the registration information to be deregistered
     */
    void unregister(String topic, int recordId) throws Exception;
    /**
     * Query information
     * @param Topic where the query information locates
     * @param The recordId indicates the ID of the query information
     */
    RegisterRecord query(String topic, int recordId) throws Exception;
    /**
     * Query whether a topic exist.
     * @param topic
     */
    Boolean isExist(String topic) throws Exception;
    /**
     * Disable RegisterServerHandler.
     */
    void shutdown() throws Exception;
}
```

In addition to the preceding APIs, Flink provides ZookeeperRegisterHandler.

– NettySink operator

```
Class NettySink(String name,  
String topic,  
RegisterServerHandler registerServerHandler,  
int numberOfSubscribedJobs)
```

- name: Name of a current NettySink.
- topic: The topic that generates data for the current NettySink. Different NettySinks must use different topics. Otherwise, the subscription may be disordered and data transmission may be abnormal.
- registerServerHandler: Handler of the registration server.
- numberOfSubscribedJobs: Specific number of jobs that subscribe the current NettySink. NettySink sends data only when all subscribers are connected to NettySink.

– NettySource operator

```
Class NettySource(String name,  
String topic,  
RegisterServerHandler registerServerHandler)
```

- name: name of the NettySource. The NettySource must be unique (concurrency excluded). Otherwise, connection with NettySink may be conflicted.
- topic: topic of subscribed NettySink.
- registerServerHandler: Handler of the registration server.

 NOTE

The concurrency of NettySource and the concurrency NettySink must be the same. Otherwise, the connection cannot be created.

Development Approach

1. There are three jobs, on publisher and two subscribers.
2. The publisher transforms generated data into byte[] and sends them the subscribers.
3. After receiving the byte[], subscribers transform data into the string type and print sampled data.

10.3.4.2 Java Sample Code

Following is the main logic code for demonstration.

For details about the complete code, see the following:

- com.huawei.bigdata.flink.examples.UserSource.
- com.huawei.bigdata.flink.examples.TestPipeline_NettySink.
- com.huawei.bigdata.flink.examples.TestPipeline_NettySource1.

- `com.huawei.bigdata.flink.examples.TestPipeline_NettySource2`.

1. The publisher customizes source operators to generate data.

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.streaming.api.functions.source.RichParallelSourceFunction;

import java.io.Serializable;

public class UserSource extends RichParallelSourceFunction<Tuple2<Integer, String>> implements
Serializable {

    private boolean isRunning = true;

    public void open(Configuration configuration) throws Exception {
        super.open(configuration);
    }

    /**
     * Data generation function, which is used to generate 10000 pieces of data each second.
     */
    public void run(SourceContext<Tuple2<Integer, String>> ctx) throws Exception {

        while(isRunning) {
            for (int i = 0; i < 10000; i++) {
                ctx.collect(Tuple2.of(i, "hello-" + i));
            }
            Thread.sleep(1000);
        }
    }

    public void close() {
        isRunning = false;
    }

    public void cancel() {
        isRunning = false;
    }
}
```

2. Code for the publisher:

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.netty.sink.NettySink;
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler;

public class TestPipeline_NettySink {

    public static void main(String[] args) throws Exception{

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        //Set the concurrency of job to 2.
        env.setBufferTimeout(2);

        //Create a ZookeeperRegisterServerHandler.
        ZookeeperRegisterServerHandler zkRegisterServerHandler = new ZookeeperRegisterServerHandler();
        // Add a customized source operator.
        env.addSource(new UserSource())
            .keyBy(0)
            .map(new MapFunction<Tuple2<Integer,String>, byte[]>() {
                //Transform the to-be-sent data into a byte array.
            })

        @Override
        public byte[] map(Tuple2<Integer, String> integerStringTuple2) throws Exception {
```

```
        return integerStringTuple2.f1.getBytes();
    }
    }).addSink(new NettySink("NettySink-1", "TOPIC-2", zkRegisterServerHandler, 2)); //NettySink
transmits the data.

    env.execute();
}
}
```

3. Code for the first subscriber.

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.netty.source.NettySource;
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler;

public class TestPipeline_NettySource1 {

    public static void main(String[] args) throws Exception{

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        // Set the concurrency of job to 2.
        env.setParallelism(2);

        // Create a ZookeeperRegisterServerHandler.
        ZookeeperRegisterServerHandler zkRegisterServerHandler = new ZookeeperRegisterServerHandler();
        //Add a NettySource operator to receive messages from the publisher.
        env.addSource(new NettySource("NettySource-1", "TOPIC-2", zkRegisterServerHandler)
            .map(new MapFunction<byte[], String>() {
                // Transform the received byte stream into character strings
                @Override
                public String map(byte[] b) {
                    return new String(b);
                }
            }).print();

        env.execute();
    }
}
```

4. Code for the second subscriber.

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.netty.source.NettySource;
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler;

public class TestPipeline_NettySource2 {

    public static void main(String[] args) throws Exception {

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        // Set the concurrency of job to 2.
        env.setParallelism(2);

        //Create a ZookeeperRegisterServerHandler.
        ZookeeperRegisterServerHandler zkRegisterServerHandler = new ZookeeperRegisterServerHandler();
        //Add a NettySource operator to receive messages from the publisher.
        env.addSource(new NettySource("NettySource-2", "TOPIC-2", zkRegisterServerHandler)
            .map(new MapFunction<byte[], String>() {
                //Transform the received byte array into character strings.
                @Override
                public String map(byte[] b) {
                    return new String(b);
                }
            })
    }
}
```

```
    }).print();  
    env.execute();  
  }  
}
```

10.3.4.3 Scala Sample Code

Following is the main logic code for demonstration.

For details about the complete code, see the following:

- com.huawei.bigdata.flink.examples.UserSource.
- com.huawei.bigdata.flink.examples.TestPipeline_NettySink.
- com.huawei.bigdata.flink.examples.TestPipeline_NettySource1.
- com.huawei.bigdata.flink.examples.TestPipeline_NettySource2.

1. Code for sending messages:

```
package com.huawei.bigdata.flink.examples  
  
case class Inforamtion(index: Int, content: String) {  
  def this() = this(0, "")  
}
```

2. The publisher customizes source operators to generate data.

```
package com.huawei.bigdata.flink.examples  
  
import org.apache.flink.configuration.Configuration  
import org.apache.flink.streaming.api.functions.source.RichParallelSourceFunction  
import org.apache.flink.streaming.api.functions.source.SourceFunction.SourceContext  
  
class UserSource extends RichParallelSourceFunction[Inforamtion] with Serializable {  
  var isRunning = true  
  
  override def open(parameters: Configuration): Unit = {  
    super.open(parameters)  
  }  
  
  // Generate 10000 pieces of data each second.  
  override def run(sourceContext: SourceContext[Inforamtion]) = {  
    while (isRunning) {  
      for (i <- 0 until 10000) {  
        sourceContext.collect(Inforamtion(i, "hello-" + i));  
      }  
      Thread.sleep(1000)  
    }  
  }  
  
  override def close(): Unit = super.close()  
  
  override def cancel() = {  
    isRunning = false  
  }  
}
```

3. Code for the publisher:

```
package com.huawei.bigdata.flink.examples  
  
import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
```

```
import org.apache.flink.streaming.connectors.netty.sink.NettySink
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler
import org.apache.flink.streaming.api.scala._

object TestPipeline_NettySink {

  def main(args: Array[String]): Unit = {

    val env = StreamExecutionEnvironment.getExecutionEnvironment
    // Set the concurrency of job to 2.
    env.setParallelism(2)
    //Set Zookeeper as the registration server
    val zkRegisterServerHandler = new ZookeeperRegisterServerHandler
    //Add a user-defined operator to generate data.
    env.addSource(new UserSource) .keyBy(0).map(x=>x.content.getBytes)//Transform the to-be-sent data
    into a byte array.
    .addSink(new NettySink("NettySink-1", "TOPIC-2", zkRegisterServerHandler, 2))//Add NettySink
    operator to send data.
    env.execute()
  }
}
```

4. Code for the first subscriber

```
package com.huawei.bigdata.flink.examples

import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
import org.apache.flink.streaming.connectors.netty.source.NettySource
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler
import org.apache.flink.streaming.api.scala._

import scala.util.Random

object TestPipeline_NettySource1 {

  def main(args: Array[String]): Unit = {

    val env = StreamExecutionEnvironment.getExecutionEnvironment
    // Set the concurrency of job to 2.
    env.setParallelism(2)
    //Set ZooKeeper as the registration server
    val zkRegisterServerHandler = new ZookeeperRegisterServerHandler
    //Add a NettySource operator to receive messages from the publisher.
    env.addSource(new NettySource("NettySource-1", "TOPIC-2", zkRegisterServerHandler))
    .map(x => (1, new String(x)))//Add a NettySource operator to receive messages from the publisher.
    .filter(x => {
      Random.nextInt(50000) == 10
    })
    .print

    env.execute()
  }
}
```

5. Code for the second subscriber.

```
package com.huawei.bigdata.flink.examples

import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
import org.apache.flink.streaming.connectors.netty.source.NettySource
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler
import org.apache.flink.streaming.api.scala._

import scala.util.Random

object TestPipeline_NettySource2 {

  def main(args: Array[String]): Unit = {
```

```
val env = StreamExecutionEnvironment.getExecutionEnvironment
//Set the concurrency of job to 2.
env.setParallelism(2)
//Set the concurrency of job to 2.
val zkRegisterServerHandler = new ZookeeperRegisterServerHandler
//Add NettySource operator and receive data.

env.addSource(new NettySource("NettySource-2", "TOPIC-2", zkRegisterServerHandler))
    .map(x=>(2, new String(x)))//Transform the received byte array into character strings.

    .filter(x=>{
        Random.nextInt(50000) == 10
    })
    .print()

env.execute()
}
```

10.3.5 Stream SQL Join Program

10.3.5.1 Scenario

Scenario

Assume that a Flink service (service 1) receives a message every second. The message records the basic information about a user, including the name, gender, and age. Another Flink service (service 2) receives a message irregularly, and the message records the name and career information about the user.

To meet the requirements of some services, the Flink application is developed to achieve the following function: uses the username recorded in the message received by service 2 as the keyword to jointly query two pieces of service data.

Data Planning

The data of service 1 is stored in the Kafka component. Service 1 sends data (requiring Kafka user rights) to and receives data from the Kafka component. For details about how to configure Kafka, see the data planning section of [Data Preparation](#)

Service 2 receives messages using the socket. You can run the **netcat** command to input the analog data source.

- Run the **netcat -l -p <port>** command to start a simple text server.
- After starting the application to connect to the port monitored by **netcat**, enter the data information to the netcat terminal.

Development Approach

1. Start the Flink Kafka Producer application to send data to Kafka.
2. Start the Flink Kafka Consumer application to receive data from Kafka, construct **Table1**, and ensure that the Topic is the same as that of Producer.
3. Read data from the socket and construct **Table2**.
4. Use Flink SQL to query and print **Table1** and **Table2**.

10.3.5.2 Java Sample Code

Function

In the Flink application, this code invokes the flink-connector-kafka module's API to generate and consume data.

Sample Code

If the user needs to use FusionInsight Kafka interconnected with the security mode before the development, obtain the **kafka-clients-*.jar** JAR file from the Kafka client directory.

The following lists Producer, Consumer, and the main logic code used by Flink Stream SQL Join.

For the complete code, see **com.huawei.bigdata.flink.examples.WriteIntoKafka** and **com.huawei.bigdata.flink.examples.SqlJoinWithSocket**.

1. A piece of user information is generated in Kafka every second. The user information includes the name, age, and gender.

```
//Producer code
public class WriteIntoKafka {
    public static void main(String[] args) throws Exception {
        //Print the command reference for flink run.
        System.out.println("use command as: ");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21005");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21007 --security.protocol
SASL_PLAINTEXT --sasl.kerberos.service.name kafka");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21008 --security.protocol SSL --
ssl.truststore.location /home/truststore.jks --ssl.truststore.password huawei");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21009 --security.protocol
SASL_SSL --sasl.kerberos.service.name kafka --ssl.truststore.location /home/truststore.jks --
ssl.truststore.password huawei");
        System.out.println("*****");
        System.out.println("<topic> is the kafka topic name");
        System.out.println("<bootstrap.servers> is the ip:port list of brokers");
        System.out.println("*****");

        //Construct the execution environment.
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        //Set the concurrency.
        env.setParallelism(1);
        //Parse the running parameters.
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        //Construct a flow diagram and write the data generated from self-defined sources to Kafka.
        DataStream<String> messageStream = env.addSource(new SimpleStringGenerator());
        FlinkKafkaProducer<String> producer = new FlinkKafkaProducer<>(paraTool.get("topic"),
            new SimpleStringSchema(),
            paraTool.getProperties());
        producer.setWriteTimestampToKafka(true);
        messageStream.addSink(producer);
        //Invoke execute to trigger the execution.
        env.execute();
    }
}

//Customize the sources and generate a message every second.
public static class SimpleStringGenerator implements SourceFunction<String> {
    static final String[] NAME = {"Carry", "Alen", "Mike", "Ian", "John", "Kobe", "James"};
    static final String[] SEX = {"MALE", "FEMALE"};
```

```

static final int COUNT = NAME.length;
boolean running = true;
Random rand = new Random(47);
@Override
//Use rand to randomly generate a combination of the name, gender, and age.
public void run(SourceContext<String> ctx) throws Exception {
    while (running) {
        int i = rand.nextInt(COUNT);
        int age = rand.nextInt(70);
        String sexy = SEX[rand.nextInt(2)];
        ctx.collect(NAME[i] + "," + age + "," + sexy);
        thread.sleep(1000);
    }
}
@Override
public void cancel() {
    running = false;
}
}
}

```

2. Generate **Table1** and **Table2**, use Join to jointly query **Table1** and **Table2**, and print the output result.

```

public class SqlJoinWithSocket {
    public static void main(String[] args) throws Exception{
        final String hostname;
        final int port;
        System.out.println("use command as: ");
        System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket" +
            " /opt/test.jar --topic topic-test -bootstrap.servers xxx.x.x.x.x:21005 --hostname
xxx.x.x.x.x --port xxx");
        System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket" +
            " /opt/test.jar --topic topic-test -bootstrap.servers xxx.x.x.x.x:21007 --security.protocol
SASL_PLAINTEXT --sasl.kerberos.service.name kafka"
            + "--hostname xxx.x.x.x.x --port xxx");
        System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket" +
            " /opt/test.jar --topic topic-test -bootstrap.servers xxx.x.x.x.x:21008 --security.protocol
SSL --ssl.truststore.location /home/truststore.jks "
            + "--ssl.truststore.password huawei --hostname xxx.x.x.x.x --port xxx");
        System.out.println("*****");
        System.out.println("<topic> is the kafka topic name");
        System.out.println("<bootstrap.servers> is the ip:port list of brokers");
        System.out.println("*****");
        try {
            final ParameterTool params = ParameterTool.fromArgs(args);
            hostname = params.has("hostname") ? params.get("hostname") : "localhost";
            port = params.getInt("port");
        } catch (Exception e) {
            System.err.println("No port specified. Please run 'FlinkStreamSqlJoinExample " +
                "--hostname <hostname> --port <port>', where hostname (localhost by default) " +
                "and port is the address of the text server");
            System.err.println("To start a simple text server, run 'netcat -l -p <port>' and " +
                "type the input text into the command line");
            return;
        }

        EnvironmentSettings fsSettings =
            EnvironmentSettings.newInstance().useOldPlanner().inStreamingMode().build();
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        StreamTableEnvironment tableEnv = StreamTableEnvironment.create(env, fsSettings);
        //Perform processing based on EventTime.
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
        env.setParallelism(1);
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        //Use Stream1 to read data from Kafka.
        DataStream<Tuple3<String, String, String>> kafkaStream = env.addSource(new
            FlinkKafkaConsumer<>(paraTool.get("topic"),
                new SimpleStringSchema(),
                paraTool.getProperties())).map(new MapFunction<String, Tuple3<String, String, String>>()
        {

```

```
@Override
public Tuple3<String, String, String> map(String s) throws Exception {
    String[] word = s.split(",");
    return new Tuple3<>(word[0], word[1], word[2]);
}
});
//Register Stream1 as Table1.
tableEnv.registerDataStream("Table1", kafkaStream, "name, age, sexy, proctime.proctime");
//Use Stream2 to read data from the socket.
DataStream<Tuple2<String, String>> socketStream = env.socketTextStream(hostname, port,
"\n");
map(new MapFunction<String, Tuple2<String, String>>() {
    @Override
    public Tuple2<String, String> map(String s) throws Exception {
        String[] words = s.split("\\s");
        if (words.length < 2) {
            return new Tuple2<>();
        }
        return new Tuple2<>(words[0], words[1]);
    }
});
//Register Stream2 as Table2.
tableEnv.registerDataStream("Table2", socketStream, "name, job, proctime.proctime");
//Run SQL Join to perform a combined query.
Table result = tableEnv.sqlQuery("SELECT t1.name, t1.age, t1.sexy, t2.job, t2.proctime as shiptime
\n" +
    "FROM Table1 AS t1\n" +
    "JOIN Table2 AS t2\n" +
    "ON t1.name = t2.name\n" +
    "AND t1.proctime BETWEEN t2.proctime - INTERVAL '1' SECOND AND t2.proctime +
INTERVAL '1' SECOND");
//Convert the query result into the stream and print the output.
tableEnv.toAppendStream(result, Row.class).print();
env.execute();
}
}
```

10.4 Debugging the Application

10.4.1 Compiling and Running the Application

Scenarios

After the program code is developed, you can upload the code to the Linux client for running. The running procedures of applications developed in Scala or Java are the same.

NOTE

Applications in Flink On YARN mode are allowed to run in a Linux-based environment, but not in a Windows-based environment.

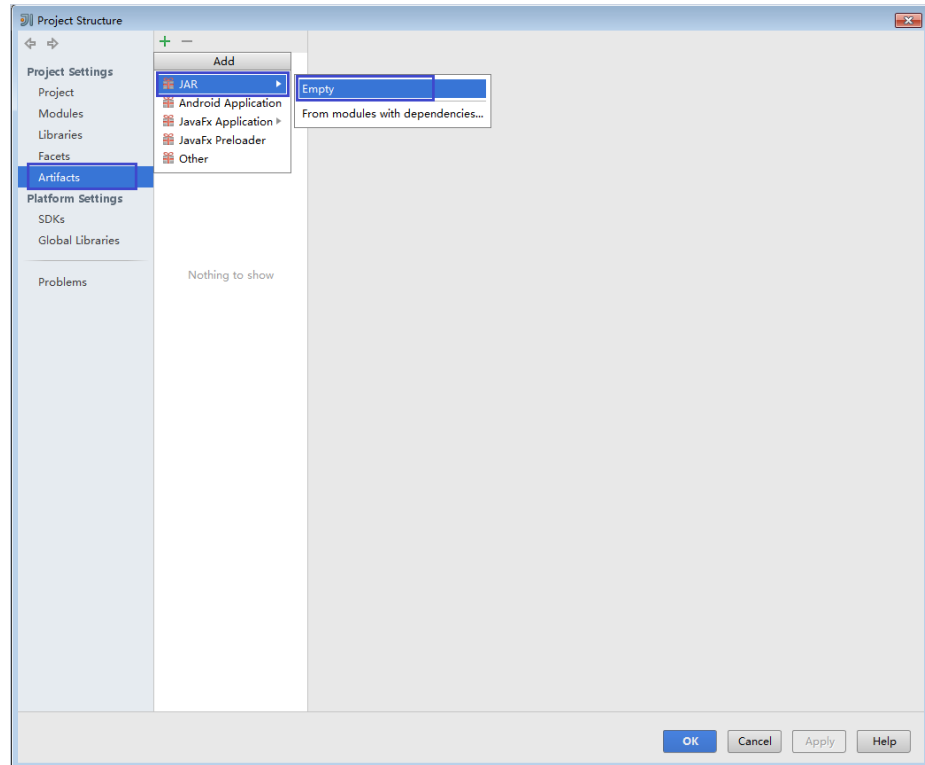
Procedure

Step 1 In IntelliJ IDEA, configure the Artifacts information about the project before the **.jar** package is created.

1. On the main page of IntelliJ IDEA, choose **File >Project Structures...** to access the **Project Structure** page.

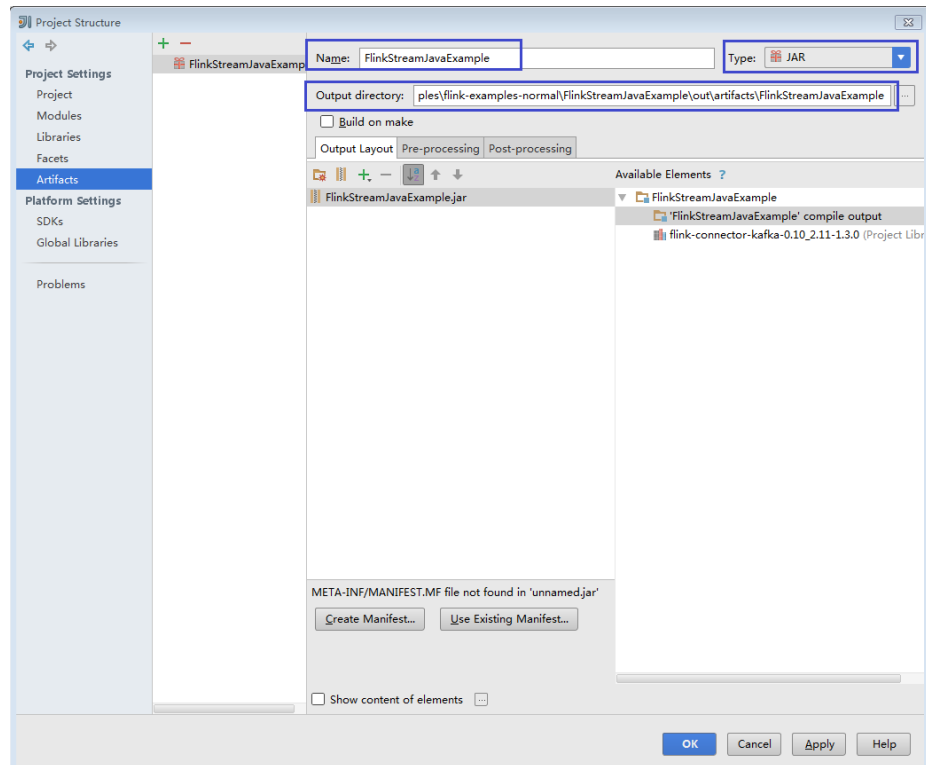
2. On the **Project Structure** page, select **Artifacts**, click the icon "+" and choose **JAR > Empty**

Figure 10-40 Adding Artifacts



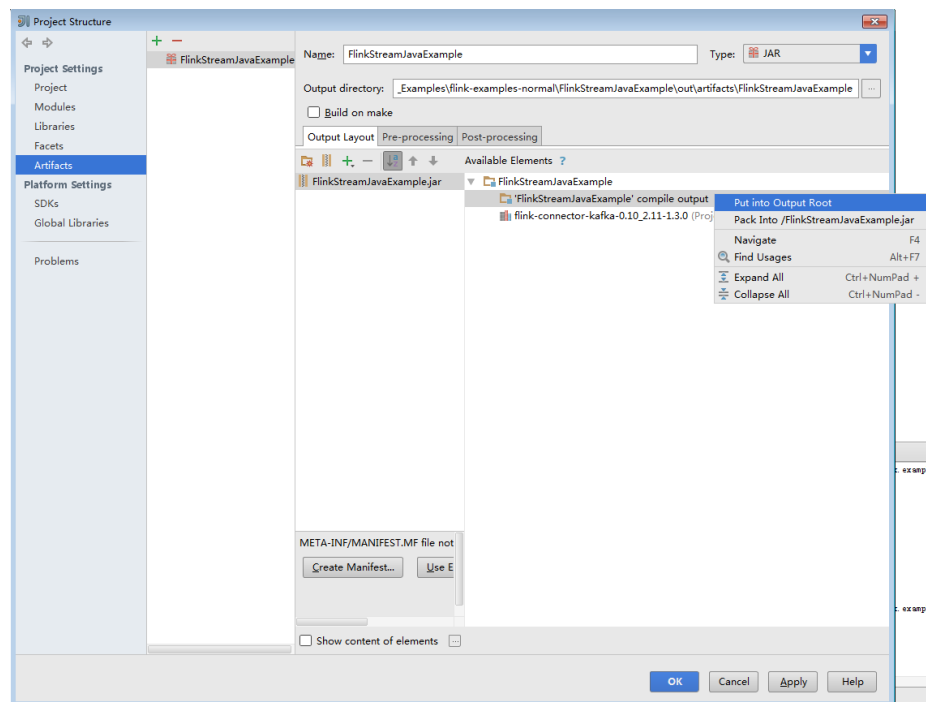
3. Configure the name, type, and output directory of the **.jar** package based on the actual condition.

Figure 10-41 Setting basic information



4. Right-click 'FlinkStreamJavaExample' compile output, choose **Put into Output Root**, and click **Apply**.

Figure 10-42 Choosing Put into Output Root

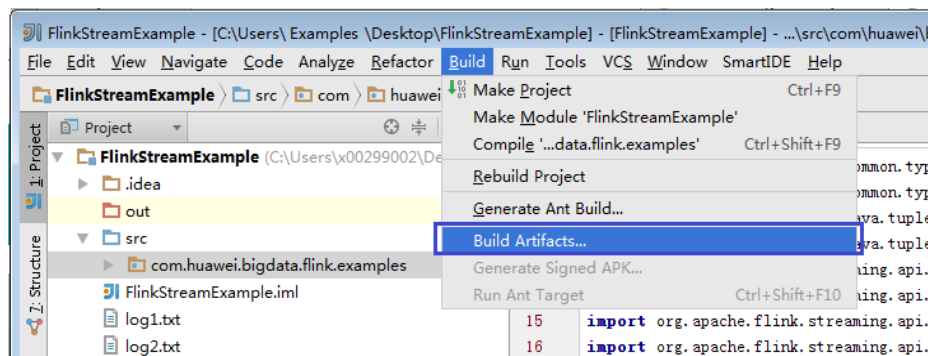


5. Click **OK** to complete the configuration.

Step 2 Generate the **.jar** package.

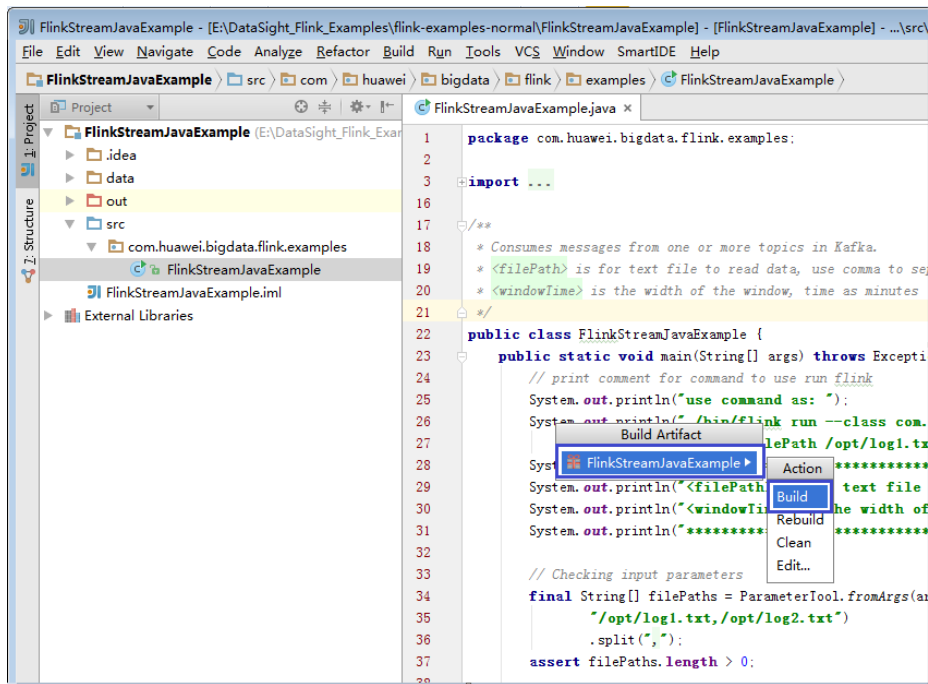
1. On the main page of IntelliJ IDEA, choose **Build > Build Artifacts...**

Figure 10-43 Choosing Build Artifacts



2. On the displayed menu, choose **FlinkStreamJavaExample > Build** to create a jar.

Figure 10-44 Generating the .jar package



3. If the following information is displayed in the event log, the **.jar** package is created successfully. You can obtain the **.jar** package from the directory configured in [Step 1.3](#).

21:25:43 Compilation completed successfully in 36 sec

Step 3 Copy the **.jar** package (for example **FlinkStreamJavaExample.jar**) created in [Step 2](#) to the Flink running environment (Flink client), for example, **/opt/client**, and then in that directory, create the **conf** folder and copy the required configuration files to the **conf** folder. For details, see [Preparing an Operating Environment](#), to run the Flink application.

In a Linux-based environment, Flink cluster needs to be started in advance. Run the **yarn session** command on Flink client and start Flink clusters. For example:

cd /opt/client/Flink/flink

bin/yarn-session.sh -jm 1024 -tm 1024 -t conf/ssl/

 **NOTE**

- Before running the **yarn-session.sh** command, copy the running dependency package of the Flink application to client directory **#{client_install_home}/Flink/flink/lib**. For details about the application running dependency package, see [Reference Information](#).
- When a Flink task is running, do not restart the HDFS service or all DataNode instances. Otherwise, the Flink task may fail, resulting loss of temporary data.
- The **ssl/** directory is used to store SSL configuration files of SSL keystore and truststore.
- Run the DataStream sample program in Scala and Java.

Go to the Flink client directory and call the **bin/flink run** script to run the code.

- Java

```
bin/flink run --class  
com.huawei.bigdata.flink.examples.FlinkStreamJavaExample /opt/  
client/FlinkStreamJavaExample.jar --filePath /opt/log1.txt,/opt/  
log2.txt --windowTime 2
```

- Scala

```
bin/flink run --class  
com.huawei.bigdata.flink.examples.FlinkStreamScalaExample /opt/  
client/FlinkStreamScalaExample.jar --filePath /opt/log1.txt,/opt/  
log2.txt --windowTime 2
```

 **NOTE**

The **log1.txt** and **log2.txt** files must be stored on each node where the Yarn NodeManager instance is deployed and the permission is 755.

Table 10-9 Parameter description

Parameter	Description
<filePath>	File path in the local file system. The /opt/log1.txt and /opt/log2.txt files must be placed on each node. The default value can be retained or changed.
<windowTime>	Duration of the window. The unit is minute. The default value can be retained or changed.

- Run the following code to interconnecting with Kafka (in Scala and Java.)
Execution of the production data command to start the program.

```
bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/client/  
FlinkKafkaJavaExample.jar <topic> <bootstrap.servers> [security.protocol] [sas.l.kerberos.service.name]  
[ssl.truststore.location] [ssl.truststore.password] [kerberos.domain.name]
```

Execution of the consumption data command to start the program.

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/client/  
FlinkKafkaJavaExample.jar <topic> <bootstrap.servers> [security.protocol] [sas.l.kerberos.service.name]  
[ssl.truststore.location] [ssl.truststore.password] [kerberos.domain.name]
```

Table 10-10 Parameter description

Parameter	Description	Mandatory (Yes/No)
topic	The name of a Kafka topic.	Yes
bootstrap.server	The list of IP addresses or ports of broker clusters.	Yes
security.protocol	<p>The parameter need be set to protocols PLAINTEXT (optional), SASL_PLAINTEXT, SSL, and SASL_SSL, and the corresponding FusionInsight Kafka ports are 9092, 21007, 9093, and 21009.</p> <ul style="list-style-type: none"> - If the SASL is configured, the value of sasl.kerberos.service.name must be set to kafka, the value of kerberos.domain.name must be set to kerberos domain name and the security.kerberos.login parameter in conf/flink-conf.yaml are mandatory. - If SSL is configured, ssl.truststore.location (which species the path of truststore) and ssl.truststore.password (which specifies the password of truststore) are mandatory. 	<p>No</p> <p>NOTE</p> <ul style="list-style-type: none"> - If this parameter is not configured, the Kafka is non-secure. - If SSL needs to be configured, see "Kafka Development Guide" > "SSL Encryption Function Used by a Client" to determine the file generation mode of the truststore.jks file.

 NOTE

- To run this example project, set `allow.everyone.if.no.acl.found` to true. For details, see [Configuring Kafka](#).
- Following .jar packages need to be added to Kafka applications:
 - **flink-dist_*.jar** in the **lib** directory under the installation directory of Flink server.
 - **flink-connector-kafka_*.jar** in the **opt** directory under the installation directory of Flink server.
 - **kafka-clients-*.jar** in the **lib** directory under the installation directory of Kafka server or client.
 - If the **truststore.jks** file path is an absolute path, the **truststore.jks** file must be stored in the specified directory of each Yarn nodemanager. If the path is a relative path, add the upload directory when running the **yarn-session.sh** script. For example, run the **yarn-session.sh -t config/ ***** command before running **Command 4**:

Following is the example code corresponding to four protocols, take `ReadFromKafka` as an example:

- Command 1:

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/client/
FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:9092
```

- Command 2:

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/client/
FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:21007 --
security.protocol SASL_PLAINTEXT --sasl.kerberos.service.name kafka --kerberos.domain.name
hadoop.hadoop.com
```

- Command 3:

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/client/
FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:9093 --
security.protocol SSL --ssl.truststore.location /home/truststore.jks --ssl.truststore.password
huawei
```

- Command 4:

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/client/
FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:21009 --
security.protocol SASL_SSL --sasl.kerberos.service.name kafka --ssl.truststore.location /config/
truststore.jks --ssl.truststore.password huawei --kerberos.domain.name hadoop.hadoop.com
```

- Asynchronous checkpoint mechanism (in Scala and Java).

The `proctime` is used as the timestamp for `DataStream` in Java, and the event time is used as the timestamp for `DataStream` in Scala. Following are examples of commands:

- Save the Checkpoint snapshot information to HDFS.

▪ Java

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkProcessingTimeAPIMain /opt/
client/FlinkCheckpointJavaExample.jar --chkPath hdfs://hacluster/flink/checkpoint/
```

▪ Scala

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkEventTimeAPIChkMain /opt/
client/FlinkCheckpointScalaExample.jar --chkPath hdfs://hacluster/flink/checkpoint/
```

- Save the Checkpoint snapshot information to a local path.

▪ Java

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkProcessingTimeAPIMain /opt/
client/FlinkCheckpointJavaExample.jar --chkPath file:///home/zzz/flink-checkpoint/
```

- Scala

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkEventTimeAPIChkMain /opt/
client/FlinkCheckpointScalaExample.jar --ckkPath file:///home/zzz/flink-checkpoint/
```

 NOTE

- Path to checkpoint source file: **flink/checkpoint/checkpoint/fd5f5b3d08628d83038a30302b611/chk-X/4f854bf4-ea54-4595-a9d9-9b9080779ffe**
 - flink/checkpoint** // The specified root directory.
 - fd5f5b3d08628d83038a30302b611** // The second-level directory named after jobID
 - chk-X** // The third-level directory. X indicates the checkpoint numbers.
 - 4f854bf4-ea54-4595-a9d9-9b9080779ffe** // Source files of checkpoint.
- If Flink is in cluster mode, use the HDFS path, because a local path can only be used when Flink is in local mode.
- Run the pipeline sample program.
 - Java
 - i. Start the publisher job.


```
bin/flink run -p 2 --class com.huawei.bigdata.flink.examples.TestPipeline_NettySink /opt/
client/FlinkPipelineJavaExample.jar
```
 - ii. Start the subscriber Job1.


```
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipeline_NettySource1 /opt/
client/FlinkPipelineJavaExample.jar
```
 - iii. Start the subscriber Job2.


```
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipeline_NettySource2 /opt/
client/FlinkPipelineJavaExample.jar
```
 - Scala
 - i. Start the publisher job.


```
bin/flink run -p 2 --class com.huawei.bigdata.flink.examples.TestPipeline_NettySink /opt/
client/FlinkPipelineScalaExample.jar
```
 - ii. Start the subscriber Job1.


```
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipeline_NettySource1 /opt/
client/FlinkPipelineScalaExample.jar
```
 - iii. Start the subscriber Job1.


```
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipeline_NettySource2 /opt/
client/FlinkPipelineScalaExample.jar
```
- Running the Stream SQL Join sample program
 - a. Start the program to generate data for Kafka. For details about Kafka configuration, see [• Run the following code to...](#)

```
bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/client/
FlinkStreamSqlJoinExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.xxx:9092
```
 - b. Run the **netcat** command on any node in the cluster to wait for the connection of the application.

```
netcat -l -p 9000
```
 - c. Start the application to accept the socket data and perform the combined query.

```
bin/flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket /opt/client/
FlinkStreamSqlJoinExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.xxx:9092 --
hostname xxx.xxx.xxx.xxx --port 9000
```

----End

 NOTE

For details about sample projects of dependency packages provided by Flink, see [Reference Information](#).

10.4.2 Viewing the Debugging Result

Scenarios

After a Flink application completes running, you can view the running result, or use Apache Flink Dashboard to view application running status.

Procedure

- **View the running result of the Flink application.**

If you want to check the execution result, view the Stdout log of TaskManager on the Apache Flink Dashboard.

If the execution result is exported to a file or a location specified by Flink, view the result from the exported file or the location. The checkpoint, pipeline, and join between configuration tables and streams are used as examples.

- **View checkpoint results and files**

- The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Checkpoints** to view the submitted jobs as shown in [Figure 10-45](#). Choose **Task Managers > Stdout** to view the running result, as shown in [Figure 10-46](#).

Figure 10-45 submitted jobs

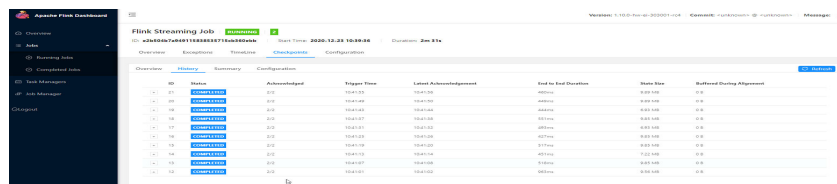
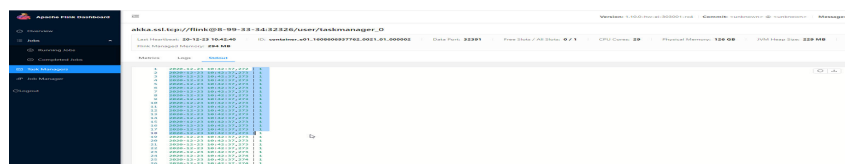


Figure 10-46 execution result



- Either the following methods can be used to view the checkpoint file:
 - If the checkpoint snapshot information is saved in the HDFS, run the **hdfs dfs -ls hdfs://hacluster/flink/checkpoint/** command to view checkpoint files.
 - If the checkpoint snapshot information is saved to a local file, log in to each node to view checkpoint files.

- **View pipeline results**

- The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Running Jobs** to view the running jobs as shown in [Figure 10-47](#). Choose **Task Managers**. You can see two tasks, as shown in [Figure 10-48](#). Click any task, choose **Stdout** to view the output of the task, as shown in [Figure 10-49](#) and [Figure 10-50](#).

Figure 10-47 running jobs

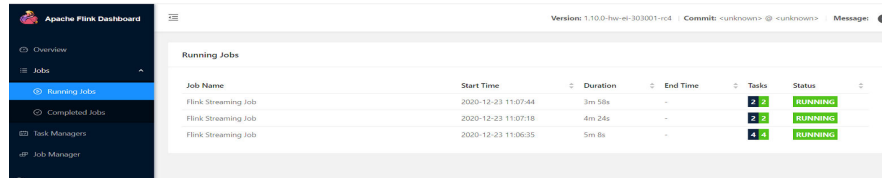


Figure 10-48 submitted Tasks

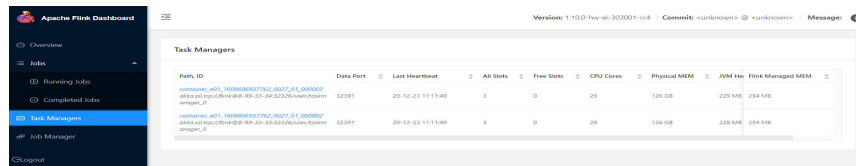


Figure 10-49 output of task1

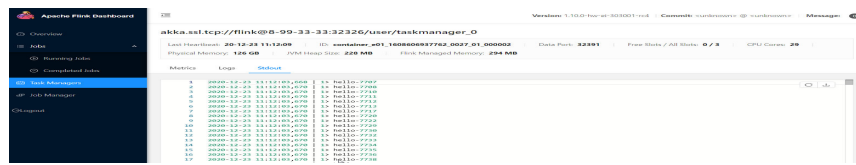
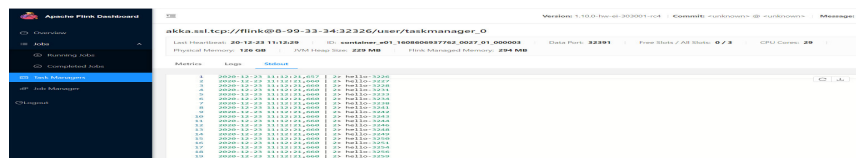


Figure 10-50 output of task2



- View the JOIN result of configuration table and steams**

 - The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Completed Jobs** to view the completed job as shown in [Figure 10-51](#). Choose **Task Managers**, you can see submitted task, as shown in [Figure 10-52](#). Choose **Stdout** to view the running result, as shown in [Figure 10-53](#).

Figure 10-51 completed job

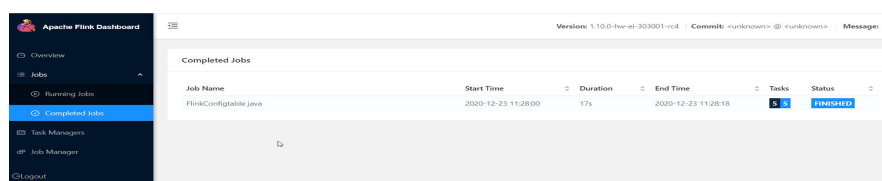


Figure 10-52 submitted task

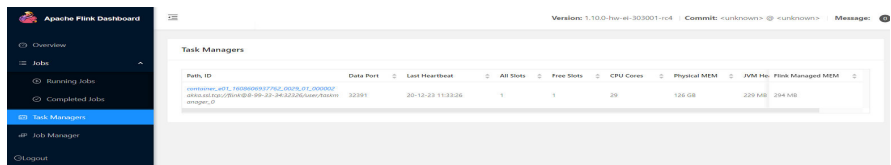


Figure 10-53 execution result



- **View the result of DataStream**
 - The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Completed Jobs** to view the completed job as shown in [Figure 10-54](#). Choose **Task Managers**, you can see submitted task, as shown in [Figure 10-55](#). Choose **Stdout** to view the running result, as shown in [Figure 10-56](#).

Figure 10-54 completed job

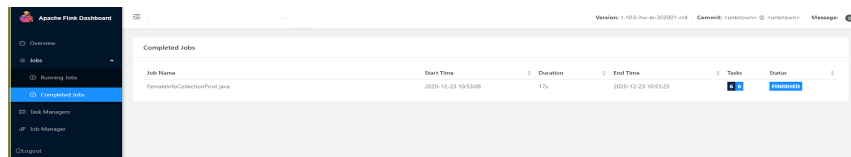


Figure 10-55 submitted task

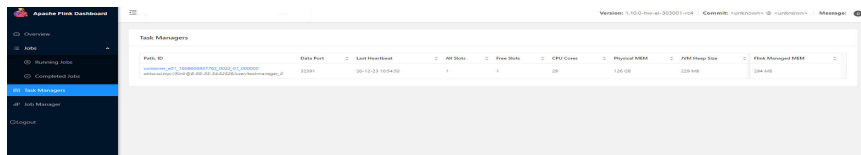
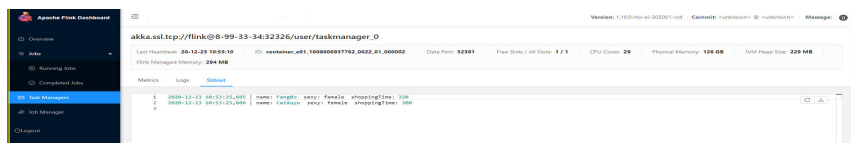


Figure 10-56 execution result



- **View the result of stream sql join**
 - The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Running Jobs** to view the running jobs as shown in [Figure 10-57](#). Choose **Task Managers**, you can see submitted task, as shown in [Figure 10-58](#). Choose **Stdout** to view the running result, as shown in [Figure 10-59](#).

Figure 10-57 running jobs

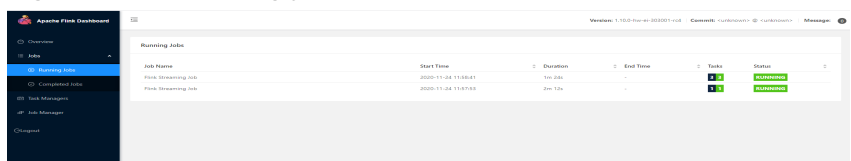


Figure 10-58 submitted task

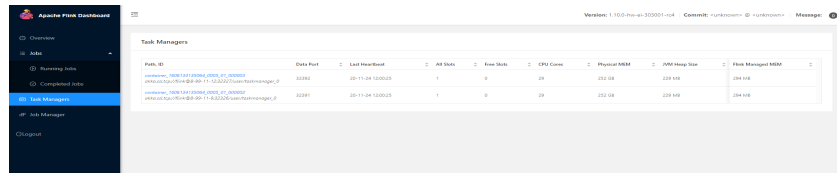
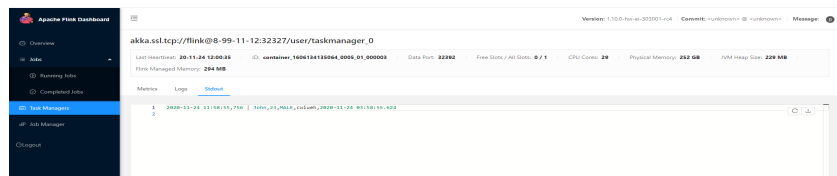


Figure 10-59 execution result



- View the result of produce and consume data in Kafka
 - The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Running Jobs** to view the running jobs as shown in [Figure 10-60](#). Choose **Task Managers**, you can see submitted task, as shown in [Figure 10-61](#). Choose **Stdout** to view the running result, as shown in [Figure 10-62](#).

Figure 10-60 running jobs

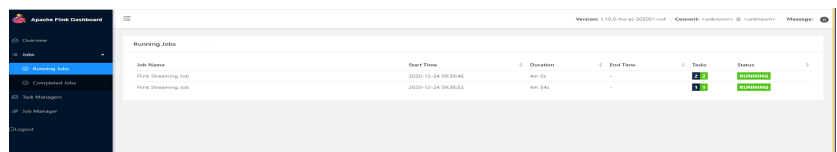
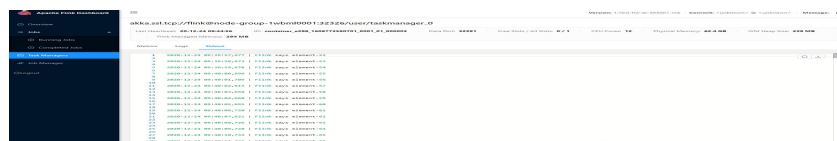


Figure 10-61 submitted task



Figure 10-62 execution result



- Use Apache Flink Dashboard to view the running status of the Flink application.

The Apache Flink Dashboard mainly includes Overview, Running Jobs, Completed Jobs, Task Managers, Job Manager and Logout and so on.

On In the YARN web UI, find the desired Flink application. Click the **ApplicationMaster** at the last column of the application to switch to the Apache Flink Dashboard.

View the print results of the program execution: find the corresponding **Task Manager** to see the corresponding **Stdout** tag log information.

- View Flink logs.

Three methods can be used to obtain Flink logs:

- Log in to the Apache Flink Dashboard and view logs of TaskManagers and JobManager.

- Log in to the YARN web UI to view logs about JobManager and GC.

On the YARN web UI window, find the desired Flink application. Click the **ID** of the application. On the switched page, click **Logs** in the Logs column.

- On the Yarn client, obtain or view logs of Task Managers and Job Manager.

- i. Download and install the Yarn client, for example, in the /opt/client directory.

- ii. Log in to the node where the client is installed as the client installation user.

- iii. Run the following command to switch to the client installation directory:

```
cd /opt/client
```

- iv. Run the following command to configure environment variables:

```
source bigdata_env
```

- v. If the cluster employs the security mode, run the following command to authenticate the user. If the normal mode is used, skip this step.

```
kinit component service user
```

- vi. Run the following commands to obtain container information of the Flink cluster:

```
yarn logs -applicationId application_* -show_application_log_info
```

```
root@hadoop001:~/opt/flinkclient/Flink/flink # yarn logs -applicationId application_1547547065745_0001 -show_application_log_info
WARNING: YARN_CONF_DIR has been replaced by HADOOP_CONF_DIR. Using value of YARN_CONF_DIR.
Application State: Running.
Container: container_1547547065745_0001_01_000001 on hadoop001:26009
Container: container_1547547065745_0001_01_000002 on hadoop001:26009
Container: container_1547547065745_0001_01_000003 on hadoop001:26009
Container: container_1547547065745_0001_01_000004 on hadoop001:26009
```

- vii. Run the following command to obtain run logs of the specified container. Generally, container_*_000001 is the container where the Job Manager is running.

```
yarn logs -applicationId application_* --containerId container_1547547065745_0001_01_000004 -out logdir/
```

```
root@hadoop001:~/opt/flinkclient/Flink/flink/logdir/container_1547547065745_0001_01_000004_26009 # ll
total 172
-rw-r--r-- 1 root root 170605 Jan 17 10:24 container_1547547065745_0001_01_000004
root@hadoop001:~/opt/flinkclient/Flink/flink/logdir/container_1547547065745_0001_01_000004_26009 #
```

After the command is executed, container run logs, including run logs of the Task Manager and Job Manager and GC logs, are downloaded to the local host.

- viii. You can also run a command to obtain the log with the specified name:

The following command is used to obtain the container log list.

```
yarn logs -applicationId application_* -show_container_log_info --containerId container_1547547065745_0001_01_000004
```

```

root@ecs-m3:/opt/flinkclient/Flink/Flink/logdir/ecs-m3-26009 # yarn logs -applicationId application_1547547065745_0001 -sho
WARNING: YARN_CONF_DIR has been replaced by HADOOP_CONF_DIR. Using value of YARN_CONF_DIR.
Container: container_1547547065745_0001_01_000004 on ecs-m3-26009
-----
LogFile                               LogLength                               LastModificationTime                    LogAggregationType
-----
container-localizer-syslog              184                                     Wed Jan 16 17:49:27 +0800 2019      LOCAL
taskmanager.log                         131430                                  Wed Jan 16 17:49:35 +0800 2019      LOCAL
rc-log_0.current                         17952                                    Thu Jan 17 10:22:26 +0800 2019      LOCAL
taskmanager.out                          0                                       Wed Jan 16 17:49:31 +0800 2019      LOCAL
launch_container.sh                     12232                                   Wed Jan 16 17:49:31 +0800 2019      LOCAL
directory.info                           3463                                    Wed Jan 16 17:49:31 +0800 2019      LOCAL
taskmanager.err                          1860                                    Wed Jan 16 17:49:31 +0800 2019      LOCAL
prelaunch.out                            100                                     Wed Jan 16 17:49:31 +0800 2019      LOCAL
prelaunch.err                            0                                       Wed Jan 16 17:49:31 +0800 2019      LOCAL

```

Download **taskmanager.log** to the local host.

```

yarn logs -applicationId application_* --containerId
container_1547547065745_0001_01_000004 -log_files
taskmanager.log -out localpath

```

10.5 More Information

10.5.1 Introduction to Common APIs

10.5.1.1 Java

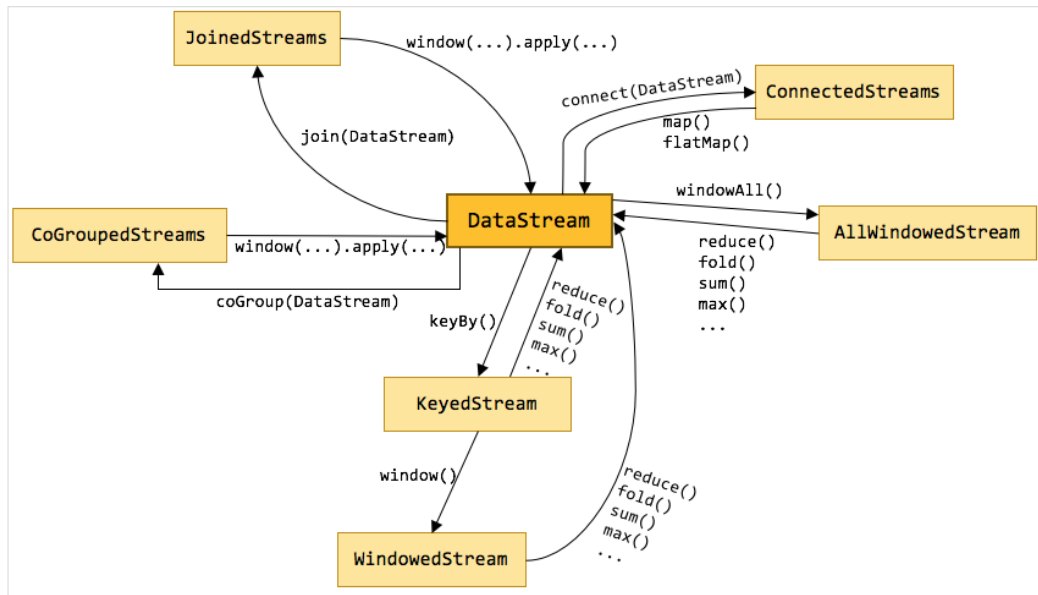
To avoid API compatibility or reliability issues after updates to the open-source Flink, recommended that APIs of the matching version are recommended.

Common APIs of Flink

Flink mainly uses the following APIs:

- **StreamExecutionEnvironment**: provides the execution environment, which is the basis of Flink stream processing.
- **DataStream**: represents streaming data. DataStream can be regarded as unmodifiable collection that contains duplicate data. The number of elements in DataStream are unlimited.
- **KeyedStream**: generates the stream by performing keyBy grouping operations on DataStream. Data is grouped based on the specified key value.
- **WindowedStream**: generates the stream by performing the window function on KeyedStream, sets the window type, and defines window triggering conditions.
- **AllWindowedStream**: generates streams by performing the window function on DataStream, sets the window type, defines window triggering conditions, and performs operations on the window data.
- **ConnectedStreams**: generates streams by connecting the two DataStreams and retaining the original data type, and performs the map or flatMap operation.
- **JoinedStreams**: performs equijoin (which is performed when two values are equal, for example, a.id = b.id) operation to data in the window. The join operation is a special scenario of coGroup operation.
- **CoGroupedStreams**: performs the coGroup operation on data in the window. CoGroupedStreams can perform various types of join operations.

Figure 10-63 Conversion of Flink stream types



Data Stream Source

Table 10-11 APIs about data stream source

API	Description
public final <OUT> DataStreamSource<OUT> fromElements(OUT... data)	Obtain user-defined data of multiple elements as the data stream source. <ul style="list-style-type: none"> • type indicates the data type of an element. • data indicates the data of multiple elements.
public final <OUT> DataStreamSource<OUT> fromElements(Class<OUT> type, OUT... data)	
public <OUT> DataStreamSource<OUT> fromCollection(Collection<OUT> data)	Obtain the user-defined data collection as the data stream source. <ul style="list-style-type: none"> • type indicates the data type of elements in the collection. • typeInfo indicates the type information obtained based on the element data type in the collection. • data indicates the iterator.
public <OUT> DataStreamSource<OUT> fromCollection(Collection<OUT> data, TypeInformation<OUT> typeInfo)	
public <OUT> DataStreamSource<OUT> fromCollection(Iterator<OUT> data, Class<OUT> type)	
public <OUT> DataStreamSource<OUT> fromCollection(Iterator<OUT> data, TypeInformation<OUT> typeInfo)	

API	Description
<pre>public <OUT> DataStreamSource<OUT> fromParallelCollection(SplittableItera- tor<OUT> iterator, Class<OUT> type)</pre>	<p>Obtain the user-defined data collection as parallel data stream source.</p> <ul style="list-style-type: none"> • type indicates the data type of elements in the collection. • typeInfo indicates the type information obtained based on the element data type in the collection. • iterator indicates the iterator that can be divided into multiple partitions.
<pre>public <OUT> DataStreamSource<OUT> fromParallelCollection(SplittableItera- tor<OUT> iterator, TypeInfo<OUT> typeInfo)</pre>	
<pre>private <OUT> DataStreamSource<OUT> fromParallelCollection(SplittableItera- tor<OUT> iterator, TypeInfo<OUT> typeInfo, String operatorName)</pre>	
<pre>public DataStreamSource<Long> generate Sequence(long from, long to)</pre>	<p>Obtain a sequence of user-defined data as the data stream source.</p> <ul style="list-style-type: none"> • from indicates the starting point of numbers. • to indicates the end point of numbers.
<pre>public DataStreamSource<String> readTextFile(String filePath)</pre>	<p>Obtain the user-defined text file from a specific path as the data stream source.</p> <ul style="list-style-type: none"> • filePath indicates the path of the text file. • charsetName indicates the encoding format.
<pre>public DataStreamSource<String> readTextFile(String filePath, String charsetName)</pre>	
<pre>public <OUT> DataStreamSource<OUT> readFile(FileInputformat<OUT> inputformat, String filePath)</pre>	<p>Obtain the user-defined file from a specific path as the data stream source.</p> <ul style="list-style-type: none"> • filePath indicates the file path. • inputformat indicates the format of the file. • watchType indicates the file processing mode, which can be PROCESS_ONCE or PROCESS_CONTINUOUSLY. • interval indicates the interval for processing directories or files.
<pre>public <OUT> DataStreamSource<OUT> readFile(FileInputformat<OUT> inputformat, String filePath, FileProcessingMode watchType, long interval)</pre>	

API	Description
<pre>public <OUT> DataStreamSource<OUT> readFile(FileInputformat<OUT> inputformat, String filePath, FileProcessingMode watchType, long interval, TypeInformation<OUT> typeInformation)</pre>	
<pre>public DataStreamSource<String> socketTextStream(String hostname, int port, String delimiter, long maxRetry)</pre>	<p>Obtain user-defined socket data as the data stream source.</p> <ul style="list-style-type: none"> • hostname indicates the host name of the socket server. • port indicates the listening port of the server. • delimiter indicates the separator of messages. • maxRetry refers to the maximum retry times that can be triggered by abnormal connections.
<pre>public DataStreamSource<String> socketTextStream(String hostname, int port, String delimiter)</pre>	
<pre>public DataStreamSource<String> socketTextStream(String hostname, int port)</pre>	
<pre>public <OUT> DataStreamSource<OUT> addSource(SourceFunction<OUT> function)</pre>	<p>Customize the SourceFunction and addSource methods to add data sources such as Kafka. The implementation method is the run method of SourceFunction.</p> <ul style="list-style-type: none"> • function indicates the user-defined SourceFunction function. • sourceName indicates the name of data source. • typeInfo indicates the type information obtained based on the element data type.
<pre>public <OUT> DataStreamSource<OUT> addSource(SourceFunction<OUT> function, String sourceName)</pre>	
<pre>public <OUT> DataStreamSource<OUT> addSource(SourceFunction<OUT> function, TypeInformation<OUT> typeInfo)</pre>	
<pre>public <OUT> DataStreamSource<OUT> addSource(SourceFunction<OUT> function, String sourceName, TypeInformation<OUT> typeInfo)</pre>	

Data Output

Table 10-12 APIs about data output

API	Description
public DataStreamSink<T> print()	Print data as the standard output stream.
public DataStreamSink<T> printToErr()	Print data as the standard error output stream.
public DataStreamSink<T> writeAsText(String path)	Write data to a specific text file.
public DataStreamSink<T> writeAsText(String path, WriteMode writeMode)	<ul style="list-style-type: none"> • path indicates the path of the text file. • writeMode indicates the writing mode, which can be OVERWRITE or NO_OVERWRITE.
public DataStreamSink<T> writeAsCsv(String path)	Write data to a specific .csv file.
public DataStreamSink<T> writeAsCsv(String path, WriteMode writeMode)	<ul style="list-style-type: none"> • path indicates the path of the text file. • writeMode indicates the writing mode, which can be OVERWRITE or NO_OVERWRITE.
public <X extends Tuple> DataStreamSink<T> writeAsCsv(String path, WriteMode writeMode, String rowDelimiter, String fieldDelimiter)	<ul style="list-style-type: none"> • rowDelimiter indicates the row separator. • fieldDelimiter indicates the column separator.
public DataStreamSink<T> writeToSocket(String hostName, int port, SerializationSchema<T> schema)	Write data to the socket connection. <ul style="list-style-type: none"> • hostName indicates the host name. • port indicates the port number.
public DataStreamSink<T> writeUsingOutputformat(Outputformat<T> format)	Write data to a file, for example, a binary file.
public DataStreamSink<T> addSink(SinkFunction<T> sinkFunction)	Export data in a user-defined manner. The flink-connectors allows the addSink method to support exporting data to Kafka. The implementation method is the invoke method of SinkFunction.

Filtering and Mapping

Table 10-13 APIs about filtering and mapping

API	Description
public <R> SingleOutputStreamOperator<R> map(MapFunction<T, R> mapper)	Transform an element into another element of the same type.
public <R> SingleOutputStreamOperator<R> flatMap(FlatMapFunction<T, R> flatMapper)	Transform an element into zero, one, or multiple elements.
public SingleOutputStreamOperator<T> filter(FilterFunction<T> filter)	Run a Boolean function on each element and retain elements that return true .

Aggregation

Table 10-14 APIs about aggregation

API	Description
public KeyedStream<T, Tuple> keyBy(int... fields)	Logically divide a stream into multiple partitions, each containing elements with the same key. The partitioning is internally implemented using hash partition. A KeyedStream is returned.
public KeyedStream<T, Tuple> keyBy(String... fields)	
public <K> KeyedStream<T, K> keyBy(KeySelector<T, K> key)	Return the KeyedStream after the KeyBy operation, and call the KeyedStream function, such as reduce, fold, min, minby, max, maxby, sum, and sumby. <ul style="list-style-type: none"> • fields indicates the numbers of columns or names of member variables. • key indicates the user-defined basis for partitioning.
public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> reducer)	Perform reduce on KeyedStream in a rolling manner. Aggregate the current element and the last reduced value into a new value. The types of the three values are the same.

API	Description
public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> folder)	Perform fold operation on KeyedStream based on an initial value in a rolling manner. Aggregate the current element and the last folded value into a new value. The input and returned values of Fold can be different.
public SingleOutputStreamOperator<T> sum(int positionToSum)	Calculate the sum in a KeyedStream in a rolling manner. positionToSum and field indicate calculating the sum of a specific column.
public SingleOutputStreamOperator<T> sum(String field)	
public SingleOutputStreamOperator<T> min(int positionToMin)	Calculate the minimum value in a KeyedStream in a rolling manner. min returns the minimum value, without guarantee of the correctness of columns of non-minimum values. positionToMin and field indicate calculating the minimum value of a specific column.
public SingleOutputStreamOperator<T> min(String field)	
public SingleOutputStreamOperator<T> max(int positionToMax)	Calculate the maximum value in KeyedStream in a rolling manner. max returns the maximum value, without guarantee of the correctness of columns of non-maximum values. positionToMax and field indicate calculating the maximum value of a specific column.
public SingleOutputStreamOperator<T> max(String field)	
public SingleOutputStreamOperator<T> minBy(int positionToMinBy)	Obtain the row where the minimum value of a column locates in a KeyedStream. minBy returns all elements of that row. <ul style="list-style-type: none"> • positionToMinBy indicates the column on which the minBy operation is performed. • first indicates whether to return the first or last minimum value.
public SingleOutputStreamOperator<T> minBy(String positionToMinBy)	
public SingleOutputStreamOperator<T> minBy(int positionToMinBy, boolean first)	
public SingleOutputStreamOperator<T> minBy(String field, boolean first)	
public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy)	Obtain the row where the maximum value of a column locates in a KeyedStream. maxBy returns all elements of that row.
public SingleOutputStreamOperator<T> maxBy(String positionToMaxBy)	

API	Description
public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy, boolean first)	<ul style="list-style-type: none"> • positionToMaxBy indicates the column on which the maxBy operation is performed. • first indicates whether to return the first or last maximum value.
public SingleOutputStreamOperator<T> maxBy(String field, boolean first)	

DataStream Distribution

Table 10-15 APIs about DataStream distribution

API	Description
public <K> DataStream<T> partitionCustom(Partitioner<K> partitioner, int field)	<p>Use a user-defined partitioner to select target task for each element.</p> <ul style="list-style-type: none"> • partitioner indicates the user-defined method for repartitioning. • field indicates the input parameters of partitioner. • keySelector indicates the user-defined input parameters of partitioner.
public <K> DataStream<T> partitionCustom(Partitioner<K> partitioner, String field)	
public <K> DataStream<T> partitionCustom(Partitioner<K> partitioner, KeySelector<T, K> keySelector)	
public DataStream<T> shuffle()	Randomly and evenly partition elements.
public DataStream<T> rebalance()	Partition elements in round-robin manner, ensuring load balance of each partition. This partitioning is helpful to data with skewness.
public DataStream<T> rescale()	Distribute elements into downstream subset in round-robin manner.
public DataStream<T> broadcast()	Broadcast each element to all partitions.

Project Capabilities

Table 10-16 APIs about projecting

API	Description
<pre>public <R extends Tuple> SingleOutputStreamOperator<R> project(int... fieldIndexes)</pre>	<p>Select some field subset from the tuple.</p> <p>fieldIndexes indicates some sequences of the tuple.</p> <p>NOTE Only tuple data type is supported by the project API.</p>

Configuring the eventtime Attribute

Table 10-17 APIs about configuring the eventtime attribute

API	Description
<pre>public SingleOutputStreamOperator<T> assignTimestampsAndWatermarks(AssignerWithPeriodicWatermarks<T> timestampAndWatermarkAssigner)</pre>	<p>Extract timestamp from records, enabling event time window to trigger computing.</p>
<pre>public SingleOutputStreamOperator<T> assignTimestampsAndWatermarks(AssignerWithPunctuatedWatermarks<T> timestampAndWatermarkAssigner)</pre>	

Table 10-18 lists differences of `AssignerWithPeriodicWatermarks` and `AssignerWithPunctuatedWatermarks` APIs.

Table 10-18 Difference of AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks APIs

Parameter	Description
AssignerWithPeriodicWatermarks	Generate Watermark based on the getConfig().setAutoWatermarkInterval(200L) timestamp of StreamExecutionEnvironment class.
AssignerWithPunctuatedWatermarks	Generate a Watermark each time an element is received. Watermarks can be different based on received elements.

Iteration

Table 10-19 APIs about iteration

API	Description
public IterativeStream<T> iterate()	In the flow, create in a feedback loop to redirect the output of an operator to a preceding operator.
public IterativeStream<T> iterate(long maxWaitTimeMillis)	<p>NOTE</p> <ul style="list-style-type: none"> This API is helpful to algorithms that require constant update of models. long maxWaitTimeMillis: The timeout period of each round of iteration.

Stream Splitting

Table 10-20 APIs about stream splitting

API	Description
public SplitStream<T> split(OutputSelector<T> outputSelector)	Use OutputSelector to rewrite select method, specify stream splitting basis (by tagging), and construct SplitStream streams. That is, a string is tagged for each element, so that a stream of a specific tag can be selected or created.
public DataStream<OUT> select(String... outputNames)	Select one or multiple streams from a SplitStream. outputNames indicates the sequence of string tags created for each element using the split method.

Window

Windows can be classified as tumbling windows and sliding windows.

- APIs such as Window, TimeWindow, CountWindow, WindowAll, TimeWindowAll, and CountWindowAll can be used to generate windows.
- APIs such as Window Apply, Window Reduce, Window Fold, and Aggregations on windows can be used to operate windows.
- Multiple Window Assigners are supported, including TumblingEventTimeWindows, TumblingProcessingTimeWindows, SlidingEventTimeWindows, SlidingProcessingTimeWindows, EventTimeSessionWindows, ProcessingTimeSessionWindows, and GlobalWindows.
- ProcessingTime, EventTime, and IngestionTime are supported.
- Timestamp modes EventTime: AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks are supported.

Table 10-21 lists APIs for generating windows.

Table 10-21 APIs for generating windows

API	Description
<pre>public <W extends Window> WindowedStream<T, KEY, W> window(WindowAssigner<? super T, W> assigner)</pre>	Define windows in partitioned KeyedStreams. A window groups each key according to some characteristics (for example, data received within the latest 5 seconds).
<pre>public <W extends Window> AllWindowedStream<T, W> windowAll(WindowAssigner<? super T, W> assigner)</pre>	Define windows in DataStreams.
<pre>public WindowedStream<T, KEY, TimeWindow> timeWindow(Time size)</pre>	Define time windows in partitioned KeyedStreams, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters. <ul style="list-style-type: none"> • size indicates the duration of the window. • slide indicates the sliding time of window. NOTE <ul style="list-style-type: none"> • WindowedStream and AllWindowedStream indicates two types of streams. • If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.
<pre>public WindowedStream<T, KEY, TimeWindow> timeWindow(Time size, Time slide)</pre>	

API	Description
public AllWindowedStream<T, TimeWindow> timeWindowAll(Time size)	Define time windows in partitioned DataStream, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters. <ul style="list-style-type: none"> ● size indicates the duration of the window. ● slide indicates the sliding time of window.
public AllWindowedStream<T, TimeWindow> timeWindowAll(Time size, Time slide)	
public WindowedStream<T, KEY, GlobalWindow> countWindow(long size)	Divide windows according to the number of elements and define windows in partitioned KeyedStreams. <ul style="list-style-type: none"> ● size indicates the duration of the window. ● slide indicates the sliding time of window. NOTE <ul style="list-style-type: none"> ● WindowedStream and AllWindowedStream indicates two types of streams. ● If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.
public WindowedStream<T, KEY, GlobalWindow> countWindow(long size, long slide)	
public AllWindowedStream<T, GlobalWindow> countWindowAll(Time size)	Divide windows according to the number of elements and define windows in DataStreams. <ul style="list-style-type: none"> ● size indicates the duration of the window. ● slide indicates the sliding time of window.
public AllWindowedStream<T, GlobalWindow> countWindowAll(Time size, Time slide)	

Table 10-22 lists APIs for operating windows.

Table 10-22 APIs for operating windows

Method	API	Description
Window	public <R> SingleOutputStreamOperator<R> apply(WindowFunction<T, R, K, W> function)	Apply a general function to a window. The data in the window is calculated as a whole. <ul style="list-style-type: none"> ● function indicates the window function to be executed. ● resultType indicates the type of returned data.

Method	API	Description
	<pre>public <R> SingleOutputStreamOperator<R> apply(WindowFunction<T, R, K, W> function, TypeInfo<R> resultType)</pre>	
	<pre>public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> function)</pre>	<p>Apply a reduce function to the window and return the result.</p> <ul style="list-style-type: none"> • reduceFunction indicates the reduce function to be executed. • function of WindowFunction indicates triggering an operation to the window after a reduce operation. • resultType indicates the type of returned data.
	<pre>public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, WindowFunction<T, R, K, W> function)</pre>	
	<pre>public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, WindowFunction<T, R, K, W> function, TypeInfo<R> resultType)</pre>	
	<pre>public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function)</pre>	<p>Apply a fold function to the window and return the result.</p> <ul style="list-style-type: none"> • initialValue indicates the initial value. • foldFunction indicates the fold function. • function of WindowFunction indicates triggering an operation to the window after a fold operation. • resultType indicates the type of returned data.
	<pre>public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function, TypeInfo<R> resultType)</pre>	
	<pre>public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, WindowFunction<ACC, R, K, W> function)</pre>	

Method	API	Description
	<pre>public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, WindowFunction<ACC, R, K, W> function, TypeInformation<ACC> foldAccumulatorType, TypeInformation<R> resultType)</pre>	
Window All	<pre>public <R> SingleOutputStreamOperator<R> apply(AllWindowFunction<T, R, W> function)</pre>	Apply a general function to a window. The data in the window is calculated as a whole.
	<pre>public <R> SingleOutputStreamOperator<R> apply(AllWindowFunction<T, R, W> function, TypeInformation<R> resultType)</pre>	
	<pre>public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> function)</pre>	<p>Apply a reduce function to the window and return the result.</p> <ul style="list-style-type: none"> ● reduceFunction indicates the reduce function to be executed. ● AllWindowFunction indicates triggering an operation to the window after a reduce operation. ● resultType indicates the type of returned data.
	<pre>public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, AllWindowFunction<T, R, W> function)</pre>	
	<pre>public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, AllWindowFunction<T, R, W> function, TypeInformation<R> resultType)</pre>	
<pre>public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function)</pre>	<p>Apply a fold function to the window and return the result.</p> <ul style="list-style-type: none"> ● initialValue indicates the initial value. ● foldFunction indicates the fold function. 	

Method	API	Description
	<pre>public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function, TypeInformation<R> resultType)</pre>	<ul style="list-style-type: none"> • function of WindowFunction indicates triggering an operation to the window after a fold operation. • resultType indicates the type of returned data.
	<pre>public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, AllWindowFunction<ACC, R, W> function)</pre>	
	<pre>public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, AllWindowFunction<ACC, R, W> function, TypeInformation<ACC> foldAccumulatorType, TypeInformation<R> resultType)</pre>	
Window and Window All	<pre>public SingleOutputStreamOperator<T> sum(int positionToSum)</pre>	Sum a specified column of the window data. field and positionToSum indicate a specific column of the data.
	<pre>public SingleOutputStreamOperator<T> sum(String field)</pre>	
	<pre>public SingleOutputStreamOperator<T> min(int positionToMin)</pre>	Calculate the minimum value of a specified column of the window data. min returns the minimum value, without guarantee of the correctness of columns of non-minimum values. positionToMin and field indicate calculating the minimum value of a specific column.
	<pre>public SingleOutputStreamOperator<T> min(String field)</pre>	
	<pre>public SingleOutputStreamOperator<T> minBy(int positionToMinBy)</pre>	Obtain the row where the minimum value of a column locates in the window data. minBy returns all elements of that row.
	<pre>public SingleOutputStreamOperator<T> minBy(String positionToMinBy)</pre>	

Method	API	Description
	public SingleOutputStreamOperator<T> minBy(int positionToMinBy, boolean first)	<ul style="list-style-type: none"> • positionToMinBy indicates the column on which the minBy operation is performed. • first indicates whether to return the first or last minimum value.
	public SingleOutputStreamOperator<T> minBy(String field, boolean first)	
	public SingleOutputStreamOperator<T> max(int positionToMax)	Calculate the maximum value of a specified column of the window data. max returns the maximum value, without guarantee of the correctness of columns of non-maximum values. positionToMax and field indicate calculating the maximum value of a specific column.
	public SingleOutputStreamOperator<T> max(String field)	
	The default public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy)//true	Obtain the row where the maximum value of a column locates in the window data. maxBy returns all elements of that row. <ul style="list-style-type: none"> • positionToMaxBy indicates the column on which the maxBy operation is performed. • first indicates whether to return the first or last maximum value.
	The default public SingleOutputStreamOperator<T> maxBy(String positionToMaxBy)//true	
	public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy, boolean first)	
	public SingleOutputStreamOperator<T> maxBy(String field, boolean first)	
	public SingleOutputStreamOperator<T> maxBy(String field, boolean first)	

Combining Multiple DataStreams

Table 10-23 APIs about combining multiple DataStreams

API	Description
public final DataStream<T> union(DataStream<T>... streams)	Perform union operation on multiple DataStreams to generate a new data stream containing all data from original DataStreams. NOTE If you perform union operation on a piece of data with itself, there are two copies of the same data.

API	Description
<pre>public <R> ConnectedStreams<T, R> connect(DataStream<R> > dataStream)</pre>	Connect two DataStreams and retain the original type. The connect API method allows two DataStreams to share statuses. After ConnectedStreams is generated, map or flatMap operation can be called.
<pre>public <R> SingleOutputStrea- mOperator<R> map(CoMapFunction<I N1, IN2, R> coMapper)</pre>	Perform mapping operation, which is similar to map and flatMap operation in a ConnectedStreams, on elements. After the operation, the type of the new DataStreams is string.
<pre>public <R> SingleOutputStrea- mOperator<R> flatMap(CoFlatMapFun- ction<IN1, IN2, R> coFlatMapper)</pre>	Perform mapping operation, which is similar to map and flatMap operation in a ConnectedStreams, on elements.

Join Operation

Table 10-24 APIs about join operation

API	Description
<pre>public <T2> JoinedStreams<T, T2> join(DataStream<T2> otherStream)</pre>	Join two DataStreams using a given key in a specified window.
<pre>public <T2> CoGroupedStreams<T, T2> coGroup(DataStream< T2> otherStream)</pre>	Co-group two DataStreams using a given key in a specified window.

10.5.1.2 Scala

To avoid API compatibility or reliability issues after updates to the open-source Flink, recommended that APIs of the matching version are recommended.

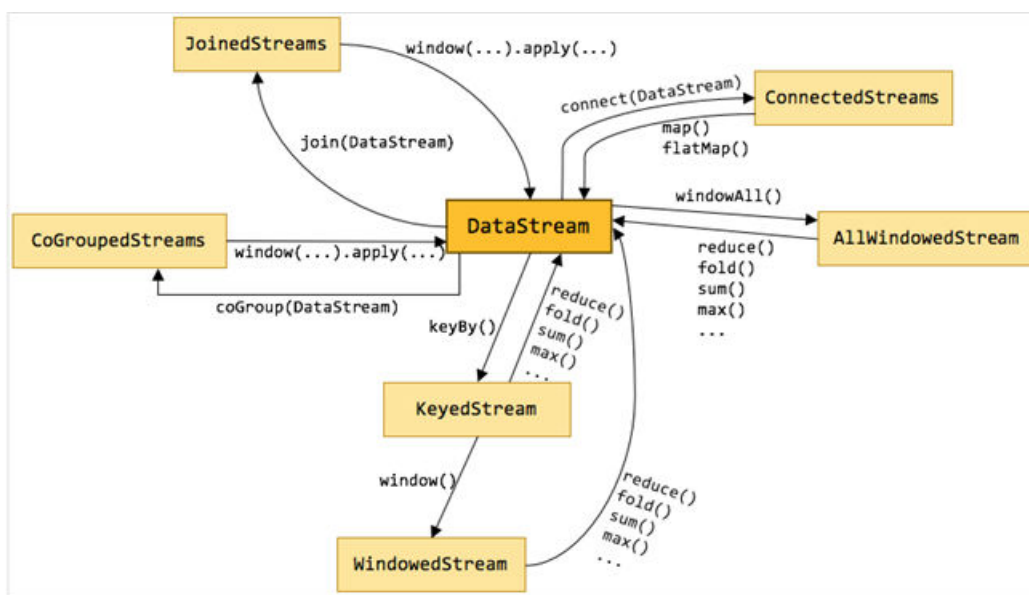
Common APIs of Flink

Flink mainly uses the following APIs:

- **StreamExecutionEnvironment**: provides the execution environment, which is the basis of Flink stream processing.

- **DataStream**: represents streaming data. DataStream can be regarded as unmodifiable collection that contains duplicate data. The number of elements in DataStream are unlimited.
- **KeyedStream**: generates the stream by performing keyBy grouping operations on DataStream. Data is grouped based on the specified key value.
- **WindowedStream**: generates the stream by performing the window function on KeyedStream, sets the window type, and defines window triggering conditions.
- **AllWindowedStream**: generates streams by performing the window function on DataStream, sets the window type, defines window triggering conditions, and performs operations on the window data.
- **ConnectedStreams**: generates streams by connecting the two DataStreams and retaining the original data type, and performs the map or flatMap operation.
- **JoinedStreams**: performs equijoin operation to data in the window. The join operation is a special scenario of coGroup operation.
- **CoGroupedStreams**: performs the coGroup operation on data in the window. CoGroupedStreams can perform various types of join operations.

Figure 10-64 Conversion of Flink stream types



Data Stream Source

Table 10-25 APIs about data stream source

API	Description
def fromElements[T: TypeInformation] (data: T*): DataStream[T]	Obtain user-defined data of multiple elements as the data stream source. data is the specific data of multiple elements.

API	Description
def fromCollection[T: TypeInformation] (data: Seq[T]): DataStream[T]	Obtain the user-defined data collection as input data stream. data can be a data collection or a data body that can be iterated.
def fromCollection[T: TypeInformation] (data: Iterator[T]): DataStream[T]	
def fromParallelCollection[T: TypeInformation] (data: SplittableIterator[T]): DataStream[T]	Obtain the user-defined data collection as parallel data stream source. data indicates the iterator that can be divided into multiple partitions.
def generateSequence(from: Long, to: Long): DataStream[Long]	Obtain a sequence of user-defined data as the data stream source. <ul style="list-style-type: none"> • from indicates the starting point of numbers. • to indicates the end point of numbers.
def readTextFile(filePath: String): DataStream[String]	Obtain the user-defined text file from a specific path as the data stream source. <ul style="list-style-type: none"> • filePath indicates the path of the text file. • charsetName indicates the encoding format.
def readTextFile(filePath: String, charsetName: String): DataStream[String]	
def readFile[T: TypeInformation] (inputFormat: FileInputFormat[T], filePath: String)	Obtain the user-defined file from a specific path as the data stream source. <ul style="list-style-type: none"> • filePath indicates the file path. • inputFormat indicates the format of the file. • watchType indicates the file processing mode, which can be PROCESS_ONCE or PROCESS_CONTINUOUSLY. • interval indicates the interval for processing directories or files.
def readFile[T: TypeInformation] (inputFormat: FileInputFormat[T], filePath: String, watchType: FileProcessingMode, interval: Long): DataStream[T]	
def socketTextStream(hostname: String, port: Int, delimiter: Char = '\n', maxRetry: Long = 0): DataStream[String]	Obtain user-defined socket data as the data stream source. <ul style="list-style-type: none"> • hostname indicates the host name of the socket server. • port indicates the listening port of the server. • delimiter and maxRetry are not supported by Scala APIs.

API	Description
def addSource[T: TypeInformation] (function: SourceFunction[T]): DataStream[T]	Customize the SourceFunction and addSource methods to add data sources such as Kafka. The implementation method is the run method of SourceFunction. <ul style="list-style-type: none"> ● function indicates the user-defined SourceFunction function. ● Simplified format is supported by Scala.
def addSource[T: TypeInformation] (function: SourceContext[T] => Unit): DataStream[T]	

Data Output

Table 10-26 APIs about data output

API	Description
def print(): DataStreamSink[T]	Print data as the standard output stream.
def printToErr()	Print data as the standard error output stream.
def writeAsText(path: String): DataStreamSink[T]	Write data to a specific text file. <ul style="list-style-type: none"> ● path indicates the path of the text file. ● writeMode indicates the writing mode, which can be OVERWRITE or NO_OVERWRITE.
def writeAsText(path: String, writeMode: FileSystem.WriteMode): DataStreamSink[T]	
def writeAsCsv(path: String): DataStreamSink[T]	Write data to a specific .csv file. <ul style="list-style-type: none"> ● path indicates the path of the text file. ● writeMode indicates the writing mode, which can be OVERWRITE or NO_OVERWRITE. ● rowDelimiter indicates the row separator. ● fieldDelimiter indicates the column separator.
def writeAsCsv(path: String, writeMode: FileSystem.WriteMode): DataStreamSink[T]	
def writeAsCsv(path: String, writeMode: FileSystem.WriteMode, rowDelimiter: String, fieldDelimiter: String): DataStreamSink[T]	
def writeUsingOutputFormat(format: OutputFormat[T]): DataStreamSink[T]	Write data to a file, for example, a binary file.
def writeToSocket(hostname: String, port: Integer, schema: SerializationSchema[T]): DataStreamSink[T]	Write data to the socket connection. <ul style="list-style-type: none"> ● hostName indicates the host name. ● port indicates the port number.

API	Description
def addSink(sinkFunction: SinkFunction[T]): DataStreamSink[T]	Export data in a user-defined manner. The flink-connectors allows addSink method to support exporting data to Kafka. The implementation method is the invoke method of SinkFunction.
def addSink(fun: T => Unit): DataStreamSink[T]	

Filtering and Mapping

Table 10-27 APIs about filtering and mapping

API	Description
def map[R: TypeInformation](fun: T => R): DataStream[R]	Transform an element into another element of the same type.
def map[R: TypeInformation](mapper: MapFunction[T, R]): DataStream[R]	
def flatMap[R: TypeInformation](flatMapter: FlatMapFunction[T, R]): DataStream[R]	Transform an element into zero, one, or multiple elements.
def flatMap[R: TypeInformation](fun: (T, Collector[R]) => Unit): DataStream[R]	
def flatMap[R: TypeInformation](fun: T => TraversableOnce[R]): DataStream[R]	
def filter(filter: FilterFunction[T]): DataStream[T]	Run a Boolean function on each element and retain elements that return true .
def filter(fun: T => Boolean): DataStream[T]	

Aggregation

Table 10-28 APIs about aggregation

API	Description
def keyBy(fields: Int*): KeyedStream[T, JavaTuple]	Logically divide a stream into multiple partitions, each containing elements with the same key. The partitioning is internally implemented using hash partition. A KeyedStream is returned.
def keyBy(firstField: String, otherFields: String*): KeyedStream[T, JavaTuple]	

API	Description
def keyBy[K: TypeInformation](fun: T => K): KeyedStream[T, K]	Return the KeyedStream after the KeyBy operation, and call the KeyedStream function, such as reduce, fold, min, minby, max, maxby, sum, and sumby. <ul style="list-style-type: none"> • fields indicates the IDs of certain columns • firstField and otherFields are names of member variables. • key indicates the user-defined basis for partitioning.
def reduce(fun: (T, T) => T): DataStream[T]	Perform reduce on KeyedStream in a rolling manner. Aggregate the current element and the last reduced value into a new value. The types of the three values are the same.
def reduce(reducer: ReduceFunction[T]): DataStream[T]	
def fold[R: TypeInformation](initialValue: R)(fun: (R,T) => R): DataStream[R]	Perform fold operation on KeyedStream based on an initial value in a rolling manner. Aggregate the current element and the last folded value into a new value. The input and returned values of Fold can be different.
def fold[R: TypeInformation](initialValue: R, folder: FoldFunction[T,R]): DataStream[R]	
def sum(position: Int): DataStream[T]	Calculate the sum in a KeyedStream in a rolling manner. position and field indicate calculating the sum of a specific column.
def sum(field: String): DataStream[T]	
def min(position: Int): DataStream[T]	Calculate the minimum value in a KeyedStream in a rolling manner. min returns the minimum value, without guarantee of the correctness of columns of non-minimum values. position and field indicate calculating the minimum value of a specific column.
def min(field: String): DataStream[T]	
def max(position: Int): DataStream[T]	Calculate the maximum value in KeyedStream in a rolling manner. max returns the maximum value, without guarantee of the correctness of columns of non-maximum values. position and field indicate calculating the maximum value of a specific column.
def max(field: String): DataStream[T]	
def minBy(position: Int): DataStream[T]	Obtain the row where the minimum value of a column locates in a

API	Description
def minBy(field: String): DataStream[T]	KeyedStream. minBy returns all elements of that row. position and field indicate the column on which the minBy operation is performed.
def maxBy(position: Int): DataStream[T]	Obtain the row where the maximum value of a column locates in a KeyedStream. maxBy returns all elements of that row. position and field indicate the column on which the maxBy operation is performed.
def maxBy(field: String): DataStream[T]	

DataStream Distribution

Table 10-29 APIs about DataStream distribution

API	Description
def partitionCustom[K: TypeInfo](partitioner: Partitioner[K], field: Int) : DataStream[T]	Use a user-defined partitioner to select target task for each element. <ul style="list-style-type: none"> • partitioner indicates the user-defined method for repartitioning. • field indicates the input parameters of partitioner. • keySelector indicates the user-defined input parameters of partitioner.
def partitionCustom[K: TypeInfo](partitioner: Partitioner[K], field: String):DataStream[T]	
def partitionCustom[K: TypeInfo](partitioner: Partitioner[K], fun: T => K): DataStream[T]	
def shuffle: DataStream[T]	Randomly and evenly partition elements.
def rebalance: DataStream[T]	Partition elements in round-robin manner, ensuring load balance of each partition. This partitioning is helpful to data with skewness.
def rescale: DataStream[T]	Distribute elements into downstream subset in round-robin manner. NOTE The method for checking the code is similar to the rebalance method.

API	Description
def broadcast: DataStream[T]	Broadcast each element to all partitions.

Configuring the eventtime Attribute

Table 10-30 APIs about configuring the eventtime attribute

API	Description
def assignTimestampsAndWatermarks(assigner: AssignerWithPeriodicWatermarks[T]): DataStream[T]	Extract timestamp from records, enabling event time window to trigger computing.
def assignTimestampsAndWatermarks(assigner: AssignerWithPunctuatedWatermarks[T]): DataStream[T]	

Iteration

Table 10-31 APIs about iteration

API	Description
def iterate[R] (stepFunction: DataStream[T] => (DataStream[T], DataStream[R]),maxWaitTimeMillis:Long = 0,keepPartitioning: Boolean = false) : DataStream[R]	In the flow, create in a feedback loop to redirect the output of an operator to a preceding operator. NOTE <ul style="list-style-type: none"> This API is helpful to algorithms that require constant update of models. long maxWaitTimeMillis: The timeout period of each round of iteration.
def iterate[R, F: TypeInformation] (stepFunction: ConnectedStreams[T, F] => (DataStream[F], DataStream[R]),maxWaitTimeMillis:Long): DataStream[R]	

Stream Splitting

Table 10-32 APIs about stream splitting

API	Description
def split(selector: OutputSelector[T]): SplitStream[T]	Use OutputSelector to rewrite select method, specify stream splitting basis (by tagging), and construct SplitStream streams. That is, a string is tagged for each element, so that a stream of a specific tag can be selected or created.
def select(outputNames: String*): DataStream[T]	Select one or multiple streams from a SplitStream. outputNames indicates the sequence of string tags created for each element using the split method.

Window

Windows can be classified as tumbling windows and sliding windows.

- APIs such as Window, TimeWindow, CountWindow, WindowAll, TimeWindowAll, and CountWindowAll can be used to generate windows.
- APIs such as Window Apply, Window Reduce, Window Fold, and Aggregations on windows can be used to operate windows.
- Multiple Window Assigners are supported, including TumblingEventTimeWindows, TumblingProcessingTimeWindows, SlidingEventTimeWindows, SlidingProcessingTimeWindows, EventTimeSessionWindows, ProcessingTimeSessionWindows, and GlobalWindows.
- ProcessingTime, EventTime, and IngestionTime are supported.
- Timestamp modes EventTime: AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks are supported.

[Table9 APIs for generating windows](#) lists APIs for generating windows.

Table 10-33 APIs for generating windows

API	Description
def window[W <: Window](assigner: WindowAssigner[_ >: T, W]): WindowedStream[T, K, W]	Define windows in partitioned KeyedStreams. A window groups each key according to some characteristics (for example, data received within the latest 5 seconds).

API	Description
def windowAll[W <: Window](assigner: WindowAssigner[_ >: T, W]): AllWindowedStream[T, W]	Define windows in DataStreams.
def timeWindow(size: time WindowedStream[T, K, TimeWindow]	Define time windows in partitioned KeyedStreams, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters.
def countWindow(size: Long, slide: Long): WindowedStream[T, K, GlobalWindow]	<ul style="list-style-type: none"> • size indicates the duration of the window. • slide indicates the sliding time of window. <p>NOTE</p> <ul style="list-style-type: none"> • WindowedStream and AllWindowedStream indicates two types of streams. • If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.
def timeWindowAll(size: time AllWindowedStream[T, TimeWindow]	Define time windows in partitioned DataStream, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters.
def timeWindowAll(size: Time, slide: time AllWindowedStream[T, TimeWindow]	<ul style="list-style-type: none"> • size indicates the duration of the window. • slide indicates the sliding time of window.
def countWindow(size: Long, slide: Long): WindowedStream[T, K, GlobalWindow]	Divide windows according to the number of elements and define windows in partitioned KeyedStreams.
def countWindow(size: Long): WindowedStream[T, K, GlobalWindow]	<ul style="list-style-type: none"> • size indicates the duration of the window. • slide indicates the sliding time of window. <p>NOTE</p> <ul style="list-style-type: none"> • WindowedStream and AllWindowedStream indicates two types of streams. • If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.
def countWindowAll(size: Long, slide: Long): AllWindowedStream[T, GlobalWindow]	Divide windows according to the number of elements and define windows in DataStreams.
	<ul style="list-style-type: none"> • size indicates the duration of the window. • slide indicates the sliding time of window.

API	Description
def countWindowAll(size: Long): AllWindowedStream[T, GlobalWindow]	

Table 10-34 lists APIs for operating windows.

Table 10-34 APIs for operating windows

Method	API	Description
Window	def apply[R: TypeInformation] (function: WindowFunction[T, R, K, W]): DataStream[R]	Apply a general function to a window. The data in the window is calculated as a whole.
	def apply[R: TypeInformation] (function: (K, W, Iterable[T], Collector[R]) => Unit): DataStream[R]	function indicates the window function to be executed.
	def reduce(function: ReduceFunction[T]): DataStream[T]	Apply a reduce function to the window and return the result.
	def reduce(function: (T, T) => T): DataStream[T]	<ul style="list-style-type: none"> • reduceFunction indicates the reduce function to be executed. • function of WindowFunction indicates triggering an operation to the window after a reduce operation.
	def reduce[R: TypeInformation] (preAggregator: ReduceFunction[T], function: WindowFunction[T, R, K, W]): DataStream[R]	
	def reduce[R: TypeInformation] (preAggregator: (T, T) => T, windowFunction: (K, W, Iterable[T], Collector[R]) => Unit): DataStream[R]	
	def fold[R: TypeInformation] (initialValue: R, function: FoldFunction[T,R]): DataStream[R]	Apply a fold function to the window and return the result.
def fold[R: TypeInformation] (initialValue: R)(function: (R, T) => R): DataStream[R]	<ul style="list-style-type: none"> • initialValue indicates the initial value. • foldFunction indicates the fold function. 	

Method	API	Description
	<pre>def fold[ACC: TypelInformation, R: TypelInformation](initialValue: ACC, foldFunction: FoldFunction[T, ACC], function: WindowFunction[ACC, R, K, W]): DataStream[R]</pre>	<ul style="list-style-type: none"> • function of WindowFunction indicates triggering an operation to the window after a fold operation.
	<pre>def fold[ACC: TypelInformation, R: TypelInformation](initialValue: ACC, foldFunction: (ACC, T) => ACC, windowFunction: (K, W, Iterable[ACC], Collector[R]) => Unit): DataStream[R]</pre>	
Window All	<pre>def apply[R: TypelInformation] (function: AllWindowFunction[T, R, W]): DataStream[R]</pre>	Apply a general function to a window. The data in the window is calculated as a whole.
	<pre>def apply[R: TypelInformation] (function: (W, Iterable[T], Collector[R]) => Unit): DataStream[R]</pre>	
	<pre>def reduce(function: ReduceFunction[T]): DataStream[T]</pre>	Apply a reduce function to the window and return the result. <ul style="list-style-type: none"> • reduceFunction indicates the reduce function to be executed. • AllWindowFunction indicates triggering an operation to the window after a reduce operation.
	<pre>def reduce(function: (T, T) => T): DataStream[T]</pre>	
	<pre>def reduce[R: TypelInformation] (preAggregator: ReduceFunction[T], windowFunction: AllWindowFunction[T, R, W]): DataStream[R]</pre>	
<pre>def reduce[R: TypelInformation] (preAggregator: (T, T) => T, windowFunction: (W, Iterable[T], Collector[R]) => Unit): DataStream[R]</pre>		

Method	API	Description
	<pre>def fold[R: TypeInformation] (initialValue: R, function: FoldFunction[T,R]): DataStream[R]</pre>	<p>Apply a fold function to the window and return the result.</p> <ul style="list-style-type: none"> • initialValue indicates the initial value. • foldFunction indicates the fold function. • function of WindowFunction indicates triggering an operation to the window after a fold operation.
	<pre>def fold[R: TypeInformation] (initialValue: R)(function: (R, T) => R): DataStream[R]</pre>	
	<pre>def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, preAggregator: FoldFunction[T, ACC], windowFunction: AllWindowFunction[ACC, R, W]): DataStream[R]</pre>	
	<pre>def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, preAggregator: (ACC, T) => ACC, windowFunction: (W, Iterable[ACC], Collector[R]) => Unit): DataStream[R]</pre>	
Window and Window All	<pre>def sum(position: Int): DataStream[T]</pre>	Sum a specified column of the window data.
	<pre>def sum(field: String): DataStream[T]</pre>	field and position indicate a specific column of the data.
	<pre>def min(position: Int): DataStream[T]</pre>	Calculate the minimum value of a specified column of the window data. min returns the minimum value, without guarantee of the correctness of columns of non-minimum values. position and field indicate calculating the minimum value of a specific column.
	<pre>def min(field: String): DataStream[T]</pre>	
	<pre>def max(position: Int): DataStream[T]</pre>	Calculate the maximum value of a specified column of the window data. max returns the maximum value, without guarantee of the correctness of columns of non-maximum values. position and field indicate calculating the maximum value of a specific column.
	<pre>def max(field: String): DataStream[T]</pre>	

Method	API	Description
	def minBy(position: Int): DataStream[T]	Obtain the row where the minimum value of a column locates in the window data. minBy returns all elements of that row. position and field indicate the column on which the minBy operation is performed.
	def minBy(field: String): DataStream[T]	
	def maxBy(position: Int): DataStream[T]	Obtain the row where the maximum value of a column locates in the window data. maxBy returns all elements of that row. position and field indicate the column on which the maxBy operation is performed.
	def maxBy(field: String): DataStream[T]	

Combining Multiple DataStreams

Table 10-35 APIs about combining multiple DataStreams

API	Description
def union(dataStreams: DataStream[T]*): DataStream[T]	Perform union operation on multiple DataStreams to generate a new data stream containing all data from original DataStreams. NOTE If you perform union operation on a piece of data with itself, there are two copies of the same data.
def connect[T2] (dataStream: DataStream[T2]): ConnectedStreams[T, T2]	Connect two DataStreams and retain the original type. The connect API method allows two DataStreams to share statuses. After ConnectedStreams is generated, map or flatMap operation can be called.
def map[R: TypeInformation] (coMapper: CoMapFunction[IN1, IN2, R]): DataStream[R]	Perform mapping operation, which is similar to map and flatMap operation in a ConnectedStreams, on elements. After the operation, the type of the new DataStreams is string.
def map[R: TypeInformation](fun1: IN1 => R, fun2: IN2 => R): DataStream[R]	

API	Description
<pre>def flatMap[R: TypeInformation] (coFlatMapper: CoFlatMapFunction[IN1, IN2, R]): DataStream[R]</pre>	<p>Perform union operation on multiple DataStreams to generate a new data stream containing all data from original DataStreams.</p> <p>NOTE If you perform union operation on a piece of data with itself, there are two copies of the same data.</p>
<pre>def flatMap[R: TypeInformation](fun1: (IN1, Collector[R]) => Unit, fun2: (IN2, Collector[R]) => Unit): DataStream[R]</pre>	
<pre>def flatMap[R: TypeInformation] (fun1: IN1 => TraversableOnce[R], fun2: IN2 => TraversableOnce[R]): DataStream[R]</pre>	

Join Operation

Table 10-36 APIs about join operation

API	Description
<pre>def join[T2] (otherStream: DataStream[T2]): JoinedStreams[T, T2]</pre>	<p>Join two DataStreams using a given key in a specified window.</p> <p>The key value of the join operation is specified by the where and equalTo method, indicating filtering data with equivalent conditions from two DataStreams.</p>
<pre>def coGroup[T2] (otherStream: DataStream[T2]): CoGroupedStreams[T, T2]</pre>	<p>Co-group two DataStreams using a given key in a specified window.</p> <p>The key value of the coGroup operation is specified by the where and equalTo method, indicating partitioning two DataStreams using equivalent conditions.</p>

10.5.2 Overview of RESTful APIs

Flink has a monitoring API that can be used to query status and statistics of running jobs, as well as recent completed jobs. This monitoring API is used by Apache Flink Dashboard.

The monitoring API is a RESTful API that accepts HTTP GET requests and responds with JSON data. RESTful API is a set of APIs used to log in to the web server. In

Flink, web server is a module of JobManager and shares the same process with JobManager. By default, the listening port of web server is 8081. If you want to change the listening port, modify **jobmanager.web.port** in the **flink-conf.yaml** file.

The *Netty* and the *Netty Router* library are used to handle REST requests and analysis URLs.

RESTful APIs are executed through HTTP requests.

The format of the HTTP requests is `http://<JobManager_IP>:<JobManager_Port><Path>`

The *JobManager_IP* indicates the IP address of JobManager, *JobManager_Port* indicates the listening port of JobManager, and *Path* indicates the path. For details, see [Table 10-37](#). For example, `http://10.162.181.57:32261/config`.

 **NOTE**

If you want to modify the configuration file **flink-conf.yaml** of the Flink Client, add to-be-visited IP addresses (separated with commas) in **jobmanager.web.allow-access-address** and **jobmanager.web.access-control-allow-origin** parameters.

[Table 10-37](#) lists all RESTful API paths supported by Flink.

Table 10-37 Paths supported by Flink

Path	Description
/config	Some information about the monitoring API and the server setup.
/logout	Some information about the logout.
/overview	Simple summary of the Flink cluster status.
/jobs	IDs of the jobs, grouped by status <i>running</i> , <i>finished</i> , <i>failed</i> , <i>canceled</i> .
/jobmanager/config	the configuration of the jobmanager.
/joboverview	Jobs, grouped by status, each with a small summary of its status.
/joboverview/running	Jobs, grouped by status, each with a small summary of its status. The same as /joboverview , but containing only currently running jobs.
/joboverview/completed	Jobs, grouped by status, each with a small summary of its status. The same as /joboverview , but containing only completed (finished, canceled, or failed) jobs.

Path	Description
/jobs/<jobid>	Summary of one job, listing dataflow plan, status, timestamps of state transitions, aggregate information for each vertex (operator).
/jobs/<jobid>/vertices	Currently the same as /jobs/<jobid> .
/jobs/<jobid>/config	The user-defined execution config used by the job.
/jobs/<jobid>/exceptions	The non-recoverable exceptions that have been observed by the job. The truncated flag defines whether more exceptions occurred, but are not listed, because the response would otherwise get too big.
/jobs/<jobid>/accumulators	The aggregated user accumulators plus job accumulators.
/jobs/<jobid>/checkpoints	checkpoint stats for a job.
/jobs/<jobid>/metrics	a job a list of all available metrics.
/jobs/<jobid>/vertices/<vertexid>	Information about one specific vertex, with a summary for each of its subtasks.
/jobs/<jobid>/vertices/<vertexid>/subtasktimes	This request returns the timestamps for the state transitions of all subtasks of a given vertex. These can be used, for example, to create time-line comparisons between subtasks.
/jobs/<jobid>/vertices/<vertexid>/taskmanagers	TaskManager statistics for one specific vertex. This is an aggregation of subtask statistics returned by /jobs/<jobid>/vertices/<vertexid> .
/jobs/<jobid>/vertices/<vertexid>/accumulators	The aggregated user-defined accumulators, for a specific vertex.
/jobs/<jobid>/vertices/<vertexid>/checkpoints	checkpoint stats for a single job vertex.
/jobs/<jobid>/vertices/<vertexid>/backpressure	back pressure stats for a single job vertex and all its sub tasks.
/jobs/<jobid>/vertices/<vertexid>/metrics	a given task of the values for a set of metrics.

Path	Description
/jobs/<jobid>/vertices/<vertexid>/subtasks/accumulators	Gets all user-defined accumulators for all subtasks of a given vertex. These are the individual accumulators that are returned in aggregated form by the request /jobs/<jobid>/vertices/<vertexid>/accumulators .
/jobs/<jobid>/vertices/<vertexid>/subtasks/<subtasknum>	Summary of the current or latest execution attempt of a specific subtask. See below for a sample.
/jobs/<jobid>/vertices/<vertexid>/subtasks/<subtasknum>/attempts/<attempt>	Summary of a specific execution attempt of a specific subtask. Multiple execution attempts happen in case of failure/recovery.
/jobs/<jobid>/vertices/<vertexid>/subtasks/<subtasknum>/attempts/<attempt>/accumulators	The accumulators collected for one specific subtask during one specific execution attempt (multiple attempts happen in case of failure/recovery).
/jobs/<jobid>/plan	The dataflow plan of a job. The plan is also included in the job summary (/jobs/<jobid>).
/taskmanagers	Some information of the TaskManagers.
/taskmanagers/<taskmanagerid>/metrics	the metrics information of a TaskManager.
/taskmanagers/<taskmanagerid>/log	the log information of a TaskManager.
/taskmanagers/<taskmanagerid>/stdout	the stdout of a TaskManager.
/jobmanager/log	the log of Jobmanager.
/jobmanager/stdout	the stdout of Jobmanager.
/jobmanager/metrics	the metrics of Jobmanager.
/*	services requests to web frontend's static files, such as HTML, CSS, or JS files.

Table 10-38 describes variables listed in **Table 10-37**.

Table 10-38 Description of variables

Variable	Description
jobid	ID of jobs
vertexid	Vertexes ID of the flow diagram.
subtasknum	Sum of subtasks.
attempt	Times of attempts
taskmanagerid	ID of TaskManager

10.5.3 Overview of Savepoints CLI

Overview

Savepoints are externally stored checkpoints that you can use to stop-and-resume or update your Flink programs. They use Flink's checkpoint mechanism to create a snapshot of the state of your streaming program and write the checkpoint meta data out to an external file system.

It is highly recommended that you adjust your programs as described in this section in order to be able to upgrade your programs in the future. The main required change is to manually specify operator IDs via the **uid(String)** method. These IDs are used to scope the state of each operator.

```
DataStream<String> stream = env
//Statefulsource(e.g.Kafka)withID
.addSource(new StatefulSource())
.uid("source-id") //IDforthesourceoperator
.shuffle()
//StatefulmapperwithID
.map(new StateFulMapper())
.uid("mapper-id") //IDforthemapper
//Statelessprintingsink
.print(); //Auto-generatedID
```

Savepoint Recovery

If you do not specify the IDs manually, they will be generated automatically. You can automatically restore from the savepoint if these IDs do not change. The generated IDs depend on the structure of your program and are sensitive to program changes. Therefore, it is highly recommended to assign these IDs manually. When a savepoint is triggered, a single savepoint file will be created containing the checkpoint metadata. The actual checkpoint state will be kept around in the configured checkpoint directory, for example, with a FsStateBackend or RocksDBStateBackend:

1. Trigger a savepoint.
\$ bin/flink savepoint jobId [targetDirectory]

This command will trigger a savepoint for the job with ID:*jobid*. Furthermore, you can specify a target file system directory to store the savepoint. The directory must be accessible by JobManager. The targetDirectory is optional. If

`targetDirectory` is not configured, the directory specified by **`state.savepoints.dir`** in the configuration file is used to store savepoint.

You can configure a default savepoint target directory via the **`state.savepoints.dir`** key in the **`flink-conf.yaml`** file.

```
# Default savepoint target directory
```

NOTE

You are advised to configure `targetDirectory` to an HDFS path.

For example:

```
bin/flink savepoint 405af8c02cf6dc069a0f9b7a1f7be088 hdfs://savepoint.
```

2. Cancel a job with a savepoint.

```
$ bin/flink cancel -s [targetDirectory] jobId
```

This will atomically trigger a savepoint for the job with ID: *jobid* and cancel the job. Furthermore, you can specify a target file system directory to store the savepoint. The directory must be accessible by JobManager.

3. Resume jobs.

- Resume from a savepoint.

```
$ bin/flink run -s savepointPath [runArgs]
```

This command submits a job and specifies the savepoint path. The execution will resume from the respective savepoint state.

NOTE

`runArgs` is a user-defined parameter with parameter format and name varying depending on users.

- Allow non-restored state.

```
$ bin/flink run -s savepointPath -n [runArgs]
```

By default the resume operation will try to map all state of the savepoint back to the program you are restoring with. If you dropped an operator, you can skip the state that cannot be mapped to the new program via `-allowNonRestoredState` (short: `-n`).

4. Dispose savepoints.

```
$ bin/flink savepoint -d savepointPath
```

This command disposes the savepoint stored in: *savepointPath*.

Precautions

- Chained operators are identified by the ID of the first task. It is not possible to manually assign an ID to an intermediate chained task, for example, in the chain [a -> b -> c] only **a** can have its ID assigned manually, but not **b** or **c**. To work around this, you can manually define the task chains. To manually define chains, use the **`disableChaining()`** interface. See the following example:

```
env.addSource(new GetDataSource())
  .keyBy(0)
  .timeWindow(Time.seconds(2)).uid("window-id")
  .reduce(_+_).uid("reduce-id")
  .map(f=>(f,1)).disableChaining().uid("map-id")
  .print().disableChaining().uid("print-id")
```

- During job upgrade, the data type of operators cannot be changed.

10.5.4 Introduction to Flink Client CLI

Common CLIs

Common Flink CLIs are as follows:

1. yarn-session.sh

- You can run **yarn-session.sh** to start a standing Flink cluster to receive tasks submitted by clients. Run the following command to start a Flink cluster with three TaskManager instances:

```
bin/yarn-session.sh
```

- Run the following command to obtain other parameters of **yarn-session.sh**:

```
bin/yarn-session.sh -help
```

2. flink

- You can run Flink commands to submit a Flink job to a standing Flink cluster or to execute the job in single-server mode.

- Run the following command to submit a Flink job to a standing Flink cluster:

```
bin/flink run ../examples/streaming/WindowJoin.jar
```

NOTE

Before using the command to submit a task, you need to run **yarn-session.sh** to start the Flink cluster.

- Run the following command to execute a per-job YARN Cluster mode:

```
bin/flink run -m yarn-cluster ../examples/streaming/WindowJoin.jar
```

NOTE

The `-m yarn-cluster` parameter is used to specify a job to independently start a Flink cluster.

- List scheduled and running jobs (including their JobIDs):

```
bin/flink list
```

- Cancel a job:

```
bin/flink cancel <jobID>
```

- Stop a job (streaming jobs only):

```
bin/flink stop <jobID>
```

NOTE

The difference between cancelling and stopping a (streaming) job is the following:

- Cancel a job: On a cancel call, the operators in a job immediately receive a `cancel()` method call to cancel them as soon as possible. If operators are not stopping after the cancel call, Flink will start interrupting the thread periodically until it stops.
- Stop a job: Stop is only available for jobs which use sources that implement the `StoppableFunction` interface. The job will keep running until all sources properly shut down. Stop a job is more graceful than cancel, but it may cause the job to stop failing.

- Run the following command to obtain other parameters of Flink commands:
`bin/flink --help`

Precautions

- If **yarn-session.sh** uses **-z** to configure the specified ZooKeeper NameSpace, you need to use **-yid** to specify the application ID and use **-yz** to specify the ZooKeeper NameSpace when using **flink run**. The NameSpaces must be the same.

Example:

```
bin/yarn-session.sh -z YARN101  
bin/flink run -yid application_****_**** -yz YARN101 examples/streaming/WindowJoin.jar
```

- If **yarn-session.sh** does not use **-z** to configure the specified ZooKeeper NameSpace, do not use **-yz** to specify the ZooKeeper NameSpace when using **flink run**.

Example:

```
bin/yarn-session.sh  
bin/flink run examples/streaming/WindowJoin.jar
```

- You can use **-yz** to specify a ZooKeeper NameSpace when using **flink run -m yarn-cluster** to start a cluster.
- A NameSpace cannot be shared by multiple clusters.
- If you use **-z** to specify a ZooKeeper NameSpace when starting a cluster or submitting a job, you need to use **-z** again to specify the NameSpace when deleting, stopping, or querying the job or triggering the savepoint.

10.5.5 FAQ

10.5.5.1 Savepoints-related Problems

1. Should I assign IDs to all operators of the job?

Strictly speaking, you can assign IDs to only operators with statuses because savepoints save only statuses of operators that have statuses and not operators without statuses.

However, in actual situations, you are advised to allocate IDs to all operators because some internal operators, such as window operators of Flink have statuses. Whether an operator has status or not is not obvious. If you are specific that an operator does not have status, you do not need to call **uid()** to allocate an ID to the operator.

2. What would be the impact if I add an operator with status while upgrading the job?

If you add an operator with status to the job, the status of the operator is not saved in the savepoint and thus the status cannot be recovered. The operator is processed as an operator without status and is executed from the start.

3. What would be the impact if I delete an operator with status while upgrading the job?

By default, savepoints attempt to recover all saved statuses. If the savepoint saves the status of the deleted operator, recovery fails.

You can run the following command and use the **-allowNonRestoredState** (-n in the following command) parameter to skip recovering the status of the deleted operator:

```
$ bin/flink run -s savepointPath -n [runArgs]
```

4. What would be the impact if I rearrange the sequence of operators with statuses?
 - If you have allocated IDs to the operators, the statuses would be recovered normally.
 - If you do not allocate IDs to the operators, IDs would be automatically allocated to the operators in the new sequence. Then, the status recovery would fail.
5. What would be the impact if I delete or add an operator without status or rearrange the sequence of operators without statuses?
 - If you have allocated IDs to operators with statuses, operators without statuses do not affect status recovery from savepoints.
 - If you do not allocate IDs to operators, operators with statuses may be allocated with new IDs due to the sequence change. This would cause status recovery failure.

6. What would be the impact if I change the operator concurrency during the status recovery?

If the Flink version is higher than 1.2.0 and discarded status APIs, such as checkpointed, are not used, you can recover statuses from savepoints. Otherwise, statuses cannot be recovered.

10.5.5.2 What If the Chrome Browser Cannot Display the Title

Question

What to do if the title is not displayed when I use Chrome browser to access the Apache Flink Dashboard? This section takes the Tasks field as an example. When the pointer is placed on a colorful box in Tasks, the title of the box is not displayed, as shown in [Figure 10-65](#). [Figure 10-66](#) shows the normal situation when the title is displayed.

Figure 10-65 Title not displayed on the page

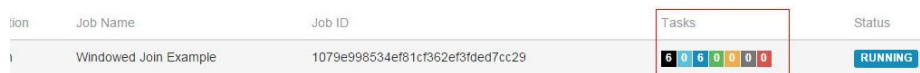
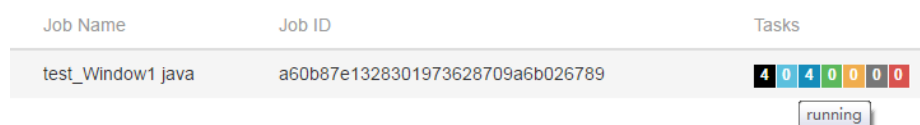


Figure 10-66 Title displayed normally



Answer

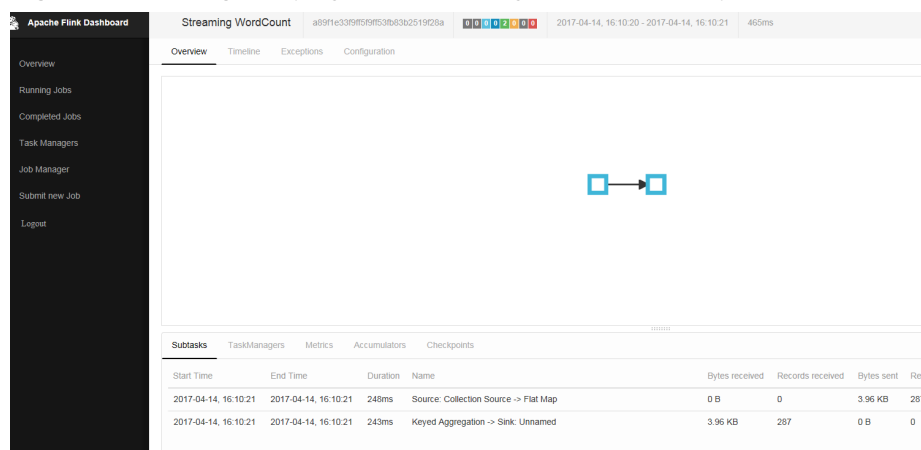
If the Chrome browser does not display the title, perform the following procedure:
Check whether a tool that affects the tooltip display by the Chrome browser is running. If so, shut down the tool.

10.5.5.3 What If the Page Is Displayed Abnormally on Internet Explorer 10/11

Question

What to do if Internet Explorer 10/11 does not display operator texts, as shown in [Figure 10-67](#)?

Figure 10-67 Page displayed abnormally on Internet Explorer 10/11



Answer

Flink uses the foreignObject element to draw scalable vector graphics (SVGs) but Internet Explorer 10/11 does not support foreignObject and thus operators cannot be displayed normally. Google Chrome browser is recommended.

10.5.5.4 What If Checkpoint Is Executed Slowly in RocksDBStateBackend Mode When the Data Amount Is Large

Question

What to do if checkpoint is executed slowly in RocksDBStateBackend mode when the data amount is large?

Cause Analysis

Customized windows are used and the window state is ListState. There are many values under the same key. In the case of a new value, the merge operation of RocksDB is used. When calculation is triggered, all values under the key are read.

- The RocksDB mode is merge() > merge() ... > merge() > read(). Data reading in this mode consumes much time, as shown in [Figure 10-68](#).

- The source operator sends a large amount of data in a short period of time and the data keys are the same. The window operator fails to process data fast enough and barriers accumulate in the cache. The time consumed for snapshot preparation is too long and the window operator cannot report snapshot completion to CheckpointCoordinator in the specified time. Therefore, CheckpointCoordinator determines snapshot preparation failure, as shown in [Figure 10-69](#).

Figure 10-68 Time monitoring

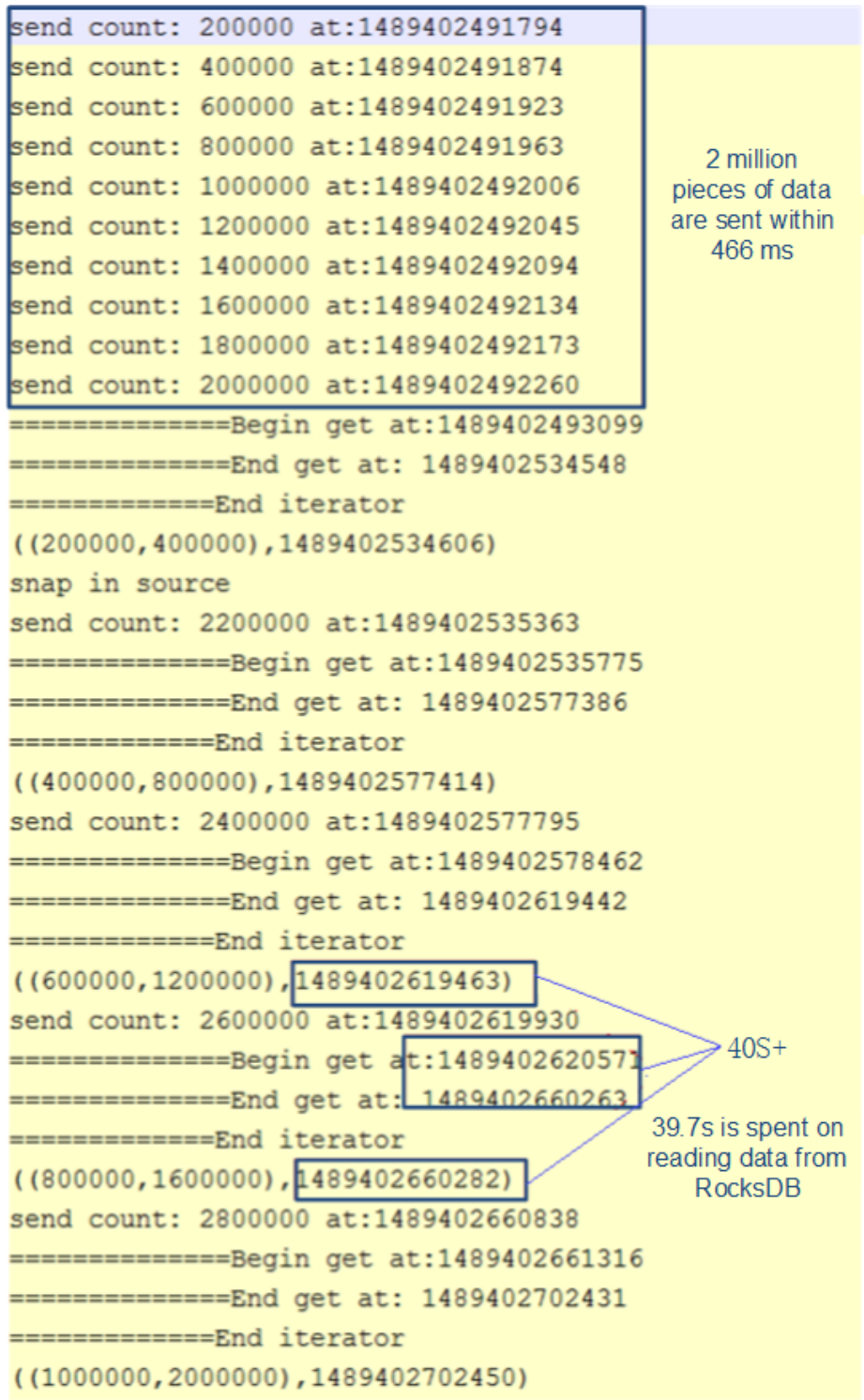
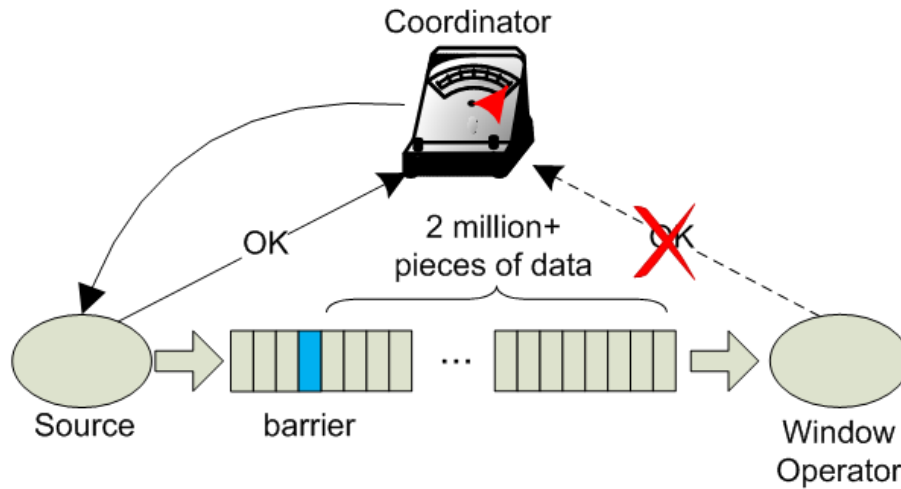


Figure 10-69 Relationship



Answer

Flink introduces the third-party software package RocksDB, whose defect causes the problem. You are advised to set checkpoint to FsStateBackend mode.

Set checkpoint to FsStateBackend mode in the application code as follows:

```
env.setStateBackend(new FsStateBackend("hdfs://hacluster/flink/checkpoint/"));
```

10.5.5.5 What If yarn-session Start Fails When blob.storage.directory Is Set to /home

Question

When **blob.storage.directory** is set to **/home**, the **blobStore-UUID** file cannot be created in **/home**. This causes yarn-session start failure

Answer

Step 1 It is recommended that **blob.storage.directory** be set to **/tmp** or **/opt/huawei/Bigdata/tmp**.

Step 2 When you set **blob.storage.directory** to a customized directory, manually grant permissions to the directory. This section takes the **admin** user of FusionInsight as an example.

1. Modify **conf/flink-conf.yaml** on the Flink client and run the **blob.storage.directory: /home/testdir/testdirdir/xxx** command.
2. Create the **/home/testdir** directory (level 1 is enough) and set the directory to be managed by the **admin** user.

```
SZV1000064084:/home # id admin
uid=20000(admin) gid=9998(ficommon) groups=9998(ficommon),8003(System_administrator_186)
SZV1000064084:/home # chown admin:ficommon testdir/ -R
```

NOTE

The **testdirdir/xxx** directory under the **/home/testdir/** directory is automatically created on each node when Flink cluster starts.

3. Run `./bin/yarn-session.sh -jm 2048 -tm 3072` on the client to check that `yarn-session` is normally started and the directory is successfully created.

```
SZV1000064084:/home # ll testdir/
total 4
drwxr-x-- 3 admin ficommon 4096 Mar 13 11:55 testdirdir
SZV1000064084:/home # ll testdir/testdirdir/
total 4
drwxr-x-- 4 admin ficommon 4096 Mar 13 11:55 xxx
SZV1000064084:/home # ll testdir/testdirdir/xxx/
total 8
drwxr-x-- 2 admin ficommon 4096 Mar 13 11:55 blobStore-6fb3f049-ecf3-49ac-9fc9-95ad0aeeffd3
drwxr-x-- 2 admin ficommon 4096 Mar 13 11:55 blobStore-ad89b118-8545-4ece-8cae-1334b01de857
```

----End

10.5.5.6 Why Does Non-static KafkaPartitioner Class Object Fail to Construct FlinkKafkaProducer010?

Question

After Flink kernel is upgraded to 1.3.0 or later versions, if Kafka calls the `FlinkKafkaProducer010` that contains the non-static `KafkaPartitioner` class object as parameter to construct functions, an error is reported.

The error message is as follows:

```
org.apache.flink.api.common.InvalidProgramException: The implementation of the FlinkKafkaPartitioner is not serializable. The object probably contains or references non serializable fields.
```

Answer

In the 1.3.0 version of Flink, the `FlinkKafkaDelegatePartitioner` class is added, so that Flink allows APIs that use `KafkaPartitioner`, for example, `FlinkKafkaProducer010` that contains `KafkaPartitioner` object, to construct functions.

The `FlinkKafkaDelegatePartitioner` class defines the member variable `kafkaPartitioner`.

```
private final KafkaPartitioner<T> kafkaPartitioner;
```

When Flink input parameter `KafkaPartitioner` constructs `FlinkKafkaProducer010`, the call stack is as follows:

```
FlinkKafkaProducer010(String topicId, KeyedSerializationSchema<T> serializationSchema, Properties producerConfig, KafkaPartitioner<T> customPartitioner)
-> FlinkKafkaProducer09(String topicId, KeyedSerializationSchema<IN> serializationSchema, Properties producerConfig, FlinkKafkaPartitioner<IN> customPartitioner)
----> FlinkKafkaProducerBase(String defaultTopicId, KeyedSerializationSchema<IN> serializationSchema, Properties producerConfig, FlinkKafkaPartitioner<IN> customPartitioner)
-----> ClosureCleaner::clean(Object func, boolean checkSerializable)
```

Run the `KafkaPartitioner` object to construct a `FlinkKafkaDelegatePartitioner` object, and then check whether the object can be serializable. The `ClosureCleaner::clean` function is a static function. If the `KafkaPartitioner` object in a case is non-static, the `ClosureCleaner::clean` function cannot access the non-static member variable `kafkaPartitioner` in the `KafkaDelegatePartitioner` class and an exception is reported.

Either of the following methods can be used to solve the problem:

- Change the KafkaPartitioner class into static class.
- Use the FlinkKafkaProducer010 that contains FlinkKafkaPartitioner as the parameter to construct functions. In this case, FlinkKafkaDelegatePartitioner is not constructed and the exception about member variable is avoided.

10.5.5.7 When I Use a Newly Created Flink User to Submit Tasks, Why Does the Task Submission Fail and a Message Indicating Insufficient Permission on ZooKeeper Directory Is Displayed?

Question

When I use a newly-created Flink user to submit tasks, the task submission fails because of insufficient permission on the ZooKeeper directory. The error message in the log is as follows:

```
NoAuth for /flink_base/flink/application_1499222480199_0013
```

Answer

1. Check whether the permission on the /flink_base directory in ZooKeeper is 'world,'anyone: cdrwa; If no, change the permission on the /flink_base directory to 'world,'anyone: cdrwa and go to [step 2](#). If yes, go to [step 2](#).
2. In the configuration file of Flink, the default value of high-availability.zookeeper.client.acl is creator, indicating that only the creator of the directory has permission on it. The user created later has no access to the /flink_base/flink directory in ZooKeeper because only the user created earlier has permission on it.

To solve the problem, perform the following operation as the newly-created user:

- a. Check the configuration file **conf/flink-conf.yaml** on the client.
- b. Modify the parameter **high-availability.zookeeper.path.root** to the corresponding ZooKeeper directory, for example, /flink2.
- c. Submit tasks again.

10.5.5.8 Why Cannot I Access the Apache Flink Dashboard?

Question

Why cannot I access the Apache Flink Dashboard through the URL: http://IP address of JobManager:port of JobManager.

Answer

The IP address of the computer you used has not been added to the whitelist of Apache Flink Dashboard. To solve this problem, modify the **conf/flink-conf.yaml** configuration file as follows:

1. Check whether the value of **jobmanager.web.ssl.enabled** is **false**. If not, set it to **false**.
2. Check whether the IP address of the computer you used has been added to the values of **jobmanager.web.access-control-allow-origin** and

jobmanager.web.allow-access-address. If the IP address has not been added, add it to the two parameters:
jobmanager.web.access-control-allow-origin:*IP address of the computer where the browser is installed*
jobmanager.web.allow-access-address:*IP address of the computer where the browser is installed*

10.5.5.9 How Do I View the Debugging Information Printed Using System.out.println or Export the Debugging Information to a Specified File?

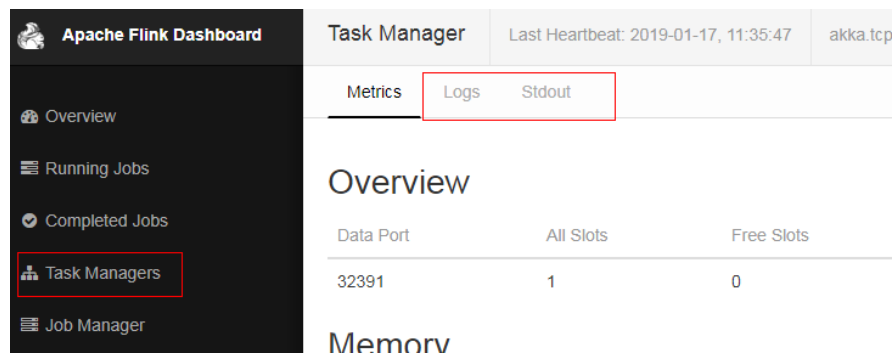
Question

System.out.println is added to the Flink service codes for printing debugging information. How do I view the debugging log? How do I export service logs to a specified file to differentiate service logs from run logs?

Answer

All run logs of Flink are printed to the local directory of Yarn. By default, all logs are exported to **taskmanager.log** in the local directory of Yarn container. All logs generated by invoking System.out will be exported to the **taskmanager.out** file. You can perform as follows to view the logs:

1. Log in to the native Flink web page.
2. Choose **Task Managers > Logs** or **Task Managers > Stdout** on the left to view log information.



Configure the function of printing service logs and Task Manager run logs separately:

NOTE

If service logs and Task Manager run logs are separately printed, service logs are not exported to the **taskmanager.log** file and cannot be viewed on the web page.

1. Modify the configuration file **logback.xml** in the **conf** directory of the client. Add the following log configuration information to the file. Modify the information in bold according to the actual situation.

```
<appender name="TEST" class="ch.qos.logback.core.rolling.RollingFileAppender">  
  <file>/path/test.log</file>  
  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">  
    <fileNamePattern>/path/test.log.%i</fileNamePattern>  
    <minIndex>1</minIndex>  
    <maxIndex>20</maxIndex>  
  </rollingPolicy>  
  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">  
    <maxFileSize>20MB</maxFileSize>  
  </triggeringPolicy>
```

```
<encoder>
  <pattern>%d{"yyyy-MM-dd HH:mm:ss,SSS"} | %m %n</pattern>
</encoder>
</appender>

<logger name="com.huawei.bigdata.flink.examples" additivity="false">
  <level value="INFO"/>
  <appender-ref ref="TEST"/>
</logger>
```

2. Run `yarn-session.sh` to submit the task.

NOTE

If the configuration file `logback.xml` contains `<file>/path/test.log</file>`, ensure that the user (configured `flink-conf.yaml`) used for running the task has the write and read permissions on the directory.

10.5.5.10 Incorrect GLIBC Version

Question

When **State Backend** is set to **RocksDB** for a Flink task, the following error message is displayed:

```
Caused by: java.lang.UnsatisfiedLinkError: /srv/BigData/hadoop/data1/nm/usercache/***/appcache/application_****/rocksdb-lib-****/librocksdbjni-linux64.so: /lib64/libpthread.so.0: version `GLIBC_2.12` not found (required by /srv/BigData/hadoop/***/librocksdbjni-linux64.so)
at java.lang.ClassLoader$NativeLibrary.load(Native Method)
at java.lang.ClassLoader.loadLibrary0(ClassLoader.java:1965)
at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1890)
at java.lang.Runtime.load0(Runtime.java:795)
at java.lang.System.load(System.java:1062)
at org.rocksdb.NativeLibraryLoader.loadLibraryFromJar(NativeLibraryLoader.java:78)
at org.rocksdb.NativeLibraryLoader.loadLibrary(NativeLibraryLoader.java:56)
at
org.apache.flink.contrib.streaming.state.RocksDBStateBackend.ensureRocksDBIsLoaded(RocksDBStateBackend.java:734)
... 11 more
```

Possible Causes

The version of the system where the task runs and the version of the system where the compilation environment locates are different, resulting in the incompatibility of GLIBC versions.

Troubleshooting Method

Run the `strings /lib64/libpthread.so.0 | grep GLIBC` command to check whether the GLIBC version is earlier than 2.12.

Procedure

If the version of the GLIBC is too early, use the file of later version (2.12) to replace `libpthread-*.so`. (This is a link file. You need to replace only the file that is linked to this link file.)

References

None

11 Flink Development Guide (Normal Mode)

11.1 Overview

11.1.1 Application Development

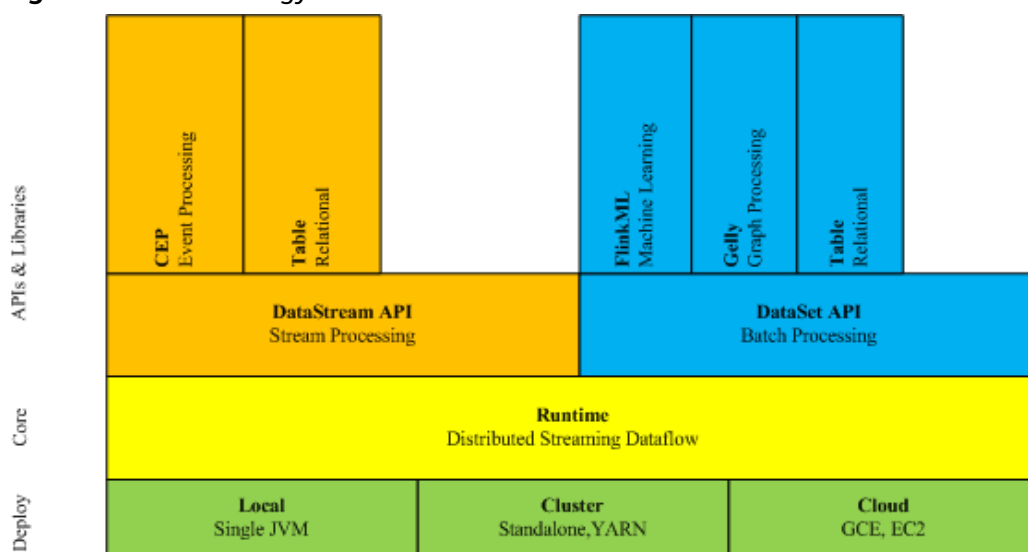
Overview

Flink is a unified computing framework that supports both batch processing and stream processing. It provides a stream data processing engine that supports data distribution and parallel computing. Flink features stream processing and is a top open-source stream processing engine in the industry.

Flink provides high-concurrency pipeline data processing, millisecond-level latency, and high reliability, making it suitable for low-latency data processing.

[Figure 11-1](#) shows the technology stack of Flink.

Figure 11-1 Technology stack of Flink



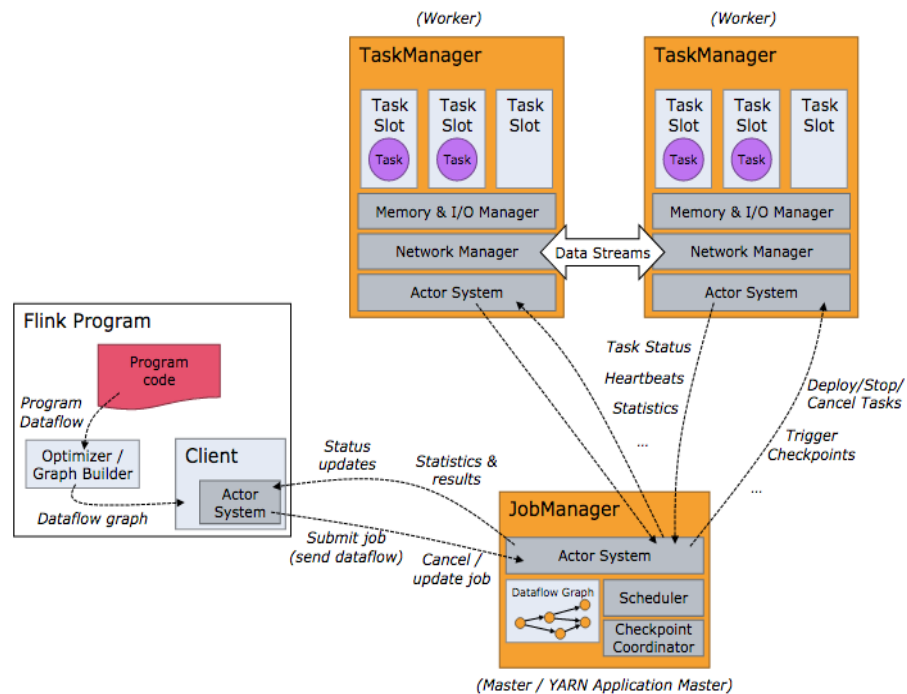
The following lists the key features of Flink in the current version:

- DataStream
- Checkpoint
- Window
- Job Pipeline
- Configuration Table

Architecture

Figure 11-2 shows the architecture of Flink.

Figure 11-2 Flink architecture



As shown in Figure 11-2, the entire Flink system consists of three parts:

- **Client**
Flink client is used to submit jobs (streaming jobs) to Flink.
- **TaskManager**
TaskManager (also called worker) is a service execution node of Flink. It executes specific tasks. A Flink system could have multiple TaskManagers. These TaskManagers are equivalent to each other.
- **JobManager**
JobManager (also called master) is a management node of Flink. It manages all TaskManagers and schedules tasks submitted by users to specific TaskManagers. In high-availability (HA) mode, multiple JobManagers are

deployed. Among these JobManagers, one of which is selected as the leader, and the others are standby.

Flink provides the following features:

- Low latency
Millisecond-level processing capability.
- Exactly once
Asynchronous snapshot mechanism, ensuring that all data is processed only once.
- High availability
Leader/Standby JobManagers, preventing single point of failure (SPOF).
- Scale out
Manual scale out supported by TaskManagers.

Flink Development APIs

Flink DataStream API can be developed using Scala and Java languages, as shown in [Table 11-1](#).

Table 11-1 Flink DataStream API

Function	Description
Scala API	API in Scala, which can be used for data processing, such as filtering, joining, windowing, and aggregation. The Scala API is recommended for development because Scala is concise and easy to understand.
Java API	API in Java, which can be used for data processing, such as filtering, joining, windowing, and aggregation.

11.1.2 Basic Concepts

Basic Concepts

- **DataStream**
DataStream is the minimum unit of Flink processing and is one of core concepts of Flink. DataStreams are initially imported from external systems in formats of socket, Kafka, and files. After being processed by Flink, DataStreams are exported to external systems in formats of socket, Kafka, and files.
- **Data Transformation**
A data transformation is a data processing unit that transforms one or multiple DataStreams into a new DataStream.
Data transformation can be classified as follows:
 - One-to-one transformation, for example, map.
 - One-to-zero, one-to-one, or one-to-multiple transformation, for example, flatMap.

- One-to-zero or one-to-one transformation, for example, filter.
- Multiple-to-one transformation, for example, union.
- Transformation of multiple aggregations, for example, window and keyby.
- **Checkpoint**
Checkpoint is the most important Flink mechanism to ensure reliable data processing. Checkpoints ensure that all application statuses can be recovered from a checkpoint in case of failure occurs and data is processed exactly once.
- **Savepoint**
Savepoints are externally stored checkpoints that you can use to stop-and-resume or update your Flink programs. After the upgrade, you can set the task status to the savepoint storage status and start the restoration, ensuring data continuity.

11.1.3 Development Process

Development Process of a Flink Application

Figure 11-3 shows the Flink development process:

Figure 11-3 Development process of a Flink application

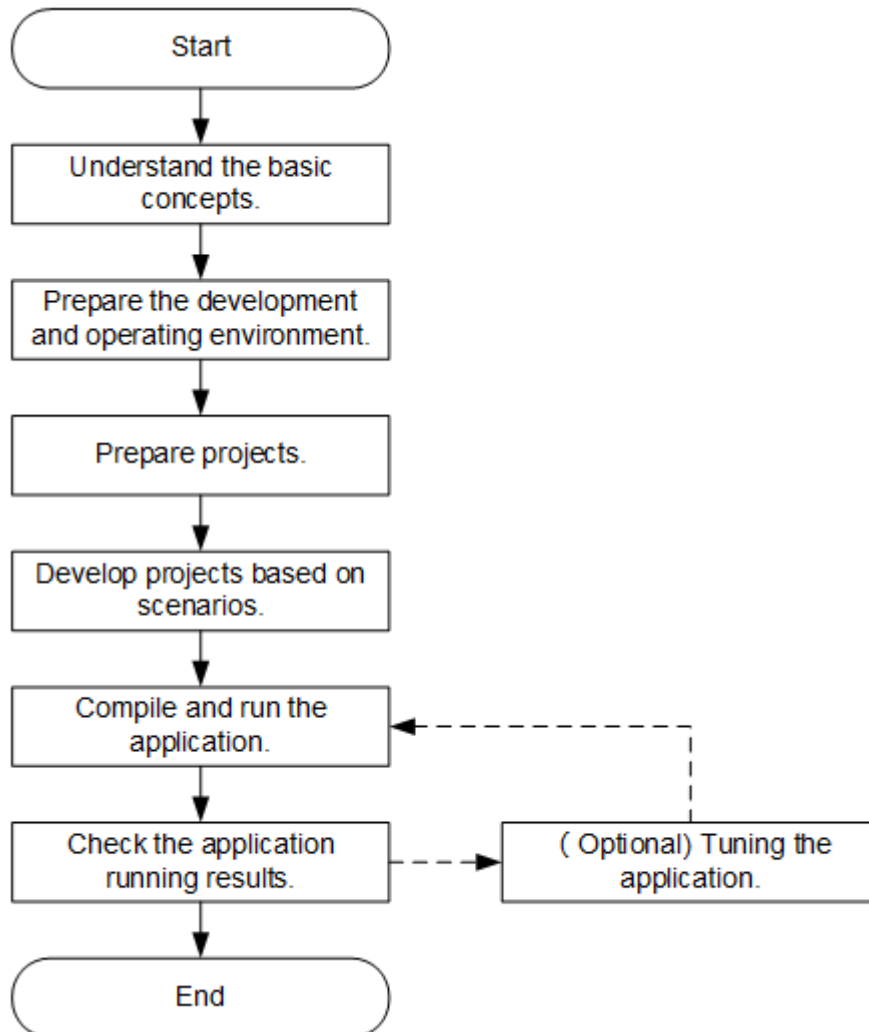


Table 11-2 Description of the development process

Phase	Description	Reference
Understand the basic concepts.	Before the development process, you are advised to gain a basic understanding of Flink.	Basic Concepts
Prepare the development and operating environment.	Flink applications can be developed using Scala or Java. You are advised to use IntelliJ IDEA tool to configure the development environment as instructed, based on your development language. The running environment of Flink is the Flink client. Install and configure the client as instructed.	Preparing for Development and Operating Environment
Prepare projects.	Flink provides sample projects for you to import and learn. Alternatively, you can create a Flink project as instructed.	Configuring and Importing a Sample Project Creating a Project (Optional)
Develop the project.	Sample projects in Scala and Java are provided to help you quickly understand programming interfaces of Flink components.	Developing an Application
Compile and run the application.	Guidance is provided for you to compile and run a developed application.	Compiling and Running the Application
Check running results.	Application running results are stored in a path specified by you. You can also view application running status through Apache Flink Dashboard.	Viewing the Debugging Result
Tune the application.	Tune the application to meet certain service requirements. After the tuning is complete, the application needs to be compiled and run again.	"Flink Performance Tuning" in the Component Operation Guide

11.2 Environment Preparation

11.2.1 Preparing for Development and Operating Environment

Preparing Development Environment

[Table 11-3](#) describes the environment required for application development.

Table 11-3 Development environment

Item	Description
OS	<ul style="list-style-type: none"> Development environment: Windows OS. Windows 7 or later is supported. Operating environment: Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> Oracle JDK versions: 1.8 IBM JDK versions: 1.8.5.11 For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> OpenJDK: 1.8.0_272
IntelliJ IDEA installation and configuration	IntelliJ IDEA is a tool used to develop Flink applications. The version must be 2019.1 or other compatible version.
Scala installation	Install Scala is the basic configuration for the Scala development environment. The required version is 2.11.7.
Scala plug-in installation	Installing Scala plug-ins is the basic configuration for the Scala development environment. The required version is 1.5.4.
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
7-zip	It is a tool used to decompress .zip and .rar packages. The 7-Zip 16.04 is supported.

Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client on the node. For example, the client installation directory is **/opt/client**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. Log in to the FusionInsight Manager portal. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight_Cluster_1_Services_ClientConfig\Flink\config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/client/conf**.

For example, if the client software package is **FusionInsight_Cluster_1_Services_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Flink/config/* root@IP address of the client node:/opt/client/conf
```

[Table 11-4](#) describes the main configuration files.

Table 11-4 Configuration file

Document Name	Function
core-site.xml	Configures Flink parameters.
hdfs-site.xml	Configures hdfs parameters.
yarn-site.xml	Configures yarn parameters.
flink-conf.yaml	Configures Flink parameters.

- c. Check the network connection of the client node.
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

11.2.2 Configuring and Importing a Sample Project

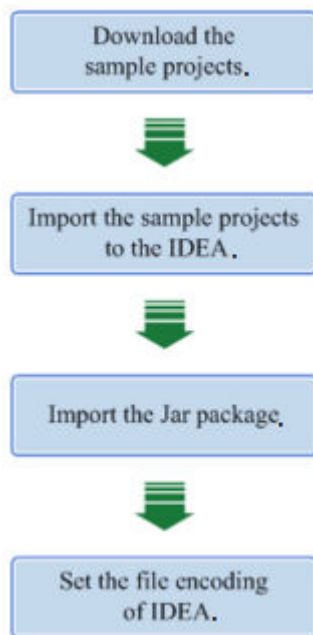
Scenario

Flink provides sample projects for multiple scenarios, including Java and Scala sample projects, to help you quickly learn Flink projects.

Methods to import Java and Scala projects are the same.

The following example describes how to import Java sample code. [Figure 11-4](#) shows the operation process.

Figure 11-4 Process of importing sample projects



Procedure

Step 1 Obtain the sample project folder **flink-examples-normal** in the **src\flink-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

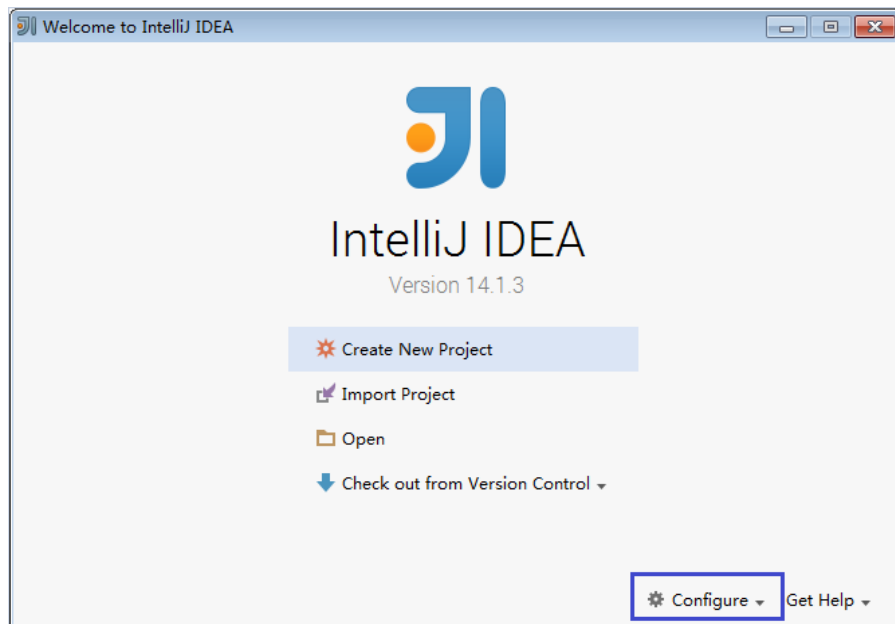
NOTE

- In security mode, obtain the sample project **flink-examples-security** from the **src\flink-examples** folder.
- In normal mode, obtain the sample project **flink-examples-normal** from the **src\flink-examples** folder.

Step 2 Before importing the sample project, configure JDK for IntelliJ IDEA.

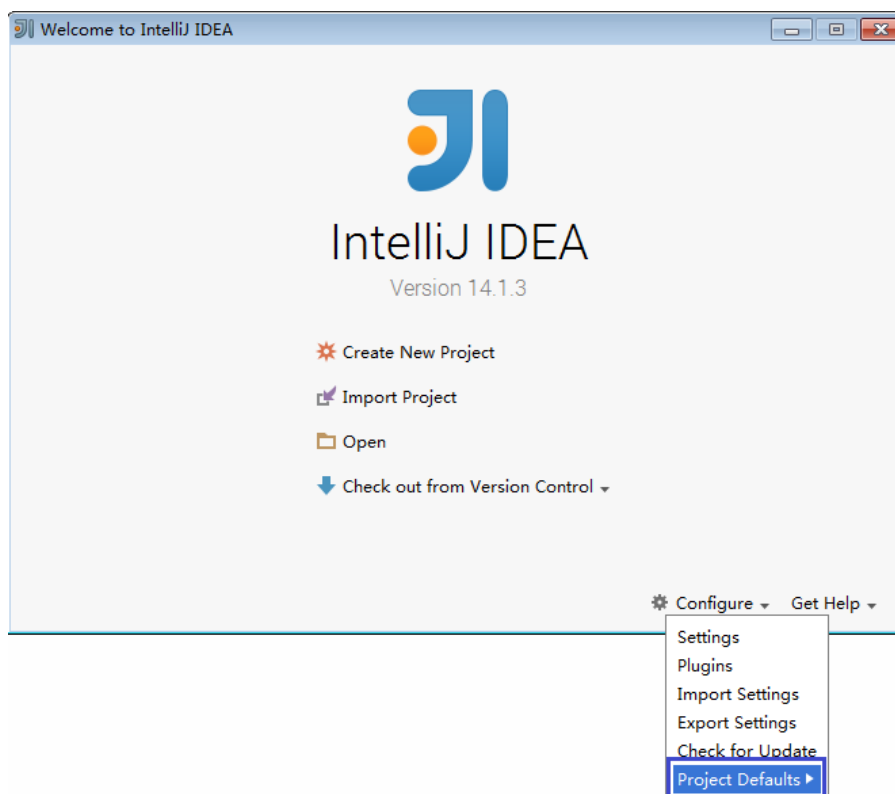
1. Start IntelliJ IDEA and click **Configure**.

Figure 11-5 Choosing Configure



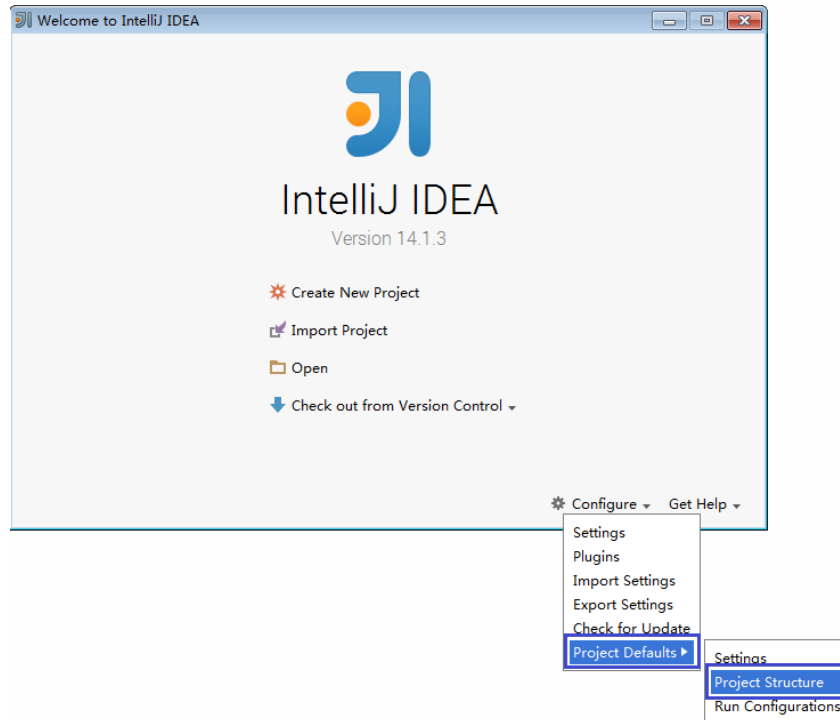
2. Choose **Project Defaults** from the **Configure** drop-down list.

Figure 11-6 Choosing Project Defaults



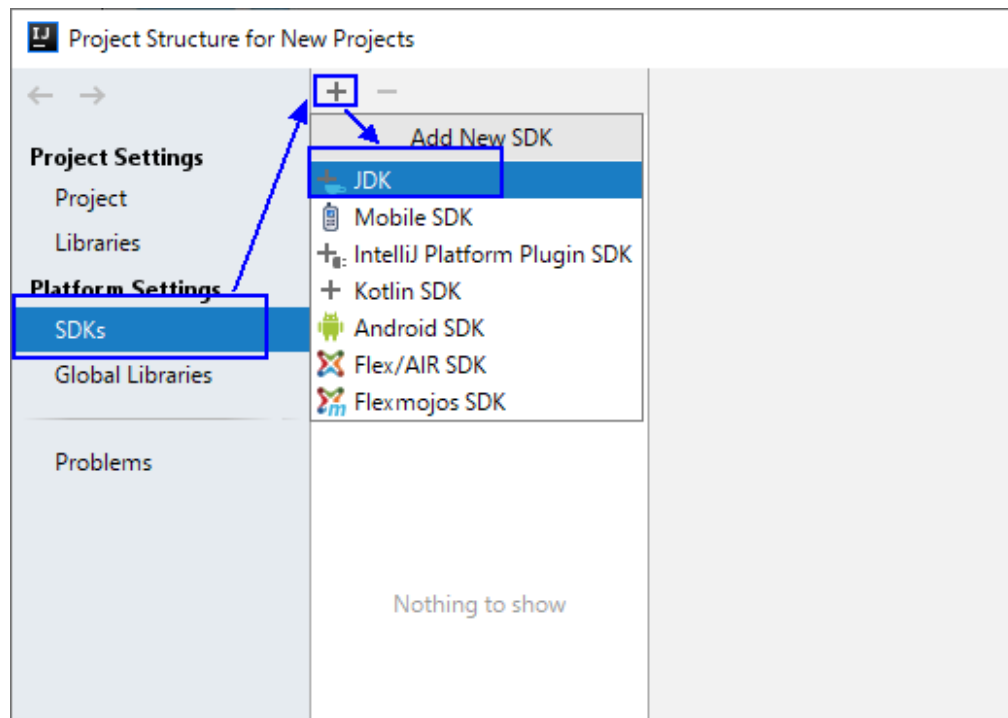
3. Choose **Project Structure** from the **Project Defaults** submenu.

Figure 11-7 Project Defaults



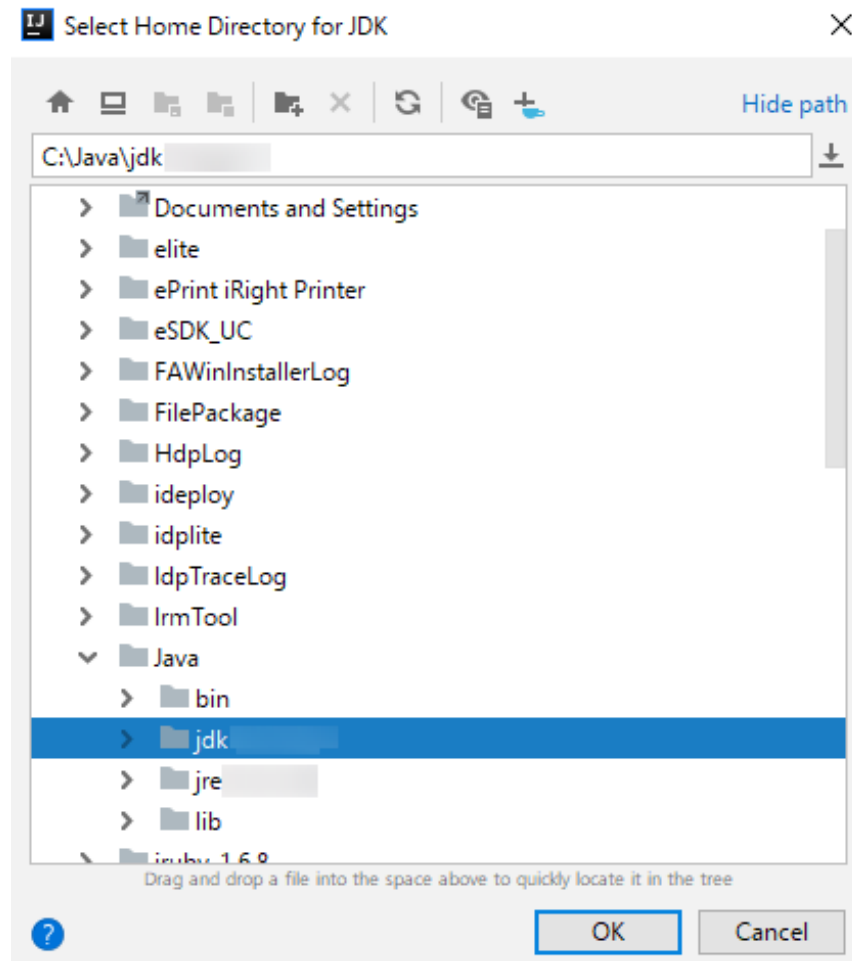
4. On the **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 11-8 Adding the JDK



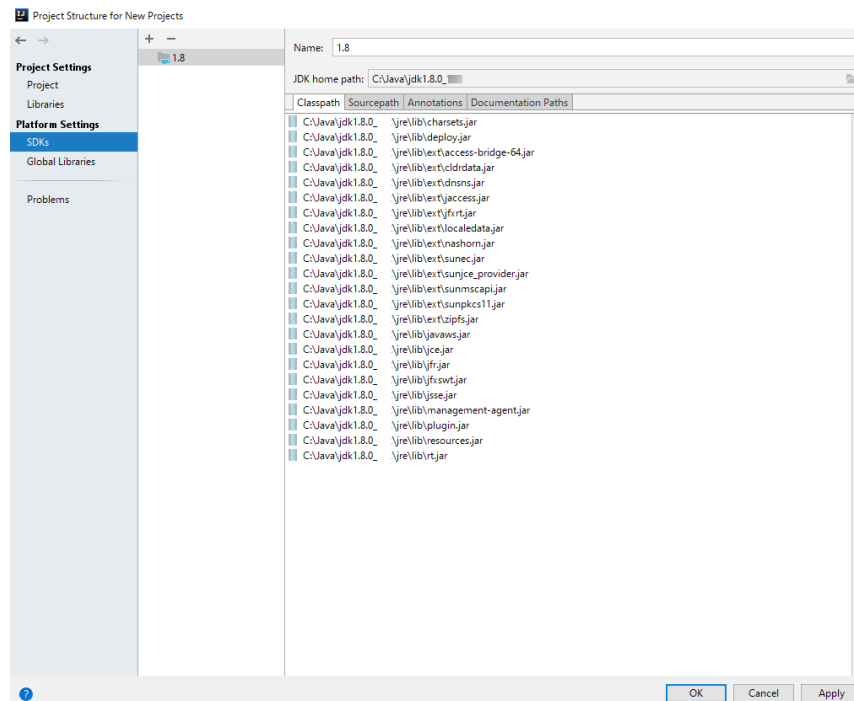
5. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 11-9 Selecting the JDK directory



6. After the JDK is selected, click **OK** to complete the configuration.

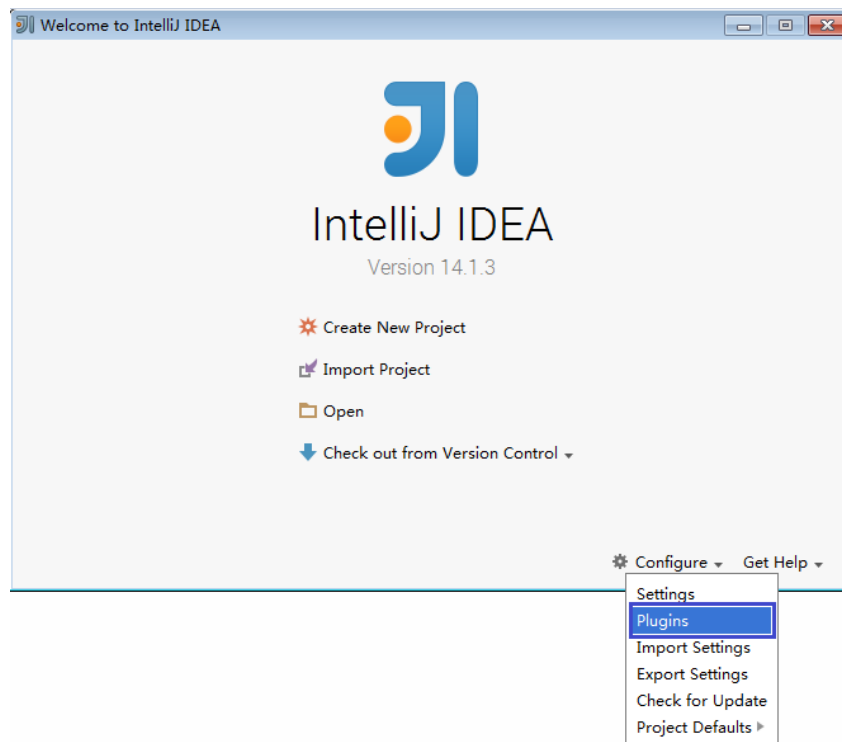
Figure 11-10 Completing the JDK configuration



Step 3 (Optional) If a Scala sample project is imported, install Scala plug-ins in IntelliJ IDEA.

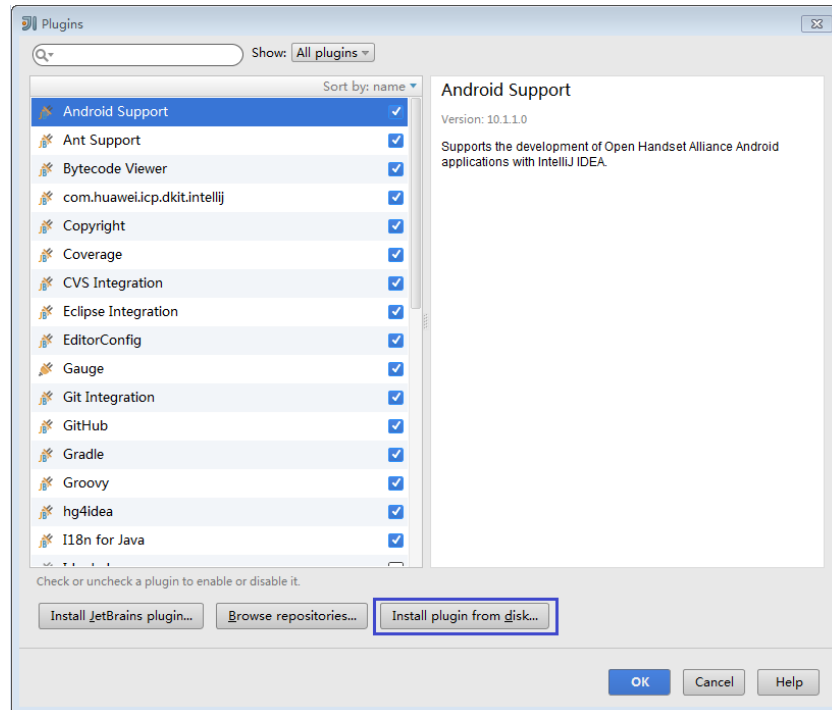
1. Choose **Plugins** from the **Configure** drop-down list.

Figure 11-11 Plugins



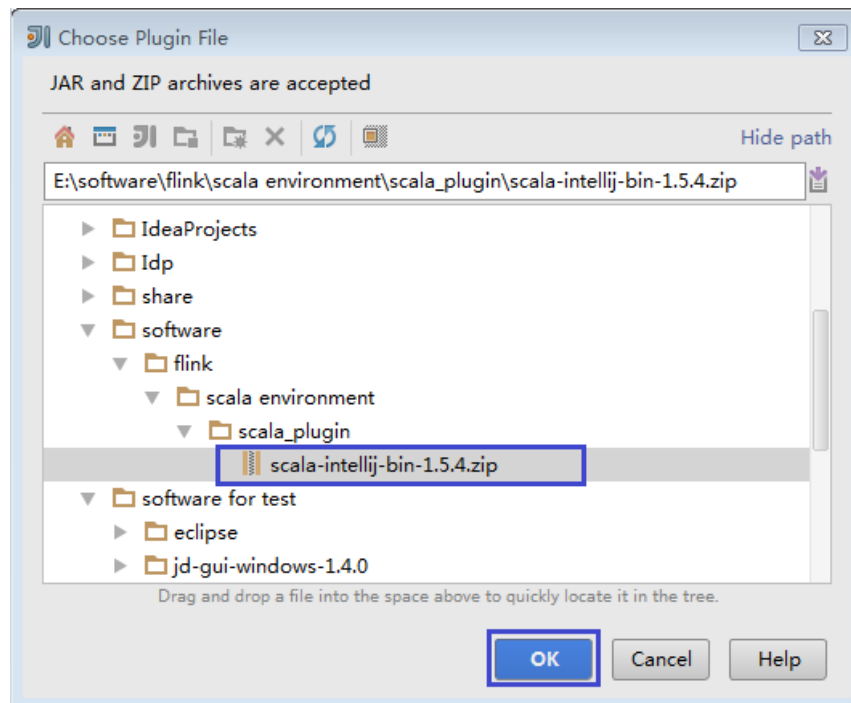
2. On the **Plugins** page, click **Install plugin from disk**.

Figure 11-12 Installing plugins from disk



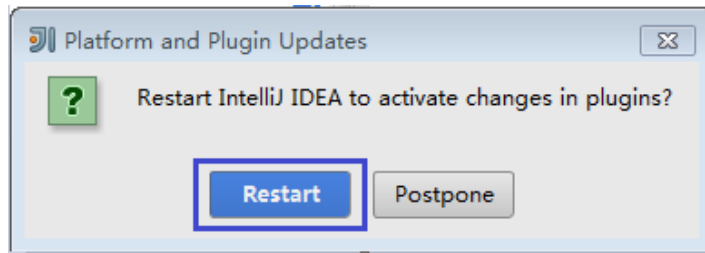
3. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.

Figure 11-13 Choose Plugin File



4. On the **Plugins** page, click **Apply** to install the Scala plugins.
5. On the **Platform and Plugin Updates** page that is displayed, click **Restart** to make the configurations take effect.

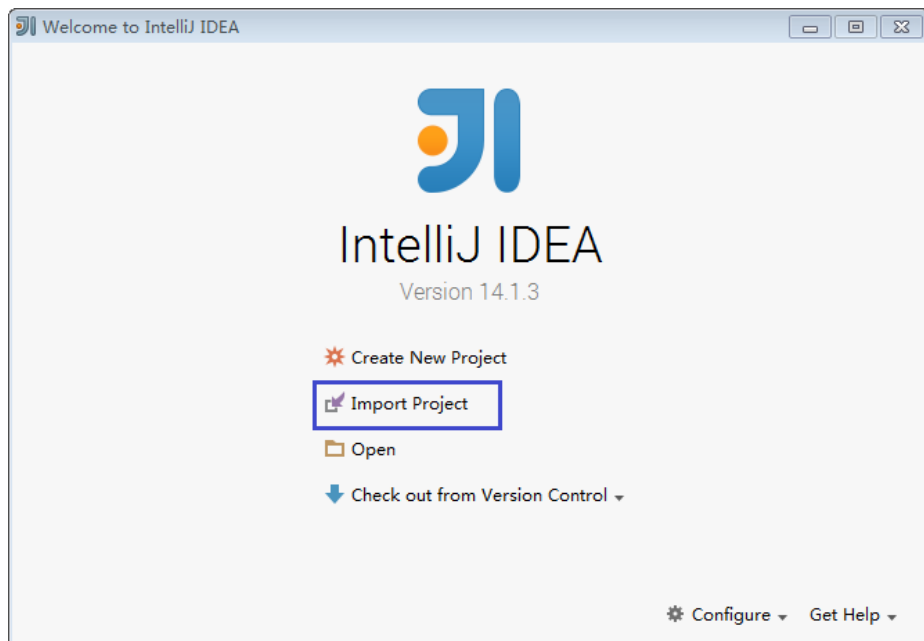
Figure 11-14 Platform and Plugin Updates



Step 4 Import the Java sample project to IDEA.

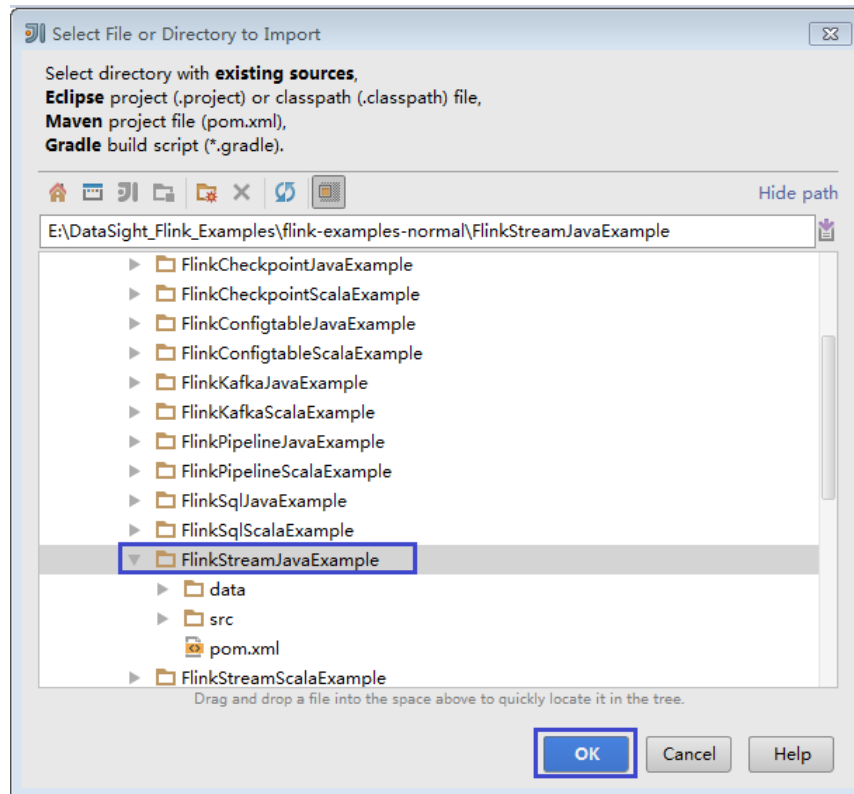
1. Start IntelliJ IDEA. On the **Quick Start** page, select **Import Project**.
Alternatively, for the used IDEA tool, add the project directly from the IDEA home page. Choose **File > Import project...** to import a project.

Figure 11-15 Importing a project (on the Quick Start page)



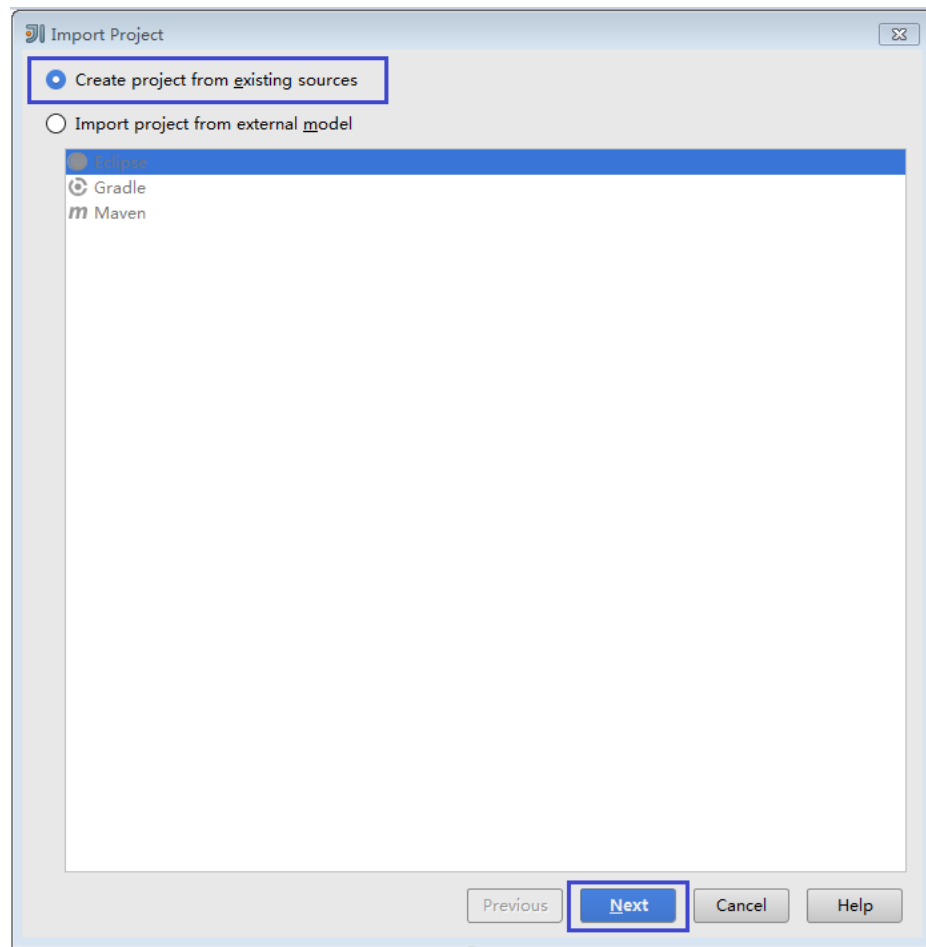
2. Select the directory for storing the imported projects and click **OK**.

Figure 11-16 Select File or Directory to Import



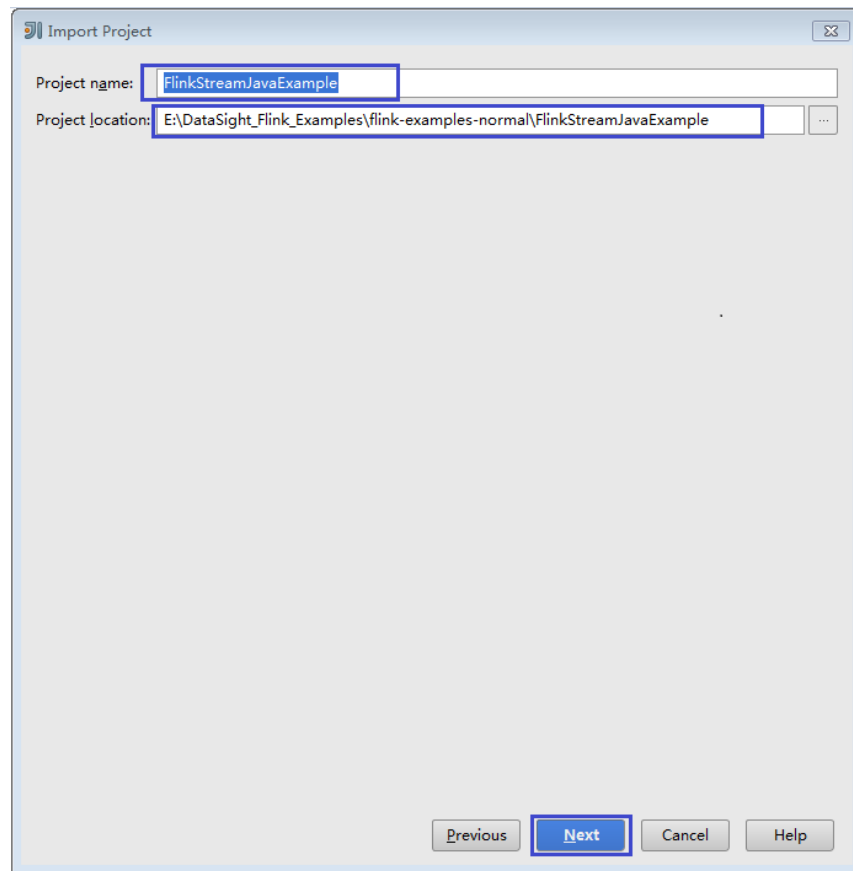
3. Select **Create project from existing sources** and click **Next**.

Figure 11-17 Create project from existing sources



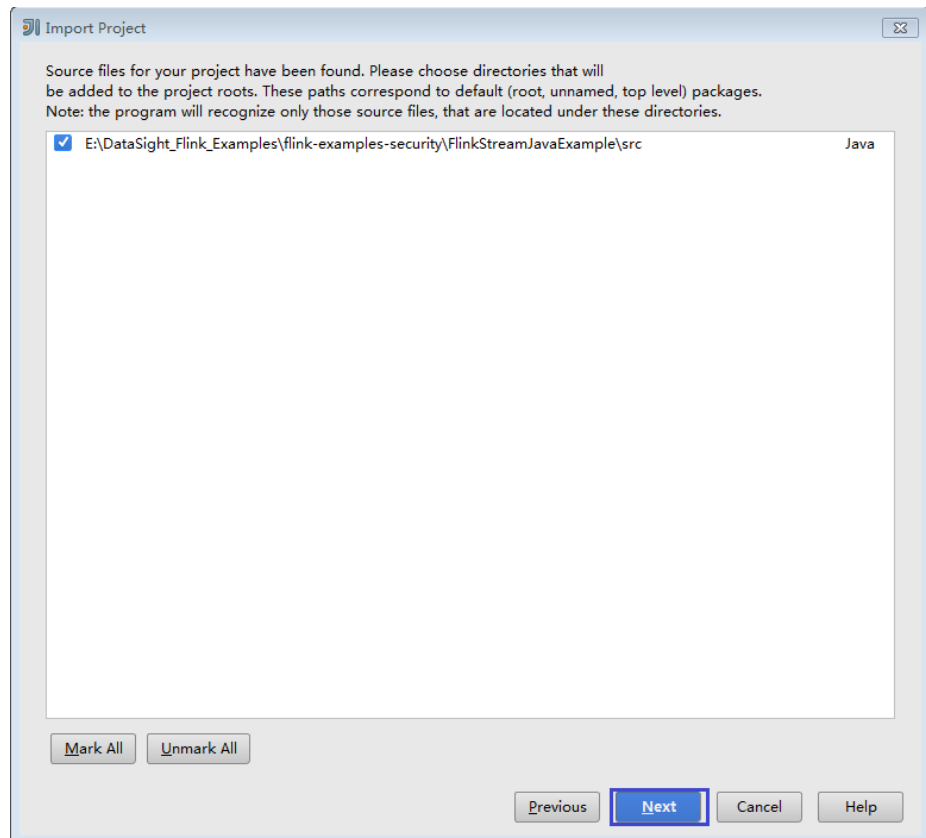
4. Confirm the project location and project name, and click **Next**.

Figure 11-18 Import Project



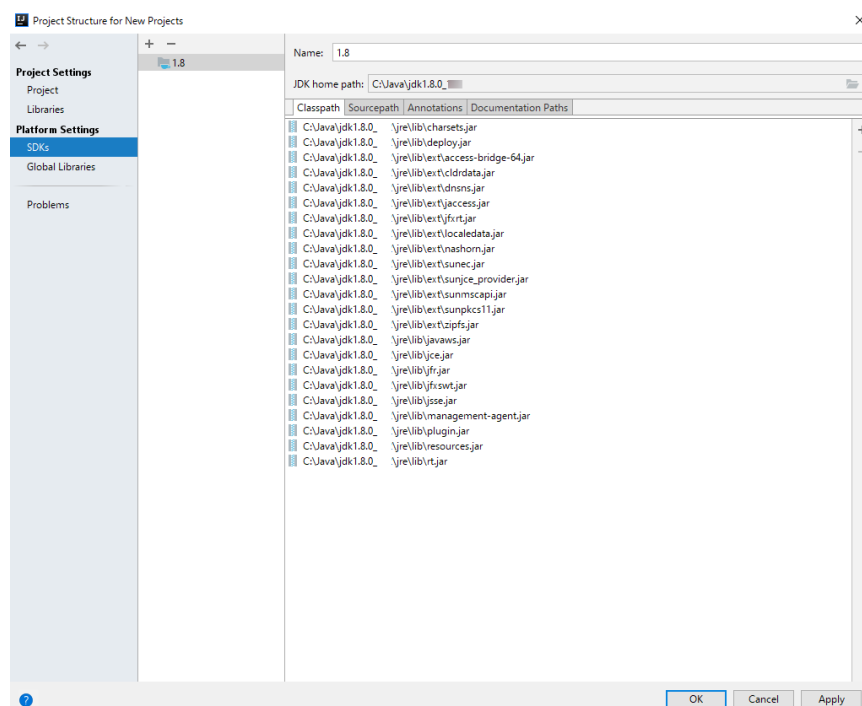
5. Retain the default value of the **root** directory for the project to be imported and click **Next**.

Figure 11-19 Import Project



6. Retain the default dependency library that is automatically recognized by IDEA and the suggested module structure, and click **Next**.
7. Confirm the JDK to be used by the project and click **Next**.

Figure 11-20 Select project SDK



8. After the import is complete, click **Finish**. The imported sample project is displayed on the IDEA home page.

Figure 11-21 Completing the import

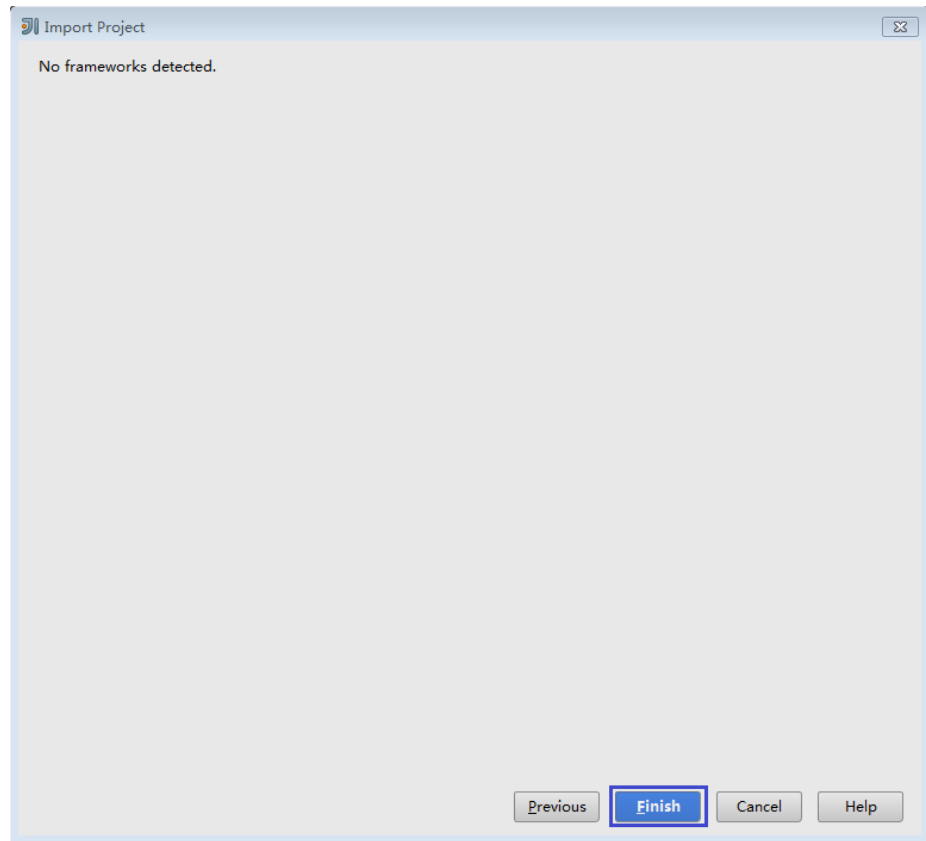
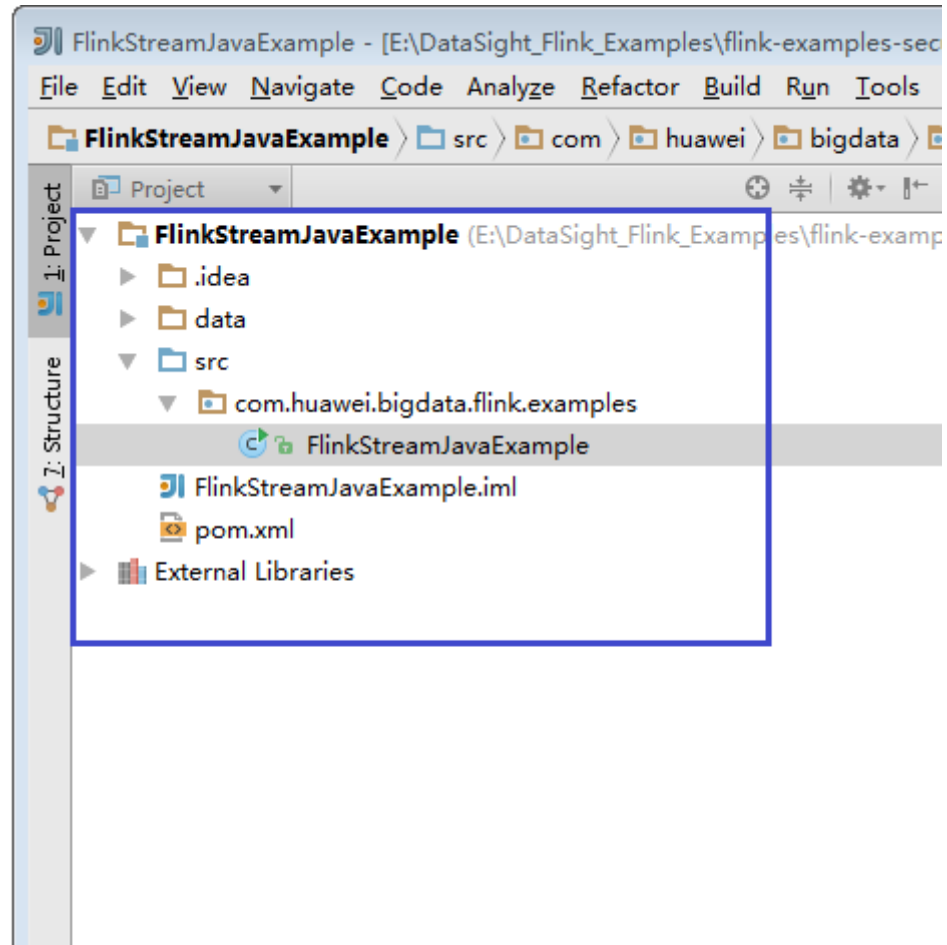


Figure 11-22 Imported project



Step 5 Import the dependency JAR file for the sample project.

If the sample project code is obtained from an open-source image site, dependency JAR files are automatically downloaded after Maven is configured.

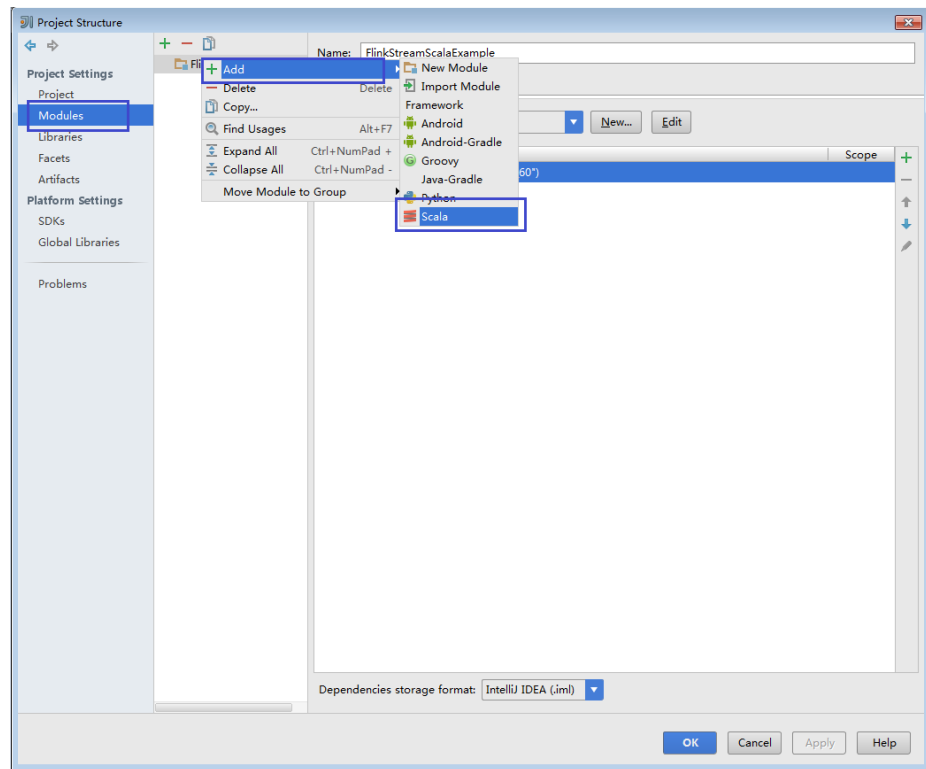
NOTE

If other FusionInsight components such as Kafka are used in the sample code, obtain them from the installation directory of these FusionInsight components. For details about dependency packages of sample projects, see [Reference Information](#).

Step 6 (Optional) If a Scala sample application is imported, configure a language for the project.

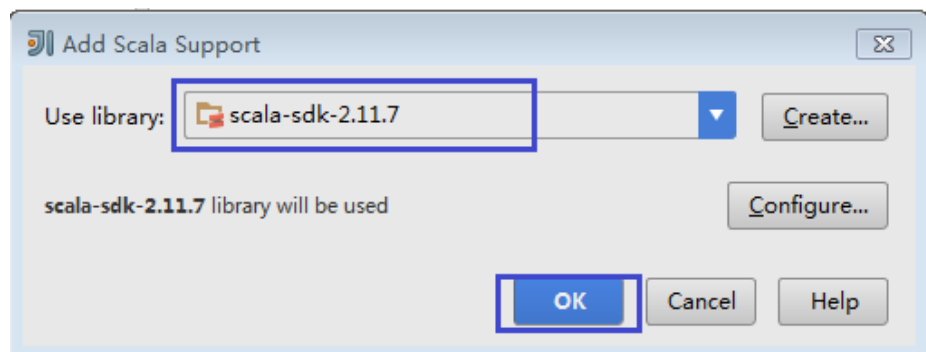
1. On the IDEA home page, choose **File > Project Structures...** to go to the **Project Structure** page.
2. Choose **Modules**, right-click a project name, and choose **Add > Scala**.

Figure 11-23 Selecting Scala



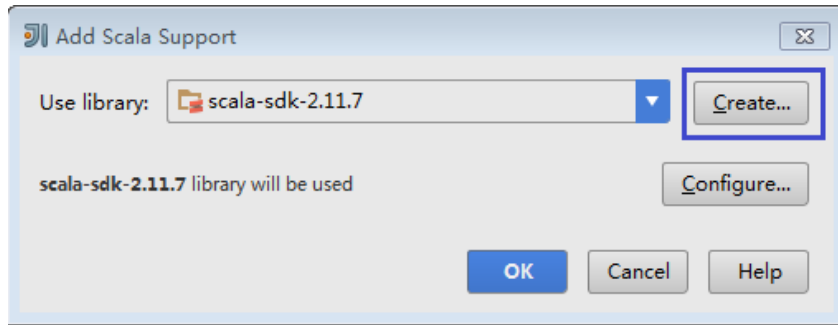
3. Wait until IDEA identifies Scala SDK, select the dependency JAR files in the **Add Scala Support** dialog box, and then click **OK**.

Figure 11-24 Add Scala Support



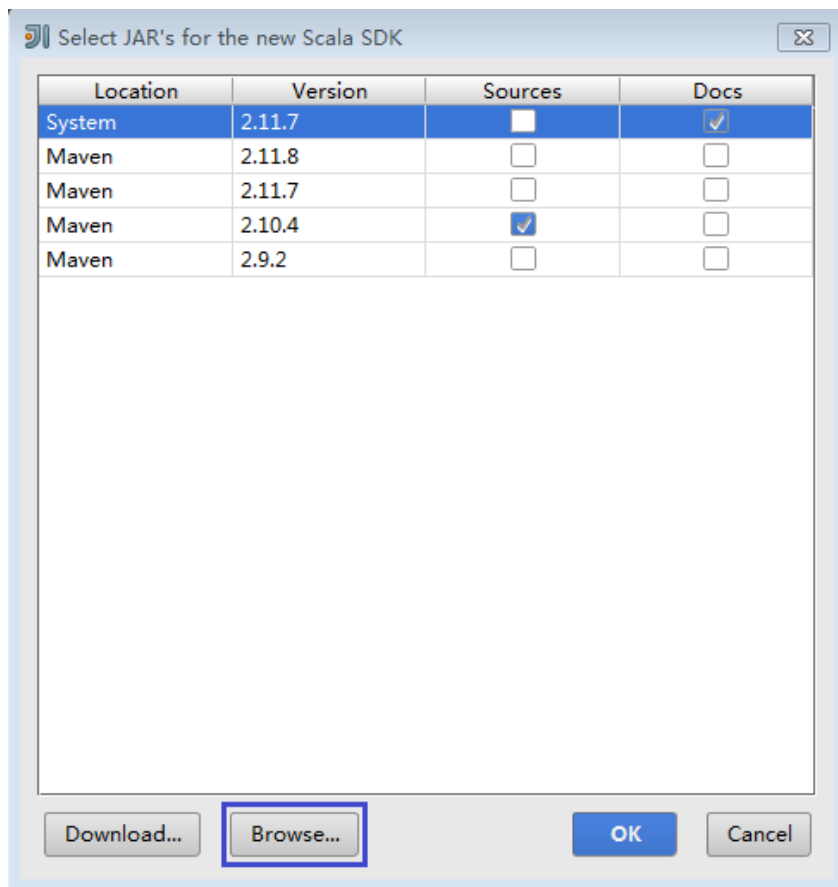
4. If IDEA fails to identify Scala SDK, create a Scala SDK.
 - a. Click **Create....**

Figure 11-25 Create...



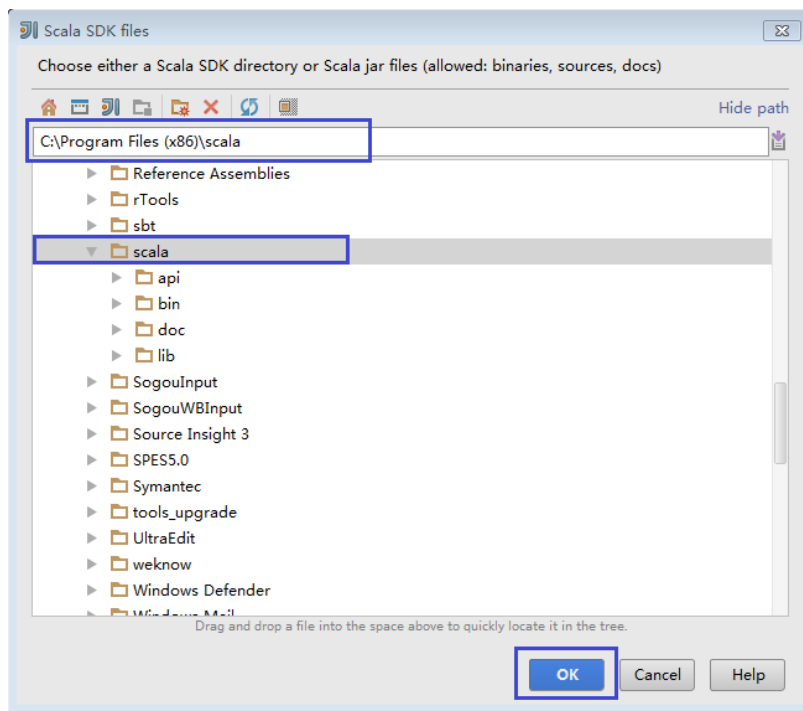
- b. On the **Select JAR's for the new Scala SDK** page, click **Browse...**

Figure 11-26 Select JAR's for the new Scala SDK



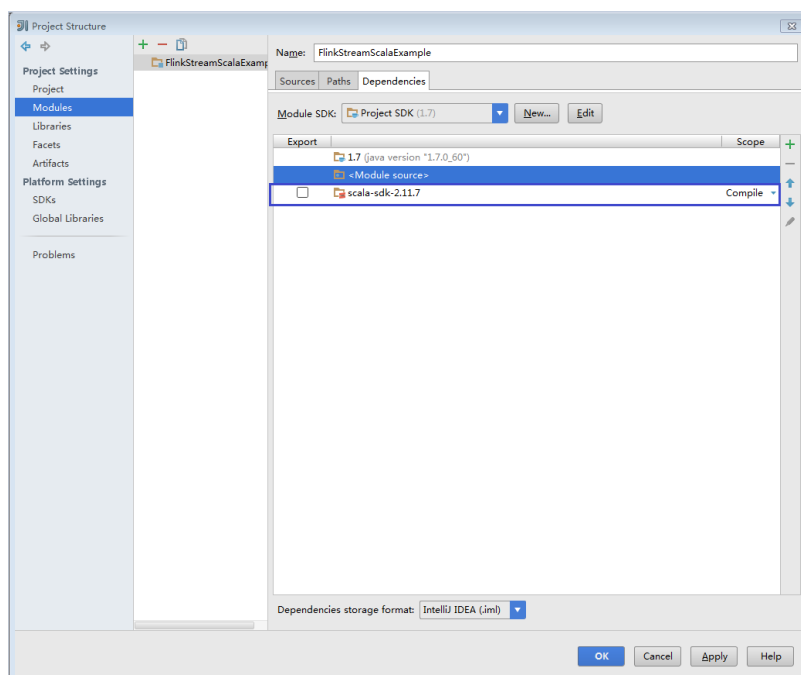
- c. On the **Scala SDK files** page, select the **scala sdk** directory and click **OK**.

Figure 11-27 Scala SDK files



5. Click **OK**.

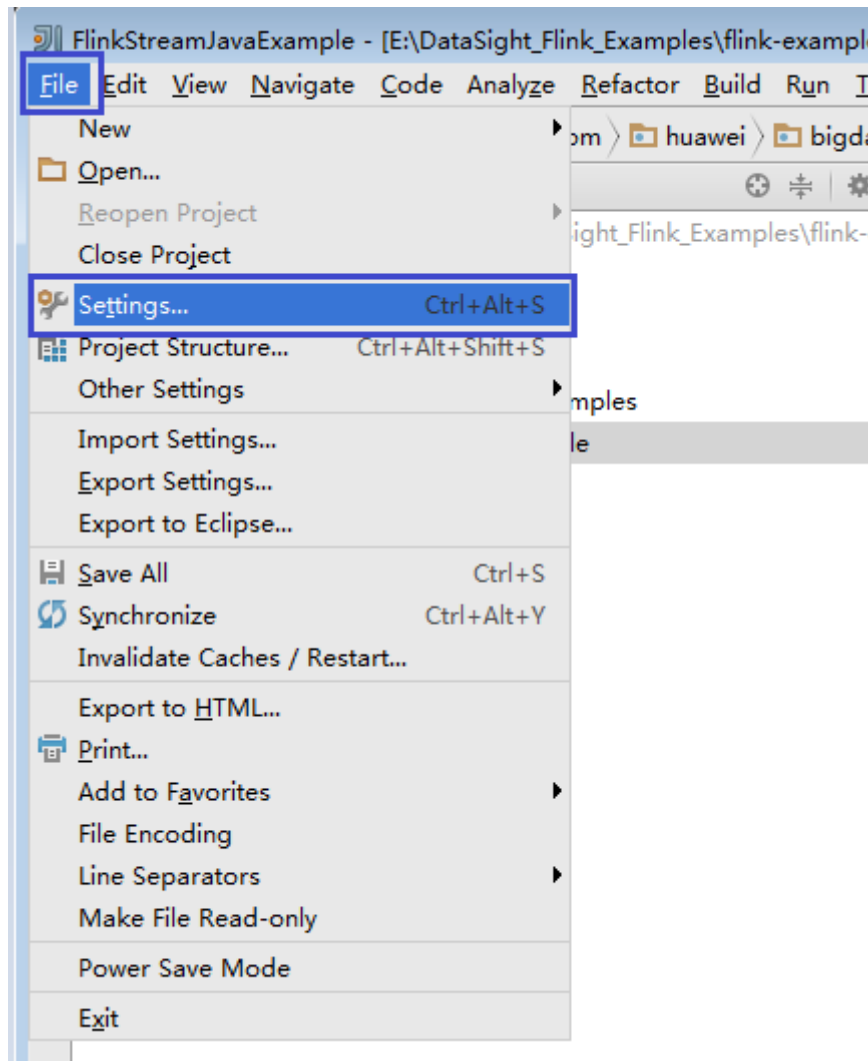
Figure 11-28 Configuration successful



Step 7 Configure the text file encoding format of IDEA to prevent garbled characters.

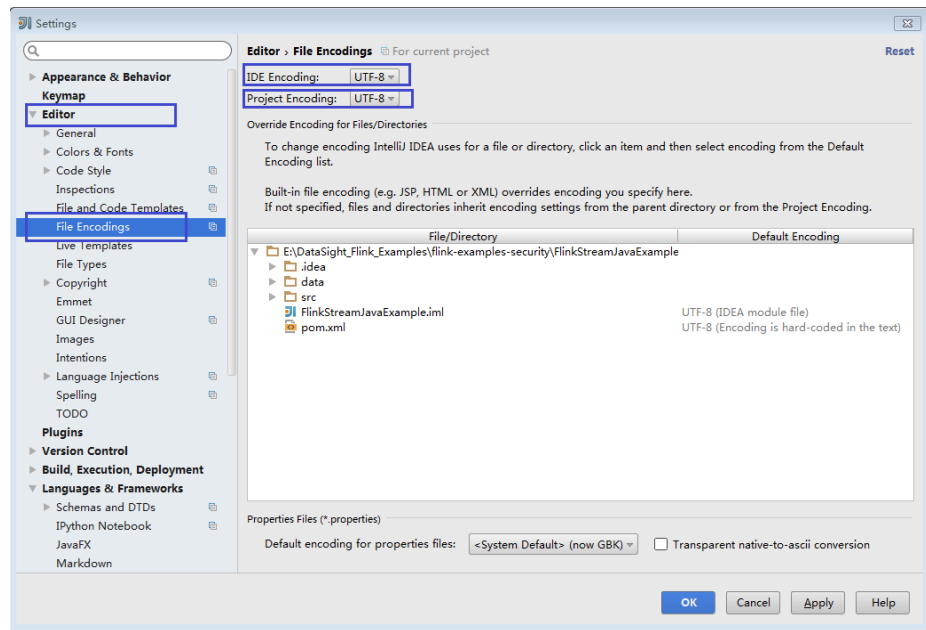
1. On the IDEA home page, choose **File > Settings...**

Figure 11-29 Choosing Settings



2. Configure encoding.
 - a. On the **Settings** page, unfold **Editor**, and choose **File Encodings**.
 - b. Select **UTF-8** from the **IDE Encoding** and **Project Encoding** drop-down lists on the right.
 - c. Click **Apply**.
 - d. Click **OK** to complete the encoding settings.

Figure 11-30 Settings



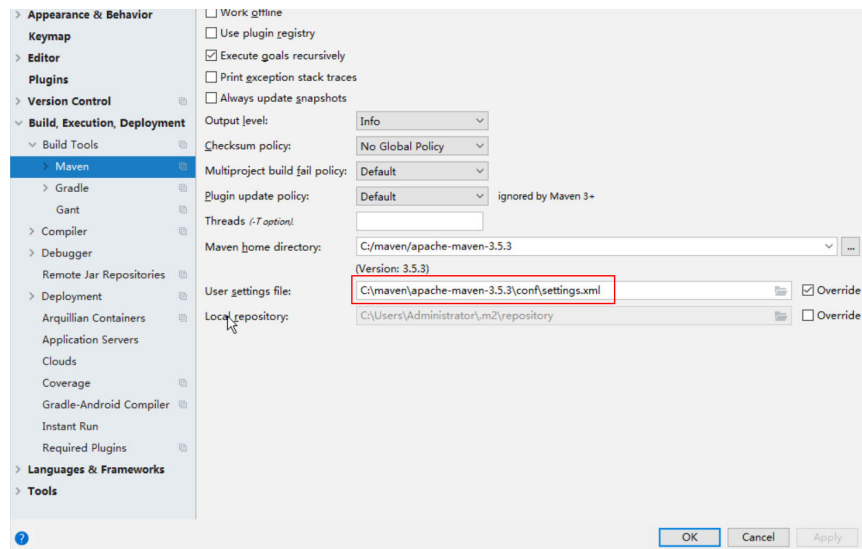
NOTE

- Obtain related dependency packages from the Flink server installation directory.
- Obtain Kafka dependency packages from the Kafka environment.
- For details about the dependency packages, see [Reference Information](#).

Step 8 Configure Maven.

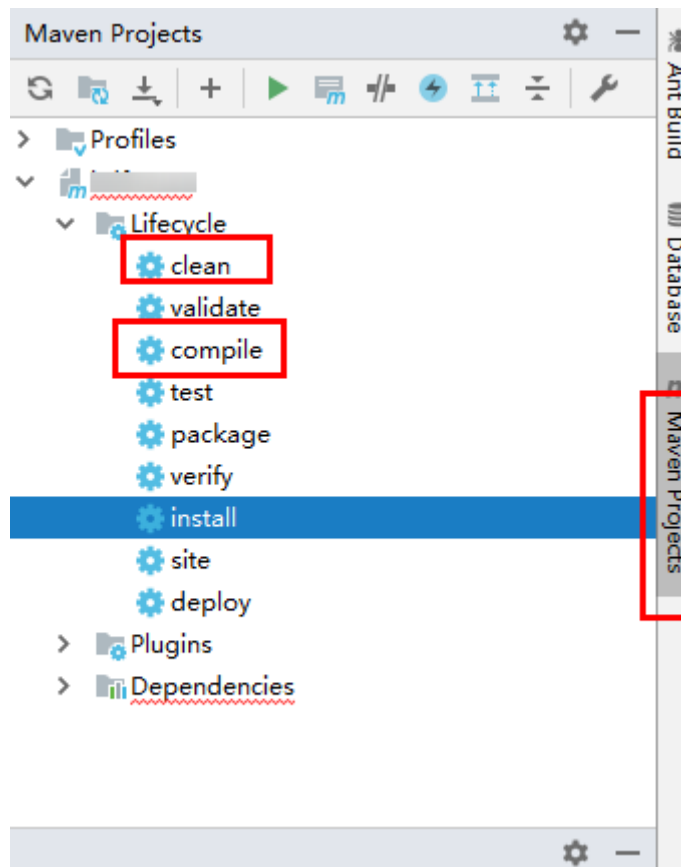
1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is *<Local Maven installation directory>\conf\settings.xml*.

Figure 11-31 Directory for storing the settings.xml file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply**, and then click **OK**.
5. On the right of the IntelliJ IDEA home page, click **Maven Projects**. On the **Maven Projects** page, choose *Project name* > **Lifecycle** and run the **clean** and **compile** scripts.

Figure 11-32 Maven Projects page



----End

Reference Information

The **lib** and **opt** directories of the Flink client contain Flink JAR files. By default, the **lib** directory contains the Flink core JAR file, and the **opt** directory contains the JAR file (for example, **flink-connector-kafka*.jar**) for connecting to external components. If it is required during application development, manually copy the related JAR files to the **lib** directory.

The dependency packages of the sample projects provided by Flink are as follows:

- Sample project of the DataStream application (in Java or Scala)
 - `flink-dist_*.jar`

NOTE

flink-dist_*.jar can be obtained from **lib** in the installation directory of the Flink client or server.

- Sample projects of the application for producing and consuming data in Kafka (in Java or Scala)
 - `kafka-clients-*.jar`
 - `flink-connector-kafka_*.jar`

 NOTE

- **flink-connector-kafka_*.jar** can be obtained from **opt** in the installation directory of the Flink client or server.
- **kafka-clients-*.jar** is provided by Kafka and can be obtained from the **lib** directory in the installation directory of the Kafka client or server.
- Sample project of the asynchronous checkpoint mechanism application (in Java or Scala)
 - `flink-dist_*.jar`
- Sample projects of pipeline (in Java or Scala)
 - `flink-connector-netty_*.jar`
 - `flink-dist_*.jar`
 - `flink-shaded-curator-*.jar`
 - `curator-client-2.12.0.jar`
 - `curator-framework-2.12.0.jar`

 NOTE

- **flink-shaded-curator-*.jar** can be obtained from **opt** in the installation directory of the Flink client or server.
- **flink-connector-netty_*.jar**, **curator-client-2.12.0.jar**, and **curator-framework-2.12.0.jar** can be obtained from the **lib** folder generated after the secondary development sample code is compiled.
- **flink-shaded-curator-*.jar** applies only to MRS 3.0.X clusters.
- **curator-client-2.12.0.jar** and **curator-framework-2.12.0.jar** apply only to MRS 3.1.X clusters.
- Sample projects of Stream SQL JOIN (in Java)
 - `kafka-clients-*.jar`
 - `flink-connector-kafka_2.11*.jar`
 - `flink-connector-kafka-base_*.jar`
 - `flink-connector-kafka_*.jar`
 - `flink-dist_2.11*.jar`
 - `flink-table_2.11*.jar`

 NOTE

- **flink-connector-kafka-base_*.jar** and **flink-connector-kafka_*.jar** can be obtained from the **opt** directory in the installation directory of the Flink client or server.
- **flink-connector-kafka-base_*.jar** and **flink-connector-kafka_*.jar** apply only to MRS 3.0.X clusters.
- **flink-connector-kafka_2.11*.jar**, **flink-dist_2.11*.jar**, and **flink-table_2.11*.jar** apply only to MRS 3.1.X clusters.

11.2.3 Creating a Project (Optional)

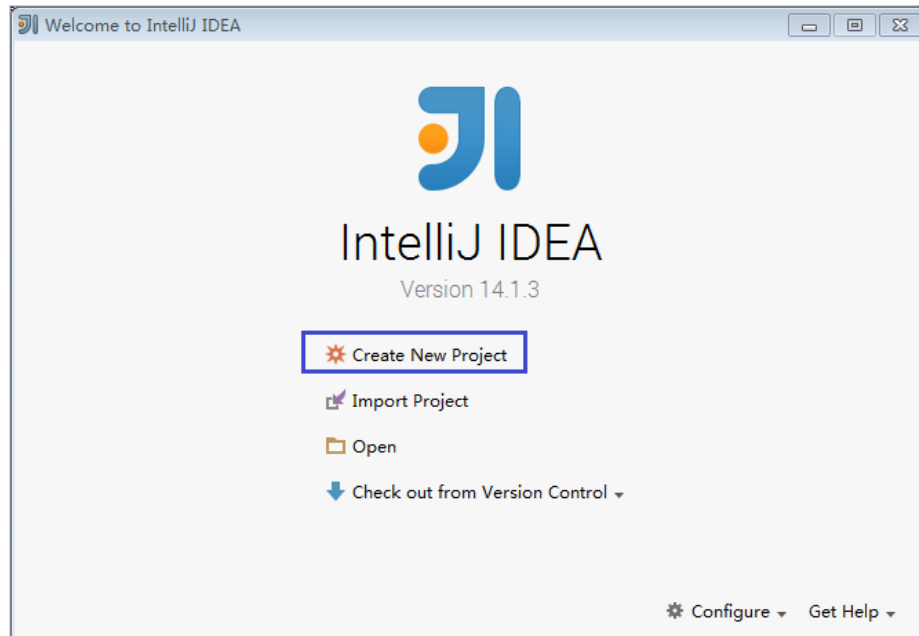
Scenarios

IntelliJ IDEA can be used to create a Flink project. A Java project is used as an example to illustrate how to create a Flink project.

Procedure

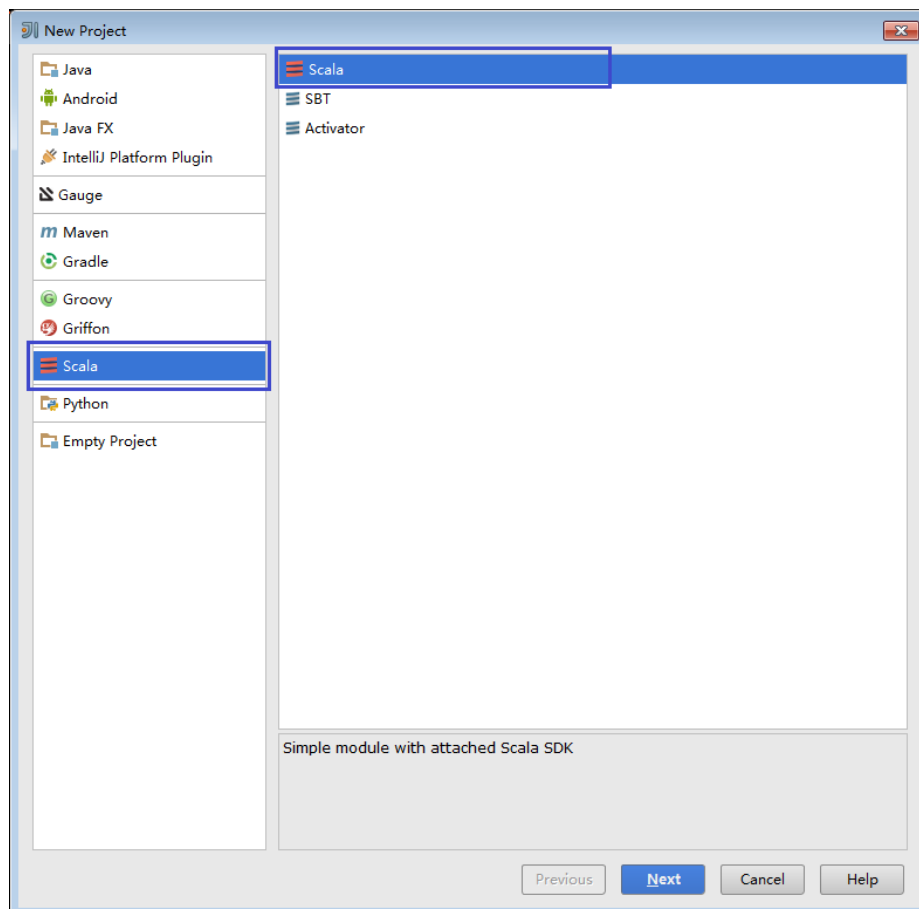
Step 1 Start the IDEA and choose **Create New Project**.

Figure 11-33 Creating a project



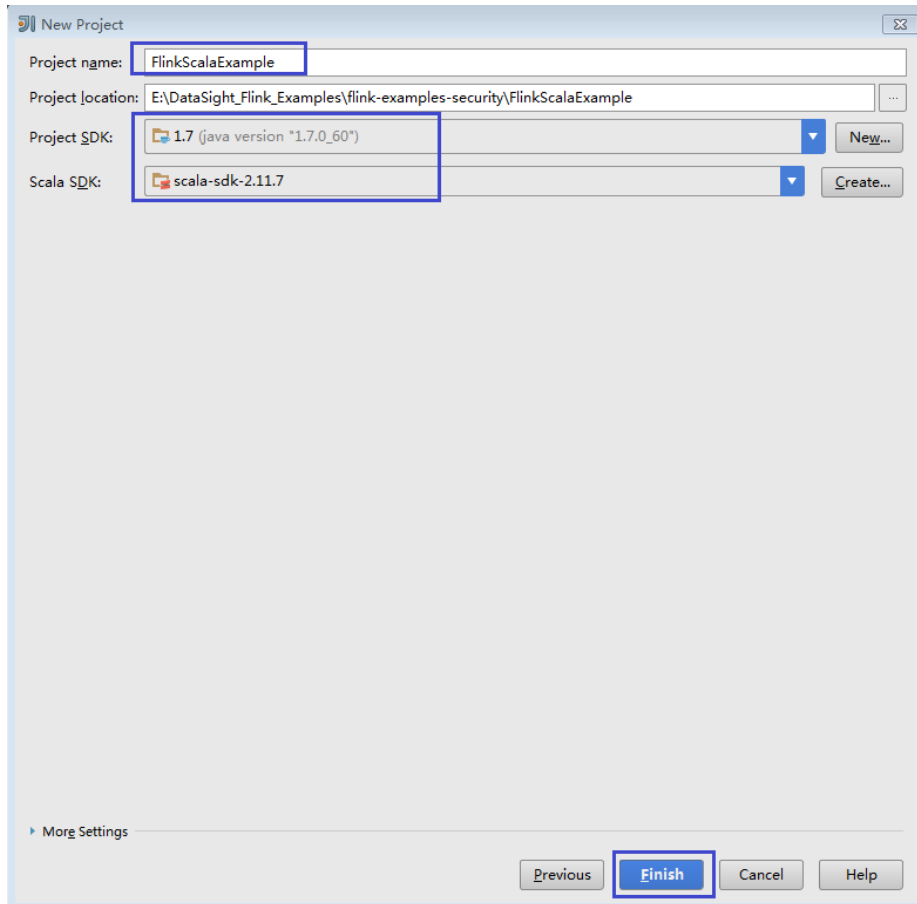
Step 2 On the **New Project** page, choose a Scala development environment, choose **Scala Module**, and then click **Next**. If you want to create a Java project, choose corresponding parameters of Java.

Figure 11-34 Selecting a development environment



Step 3 On the New Project page, enter the project name and project location, select the JDK version and Scala SDK, and then click **Finish**.

Figure 11-35 Filling in project information



----End

11.3 Developing an Application

11.3.1 DataStream Application

11.3.1.1 Scenarios

Scenarios

Develop a DataStream application of Flink to perform the following operations on logs about dwell durations of netizens for shopping online.

NOTE

The DataStream application can run in both the Windows environment and the Linux environment.

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours in total in a real time manner.

- The first column in the log file records names, the second column records genders, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

log1.txt: logs collected on Saturday. The log file can be obtained from the **data** directory of the sample project.

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

log2.txt: logs collected on Sunday. The log file can be obtained from the **data** directory of the sample project.

```
LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
FangBo,female,50
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
CaiXuyu,female,50
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
FangBo,female,50
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

Data Planning

Data of DataStream sample project is stored in a **.txt** file.

Place the **log1.txt** and **log2.txt** in two directories, for example, **/opt/log1.txt** and **/opt/log2.txt**.

NOTE

- If the data file is stored in the local file system, the data file must be stored in the specified directory on all nodes where Yarn NodeManager is deployed, and the running user access permission must be set.
- Alternatively, store the data file on HDFS and set the file read path in the program to the HDFS path, for example, **hdfs://hacluster/path/to/file**.

Development Approach

Collect the information about female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

1. Read text data, generate DataStreams, and parse data to generate UserRecord information.
2. Filter the data about the time that female netizens spend online.

3. Perform keyby operation based on the name and gender, and collect the time that female netizens spend online within a time window.
4. Filter data about users whose consecutive online duration exceeds the threshold, and obtain the result.

11.3.1.2 Java Sample Code

Function Description

Collect the information about female netizens who continuously spend more than 2 hour in online shopping and print the result.

DataStream FlinkStreamJavaExampleSample Code

The following code segment is an example. For details, see [com.huawei.bigdata.flink.examples.FlinkStreamJavaExample](#).

```
//Parameter description:
//<filePath> indicates paths where text is read. Paths are separated by commas (.).
//<windowTime> indicates the period covered by statistics window. The unit is minute.
public class FlinkStreamJavaExample {
    public static void main(String[] args) throws Exception {
        //Print reference command for executing flink run.
        System.out.println("use command as: ");
        System.out.println("./bin/flink run --class
com.huawei.bigdata.flink.examples.FlinkStreamJavaExample /opt/test.jar --filePath /opt/log1.txt,/opt/
log2.txt --windowTime 2");
        System.out.println("*****");
        System.out.println("<filePath> is for text file to read data, use comma to separate");
        System.out.println("<windowTime> is the width of the window, time as minutes");
        System.out.println("*****");

        //Paths where text is read. Paths are separated by commas (,)
        final String[] filePaths = ParameterTool.fromArgs(args).get("filePath", "/opt/log1.txt,/opt/
log2.txt").split(",");
        assert filePaths.length > 0;

        //windowTime specifies the size of the time window. By default, a window with 2 minutes as the size
can read all data in a text file.
        final int windowTime = ParameterTool.fromArgs(args).getInt("windowTime", 2);

        //Build the execution environment and run eventTime to process window data.
        final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
        env.setParallelism(1);

        //Read DataStream of the text.
        DataStream<String> unionStream = env.readTextFile(filePaths[0]);
        if (filePaths.length > 1) {
            for (int i = 1; i < filePaths.length; i++) {
                unionStream = unionStream.union(env.readTextFile(filePaths[i]));
            }
        }

        //Convert data, build the logic for the entire data process, calculate, and print results.
        unionStream.map(new MapFunction<String, UserRecord>() {
            @Override
            public UserRecord map(String value) throws Exception {
                return getRecord(value);
            }
        }).assignTimestampsAndWatermarks(
            new Record2TimestampExtractor()
        ).filter(new FilterFunction<UserRecord>() {
            @Override
```

```
        public boolean filter(UserRecord value) throws Exception {
            return value.sexy.equals("female");
        }
    }).keyBy(
        new UserRecordSelector()
    ).window(
        TumblingEventTimeWindows.of(Time.minutes(windowTime))
    ).reduce(new ReduceFunction<UserRecord>() {
        @Override
        public UserRecord reduce(UserRecord value1, UserRecord value2)
            throws Exception {
            value1.shoppingTime += value2.shoppingTime;
            return value1;
        }
    }).filter(new FilterFunction<UserRecord>() {
        @Override
        public boolean filter(UserRecord value) throws Exception {
            return value.shoppingTime > 120;
        }
    }).print();

    //Call execute to trigger the execution.
    env.execute("FemaleInfoCollectionPrint java");
}

//Build the keyword of keyBy as the grouping basis.
private static class UserRecordSelector implements KeySelector<UserRecord, Tuple2<String, String>> {
    @Override
    public Tuple2<String, String> getKey(UserRecord value) throws Exception {
        return Tuple2.of(value.name, value.sexy);
    }
}

//Parse the row data of the text and build UserRecord data structure.
private static UserRecord getRecord(String line) {
    String[] elems = line.split(",");
    assert elems.length == 3;
    return new UserRecord(elems[0], elems[1], Integer.parseInt(elems[2]));
}

//Defines UserRecord data structure and rewrite toString printing method.
public static class UserRecord {
    private String name;
    private String sexy;
    private int shoppingTime;

    public UserRecord(String n, String s, int t) {
        name = n;
        sexy = s;
        shoppingTime = t;
    }

    public String toString() {
        return "name: " + name + " sexy: " + sexy + " shoppingTime: " + shoppingTime;
    }
}

//Build class that inherits AssignerWithPunctuatedWatermarks to configure eventTime and waterMark.
private static class Record2TimestampExtractor implements
AssignerWithPunctuatedWatermarks<UserRecord> {

    //add tag in the data of datastream elements
    @Override
    public long extractTimestamp(UserRecord element, long previousTimestamp) {
        return System.currentTimeMillis();
    }
}

//give the watermark to trigger the window to execute, and use the value to check if the window
elements are ready
```

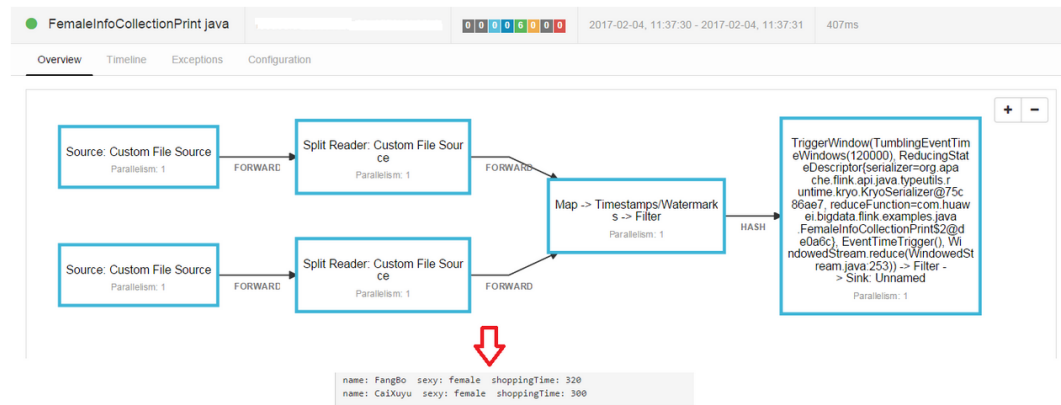
```
@Override
public Watermark checkAndGetNextWatermark(UserRecord element, long extractedTimestamp) {
    return new Watermark(extractedTimestamp - 1);
}
}
```

The following is the command output:

```
name: FangBo sexy: female shoppingTime: 320
name: CaiXuyu sexy: female shoppingTime: 300
```

Figure 11-36 shows the process of execution.

Figure 11-36 Process of execution



11.3.1.3 Scala Sample Code

Function Description

Collect the information about female netizens who continuously spend more than 2 hour in online shopping and print the result.

DataStream FlinkStreamScalaExampleSample Code

The following code is an example. For details, see `com.huawei.bigdata.flink.examples.FlinkStreamScalaExample`.

```
//Parameter description
//filePath indicates paths where text is read. Paths are separated by commas (.).
//windowTime indicates the period covered by statistics window. The unit is minute.
object FlinkStreamScalaExample {
def main(args: Array[String]) {
    //Print the reference command used to execute flink run.
    System.out.println("use command as:")
    System.out.println("./bin/flink run --class
com.huawei.bigdata.flink.examples.FlinkStreamScalaExample /opt/test.jar --filePath /opt/log1.txt,/opt/
log2.txt --windowTime 2")
    System.out.println("*****")
    System.out.println("<filePath> is for text file to read data, use comma to separate")
    System.out.println("<windowTime> is the width of the window, time as minutes")
    System.out.println("*****")

    //Paths where text is read. Paths are separated by commas (.)
    val filePaths = ParameterTool.fromArgs(args).get("filePath",
"/opt/log1.txt,/opt/log2.txt").split(",").map(_.trim)
```

```
assert(filePaths.length > 0)

//windowTime specifies the size of the time window. By default, a window with 2 minutes as the size can
read all data in a text file.
val windowTime = ParameterTool.fromArgs(args).getInt("windowTime", 2)

//Build the execution environment and run eventTime to process window data.
val env = StreamExecutionEnvironment.getExecutionEnvironment
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
env.setParallelism(1)
//Read DataStream of the text.
val unionStream = if (filePaths.length > 1) {
  val firstStream = env.readTextFile(filePaths.apply(0))
  firstStream.union(filePaths.drop(1).map(it => env.readTextFile(it)): _)
} else {
  env.readTextFile(filePaths.apply(0))
}

//Convert data, build the logic for the entire data process, calculate, and print results.
unionStream.map(getRecord(_))
  .assignTimestampsAndWatermarks(new Record2TimestampExtractor)
  .filter(_._sexy == "female")
  .keyBy("name", "sexy")
  .window(TumblingEventTimeWindows.of(Time.minutes(windowTime)))
  .reduce((e1, e2) => UserRecord(e1.name, e1.sexy, e1.shoppingTime + e2.shoppingTime))
  .filter(_._shoppingTime > 120).print()

//Call execute to trigger the execution
env.execute("FemaleInfoCollectionPrint scala")
}

//Parse the row data of the text and build UserRecord data structure.
def getRecord(line: String): UserRecord = {
  val elems = line.split(",")
  assert(elems.length == 3)
  val name = elems(0)
  val sexy = elems(1)
  val time = elems(2).toInt
  UserRecord(name, sexy, time)
}

//Defines UserRecord data structure.
case class UserRecord(name: String, sexy: String, shoppingTime: Int)

//Build class that inherits AssignerWithPunctuatedWatermarks to configure eventTime and waterMark.
private class Record2TimestampExtractor extends AssignerWithPunctuatedWatermarks[UserRecord] {

  //add tag in the data of datastream elements
  override def extractTimestamp(element: UserRecord, previousTimestamp: Long): Long = {
    System.currentTimeMillis()
  }

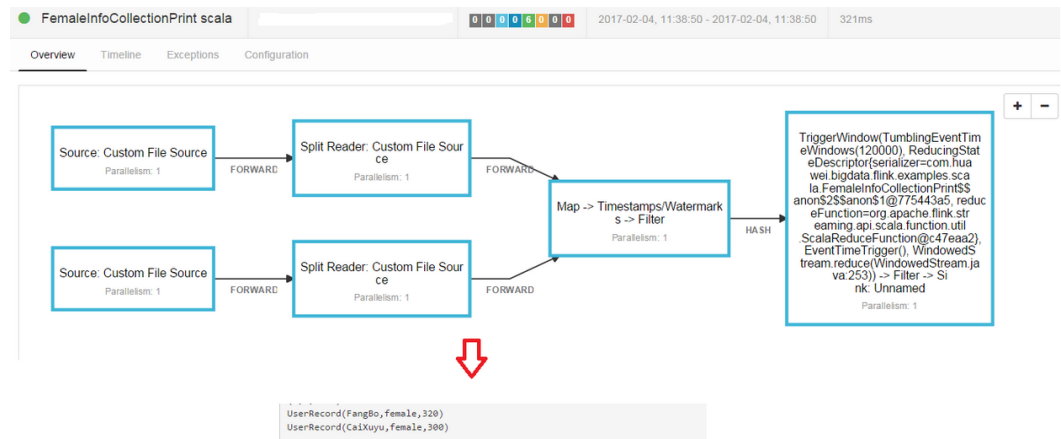
  //give the watermark to trigger the window to execute, and use the value to check if the window
elements are ready
  def checkAndGetNextWatermark(lastElement: UserRecord,
    extractedTimestamp: Long): Watermark = {
    new Watermark(extractedTimestamp - 1)
  }
}
}
```

The following is the command output:

```
UserRecord(FangBo,female,320)
UserRecord(CaiXuyu,female,300)
```

Figure 11-37 shows the process of execution.

Figure 11-37 Process of execution



11.3.2 Interconnecting with Kafka

11.3.2.1 Scenarios

Scenarios

Assume that a Flink service receives one word record every 1 second.

Develop a Flink application that can generate output of prefixed message contents.

Data Planning

Sample project data of Flink is stored in Kafka. A user with Kafka permission can send data to Kafka and receive data from it.

1. Ensure that clusters, including HDFS, YARN, Flink, and Kafka are successfully installed.
2. Create a topic.

The format of the command is following:

```
bin/kafka-topics.sh --create --zookeeper {zkQuorum}/kafka --partitions {partitionNum} --replication-factor {replicationNum} --topic {Topic}
```

Table 11-5

parameter	Description
{zkQuorum}	ZooKeeper cluster information. The format is IP:port.
{PartitionNum}	The number of partitions for the topic.
{ReplicationNum}	The number of copies of each partition for the topic.

parameter	Description
{Topic}	The topic name.

Assume that the IP:ports of ZooKeeper clusters are 10.96.101.32:2181, 10.96.101.251:2181, 10.96.101.177:2181, and 10.91.8.160:2181, and the topic named is topic1. The command for creating a topic is as follows:

```
bin/kafka-topics.sh --create --zookeeper
10.96.101.32:2181,10.96.101.251:2181,10.96.101.177:2181,10.91.8.160:2181/kafka --partitions 5 --
replication-factor 1 --topic topic1
```

Development Approach

1. Start the Flink Kafka Producer to send data to Kafka.
2. Start Flink Kafka Consumer to receive data from Kafka. Ensure that topics of Kafka Consumer are consistent with that of Kafka Producer.
3. Add prefix to the data content and print the result.

11.3.2.2 Java Sample Code

Function Description

In a Flink application, call API of the **flink-connector-kafka** module to produce and consume data.

Sample Code

Following is the main logic code of Kafka Consumer and Kafka Producer.

For the complete code, see `com.huawei.bigdata.flink.examples.WriteIntoKafka` and `com.huawei.bigdata.flink.examples.ReadFromKafka`.

```
//producer code
public class WriteIntoKafka {
    public static void main(String[] args) throws Exception {
        //Print the reference command of flink run.
        System.out.println("use command as: ");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test --bootstrap.servers 10.91.8.218:9092");
        System.out.println("*****");
        System.out.println("<topic> is the kafka topic name");
        System.out.println("<bootstrap.servers> is the ip:port list of brokers");
        System.out.println("*****");

        //Build the execution environment.
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        //Configure the parallelism.
        env.setParallelism(1);
        //Parse the execution parameter.
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        //Build the StreamGraph and write data generated by customized source into Kafka.
        DataStream<String> messageStream = env.addSource(new SimpleStringGenerator());
        messageStream.addSink(new FlinkKafkaProducer<>(paraTool.get("topic"),
            new SimpleStringSchema(),
            paraTool.getProperties()));
        //Call execute to trigger the execution.
        env.execute();
    }
}
```

```
//Customize source to continuously generate messages every one second.
public static class SimpleStringGenerator implements SourceFunction<String> {
    private static final long serialVersionUID = 2174904787118597072L;
    boolean running = true;
    long i = 0;

    @Override
    public void run(SourceContext<String> ctx) throws Exception {
        while (running) {
            ctx.collect("element-" + (i++));
            Thread.sleep(1000);
        }
    }

    @Override
    public void cancel() {
        running = false;
    }
}

//consumer code
public class ReadFromKafka {
    public static void main(String[] args) throws Exception {
        //Print the reference command of flink run.
        System.out.println("use command as: ");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:9092");
        System.out.println("*****");
        System.out.println("<topic> is the kafka topic name");
        System.out.println("<bootstrap.servers> is the ip:port list of brokers");
        System.out.println("*****");

        //Build the execution environment.
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        //Configure the parallelism.
        env.setParallelism(1);
        //Parse the execution parameter.
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        //Build the StreamGraph, read data from Kafka and print the result in another row.
        DataStream<String> messageStream = env.addSource(new FlinkKafkaConsumer<>(paraTool.get("topic"),
            new SimpleStringSchema(),
            paraTool.getProperties()));
        messageStream.rebalance().map(new MapFunction<String, String>() {
            @Override
            public String map(String s) throws Exception {
                return "Flink says " + s + System.getProperty("line.separator");
            }
        }).print();
        //Call execute to trigger the execution.
        env.execute();
    }
}
```

11.3.2.3 Scala Sample Code

Function Description

In a Flink application, call API of the **flink-connector-kafka** module to produce and consume data.

Sample Code

Following is the main logic code of Kafka Consumer and Kafka Producer.

For the complete code, see `com.huawei.bigdata.flink.examples.WriteIntoKafka` and `com.huawei.bigdata.flink.examples.ReadFromKafka`.

```
//Code of producer
object WriteIntoKafka {
  def main(args: Array[String]) {
    //Print the reference command of flink run.
    System.out.println("use command as: ")
    System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
      " /opt/test.jar --topic topic-test --bootstrap.servers 10.91.8.218:9092")
    System.out.println("*****")
    System.out.println("<topic> is the kafka topic name")
    System.out.println("<bootstrap.servers> is the ip:port list of brokers")
    System.out.println("*****")

    //Build the execution environment.
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    //Configure the parallelism.
    env.setParallelism(1)
    //Parse the execution parameter.
    val paraTool = ParameterTool.fromArgs(args)
    //Build the StreamGraph and write data generated by customized source into Kafka.
    val messageStream: DataStream[String] = env.addSource(new SimpleStringGenerator)
    messageStream.addSink(new FlinkKafkaProducer(
      paraTool.get("topic"), new SimpleStringSchema, paraTool.getProperties))
    //Call execute to trigger the execution.
    env.execute
  }
}

//Customize source to continuously generate messages every one second.
class SimpleStringGenerator extends SourceFunction[String] {
  var running = true
  var i = 0

  override def run(ctx: SourceContext[String]) {
    while (running) {
      ctx.collect("element-" + i)
      i += 1
      Thread.sleep(1000)
    }
  }

  override def cancel() {
    running = false
  }
}

//consumer code
object ReadFromKafka {
  def main(args: Array[String]) {
    //Print the reference command of flink run.
    System.out.println("use command as: ")
    System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka" +
      " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:9092")
    System.out.println("*****")
    System.out.println("<topic> is the kafka topic name")
    System.out.println("<bootstrap.servers> is the ip:port list of brokers")
    System.out.println("*****")

    //Build the execution environment
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    //Configure the parallelism.
    env.setParallelism(1)
    //Parse the execution parameter.
    val paraTool = ParameterTool.fromArgs(args)
    //Build the StreamGraph, read data from Kafka and print the result in another row.
    val messageStream = env.addSource(new FlinkKafkaConsumer(
      paraTool.get("topic"), new SimpleStringSchema, paraTool.getProperties))
  }
}
```

```
messageStream
  .map(s => "Flink says " + s + System.getProperty("line.separator")).print()
  //Call execute to trigger the execution
  env.execute()
}
```

11.3.3 Asynchronous Checkpoint Mechanism

11.3.3.1 Scenarios

Scenarios

Assume that you want to collect data volume in the window covering preceding 4 seconds at the interval of one second, and achieve strict consistency of status. In this case, when the application is recovered from failure, all operator statuses are the same.

Data Planning

1. Customized operators generate about 10000 pieces of data per second.
2. Generated data is of four tuples (Long, String, String, Integer).
3. Statistic results are printed on the devices.
4. Printed data is of the long type.

Development Approach

1. The source operator sends 10000 pieces of data and injects the data to the window operator every second.
2. The window operator calculates the data volume of preceding 4 seconds at the interval of one second.
3. The statistics is printed to the device at the interval of one second. Please refer to the specific [Viewing the Debugging Result](#)
4. The checkpoint is triggered at the interval of 6 seconds and the checkpoint result is stored in HDFS.

11.3.3.2 Java Sample Code

Function Description

Assume that you want to collect data volume in the window covering preceding 4 seconds at the interval of one second, and achieve strict consistency of status.

Sample Code

1. Snapshot data
The snapshot data is used to store number of data pieces recorded by operators during creation of snapshots.

```
import java.io.Serializable;

//The class is a part of the snapshot and is used to store user-defined statuses.
```

```
public class UDFState implements Serializable {
    private long count;

    //Initialize user-defined statuses.
    public UDFState() {
        count = 0L;
    }

    //Configure self-defined statuses.
    public void setState(long count) {
        this.count = count;
    }

    //Obtain user-defined statuses.
    public long getState() {
        return this.count;
    }
}
```

2. Data source with checkpoints

Code of the source operator. The code can be used to send 10000 pieces after each pause of one second. When a snapshot is created, number of sent data pieces is recorded in UDFState. When the snapshot is used for restoration, the number of sent data pieces recorded in UDFState is read and assigned to the count variable.

```
import org.apache.flink.api.java.tuple.Tuple4;
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed;
import org.apache.flink.streaming.api.functions.source.RichSourceFunction;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

//The class is the source operator with checkpoint.
public class SEventSourceWithChk extends RichSourceFunction<Tuple4<Long, String, String, Integer>>
implements ListCheckpointed<UDFState> {
    private Long count = 0L;
    private boolean isRunning = true;
    private String alphabet =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyz
xyzABCDEFGHJKLMNOPQRSTUVWXYZ0987654321";

    //The main logic of the operator is to inject 10000 tuples to the StreamGraph.
    public void run(SourceContext<Tuple4<Long, String, String, Integer>> ctx) throws Exception {
        Random random = new Random();
        while(isRunning) {
            for (int i = 0; i < 10000; i++) {
                ctx.collect(Tuple4.of(random.nextLong(), "hello-" + count, alphabet, 1))
                count++;
            }
            Thread.sleep(1000);
        }
    }

    //Call this when the task is canceled.
    public void cancel() {
        isRunning = false;
    }

    //Customize a snapshot.
    public List<UDFState> snapshotState(long l, long ll) throws Exception {
        UDFState udfState = new UDFState();
        List<UDFState> listState = new ArrayList<UDFState>();
        udfState.setState(count);
        listState.add(udfState);
        return listState;
    }
}
```

```
//Restore data from customized snapshots.
public void restoreState(List<UDFState> list) throws Exception {
    UDFState udfState = list.get(0);
    count = udfState.getState();
}
}
```

3. Definition of window with checkpoint.

This code is about the window operator and is used to calculate number or tuples in the window.

```
import org.apache.flink.api.java.tuple.Tuple;
import org.apache.flink.api.java.tuple.Tuple4;
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed;
import org.apache.flink.streaming.api.functions.windowing.WindowFunction;
import org.apache.flink.streaming.api.windowing.windows.TimeWindow;
import org.apache.flink.util.Collector;

import java.util.ArrayList;
import java.util.List;

//The class is the window operator with checkpoint.
public class WindowStatisticWithChk implements WindowFunction<Tuple4<Long, String, String, Integer>, Long, Tuple, TimeWindow>, ListCheckpointed<UDFState> {
    private Long total = 0L;

    //The implementation logic of the window operator, which is used to calculate the number of tuples in a window.
    void apply(Tuple key, TimeWindow window, Iterable<Tuple4<Long, String, String, Integer>> input, Collector<Long> out) throws Exception {
        long count = 0L;
        for (Tuple4<Long, String, String, Integer> event : input) {
            count++;
        }
        total += count;
        out.collect(count);
    }

    //Customize snapshot.
    public List<UDFState> snapshotState(Long l, Long ll) {
        List<UDFState> listState = new ArrayList<UDFState>();
        UDFState udfState = new UDFState();
        udfState.setState(total);
        listState.add(udfState);
        return listState;
    }

    //Restore data from customized snapshots.
    public void restoreState(List<UDFState> list) throws Exception {
        UDFState udfState = list.get(0);
        total = udfState.getState();
    }
}
```

4. Application code

The code is about the definition of StreamGraph and is used to implement services. The proctime is used as the timestamp for triggering the window.

```
import org.apache.flink.runtime.state.filesystem.FsStateBackend;
import org.apache.flink.streaming.api.CheckpointingMode;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.windowing.assigners.SlidingProcessingTimeWindows;
import org.apache.flink.streaming.api.windowing.time.Time;

public class FlinkProcessingTimeAPIChkMain {
    public static void main(String[] args) throws Exception{

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        //Set configurations and enable checkpoint.
```

```
env.setStateBackend(new FsStateBackend("hdfs://hacluster/flink/checkpoint/"));
env.getCheckpointConfig.setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE);
env.getCheckpointConfig.setCheckpointInterval(6000);

//Application logic.
env.addSource(new SEventSourceWithChk())
    .keyBy(0)
    .window(SlidingProcessingTimeWindows.of(Time.seconds(4), Time.seconds(1)))
    .apply(new WindowStatisticWithChk())
    .print()

env.execute();
}
```

11.3.3.3 Scala Sample Code

Function Description

Assume that you want to collect data volume in the window covering preceding 4 seconds at the interval of one second, and achieve strict consistency of status.

Sample Code

1. Formats of sent data.

```
case class SEvent(id: Long, name: String, info: String, count: Int)
```

2. Snapshot data

The snapshot data is used to store number of data pieces recorded by operators during creation of snapshots.

```
//User-defined statuses.
```

```
class UDFState extends Serializable{
    private var count = 0L
```

```
    //Configure user-defined statuses.
    def setState(s: Long) = count = s
```

```
    //Obtain user-defined statuses.
    def getState = count
}
```

3. Data source with checkpoints

Code of the source operator. The code can be used to send 10000 pieces after each pause of one second. When a snapshot is created, number of sent data pieces is recorded in UDFState. When the snapshot is used for restoration, the number of sent data pieces recorded in UDFState is read and assigned to the *count* variable.

```
import java.util
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed
import org.apache.flink.streaming.api.functions.source.RichSourceFunction
import org.apache.flink.streaming.api.functions.source.SourceFunction.SourceContext
```

```
//The class is the source operator with checkpoint.
```

```
class SEventSourceWithChk extends RichSourceFunction[SEvent] with ListCheckpointed[UDFState]{
    private var count = 0L
    private var isRunning = true
    private val alphabet =
"abcdefghijklmnopqrstuvwxyABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321"
```

```
    //The logic of the operator is to inject 10000 tuples to the StreamGraph.
    override def run(sourceContext: SourceContext[SEvent]): Unit = {
```



```

while(isRunning) {
  for (i <- 0 until 10000) {
    sourceContext.collect(SEvent(1, "hello-"+count, alphabet,1))
    count += 1L
  }
  Thread.sleep(1000)
}

//Call this when the task is canceled.
override def cancel(): Unit = {
  isRunning = false;
}

override def close(): Unit = super.close()

//Create a snapshot.
override def snapshotState(l: Long, l1: Long): util.List[UDFState] = {
  val udfList: util.ArrayList[UDFState] = new util.ArrayList[UDFState]
  val udfState = new UDFState
  udfState.setState(count)
  udfList.add(udfState)
  udfList
}

//Obtain status from the snapshot.
override def restoreState(list: util.List[UDFState]): Unit = {
  val udfState = list.get(0)
  count = udfState.getState
}
}

```

4. Definition of window with checkpoint.

This code is about the window operator and is used to calculate number or tuples in the window.

```

import java.util
import org.apache.flink.api.java.tuple.Tuple
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed
import org.apache.flink.streaming.api.scala.function.WindowFunction
import org.apache.flink.streaming.api.windowing.windows.TimeWindow
import org.apache.flink.util.Collector

//The class is the window operator with checkpoint.
class WindowStatisticWithChk extends WindowFunction[SEvent, Long, Tuple, TimeWindow] with
ListCheckpointed[UDFState]{
  private var total = 0L

  //The implementation logic of the window operator, which is used to calculate the number of
  tuples in a window.
  override def apply(key: Tuple, window: TimeWindow, input: Iterable[SEvent], out: Collector[Long]):
Unit = {
    var count = 0L
    for (event <- input) {
      count += 1L
    }
    total += count
    out.collect(count)
  }

  //Customize a snapshot.
  override def snapshotState(l: Long, l1: Long): util.List[UDFState] = {
    val udfList: util.ArrayList[UDFState] = new util.ArrayList[UDFState]
    val udfState = new UDFState
    udfState.setState(total)
    udfList.add(udfState)
    udfList
  }

  //Restore data from customized snapshots.
}

```

```
override def restoreState(list: util.List[UDFState]): Unit = {  
  val udfState = list.get(0)  
  total = udfState.getState  
}  
}
```

5. Application code

The code is about the definition of StreamGraph and is used to implement services. The **event time** is used as the timestamp for triggering the window.

```
import com.huawei.rt.flink.core.{SEvent, SEventSourceWithChk, WindowStatisticWithChk}  
import org.apache.flink.contrib.streaming.state.RocksDBStateBackend  
import org.apache.flink.streaming.api.functions.AssignerWithPeriodicWatermarks  
import org.apache.flink.streaming.api.{CheckpointingMode, TimeCharacteristic}  
import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment  
import org.apache.flink.streaming.api.watermark.Watermark  
import org.apache.flink.streaming.api.windowing.assigners.SlidingEventTimeWindows  
import org.apache.flink.streaming.api.windowing.time.Time  
import org.apache.flink.api.scala._  
import org.apache.flink.runtime.state.filesystem.FsStateBackend  
import org.apache.flink.streaming.api.environment.CheckpointConfig.ExternalizedCheckpointCleanup  
  
object FlinkEventTimeAPIChkMain {  
  def main(args: Array[String]): Unit = {  
    val env = StreamExecutionEnvironment.getExecutionEnvironment  
    env.setStateBackend(new FsStateBackend("hdfs://hacluster/flink/checkpoint/"))  
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)  
    env.getConfig.setAutoWatermarkInterval(2000)  
    env.getCheckpointConfig.setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE)  
    env.getCheckpointConfig.setCheckpointInterval(6000)  
  
    //Application logic.  
    env.addSource(new SEventSourceWithChk)  
    .assignTimestampsAndWatermarks(new AssignerWithPeriodicWatermarks[SEvent] {  
      //Configure watermark.  
      override def getCurrentWatermark: Watermark = {  
        new Watermark(System.currentTimeMillis())  
      }  
      //Add timestamp for each tuple.  
      override def extractTimestamp(t: SEvent, l: Long): Long = {  
        System.currentTimeMillis()  
      }  
    })  
    .keyBy(0)  
    .window(SlidingEventTimeWindows.of(Time.seconds(4), Time.seconds(1)))  
    .apply(new WindowStatisticWithChk)  
    .print()  
    env.execute()  
  }  
}
```

11.3.4 Job Pipeline Program

11.3.4.1 Scenario

The concurrency of NettySource and the concurrency NettySink must be the same. Otherwise, the connection cannot be created.

Scenario

Assume that there are three jobs, a publisher and two subscribers. The publisher generates 10000 pieces of data each second. Data is sent by the NettySink operator from the publisher job to downstream subscribers which subscribe a same piece of data.

Data Planning

1. The publisher uses customized operators to generate about 10000 pieces of data per second.
2. The data contains two attributes, which are of integer and string types.
3. Configuration file
 - `nettyconnector.registerserver.topic.storage`: (Mandatory) Configures the path (on a third-party server) to information about IP address, port numbers, and concurrency of NettySink. For example:
`nettyconnector.registerserver.topic.storage: /flink/nettyconnector`
 - `nettyconnector.sinkserver.port.range`: (Mandatory) Configures the range of port numbers of NettySink. For example:
`nettyconnector.sinkserver.port.range: 28444-28943`
 - `nettyconnector.sinkserver.subnet`: Configure the network domain. For example:
`nettyconnector.sinkserver.subnet: 10.162.0.0/16`
4. Description of APIs

- RegisterServer API

RegisterServerHandler stores information such as IP address, port number, and concurrency of NettySink for the connection with NettySource. Following APIs are provided for users:

```
public interface RegisterServerHandler {

    /**
     * Start the RegisterServer
     *
     * @param The configuration indicates the configuration type of Flink.
     */
    void start(Configuration configuration) throws Exception;

    /**
     * Create a topic node (directory) on the RegisterServer
     *
     * @param The topic indicates the name of the topic node
     */
    void createTopicNode(String topic) throws Exception;

    /**
     * Register the information to a topic node (directory)
     *
     * @param topic @param The topic indicates the directory to be registered with
     * @param The registerRecord indicates the information to be registered
     */
    void register(String topic, RegisterRecord registerRecord) throws Exception;

    /**
     * Delete the topic node.
     * @param The topic indicates the topic to be deleted.
     */
    void deleteTopicNode(String topic) throws Exception;

    /**
     * Deregister the registration inDeregister the registration information formation
     * @param The topic indicates the topic where the registration information locates.
     * @param The recordId indicates the ID of the registration information to be deregistered
     */
    void unregister(String topic, int recordId) throws Exception;

    /**
     * Query information
     * @param Topic where the query information locates
     * @param The recordId indicates the ID of the query information
     */
}
```

```
*/  
RegisterRecord query(String topic, int recordId) throws Exception;  
/**  
 * Query whether a topic exist.  
 * @param topic  
 */  
Boolean isExist(String topic) throws Exception;  
/**  
 *Disable RegisterServerHandler.  
 */  
void shutdown() throws Exception;
```

In addition to the preceding APIs, Flink provides ZookeeperRegisterHandler.

– NettySink operator

```
Class NettySink(String name,  
String topic,  
RegisterServerHandler registerServerHandler,  
int numberOfSubscribedJobs)
```

- name: Name of a current NettySink.
- topic: The topic that generates data for the current NettySink. Different NettySinks must use different topics. Otherwise, the subscription may be disordered and data transmission may be abnormal.
- registerServerHandler: Handler of the registration server.
- numberOfSubscribedJobs: Specific number of jobs that subscribe the current NettySink. NettySink sends data only when all subscribers are connected to NettySink.

– NettySource operator

```
Class NettySource(String name,  
String topic,  
RegisterServerHandler registerServerHandler)
```

- name: name of the NettySource. The NettySource must be unique (concurrency excluded). Otherwise, connection with NettySink may be conflicted.
- topic: topic of subscribed NettySink.
- registerServerHandler: Handler of the registration server.

 **NOTE**

The concurrency of NettySource and the concurrency NettySink must be the same. Otherwise, the connection cannot be created.

Development Approach

1. There are three jobs, on publisher and two subscribers.
2. The publisher transforms generated data into byte[] and send them the subscribers.
3. After receiving the byte[], subscribers transform data into the string type and print sampled data.

11.3.4.2 Java Sample Code

Following is the main logic code for demonstration.

For details about the complete code, see the following:

- `com.huawei.bigdata.flink.examples.UserSource`.
- `com.huawei.bigdata.flink.examples.TestPipelineNettySink`.
- `com.huawei.bigdata.flink.examples.TestPipelineNettySource1`.
- `com.huawei.bigdata.flink.examples.TestPipelineNettySource2`.

1. The publisher customizes source operators to generate data.

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.streaming.api.functions.source.RichParallelSourceFunction;

import java.io.Serializable;

public class UserSource extends RichParallelSourceFunction<Tuple2<Integer, String>> implements
Serializable {

    private boolean isRunning = true;

    public void open(Configuration configuration) throws Exception {
        super.open(configuration);
    }

    /**
     * Data generation function, which is used to generate 10000 pieces of data each second.
     */
    public void run(SourceContext<Tuple2<Integer, String>> ctx) throws Exception {

        while(isRunning) {
            for (int i = 0; i < 10000; i++) {
                ctx.collect(Tuple2.of(i, "hello-" + i));
            }
            Thread.sleep(1000);
        }
    }

    public void close() {
        isRunning = false;
    }

    public void cancel() {
        isRunning = false;
    }
}
```

2. Code for the publisher:

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.netty.sink.NettySink;
import org.apache.flink.streaming.connectors.netty.util.ZookeeperRegisterServerHandler;

public class TestPipelineNettySink {

    public static void main(String[] args) throws Exception{

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        //Set the concurrency of job to 2.
```

```
env.setBufferTimeout(2);

//Create a ZookeeperRegisterServerHandler.
ZookeeperRegisterServerHandler zkRegisterServerHandler = new ZookeeperRegisterServerHandler();
// Add a customized source operator.
env.addSource(new UserSource())
    .keyBy(0)
    .map(new MapFunction<Tuple2<Integer,String>, byte[]>() {
        //Transform the to-be-sent data into a byte array.

@Override
    public byte[] map(Tuple2<Integer, String> integerStringTuple2) throws Exception {
        return integerStringTuple2.f1.getBytes();
    }
    }).addSink(new NettySink("NettySink-1", "TOPIC-2", zkRegisterServerHandler, 2));//NettySink
transmits the data.

env.execute();
}
}
```

3. Code for the first subscriber.

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.netty.source.NettySource;
import org.apache.flink.streaming.connectors.netty.util.ZookeeperRegisterServerHandler;

import java.nio.charset.Charset;

public class TestPipelineNettySource1 {

    public static void main(String[] args) throws Exception{

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        // Set the concurrency of job to 2.

env.setParallelism(2);

// Create a ZookeeperRegisterServerHandler.
ZookeeperRegisterServerHandler zkRegisterServerHandler = new ZookeeperRegisterServerHandler();
//Add a NettySource operator to receive messages from the publisher.
env.addSource(new NettySource("NettySource-1", "TOPIC-2", zkRegisterServerHandler))
    .map(new MapFunction<byte[], String>() {
        // Transform the received byte stream into character strings
@Override
    public String map(byte[] bytes) {
        return new String(bytes, Charset.forName("UTF-8"));
    }
    }).print();

env.execute();
}
}
```

4. Code for the second subscriber.

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.netty.source.NettySource;
import org.apache.flink.streaming.connectors.netty.util.ZookeeperRegisterServerHandler;

import java.nio.charset.Charset;

public class TestPipelineNettySource2 {
```

```
public static void main(String[] args) throws Exception {
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    // Set the concurrency of job to 2.
    env.setParallelism(2);

    //Create a ZookeeperRegisterServerHandler.
    ZookeeperRegisterServerHandler zkRegisterServerHandler = new ZookeeperRegisterServerHandler();
    //Add a NettySource operator to receive messages from the publisher.
    env.addSource(new NettySource("NettySource-2", "TOPIC-2", zkRegisterServerHandler))
        .map(new MapFunction<byte[], String>() {
            //Transform the received byte array into character strings.
            @Override
            public String map(byte[] bytes) {
                return new String(bytes, Charset.forName("UTF-8"));
            }
        }).print();

    env.execute();
}
```

11.3.4.3 Scala Sample Code

Following is the main logic code for demonstration.

For details about the complete code, see the following:

- com.huawei.bigdata.flink.examples.UserSource.
- com.huawei.bigdata.flink.examples.TestPipeline_NettySink.
- com.huawei.bigdata.flink.examples.TestPipeline_NettySource1.
- com.huawei.bigdata.flink.examples.TestPipeline_NettySource2.

1. Code for sending messages:

```
package com.huawei.bigdata.flink.examples

case class Inforamtion(index: Int, content: String) {

    def this() = this(0, "")
}
```

2. The publisher customizes source operators to generate data.

```
package com.huawei.bigdata.flink.examples

import org.apache.flink.configuration.Configuration
import org.apache.flink.streaming.api.functions.source.RichParallelSourceFunction
import org.apache.flink.streaming.api.functions.source.SourceFunction.SourceContext

class UserSource extends RichParallelSourceFunction[Inforamtion] with Serializable{

    var isRunning = true

    override def open(parameters: Configuration): Unit = {
        super.open(parameters)
    }

    // Generate 10000 pieces of data each second.
    override def run(sourceContext: SourceContext[Inforamtion]) = {

        while (isRunning) {
            for (i <- 0 until 10000) {
                sourceContext.collect(Inforamtion(i, "hello-" + i));
            }
        }
    }
}
```

```
    Thread.sleep(1000)
  }
}

override def close(): Unit = super.close()

override def cancel() = {
  isRunning = false
}
}
```

3. Code for the publisher:

```
package com.huawei.bigdata.flink.examples

import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
import org.apache.flink.streaming.connectors.netty.sink.NettySink
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler
import org.apache.flink.streaming.api.scala._

object TestPipeline_NettySink {

  def main(args: Array[String]): Unit = {

    val env = StreamExecutionEnvironment.getExecutionEnvironment
    // Set the concurrency of job to 2.
    env.setParallelism(2)
    //Set Zookeeper as the registration server
    val zkRegisterServerHandler = new ZookeeperRegisterServerHandler
    //Add a user-defined operator to generate data.
    env.addSource(new UserSource) .keyBy(0).map(x=>x.content.getBytes)//Transform the to-be-sent data
    into a byte array.
    .addSink(new NettySink("NettySink-1", "TOPIC-2", zkRegisterServerHandler, 2))//Add NettySink
    operator to send data.
    env.execute()
  }
}
```

4. Code for the first subscriber

```
package com.huawei.bigdata.flink.examples

import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
import org.apache.flink.streaming.connectors.netty.source.NettySource
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler
import org.apache.flink.streaming.api.scala._

import scala.util.Random

object TestPipeline_NettySource1 {

  def main(args: Array[String]): Unit = {

    val env = StreamExecutionEnvironment.getExecutionEnvironment
    // Set the concurrency of job to 2.
    env.setParallelism(2)
    //Set ZooKeeper as the registration server
    val zkRegisterServerHandler = new ZookeeperRegisterServerHandler
    //Add a NettySource operator to receive messages from the publisher.
    env.addSource(new NettySource("NettySource-1", "TOPIC-2", zkRegisterServerHandler))
    .map(x => (1, new String(x)))//Add a NettySource operator to receive messages from the publisher.
    .filter(x => {
      Random.nextInt(50000) == 10
    })
    .print

    env.execute()
  }
}
```


5. Code for the second subscriber.

```
package com.huawei.bigdata.flink.examples

import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
import org.apache.flink.streaming.connectors.netty.source.NettySource
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler
import org.apache.flink.streaming.api.scala._

import scala.util.Random

object TestPipeline_NettySource2 {

  def main(args: Array[String]): Unit = {

    val env = StreamExecutionEnvironment.getExecutionEnvironment
    //Set the concurrency of job to 2.
    env.setParallelism(2)
    //Set the concurrency of job to 2.
    val zkRegisterServerHandler = new ZookeeperRegisterServerHandler
    //Add NettySource operator and receive data.

    env.addSource(new NettySource("NettySource-2", "TOPIC-2", zkRegisterServerHandler))
      .map(x=>(2, new String(x)))//Transform the received byte array into character strings.

      .filter(x=>{
        Random.nextInt(50000) == 10
      })
      .print()

    env.execute()
  }
}
```

11.3.5 Stream SQL Join Program

11.3.5.1 Scenario

Scenario

Assume that a Flink service (service 1) receives a message every second. The message records the basic information about a user, including the name, gender, and age. Another Flink service (service 2) receives a message irregularly, and the message records the name and career information about the user.

To meet the requirements of some services, the Flink application is developed to achieve the following function: uses the username recorded in the message received by service 2 as the keyword to jointly query two pieces of service data.

Data Planning

- The data of service 1 is stored in the Kafka component. Service 1 sends data (requiring Kafka user rights) to and receives data from the Kafka component. For details about how to configure Kafka, see the data planning section of [Data Planning](#).
- Service 2 receives messages using the socket. You can run the **netcat** command to input the analog data source.
 - Run the **netcat -l -p <port>** command to start a simple text server.

- After starting the application to connect to the port monitored by **netcat**, enter the data information to the netcat terminal.

Development Approach

1. Start the Flink Kafka Producer application to send data to Kafka.
2. Start the Flink Kafka Consumer application to receive data from Kafka, construct **Table1**, and ensure that the Topic is the same as that of Producer.
3. Read data from the socket and construct **Table2**.
4. Use Flink SQL to query and print **Table1** and **Table2**.

11.3.5.2 Java Sample Code

Function

In the Flink application, this code invokes the flink-connector-kafka module's API to generate and consume data.

Sample Code

If the user needs to use FusionInsight Kafka interconnected with the security mode before the development, obtain the **kafka-client-*.jar** JAR file from the FusionInsight client directory.

The following lists Producer, Consumer, and the main logic code used by Flink Stream SQL Join.

For the complete code, see

com.huawei.bigdata.flink.examples.WriteIntoKafka and **com.huawei.bigdata.flink.examples.SqlJoinWithSocket**.

1. A piece of user information is generated in Kafka every second. The user information includes the name, age, and gender.

```
//Producer code
public class WriteIntoKafka {
    public static void main(String[] args) throws Exception {
        //Print the command reference for flink run.
        System.out.println("use command as: ");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:9092");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21007 --security.protocol
SASL_PLAINTEXT --sasl.kerberos.service.name kafka");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:9093 --security.protocol SSL --
ssl.truststore.location /home/truststore.jks --ssl.truststore.password huawei");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21009 --security.protocol
SASL_SSL --sasl.kerberos.service.name kafka --ssl.truststore.location /home/truststore.jks --
ssl.truststore.password huawei");
        System.out.println("*****");
        System.out.println("<topic> is the kafka topic name");
        System.out.println("<bootstrap.servers> is the ip:port list of brokers");
        System.out.println("*****");

        //Construct the execution environment.
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        //Set the concurrency.
        env.setParallelism(1);
    }
}
```

```
//Parse the running parameters.
ParameterTool paraTool = ParameterTool.fromArgs(args);
//Construct a flow diagram and write the data generated from self-defined sources to Kafka.
DataStream<String> messageStream = env.addSource(new SimpleStringGenerator());
FlinkKafkaProducer<String> producer = new FlinkKafkaProducer<>(paraTool.get("topic"),
    new SimpleStringSchema(),
    paraTool.getProperties());
producer.setWriteTimestampToKafka(true);
messageStream.addSink(producer);
//Invoke execute to trigger the execution.
env.execute();
}
//Customize the sources and generate a message every second.
public static class SimpleStringGenerator implements SourceFunction<String> {
    static final String[] NAME = {"Carry", "Alen", "Mike", "Ian", "John", "Kobe", "James"};
    static final String[] SEX = {"MALE", "FEMALE"};
    static final int COUNT = NAME.length;
    boolean running = true;
    Random rand = new Random(47);
    @Override
    //Use rand to randomly generate a combination of the name, gender, and age.
    public void run(SourceContext<String> ctx) throws Exception {
        while (running) {
            int i = rand.nextInt(COUNT);
            int age = rand.nextInt(70);
            String sexy = SEX[rand.nextInt(2)];
            ctx.collect(NAME[i] + "," + age + "," + sexy);
            thread.sleep(1000);
        }
    }
    @Override
    public void cancel() {
        running = false;
    }
}
}
```

2. Generate **Table1** and **Table2**, use Join to jointly query **Table1** and **Table2**, and print the output result.

```
public class SqlJoinWithSocket {
    public static void main(String[] args) throws Exception{
        final String hostname;
        final int port;
        System.out.println("use command as: ");
        System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket" +
            " /opt/test.jar --topic topic-test -bootstrap.servers xxx.xxx.xxx.xxx:9092 --hostname
xxx.xxx.xxx.xxx --port xxx");
        System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket" +
            " /opt/test.jar --topic topic-test -bootstrap.servers xxx.xxx.xxx.xxx:21007 --security.protocol
SASL_PLAINTEXT --sasl.kerberos.service.name kafka"
            + "--hostname xxx.xxx.xxx.xxx --port xxx");
        System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket" +
            " /opt/test.jar --topic topic-test -bootstrap.servers xxx.xxx.xxx.xxx:9093 --security.protocol
SSL --ssl.truststore.location /home/truststore.jks "
            + "--ssl.truststore.password huawei --hostname xxx.xxx.xxx.xxx --port xxx");
        System.out.println("*****");
        System.out.println("<topic> is the kafka topic name");
        System.out.println("<bootstrap.servers> is the ip:port list of brokers");
        System.out.println("*****");
        try {
            final ParameterTool params = ParameterTool.fromArgs(args);
            hostname = params.has("hostname") ? params.get("hostname") : "localhost";
            port = params.getInt("port");
        } catch (Exception e) {
            System.err.println("No port specified. Please run 'FlinkStreamSqlJoinExample " +
                "--hostname <hostname> --port <port>', where hostname (localhost by default) " +
                "and port is the address of the text server");
            System.err.println("To start a simple text server, run 'netcat -l -p <port>' and " +
                "type the input text into the command line");
        }
        return;
    }
}
```

```
    }

    EnvironmentSettings fsSettings =
EnvironmentSettings.newInstance().useOldPlanner().inStreamingMode().build();
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    StreamTableEnvironment tableEnv = StreamTableEnvironment.create(env, fsSettings);
    //Perform processing based on EventTime.
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
    env.setParallelism(1);
    ParameterTool paraTool = ParameterTool.fromArgs(args);
    //Use Stream1 to read data from Kafka.
    DataStream<Tuple3<String, String, String>> kafkaStream = env.addSource(new
FlinkKafkaConsumer<>(paraTool.get("topic"),
    new SimpleStringSchema(),
    paraTool.getProperties()).map(new MapFunction<String, Tuple3<String, String, String>>()
{
    @Override
    public Tuple3<String, String, String> map(String s) throws Exception {
    String[] word = s.split(",");
    return new Tuple3<>(word[0], word[1], word[2]);
    }
});
//Register Stream1 as Table1.
tableEnv.registerDataStream("Table1", kafkaStream, "name, age, sexy, proctime.proctime");
//Use Stream2 to read data from the socket.
DataStream<Tuple2<String, String>> socketStream = env.socketTextStream(hostname, port,
"\n").
    map(new MapFunction<String, Tuple2<String, String>>() {
    @Override
    public Tuple2<String, String> map(String s) throws Exception {
    String[] words = s.split("\\s");
    if (words.length < 2) {
    return new Tuple2<>();
    }
    return new Tuple2<>(words[0], words[1]);
    }
});
//Register Stream2 as Table2.
tableEnv.registerDataStream("Table2", socketStream, "name, job, proctime.proctime");
//Run SQL Join to perform a combined query.
Table result = tableEnv.sqlQuery("SELECT t1.name, t1.age, t1.sexy, t2.job, t2.proctime as shiptime
\n" +
    "FROM Table1 AS t1\n" +
    "JOIN Table2 AS t2\n" +
    "ON t1.name = t2.name\n" +
    "AND t1.proctime BETWEEN t2.proctime - INTERVAL '1' SECOND AND t2.proctime +
INTERVAL '1' SECOND");
//Convert the query result into the stream and print the output.
tableEnv.toAppendStream(result, Row.class).print();
env.execute();
}
}
```

11.3.6 Interconnecting Flink with Cloud Search Service

11.3.6.1 Scenario Description

Scenario

In this example, Flink consumes a custom data source and writes the consumed data to Elasticsearch or CSS.

This section describes how to build an Elasticsearch sink and set parameters so that Flink can write data to Elasticsearch.

Flink 1.12.0 or later and Elasticsearch 7.x or later are supported.

Data Preparation

- If a custom data source is used, ensure that the source cluster and the destination cluster can communicate with each other through a network port.
- If external data sources, such as Kafka and MySQL, are used, ensure that the corresponding user has required data operation permissions.

Development Guideline

1. Import the dependency packages of Flink. The version must be the same as the Flink version in the cluster.
2. Build a data source at the source.
3. Build the Elasticsearch data source at the destination. (You can use the `setRestClientFactory` method to configure `UserRestClientFactory` when you build the data source.)
4. Build a Flink execution environment.

11.3.6.2 Java Sample Code

Description

Currently, a data source that continuously generates four strings is implemented based on the random number generator for writing data.

Sample Code

The following code snippet is used as an example. For complete code, see **com.huawei.bigdata.flink.examples**.

```
public class FlinkEsSinkExample {
    public static void main(String[] args) throws Exception {

        System.out.println("use command as:");
        System.out.println(
            "flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.FlinkEsSinkExample /<JAR_PATH>/FlinkEsSinkExample-1.0.jar");
        System.out.println(
            "*****");
        System.out.println("<JAR_PATH> is the path of jar file");
        System.out.println(
            "*****");
        //init flink execution environment
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        //create a dataSource according to RandomDataGenerator
        //of course, you can create any datasource from kafka, mysql, text file and so on
        DataGeneratorSource<String> dataGeneratorSource =
            new DataGeneratorSource<>(new DataGenerator<String>() {
                RandomDataGenerator generator;

                @Override
                public void open(String name, FunctionInitializationContext context,
                    RuntimeContext runtimeContext) throws Exception {
                    generator = new RandomDataGenerator();
                }
            });

        @Override
```

```
public boolean hasNext() {
    return true;
}

@Override
public String next() {
    return generator.nextHexString(4);
}
});

//create HttpHost list which are needed by elasticsearch sink
List<HttpHost> httpHosts = new ArrayList<>();
httpHosts.add(new HttpHost("172.16.0.117", 9200, "http"));
//httpHosts.add(new HttpHost("172.16.0.xxx", 9200, "http"));
ElasticsearchSink.Builder<String> esSinkBuilder =
    new ElasticsearchSink.Builder<>(httpHosts, new ElasticsearchSinkFunction<String>() {
        public IndexRequest createIndexRequest(String element) {
            Map<String, String> json = new HashMap<>();
            json.put("data", element);
            //init index request,put add real data as index request source
            return Requests.indexRequest().index("my-index").id(element).source(json);
        }
    });

@Override
public void process(String s, RuntimeContext runtimeContext,
    RequestIndexer requestIndexer) {
    requestIndexer.add(createIndexRequest(s));
}
});

// configuration for the bulk requests; this instructs the sink to emit after every element, otherwise they
would be buffered
esSinkBuilder.setBulkFlushMaxActions(1);

//add data source
DataStream<String> stringDataStream =
    env.addSource(dataGeneratorSource, "DataGeneratorSource").returns(Types.STRING);
//add elstic search sink
stringDataStream.addSink(esSinkBuilder.build());
env.execute();
}
}
```

11.4 Debugging the Application

11.4.1 Compiling and Running the Application

Scenarios

After the program code is developed, you can upload the code to the Linux client for running. The running procedures of applications developed in Scala or Java are the same.

NOTE

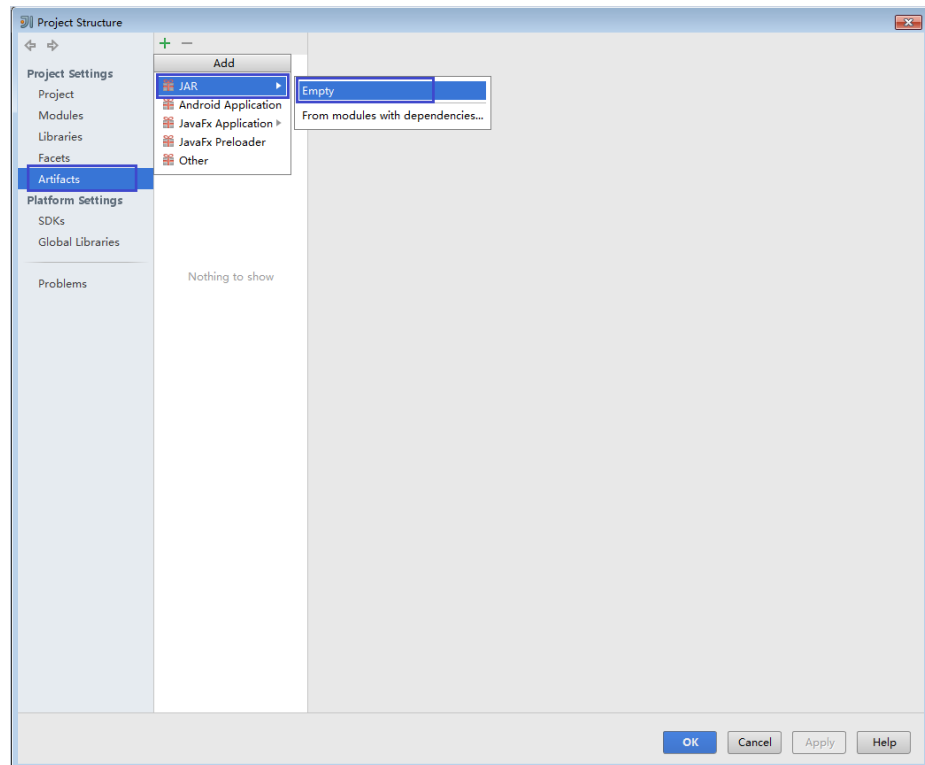
Applications in Flink On YARN mode are allowed to run in a Linux-based environment, but not in a Windows-based environment.

Procedure

- Step 1** In IntelliJ IDEA, configure the Artifacts information about the project before the **.jar** package is created.

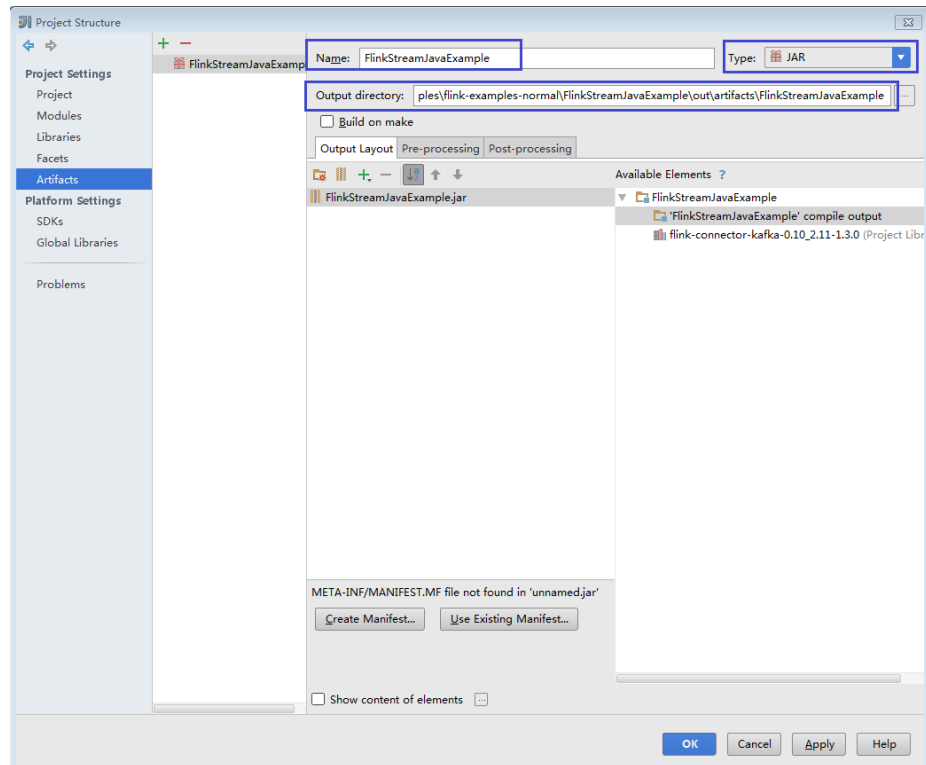
1. On the main page of IntelliJ IDEA, choose **File >Project Structures...** to access the **Project Structure** page.
2. On the **Project Structure** page, select **Artifacts**, click **+** and choose **JAR > Empty**

Figure 11-38 Adding Artifacts



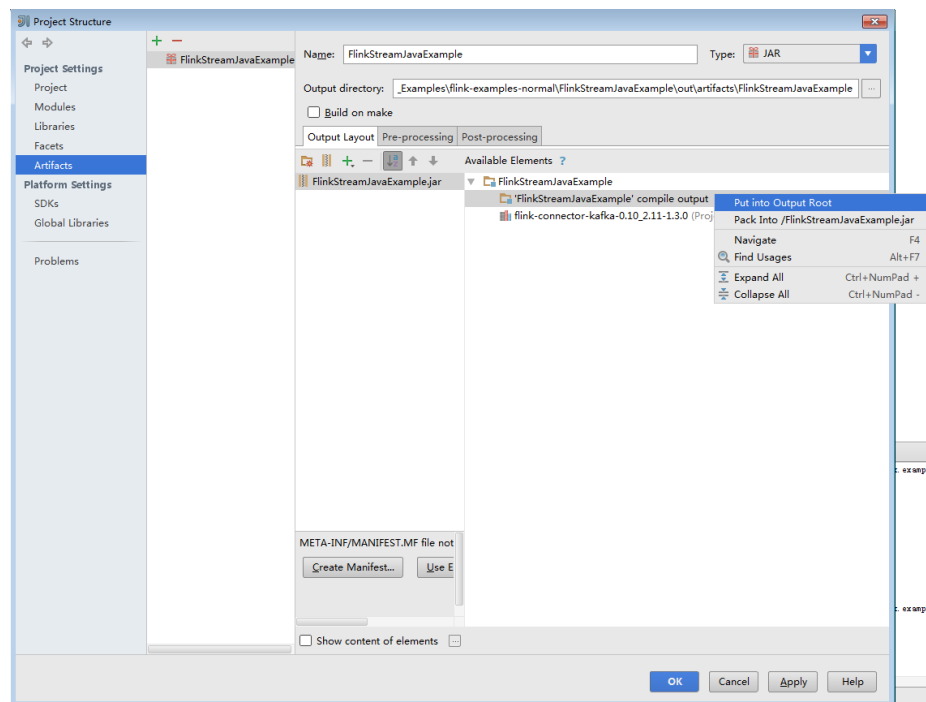
3. Configure the name, type, and output directory of the **.jar** package based on the actual condition.

Figure 11-39 Setting basic information



4. Right-click 'FlinkStreamJavaExample' compile output, choose **Put into Output Root**, and click **Apply**.

Figure 11-40 Choosing Put into Output Root

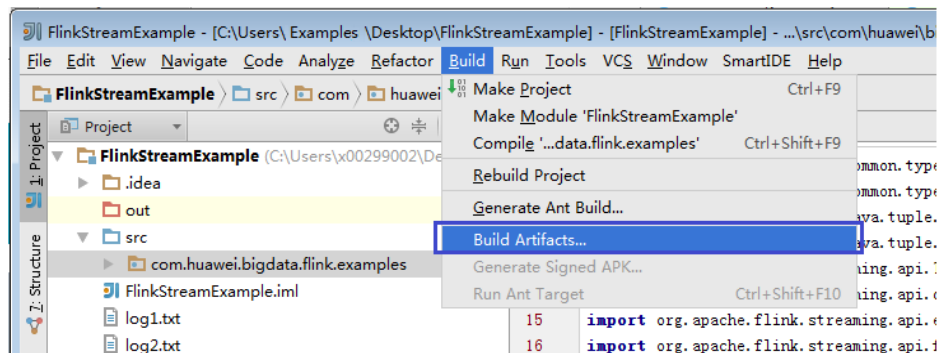


5. Click **OK** to complete the configuration.

Step 2 Generate the **.jar** package.

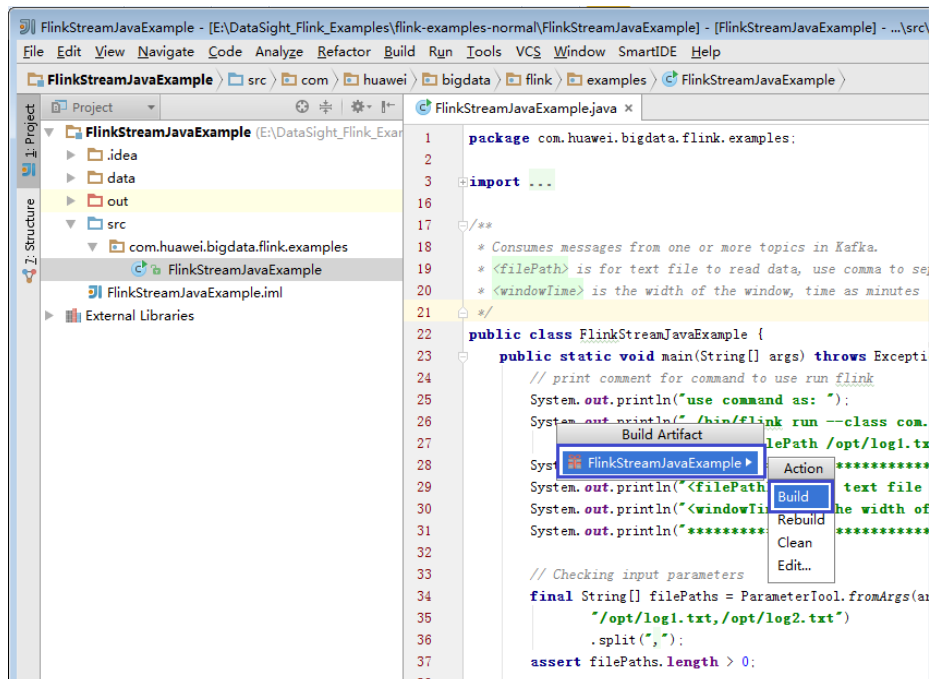
1. On the main page of IntelliJ IDEA, choose **Build > Build Artifacts...**

Figure 11-41 Choosing Build Artifacts



2. On the displayed menu, choose **FlinkStreamJavaExample > Build** to create a jar.

Figure 11-42 Generating the .jar package



3. If the following information is displayed in the event log, the **.jar** package is created successfully. You can obtain the **.jar** package from the directory configured in [Step 1.3](#).

21:25:43 Compilation completed successfully in 36 sec

Step 3 Copy the **.jar** package (for example **FlinkStreamJavaExample.jar**) created in [Step 2](#) to the Flink running environment (Flink client), for example, `/opt/client`, and then in that directory, create the **conf** folder and copy the required configuration files to the **conf** folder. For details, see [Preparing an Operating Environment](#), to run the Flink application.

In a Linux-based environment, Flink cluster needs to be started in advance. Run the **yarn session** command on Flink client and start Flink clusters. For example:

```
bin/yarn-session.sh -jm 1024 -tm 1024
```

 NOTE

- Before running the **yarn-session.sh** command, copy the running dependency package of the Flink application to client directory **#{client_install_home}/Flink/flink/lib**. For details about the application running dependency package, see [Reference Information](#).
- The dependencies of different sample projects may conflict. When running a new sample project, you need to remove the dependencies copied from the old sample project to the **{client_install_home}/Flink/flink/lib** directory on the client.
- Run the **source bingdata_env** command in the client installation directory before running the **yarn-session.sh** command.
- The **yarn-session.sh** command must be run in the **/Flink client installation directory/ Flink/flink** directory, for example, **/opt/client/Flink/flink**.
- When a Flink task is running, do not restart the HDFS service or all DataNode instances. Otherwise, the Flink task may fail, resulting loss of temporary data.
- Ensure that the user permissions on the JAR package and configuration file are the same as those on the Flink client. For example, the user is **omm** and the permissions are **755**.
- Run the DataStream sample program in Scala and Java.

Go to the Flink client directory and call the **bin/flink run** script to run the code.

– Java

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkStreamJavaExample /opt/client/FlinkStreamJavaExample.jar --filePath /opt/log1.txt,/opt/log2.txt --windowTime 2
```

– Scala

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkStreamScalaExample /opt/client/FlinkStreamScalaExample.jar --filePath /opt/log1.txt,/opt/log2.txt --windowTime 2
```

 NOTE

The **log1.txt** and **log2.txt** files must be stored on each node where the Yarn NodeManager instance is deployed and the permission is 755.

Table 11-6 Parameter description

Parameter	Description
<filePath>	File path in the local file system. The /opt/log1.txt and /opt/log2.txt files must be placed on each node. The default value can be retained or changed.
<windowTime>	Duration of the window. The unit is minute. The default value can be retained or changed.

- Run the following code to interconnecting with Kafka (in Scala and Java.)

Execution of the production data command to start the program.

```
bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/client/FlinkKafkaJavaExample.jar <topic> <bootstrap.servers> [security.protocol] [sas.l.kerberos.service.name] [ssl.truststore.location] [ssl.truststore.password]
```

Execution of the consumption data command to start the program.

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/client/FlinkKafkaJavaExample.jar <topic> <bootstrap.servers> [security.protocol] [sas.l.kerberos.service.name] [ssl.truststore.location] [ssl.truststore.password]
```

Table 11-7 Parameter description

Parameter	Description	Mandatory (Yes/No)
topic	The name of a Kafka topic.	Yes
bootstrap.server	The list of IP addresses or ports of broker clusters.	Yes
security.protocol	<p>The parameter need be set to protocols PLAINTEXT (optional), SASL_PLAINTEXT, SSL, and SASL_SSL, and the corresponding FusionInsight Kafka ports are 21005, 21007, 21008, and 21009.</p> <ul style="list-style-type: none"> - If the SASL is configured, the value of sasl.kerberos.service.name must be set to kafka and the security.kerberos.login parameter in conf/flink-conf.yaml are mandatory. - If SSL is configured, ssl.truststore.location (which species the path of truststore) and ssl.truststore.password (which specifies the password of truststore) are mandatory. 	<p>No</p> <p>NOTE</p> <ul style="list-style-type: none"> - If this parameter is not configured, the Kafka is non-secure. - If SSL needs to be configured, see "Kafka Development Guide" > "SSL Encryption Function Used by a Client" to determine the file generation mode of the truststore.jks file.

Following is the example code corresponding to four protocols, take ReadFromKafka as an example:

- **Command 1:**
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/client/FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:9092
- **Command 2:**
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/client/FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:21005 --security.protocol PLAINTEXT --sasl.kerberos.service.name kafka --kerberos.domain.name hadoop.hadoop.com

- **Command 3:**
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/client/FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:9093 --security.protocol SSL --ssl.truststore.location /home/truststore.jks --ssl.truststore.password huawei
- **Command 4:**
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/client/FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:21005 --security.protocol PLAINTEXT --sas.l.kerberos.service.name kafka --ssl.truststore.location /config/truststore.jks --ssl.truststore.password huawei --kerberos.domain.name hadoop.hadoop.com

- **Asynchronous checkpoint mechanism (in Scala and Java).**

The proctime is used as the timestamp for DataStream in Java, and the event time is used as the timestamp for DataStream in Scala. Following are examples of commands:

Save the Checkpoint snapshot information to HDFS.

- **Java**
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkProcessingTimeAPIMain /opt/client/FlinkCheckpointJavaExample.jar --chkPath hdfs://hacluster/flink/checkpoint/
- **Scala**
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkEventTimeAPIChkMain /opt/client/FlinkCheckpointScalaExample.jar --chkPath hdfs://hacluster/flink/checkpoint/

NOTE

- Path to checkpoint source file: **flink/checkpoint/checkpoint/fd5f5b3d08628d83038a30302b611/chk-X/4f854bf4-ea54-4595-a9d9-9b9080779ffe**
flink/checkpoint // The specified root directory.
fd5f5b3d08628d83038a30302b611 // The second-level directory named after jobID
chk-X // The third-level directory. X indicates the checkpoint numbers.
4f854bf4-ea54-4595-a9d9-9b9080779ffe // Source files of checkpoint.
- If Flink is in cluster mode, use the HDFS path, because a local path can only be used when Flink is in local mode.
- **Run the pipeline sample program.**
 - **Java**
 - i. **Start the publisher job.**
bin/flink run -p 2 --class com.huawei.bigdata.flink.examples.TestPipelineNettySink /opt/client/FlinkPipelineJavaExample.jar
 - ii. **Start the subscriber Job1.**
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipelineNettySource1 /opt/client/FlinkPipelineJavaExample.jar
 - iii. **Start the subscriber Job2.**
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipelineNettySource2 /opt/client/FlinkPipelineJavaExample.jar
 - **Scala**
 - i. **Start the publisher job.**
bin/flink run -p 2 --class com.huawei.bigdata.flink.examples.TestPipeline_NettySink /opt/client/FlinkPipelineScalaExample.jar
 - ii. **Start the subscriber Job1.**
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipeline_NettySource1 /opt/client/FlinkPipelineScalaExample.jar
 - iii. **Start the subscriber Job1.**
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipeline_NettySource2 /opt/client/FlinkPipelineScalaExample.jar

- Running the Stream SQL Join sample program
 - a. Start the program to generate data for Kafka. For details about Kafka configuration, see [Run the following code t...](#)

```
bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/client/
FlinkStreamSqlJoinExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.xxx:9092
```

- b. Run the **netcat** command on any node in the cluster to wait for the connection of the application.

```
netcat -l -p 9000
```

 **NOTE**

If "command not found" is displayed, install Netcat and run the command again.

- c. Start the application to accept the socket data and perform the combined query.

```
bin/flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket /opt/client/
FlinkStreamSqlJoinExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.xxx:9092 --
hostname xxx.xxx.xxx.xxx --port 9000
```

----End

11.4.2 Viewing the Debugging Result

Scenarios

After a Flink application completes running, you can view the running result, or use Apache Flink Dashboard to view application running status.

Procedure

- **View the running result of the Flink application.**

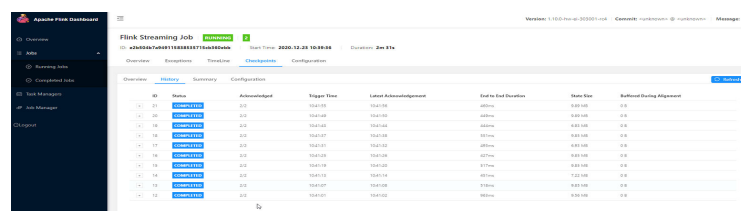
If you want to check the execution result, view the Stdout log of TaskManager on the Apache Flink Dashboard.

If the execution result is exported to a file or a location specified by Flink, view the result from the exported file or the location. The checkpoint, pipeline, and join between configuration tables and streams are used as examples.

- **View checkpoint results and files**

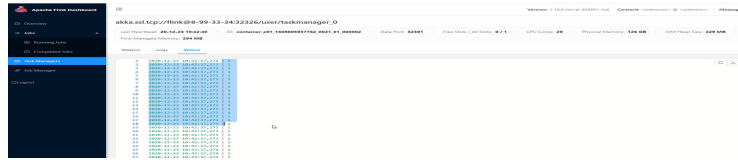
- The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Checkpoints** to view the submitted jobs as shown in [Figure 11-43](#). Choose **Task Managers > Stdout** to view the running result, as shown in [Figure 11-44](#).

Figure 11-43 Submitted jobs



ID	Status	Acknowledgment	Trigger Time	Latest Acknowledgment	End to End Duration	State Size	Backlog/Waiting Acknowledgment
10	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
11	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
12	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
13	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
14	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
15	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
16	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
17	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
18	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
19	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
20	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
21	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
22	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
23	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
24	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
25	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
26	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
27	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
28	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
29	Completed	1541-02	1541-02	1541-02	00:00	0 B	0
30	Completed	1541-02	1541-02	1541-02	00:00	0 B	0

Figure 11-44 Execution result



- Run the `hdfs dfs -ls hdfs://hacluster/flink/checkpoint/` command to view the checkpoint snapshot information in the HDFS.
- **View pipeline results**
 - The results are stored in the `taskmanager.out` file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Running Jobs** to view the running jobs as shown in Figure 11-45. Choose **Task Managers**. You can see two tasks, as shown in Figure 11-46. Click any task, choose **Stdout** to view the output of the task, as shown in Figure 11-47 and Figure 11-48.

Figure 11-45 Running jobs

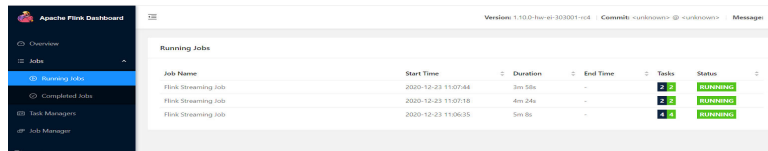


Figure 11-46 Submitted tasks

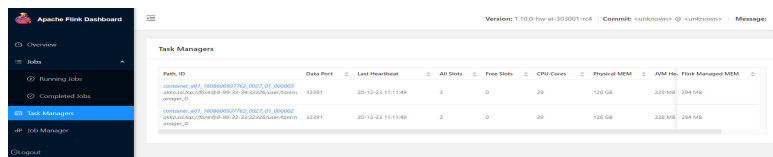


Figure 11-47 Output of task1

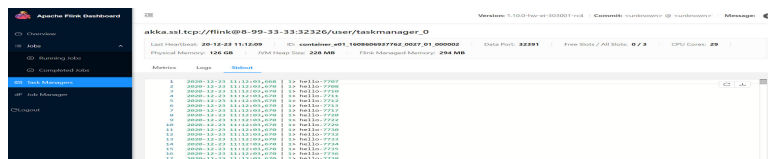
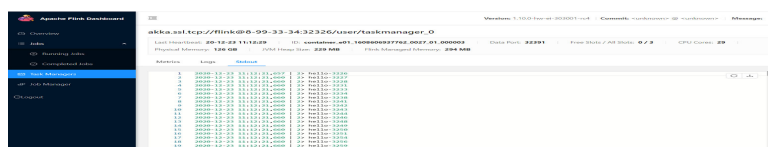


Figure 11-48 Output of task2



- **View the result of DataStream**
 - The results are stored in the `taskmanager.out` file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Completed Jobs** to view the completed job as shown in Figure 11-49. Choose **Task Managers**, you can see submitted task, as shown in Figure 11-50. Choose **Stdout** to view the running result, as shown in Figure 11-51.

Figure 11-55 Running jobs

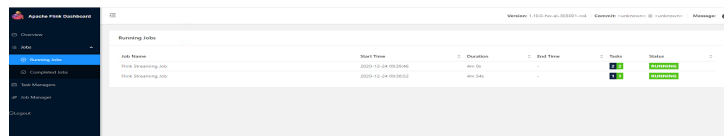


Figure 11-56 Submitted task

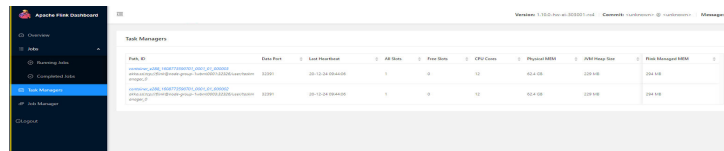
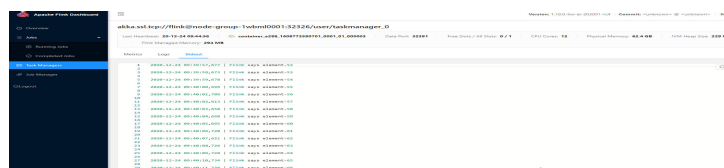


Figure 11-57 Execution result



- **Use Apache Flink Dashboard to view the running status of the Flink application.**

The Apache Flink Dashboard mainly includes Overview, Running Jobs, Completed Jobs, Task Managers, Job Manager and Logout and so on.

On In the YARN web UI, find the desired Flink application. Click the **ApplicationMaster** at the last column of the application to switch to the Apache Flink Dashboard.

View the print results of the program execution: find the corresponding **Task Manager** to see the corresponding **Stdout** tag log information.

- **View Flink logs.**

Three methods can be used to obtain Flink logs:

- Log in to the Apache Flink Dashboard and view logs of TaskManagers and JobManager.
- Log in to the YARN web UI to view logs about JobManager and GC.

On the YARN web UI wind, find the desired Flink application. Click the **ID** of the application. On the switched page, click **Logs** in the Logs column.

- On the Yarn client, obtain or view logs of Task Managers and Job Manager.

i. Download and install the Yarn client, for example, in the /opt/client directory.

ii. Use PuTTY to log in to the node where the client is installed as the client installation user.

iii. Run the following command to switch to the client installation directory:

cd /opt/client

iv. Run the following command to configure environment variables:

source bigdata_env

- v. If the cluster employs the security mode, run the following command to authenticate the user. If the normal mode is used, skip this step.

kinit component service user

- vi. Run the following commands to obtain container information of the Flink cluster:

yarn logs -applicationId application_* -show_application_log_info

```

root@hadoop001:~/opt/flinkclient/Flink/flink # yarn logs -applicationId application_1547547065745_0001 -show_application_log_info
WARNING: YARN_CONF_DIR has been replaced by HADOOP_CONF_DIR. Using value of YARN_CONF_DIR.
Application State: Running.
Container: container_1547547065745_0001_01_000001 on hadoop001:26009
Container: container_1547547065745_0001_01_000002 on hadoop001:26009
Container: container_1547547065745_0001_01_000003 on hadoop001:26009
Container: container_1547547065745_0001_01_000004 on hadoop001:26009
    
```

- vii. Run the following command to obtain run logs of the specified container. Generally, container_*_000001 is the container where the Job Manager is running.

yarn logs -applicationId application_* --containerId container_1547547065745_0001_01_000004 -out logdir/

```

root@hadoop001:~/opt/flinkclient/Flink/flink/logdir/hadoop001-26009 # ll
total 172
-rw-r--r-- 1 root root 170605 Jan 17 10:24 container_1547547065745_0001_01_000004
root@hadoop001:~/opt/flinkclient/Flink/flink/logdir/hadoop001-26009 #
    
```

After the command is executed, container run logs, including run logs of the Task Manager and Job Manager and GC logs, are downloaded to the local host.

- viii. You can also run a command to obtain the log with the specified name:

The following command is used to obtain the container log list.

yarn logs -applicationId application_* -show_container_log_info --containerId container_1547547065745_0001_01_000004

```

root@hadoop001:~/opt/flinkclient/Flink/flink/logdir/hadoop001-26009 # yarn logs -applicationId application_1547547065745_0001 -show_container_log_info
WARNING: YARN_CONF_DIR has been replaced by HADOOP_CONF_DIR. Using value of YARN_CONF_DIR.
Container: container_1547547065745_0001_01_000004 on hadoop001:26009
-----
LogFile                               LogLength  LastModificationTime  LogAggregationType
-----
container-localizer-syslog             184        Wed Jan 16 17:49:27 +0800 2019                LOCAL
taskmanager.log                       131430     Wed Jan 16 17:49:35 +0800 2019                LOCAL
gc.log@current                         17952     Thu Jan 17 10:22:26 +0800 2019                LOCAL
taskmanager.out                        0         Wed Jan 16 17:49:31 +0800 2019                LOCAL
launch_container.sh                   12232     Wed Jan 16 17:49:31 +0800 2019                LOCAL
directory.info                         3661      Wed Jan 16 17:49:31 +0800 2019                LOCAL
taskmanager.err                       1060      Wed Jan 16 17:49:31 +0800 2019                LOCAL
prelaunch.out                          100       Wed Jan 16 17:49:31 +0800 2019                LOCAL
prelaunch.err                          0         Wed Jan 16 17:49:31 +0800 2019                LOCAL
    
```

Download **taskmanager.log** to the local host.

yarn logs -applicationId application_* --containerId container_1547547065745_0001_01_000004 -log_files taskmanager.log -out localpath

11.5 More Information

11.5.1 Introduction to Common APIs

11.5.1.1 Java

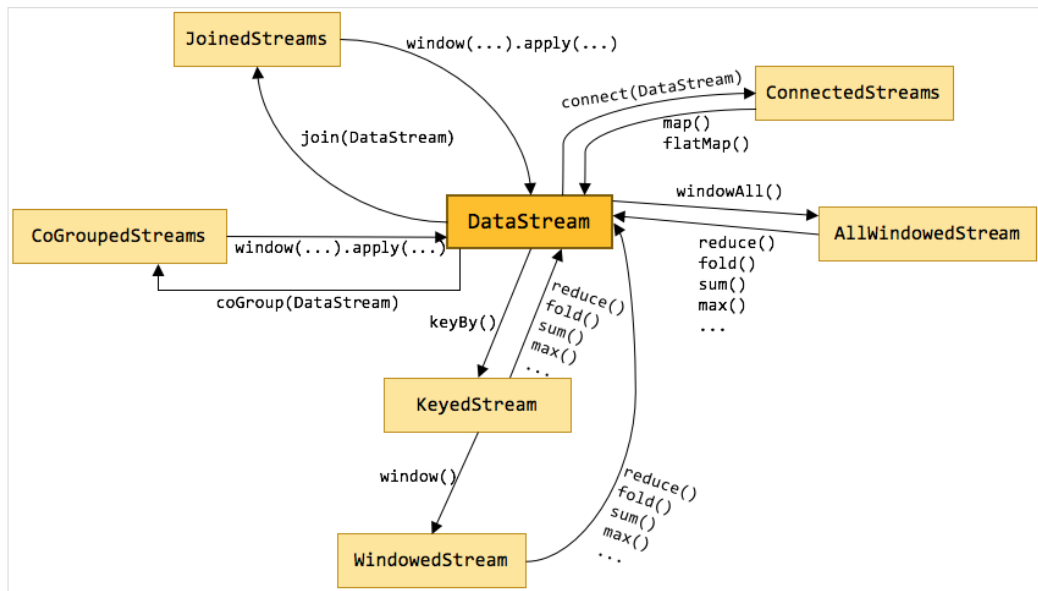
To avoid API compatibility or reliability issues after updates to the open-source Flink, recommended that APIs of the matching version are recommended.

Common APIs of Flink

Flink mainly uses the following APIs:

- **StreamExecutionEnvironment**: provides the execution environment, which is the basis of Flink stream processing.
- **DataStream**: represents streaming data. DataStream can be regarded as unmodifiable collection that contains duplicate data. The number of elements in DataStream are unlimited.
- **KeyedStream**: generates the stream by performing keyBy grouping operations on DataStream. Data is grouped based on the specified key value.
- **WindowedStream**: generates the stream by performing the window function on KeyedStream, sets the window type, and defines window triggering conditions.
- **AllWindowedStream**: generates streams by performing the window function on DataStream, sets the window type, defines window triggering conditions, and performs operations on the window data.
- **ConnectedStreams**: generates streams by connecting the two DataStreams and retaining the original data type, and performs the map or flatMap operation.
- **JoinedStreams**: performs equijoin (which is performed when two values are equal, for example, a.id = b.id) operation to data in the window. The join operation is a special scenario of coGroup operation.
- **CoGroupedStreams**: performs the coGroup operation on data in the window. CoGroupedStreams can perform various types of join operations.

Figure 11-58 Conversion of Flink stream types



Data Stream Source

Table 11-8 APIs about data stream source

API	Description
public final <OUT> DataStreamSource<OUT> fromElements(OUT... data)	Obtain user-defined data of multiple elements as the data stream source. <ul style="list-style-type: none"> • type indicates the data type of an element. • data indicates the data of multiple elements.
public final <OUT> DataStreamSource<OUT> fromElements(Class<OUT> type, OUT... data)	
public <OUT> DataStreamSource<OUT> fromCollection(Collection<OUT> data)	Obtain the user-defined data collection as the data stream source. <ul style="list-style-type: none"> • type indicates the data type of elements in the collection. • typeInfo indicates the type information obtained based on the element data type in the collection. • data indicates the iterator.
public <OUT> DataStreamSource<OUT> fromCollection(Collection<OUT> data, TypeInformation<OUT> typeInfo)	
public <OUT> DataStreamSource<OUT> fromCollection(Iterator<OUT> data, Class<OUT> type)	
public <OUT> DataStreamSource<OUT> fromCollection(Iterator<OUT> data, TypeInformation<OUT> typeInfo)	
public <OUT> DataStreamSource<OUT> fromParallelCollection(SplittableIterator<OUT> iterator, Class<OUT> type)	Obtain the user-defined data collection as parallel data stream source. <ul style="list-style-type: none"> • type indicates the data type of elements in the collection. • typeInfo indicates the type information obtained based on the element data type in the collection. • iterator indicates the iterator that can be divided into multiple partitions.
public <OUT> DataStreamSource<OUT> fromParallelCollection(SplittableIterator<OUT> iterator, TypeInformation<OUT> typeInfo)	
private <OUT> DataStreamSource<OUT> fromParallelCollection(SplittableIterator<OUT> iterator, TypeInformation<OUT> typeInfo, String operatorName)	

API	Description
public DataSource<Long> generateSequence(long from, long to)	Obtain a sequence of user-defined data as the data stream source. <ul style="list-style-type: none"> • from indicates the starting point of numbers. • to indicates the end point of numbers.
public DataSource<String> readTextFile(String filePath)	Obtain the user-defined text file from a specific path as the data stream source. <ul style="list-style-type: none"> • filePath indicates the path of the text file. • charsetName indicates the encoding format.
public DataSource<String> readTextFile(String filePath, String charsetName)	
public <OUT> DataSource<OUT> readFile(FileInputFormat<OUT> inputFormat, String filePath)	Obtain the user-defined file from a specific path as the data stream source. <ul style="list-style-type: none"> • filePath indicates the file path. • inputFormat indicates the format of the file. • watchType indicates the file processing mode, which can be PROCESS_ONCE or PROCESS_CONTINUOUSLY. • interval indicates the interval for processing directories or files.
public <OUT> DataSource<OUT> readFile(FileInputFormat<OUT> inputFormat, String filePath, FileProcessingMode watchType, long interval)	
public <OUT> DataSource<OUT> readFile(FileInputFormat<OUT> inputFormat, String filePath, FileProcessingMode watchType, long interval, TypeInfo<OUT> typeInformation)	
public DataSource<String> socketTextStream(String hostname, int port, String delimiter, long maxRetry)	Obtain user-defined socket data as the data stream source. <ul style="list-style-type: none"> • hostname indicates the host name of the socket server. • port indicates the listening port of the server. • delimiter indicates the separator of messages. • maxRetry refers to the maximum retry times that can be triggered by abnormal connections.
public DataSource<String> socketTextStream(String hostname, int port, String delimiter)	
public DataSource<String> socketTextStream(String hostname, int port)	

API	Description
public <OUT> DataStreamSource<OUT> addSource(SourceFunction<OUT> function)	Customize the SourceFunction and addSource methods to add data sources such as Kafka. The implementation method is the run method of SourceFunction. <ul style="list-style-type: none"> • function indicates the user-defined SourceFunction function. • sourceName indicates the name of data source. • typeInfo indicates the type information obtained based on the element data type.
public <OUT> DataStreamSource<OUT> addSource(SourceFunction<OUT> function, String sourceName)	
public <OUT> DataStreamSource<OUT> addSource(SourceFunction<OUT> function, TypeInformation<OUT> typeInfo)	
public <OUT> DataStreamSource<OUT> addSource(SourceFunction<OUT> function, String sourceName, TypeInformation<OUT> typeInfo)	

Data Output

Table 11-9 APIs about data output

API	Description
public DataStreamSink<T> print()	Print data as the standard output stream.
public DataStreamSink<T> printToErr()	Print data as the standard error output stream.
public DataStreamSink<T> writeAsText(String path)	Write data to a specific text file. <ul style="list-style-type: none"> • path indicates the path of the text file. • writeMode indicates the writing mode, which can be OVERWRITE or NO_OVERWRITE.
public DataStreamSink<T> writeAsText(String path, WriteMode writeMode)	
public DataStreamSink<T> writeAsCsv(String path)	Write data to a specific .csv file. <ul style="list-style-type: none"> • path indicates the path of the text file. • writeMode indicates the writing mode, which can be OVERWRITE or NO_OVERWRITE. • rowDelimiter indicates the row separator.
public DataStreamSink<T> writeAsCsv(String path, WriteMode writeMode)	

API	Description
public <X extends Tuple> DataStreamSink<T> writeAsCsv(String path, WriteMode writeMode, String rowDelimiter, String fieldDelimiter)	<ul style="list-style-type: none"> • fieldDelimiter indicates the column separator.
public DataStreamSink<T> writeToSocket(String hostName, int port, SerializationSchema<T> schema)	<p>Write data to the socket connection.</p> <ul style="list-style-type: none"> • hostName indicates the host name. • port indicates the port number.
public DataStreamSink<T> writeUsingOutputformat(Outputformat<T> format)	Write data to a file, for example, a binary file.
public DataStreamSink<T> addSink(SinkFunction<T> sinkFunction)	Export data in a user-defined manner. The flink-connectors allows the addSink method to support exporting data to Kafka. The implementation method is the invoke method of SinkFunction.

Filtering and Mapping

Table 11-10 APIs about filtering and mapping

API	Description
public <R> SingleOutputStreamOperator<R> map(MapFunction<T, R> mapper)	Transform an element into another element of the same type.
public <R> SingleOutputStreamOperator<R> flatMap(FlatMapFunction<T, R> flatMapper)	Transform an element into zero, one, or multiple elements.
public SingleOutputStreamOperator<T> filter(FilterFunction<T> filter)	Run a Boolean function on each element and retain elements that return true .

Aggregation

Table 11-11 APIs about aggregation

API	Description
public KeyedStream<T, Tuple> keyBy(int... fields)	Logically divide a stream into multiple partitions, each containing elements with the same key. The partitioning is

API	Description
public KeyedStream<T, Tuple> keyBy(String... fields)	internally implemented using hash partition. A KeyedStream is returned.
public <K> KeyedStream<T, K> keyBy(KeySelector<T, K> key)	Return the KeyedStream after the KeyBy operation, and call the KeyedStream function, such as reduce, fold, min, minby, max, maxby, sum, and sumby. <ul style="list-style-type: none"> • fields indicates the numbers of columns or names of member variables. • key indicates the user-defined basis for partitioning.
public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> reducer)	Perform reduce on KeyedStream in a rolling manner. Aggregate the current element and the last reduced value into a new value. The types of the three values are the same.
public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> folder)	Perform fold operation on KeyedStream based on an initial value in a rolling manner. Aggregate the current element and the last folded value into a new value. The input and returned values of Fold can be different.
public SingleOutputStreamOperator<T> sum(int positionToSum)	Calculate the sum in a KeyedStream in a rolling manner.
public SingleOutputStreamOperator<T> sum(String field)	positionToSum and field indicate calculating the sum of a specific column.
public SingleOutputStreamOperator<T> min(int positionToMin)	Calculate the minimum value in a KeyedStream in a rolling manner. min returns the minimum value, without guarantee of the correctness of columns of non-minimum values.
public SingleOutputStreamOperator<T> min(String field)	positionToMin and field indicate calculating the minimum value of a specific column.
public SingleOutputStreamOperator<T> max(int positionToMax)	Calculate the maximum value in KeyedStream in a rolling manner. max returns the maximum value, without guarantee of the correctness of columns of non-maximum values.
public SingleOutputStreamOperator<T> max(String field)	positionToMax and field indicate calculating the maximum value of a specific column.

API	Description
public SingleOutputStreamOperator<T> minBy(int positionToMinBy)	Obtain the row where the minimum value of a column locates in a KeyedStream. minBy returns all elements of that row. <ul style="list-style-type: none"> • positionToMinBy indicates the column on which the minBy operation is performed. • first indicates whether to return the first or last minimum value.
public SingleOutputStreamOperator<T> minBy(String positionToMinBy)	
public SingleOutputStreamOperator<T> minBy(int positionToMinBy, boolean first)	
public SingleOutputStreamOperator<T> minBy(String field, boolean first)	
public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy)	Obtain the row where the maximum value of a column locates in a KeyedStream. maxBy returns all elements of that row. <ul style="list-style-type: none"> • positionToMaxBy indicates the column on which the maxBy operation is performed. • first indicates whether to return the first or last maximum value.
public SingleOutputStreamOperator<T> maxBy(String positionToMaxBy)	
public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy, boolean first)	
public SingleOutputStreamOperator<T> maxBy(String field, boolean first)	

DataStream Distribution

Table 11-12 APIs about DataStream distribution

API	Description
public <K> DataStream<T> partitionCustom(Partitioner<K> partitioner, int field)	Use a user-defined partitioner to select target task for each element. <ul style="list-style-type: none"> • partitioner indicates the user-defined method for repartitioning. • field indicates the input parameters of partitioner. • keySelector indicates the user-defined input parameters of partitioner.
public <K> DataStream<T> partitionCustom(Partitioner<K> partitioner, String field)	

API	Description
public <K> DataStream<T> partitionCustom(Parti- tioner<K> partitioner, KeySelector<T, K> keySelector)	
public DataStream<T> shuffle()	Randomly and evenly partition elements.
public DataStream<T> rebalance()	Partition elements in round-robin manner, ensuring load balance of each partition. This partitioning is helpful to data with skewness.
public DataStream<T> rescale()	Distribute elements into downstream subset in round-robin manner.
public DataStream<T> broadcast()	Broadcast each element to all partitions.

Project Capabilities

Table 11-13 APIs about projecting

API	Description
public <R extends Tuple> SingleOutputStrea- mOperator<R> project(int... fieldIndexes)	Select some field subset from the tuple. fieldIndexes indicates some sequences of the tuple. NOTE Only tuple data type is supported by the project API.

Configuring the eventtime Attribute

Table 11-14 APIs about configuring the eventtime attribute

API	Description
public SingleOutputStreamOperator<T> assignTimestampsAndWatermarks(AssignerWithPeriodicWatermarks<T> timestampAndWatermarkAssigner)	Extract timestamp from records, enabling event time window to trigger computing.
public SingleOutputStreamOperator<T> assignTimestampsAndWatermarks(AssignerWithPunctuatedWatermarks<T> timestampAndWatermarkAssigner)	

Table 11-15 lists differences of AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks APIs.

Table 11-15 Difference of AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks APIs

Parameter	Description
AssignerWithPeriodicWatermarks	Generate Watermark based on the getConfig().setAutoWatermarkInterval(200L) timestamp of StreamExecutionEnvironment class.
AssignerWithPunctuatedWatermarks	Generate a Watermark each time an element is received. Watermarks can be different based on received elements.

Iteration

Table 11-16 APIs about iteration

API	Description
public IterativeStream<T> iterate()	In the flow, create in a feedback loop to redirect the output of an operator to a preceding operator.
public IterativeStream<T> iterate(long maxWaitTimeMillis)	<p>NOTE</p> <ul style="list-style-type: none"> This API is helpful to algorithms that require constant update of models. long maxWaitTimeMillis: The timeout period of each round of iteration.

Stream Splitting

Table 11-17 APIs about stream splitting

API	Description
public SplitStream<T> split(OutputSelector<T> outputSelector)	Use OutputSelector to rewrite select method, specify stream splitting basis (by tagging), and construct SplitStream streams. That is, a string is tagged for each element, so that a stream of a specific tag can be selected or created.
public DataStream<OUT> select(String... outputNames)	Select one or multiple streams from a SplitStream. outputNames indicates the sequence of string tags created for each element using the split method.

Window

Windows can be classified as tumbling windows and sliding windows.

- APIs such as Window, TimeWindow, CountWindow, WindowAll, TimeWindowAll, and CountWindowAll can be used to generate windows.
- APIs such as Window Apply, Window Reduce, Window Fold, and Aggregations on windows can be used to operate windows.
- Multiple Window Assigners are supported, including TumblingEventTimeWindows, TumblingProcessingTimeWindows, SlidingEventTimeWindows, SlidingProcessingTimeWindows, EventTimeSessionWindows, ProcessingTimeSessionWindows, and GlobalWindows.
- ProcessingTime, EventTime, and IngestionTime are supported.
- Timestamp modes EventTime: AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks are supported.

Table 11-18 lists APIs for generating windows.

Table 11-18 APIs for generating windows

API	Description
<pre>public <W extends Window> WindowedStream<T, KEY, W> window(WindowAssigner<? super T, W> assigner)</pre>	<p>Define windows in partitioned KeyedStreams. A window groups each key according to some characteristics (for example, data received within the latest 5 seconds).</p>
<pre>public <W extends Window> AllWindowedStream<T, W> windowAll(WindowAssigner<? super T, W> assigner)</pre>	<p>Define windows in DataStreams.</p>
<pre>public WindowedStream<T, KEY, TimeWindow> timeWindow(Time size)</pre>	<p>Define time windows in partitioned KeyedStreams, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters.</p> <ul style="list-style-type: none"> • size indicates the duration of the window. • slide indicates the sliding time of window. <p>NOTE</p> <ul style="list-style-type: none"> • WindowedStream and AllWindowedStream indicates two types of streams. • If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.
<pre>public WindowedStream<T, KEY, TimeWindow> timeWindow(Time size, Time slide)</pre>	
<pre>public AllWindowedStream<T, TimeWindow> timeWindowAll(Time size)</pre>	<p>Define time windows in partitioned DataStream, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters.</p> <ul style="list-style-type: none"> • size indicates the duration of the window. • slide indicates the sliding time of window.
<pre>public AllWindowedStream<T, TimeWindow> timeWindowAll(Time size, Time slide)</pre>	
<pre>public WindowedStream<T, KEY, GlobalWindow> countWindow(long size)</pre>	<p>Divide windows according to the number of elements and define windows in partitioned KeyedStreams.</p> <ul style="list-style-type: none"> • size indicates the duration of the window. • slide indicates the sliding time of window.

API	Description
public WindowedStream<T, KEY, GlobalWindow> countWindow(long size, long slide)	<p>NOTE</p> <ul style="list-style-type: none"> WindowedStream and AllWindowedStream indicates two types of streams. If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.
public AllWindowedStream<T, GlobalWindow> countWindowAll(Time size)	<p>Divide windows according to the number of elements and define windows in DataStreams.</p> <ul style="list-style-type: none"> size indicates the duration of the window. slide indicates the sliding time of window.
public AllWindowedStream<T, GlobalWindow> countWindowAll(Time size, Time slide)	

Table 11-19 lists APIs for operating windows.

Table 11-19 APIs for operating windows

Method	API	Description
Window	public <R> SingleOutputStreamOperator<R> apply(WindowFunction<T, R, K, W> function)	<p>Apply a general function to a window. The data in the window is calculated as a whole.</p> <ul style="list-style-type: none"> function indicates the window function to be executed. resultType indicates the type of returned data.
	public <R> SingleOutputStreamOperator<R> apply(WindowFunction<T, R, K, W> function, TypeInformation<R> resultType)	
	public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> function)	<p>Apply a reduce function to the window and return the result.</p> <ul style="list-style-type: none"> reduceFunction indicates the reduce function to be executed. function of WindowFunction indicates triggering an operation to the window after a reduce operation. resultType indicates the type of returned data.
	public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, WindowFunction<T, R, K, W> function)	

Method	API	Description
	<pre>public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, WindowFunction<T, R, K, W> function, TypeInformation<R> resultType)</pre>	
	<pre>public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function)</pre>	<p>Apply a fold function to the window and return the result.</p> <ul style="list-style-type: none"> • initialValue indicates the initial value. • foldFunction indicates the fold function. • function of WindowFunction indicates triggering an operation to the window after a fold operation. • resultType indicates the type of returned data.
	<pre>public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function, TypeInformation<R> resultType)</pre>	
	<pre>public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, WindowFunction<ACC, R, K, W> function)</pre>	
	<pre>public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, WindowFunction<ACC, R, K, W> function, TypeInformation<ACC> foldAccumulatorType, TypeInformation<R> resultType)</pre>	
Window All	<pre>public <R> SingleOutputStreamOperator<R> apply(AllWindowFunction<T, R, W> function)</pre>	Apply a general function to a window. The data in the window is calculated as a whole.

Method	API	Description
	<pre>public <R> SingleOutputStreamOperator<R> apply(AllWindowFunction<T, R, W> function, TypeInfo<R> resultType)</pre>	
	<pre>public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> function)</pre>	<p>Apply a reduce function to the window and return the result.</p> <ul style="list-style-type: none"> ● reduceFunction indicates the reduce function to be executed. ● AllWindowFunction indicates triggering an operation to the window after a reduce operation. ● resultType indicates the type of returned data.
	<pre>public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, AllWindowFunction<T, R, W> function)</pre>	
	<pre>public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, AllWindowFunction<T, R, W> function, TypeInfo<R> resultType)</pre>	
	<pre>public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function)</pre>	<p>Apply a fold function to the window and return the result.</p> <ul style="list-style-type: none"> ● initialValue indicates the initial value. ● foldFunction indicates the fold function. ● function of WindowFunction indicates triggering an operation to the window after a fold operation. ● resultType indicates the type of returned data.
	<pre>public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function, TypeInfo<R> resultType)</pre>	
	<pre>public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, AllWindowFunction<ACC, R, W> function)</pre>	

Method	API	Description
	<pre>public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, AllWindowsFunction<ACC, R, W> function, TypeInfo<ACC> foldAccumulatorType, TypeInfo<R> resultType)</pre>	
Window and Window All	<pre>public SingleOutputStreamOperator<T> sum(int positionToSum)</pre>	Sum a specified column of the window data. field and positionToSum indicate a specific column of the data.
	<pre>public SingleOutputStreamOperator<T> sum(String field)</pre>	
	<pre>public SingleOutputStreamOperator<T> min(int positionToMin)</pre>	Calculate the minimum value of a specified column of the window data. min returns the minimum value, without guarantee of the correctness of columns of non-minimum values. positionToMin and field indicate calculating the minimum value of a specific column.
	<pre>public SingleOutputStreamOperator<T> min(String field)</pre>	
	<pre>public SingleOutputStreamOperator<T> minBy(int positionToMinBy)</pre>	Obtain the row where the minimum value of a column locates in the window data. minBy returns all elements of that row. <ul style="list-style-type: none"> • positionToMinBy indicates the column on which the minBy operation is performed. • first indicates whether to return the first or last minimum value.
	<pre>public SingleOutputStreamOperator<T> minBy(String positionToMinBy)</pre>	
	<pre>public SingleOutputStreamOperator<T> minBy(int positionToMinBy, boolean first)</pre>	
	<pre>public SingleOutputStreamOperator<T> minBy(String field, boolean first)</pre>	
<pre>public SingleOutputStreamOperator<T> max(int positionToMax)</pre>	Calculate the maximum value of a specified column of the window data. max returns the maximum value, without guarantee of the correctness of columns of non-maximum values.	

Method	API	Description
	public SingleOutputStreamOperator<T> max(String field)	positionToMax and field indicate calculating the maximum value of a specific column.
	The default public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy)//true	Obtain the row where the maximum value of a column locates in the window data. maxBy returns all elements of that row. <ul style="list-style-type: none"> • positionToMaxBy indicates the column on which the maxBy operation is performed. • first indicates whether to return the first or last maximum value.
	The default public SingleOutputStreamOperator<T> maxBy(String positionToMaxBy)//true	
	public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy, boolean first)	
	public SingleOutputStreamOperator<T> maxBy(String field, boolean first)	
	public SingleOutputStreamOperator<T> maxBy(String field, boolean first)	

Combining Multiple DataStreams

Table 11-20 APIs about combining multiple DataStreams

API	Description
public final DataStream<T> union(DataStream<T>... streams)	Perform union operation on multiple DataStreams to generate a new data stream containing all data from original DataStreams. NOTE If you perform union operation on a piece of data with itself, there are two copies of the same data.
public <R> ConnectedStreams<T, R> connect(DataStream<R> > dataStream)	Connect two DataStreams and retain the original type. The connect API method allows two DataStreams to share statuses. After ConnectedStreams is generated, map or flatMap operation can be called.
public <R> SingleOutputStreamOperator<R> map(CoMapFunction<I N1, IN2, R> coMapper)	Perform mapping operation, which is similar to map and flatMap operation in a ConnectedStreams, on elements. After the operation, the type of the new DataStreams is string.

API	Description
<pre>public <R> SingleOutputStrea- mOperator<R> flatMap(CoFlatMapFun ction<IN1, IN2, R> coFlatMapper)</pre>	Perform mapping operation, which is similar to map and flatMap operation in a ConnectedStreams, on elements.

Join Operation

Table 11-21 APIs about join operation

API	Description
<pre>public <T2> JoinedStreams<T, T2> join(DataStream<T2> otherStream)</pre>	Join two DataStreams using a given key in a specified window.
<pre>public <T2> CoGroupedStreams<T, T2> coGroup(DataStream< T2> otherStream)</pre>	Co-group two DataStreams using a given key in a specified window.

11.5.1.2 Scala

To avoid API compatibility or reliability issues after updates to the open-source Flink, recommended that APIs of the matching version are recommended.

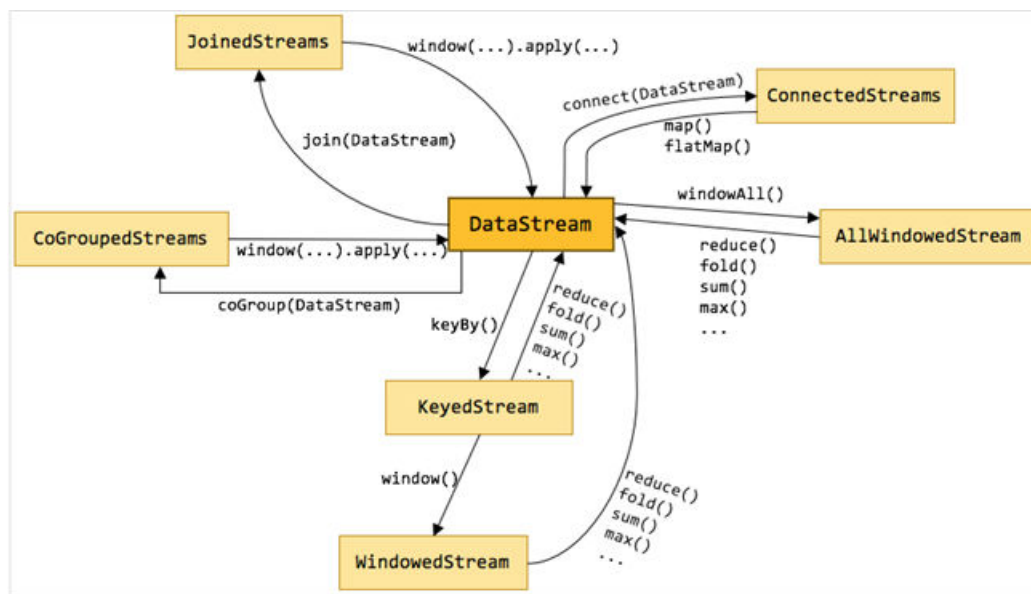
Common APIs of Flink

Flink mainly uses the following APIs:

- **StreamExecutionEnvironment**: provides the execution environment, which is the basis of Flink stream processing.
- **DataStream**: represents streaming data. DataStream can be regarded as unmodifiable collection that contains duplicate data. The number of elements in DataStream are unlimited.
- **KeyedStream**: generates the stream by performing keyBy grouping operations on DataStream. Data is grouped based on the specified key value.
- **WindowedStream**: generates the stream by performing the window function on KeyedStream, sets the window type, and defines window triggering conditions.
- **AllWindowedStream**: generates streams by performing the window function on DataStream, sets the window type, defines window triggering conditions, and performs operations on the window data.

- **ConnectedStreams**: generates streams by connecting the two DataStreams and retaining the original data type, and performs the map or flatMap operation.
- **JoinedStreams**: performs equijoin operation to data in the window. The join operation is a special scenario of coGroup operation.
- **CoGroupedStreams**: performs the coGroup operation on data in the window. CoGroupedStreams can perform various types of join operations.

Figure 11-59 Conversion of Flink stream types



Data Stream Source

Table 11-22 APIs about data stream source

API	Description
def fromElements[T: TypeInformation] (data: T*): DataStream[T]	Obtain user-defined data of multiple elements as the data stream source. data is the specific data of multiple elements.
def fromCollection[T: TypeInformation] (data: Seq[T]): DataStream[T]	Obtain the user-defined data collection as input data stream. data can be a data collection or a data body that can be iterated.
def fromCollection[T: TypeInformation] (data: Iterator[T]): DataStream[T]	
def fromParallelCollection[T: TypeInformation] (data: SplittableIterator[T]): DataStream[T]	Obtain the user-defined data collection as parallel data stream source. data indicates the iterator that can be divided into multiple partitions.

API	Description
<pre>def generateSequence(from: Long, to: Long): DataStream[Long]</pre>	<p>Obtain a sequence of user-defined data as the data stream source.</p> <ul style="list-style-type: none"> • from indicates the starting point of numbers. • to indicates the end point of numbers.
<pre>def readTextFile(filePath: String): DataStream[String]</pre>	<p>Obtain the user-defined text file from a specific path as the data stream source.</p> <ul style="list-style-type: none"> • filePath indicates the path of the text file. • charsetName indicates the encoding format.
<pre>def readTextFile(filePath: String, charsetName: String): DataStream[String]</pre>	
<pre>def readFile[T: TypeInformation] (inputFormat: FileInputFormat[T], filePath: String)</pre>	<p>Obtain the user-defined file from a specific path as the data stream source.</p> <ul style="list-style-type: none"> • filePath indicates the file path. • inputFormat indicates the format of the file. • watchType indicates the file processing mode, which can be PROCESS_ONCE or PROCESS_CONTINUOUSLY. • interval indicates the interval for processing directories or files.
<pre>def readFile[T: TypeInformation] (inputFormat: FileInputFormat[T], filePath: String, watchType: FileProcessingMode, interval: Long): DataStream[T]</pre>	
<pre>def socketTextStream(hostname: String, port: Int, delimiter: Char = '\n', maxRetry: Long = 0): DataStream[String]</pre>	<p>Obtain user-defined socket data as the data stream source.</p> <ul style="list-style-type: none"> • hostname indicates the host name of the socket server. • port indicates the listening port of the server. • delimiter and maxRetry are not supported by Scala APIs.
<pre>def addSource[T: TypeInformation] (function: SourceFunction[T]): DataStream[T]</pre>	<p>Customize the SourceFunction and addSource methods to add data sources such as Kafka. The implementation method is the run method of SourceFunction.</p> <ul style="list-style-type: none"> • function indicates the user-defined SourceFunction function. • Simplified format is supported by Scala.
<pre>def addSource[T: TypeInformation] (function: SourceContext[T] => Unit): DataStream[T]</pre>	

Data Output

Table 11-23 APIs about data output

API	Description
def print(): DataStreamSink[T]	Print data as the standard output stream.
def printToErr()	Print data as the standard error output stream.
def writeAsText(path: String): DataStreamSink[T]	Write data to a specific text file. <ul style="list-style-type: none"> • path indicates the path of the text file. • writeMode indicates the writing mode, which can be OVERWRITE or NO_OVERWRITE.
def writeAsText(path: String, writeMode: FileSystem.WriteMode): DataStreamSink[T]	
def writeAsCsv(path: String): DataStreamSink[T]	Write data to a specific .csv file. <ul style="list-style-type: none"> • path indicates the path of the text file. • writeMode indicates the writing mode, which can be OVERWRITE or NO_OVERWRITE. • rowDelimiter indicates the row separator. • fieldDelimiter indicates the column separator.
def writeAsCsv(path: String, writeMode: FileSystem.WriteMode): DataStreamSink[T]	
def writeAsCsv(path: String, writeMode: FileSystem.WriteMode, rowDelimiter: String, fieldDelimiter: String): DataStreamSink[T]	
def writeUsingOutputFormat(format: OutputFormat[T]): DataStreamSink[T]	Write data to a file, for example, a binary file.
def writeToSocket(hostname: String, port: Integer, schema: SerializationSchema[T]): DataStreamSink[T]	Write data to the socket connection. <ul style="list-style-type: none"> • hostName indicates the host name. • port indicates the port number.
def addSink(sinkFunction: SinkFunction[T]): DataStreamSink[T]	Export data in a user-defined manner. The flink-connectors allows addSink method to support exporting data to Kafka. The implementation method is the invoke method of SinkFunction.
def addSink(fun: T => Unit): DataStreamSink[T]	

Filtering and Mapping

Table 11-24 APIs about filtering and mapping

API	Description
def map[R: TypeInformation](fun: T => R): DataStream[R]	Transform an element into another element of the same type.
def map[R: TypeInformation](mapper: MapFunction[T, R]): DataStream[R]	
def flatMap[R: TypeInformation](flatMapter: FlatMapFunction[T, R]): DataStream[R]	Transform an element into zero, one, or multiple elements.
def flatMap[R: TypeInformation](fun: (T, Collector[R]) => Unit): DataStream[R]	
def flatMap[R: TypeInformation](fun: T => TraversableOnce[R]): DataStream[R]	
def filter(filter: FilterFunction[T]): DataStream[T]	Run a Boolean function on each element and retain elements that return true .
def filter(fun: T => Boolean): DataStream[T]	

Aggregation

Table 11-25 APIs about aggregation

API	Description
def keyBy(fields: Int*): KeyedStream[T, JavaTuple]	Logically divide a stream into multiple partitions, each containing elements with the same key. The partitioning is internally implemented using hash partition. A KeyedStream is returned. Return the KeyedStream after the KeyBy operation, and call the KeyedStream function, such as reduce, fold, min, minby, max, maxby, sum, and sumby.
def keyBy(firstField: String, otherFields: String*): KeyedStream[T, JavaTuple]	
	<ul style="list-style-type: none"> • fields indicates the IDs of certain columns • firstField and otherFields are names of member variables.

API	Description
def keyBy[K: TypeInformation](fun: T => K): KeyedStream[T, K]	<ul style="list-style-type: none"> key indicates the user-defined basis for partitioning.
def reduce(fun: (T, T) => T): DataStream[T]	Perform reduce on KeyedStream in a rolling manner. Aggregate the current element and the last reduced value into a new value. The types of the three values are the same.
def reduce(reducer: ReduceFunction[T]): DataStream[T]	
def fold[R: TypeInformation](initialValue: R)(fun: (R,T) => R): DataStream[R]	Perform fold operation on KeyedStream based on an initial value in a rolling manner. Aggregate the current element and the last folded value into a new value. The input and returned values of Fold can be different.
def fold[R: TypeInformation](initialValue: R, folder: FoldFunction[T,R]): DataStream[R]	
def sum(position: Int): DataStream[T]	Calculate the sum in a KeyedStream in a rolling manner. position and field indicate calculating the sum of a specific column.
def sum(field: String): DataStream[T]	
def min(position: Int): DataStream[T]	Calculate the minimum value in a KeyedStream in a rolling manner. min returns the minimum value, without guarantee of the correctness of columns of non-minimum values. position and field indicate calculating the minimum value of a specific column.
def min(field: String): DataStream[T]	
def max(position: Int): DataStream[T]	Calculate the maximum value in KeyedStream in a rolling manner. max returns the maximum value, without guarantee of the correctness of columns of non-maximum values. position and field indicate calculating the maximum value of a specific column.
def max(field: String): DataStream[T]	
def minBy(position: Int): DataStream[T]	Obtain the row where the minimum value of a column locates in a KeyedStream. minBy returns all elements of that row. position and field indicate the column on which the minBy operation is performed.
def minBy(field: String): DataStream[T]	
def maxBy(position: Int): DataStream[T]	Obtain the row where the maximum value of a column locates in a

API	Description
def maxBy(field: String): DataStream[T]	KeyedStream. maxBy returns all elements of that row. position and field indicate the column on which the maxBy operation is performed.

DataStream Distribution

Table 11-26 APIs about DataStream distribution

API	Description
def partitionCustom[K: TypeInfoInformation](partitioner: Partitioner[K], field: Int) : DataStream[T]	Use a user-defined partitioner to select target task for each element. <ul style="list-style-type: none"> • partitioner indicates the user-defined method for repartitioning. • field indicates the input parameters of partitioner. • keySelector indicates the user-defined input parameters of partitioner.
def partitionCustom[K: TypeInfoInformation](partitioner: Partitioner[K], field: String):DataStream[T]	
def partitionCustom[K: TypeInfoInformation](partitioner: Partitioner[K], fun: T => K): DataStream[T]	
def shuffle: DataStream[T]	Randomly and evenly partition elements.
def rebalance: DataStream[T]	Partition elements in round-robin manner, ensuring load balance of each partition. This partitioning is helpful to data with skewness.
def rescale: DataStream[T]	Distribute elements into downstream subset in round-robin manner. NOTE The method for checking the code is similar to the rebalance method.
def broadcast: DataStream[T]	Broadcast each element to all partitions.

Configuring the eventtime Attribute

Table 11-27 APIs about configuring the eventtime attribute

API	Description
<pre>def assignTimestampsAndWatermarks(assigner: AssignerWithPeriodicWatermarks[T]): DataStream[T]</pre>	Extract timestamp from records, enabling event time window to trigger computing.
<pre>def assignTimestampsAndWatermarks(assigner: AssignerWithPunctuatedWatermarks[T]): DataStream[T]</pre>	

Iteration

Table 11-28 APIs about iteration

API	Description
<pre>def iterate[R] (stepFunction: DataStream[T] => (DataStream[T], DataStream[R]),maxWaitTimeMillis:Long = 0,keepPartitioning: Boolean = false) : DataStream[R]</pre>	In the flow, create in a feedback loop to redirect the output of an operator to a preceding operator. NOTE <ul style="list-style-type: none"> This API is helpful to algorithms that require constant update of models. long maxWaitTimeMillis: The timeout period of each round of iteration.
<pre>def iterate[R, F: TypeInformation] (stepFunction: ConnectedStreams[T, F] => (DataStream[F], DataStream[R]),maxWaitTimeMillis:Long): DataStream[R]</pre>	

Stream Splitting

Table 11-29 APIs about stream splitting

API	Description
def split(selector: OutputSelector[T]): SplitStream[T]	Use OutputSelector to rewrite select method, specify stream splitting basis (by tagging), and construct SplitStream streams. That is, a string is tagged for each element, so that a stream of a specific tag can be selected or created.
def select(outputNames: String*): DataStream[T]	Select one or multiple streams from a SplitStream. outputNames indicates the sequence of string tags created for each element using the split method.

Window

Windows can be classified as tumbling windows and sliding windows.

- APIs such as Window, TimeWindow, CountWindow, WindowAll, TimeWindowAll, and CountWindowAll can be used to generate windows.
- APIs such as Window Apply, Window Reduce, Window Fold, and Aggregations on windows can be used to operate windows.
- Multiple Window Assigners are supported, including TumblingEventTimeWindows, TumblingProcessingTimeWindows, SlidingEventTimeWindows, SlidingProcessingTimeWindows, EventTimeSessionWindows, ProcessingTimeSessionWindows, and GlobalWindows.
- ProcessingTime, EventTime, and IngestionTime are supported.
- Timestamp modes EventTime: AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks are supported.

[Table9 APIs for generating windows](#) lists APIs for generating windows.

Table 11-30 APIs for generating windows

API	Description
def window[W <: Window](assigner: WindowAssigner[_ >: T, W]): WindowedStream[T, K, W]	Define windows in partitioned KeyedStreams. A window groups each key according to some characteristics (for example, data received within the latest 5 seconds. For details about windows, see: https://ci.apache.org/projects/flink/flink-docs-release-1.10/dev/event_time.html).

API	Description
<pre>def windowAll[W <: Window](assigner: WindowAssigner[_ >: T, W]): AllWindowedStream[T, W]</pre>	Define windows in DataStreams.
<pre>def timeWindow(size: time WindowedStream[T, K, TimeWindow]</pre>	Define time windows in partitioned KeyedStreams, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters. <ul style="list-style-type: none"> • size indicates the duration of the window. • slide indicates the sliding time of window. NOTE <ul style="list-style-type: none"> • WindowedStream and AllWindowedStream indicates two types of streams. • If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.
<pre>def countWindow(size: Long, slide: Long): WindowedStream[T, K, GlobalWindow]</pre>	
<pre>def timeWindowAll(size: time AllWindowedStream[T, TimeWindow]</pre>	Define time windows in partitioned DataStream, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters. <ul style="list-style-type: none"> • size indicates the duration of the window. • slide indicates the sliding time of window.
<pre>def timeWindowAll(size: Time, slide: time AllWindowedStream[T, TimeWindow]</pre>	
<pre>def countWindow(size: Long, slide: Long): WindowedStream[T, K, GlobalWindow]</pre>	Divide windows according to the number of elements and define windows in partitioned KeyedStreams. <ul style="list-style-type: none"> • size indicates the duration of the window. • slide indicates the sliding time of window. NOTE <ul style="list-style-type: none"> • WindowedStream and AllWindowedStream indicates two types of streams. • If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.
<pre>def countWindow(size: Long): WindowedStream[T, K, GlobalWindow]</pre>	
<pre>def countWindowAll(size: Long, slide: Long): AllWindowedStream[T, GlobalWindow]</pre>	Divide windows according to the number of elements and define windows in DataStreams. <ul style="list-style-type: none"> • size indicates the duration of the window. • slide indicates the sliding time of window.

API	Description
def countWindowAll(size: Long): AllWindowedStream[T, GlobalWindow]	

Table 11-31 lists APIs for operating windows.

Table 11-31 APIs for operating windows

Method	API	Description
Window	def apply[R: TypeInformation] (function: WindowFunction[T, R, K, W]): DataStream[R]	Apply a general function to a window. The data in the window is calculated as a whole.
	def apply[R: TypeInformation] (function: (K, W, Iterable[T], Collector[R]) => Unit): DataStream[R]	function indicates the window function to be executed.
	def reduce(function: ReduceFunction[T]): DataStream[T]	Apply a reduce function to the window and return the result.
	def reduce(function: (T, T) => T): DataStream[T]	<ul style="list-style-type: none"> • reduceFunction indicates the reduce function to be executed. • function of WindowFunction indicates triggering an operation to the window after a reduce operation.
	def reduce[R: TypeInformation] (preAggregator: ReduceFunction[T], function: WindowFunction[T, R, K, W]): DataStream[R]	
	def reduce[R: TypeInformation] (preAggregator: (T, T) => T, windowFunction: (K, W, Iterable[T], Collector[R]) => Unit): DataStream[R]	
	def fold[R: TypeInformation] (initialValue: R, function: FoldFunction[T,R]): DataStream[R]	Apply a fold function to the window and return the result.
def fold[R: TypeInformation] (initialValue: R)(function: (R, T) => R): DataStream[R]	<ul style="list-style-type: none"> • initialValue indicates the initial value. • foldFunction indicates the fold function. 	

Method	API	Description
	<pre>def fold[ACC: TypelInformation, R: TypelInformation](initialValue: ACC, foldFunction: FoldFunction[T, ACC], function: WindowFunction[ACC, R, K, W]): DataStream[R]</pre>	<ul style="list-style-type: none"> • function of WindowFunction indicates triggering an operation to the window after a fold operation.
	<pre>def fold[ACC: TypelInformation, R: TypelInformation](initialValue: ACC, foldFunction: (ACC, T) => ACC, windowFunction: (K, W, Iterable[ACC], Collector[R]) => Unit): DataStream[R]</pre>	
Window All	<pre>def apply[R: TypelInformation] (function: AllWindowFunction[T, R, W]): DataStream[R]</pre>	Apply a general function to a window. The data in the window is calculated as a whole.
	<pre>def apply[R: TypelInformation] (function: (W, Iterable[T], Collector[R]) => Unit): DataStream[R]</pre>	
	<pre>def reduce(function: ReduceFunction[T]): DataStream[T]</pre>	<p>Apply a reduce function to the window and return the result.</p> <ul style="list-style-type: none"> • reduceFunction indicates the reduce function to be executed. • AllWindowFunction indicates triggering an operation to the window after a reduce operation.
	<pre>def reduce(function: (T, T) => T): DataStream[T]</pre>	
	<pre>def reduce[R: TypelInformation] (preAggregator: ReduceFunction[T], windowFunction: AllWindowFunction[T, R, W]): DataStream[R]</pre>	
	<pre>def reduce[R: TypelInformation] (preAggregator: (T, T) => T, windowFunction: (W, Iterable[T], Collector[R]) => Unit): DataStream[R]</pre>	

Method	API	Description
	<pre>def fold[R: TypeInformation] (initialValue: R, function: FoldFunction[T,R]): DataStream[R]</pre>	<p>Apply a fold function to the window and return the result.</p> <ul style="list-style-type: none"> • initialValue indicates the initial value. • foldFunction indicates the fold function. • function of WindowFunction indicates triggering an operation to the window after a fold operation.
	<pre>def fold[R: TypeInformation] (initialValue: R)(function: (R, T) => R): DataStream[R]</pre>	
	<pre>def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, preAggregator: FoldFunction[T, ACC], windowFunction: AllWindowFunction[ACC, R, W]): DataStream[R]</pre>	
	<pre>def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, preAggregator: (ACC, T) => ACC, windowFunction: (W, Iterable[ACC], Collector[R]) => Unit): DataStream[R]</pre>	
Window and Window All	<pre>def sum(position: Int): DataStream[T]</pre>	Sum a specified column of the window data.
	<pre>def sum(field: String): DataStream[T]</pre>	field and position indicate a specific column of the data.
	<pre>def min(position: Int): DataStream[T]</pre>	Calculate the minimum value of a specified column of the window data. min returns the minimum value, without guarantee of the correctness of columns of non-minimum values. position and field indicate calculating the minimum value of a specific column.
	<pre>def min(field: String): DataStream[T]</pre>	
	<pre>def max(position: Int): DataStream[T]</pre>	Calculate the maximum value of a specified column of the window data. max returns the maximum value, without guarantee of the correctness of columns of non-maximum values. position and field indicate calculating the maximum value of a specific column.
	<pre>def max(field: String): DataStream[T]</pre>	

Method	API	Description
	def minBy(position: Int): DataStream[T]	Obtain the row where the minimum value of a column locates in the window data. minBy returns all elements of that row. position and field indicate the column on which the minBy operation is performed.
	def minBy(field: String): DataStream[T]	
	def maxBy(position: Int): DataStream[T]	Obtain the row where the maximum value of a column locates in the window data. maxBy returns all elements of that row. position and field indicate the column on which the maxBy operation is performed.
	def maxBy(field: String): DataStream[T]	

Combining Multiple DataStreams

Table 11-32 APIs about combining multiple DataStreams

API	Description
def union(dataStreams: DataStream[T]*): DataStream[T]	Perform union operation on multiple DataStreams to generate a new data stream containing all data from original DataStreams. NOTE If you perform union operation on a piece of data with itself, there are two copies of the same data.
def connect[T2] (dataStream: DataStream[T2]): ConnectedStreams[T, T2]	Connect two DataStreams and retain the original type. The connect API method allows two DataStreams to share statuses. After ConnectedStreams is generated, map or flatMap operation can be called.
def map[R: TypeInformation] (coMapper: CoMapFunction[IN1, IN2, R]): DataStream[R]	Perform mapping operation, which is similar to map and flatMap operation in a ConnectedStreams, on elements. After the operation, the type of the new DataStreams is string.
def map[R: TypeInformation](fun1: IN1 => R, fun2: IN2 => R): DataStream[R]	

API	Description
<pre>def flatMap[R: TypeInformation] (coFlatMapper: CoFlatMapFunction[IN1, IN2, R]): DataStream[R]</pre>	<p>Perform union operation on multiple DataStreams to generate a new data stream containing all data from original DataStreams.</p> <p>NOTE If you perform union operation on a piece of data with itself, there are two copies of the same data.</p>
<pre>def flatMap[R: TypeInformation](fun1: (IN1, Collector[R]) => Unit, fun2: (IN2, Collector[R]) => Unit): DataStream[R]</pre>	
<pre>def flatMap[R: TypeInformation] (fun1: IN1 => TraversableOnce[R], fun2: IN2 => TraversableOnce[R]): DataStream[R]</pre>	

Join Operation

Table 11-33 APIs about join operation

API	Description
<pre>def join[T2] (otherStream: DataStream[T2]): JoinedStreams[T, T2]</pre>	<p>Join two DataStreams using a given key in a specified window.</p> <p>The key value of the join operation is specified by the where and equalTo method, indicating filtering data with equivalent conditions from two DataStreams.</p>
<pre>def coGroup[T2] (otherStream: DataStream[T2]): CoGroupedStreams[T, T2]</pre>	<p>Co-group two DataStreams using a given key in a specified window.</p> <p>The key value of the coGroup operation is specified by the where and equalTo method, indicating partitioning two DataStreams using equivalent conditions.</p>

11.5.2 Overview of RESTful APIs

Flink has a monitoring API that can be used to query status and statistics of running jobs, as well as recent completed jobs. This monitoring API is used by Apache Flink Dashboard.

The monitoring API is a RESTful API that accepts HTTP GET requests and responds with JSON data. RESTful API is a set of APIs used to log in to the web server. In

Flink, web server is a module of JobManager and shares the same process with JobManager. By default, the listening port of web server is 8081. If you want to change the listening port, modify **jobmanager.web.port** in the **flink-conf.yaml** file.

The *Netty* and the *Netty Router* library are used to handle REST requests and analysis URLs.

RESTful APIs are executed through HTTP requests.

The format of the HTTP requests is `http://<JobManager_IP>:<JobManager_Port><Path>`

The *JobManager_IP* indicates the IP address of JobManager, *JobManager_Port* indicates the listening port of JobManager, and *Path* indicates the path. For details, see [Table 11-34](#). For example, `http://10.162.181.57:32261/config`.

 **NOTE**

If you want to modify the configuration file **flink-conf.yaml** of the Flink Client, add to-be-visited IP addresses (separated with commas) in **jobmanager.web.allow-access-address** and **jobmanager.web.access-control-allow-origin** parameters.

[Table 11-34](#) lists all RESTful API paths supported by Flink.

Table 11-34 Paths supported by Flink

Path	Description
/config	Some information about the monitoring API and the server setup.
/logout	Some information about the logout.
/overview	Simple summary of the Flink cluster status.
/jobs	IDs of the jobs, grouped by status <i>running</i> , <i>finished</i> , <i>failed</i> , <i>canceled</i> .
/jobmanager/config	the configuration of the jobmanager.
/joboverview	Jobs, grouped by status, each with a small summary of its status.
/joboverview/running	Jobs, grouped by status, each with a small summary of its status. The same as /joboverview , but containing only currently running jobs.
/joboverview/completed	Jobs, grouped by status, each with a small summary of its status. The same as /joboverview , but containing only completed (finished, canceled, or failed) jobs.

Path	Description
/jobs/<jobid>	Summary of one job, listing dataflow plan, status, timestamps of state transitions, aggregate information for each vertex (operator).
/jobs/<jobid>/vertices	Currently the same as /jobs/<jobid> .
/jobs/<jobid>/config	The user-defined execution config used by the job.
/jobs/<jobid>/exceptions	The non-recoverable exceptions that have been observed by the job. The truncated flag defines whether more exceptions occurred, but are not listed, because the response would otherwise get too big.
/jobs/<jobid>/accumulators	The aggregated user accumulators plus job accumulators.
/jobs/<jobid>/checkpoints	checkpoint stats for a job.
/jobs/<jobid>/metrics	a job a list of all available metrics.
/jobs/<jobid>/vertices/<vertexid>	Information about one specific vertex, with a summary for each of its subtasks.
/jobs/<jobid>/vertices/<vertexid>/subtasktimes	This request returns the timestamps for the state transitions of all subtasks of a given vertex. These can be used, for example, to create time-line comparisons between subtasks.
/jobs/<jobid>/vertices/<vertexid>/taskmanagers	TaskManager statistics for one specific vertex. This is an aggregation of subtask statistics returned by /jobs/<jobid>/vertices/<vertexid> .
/jobs/<jobid>/vertices/<vertexid>/accumulators	The aggregated user-defined accumulators, for a specific vertex.
/jobs/<jobid>/vertices/<vertexid>/checkpoints	checkpoint stats for a single job vertex.
/jobs/<jobid>/vertices/<vertexid>/backpressure	back pressure stats for a single job vertex and all its sub tasks.
/jobs/<jobid>/vertices/<vertexid>/metrics	a given task of the values for a set of metrics.

Path	Description
/jobs/<jobid>/vertices/<vertexid>/subtasks/accumulators	Gets all user-defined accumulators for all subtasks of a given vertex. These are the individual accumulators that are returned in aggregated form by the request /jobs/<jobid>/vertices/<vertexid>/accumulators .
/jobs/<jobid>/vertices/<vertexid>/subtasks/<subtasknum>	Summary of the current or latest execution attempt of a specific subtask. See below for a sample.
/jobs/<jobid>/vertices/<vertexid>/subtasks/<subtasknum>/attempts/<attempt>	Summary of a specific execution attempt of a specific subtask. Multiple execution attempts happen in case of failure/recovery.
/jobs/<jobid>/vertices/<vertexid>/subtasks/<subtasknum>/attempts/<attempt>/accumulators	The accumulators collected for one specific subtask during one specific execution attempt (multiple attempts happen in case of failure/recovery).
/jobs/<jobid>/plan	The dataflow plan of a job. The plan is also included in the job summary (/jobs/<jobid>).
/taskmanagers	Some information of the TaskManagers.
/taskmanagers/<taskmanagerid>/metrics	the metrics information of a TaskManager.
/taskmanagers/<taskmanagerid>/log	the log information of a TaskManager.
/taskmanagers/<taskmanagerid>/stdout	the stdout of a TaskManager.
/jobmanager/log	the log of the jobmanager.
/jobmanager/stdout	the stdout of the jobmanager.
/jobmanager/metrics	the metrics of the jobmanager.
/*	services requests to web frontend's static files, such as HTML, CSS, or JS files.

Table 11-35 describes variables listed in **Table 11-34**.

Table 11-35 Description of variables

Variable	Description
jobid	ID of jobs
vertexid	Vertexes ID of the flow diagram.
subtasknum	Sum of subtasks.
attempt	Times of attempts
taskmanagerid	ID of TaskManager

11.5.3 Overview of Savepoints CLI

Overview

Savepoints are externally stored checkpoints that you can use to stop-and-resume or update your Flink programs. They use Flink's checkpoint mechanism to create a snapshot of the state of your streaming program and write the checkpoint meta data out to an external file system.

It is highly recommended that you adjust your programs as described in this section in order to be able to upgrade your programs in the future. The main required change is to manually specify operator IDs via the **uid(String)** method. These IDs are used to scope the state of each operator.

```
DataStream<String> stream = env
//Statefulsource(e.g.Kafka)withID
.addSource(new StatefulSource())
.uid("source-id") //IDforthesourceoperator
.shuffle()
//StatefulmapperwithID
.map(new StateFulMapper())
.uid("mapper-id") //IDforthemapper
//Statelessprintingsink
.print(); //Auto-generatedID
```

Savepoint Recovery

If you do not specify the IDs manually, they will be generated automatically. You can automatically restore from the savepoint if these IDs do not change. The generated IDs depend on the structure of your program and are sensitive to program changes. Therefore, it is highly recommended to assign these IDs manually. When a savepoint is triggered, a single savepoint file will be created containing the checkpoint metadata. The actual checkpoint state will be kept around in the configured checkpoint directory, for example, with a FsStateBackend or RocksDBStateBackend:

1. Trigger a savepoint.
\$ bin/flink savepoint jobId [targetDirectory]

This command will trigger a savepoint for the job with ID:*jobid*. Furthermore, you can specify a target file system directory to store the savepoint. The directory must be accessible by JobManager. The targetDirectory is optional. If

`targetDirectory` is not configured, the directory specified by **`state.savepoints.dir`** in the configuration file is used to store savepoint.

You can configure a default savepoint target directory via the **`state.savepoints.dir`** key in the **`flink-conf.yaml`** file.

```
# Default savepoint target directory
```

NOTE

You are advised to configure `targetDirectory` to an HDFS path.

For example:

```
bin/flink savepoint 405af8c02cf6dc069a0f9b7a1f7be088 hdfs://savepoint.
```

2. Cancel a job with a savepoint.

```
$ bin/flink cancel -s [targetDirectory] jobId
```

This will atomically trigger a savepoint for the job with ID:*jobid* and cancel the job. Furthermore, you can specify a target file system directory to store the savepoint. The directory must be accessible by JobManager.

3. Resume jobs.

- Resume from a savepoint.

```
$ bin/flink run -s savepointPath [runArgs]
```

This command submits a job and specifies the savepoint path. The execution will resume from the respective savepoint state.

NOTE

`runArgs` is a user-defined parameter with parameter format and name varying depending on users.

- Allow non-restored state.

```
$ bin/flink run -s savepointPath -n [runArgs]
```

By default the resume operation will try to map all state of the savepoint back to the program you are restoring with. If you dropped an operator, you can skip the state that cannot be mapped to the new program via `-allowNonRestoredState` (short: `-n`).

4. Dispose savepoints.

```
$ bin/flink savepoint -d savepointPath
```

This command disposes the savepoint stored in: *savepointPath*.

Precautions

- Chained operators are identified by the ID of the first task. It is not possible to manually assign an ID to an intermediate chained task, for example, in the chain [a -> b -> c] only **a** can have its ID assigned manually, but not **b** or **c**. To work around this, you can manually define the task chains. To manually define chains, use the **`disableChaining()`** interface. See the following example:

```
env.addSource(new GetDataSource())
  .keyBy(0)
  .timeWindow(Time.seconds(2)).uid("window-id")
  .reduce(_+_).uid("reduce-id")
  .map(f=>(f,1)).disableChaining().uid("map-id")
  .print().disableChaining().uid("print-id")
```

- During job upgrade, the data type of operators cannot be changed.

11.5.4 Introduction to Flink Client CLI

Common CLIs

Common Flink CLIs are as follows:

1. `yarn-session.sh`

- You can run **yarn-session.sh** to start a standing Flink cluster to receive tasks submitted by clients. Run the following command to start a Flink cluster with three TaskManager instances:

```
bin/yarn-session.sh
```

- Run the following command to obtain other parameters of **yarn-session.sh**:

```
bin/yarn-session.sh -help
```

2. `flink`

- You can run Flink commands to submit a Flink job to a standing Flink cluster or to execute the job in single-server mode.

- Run the following command to submit a Flink job to a standing Flink cluster:

```
bin/flink run ../examples/streaming/WindowJoin.jar
```

 **NOTE**

Before using the command to submit a task, you need to run **yarn-session.sh** to start the Flink cluster.

- Run the following command to execute a per-job YARN Cluster mode:

```
bin/flink run -m yarn-cluster ../examples/streaming/WindowJoin.jar
```

 **NOTE**

The `-m yarn-cluster` parameter is used to specify a job to independently start a Flink cluster.

- List scheduled and running jobs (including their JobIDs):

```
bin/flink list
```

- Cancel a job:

```
bin/flink cancel <jobID>
```

- Stop a job (streaming jobs only):

```
bin/flink stop <jobID>
```

 **NOTE**

The difference between cancelling and stopping a (streaming) job is the following:

- Cancel a job: On a cancel call, the operators in a job immediately receive a `cancel()` method call to cancel them as soon as possible. If operators are not stopping after the cancel call, Flink will start interrupting the thread periodically until it stops.
- Stop a job: Stop is only available for jobs which use sources that implement the `StoppableFunction` interface. The job will keep running until all sources properly shut down. Stop a job is more graceful than cancel, but it may cause the job to stop failing.

- Run the following command to obtain other parameters of Flink commands:
`bin/flink --help`

Precautions

- If **yarn-session.sh** uses **-z** to configure the specified ZooKeeper NameSpace, you need to use **-yid** to specify the applicationID and use **-yz** to specify the ZooKeeper NameSpace when using **flink run**. The NameSpaces must be the same.

Example:

```
bin/yarn-session.sh -z YARN101  
bin/flink run -yid application_****_**** -yz YARN101 examples/streaming/WindowJoin.jar
```

- If **yarn-session.sh** does not use **-z** to configure the specified ZooKeeper NameSpace, do not use **-yz** to specify the ZooKeeper NameSpace when using **flink run**.

Example:

```
bin/yarn-session.sh  
bin/flink run examples/streaming/WindowJoin.jar
```

- You can use **-yz** to specify a ZooKeeper NameSpace when using **flink run -m yarn-cluster** to start a cluster.
- A NameSpace cannot be shared by multiple clusters.
- If you use **-z** to specify a ZooKeeper NameSpace when starting a cluster or submitting a job, you need to use **-z** again to specify the NameSpace when deleting, stopping, or querying the job or triggering the savepoint.

11.5.5 FAQ

11.5.5.1 Savepoints-related Problems

1. Should I assign IDs to all operators of the job?

Strictly speaking, you can assign IDs to only operators with statuses because savepoints save only statuses of operators that have statuses and not operators without statuses.

However, in actual situations, you are advised to allocate IDs to all operators because some internal operators, such as window operators of Flink have statuses. Whether an operator has status or not is not obvious. If you are specific that an operator does not have status, you do not need to call **uid()** to allocate an ID to the operator.

2. What would be the impact if I add an operator with status while upgrading the job?

If you add an operator with status to the job, the status of the operator is not saved in the savepoint and thus the status cannot be recovered. The operator is processed as an operator without status and is executed from the start.

3. What would be the impact if I delete an operator with status while upgrading the job?

By default, savepoints attempt to recover all saved statuses. If the savepoint saves the status of the deleted operator, recovery fails.

You can run the following command and use the **-allowNonRestoredState** (-n in the following command) parameter to skip recovering the status of the deleted operator:

```
$ bin/flink run -s savepointPath -n [runArgs]
```

4. What would be the impact if I rearrange the sequence of operators with statuses?
 - If you have allocated IDs to the operators, the statuses would be recovered normally.
 - If you do not allocate IDs to the operators, IDs would be automatically allocated to the operators in the new sequence. Then, the status recovery would fail.
5. What would be the impact if I delete or add an operator without status or rearrange the sequence of operators without statuses?
 - If you have allocated IDs to operators with statuses, operators without statuses do not affect status recovery from savepoints.
 - If you do not allocate IDs to operators, operators with statuses may be allocated with new IDs due to the sequence change. This would cause status recovery failure.

6. What would be the impact if I change the operator concurrency during the status recovery?

If the Flink version is higher than 1.2.0 and discarded status APIs, such as checkpointed, are not used, you can recover statuses from savepoints. Otherwise, statuses cannot be recovered.

11.5.5.2 What If the Chrome Browser Cannot Display the Title

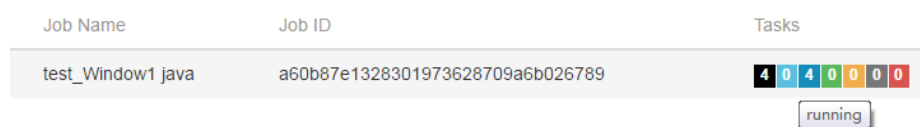
Question

What to do if the title is not displayed when I use Chrome browser to access the Apache Flink Dashboard? This section takes the Tasks field as an example. When the pointer is placed on a colorful box in Tasks, the title of the box is not displayed, as shown in [Figure 11-60](#). [Figure 11-61](#) shows the normal situation when the title is displayed.

Figure 11-60 Title not displayed on the page



Figure 11-61 Title displayed normally



Answer

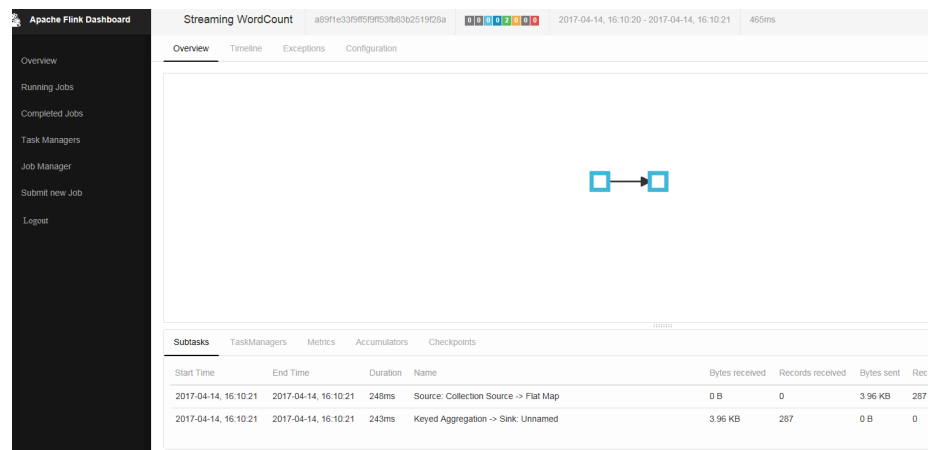
If the Chrome browser does not display the title, perform the following procedure:
Check whether a tool that affects the tooltip display by the Chrome browser is running. If so, shut down the tool.

11.5.5.3 What If the Page Is Displayed Abnormally on Internet Explorer 10/11

Question

What to do if Internet Explorer 10/11 does not display operator texts, as shown in [Figure 11-62](#)?

Figure 11-62 Page displayed abnormally on Internet Explorer 10/11



Answer

Flink uses the foreignObject element to draw scalable vector graphics (SVGs) but Internet Explorer 10/11 does not support foreignObject and thus operators cannot be displayed normally. Google Chrome browser is supported.

11.5.5.4 What If Checkpoint Is Executed Slowly in RocksDBStateBackend Mode When the Data Amount Is Large

Question

What to do if checkpoint is executed slowly in RocksDBStateBackend mode when the data amount is large?

Cause Analysis

Customized windows are used and the window state is ListState. There are many values under the same key. In the case of a new value, the merge operation of RocksDB is used. When calculation is triggered, all values under the key are read.

- The RocksDB mode is merge() > merge() ... > merge() > read(). Data reading in this mode consumes much time, as shown in [Figure 11-63](#).

- The source operator sends a large amount of data in a short period of time and the data keys are the same. The window operator fails to process data fast enough and barriers accumulate in the cache. The time consumed for snapshot preparation is too long and the window operator cannot report snapshot completion to CheckpointCoordinator in the specified time. Therefore, CheckpointCoordinator determines snapshot preparation failure, as shown in [Figure 11-64](#).

Figure 11-63 Time monitoring

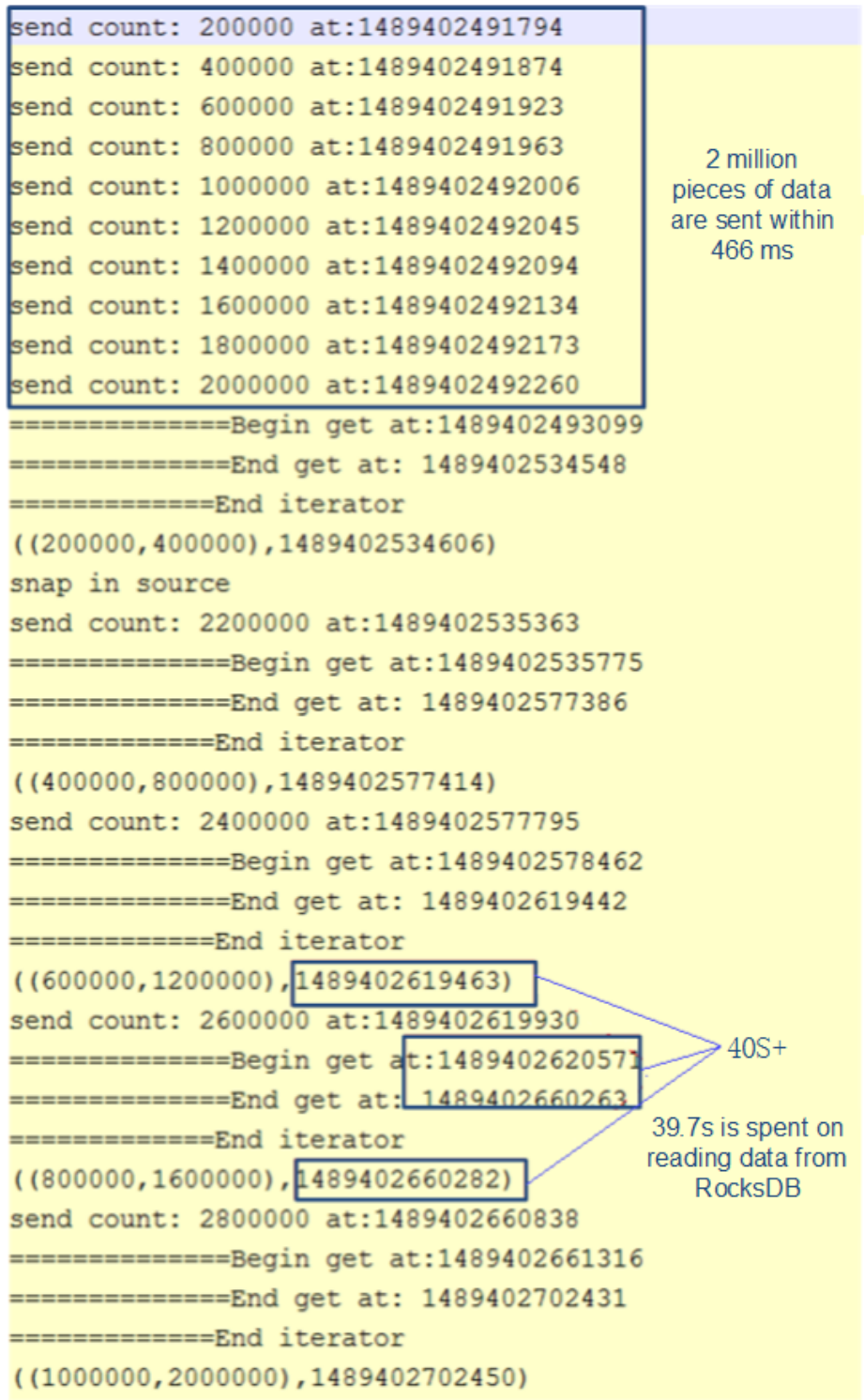
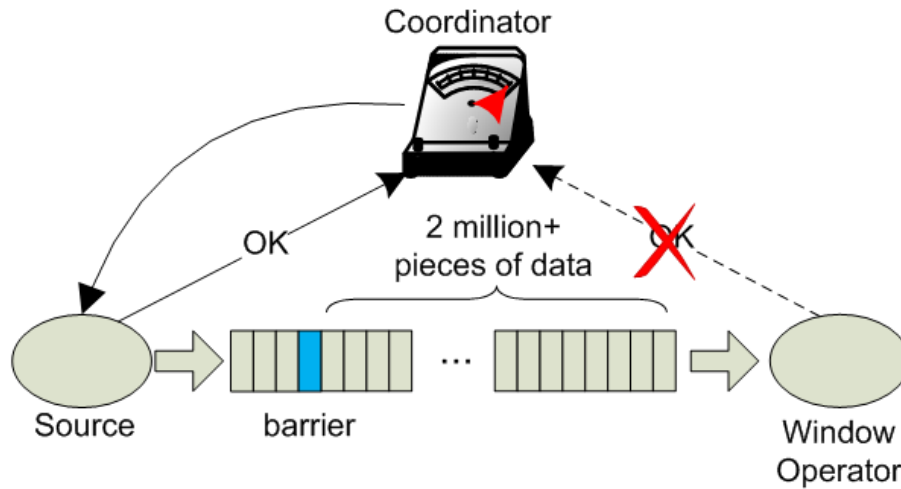


Figure 11-64 Relationship



Answer

Flink introduces the third-party software package RocksDB, whose defect causes the problem. You are advised to set checkpoint to FsStateBackend mode.

Set checkpoint to FsStateBackend mode in the application code as follows:

```
env.setStateBackend(new FsStateBackend("hdfs://hacluster/flink/checkpoint/"));
```

11.5.5.5 What If yarn-session Start Fails When blob.storage.directory Is Set to /home

Question

When **blob.storage.directory** is set to **/home**, the **blobStore-UUID** file cannot be created in **/home**. This causes yarn-session start failure

Answer

Step 1 It is recommended that **blob.storage.directory** be set to **/tmp** or **/opt/huawei/Bigdata/tmp**.

Step 2 When you set **blob.storage.directory** to a customized directory, manually grant permissions to the directory. This section takes the **admin** user of FusionInsight as an example.

1. Modify **conf/flink-conf.yaml** on the Flink client and run the **blob.storage.directory: /home/testdir/testdirdir/xxx** command.
2. Create the **/home/testdir** directory (level 1 is enough) and set the directory to be managed by the **admin** user.

```
SZV1000064084:/home # id admin
uid=20000(admin) gid=9998(ficommon) groups=9998(ficommon),8003(System_administrator_186)
SZV1000064084:/home # chown admin:ficommon testdir/ -R
```

NOTE

The testdirdir/xxx directory under the /home/testdir/ directory is automatically created on each node when Flink cluster starts.

3. Run `./bin/yarn-session.sh -jm 2048 -tm 3072` on the client to check that `yarn-session` is normally started and the directory is successfully created.

```
SZV1000064084:/home # ll testdir/
total 4
drwxr-x-- 3 admin ficommon 4096 Mar 13 11:55 testdir
SZV1000064084:/home # ll testdir/testdir/
total 4
drwxr-x-- 4 admin ficommon 4096 Mar 13 11:55 xxx
SZV1000064084:/home # ll testdir/testdir/xxx/
total 8
drwxr-x-- 2 admin ficommon 4096 Mar 13 11:55 blobStore-6fb3f049-ecf3-49ac-9fc9-95ad0aeeffd3
drwxr-x-- 2 admin ficommon 4096 Mar 13 11:55 blobStore-ad89b118-8545-4ece-8cae-1334b01de857
```

----End

11.5.5.6 Why Does Non-static KafkaPartitioner Class Object Fail to Construct FlinkKafkaProducer010?

Question

After Flink kernel is upgraded to 1.3.0 or later versions, if Kafka calls the `FlinkKafkaProducer010` that contains the non-static `KafkaPartitioner` class object as parameter to construct functions, an error is reported.

The error message is as follows:

```
org.apache.flink.api.common.InvalidProgramException: The implementation of the FlinkKafkaPartitioner is not serializable. The object probably contains or references non serializable fields.
```

Answer

In the 1.3.0 version of Flink, the `FlinkKafkaDelegatePartitioner` class is added, so that Flink allows APIs that use `KafkaPartitioner`, for example, `FlinkKafkaProducer010` that contains `KafkaPartitioner` object, to construct functions.

The `FlinkKafkaDelegatePartitioner` class defines the member variable `kafkaPartitioner`.

```
private final KafkaPartitioner<T> kafkaPartitioner;
```

When Flink input parameter `KafkaPartitioner` constructs `FlinkKafkaProducer010`, the call stack is as follows:

```
FlinkKafkaProducer010(String topicId, KeyedSerializationSchema<T> serializationSchema, Properties producerConfig, KafkaPartitioner<T> customPartitioner)
-> FlinkKafkaProducer09(String topicId, KeyedSerializationSchema<IN> serializationSchema, Properties producerConfig, FlinkKafkaPartitioner<IN> customPartitioner)
----> FlinkKafkaProducerBase(String defaultTopicId, KeyedSerializationSchema<IN> serializationSchema, Properties producerConfig, FlinkKafkaPartitioner<IN> customPartitioner)
-----> ClosureCleaner::clean(Object func, boolean checkSerializable)
```

Run the `KafkaPartitioner` object to construct a `FlinkKafkaDelegatePartitioner` object, and then check whether the object can be serializable. The `ClosureCleaner::clean` function is a static function. If the `KafkaPartitioner` object in a case is non-static, the `ClosureCleaner::clean` function cannot access the non-static member variable `kafkaPartitioner` in the `KafkaDelegatePartitioner` class and an exception is reported.

Either of the following methods can be used to solve the problem:

- Change the KafkaPartitioner class into static class.
- Use the FlinkKafkaProducer010 that contains FlinkKafkaPartitioner as the parameter to construct functions. In this case, FlinkKafkaDelegatePartitioner is not constructed and the exception about member variable is avoided.

11.5.5.7 When I Use a Newly Created Flink User to Submit Tasks, Why Does the Task Submission Fail and a Message Indicating Insufficient Permission on ZooKeeper Directory Is Displayed?

Question

When I use a newly-created Flink user to submit tasks, the task submission fails because of insufficient permission on the ZooKeeper directory. The error message in the log is as follows:

```
NoAuth for /flink_base/flink/application_1499222480199_0013
```

Answer

1. Check whether the permission on the /flink_base directory in ZooKeeper is 'world,'anyone: cdrwa; If no, change the permission on the /flink_base directory to 'world,'anyone: cdrwa and go to [step 2](#). If yes, go to [step 2](#).
2. In the configuration file of Flink, the default value of high-availability.zookeeper.client.acl is creator, indicating that only the creator of the directory has permission on it. The user created later has no access to the /flink_base/flink directory in ZooKeeper because only the user created earlier has permission on it.

To solve the problem, perform the following operation as the newly-created user:

- a. Check the configuration file **conf/flink-conf.yaml** on the client.
- b. Modify the parameter **high-availability.zookeeper.path.root** to the corresponding ZooKeeper directory, for example, /flink2.
- c. Submit tasks again.

11.5.5.8 Why Cannot I Access the Apache Flink Dashboard?

Question

Why cannot I access the Apache Flink Dashboard through the URL: http://IP address of JobManager:port of JobManager.

Answer

The IP address of the computer you used has not been added to the whitelist of Apache Flink Dashboard. To solve this problem, modify the **conf/flink-conf.yaml** configuration file as follows:

1. Check whether the value of **jobmanager.web.ssl.enabled** is **false**. If not, set it to **false**.
2. Check whether the IP address of the computer you used has been added to the values of **jobmanager.web.access-control-allow-origin** and

jobmanager.web.allow-access-address. If the IP address has not been added, add it to the two parameters:
jobmanager.web.access-control-allow-origin:*IP address of the computer where the browser is installed*
jobmanager.web.allow-access-address:*IP address of the computer where the browser is installed*

11.5.5.9 How Do I View the Debugging Information Printed Using System.out.println or Export the Debugging Information to a Specified File?

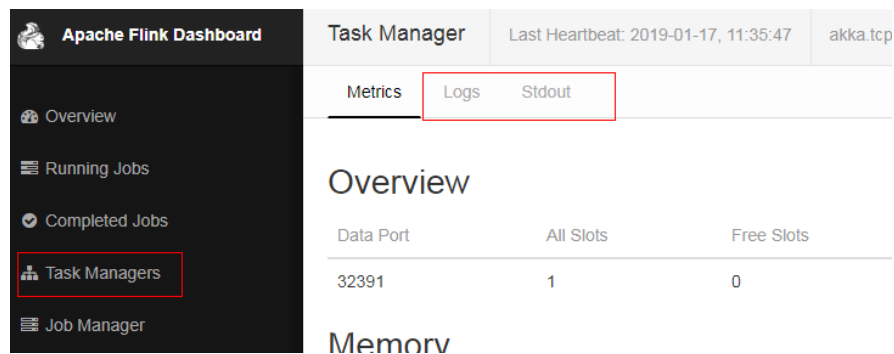
Question

System.out.println is added to the Flink service codes for printing debugging information. How do I view the debugging log? How do I export service logs to a specified file to differentiate service logs from run logs?

Answer

All run logs of Flink are printed to the local directory of Yarn. By default, all logs are exported to **taskmanager.log** in the local directory of Yarn container. All logs generated by invoking System.out will be exported to the **taskmanager.out** file. You can perform as follows to view the logs:

1. Log in to the native Flink web page.
2. Choose **Task Managers > Logs** or **Task Managers > Stdout** on the left to view log information.



Configure the function of printing service logs and Task Manager run logs separately:

NOTE

If service logs and Task Manager run logs are separately printed, service logs are not exported to the **taskmanager.log** file and cannot be viewed on the web page.

1. Modify the configuration file **logback.xml** in the **conf** directory of the client. Add the following log configuration information to the file. Modify the information in bold according to the actual situation.

```
<appender name="TEST" class="ch.qos.logback.core.rolling.RollingFileAppender">  
  <file>/path/test.log</file>  
  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">  
    <fileNamePattern>/path/test.log.%i</fileNamePattern>  
    <minIndex>1</minIndex>  
    <maxIndex>20</maxIndex>  
  </rollingPolicy>  
  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">  
    <maxFileSize>20MB</maxFileSize>  
  </triggeringPolicy>
```

```
<encoder>
  <pattern>%d{"yyyy-MM-dd HH:mm:ss,SSS"} | %m %n</pattern>
</encoder>
</appender>

<logger name="com.huawei.bigdata.flink.examples" additivity="false">
  <level value="INFO"/>
  <appender-ref ref="TEST"/>
</logger>
```

2. Run `yarn-session.sh` to submit the task.

NOTE

If the configuration file `logback.xml` contains `<file>/path/test.log</file>`, ensure that the user (configured `flink-conf.yaml`) used for running the task has the write and read permissions on the directory.

11.5.5.10 Incorrect GLIBC Version

Question

When **State Backend** is set to **RocksDB** for a Flink task, the following error message is displayed:

```
Caused by: java.lang.UnsatisfiedLinkError: /srv/BigData/hadoop/data1/nm/usercache/***/appcache/application_***/rocksdb-lib-***/librocksdbjni-linux64.so: /lib64/libpthread.so.0: version `GLIBC_2.12` not found (required by /srv/BigData/hadoop/***/librocksdbjni-linux64.so)
at java.lang.ClassLoader$NativeLibrary.load(Native Method)
at java.lang.ClassLoader.loadLibrary0(ClassLoader.java:1965)
at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1890)
at java.lang.Runtime.load0(Runtime.java:795)
at java.lang.System.load(System.java:1062)
at org.rocksdb.NativeLibraryLoader.loadLibraryFromJar(NativeLibraryLoader.java:78)
at org.rocksdb.NativeLibraryLoader.loadLibrary(NativeLibraryLoader.java:56)
at
org.apache.flink.contrib.streaming.state.RocksDBStateBackend.ensureRocksDBIsLoaded(RocksDBStateBackend.java:734)
... 11 more
```

Possible Causes

The version of the system where the task runs and the version of the system where the compilation environment locates are different, resulting in the incompatibility of GLIBC versions.

Troubleshooting Method

Run the `strings /lib64/libpthread.so.0 | grep GLIBC` command to check whether the GLIBC version is earlier than 2.12.

Procedure

If the version of the GLIBC is too early, use the file of later version (2.12) to replace `libpthread-*.so`. (This is a link file. You need to replace only the file that is linked to this link file.)

References

None

12 HBase Development Guide (Security Mode)

12.1 Overview

12.1.1 Application Development Overview

HBase Introduction

HBase is a column-oriented scalable distributed storage system featuring high reliability and high performance. HBase is designed to break through the limitation when relational databases are used to process massive data.

HBase applies to the following application scenarios:

- Massive data processing (higher than the TB or PB level).
- Scenarios that require high throughput.
- Scenarios that require efficient random read of massive data.
- Scenarios that require good scalability.
- Structured and unstructured data is concurrently processed.
- The Atomicity, Consistency, Isolation, Durability (ACID) feature supported by traditional relational databases is not required.
- HBase tables provide the following features:
 - Large: One table contains a hundred million rows and one million columns.
 - Column-oriented: Storage and rights control is implemented based on columns (families), and columns (families) are independently retrieved.
 - Sparse: Null columns do not occupy storage space, so a table is sparse.

Interface Type Introduction

The Java language is recommended for HBase application development because HBase is developed based on Java and Java is a concise, universal, and easy-to-understand language.

HBase adopts the same interfaces as those of Apache HBase.

Table 12-1 describes the functions that HBase can provide by invoking interfaces.

Table 12-1 Functions provided by HBase interfaces

Function	Description
Data CRUD function	Data creating, retrieving, updating, and deleting
Advanced feature	Filter, secondary index, and coprocessor
Management function	Table management and cluster management

12.1.2 Common Concepts

- Filter**
 Filters provide powerful features to help users improve the table data processing efficiency of HBase. Users can use the filters predefined in HBase and customized filters.
- Coprocessor**
 Coprocessors enable users to perform region-level operations and provide functions similar to those of triggers in relational database management systems (RDBMSs).
- keytab file**
 The keytab file is a key file that stores user information. In security mode, applications use the key file for API authentication on HBase.
- Client**
 Users can access the server from the client through the Java API, HBase Shell or WebUI to read and write HBase tables. The HBase client in this document indicates the HBase client installation package, see [External Interfaces](#).

12.1.3 Development Process

This document describes HBase application development based on the Java API.

For information about each phase in the development process, see [Figure 12-1](#) and [Table 12-2](#).

Figure 12-1 HBase application development process

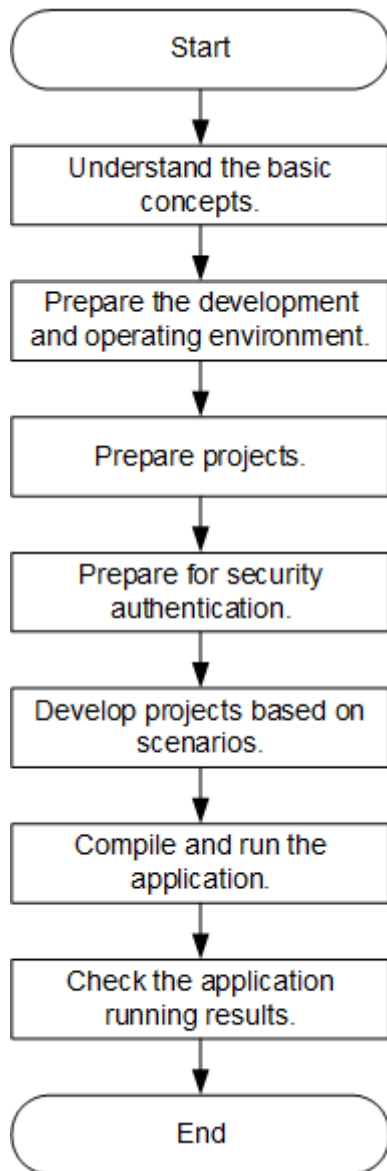


Table 12-2 HBase application development process description

Phase	Description	Reference Document
Understand the basic concepts.	Before developing an application, learn basic concepts of HBase and understand the scenario requirements and design tables.	Common Concepts

Phase	Description	Reference Document
Prepare the development and operating environment.	The Java language is recommended for the development of HBase applications. The IntelliJ IDEA tool can be used. The HBase running environment is the HBase client. Install and configure the client according to the guide.	Preparing for Development and Operating Environment
Prepare projects.	HBase provides example projects for different scenarios. You can import an example project to learn the application.	Configuring and Importing Sample Projects
Prepare for security authentication.	If you use a security cluster, you need to perform security authentication.	Preparing for Security Authentication
Develop projects based on scenarios.	An example project based on the Java language is provided, including creating a table, writing data into the table, and deleting the table.	Developing an Application
Compile and run the application.	Compile the developed application and submit it for running.	Application Commissioning
View application running results.	Application running results are written into a specified path. You can also view the application running status on the UI.	Application Commissioning

12.2 Environment Preparation

12.2.1 Preparing for Development and Operating Environment

Preparing the Development Environment

[Table 12-3](#) describes the environment required for secondary development.

Table 12-3 Development environment

Item	Description
OS	<ul style="list-style-type: none"> Development environment: Windows OS. Windows 7 or later is supported. Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> Oracle JDK versions: 1.8 IBM JDK versions: 1.8.5.11 For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> OpenJDK: 1.8.0_272 <p>NOTE The server supports only TLS V1.2 or later to ensure security. By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls.</p>
IntelliJ IDEA installation and configuration	<p>Tool used for developing HBase applications. The version must be 2019.1 or other compatible version.</p> <p>NOTE</p> <ul style="list-style-type: none"> If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK. If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK. If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK. Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.
JUnit plug-in installation	Basic configuration for the development environment.
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.

Item	Description
User development preparation	See Preparing a Developer Account for configuration.
7-zip	Used to decompress .zip and .rar packages. The 7-Zip 16.04 is supported.

Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
 - a. [Log in to the FusionInsight Manager portal](#) and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Configuration Files Only**. Select the platform type based on the type of the node where the client is to be installed (select **x86_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.
 For example, if the client configuration file package is **FusionInsight_Cluster_1_Services_Client.tar**, decompress it to obtain **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar** file. Then, decompress **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar** file to the **D:\FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles** directory on the local PC.
 - b. Go to the client configuration file decompression path **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles\HBase\config**, obtain the HBase-related configuration file. The configuration file will be imported to the configuration file directory (usually the **conf** folder) of the HBase sample project.
 The keytab authentication file obtained during the [Preparing a Developer Account](#) is also stored in this directory.
 - c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, C: \WINDOWS\system32\drivers\etc\hosts.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client on the node. For example, the client installation directory is **/opt/client**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. [Log in to the FusionInsight Manager portal](#). Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight_Cluster_1_Services_ClientConfig\HBase\config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/client/conf**.

For example, if the client software package is **FusionInsight_Cluster_1_Services_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
```

```
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
```

```
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
```

```
cd FusionInsight_Cluster_1_Services_ClientConfig
```

```
scp HBase/config/* root@IP address of the client node:/opt/client/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

12.2.2 Configuring and Importing Sample Projects

Background

Obtain the HBase development example project. Import the project to IntelliJ IDEA for learning.

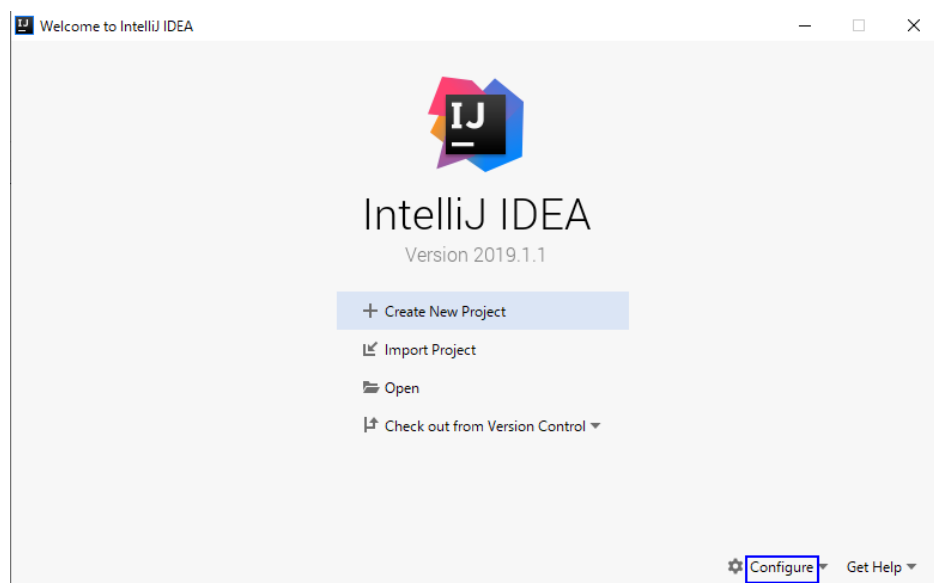
Prerequisites

Ensure that the difference between the local PC time and the MRS cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the MRS cluster time in the bottom-right corner on FusionInsight Manager.

Procedure

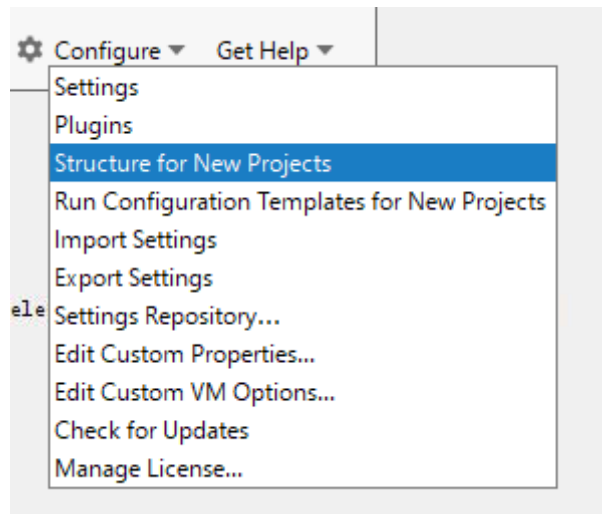
- Step 1** Obtain the sample project folder **hbase-example** in the **src/hbase-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the keytab authentication files **user.keytab** and **krb5.conf** files obtained in [Preparing a Developer Account](#) section and the cluster configuration file obtained in section [Preparing an Operating Environment](#) to the **hbase-example\src\main\resources\conf** directory of the sample project. For details about how to place configuration files and precautions for executing the sample code for other sample projects, see the **README.md** file of the corresponding sample project.
- Step 3** After the IntelliJ IDEA and JDK are installed, configure the JDK in IntelliJ IDEA.
 1. Start the IntelliJ IDEA and click **Configure**.

Figure 12-2 Quick Start



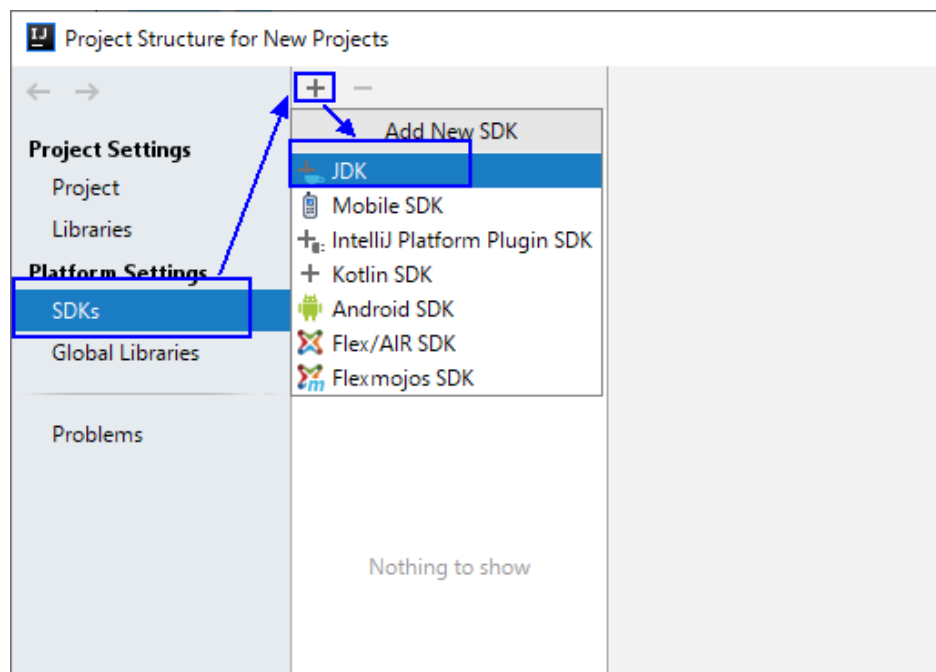
2. Click **Structure for New Projects** from the drop-down list.

Figure 12-3 Configure



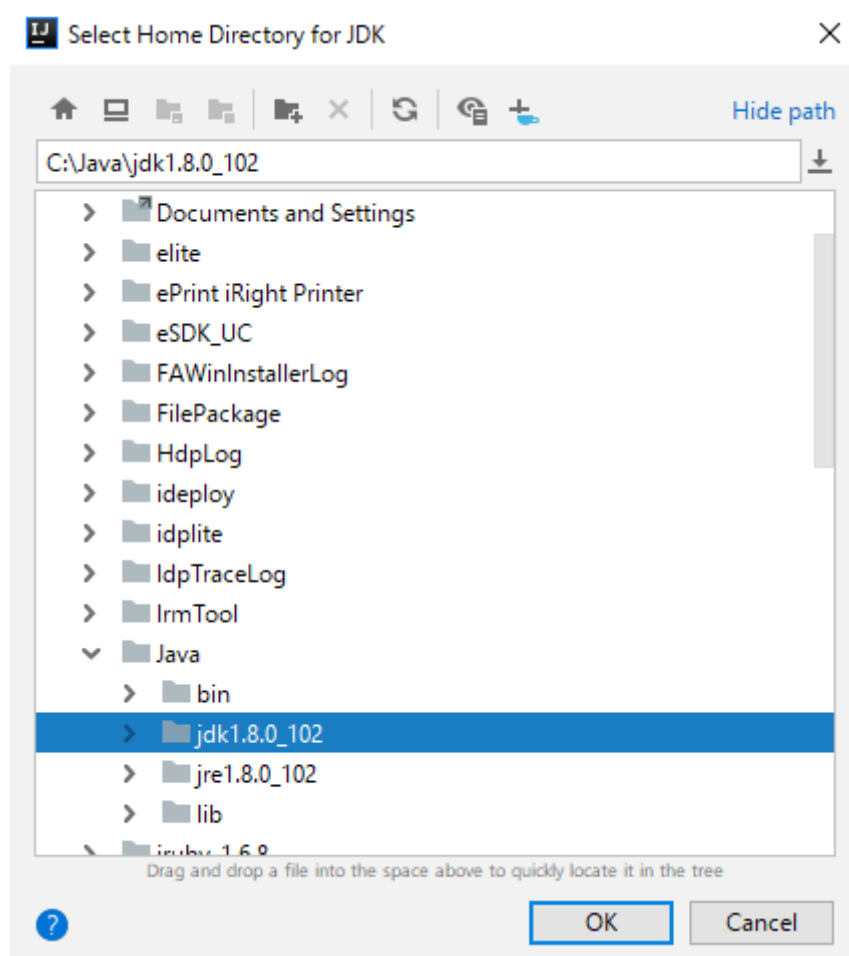
3. On the displayed **Project Structure for New Projects** page, select **SDKs** and click the plus sign (+) to add the **JDK**.

Figure 12-4 Project Structure for New Projects



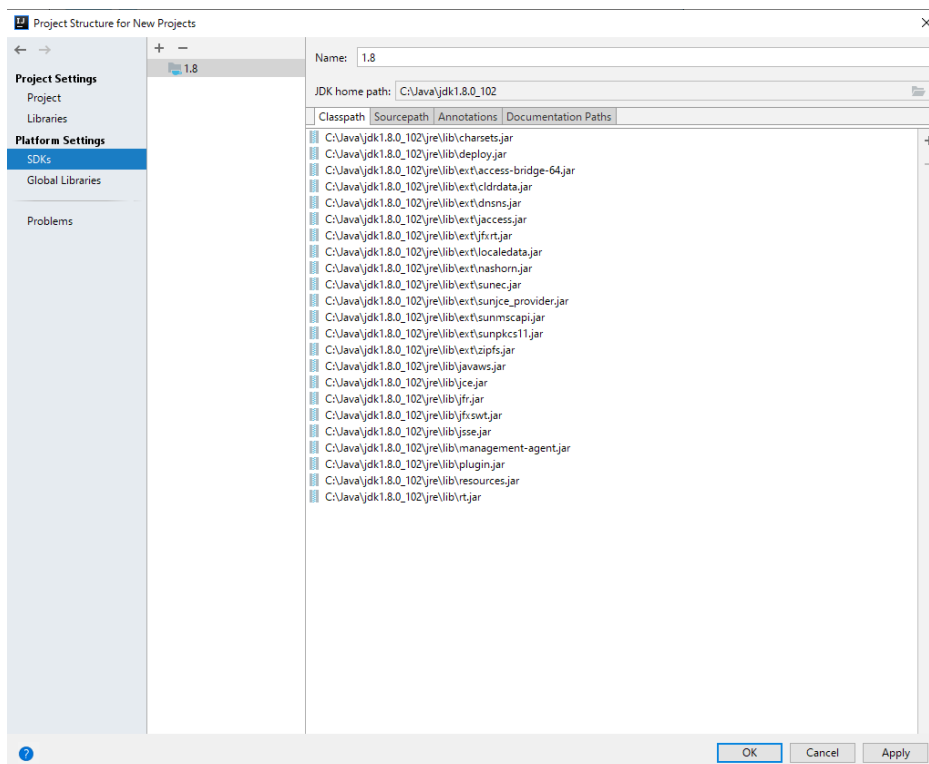
4. In the **Select Home Directory for JDK** window that is displayed, select the **JDK** directory, and click **OK**.

Figure 12-5 Select Home Directory for JDK



5. After selecting the JDK, click **OK** to complete the configuration.

Figure 12-6 Complete JDK configuration



NOTE

The operations vary by the IDEA version. The operation procedure varies according to the actual version.

Step 4 Import the example project to the IntelliJ IDEA development environment.

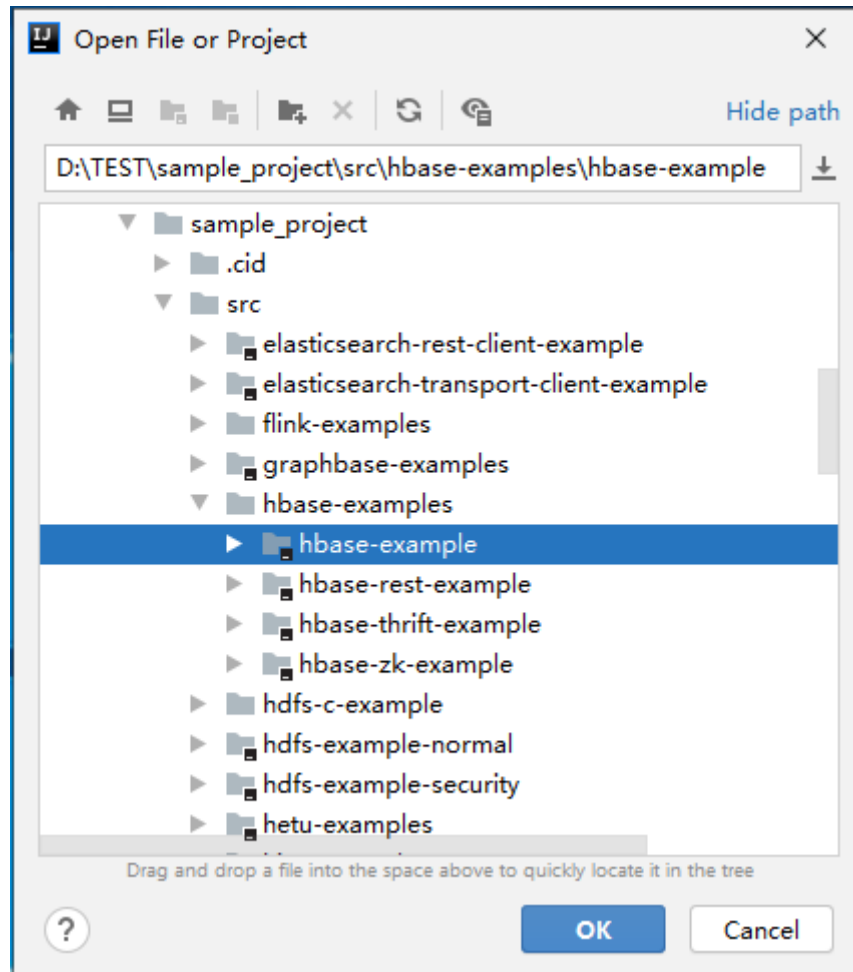
1. Start the IntelliJ IDEA, and click **Import Project** on the Open or Import. For the IDEA tool that has been used, you can choose **File > Import project...** on the main interface to import the sample project.

Figure 12-7 Open or Import (Quick Start page)



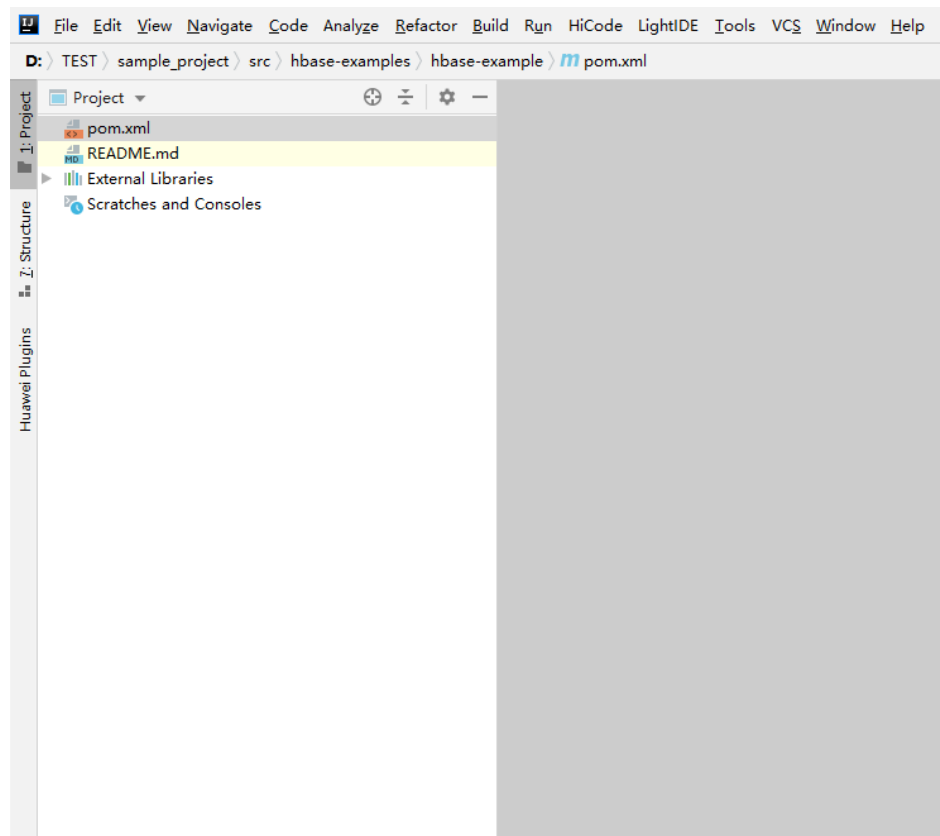
2. Select the example project folder **hbase-example**, and click **OK**.

Figure 12-8 Select File or Directory to Import



3. After the import, the imported sample project is displayed on the IDEA home page.

Figure 12-9 Successfully importing the sample project



4. Right-click **pom.xml** and choose **Add as Maven Project** from the shortcut menu to add the project as a Maven Project. If the **pom.xml** icon is as shown in [Figure 12-10](#), go to the next step.

Figure 12-10 Sample project imported as a maven project

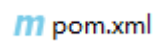
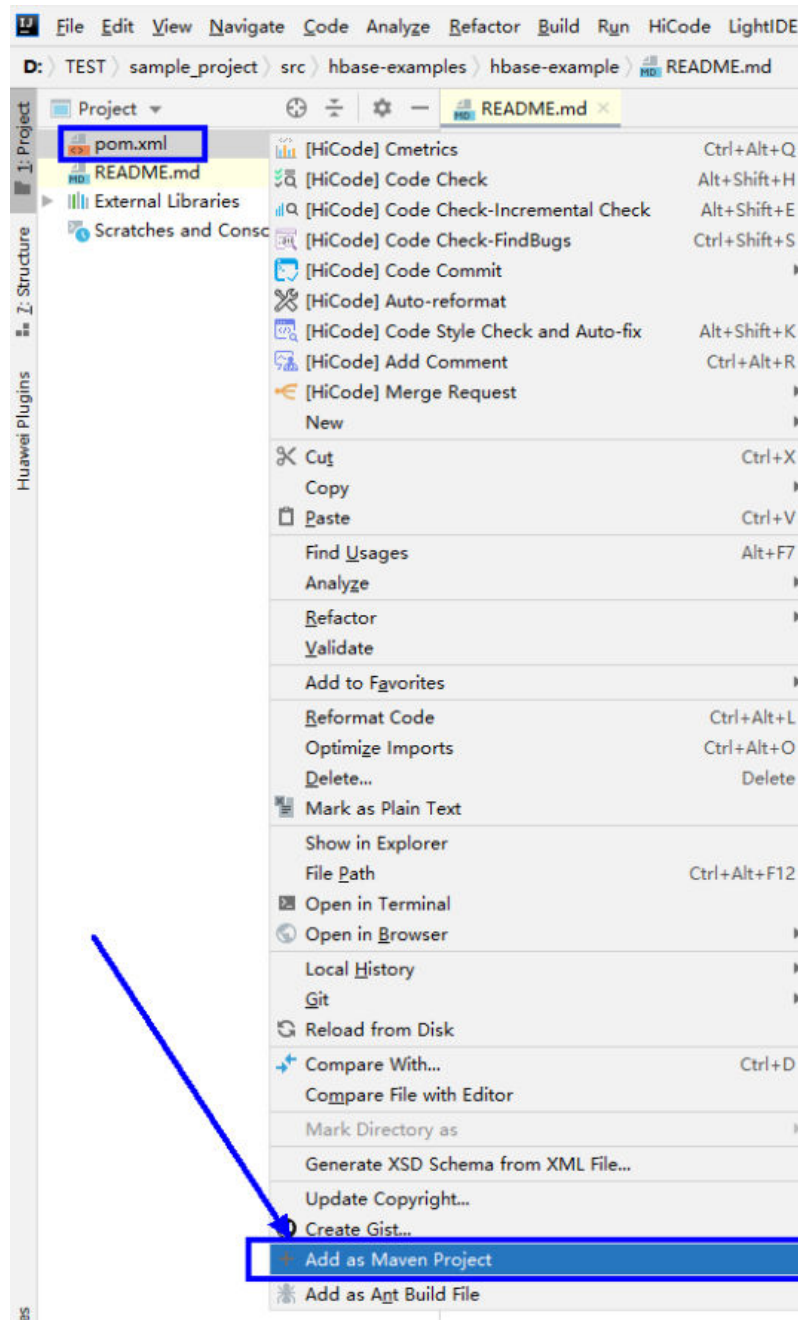
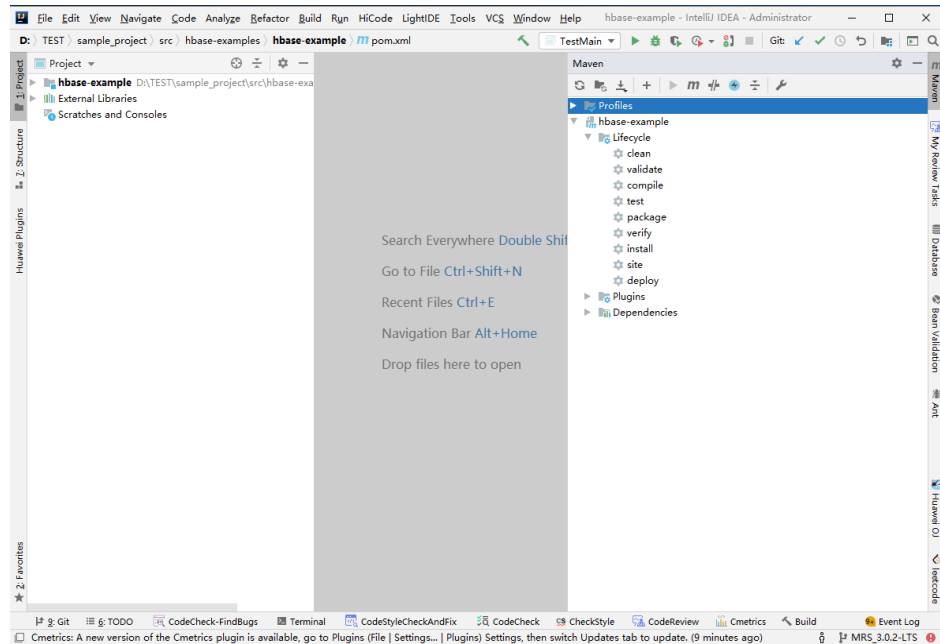


Figure 12-11 Add as Maven Project



In this case, the IDEA can identify the project as a Maven project.

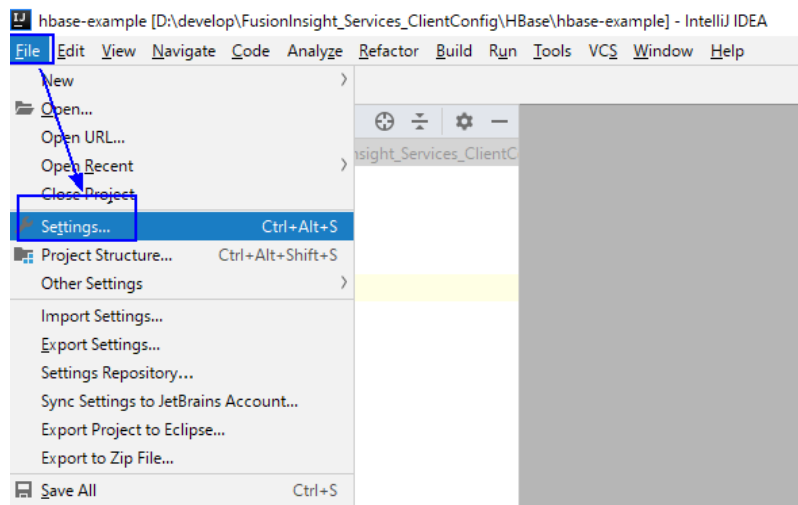
Figure 12-12 Sample project displayed in IDEA as a Maven project



Step 5 Set the Maven version used by the project.

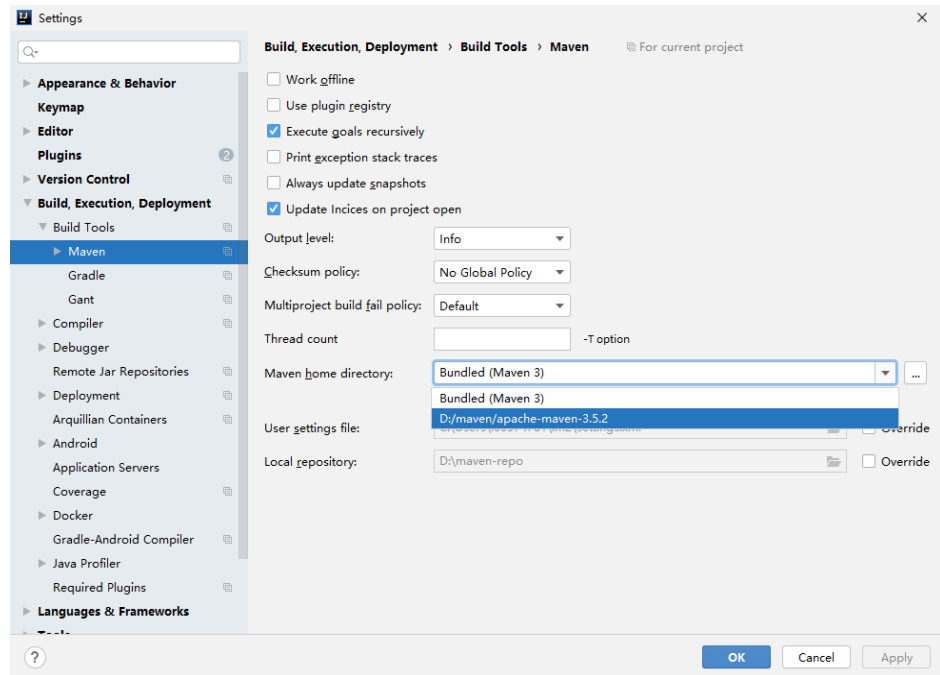
1. Choose **File > Settings...** from the main menu of IntelliJ IDEA.

Figure 12-13 Settings



2. Choose **Build, Execution, Deployment > Maven** and set **Maven home directory** to the Maven version installed on the local host.
Set **User settings file** and **Local repository** parameters based on the site requirements, and choose **Apply > OK**.

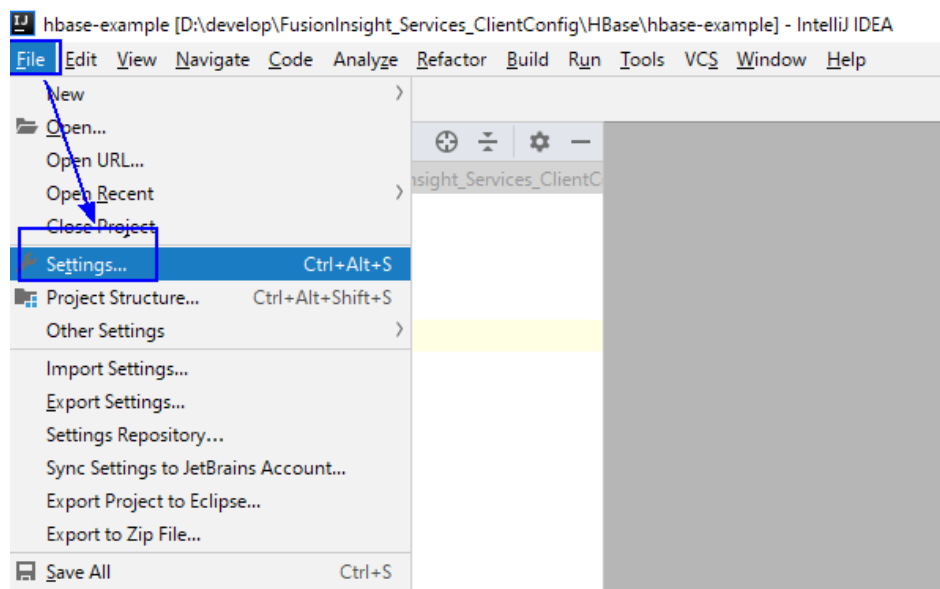
Figure 12-14 Selecting the local Maven installation directory



Step 6 Set the IntelliJ IDEA text file coding format to prevent garbled characters.

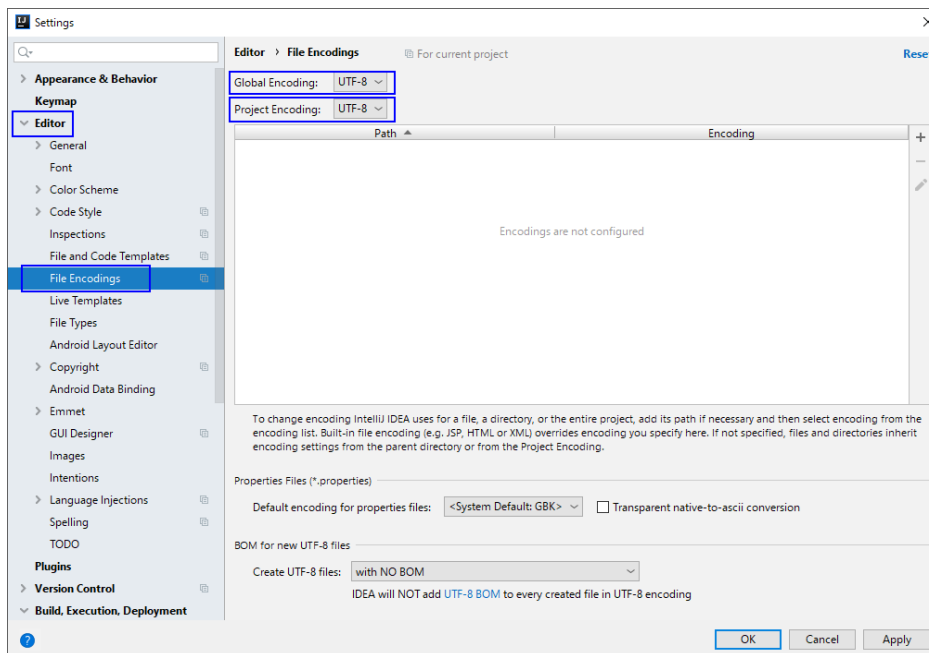
1. On the IntelliJ IDEA menu bar, choose **File >Settings...**

Figure 12-15 Settings



2. In the navigation tree of the **Settings** page, choose **Editor > File Encodings**, and select **UTF-8** for **Global Encoding** and **Project Encoding**.

Figure 12-16 File Encodings



3. Click **Apply** and **OK** to complete the encoding configuration.

----End

12.2.3 Preparing for Security Authentication

12.2.3.1 Preparing Authentication Mechanism Code

Scenario

In a security cluster environment, the components must be mutually authenticated before communicating with each other to ensure communication security. ZooKeeper and Kerberos security authentications are required for HBase application development. The **jaas.conf** file is used for ZooKeeper authentication, and the **keytab** and **krb5.conf** files are used for Kerberos security authentication. For details, see the **README.md** file of the sample code.

NOTE

- Obtain the **jaas.conf** file from the **src/hbase-examples/hbase-zk-example/src/main/resources/** directory. For details, see [Obtaining Sample Projects](#).
- For details about how to obtain the **keytab** and **krb5.conf** files, see [Step 7 in Preparing a Developer Account](#).

The code authentication mode is used for security authentication. Oracle Java and IBM Java are supported.

The following code snippet belongs to the **TestMain** class of the **com.huawei.bigdata.hbase.examples** packet.

- Code authentication
- ```
try {
 init();
```

```
login();
}
catch (IOException e) {
 LOG.error("Failed to login because ", e);
 return;
}
```

- Initializing configuration

```
private static void init() throws IOException {
 // Default load from conf directory
 conf = HBaseConfiguration.create();
 //In Windows environment
 String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() + File.separator;
[1]
 //In Linux environment
 //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
 conf.addResource(new Path(userdir + "core-site.xml"), false);
 conf.addResource(new Path(userdir + "hdfs-site.xml"), false);
 conf.addResource(new Path(userdir + "hbase-site.xml"), false);
}
```

[1] **userdir** obtains the **conf** directory in the resource path after compilation. Save the **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml** configuration files required for initialization and the user credential file used for security authentication to the **src/main/resources** directory.

- Security login

Set **userName** to the actual username based on the actual situation, for example, **developuser**.

On Windows and Linux, use the corresponding path obtaining mode.

```
private static void login() throws IOException {
 if (User.isHBaseSecurityEnabled(conf)) {
 userName = "hbaseuser1";

 //In Windows environment
 String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() +
File.separator;
 //In Linux environment
 //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;

 /*
 * if need to connect zk, please provide jaas info about zk. of course,
 * you can do it as below:
 * System.setProperty("java.security.auth.login.config", confDirPath +
 * "jaas.conf"); but the demo can help you more : Note: if this process
 * will connect more than one zk cluster, the demo may be not proper. you
 * can contact us for more help
 */
 LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, userName,
userKeytabFile);
 LoginUtil.login(userName, userKeytabFile, krb5File, conf);
 }
}
```

### 12.2.3.2 Multi-Instance Authentication in Mutual Trust Scenarios

#### Description

When multiple clusters in different security modes need to access each other's resources, the administrator can set up a mutual trust system so that users of external systems can use the system. The usage range of users in each system is called a **domain**. Each Manager system must have a unique domain name. Cross-Manager access means users to be used across domains. For details about how to

configure mutual trust between clusters, see [Managing Mutual Trust Relationships Between Managers](#).

As a user that meets the cross-domain access requirements, you can use the keytab and principal files for Kerberos security authentication obtained from one Manager system and the client configuration files of multiple Manager systems to access and invoke the HBase service of multiple clusters after one authentication login in the multi-cluster mutual trust scenario.

The following code snippets belong to the **TestMultipleLogin** class in the **com.huawei.bigdata.hbase.examples** package of the **hbase-example** sample project.

- Code authentication

```
List<String> confDirectorys = new ArrayList<>();
List<Configuration> confs = new LinkedList<>();
try {
 // conf directory
 confDirectorys.add("hadoopDomain");[1]
 confDirectorys.add("hadoop1Domain");[2]

 for (String confDir : confDirectorys) {
 confs.add(init(confDir));[3]
 }

 login(confs.get(0), confDirectorys.get(0));[4]
} catch (IOException e) {
 LOG.error("Failed to login because ", e);
 return;
}
```

[1] **hadoopDomain** indicates the name of the directory for storing user credentials and the configuration file of a cluster. The relative path of the directory is **hbase-example/src/main/resources/hadoopDomain**, which can be changed as required.

[2] **hadoop1Domain** is the name of the directory for storing the configuration file of the other cluster. The relative path of the directory is **hbase-example/src/main/resources/hadoop1Domain**, which can be changed as required.

[3] Initialize the conf objects in sequence.

[4] Perform login authentication.

- Initialization configuration

```
private static Configuration init(String confDirectoryName) throws IOException {
 // Default load from conf directory
 Configuration conf = HBaseConfiguration.create();
 //In Windows environment
 String userdir = TestMain.class.getClassLoader().getResource(confDirectoryName).getPath() +
 File.separator;
 //In Linux environment
 //String userdir = System.getProperty("user.dir") + File.separator + confDirectoryName +
 File.separator;
 conf.addResource(new Path(userdir + "core-site.xml"), false);
 conf.addResource(new Path(userdir + "hdfs-site.xml"), false);
 conf.addResource(new Path(userdir + "hbase-site.xml"), false);
 return conf;
}
```

- Security login

Set **userName** to the actual username based on the actual situation, for example, **developuser**.

```
private static void login(Configuration conf, String confDir) throwsIOException {
```

```
if (User.isHBaseSecurityEnabled(conf)) {
 userName = " developuser ";

 //In Windows environment
 String userdir = TestMain.class.getClassLoader().getResource(confDir).getPath() +
File.separator;
 //In Linux environment
 //String userdir = System.getProperty("user.dir") + File.separator + confDir + File.separator;

 userKeytabFile = userdir + "user.keytab";
 krb5File = userdir + "krb5.conf";

 /*
 * if need to connect zk, please provide jaas info about zk. of course,
 * you can do it as below:
 * System.setProperty("java.security.auth.login.config",confDirPath +
 * "jaas.conf"); but the demo can help you more : Note: if this process
 * will connect more than one zk cluster, the demo may be not proper. you
 * can contact us for more help
 */

 LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME,
userName,userKeytabFile);
 LoginUtil.login(userName, userKeytabFile, krb5File, conf);
}
}
```

### 12.2.3.3 Authentication for accessing the HBase REST Service

#### Description

When installing the HBase service, you can optionally deploy the RESTServer instance. You can access the HBase REST service to invoke HBase operations, including operations on namespaces and tables. Kerberos authentication is also required for accessing the HBase REST service.

In this scenario, initial configuration is not required. Only the **keytab** and **krb5.conf** files used for Kerberos security authentication are required. For details, see **README.md** in the sample code.

The following code snippets belong to the **HBaseRestTest** class in the **com.huawei.bigdata.hbase.examples** package of the **hbase-rest-example** sample project.

- Code authentication

Change **principal** to the actual user name, for example, **developuser**.

In Windows and Linux environments, use the corresponding path to obtain the file.

```
//In Windows environment
String userdir = HBaseRestTest.class.getClassLoader().getResource("conf").getPath() +
File.separator;[1]
//In Linux environment
//String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
String principal = "hbaseuser1";
login(principal, userKeytabFile, krb5File);
// RESTServer's hostname.
String restHostName = "10.120.16.170";[2]
String securityModeUrl = new StringBuilder("https://").append(restHostName).append(":
21309").toString();
String nonSecurityModeUrl = new StringBuilder("http://").append(restHostName).append(":
21309").toString();
HBaseRestTest test = new HBaseRestTest();
```

```
//If cluster is non-security mode,use nonSecurityModeUrl as parameter.
test.test(securityModeUrl);[3]
```

[1] **userdir** obtains the **conf** directory in the resource path after compilation. Save the **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml** configuration files required for initialization and the user credential file used for security authentication to the **src/main/resources** directory. If the **conf** directory does not exist, create it.

[2] Change the value of **restHostName** to the IP address of the node where the RestServer instance to be accessed is located, and configure the node IP address in the hosts file on the local host where the sample code is run.

[3] In security mode, access the HBase REST service in HTTPS mode and use **nonSecurityModeUrl** as the **test.test()** parameter.

- Security login

```
private static void login(String principal, String userKeytabFile, String krb5File) throws
LoginException {
 Map<String, String> options = new HashMap<>();
 options.put("useTicketCache", "false");
 options.put("useKeyTab", "true");
 options.put("keyTab", userKeytabFile);

 /**
 * Krb5 in GSS API needs to be refreshed so it does not throw the error
 * Specified version of key is not available
 */

 options.put("refreshKrb5Config", "true");
 options.put("principal", principal);
 options.put("storeKey", "true");
 options.put("doNotPrompt", "true");
 options.put("isInitiator", "true");
 options.put("debug", "true");
 System.setProperty("java.security.krb5.conf", krb5File);
 Configuration config = new Configuration() {
 @Override
 public AppConfigurationEntry[] getAppConfigurationEntry(String name) {
 return new AppConfigurationEntry[] {
 new AppConfigurationEntry("com.sun.security.auth.module.Krb5LoginModule",
 AppConfigurationEntry.LoginModuleControlFlag.REQUIRED, options)
 };
 }
 };
 subject = new Subject(false, Collections.singleton(new KerberosPrincipal(principal)),
Collections.EMPTY_SET,
Collections.EMPTY_SET);
 LoginContext loginContext = new LoginContext("Krb5Login", subject, null, config);
 loginContext.login();
}
```

### 12.2.3.4 Authentication for Accessing the ThriftServer Service

#### Scenario

HBase combines Thrift to provide HBase services for external applications. The ThriftServer instance is optional during HBase service installation. The ThriftServer system can access HBase users and has the read, write, execute, creation, and management permissions on all HBase namespaces and tables. Kerberos authentication is also required for accessing the ThriftServer service. HBase implements two sets of Thrift Server services. **hbase-thrift-example** is used to call the ThriftServer instance service.

## Procedure

- Step 1** Log in to FusionInsight Manager, choose **Cluster > Service > HBase > Configuration** and click **All Configurations**, search for and modify the parameter **hbase.thrift.security.qop** of the ThriftServer instance. The value of this parameter must be the same as that of **hbase.rpc.protection**. Save the configuration and restart the node service for the configuration to take effect.

### NOTE

The mapping between **hbase.rpc.protection** and **hbase.thrift.security.qop** is as follows:

- "privacy" - "auth-conf"
- "authentication" - "auth"
- "integrity" - "auth-int"

- Step 2** Obtain the configuration file of the ThriftServer instance in the cluster.

- Method 1: Choose **Cluster > Service > HBase > Instance**, click the ThriftServer instance to go to the details page, and obtain the configuration files **hdfs-site.xml**, **core-site.xml**, and **hbase-site.xml**.
- Method 2: Obtain the configuration files by decompressing the client file in [Preparing for Development and Operating Environment](#). Manually add the following configuration to **hbase-site.xml**. The value of **hbase.thrift.security.qop** must be the same as that in [Step 1](#).

```
<property>
<name>hbase.thrift.security.qop</name>
<value>auth</value>
</property>
<property>
<name>hbase.thrift.kerberos.principal</name>
<value>thrift/hadoop.hadoop.com@HADOOP.COM</value>
</property>
<property>
<name>hbase.thrift.keytab.file</name><value>/opt/huawei/Bigdata/FusionInsight_HD_8.1.0.1/install/
FusionInsight-HBase-2.2.3/keytabs/HBase/thrift.keytab</value>
</property>
```

----End

## Example Code

- Code authentication

The following code snippets belong to the **TestMain** class in the **com.huawei.bigdata.hbase.examples** package of the **hbase-thrift-example** sample project.

```
private static void init() throws IOException {
 // Default load from conf directory
 conf = HBaseConfiguration.create();

 String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() + File.separator;
[1]
 //In Linux environment
 //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
 conf.addResource(new Path(userdir + "core-site.xml"), false);
 conf.addResource(new Path(userdir + "hdfs-site.xml"), false);
 conf.addResource(new Path(userdir + "hbase-site.xml"), false);
}
```

[1] **userdir** obtains the **conf** directory in the resource path after compilation. The **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml** files used for

initialization configuration and the user credential file used for security authentication must be stored in the **src/main/resources/conf** directory.

- Security login

Set **userName** to the actual username based on the actual situation, for example, **developuser**.

```
private static void login() throws IOException {
 if (User.isHBaseSecurityEnabled(conf)) {
 userName = " developuser ";

 //In Windows environment
 String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() +
File.separator;
 //In Linux environment
 //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;

 userKeytabFile = userdir + "user.keytab";
 krb5File = userdir + "krb5.conf";

 /*
 * if need to connect zk, please provide jaas info about zk. of course,
 * you can do it as below:
 * System.setProperty("java.security.auth.login.config", confDirPath +
 * "jaas.conf"); but the demo can help you more : Note: if this process
 * will connect more than one zk cluster, the demo may be not proper. you
 * can contact us for more help
 */
 LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, userName,
userKeytabFile);
 LoginUtil.login(userName, userKeytabFile, krb5File, conf);
 }
}
```

- Connecting to a ThriftServer instance

```
try {
 test = new ThriftSample();
 test.test("10.120.16.170", THRIFT_PORT, conf);[2]
} catch (TException | IOException e) {
 LOG.error("Test thrift error", e);
}
```

[2] The value of the input parameter **test.test()** is the IP address of the node where the ThriftServer instance to be accessed is located. Change the IP address to the actual one. The IP address of the node must be configured in the hosts file of the local host where the sample code is run.

**THRIFT\_PORT** is the value of **hbase.regionserver.thrift.port** configured for the ThriftServer instance.

### 12.2.3.5 Authentication for Accessing Multiple ZooKeepers

#### Scenario

To avoid ZooKeeper authentication conflicts when a client process accesses a FusionInsight ZooKeeper and a third-party ZooKeeper at the same time, sample code is provided for the HBase client to access the FusionInsight ZooKeeper and for customer applications to access the third-party ZooKeeper.

The following lists the authentication configuration files in the **src/main/resources** directory.

- zoo.cfg

```
The configuration in jaas.conf used to connect fi
zookeeper.zookeeper.sasl.clientconfig=Client_new[1]
```



```
Principal of fi zookeeper server side.
zookeeper.server.principal=zookeeper/hadoop.hadoop.com[2]
Set true if the fi cluster is security mode.
The other two parameters do not take effect if the value is false.
zookeeper.sasl.client=true[3]
```

[1] **zookeeper.sasl.clientconfig**: specifies the configuration in the **jaas.conf** file used for accessing FusionInsight ZooKeeper.

[2] **zookeeper.server.principal**: specifies the principal used by a ZooKeeper server.

[3] **zookeeper.sasl.client**: If the MRS cluster works in the security mode, set this parameter to **true**. Otherwise, set this parameter to **false**. If this parameter is set to **false**, the **zookeeper.sasl.clientconfig** and **zookeeper.server.principal** parameters do not take effect.

- **jaas.conf**

```
Client_new { [4]
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 keyTab="D:\\work\\sample_project\\src\\hbase-examples\\hbase-zk-example\\target\\classes\\conf\\
 user.keytab" [5]
 principal="hbaseuser1"
 useTicketCache=false
 storeKey=true
 debug=true;
};
Client { [6]
 org.apache.zookeeper.server.auth.DigestLoginModule required
 username="bob"
 password="xxxxxx";
};
```

[4] **Client\_new**: reads configuration specified in the **zoo.cfg** file. When the name is changed, the corresponding configuration in the **zoo.cfg** file must be modified accordingly.

[5] **keyTab**: specifies the path for storing the **user.keytab** file used by the project on the host where the sample is run. Use an absolute path to better locate the file. Use \\ in the Windows and \ in Linux.

[6] **Client**: A third-party ZooKeeper uses this configuration for access. The connection authentication configuration depends on the third-party ZooKeeper version.

## 12.3 Developing an Application

### 12.3.1 Typical Scenario Description

Based on typical scenarios, users can quickly learn and master the HBase development process and know important interface functions.

#### Scenario

Develop an application to manage information about users who use service A in an enterprise. [Table 12-4](#) provides the user information. The operation process of service A is described as follows:

- Create a user information table.

- Add diplomas and titles to the user information table.
- Query user names and addresses by user ID.
- Query information by user name.
- Query information about users whose ages range from 20 to 29.
- Collect statistics on the number and the maximum, minimum, and average ages of users.
- Deregister users, and delete user data.
- Delete the user information table after service A ends.

**Table 12-4** User information

ID	Name	Gender	Age	Address
1200500020 1	Zhang San	Male	19	Shenzhen, Guangdong
1200500020 2	Li Wanting	Female	23	Shijiazhuang, Hebei
1200500020 3	Wang Ming	Male	26	Ningbo, Zhejiang
1200500020 4	Li Gang	Male	18	Xiangyang, Hubei
1200500020 5	Zhao Enru	Female	21	Shangrao, Jiangxi
1200500020 6	Chen Long	Male	32	Zhuzhou, Hunan
1200500020 7	Zhou Wei	Female	29	Nanyang, Henan
1200500020 8	Yang Yiwen	Female	30	Kaixian, Chongqing
1200500020 9	Xu Bing	Male	26	Weinan, Shaanxi
1200500021 0	Xiao Kai	Male	25	Dalian, Liaoning

## Data Planning

Proper design of the table structure, RowKeys, and column names can give full play to the advantages of HBase. In this example, the unique ID is used as the RowKey, and columns are stored in the **info** column family.

## 12.3.2 Development Idea

### Function Decomposition

Functions are decomposed based on the preceding service scenario. [Table 12-5](#) provides the functions to be developed.

**Table 12-5** Functions to be developed in HBase

No.	Procedure	Code Implementation
1	Create a table based on <a href="#">Table 12-4</a> .	For details, see <a href="#">Creating a Table</a> .
2	Import user data.	For details, see <a href="#">Inserting Data</a> .
3	Add the <b>Education</b> column family, and add diplomas and titles to the user information table.	For details, see <a href="#">Modifying a Table</a> .
4	Query user names and addresses by user ID.	For details, see <a href="#">Reading Data Using Get</a> .
5	Query information by user name.	For details, see <a href="#">Filtering Data</a> .
6	To improve query performance, create or delete secondary indexes.	For details, see <a href="#">Creating a Secondary Index</a> and <a href="#">Secondary Index-based Query</a> .
7	Deregister users, and delete user data.	For details, see <a href="#">Deleting Data</a> .
8	Delete the user information table after service A ends.	For details, see <a href="#">Deleting a Table</a> .

### Key Design Principle

HBase is a distributed database system based on the lexicographic order of RowKeys. The RowKey design has great impact on performance, so the RowKeys must be designed based on specific services.

## 12.3.3 Example Code Description

### 12.3.3.1 Configuring Log4j Log Output

#### Function Description

This section describes how to output HBase client logs to a specified log file separately from service logs to facilitate HBase problem analyzing and locating.

If the **log4j** configuration exists in the process, copy the RFA and RFAS configurations in **hbase-example\src\main\resources\log4j.properties** to the existing **log4j** configuration.

## Sample Code

```
hbase.root.logger=INFO,console,RFA //HBase client log output configuration. console: outputs to
the console; RFA: outputs to the log files.
hbase.security.logger=DEBUG,console,RFAS //HBase client security logs output configuration. console:
outputs to the console; RFAS: outputs the log files.
hbase.log.dir=/var/log/Bigdata/hbase/client/ //Log directory. Modify based on the actual directory. Ensure
that the directory has the write permission.
hbase.log.file=hbase-client.log //Log file name
hbase.log.level=INFO //Log level. If detailed logs are required for fault locating, change it
to DEBUG. The modification takes effect after restart.
hbase.log.maxbackupindex=20 //Maximum number of log files that can be saved.
Security audit appender
hbase.security.log.file=hbase-client-audit.log //Command of the audit log file
```

### 12.3.3.2 Creating Configuration

#### Function

HBase obtain configuration items by using the login method. The configuration items include user login information and security authentication information.

#### Example Codes

The following code snippet belongs to the **init** method in the **TestMain** class of the **com.huawei.bigdata.hbase.examples** package.

```
private static void init() throws IOException {
 // Default load from conf directory
 conf = HBaseConfiguration.create();
 //In Windows environment
 String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() + File.separator;
 //In Linux environment
 //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
 conf.addResource(new Path(userdir + "core-site.xml"), false);
 conf.addResource(new Path(userdir + "hdfs-site.xml"), false);
 conf.addResource(new Path(userdir + "hbase-site.xml"), false);
}
```

### 12.3.3.3 Creating Connection

#### Function

HBase creates a Connection object using the **ConnectionFactory.createConnection(configuration)** method. The transferred parameter is the Configuration created in the last step.

Connection encapsulates the connections between underlying applications and servers and ZooKeeper. Connection is instantiated using the ConnectionFactory class. Creating Connection is a heavyweight operation. Connection is thread-safe. Therefore, multiple client threads can share one Connection.

In a typical scenario, a client program uses a Connection, and each thread obtains its own Admin or Table instance and invokes the operation interface provided by the Admin or Table object. You are not advised to cache or pool Table and Admin.

The lifecycle of Connection is maintained by invokers that frees up resources by invoking close().

## Example Code

The following code snippet exemplifies login, creating Connection, and creating a table. It belongs to the **HbaseSample** method in the **HbaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
private TableName tableName = null;
private Connection conn = null;

public HBaseSample(Configuration conf) throws IOException {
 this.tableName = TableName.valueOf("hbase_sample_table");
 this.conn = ConnectionFactory.createConnection(conf);
}
```

### NOTE

Avoid invoking login code repeatedly.

## 12.3.3.4 Creating a Table

### Function

Create a table using the **createTable** method of the **org.apache.hadoop.hbase.client.Admin** object and specify the table name and column family name. Tables can be created in two modes. (The mode of creating a table using preassigned regions is strongly recommended.)

- Quickly create a table. A newly created table contains only one region which will be split into multiple new regions as data increases.
- Create a table using preassigned regions. Preassign multiple regions before creating a table. This mode accelerates data write at the beginning of massive data write.

### NOTE

The column name and column family name of an HBase table consists of letters, digits, and underscores and cannot contain any special characters.

## Example Code

The following code snippet belongs to the **testCreateTable** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testCreateTable() {
 LOG.info("Entering testCreateTable.");
 // Specify the table descriptor.
 TableDescriptorBuilder htd = TableDescriptorBuilder.newBuilder(tableName);(1)

 // Set the column family name to info.
 ColumnFamilyDescriptorBuilder hcd =
 ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("info")); (2)

 // Set data encoding methods, HBase provides DIFF,FAST_DIFF,PREFIX

 hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);
 // Set compression methods, HBase provides two default compression
 // methods:GZ and SNAPPY
```

```

// GZ has the highest compression rate, but low compression and
// decompression efficiency, fit for cold data
// SNAPPY has low compression rate, but high compression and
// decompression efficiency, fit for hot data.
// it is advised to use SNAPPY
hcd.setCompressionType(Compression.Algorithm.SNAPPY); // Note [1]
htd.setColumnFamily(hcd.build()); // (3)
Admin admin = null;
try {
 // Instantiate an Admin object.
 admin = conn.getAdmin(); // (4)
 if (!admin.tableExists(tableName)) {
 LOG.info("Creating table...");
 admin.createTable(htd.build()); // Note [2] // (5)
 LOG.info(admin.getClusterMetrics().toString());
 LOG.info(admin.listNamespaceDescriptors().toString());
 LOG.info("Table created successfully.");
 } else {
 LOG.warn("table already exists");
 }
} catch (IOException e) {
 LOG.error("Create table failed " + e);
} finally {
 if (admin != null) {
 try {
 // Close the Admin object.
 admin.close();
 } catch (IOException e) {
 LOG.error("Failed to close admin " + e);
 }
 }
}
LOG.info("Exiting testCreateTable.");
}

```

## Explanation

1. Create a table descriptor.
2. Create a column family descriptor.
3. Add the column family descriptor to the table descriptor.
4. Obtain an Admin object. The Admin object provides functions for creating a table, creating a column family, checking whether a table exists, modifying the table structure, modifying the column family structure, and deleting a table.
5. Invoke the table creation method of Admin.

## Precautions

- 1. The compression mode of a column family can be set. The code snippets are as follows:
 

```

// Set the encoding algorithm. HBase supports the DIFF, FAST_DIFF, PREFIX encoding algorithms.
hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);

// Set the file compression mode. HBase provides the GZ and SNAPPY compression algorithms by default.
// GZ provides a high compression rate but low compression and decompression performance. GZ is suitable for cold data.
// SNAPPY provides a low compression rate but high compression and decompression performance. SNAPPY is suitable for hot data.
// It is recommended that SNAPPY be enabled by default.
hcd.setCompressionType(Compression.Algorithm.SNAPPY);

```
- 2. A table can be created by specifying the start and end RowKeys or preassigning regions using RowKey arrays. The code snippets are as follows:

```
// Create a table whose regions are preassigned.
byte[][] splits = new byte[4][];
splits[0] = Bytes.toBytes("A");
splits[1] = Bytes.toBytes("H");
splits[2] = Bytes.toBytes("O");
splits[3] = Bytes.toBytes("U");
admin.createTable(htd, splits);
```

### 12.3.3.5 Deleting a Table

#### Function

Delete a table using the **deleteTable** method of **org.apache.hadoop.hbase.client.Admin**.

#### Example Code

The following code snippet belongs to the **dropTable** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void dropTable() {
 LOG.info("Entering dropTable.");
 Admin admin = null;
 try {
 admin = conn.getAdmin();
 if (admin.tableExists(tableName)) {
 // Disable the table before deleting it.
 admin.disableTable(tableName);
 // Delete table.
 admin.deleteTable(tableName); // Note[1]
 }
 LOG.info("Drop table successfully.");
 } catch (IOException e) {
 LOG.error("Drop table failed " ,e);
 } finally {
 if (admin != null) {
 try {
 // Close the Admin object.
 admin.close();
 } catch (IOException e) {
 LOG.error("Close admin failed " ,e);
 }
 }
 }
 LOG.info("Exiting dropTable.");
}
```

#### Precautions

A table can be deleted only when the table is disabled. Therefore, **deleteTable** is used together with **disableTable**, **enableTable**, **tableExists**, **isTableEnabled**, and **isTableDisabled**.

### 12.3.3.6 Modifying a Table

#### Function

Modify table information using the **modifyTable** method of **org.apache.hadoop.hbase.client.Admin**.

## Example Code

The following code snippet belongs to the **testModifyTable** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testModifyTable() {
 LOG.info("Entering testModifyTable.");

 // Specify the column family name.
 byte[] familyName = Bytes.toBytes("education");
 Admin admin = null;
 try {
 // Instantiate an Admin object.
 admin = conn.getAdmin();
 // Obtain the table descriptor.
 TableDescriptor htd = admin.getTableDescriptor(tableName);

 // Check whether the column family is specified before modification.
 if (!htd.hasColumnFamily(familyName)) {
 // Create the column descriptor.
 TableDescriptor tableBuilder = TableDescriptorBuilder.newBuilder(htd)
 .setColumnFamily(ColumnFamilyDescriptorBuilder.newBuilder(familyName).build()).build();

 // Disable the table to get the table offline before modifying
 // the table.
 admin.disableTable(tableName); // Note[1]
 // Submit a modifyTable request.
 admin.modifyTable(tableBuilder);
 // Enable the table to get the table online after modifying the
 // table.
 admin.enableTable(tableName);
 }
 LOG.info("Modify table successfully.");
 } catch (IOException e) {
 LOG.error("Modify table failed " ,e);
 } finally {
 if (admin != null) {
 try {
 // Close the Admin object.
 admin.close();
 } catch (IOException e) {
 LOG.error("Close admin failed " ,e);
 }
 }
 }
 LOG.info("Exiting testModifyTable.");
}
```

## Precautions

**modifyTable** takes effect only when a table is disabled.

### 12.3.3.7 Inserting Data

## Function

HBase is a column-oriented database. A row of data may have multiple column families, and a column family may contain multiple columns. When writing data, you must specify the columns (including the column family names and column names) to which data is written. In HBase, data (a row of data or data sets) is inserted using the **put** method of HTable.



## Example Code

The following code snippet belongs to the `testPut` method in the `HBaseSample` class of the `com.huawei.bigdata.hbase.examples` package.

```
public void testPut() {
 LOG.info("Entering testPut.");

 // Specify the column family name.
 byte[] familyName = Bytes.toBytes("info");
 // Specify the column name.
 byte[][] qualifiers = {
 Bytes.toBytes("name"), Bytes.toBytes("gender"), Bytes.toBytes("age"), Bytes.toBytes("address")
 };

 Table table = null;
 try {
 // Instantiate an HTable object.
 table = conn.getTable(tableName);
 List<Put> puts = new ArrayList<Put>();

 // Instantiate a Put object.
 Put put = putData(familyName, qualifiers,
 Arrays.asList("012005000201", "Zhang San", "Male", "19", "Shenzhen, Guangdong"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000202", "Li Wanting", "Female", "23", "Shijiazhuang, Hebei"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000203", "Wang Ming", "Male", "26", "Ningbo, Zhejiang"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000204", "Li Gang", "Male", "18", "Xiangyang, Hubei"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000205", "Zhao Enru", "Female", "21", "Shangrao, Jiangxi"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000206", "Chen Long", "Male", "32", "Zhuzhou, Hunan"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000207", "Zhou Wei", "Female", "29", "Nanyang, Henan"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000208", "Yang Yiwen", "Female", "30", "Kaixian, Chongqing"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000209", "Xu Bing", "Male", "26", "Weinan, Shaanxi"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000210", "Xiao Kai", "Male", "25", "Dalian, Liaoning"));
 puts.add(put);

 // Submit a put request.
 table.put(puts);

 LOG.info("Put successfully.");
 } catch (IOException e) {
 LOG.error("Put failed ", e);
 } finally {

```

```
 if (table != null) {
 try {
 // Close the HTable object.
 table.close();
 } catch (IOException e) {
 LOG.error("Close table failed ", e);
 }
 }
 }
 LOG.info("Exiting testPut.");
}
```

## Precautions

Multiple threads are not allowed to use the same HTable instance at the same time. HTable is a non-thread safe class. If an HTable instance is used by multiple threads at the same time, concurrency problems will occur.

### 12.3.3.8 Deleting Data

#### Function

Delete data (a row of data or data sets) using the **delete** method of a Table instance.

#### Example Code

The following code snippet belongs to the **testDelete** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testDelete() {
 LOG.info("Entering testDelete.");

 byte[] rowKey = Bytes.toBytes("012005000201");

 Table table = null;
 try {
 // Instantiate an HTable object.
 table = conn.getTable(tableName);

 // Instantiate a Delete object.
 Delete delete = new Delete(rowKey);

 // Submit a delete request.
 table.delete(delete);

 LOG.info("Delete table successfully.");
 } catch (IOException e) {
 LOG.error("Delete table failed ", e);
 } finally {
 if (table != null) {
 try {
 // Close the HTable object.
 table.close();
 } catch (IOException e) {
 LOG.error("Close table failed ", e);
 }
 }
 }
 LOG.info("Exiting testDelete.");
}
```

 NOTE

If secondary index is created in the family of the column where the deleted cell is, the index data is synchronously deleted.

### 12.3.3.9 Reading Data Using Get

#### Function

Before reading data from a table, instantiate the Table instance of the table, and then create a Get object. You can also set parameters for the Get object, such as the column family name and column name. Query results are stored in the Result object that stores multiple Cells.

#### Example Code

The following code snippet belongs to the `testGet` method in the `HBaseSample` class of the `com.huawei.bigdata.hbase.examples` package.

```
public void testGet() {
 LOG.info("Entering testGet.");
 // Specify the column family name.
 byte[] familyName = Bytes.toBytes("info");
 // Specify the column name.
 byte[][] qualifier = { Bytes.toBytes("name"), Bytes.toBytes("address") };
 // Specify RowKey.
 byte[] rowKey = Bytes.toBytes("012005000201");
 Table table = null;
 try {
 // Create the Table instance.
 table = conn.getTable(tableName);
 // Instantiate a Get object.
 Get get = new Get(rowKey);
 // Set the column family name and column name.
 get.addColumn(familyName, qualifier[0]);
 get.addColumn(familyName, qualifier[1]);
 // Submit a get request.
 Result result = table.get(get);
 // Print query results.
 for (Cell cell : result.rawCells()) {
 LOG.info("{}:{}", Bytes.toString(CellUtil.cloneRow(cell)),
 Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
 Bytes.toString(CellUtil.cloneValue(cell)));
 }
 LOG.info("Get data successfully.");
 } catch (IOException e) {
 LOG.error("Get data failed " ,e);
 } finally {
 if (table != null) {
 try {
 // Close the HTable object.
 table.close();
 } catch (IOException e) {
 LOG.error("Close table failed " ,e);
 }
 }
 }
 LOG.info("Exiting testGet.");
}
```

### 12.3.3.10 Reading Data Using Scan

#### Function

Before reading data from a table, instantiate the Table instance of the table, create a Scan object, and set parameters for the Scan object based on search criteria. To improve query efficiency, you are advised to specify **StartRow** and **StopRow**. Query results are stored in the ResultScanner object where each row of data is stored as a Result object that stores multiple Cells.

#### Example Code

The following code snippet belongs to the **testScanData** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testScanData() {
 LOG.info("Entering testScanData.");
 Table table = null;
 // Instantiate a ResultScanner object.
 ResultScanner rScanner = null;
 try {
 // Create the Configuration instance.
 table = conn.getTable(tableName);
 // Instantiate a Get object.
 Scan scan = new Scan();
 scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));
 // Set the cache size.
 scan.setCaching(1000);

 // Submit a scan request.
 rScanner = table.getScanner(scan);
 // Print query results.
 for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
 for (Cell cell : r.rawCells()) {
 LOG.info("{}:{}", Bytes.toString(CellUtil.cloneRow(cell)),
 Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
 Bytes.toString(CellUtil.cloneValue(cell)));
 }
 }
 LOG.info("Scan data successfully.");
 } catch (IOException e) {
 LOG.error("Scan data failed ", e);
 } finally {
 if (rScanner != null) {
 // Close the scanner object.
 rScanner.close();
 }
 if (table != null) {
 try {
 // Close the HTable object.
 table.close();
 } catch (IOException e) {
 LOG.error("Close table failed ", e);
 }
 }
 }
 LOG.info("Exiting testScanData.");
}
```

#### Precautions

1. You are advised to specify **StartRow** and **StopRow** to ensure good performance with a specified Scan scope.

## 2. You can set **Batch** and **Caching**.

### – **Batch**

Indicates the maximum number of records returned each time when the **next** interface is invoked using Scan. This parameter is related to the number of columns read each time.

### – **Caching**

Indicates the maximum number of **next** records returned for a remote procedure call (RPC) request. This parameter is related to the number of rows read by an RPC.

## 12.3.3.11 Filtering Data

### Function

HBase Filter is used to filter data during Scan and Get. You can specify the filter criteria, such as filtering by RowKey, column name, or column value.

### Example Code

The following code snippet belongs to the **testSingleColumnValueFilter** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testSingleColumnValueFilter() {
 LOG.info("Entering testSingleColumnValueFilter.");
 Table table = null;

 ResultScanner rScanner = null;

 try {
 table = conn.getTable(tableName);

 Scan scan = new Scan();
 scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));
 // Set the filter criteria.
 SingleColumnValueFilter filter = new SingleColumnValueFilter(
 Bytes.toBytes("info"), Bytes.toBytes("name"), CompareOperator.EQUAL,
 Bytes.toBytes("Xu Bing"));
 scan.setFilter(filter);
 // Submit a scan request.
 rScanner = table.getScanner(scan);
 // Print query results.
 for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
 for (Cell cell : r.rawCells()) {
 LOG.info("{}:{}", Bytes.toString(CellUtil.cloneRow(cell)),
 Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
 Bytes.toString(CellUtil.cloneValue(cell)));
 }
 }
 LOG.info("Single column value filter successfully.");
 } catch (IOException e) {
 LOG.error("Single column value filter failed " + e);
 } finally {
 if (rScanner != null) {
 // Close the scanner object.
 rScanner.close();
 }
 if (table != null) {
 try {
 // Close the HTable object.
 table.close();
 } catch (IOException e) {
 LOG.error("Single column value filter failed " + e);
 }
 }
 }
}
```

```
 } catch (IOException e) {
 LOG.error("Close table failed " ,e);
 }
}
}
LOG.info("Exiting testSingleColumnValueFilter.");
}
```

## Precautions

Currently, secondary indexes do not support the comparators that use objects of the `SubstringComparator` class as filters.

For example, the following sample code is not supported:

```
Scan scan = new Scan();
filterList = new FilterList(FilterList.Operator.MUST_PASS_ALL);
filterList.addFilter(new SingleColumnValueFilter(Bytes
.toBytes(columnFamily), Bytes.toBytes(qualifier),
CompareOperator.EQUAL, new SubstringComparator(substring)));
scan.setFilter(filterList);
```

### 12.3.3.12 Creating a Secondary Index

#### Function

You can manage HBase secondary indexes using methods provided in **org.apache.hadoop.hbase.hindex.client.HIndexAdmin**. This class provides methods of creating an index.

#### NOTE

Secondary indexes cannot be modified. If you need to modify them, delete old indexes and create new ones.

#### Example Code

The following code snippet belongs to the `createIndex` method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void createIndex() {
 LOG.info("Entering createIndex.");

 String indexName = "index_name";
 // Create hindex instance
 TableIndices tableIndices = new TableIndices();
 IndexSpecification iSpec = new IndexSpecification(indexName);
 iSpec.addIndexColumn(ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("info")).build(),
 "name", ValueType.STRING);// Note[1]
 tableIndices.addIndex(iSpec);

 HIndexAdmin iAdmin = null;
 Admin admin = null;
 try {

 admin = conn.getAdmin();
 iAdmin = HIndexClient.newHIndexAdmin(admin);

 // add index to the table
 iAdmin.addIndices(tableName, tableIndices);

 LOG.info("Create index successfully.");
 } catch (IOException e) {
```

```

LOG.error("Create index failed ",e);
} finally {
if (admin != null) {
try {
admin.close();
} catch (IOException e) {
LOG.error("Close admin failed ",e);
}
}
if (iAdmin != null) {
try {
// Close IndexAdmin Object
iAdmin.close();
} catch (IOException e) {
LOG.error("Close admin failed ",e);
}
}
}
LOG.info("Exiting createIndex.");
}

```

By default, newly created level-2 indexes are disabled. To enable a specified level-2 index, see the following code snippet. The following code snippet belongs to the enableIndex method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```

public void enableIndex() {
LOG.info("Entering createIndex.");

// Name of the index to be enabled
String indexName = "index_name";

List<String> indexNameList = new ArrayList<String>();
indexNameList.add(indexName);

HIndexAdmin iAdmin = null;
Admin admin = null;
try {
admin = conn.getAdmin();
iAdmin = HIndexClient.newHIndexAdmin(admin);

// Alternately, enable the specified indices
iAdmin.enableIndices(tableName, indexNameList);
LOG.info("Successfully enable indices {} of the table {}", indexNameList, tableName);
} catch (IOException e) {
LOG.error("Failed to enable indices {} of the table {} . {}", indexNameList, tableName, e);
} finally {
if (admin != null) {
try {
admin.close();
} catch (IOException e) {
LOG.error("Close admin failed ", e);
}
}
if (iAdmin != null) {
try {
iAdmin.close();
} catch (IOException e) {
LOG.error("Close admin failed ", e);
}
}
}
}
}

```

## Precautions

Create a combination index.

HBase supports creation of secondary indexes on multiple fields, for example, the name and age columns.

```
HIndexSpecification iSpecUnite = new HIndexSpecification(indexName);
iSpecUnite.addIndexColumn(new HColumnDescriptor("info"), "name", ValueType.String, 10);
iSpecUnite.addIndexColumn(new HColumnDescriptor("info"), "age", ValueType.String, 3);
```

## Related Operations

### Create an index table by running a command.

You can also use the TableIndexer tool to create an index in an existing user table.

#### NOTE

The **<table\_name>** user table must exist.

```
hbase org.apache.hadoop.hbase.hindex.mapreduce.TableIndexer -Dtablename.to.index=<table_name> -
Dindexspecs.to.add='IDX1=>cf1:[q1->datatype];cf2:[q1->datatype],[q2->datatype],[q3-
>datatype]#IDX2=>cf1:[q5->datatype]' -Dindexnames.to.build='IDX1'
```

A number sign "#" is used to separate indexes. A semicolon ";" is used to separate column families. A comma "," is used to separate columns.

**tablename.to.index:** indicates the name of the table where the index is created.

**indexspecs.to.add:** indicates the user table columns corresponding to the index.

The parameters in the command are described as follows:

- **IDX1:** indicates the index name.
- **cf1:** indicates the column family name.
- **q1:** indicates the column name.
- **datatype:** indicates the data type. Only the Integer, String, Double, Float, Long, Short, Byte and Char formats are supported.

### 12.3.3.13 Deleting an Index

#### Function

You can manage HBase secondary indexes using methods provided in **org.apache.hadoop.hbase.hindex.client.HIndexAdmin**. This class provides methods of querying and deleting an index.

#### Example Code

The following code snippet belongs to the **dropIndex** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void dropIndex() {
 LOG.info("Entering dropIndex.");
 String indexName = "index_name";
 List<String> indexNameList = new ArrayList<String>();
 indexNameList.add(indexName);

 IndexAdmin iAdmin = null;
 try {
 // Instantiate HIndexAdmin Object
 iAdmin = HIndexClient.newHIndexAdmin(conn.getAdmin());
 // Delete Secondary Index
 }
```



```
iAdmin.dropIndex(tableName, indexNameList);

LOG.info("Drop index successfully.");
} catch (IOException e) {
 LOG.error("Drop index failed.");
} finally {
 if (iAdmin != null) {
 try {
 // Close Secondary Index
 iAdmin.close();
 } catch (IOException e) {
 LOG.error("Close admin failed.");
 }
 }
}
LOG.info("Exiting dropIndex.");
}
```

### 12.3.3.14 Secondary Index-based Query

#### Function

In user tables with secondary indexes, you can use Filter to query data. The data query performance is higher than that in user tables without secondary indexes.

#### NOTE

- HIndex supports three Filter types: SingleColumnValueFilter, SingleColumnValueExcludeFilter, and SingleColumnValuePartitionFilter.
- HIndex supports the following Comparator types: binary comparator, bit comparator, long comparator, decimal comparator, double comparator, float comparator, int comparator, and null comparator.

The secondary index usage rules are as follows:

- For scenarios in which a single index is created for one or multiple columns:
  - When you use this column for AND or OR query filtering, the index is used to improve the query performance.  
For example, Filter\_Condition(IndexCol1) AND/OR Filter\_Condition(IndexCol2).
  - When you use "Index Column AND Non-Index Column" for filtering in the query, this index is used to improve the query performance.  
For example, Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol2) AND Filter\_Condition(NonIndexCol1).
  - When you use "Index Column OR Non-Index Column" for filtering in the query, the index is not used and the query performance is not improved.  
For example, Filter\_Condition(IndexCol1) AND/OR Filter\_Condition(IndexCol2) OR Filter\_Condition(NonIndexCol1).
- For scenarios in which a combination index is created for multiple columns:
  - When the columns used for query are all or part of the columns of the combination index and are in the same sequence with the combination index, the index is used to improve the query performance.  
For example, a combination index is created for C1, C2, and C3. The index takes effect in the following scenarios:  
Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol2) AND Filter\_Condition(IndexCol3)

```
Filter_Condition(IndexCol1) AND Filter_Condition(IndexCol2)
Filter_Condition(IndexCol1)
```

The index does not take effect in the following scenarios:

```
Filter_Condition(IndexCol2) AND Filter_Condition(IndexCol3)
Filter_Condition(IndexCol1) AND Filter_Condition(IndexCol3)
Filter_Condition(IndexCol2)
Filter_Condition(IndexCol3)
```

- When you use "Index Column AND Non-Index Column" for filtering in the query, this index is used to improve the query performance.

For example:

```
Filter_Condition(IndexCol1) AND Filter_Condition(NonIndexCol1)
Filter_Condition(IndexCol1) AND Filter_Condition(IndexCol2) AND
Filter_Condition(NonIndexCol1)
```

- When you use "Index Column OR Non-Index Column" for filtering in the query, the index is not used and the query performance is not improved.

For example:

```
Filter_Condition(IndexCol1) OR Filter_Condition(NonIndexCol1)
(Filter_Condition(IndexCol1) AND Filter_Condition(IndexCol2))OR
(Filter_Condition(NonIndexCol1))
```

- When multiple columns are used for query, a value range can be specified only for the last column in the combination index and the other columns can only be set to a specified value.

For example, a combination index is created for C1, C2, and C3. In range query, a value range can be set only for C3 and the filter criterion is "C1 = XXX, C2 = XXX, and C3 = value range".

- For scenarios in which secondary index is created in a user table, you can use Filter to query data. The query results of the single and combination index with filter are the same as those in the table without secondary index. The data query performance is higher than that in user tables without secondary indexes.

## Example Code

The following code snippet belongs to the **testScanDataByIndex** method in the **HbaseSample** class of the **com.huawei.hadoop.hbase.example** package.

### Example: Query data using secondary indexes.

```
public void testScanDataByIndex() {
 LOG.info("Entering testScanDataByIndex.");
 Table table = null;
 ResultScanner scanner = null;
 try {
 table = conn.getTable(tableName);

 // Create a filter for indexed column.
 Filter filter = new SingleColumnValueFilter(Bytes.toBytes("info"), Bytes.toBytes("name"),
 CompareOperator.EQUAL, "Li Gang".getBytes());
 Scan scan = new Scan();
 scan.setFilter(filter);
 scanner = table.getScanner(scan);
 LOG.info("Scan indexed data.");
 }
}
```

```
for (Result result : scanner) {
 for (Cell cell : result.rawCells()) {
 LOG.info("{}:{}", Bytes.toString(CellUtil.cloneRow(cell)),
 Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
 Bytes.toString(CellUtil.cloneValue(cell)));
 }
}
LOG.info("Scan data by index successfully.");
} catch (IOException e) {
 LOG.error("Scan data by index failed.");
} finally {
 if (scanner != null) {
 // Close the scanner object.
 scanner.close();
 }
 try {
 if (table != null) {
 table.close();
 }
 } catch (IOException e) {
 LOG.error("Close table failed.");
 }
}
LOG.info("Exiting testScanDataByIndex.");
}
```

## Precaution

Create secondary indexes for the **name** field first.

## Related Operations

Query a table using a secondary index.

The following provides an example:

Add an index to the **name** column of the **info** column family in **hbase\_sample\_table**. Run the following command on the client:

```
hbase org.apache.hadoop.hbase.index.mapreduce.TableIndexer -Dtablename.to.index=hbase_sample_table -Dindexspecs.to.add='IDX1=>info:[name->String]' -Dindexnames.to.build='IDX1'
```

Query **info:name**. Run the following command on the HBase shell client:

```
>scan 'hbase_sample_table',
{FILTER=>"SingleColumnValueFilter(family,qualifier,compareOp,comparator,filterIfMissing,latestVersionOnly)"}
```

 NOTE

Use APIs to perform complex query on the HBase shell client.

The parameters are described as follows:

- **family**: indicates the column family where the column to be queried locates, such as **info**.
- **qualifier**: indicates the column to be queried, such as **name**.
- **compareOp**: indicates the comparison operator, such as = and >.
- **comparator**: indicates the target value to be queried, such as **binary:Zhang San**.
- **filterIfMissing**: indicates whether a row is filtered if the column does not exist in this row. The default value is **false**.
- **latestVersionOnly**: indicates whether only values of the latest version are to be queried. The default value is **false**.

For example:

```
>scan hbase_sample_table',{FILTER=>"SingleColumnValueFilter('info','name',=,'binary:Zhang San',true,true)"}
```

### 12.3.3.15 Multi-Point Region Division

#### Function

You can perform multi-point division by using **org.apache.hadoop.hbase.client.HBaseAdmin**. Note that the division operations take effect on empty regions only.

In this example, the multi-point division is performed on an HBase table by using **multiSplit**. The table will be split into 5 parts: "-∞~A", "A~D", "D~F", "F~H", and "H~+∞".

#### Example Code

The following code snippet belongs to the **testMultiSplit** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testMultiSplit() {
 LOG.info("Entering testMultiSplit.");

 Table table = null;
 Admin admin = null;
 try {
 admin = conn.getAdmin();

 // initialize a HTable object
 table = conn.getTable(tableName);
 Set<HRegionInfo> regionSet = new HashSet<HRegionInfo>();
 List<HRegionLocation> regionList = conn.getRegionLocator(tableName).getAllRegionLocations();
 for(HRegionLocation hrl : regionList){
 regionSet.add(hrl.getRegionInfo());
 }
 byte[][] sk = new byte[4][];
 sk[0] = "A".getBytes();
 sk[1] = "D".getBytes();
 sk[2] = "F".getBytes();
 sk[3] = "H".getBytes();
 for (RegionInfo regionInfo : regionSet) {
 admin.multiSplitSync(regionInfo.getRegionName(), sk);
 }
 LOG.info("MultiSplit successfully.");
 } catch (Exception e) {
```

```
LOG.error("MultiSplit failed.");
} finally {
 if (table != null) {
 try {
 // Close table object
 table.close();
 } catch (IOException e) {
 LOG.error("Close table failed " ,e);
 }
 }
 if (admin != null) {
 try {
 // Close the Admin object.
 admin.close();
 } catch (IOException e) {
 LOG.error("Close admin failed " ,e);
 }
 }
}
LOG.info("Exiting testMultiSplit.");
}
```

Note that the division operations take effect on empty regions only.

### 12.3.3.16 Creating a Phoenix Table

#### Function

Phoenix can be installed on HBase to enable it to support SQL and JDBC APIs. This way, SQL users can access the HBase cluster.

#### Example Code

The following code snippet belongs to the **testCreateTable** method in the **PhoenixSample** class of the **com.huawei.bigdata.hbase.examplespackage**.

```
/**
 * Create Table
 */
public void testCreateTable() {
 LOG.info("Entering testCreateTable.");
 String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
 // Create table
 String createTableSQL =
 "CREATE TABLE IF NOT EXISTS TEST (id integer not null primary key, name varchar, "
 + "account char(6), birth date)";
 try (Connection conn = DriverManager.getConnection(url, props);
 Statement stat = conn.createStatement()) {
 // Execute Create SQL
 stat.executeUpdate(createTableSQL);
 LOG.info("Create table successfully.");
 } catch (Exception e) {
 LOG.error("Create table failed.", e);
 }
 LOG.info("Exiting testCreateTable.");
}
/**
 * Drop Table
 */
public void testDrop() {
 LOG.info("Entering testDrop.");
 String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
 // Delete table
 String dropTableSQL = "DROP TABLE TEST";

 try (Connection conn = DriverManager.getConnection(url, props);
```

```
Statement stat = conn.createStatement() {
 stat.executeUpdate(dropTableSQL);
 LOG.info("Drop successfully.");
} catch (Exception e) {
 LOG.error("Drop failed.", e);
}
LOG.info("Exiting testDrop.");
}
```

### 12.3.3.17 Writing Data to the PhoenixTable

#### Function

The Phoenix table enables data writing in HBase.

#### Example Code

The following code snippet belongs to the **testPut** method in the **PhoenixSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
/**
 * Put data
 */
public void testPut() {
 LOG.info("Entering testPut.");
 String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
 // Insert
 String upsertSQL =
 "UPSERT INTO TEST VALUES(1,'John','100000', TO_DATE('1980-01-01','yyyy-MM-dd'))";
 try (Connection conn = DriverManager.getConnection(url, props);
 Statement stat = conn.createStatement()){
 // Execute Update SQL
 stat.executeUpdate(upsertSQL);
 conn.commit();
 LOG.info("Put successfully.");
 } catch (Exception e) {
 LOG.error("Put failed.", e);
 }
 LOG.info("Exiting testPut.");
}
```

### 12.3.3.18 Reading the PhoenixTable

#### Function

The Phoenix table enables data reading.

#### Example Code

The following code snippet belongs to the **testSelect** method in the **PhoenixSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
/**
 * Select Data
 */
public void testSelect() {
 LOG.info("Entering testSelect.");
 String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
 // Query
 String querySQL = "SELECT * FROM TEST WHERE id = ?";
 Connection conn = null;
 PreparedStatement preStat = null;
```

```
Statement stat = null;
ResultSet result = null;
try {
 // Create Connection
 conn = DriverManager.getConnection(url, props);
 // Create Statement
 stat = conn.createStatement();
 // Create PreparedStatement
 preStat = conn.prepareStatement(querySQL);
 // Execute query
 preStat.setInt(1, 1);
 result = preStat.executeQuery();
 // Get result
 while (result.next()) {
 int id = result.getInt("id");
 String name = result.getString(1);
 System.out.println("id: " + id);
 System.out.println("name: " + name);
 }
 LOG.info("Select successfully.");
} catch (Exception e) {
 LOG.error("Select failed.", e);
} finally {
 if (null != result) {
 try {
 result.close();
 } catch (Exception e2) {
 LOG.error("Result close failed.", e2);
 }
 }
 if (null != stat) {
 try {
 stat.close();
 } catch (Exception e2) {
 LOG.error("Stat close failed.", e2);
 }
 }
 if (null != conn) {
 try {
 conn.close();
 } catch (Exception e2) {
 LOG.error("Connection close failed.", e2);
 }
 }
}
LOG.info("Exiting testSelect.");
}
```

### 12.3.3.19 Accessing Multiple ZooKeepers

#### Function

This function allows simultaneous access to FusionInsight ZooKeeper from the HBase client and third-party ZooKeeper from the customer application in the same client process.

#### Example code

The following code snippet is in the **TestZKSample** class of the **hbase-zk-example** \src\main\java\com\huawei\hadoop\hbase\example. You need to pay attention to the **login** and **connectApacheZK** methods.

```
private static void login(String keytabFile, String principal) throws IOException {
 conf = HBaseConfiguration.create();
 //In Windows environment
```

```
[1] String confDirPath = TestZKSample.class.getClassLoader().getResource("").getPath() + File.separator;
//In Linux environment
//String confDirPath = System.getProperty("user.dir") + File.separator + "conf" + File.separator;

// Set zoo.cfg for hbase to connect to fi zookeeper.
conf.set("hbase.client.zookeeper.config.path", confDirPath + "zoo.cfg");
if (User.isHBaseSecurityEnabled(conf)) {
 // jaas.conf file, it is included in the client package file
 System.setProperty("java.security.auth.login.config", confDirPath + "jaas.conf");
 // set the kerberos server info,point to the kerberosclient
 System.setProperty("java.security.krb5.conf", confDirPath + "krb5.conf");
 // set the keytab file name
 conf.set("username.client.keytab.file", confDirPath + keytabFile);
 // set the user's principal
 try {
 conf.set("username.client.kerberos.principal", principal);
 User.login(conf, "username.client.keytab.file", "username.client.kerberos.principal",
 InetAddress.getLocalHost().getCanonicalHostName());
 } catch (IOException e) {
 throw new IOException("Login failed.", e);
 }
}
}
private void connectApacheZK() throws IOException, org.apache.zookeeper KeeperException {
 try {
 // Create apache zookeeper connection.
 ZooKeeper digestZk = new ZooKeeper("127.0.0.1:2181", 60000, null);
 LOG.info("digest directory: {}", digestZk.getChildren("/", null));
 LOG.info("Successfully connect to apache zookeeper.");
 } catch (InterruptedException e) {
 LOG.error("Found error when connect apache zookeeper ", e);
 }
}
}
```

- [1] **userdir** obtains the **conf** directory in the resource path after compilation. Save the **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml** configuration files required for initialization and the user credential file used for security authentication to the **src/main/resources** directory.
- The **jaas.conf** file specified by the **java.security.auth.login.config** parameter in the **login** method is used to set the authentication information for accessing ZooKeeper. The example code contains the **Client\_new** and **Client** configurations. The **Client\_new** configuration is used to access FusionInsight ZooKeeper and the **Client** configuration is used to access Apache ZooKeeper.
- The **hbase.client.zookeeper.config.path** parameter in the **login** method controls the access to the FusionInsight ZooKeeper client. The following parameters are involved:
  - **zookeeper.sasl.clientconfig**: Specifies the configuration in the **jaas.conf** file used for accessing FusionInsight ZooKeeper.
  - **zookeeper.server.principal**: Specifies the principle of the ZooKeeper server.
  - **zookeeper.sasl.client**: If the MRS cluster works in the security mode, set this parameter to **true**. Otherwise, set this parameter to **false**. If this parameter is set to **false**, the **zookeeper.sasl.clientconfig** and **zookeeper.server.principal** parameters do not take effect.



### 12.3.3.20 Querying Cluster Information Using REST

#### Function

Use the REST service to transfer the URL consisting of the host and port to obtain the cluster version and status information through HTTPS.

#### Example Code

- Obtaining the cluster version information

The following code snippets are in the **getClusterVersion** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getClusterVersion(String url) {
 String endpoint = "/version/cluster";
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
 handleNormalResult((Optional<ResultModel>) result);
}
```

- Obtaining the cluster status information

The following code snippets are in the **getClusterStatus** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getClusterStatus(String url) {
 String endpoint = "/status/cluster";
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
 handleNormalResult(result);
}
```

### 12.3.3.21 Obtaining All Tables Using REST

#### Function

Use the REST service and transfer the URL consisting of the host and port to obtain all tables using HTTPS.

#### Example Code

The following code snippets are in the **getAllUserTables** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getAllUserTables(String url) {
 String endpoint = "/";
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
 handleNormalResult(result);
}
```

### 12.3.3.22 Operate Namespaces Using REST

#### Function

Use the REST service to import the URL consisting of the host and port and the specified namespace, Use HTTPS to create, query, and delete namespaces, and obtain tables in the specified namespace.

## Example Code

- Invoking methods

```
// Namespace operations.
createNamespace(url, "testNs");
getAllNamespace(url);
deleteNamespace(url, "testNs");
getAllNamespaceTables(url, "default");
```

- Creating a namespace

The following code snippets are in the **createNamespace** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void createNamespace(String url, String namespace) {
 String endpoint = "/namespaces/" + namespace;
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.POST, null);
 if (result.orElse(new ResultModel()).getStatusCode() == HttpStatus.SC_CREATED) {
 LOG.info("Create namespace '{}' success.", namespace);
 } else {
 LOG.error("Create namespace '{}' failed.", namespace);
 }
}
```

- Querying all namespaces

The following code snippets are in the **getAllNamespace** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getAllNamespace(String url) {
 String endpoint = "/namespaces";
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
 handleNormalResult(result);
}
```

- Deleting a specified namespace

The following code snippets are in the **deleteNamespace** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void deleteNamespace(String url, String namespace) {
 String endpoint = "/namespaces/" + namespace;
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.DELETE, null);
 if (result.orElse(new ResultModel()).getStatusCode() == HttpStatus.SC_OK) {
 LOG.info("Delete namespace '{}' success.", namespace);
 } else {
 LOG.error("Delete namespace '{}' failed.", namespace);
 }
}
```

- Obtain tables in a specified namespace.

The following code snippets are in the **getAllNamespaceTables** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getAllNamespaceTables(String url, String namespace) {
 String endpoint = "/namespaces/" + namespace + "/tables";
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
 handleNormalResult(result);
}
```

### 12.3.3.23 Operate Tables Using REST

#### Function

Use the REST service to transfer the URL consisting of the host and port as well as the specified **tableName** and **jsonHTD** to query, modify, create, and delete table information through HTTPS.

#### Example Code

- Invoking methods

```
// Add a table with specified info.
createTable(url, "testRest",
 "{\"name\":\"default:testRest\",\"ColumnSchema\":{\"name\":\"cf1\"},\" + \"{name\":\"cf2\"}}");

// Add column family 'testCF1' if not exist, else update the 'VERSIONS' to 3.
// Notes: The unspecified property of this column family will be updated to default value.
modifyTable(url, "testRest",
 "{\"name\":\"testRest\",\"ColumnSchema\":{\"name\":\"testCF1\", \"VERSIONS\":\"3\" +
 \"}}");

// Describe specific Table.
descTable(url, "default:testRest");

// delete a table with specified info.
deleteTable(url, "default:testRest",
 "{\"name\":\"default:testRest\",\"ColumnSchema\":{\"name\":\"testCF\", \"VERSIONS\":\"3\"}}");
```

- Querying table information

The following code snippets are in the **descTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void descTable(String url, String tableName) {
 String endpoint = "/" + tableName + "/schema";
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
 handleNormalResult((Optional<ResultModel>) result);
}
```

- Modifying table information

The following code snippets are in the **modifyTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void modifyTable(String url, String tableName, String jsonHTD) {
 LOG.info("Start modify table.");
 String endpoint = "/" + tableName + "/schema";
 JsonElement tableDesc = new JsonParser().parse(jsonHTD);

 // Add a new column family or modify it.
 handleNormalResult(sendAction(url + endpoint, MethodType.POST, tableDesc));
}
```

- Creating a table

The following code snippets are in the **createTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void createTable(String url, String tableName, String jsonHTD) {
 LOG.info("Start create table.");
 String endpoint = "/" + tableName + "/schema";
 JsonElement tableDesc = new JsonParser().parse(jsonHTD);

 // Add a table.
```

```
handleCreateTableResult(sendAction(url + endpoint, MethodType.PUT, tableDesc));
}
```

- Deleting a table

The following code snippets are in the **deleteTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void deleteTable(String url, String tableName, String jsonHTD) {
 LOG.info("Start delete table.");
 String endpoint = "/" + tableName + "/schema";
 JsonElement tableDesc = new JsonParser().parse(jsonHTD);

 // delete a table.
 handleNormalResult(sendAction(url + endpoint, MethodType.DELETE, tableDesc));
}
```

### 12.3.3.24 Accessing the ThriftServer Operation Table

#### Function

After importing the host where the ThriftServer instances are located and the port that provides services, you can create a Thrift client using the authentication credential and configuration file, access ThriftServer, and obtain table names, create a table, and delete a table based on the specified namespace.

#### Example Code

- Invoking methods

```
// Get table of specified namespace. getTableNamesByNamespace(client, "default");
// Create table. createTable(client, TABLE_NAME);
// Delete specified table.
deleteTable(client, TABLE_NAME);
```

- Obtains table names based on the specified namespace.

The following code snippets are in the **getTableNamesByNamespace** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getTableNamesByNamespace(THBaseService.Iface client, String namespace) throws
TException {
 client.getTableNamesByNamespace(namespace)
 .forEach(
 tTableName -> LOGGER.info("{} ", TableName.valueOf(tTableName.getNs()),
 tTableName.getQualifier()));
}
```

- Creating a table

The following code snippets are in the **createTable** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void createTable(THBaseService.Iface client, String tableName) throws TException,
IOException {
 TTableName table = getTableName(tableName);
 TTableDescriptor descriptor = new TTableDescriptor(table);
 descriptor.setColumns(
 Collections.singletonList(new
 TColumnFamilyDescriptor().setName(COLUMN_FAMILY.getBytes())));
 if (client.tableExists(table)) {
 LOGGER.warn("Table {} is exists, delete it.", tableName);
 client.disableTable(table);
 client.deleteTable(table);
 }
}
```

```
client.createTable(descriptor, null);
if (client.tableExists(table)) {
 LOGGER.info("Created {}. ", tableName);
} else {
 LOGGER.error("Create {} failed.", tableName);
}
}
```

- Deleting a table

The following code snippets are in the **deleteTable** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void deleteTable(THBaseService.Iface client, String tableName) throws TException,
IOException {
 TTableName table = getTableName(tableName);
 if (client.tableExists(table)) {
 client.disableTable(table);
 client.deleteTable(table);
 LOGGER.info("Deleted {}. ", tableName);
 } else {
 LOGGER.warn("{} not exist.", tableName);
 }
}
```

### 12.3.3.25 Accessing ThriftServer to Write Data

#### Function

After importing the host where the ThriftServer instances are located and the port that provides services, you can create a Thrift client using the authentication credential and configuration file, access the ThriftServer, and use Put and putMultiple to write data.

#### Example Code

- Invoking methods

```
// Write data with put.
putData(client, TABLE_NAME);
```

```
// Write data with putlist.
putDataList(client, TABLE_NAME);
```

- Using Put to write data.

The following code snippets are in the **putData** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void putData(THBaseService.Iface client, String tableName) throws TException {
 LOGGER.info("Test putData.");
 TPut put = new TPut();
 put.setRow("row1".getBytes());

 TColumnValue columnValue = new TColumnValue();
 columnValue.setFamily(COLUMN_FAMILY.getBytes());
 columnValue.setQualifier("q1".getBytes());
 columnValue.setValue("test value".getBytes());
 List<TColumnValue> columnValues = new ArrayList<>(1);
 columnValues.add(columnValue);
 put.setColumnValues(columnValues);

 ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
 client.put(table, put);
 LOGGER.info("Test putData done.");
}
```

- Using putMultiple to write data.

The following code snippets are in the **putDataList** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void putDataList(THBaseService.Iface client, String tableName) throws TException {
 LOGGER.info("Test putDataList.");
 TPut put1 = new TPut();
 put1.setRow("row2".getBytes());
 List<TPut> putList = new ArrayList<>();

 TColumnValue q1Value = new TColumnValue(ByteBuffer.wrap(COLUMN_FAMILY.getBytes()),
 ByteBuffer.wrap("q1".getBytes()), ByteBuffer.wrap("test value".getBytes()));
 TColumnValue q2Value = new TColumnValue(ByteBuffer.wrap(COLUMN_FAMILY.getBytes()),
 ByteBuffer.wrap("q2".getBytes()), ByteBuffer.wrap("test q2 value".getBytes()));
 List<TColumnValue> columnValues = new ArrayList<>(2);
 columnValues.add(q1Value);
 columnValues.add(q2Value);
 put1.setColumnValues(columnValues);
 putList.add(put1);

 TPut put2 = new TPut();
 put2.setRow("row3".getBytes());

 TColumnValue columnValue = new TColumnValue();
 columnValue.setFamily(COLUMN_FAMILY.getBytes());
 columnValue.setQualifier("q1".getBytes());
 columnValue.setValue("test q1 value".getBytes());
 put2.setColumnValues(Collections.singletonList(columnValue));
 putList.add(put2);

 ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
 client.putMultiple(table, putList);
 LOGGER.info("Test putDataList done.");
}
```

### 12.3.3.26 Accessing ThriftServer to Read Data

#### Function

After importing the host where the ThriftServer instances are located and the port that provides services, you can create a Thrift client using the authentication credential and configuration file, access the ThriftServer, and use **get** and **scan** methods to read data.

#### Example Code

- Invoking methods
 

```
// Get data with single get.
getData(client, TABLE_NAME);

// Get data with getlist.
getDataList(client, TABLE_NAME);

// Scan data.
scanData(client, TABLE_NAME);
```

- Using the **get** method to write data.

The following code snippets are in the **getData** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getData(THBaseService.Iface client, String tableName) throws TException {
 LOGGER.info("Test getData.");
```

```
TGet get = new TGet();
get.setRow("row1".getBytes());

ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
TResult result = client.get(table, get);
printResult(result);
LOGGER.info("Test getData done.");
}
```

- Using the **getlist** method to write data.

The following code snippets are in the **getDataList** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getDataList(THBaseService.Iface client, String tableName) throws TException {
 LOGGER.info("Test getDataList.");
 List<TGet> getList = new ArrayList<>();
 TGet get1 = new TGet();
 get1.setRow("row1".getBytes());
 getList.add(get1);

 TGet get2 = new TGet();
 get2.setRow("row2".getBytes());
 getList.add(get2);

 ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
 List<TResult> resultList = client.getMultiple(table, getList);
 for (TResult tResult : resultList) {
 printResult(tResult);
 }
 LOGGER.info("Test getDataList done.");
}
```

- Using the **scan** method to write data.

The following code snippets are in the **scanData** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void scanData(THBaseService.Iface client, String tableName) throws TException {
 LOGGER.info("Test scanData.");
 int scannerId = -1;
 try {
 ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
 TScan scan = new TScan();
 scan.setLimit(500);
 scannerId = client.openScanner(table, scan);
 List<TResult> resultList = client.getScannerRows(scannerId, 100);
 while (resultList != null && !resultList.isEmpty()) {
 for (TResult tResult : resultList) {
 printResult(tResult);
 }
 resultList = client.getScannerRows(scannerId, 100);
 }
 } finally {
 if (scannerId != -1) {
 client.closeScanner(scannerId);
 LOGGER.info("Closed scanner {}. ", scannerId);
 }
 }
 LOGGER.info("Test scanData done.");
}
```

### 12.3.3.27 Using HBase Dual-Read

#### Scenario

The HBase client application loads the configuration items of the active and standby clusters by customization to implement the dual-read capability. HBase dual-read is a key feature that improves the high availability of the HBase cluster system. It applies to four query scenarios: reading data using **Get**, reading data in batches using **Get**, reading data using **Scan**, and querying data using a secondary index. HBase can read data from the active and standby clusters at the same time, reducing the query glitch time. The advantages are as follows:

- High success rate: The concurrent dual-read mechanism ensures a high success rate of read requests.
- High availability: When a single cluster is faulty, the query service is not interrupted. A short network jitter does not prolong the query time.
- High generality: The dual-read feature does not support dual-write, but does not affect the original real-time write scenario.
- Ease-of-use: Client encapsulation is performed, which is not sensed by services.

#### NOTE

Restrictions on HBase dual-read:

- The HBase dual-read feature is implemented based on replication. Data read from the standby cluster may be different from that from the active cluster. Therefore, only eventual consistency can be achieved.
- Currently, the HBase dual-read feature is used only for query. When the active cluster breaks down, the latest data cannot be synchronized. As a result, the latest data cannot be queried in the standby cluster.
- A **Scan** operation of HBase may be split into multiple RPC operations. Data may not be completely the same because related session information is not synchronized between different clusters. Therefore, the dual-read feature takes effect only when an RPC operation is performed for the first time. Requests before ResultScanner close access the cluster used for the first RPC operation.
- The HBase Admin API and real-time write API access only the active cluster. Therefore, after the active cluster breaks down, the Admin API and real-time write API are unavailable, and only the **Get** and **Scan** query services are available.

#### Procedure

- Step 1** Save the keytab authentication files **user.keytab** and **krb5.conf** of the active cluster obtained when [Preparing a Developer Account](#) to the **src/main/resources/conf** secondary sample directory.
- Step 2** Obtain the client configuration files **core-site.xml**, **hbase-site.xml**, and **hdfs-site.xml** of the HBase active cluster and save them to the **src/main/resources/conf/active** directory. This directory needs to be created by yourself. For details, see [Preparing for Development and Operating Environment](#).
- Step 3** Obtain the client configuration files **core-site.xml**, **hbase-site.xml**, and **hdfs-site.xml** of the standby cluster and save them to the **src/main/resources/conf/standby** directory. For details, see [Preparing for Development and Operating Environment](#).



- Step 4** Create the **hbase-dual.xml** configuration file and save it to the **src/main/resources/conf/** directory. This directory needs to be created by yourself. For details about the configuration items in the configuration file, see [Table 12-6](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<!--Configuration file directory of the active cluster-->
 <property>
 <name>hbase.dualclient.active.cluster.configuration.path</name>
 <value>{Sample code directory}\src\main\resources\active</value>
 </property>
<!--Configuration file directory of the standby cluster-->
 <property>
 <name>hbase.dualclient.standby.cluster.configuration.path</name>
 <value>{Sample code directory}\src\main\resources\standby</value>
 </property>
<!--Connection implementation of the dual-read mode-->
 <property>
 <name>hbase.client.connection.impl</name>
 <value>org.apache.hadoop.hbase.client.HBaseMultiClusterConnectionImpl</value>
 </property>
<!--Security mode-->
 <property>
 <name>hbase.security.authentication</name>
 <value>kerberos</value>
 </property>
<!--Security mode-->
 <property>
 <name>hadoop.security.authentication</name>
 <value>kerberos</value>
 </property>
</configuration>
```

----End

## Sample Code

- Creating a dual-read configuration. The following code snippet belongs to the **init** method in **TestMain** class of the **com.huawei.bigdata.hbase.examples** packet.

```
private static void init() throws IOException {
 // Default load from conf directory
 conf = HBaseConfiguration.create();
 //In Windows environment
 String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() + File.separator;
 //In Linux environment
 //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
 conf.addResource(new Path(userdir + "hbase-dual.xml"), false);
}
```

- Determining the data source cluster
  - GET request. The following code snippet belongs to the **testGet** method in **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
Result result = table.get(get);
if (result instanceof DualResult) {
 LOG.info(((DualResult)result).getClusterId());
}
```

- Scan request. The following code snippet belongs to the **testScanData** method in **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
ResultScanner rScanner = table.getScanner(scan);
if (rScanner instanceof HBaseMultiScanner) {
 LOG.info(((HBaseMultiScanner)rScanner).getClusterId());
}
```

- The client can print metric information.  
Add the following content to the **log4j.properties** file so that the client can export metric information to the specified file: For details about the metrics, see [Printing Metric Information](#)

```
log4j.logger.DUAL=debug,DUAL
log4j.appender.DUAL=org.apache.log4j.RollingFileAppender
log4j.appender.DUAL.File=/var/log/dual.log //Local dual-read log path on the client. Change the value
to the actual directory, but ensure that the directory has the write permission.
log4j.additivity.DUAL=false
log4j.appender.DUAL.MaxFileSize=${hbase.log.maxfilesize}
log4j.appender.DUAL.MaxBackupIndex=${hbase.log.maxbackupindex}
log4j.appender.DUAL.layout=org.apache.log4j.PatternLayout
log4j.appender.DUAL.layout.ConversionPattern=%d{ISO8601} %-5p [%t] %c{2}: %m%n
```

## 12.4 Application Commissioning

### 12.4.1 Commissioning an Application in Windows

#### 12.4.1.1 Compiling and Running an Application

##### Scenario

After the code development is complete, you can run the application in the Windows development environment.

If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

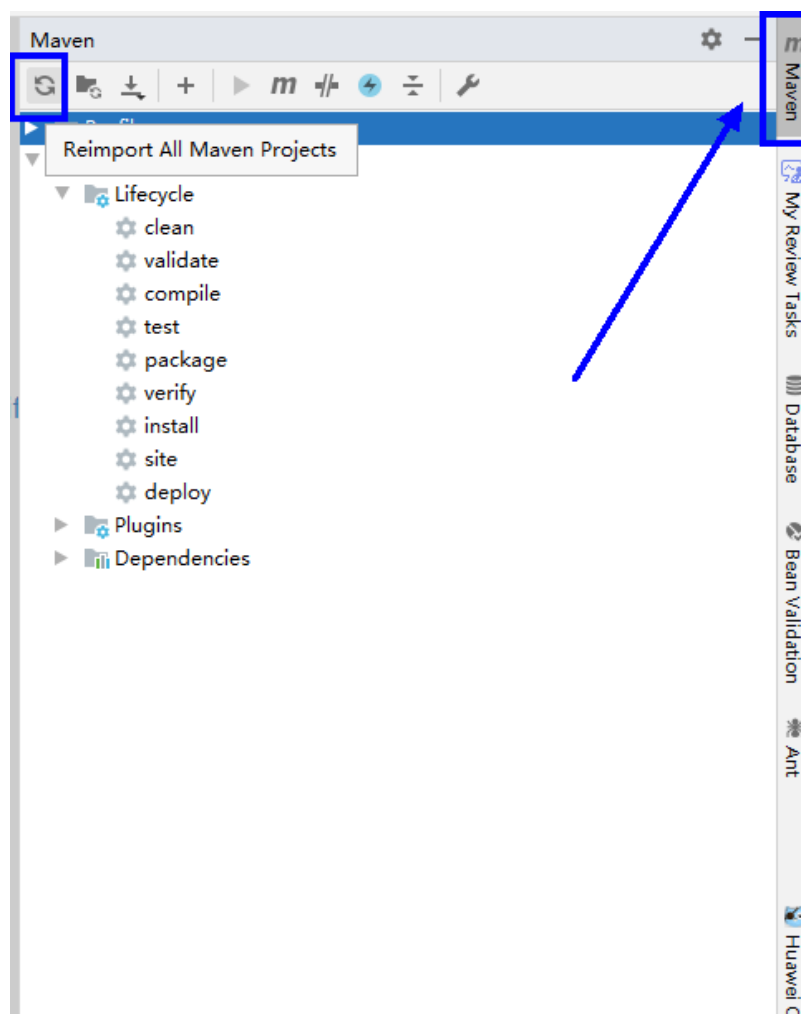
##### NOTE

- If IBM JDK is used in the Windows development environment, the application cannot be run in the Windows environment.
- You need to set the mapping between the host names and IP addresses of the access nodes in the hosts file on the local host where the sample code is run. The host names and IP addresses must be mapped one by one.

##### Procedure

- Step 1** Click **Reimport All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.

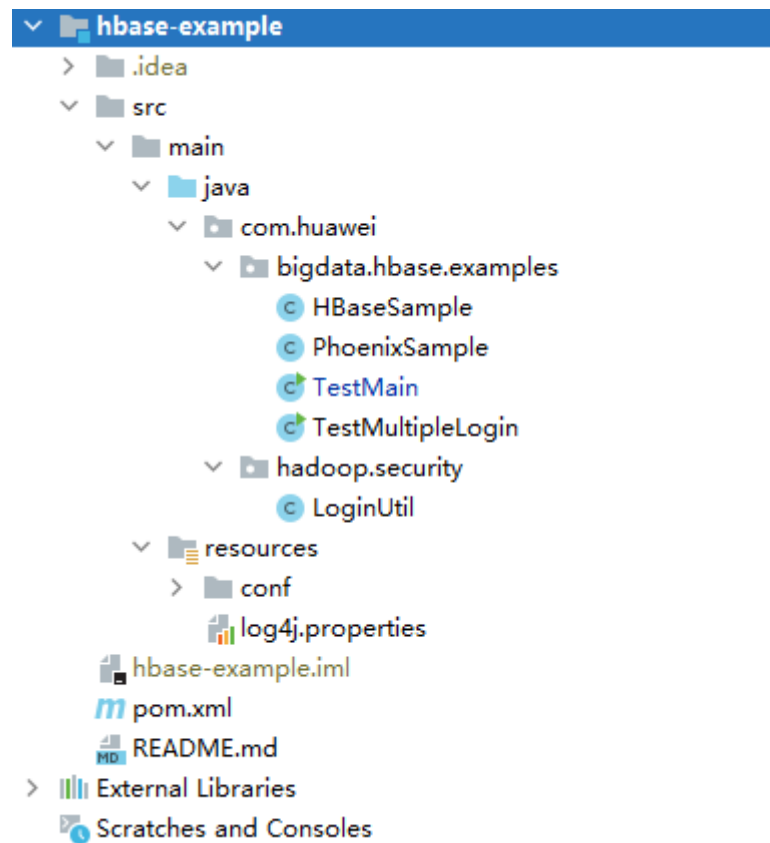
Figure 12-17 reimport projects



**Step 2** Compile and run an application.

Place the configuration file directory and modify the code to match the login user.  
See [Figure 12-18](#).

**Figure 12-18** Directory list of **hbase-example** to be compiled



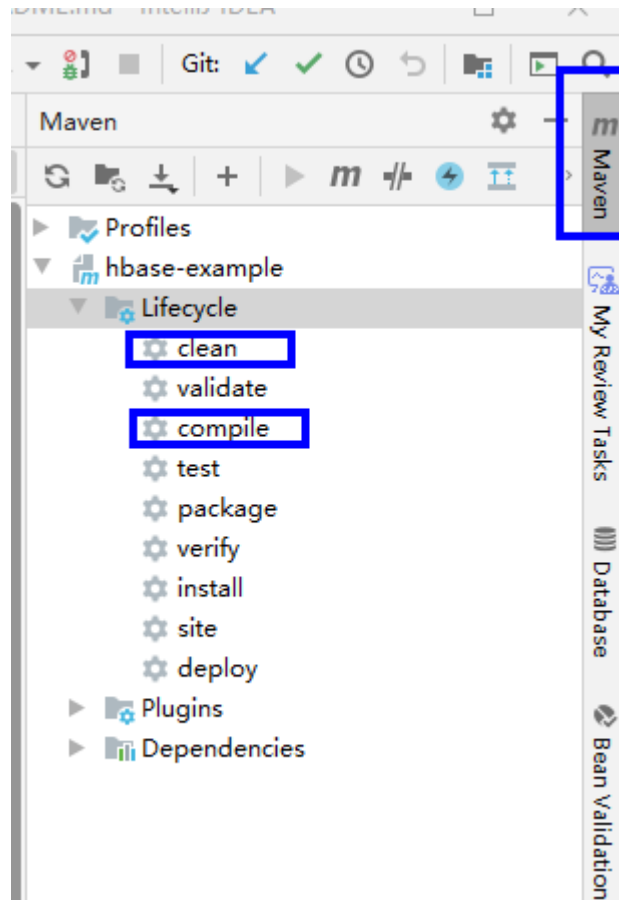
1. Compile an application.

– Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **compile**, double click **compile** to run the **compile** command of Maven.

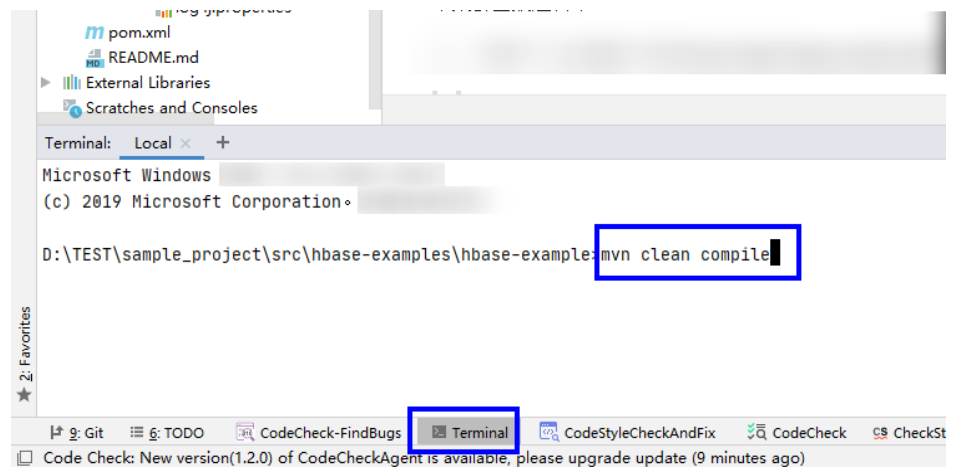
Figure 12-19 clean and compile tools of Maven



- Method 2:

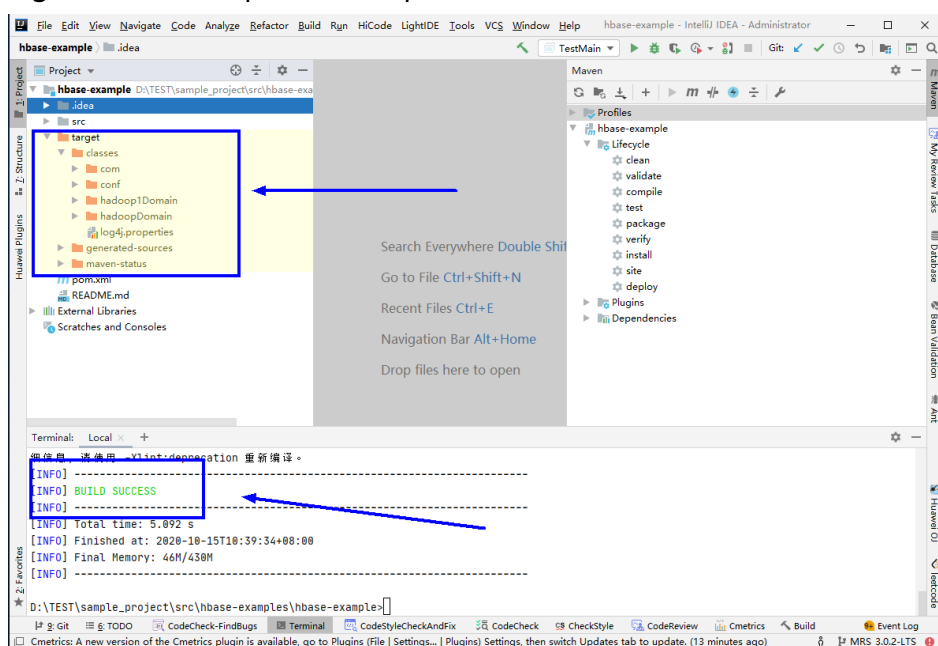
Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

Figure 12-20 Enter mvn clean compile in the IDEA Terminal text box.



After the compilation is complete, the message "Build Success" is displayed and the **target** directory is generated.

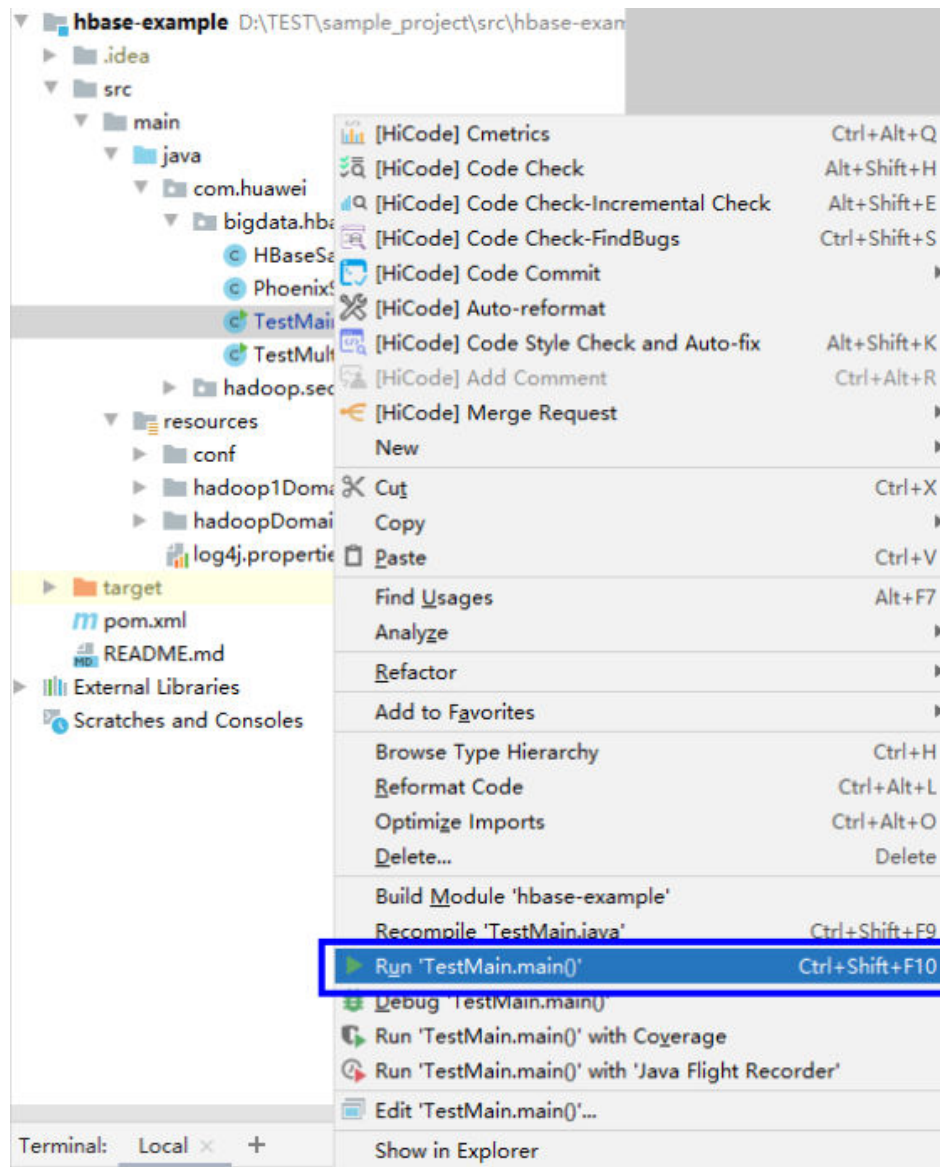
Figure 12-21 Compilation completed



2. Run the program.

Right-click the **TestMain.java** file and choose **Run>TestMain.main()** from the shortcut menu.

Figure 12-22 Run the application.



----End

### 12.4.1.2 Viewing Windows Commissioning Results

#### Scenario

After an HBase application is run, you can check the running result through one of the following methods:

- Viewing the IntelliJ IDEA running result.
- Viewing HBase logs.
- Logging in to the HBase WebUI, see **More Information > External Interfaces > WebUI**.
- Using HBase Shell command, see **More Information > External Interfaces > Shell**.

## Procedure

- The following information is displayed in the running results:

```

...
2020-09-09 22:11:48,496 INFO [main] example.TestMain: Entering testCreateTable.
2020-09-09 22:11:48,894 INFO [main] example.TestMain: Creating table...
2020-09-09 22:11:50,545 INFO [main] example.TestMain: Master:
10-1-131-140,16000,1441784082485
Number of backup masters: 1
 10-1-131-130,16000,1441784098969
Number of live region servers: 3
 10-1-131-150,16020,1441784158435
 10-1-131-130,16020,1441784126506
 10-1-131-140,16020,1441784118303
Number of dead region servers: 0
Average load: 1.0
Number of requests: 0
Number of regions: 3
Number of regions in transition: 0
2020-09-09 22:11:50,562 INFO [main] example.TestMain:
Lorg.apache.hadoop.hbase.NamespaceDescriptor;@11c6af6
2020-09-09 22:11:50,562 INFO [main] example.TestMain: Table created successfully.
2020-09-09 22:11:50,563 INFO [main] example.TestMain: Exiting testCreateTable.
2020-09-09 22:11:50,563 INFO [main] example.TestMain: Entering testMultiSplit.
2020-09-09 22:11:50,630 INFO [main] example.TestMain: MultiSplit successfully.
2020-09-09 22:11:50,630 INFO [main] example.TestMain: Exiting testMultiSplit.
2020-09-09 22:11:50,630 INFO [main] example.TestMain: Entering testPut.
2020-09-09 22:11:51,148 INFO [main] example.TestMain: Put successfully.
2020-09-09 22:11:51,148 INFO [main] example.TestMain: Exiting testPut.
2020-09-09 22:11:51,148 INFO [main] example.TestMain: Entering createIndex.
...

```

### NOTE

In the Windows environment, the following exception occurs but does not affect services.

```
java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.
```

- Log description

The log level is **INFO** by default and more detailed information can be viewed by changing the log level (**DEBUG**, **INFO**, **WARN**, **ERROR**, and **FATL**). You can modify the **log4j.properties** file to change log levels, for example:

```

hbase.root.logger=INFO,console
...
log4j.logger.org.apache.zookeeper=INFO
#log4j.logger.org.apache.hadoop.fs.FSNamesystem=DEBUG
log4j.logger.org.apache.hadoop.hbase=INFO
Make these two classes DEBUG-level. Make them DEBUG to see more zk debug.
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZKUtil=INFO
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZooKeeperWatcher=INFO
...

```

## 12.4.2 Commissioning an Application in Linux

### 12.4.2.1 Compiling and Running an Application When a Client Is Installed

#### Scenario

In a Linux environment where an HBase client is installed, you can upload the JAR package to the prepared Linux running environment and then run an application after the code development is complete.



## Prerequisites

- You have installed the HBase client.
- If the host where the client is installed is not a node in the cluster, set the mapping between the host name and the IP address in the **hosts** file on the node where the client locates. The host name and IP address must be in one-to-one mapping.

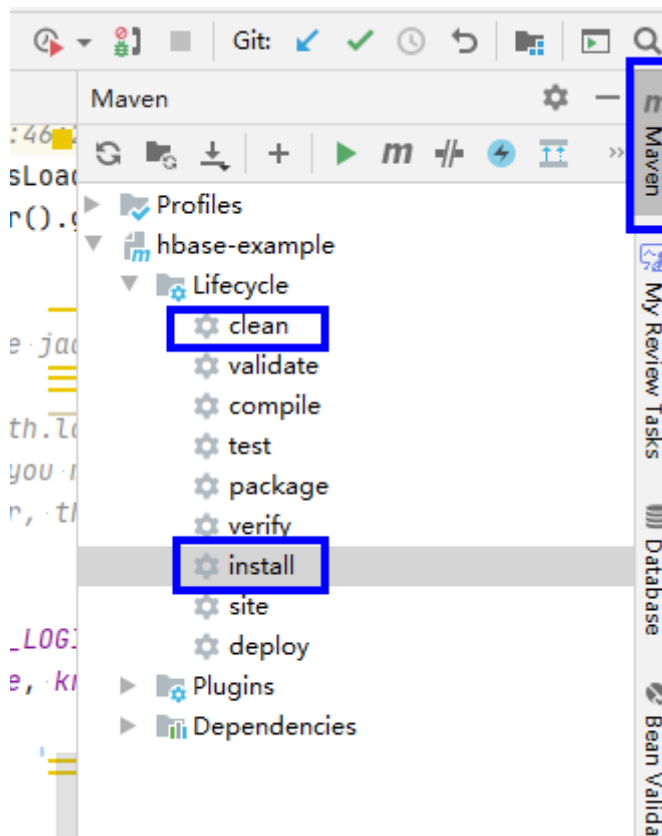
## Procedure

### Step 1 Export a JAR package.

You can build a JAR file in either of the following ways:

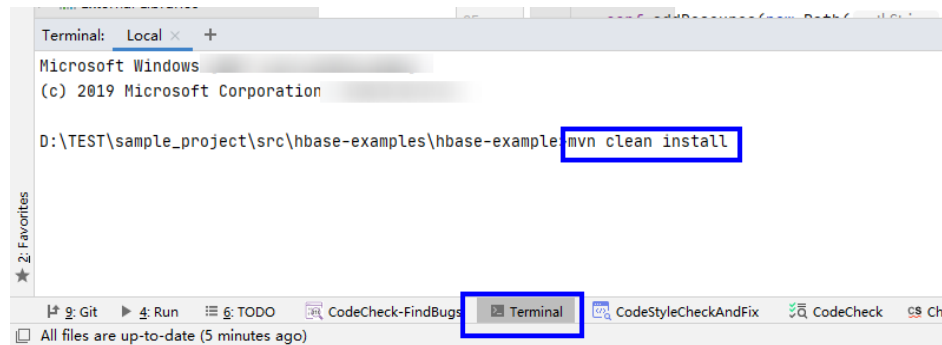
- Method 1:  
Choose **Maven** > *Sample projectname* > **Lifecycle** > **clean**, double click **clean** to run the **clean** command of Maven.  
Choose **Maven** > *Sample project name* > **Lifecycle** > **install**, double click **install** to run the **install** command of Maven.

Figure 12-23 Maven tools: **clean** and **install**



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the **pom.xml** file.

**Figure 12-24** Enter **mvn clean compile** in the IDEA **Terminal** text box.



After the compilation is completed, the message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

**Step 2** Export the JAR file on which the sample project depends.

Access the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of IDEA or by using other command line tools.

Run the command **mvn dependency:copy-dependencies -DoutputDirectory=lib**.

The **lib** folder is generated in the directory where the **pom.xml** file is located. The **lib** folder contains the JAR files on which the sample project depends.

**Step 3** Run the JAR package.

1. Before running the JAR package on the Linux client, run the following command to go to the client directory:

```
cd $BIGDATA_CLIENT_HOME
```

**NOTE**

**\$BIGDATA\_CLIENT\_HOME** is the installation directory of the HBase client.

2. Then run the following command:

```
source bigdata_env
```

**NOTE**

After the multi-instance function is enabled, run the following command to switch to the client of the specified service instance before performing application development for the HBase service instance.

For example, for HBase2, run the **source /opt/client/HBase2/component\_env** command.

3. Upload the JAR file (non-dependent JAR file) of the sample project generated in the application development environment to the **\$BIGDATA\_CLIENT\_HOME/HBase/hbase/lib** directory in the client running environment. Check the **\$BIGDATA\_CLIENT\_HOME/HBase/hbase/conf** directory based on the sample project **README.md**, and copy the configuration file and authentication file in the sample project to this directory.
4. Go to the **\$BIGDATA\_CLIENT\_HOME/HBase/hbase** directory and run the following command to run the JAR package. The task is complete.

```
hbase com.huawei.bigdata.hbase.examples.TestMain
```

In the preceding command, *hbase*  
*com.huawei.bigdata.hbase.examples.TestMain* is used as an example.

----End

## 12.4.2.2 Compiling and Running an Application When No Client Is Installed

### Scenario

In a Linux environment where no HBase client is installed, you can upload the JAR package to the Linux and then run an application after the code development is complete.

### Prerequisites

- A JDK has been installed in the Linux environment. The version of the JDK must be consistent with that of the JDK used by the JAR package exported by IntelliJ IDEA.
- If the Linux host is not a node in the cluster, set the mapping between the host name and the IP address in the **hosts** file on the node. The host name and IP address must be in one-to-one mapping.

### Procedure

**Step 1** Export a JAR package. For details, see [Step 1](#) in section [Compiling and Running an Application When a Client Is Installed](#).

**Step 2** Prepare for the required JAR packages and configuration files.

1. In the Linux environment, create a directory, for example, **/opt/test**, and create subdirectories **lib** and **conf**. Export the JAR package that the sample project depends on. For details about how to export the JAR package, see [Step 2](#) in [1.4.2.1 Compiling and Running an Application When a Client Is Installed](#). Upload this JAR file and that exported in [Step 1](#) to the **lib** directory on the Linux server. Upload the **conf** configuration file and authentication file in the sample project to the **conf** directory on Linux.
2. In **/opt/test**, create the **run.sh** script, modify the following content, and save the file:

```
#!/bin/sh
BASEDIR=`cd $(dirname $0);pwd`
cd ${BASEDIR}
for file in ${BASEDIR}/lib/*.jar
do
i_cp=${i_cp}:${file}
echo "$file"
done
for file in ${BASEDIR}/conf/*
do
i_cp=${i_cp}:${file}
done

java -cp .${i_cp} com.huawei.bigdata.hbase.examples.TestMain
```

In the preceding content, *com.huawei.bigdata.hbase.examples.TestMain* is used as an example.

**Step 3** Go to **/opt/test** and run the following commands to run the JAR packages:

```
sh run.sh
----End
```

### 12.4.2.3 Viewing Linux Commissioning Results

#### Scenario

After an HBase application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Viewing HBase logs.
- Logging in to the HBase WebUI, see **More Information > External Interfaces > WebUI**.
- Using HBase Shell command, see **More Information > External Interfaces > Shell**.

#### Procedure

The following information is displayed in the running results:

```
2280 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Entering testCreateTable.
3091 [main] WARNcom.huawei.hadoop.hbase.example.HBaseSample- table already exists
3091 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Exiting testCreateTable.
3091 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Entering testPut.
3264 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Put successfully.
3264 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Exiting testPut.
3264 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Entering testGet.
3283 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- 012005000201:info,address,Shenzhen,
Guangdong
3283 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- 012005000201:info,name,yugeZhang
San
3283 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Get data successfully.
3283 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Exiting testGet.
3283 [main] INFOorg.apache.hadoop.hbase.client.ConnectionManager$HConnectionImplementation-
Closing zookeeper sessionId=0xd000035eba278e9
3297 [main] INFOorg.apache.zookeeper.ZooKeeper- Session: 0xd000035eba278e9 closed
3297 [main-EventThread] INFOorg.apache.zookeeper.ClientCnxn- EventThread shut down
-----finish HBase -----
```

## 12.5 More Information

### 12.5.1 SQL Query

#### Function

Phoenix is an intermediate structured query language (SQL) layer built on HBase. Phoenix provides a JDBC driver that can be embedded in a client. The Phoenix query engine converts input SQL statements to one or multiple HBase scans, and compiles and executes the scan tasks to generate a standard JDBC result set.

#### Example Code

- The **hbase-example/conf/hbase-site.xml** file on the client is used to configure the temporary directory for storing query results. If the client

program configures the temporary directory in a Linux environment, configure a Linux path. If the client program configures the temporary directory in a Windows environment, configure a Windows path.

```
<property>
 <name>phoenix.spool.directory</name>
 <value>[1] Temporary directory for storing intermediate query results</value>
</property>
```

- **JAVA Example: Use the JDBC interface to access HBase.**

```
public String getURL(Configuration conf)
{
 String phoenix_jdbc = "jdbc:phoenix";
 String zkQuorum = conf.get("hbase.zookeeper.quorum");
 return phoenix_jdbc + ":" + zkQuorum;
}

public void testSQL()
{
 String tableName = "TEST";
 // Create table
 String createTableSQL = "CREATE TABLE IF NOT EXISTS TEST(id integer not null primary key,
name varchar, account char(6), birth date)";

 // Delete table
 String dropTableSQL = "DROP TABLE TEST";

 // Insert
 String upsertSQL = "UPSERT INTO TEST VALUES(1,'John','100000',
TO_DATE('1980-01-01','yyyy-MM-dd'))";

 // Query
 String querySQL = "SELECT * FROM TEST WHERE id = ?";

 // Create the Configuration instance
 Configuration conf = getConfiguration();

 // Get URL
 String URL = getURL(conf);

 Connection conn = null;
 PreparedStatement preStat = null;
 Statement stat = null;
 ResultSet result = null;

 try
 {
 // Create Connection
 conn = DriverManager.getConnection(URL);
 // Create Statement
 stat = conn.createStatement();
 // Execute Create SQL
 stat.executeUpdate(createTableSQL);
 // Execute Update SQL
 stat.executeUpdate(upsertSQL);
 // Create PreparedStatement
 preStat = conn.prepareStatement(querySQL);
 conn.commit();
 // Execute query
 preStat.setInt(1,1);
 result = preStat.executeQuery();
 // Get result
 while (result.next())
 {
 int id = result.getInt("id");
 String name = result.getString(1);
 }
 }
 catch (Exception e)
 {
```

```

 // handler exception
 }
 finally
 {
 if(null != result){
 try {
 result.close();
 } catch (Exception e2) {
 // handler exception
 }
 }
 if(null != stat){
 try {
 stat.close();
 } catch (Exception e2) {
 // handler exception
 }
 }
 if(null != conn){
 try {
 conn.close();
 } catch (Exception e2) {
 // handler exception
 }
 }
 }
}
}
}

```

**Precaution**

- You need to configure a temporary directory for storing intermediate query results in **hbase-site.xml**. The size of the query result set is restricted by the directory size.
- Phoenix provides most **java.sql** interfaces and follows the ANSI SQL standard.

**12.5.2 HBase Dual-Read Configuration Items**

This section provides the details of all the configurations required for the HBase dual-read feature.

**HBase Dual-Read Operations**

**Table 12-6** Configuration items in hbase-dual.xml

Configuration Item	Description	Default Value	Level
hbase.dualclient.active.cluster.configuration.path	HBase client configuration directory of the active cluster	None	Mandatory
hbase.dualclient.standby.cluster.configuration.path	HBase client configuration directory of the standby cluster	None	Mandatory

Configuration Item	Description	Default Value	Level
dual.client.schedule.update.table.delay.second	DR table update interval	5	Optional
hbase.dualclient.glitchtimeout.ms	Maximum glitch time can be tolerated in the active cluster	50	Optional
hbase.dualclient.slow.query.timeout.ms	Slow query alarm log	180000	Optional
hbase.dualclient.active.cluster.id	Active cluster ID	ACTIVE	Optional
hbase.dualclient.standby.cluster.id	Standby cluster ID	STANDBY	Optional
hbase.dualclient.active.executor.thread.max	Maximum size of the thread pool for processing requests to the active cluster	100	Optional
hbase.dualclient.active.executor.thread.core	Core size of the thread pool for processing requests to the active cluster	100	Optional
hbase.dualclient.active.executor.queue	Queue size of the thread pool for processing requests to the active cluster	256	Optional
hbase.dualclient.standby.executor.thread.max	Maximum size of the thread pool for processing requests to the standby cluster	100	Optional
hbase.dualclient.standby.executor.thread.core	Core size of the thread pool for processing requests to the standby cluster	100	Optional

Configuration Item	Description	Default Value	Level
hbase.dualclient.standby.executor.queue	Queue size of the thread pool for processing requests to the standby cluster	256	Optional
hbase.dualclient.clear.executor.thread.max	Maximum size of the thread pool for clearing resources	30	Optional
hbase.dualclient.clear.executor.thread.core	Core size of the thread pool for clearing resources	30	Optional
hbase.dualclient.clear.executor.queue	Queue size of the thread pool for clearing resources	Integer. MAX_VALUE	Optional
dual.client.metrics.enable	Whether to print client metric information	true	Optional
dual.client.schedule.metrics.second	Interval for printing client metric information	300	Optional
dual.client.asynchronous.enable	Whether to asynchronously request the active and standby clusters	false	Optional

## Printing Metric Information

Table 12-7 Basic specifications

Metric Name	Description	Log level
total_request_count	Total number of queries in a period	INFO
active_success_count	Number of successful queries in the active cluster in a period	INFO
active_error_count	Number of failed queries in the active cluster in a period	INFO



Metric Name	Description	Log level
active_timeout_count	Number of query timeouts in the active cluster in a period	INFO
standby_success_count	Number of successful queries in the standby cluster in a period	INFO
standby_error_count	Number of failed queries in the standby cluster in a period	INFO
Active Thread pool	Periodically printed information about the thread pool for processing requests to the active cluster	DEBUG
Standby Thread pool	Periodically printed information about the thread pool for processing requests to the standby cluster	DEBUG
Clear Thread pool	Periodically printed information about the thread pool for releasing resources	DEBUG

**Table 12-8** Histogram indicators for **GET**, **BatchGET**, and **SCAN** requests

Metric Name	Description	Log level
averageLatency(ms)	Average latency	INFO
minLatency(ms)	Minimum latency	INFO
maxLatency(ms)	Maximum latency	INFO
95thPercentileLatency(ms)	Maximum latency of 95% requests	INFO
99thPercentileLatency(ms)	Maximum latency of 99% requests	INFO
99.9PercentileLatency(ms)	Maximum latency of 99.9% requests	INFO
99.99PercentileLatency(ms)	Maximum latency of 99.99% requests	INFO

## 12.5.3 External Interfaces

### 12.5.3.1 Shell

You can directly perform operations on HBase using Shell on the server. Versions of Shell interfaces of HBase need to be consistent with those in the open-source community. For details, see <http://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/>.

Methods of running the Shell command:

Go to any directory on the HBase client and run the following command:

#### **hbase shell**

Go to the running mode of the HBase command (also called CLI client connection).

```
hbase(main):001:0>
```

Run the **help** command to obtain the help information of the HBase command parameters.

### Precautions

The **count** command does not support conditional statistics. It supports only full table statistics.

### Command to retrieve HBase replication metrics

All the required metrics will be added for the shell command "status".

- Command to view replication source metrics.

```
hbase(main):019:0> status 'replication', 'source'
```

The output is as follows:

```
version 2.2.3
1 live servers
BLR1000006595:
SOURCE: PeerID=1, SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0,
LogReadInBytes=1389, LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:44:42 CST 2016, Replication Lag=0 PeerID=3,
SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0, LogReadInBytes=1389,
LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:44:42 CST 2016, Replication Lag=0,
FailedReplicationAttempts=0
```

- Command to view replication sink metrics.

```
hbase(main):020:0> status 'replication', 'sink'
```

The output is as follows:

```
version 2.2.3
1 live servers
BLR1000006595:
SINK : AppliedBatches=0, AppliedOps=0, AppliedHFiles=0, AgeOfLastAppliedOp=0,
TimeStampsOfLastAppliedOp=Wed May 25 17:55:21 CST 2016
```

- Command to view both replication source and replication sink metrics.

***hbase(main):018:0> status 'replication'***

The output is as follows:

```
version 2.2.3
1 live servers
BLR1000006595:
SOURCE: PeerID=1, SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0,
LogReadInBytes=1389, LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:43:24 CST 2016, Replication Lag=0 PeerID=3,
SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0, LogReadInBytes=1389,
LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:43:24 CST 2016, Replication Lag=0,
FailedReplicationAttempts=0
SINK : AppliedBatches=0, AppliedOps=0, AppliedHFiles=0, AgeOfLastAppliedOp=0,
TimeStampsOfLastAppliedOp=Wed May 25 17:55:21 CST 2016
```

### 12.5.3.2 Java API

#### API Usage Suggestions

- **org.apache.hadoop.hbase.Cell** rather than **org.apache.hadoop.hbase.KeyValue** is recommended as the KV data object.
- It is recommended that **Connection connection = ConnectionFactory.createConnection(conf)** be used to create a connection. The HTablePool is abandoned.
- **org.apache.hadoop.hbase.mapreduce** rather than **org.apache.hadoop.hbase.mapred** is recommended.
- You are advised to obtain the HBase client operation object using the `getAdmin()` method of the constructed Connection object.

#### Common HBase APIs

Common HBase Java classes are as follows:

- Interface class Admin: It is the core class of HBase client applications, which encapsulates APIs for HBase management, such as table creation and table deletion. For details, see [Table 12-9](#).
- Interface class Table: It is a class for HBase read/write, which encapsulates APIs for reading and writing HBase tables. For details, see [Table 12-10](#).

**Table 12-9** org.apache.hadoop.hbase.client.Admin

Method	Description
boolean tableExists(final TableName tableName)	This method is used to check whether a specified table exists. If the table exists in the <b>hbase:meta</b> table, <b>true</b> is returned. Otherwise, <b>false</b> is returned.

Method	Description
HTableDescriptor[] listTables(String regex)	This method is used to view the user table that complies with the specified regular expression format. There are two other overload methods, one with the input parameter type of Pattern, and the other with the empty input parameter. When the input parameter is empty, all user tables are queried by default.
HTableDescriptor[] listTables(final Pattern pattern, final boolean includeSysTables)	The function of this method is similar to that of the previous method. Users can use this method to specify whether the returned result contains the system table. The previous method returns only the user table.
TableName[] listTableNames(String regex)	This method is used to view the user table that complies with the specified regular expression format. There are two other overload methods, one with the input parameter type of Pattern, and the other with the empty input parameter. When the input parameter is empty, all user tables are queried by default. The function of this method is similar to that of listTables. The only difference is that this method returns TableName[].
TableName[] listTableNames(final Pattern pattern, final boolean includeSysTables)	The function of this method is similar to that of the previous method. Users can use this method to specify whether the returned result contains the system table. The previous method returns only the user table.
void createTable(HTableDescriptor desc)	This method is used to create a table with only one region.
void createTable(HTableDescriptor desc, byte[] startKey, byte[] endKey, int numRegions)	This method is used to create a table with a specified number of regions. The endKey of the first region is the startKey, and the StartKey of the last region is the endKey. If there are a large number of regions, the invoking of this method may time out.

Method	Description
void createTable(final HTableDescriptor desc, byte[][] splitKeys)	This method is used to create a table. The number of regions in the table and the startKey of each region are determined by splitKeys. If there are a large number of regions, the invoking of this method may time out.
void createTable(final HTableDescriptor desc, final byte[][] splitKeys)	This method is used to create a table. The number of regions in the table and the startKey of each region are determined by splitKeys. This method uses asynchronous invoking, and does not wait for the created table to be online.
void deleteTable(final TableName tableName)	This method is used to delete a specified table.
public void truncateTable(final TableName tableName, final boolean preserveSplits)	This method is used to re-create a specified table. If the second parameter is set to <b>true</b> , the region of the reconstructed table is the same as that of the previous table. Otherwise, there is only one region.
void enableTable(final TableName tableName)	This method is used to enable a specified table. If there are a large number of regions in a table, the invoking of this method may time out.
void enableTableAsync(final TableName tableName)	This method is used to enable a specified table. This method uses asynchronous invoking, and does not wait for all regions to be online.
void disableTable(final TableName tableName)	This method is used to disable a specified table. If there are a large number of regions in a table, the invoking of this method may time out.
void disableTableAsync(final TableName tableName)	This method is used to disable a specified table. This method uses asynchronous invoking, and does not wait for all regions to be offline.
boolean isTableEnabled(TableName tableName)	This method is used to check whether a table is enabled. This method can be used with the enableTableAsync method to check whether the operation of enabling a table is complete.

Method	Description
boolean isTableDisabled(Table Name tableName)	This method is used to check whether a table is disabled. This method can be used with the disableTableAsync method to check whether the operation of disabling a table is complete.
void addColumn(final TableName tableName, final HColumnDescriptor column)	This method is used to add a column family to a specified table.
void deleteColumn(final TableName tableName, final HColumnDescriptor column)	This method is used to delete a specified column family from a specified table.
void modifyColumn(final TableName tableName, final HColumnDescriptor column)	This method is used to modify a specified column family.

**Table 12-10** org.apache.hadoop.hbase.client.Table

Method	Description
boolean exists(Get get)	This method is used to check whether the specified rowkey exists in the table.
boolean[] existsAll(List<Get> gets)	This method is used to check whether the specified rowkey exists in the table. The returned Boolean array result corresponds to the input parameter position.
Result get(Get get)	This method is used to read data based on the specified rowkey.
Result[] get(List<Get> gets)	This method is used to read data in batches by specifying a batch of rowkeys.
ResultScanner getScanner(Scan scan)	This method is used to obtain a scanner object in the table. The query parameters can be specified by the input parameter scan, including <b>StartRow</b> , <b>StopRow</b> , and <b>caching</b> .
void put(Put put)	This method is used to write a data record to the table.
void put(List<Put> puts)	This method is used to write a batch of data records to the table.

Method	Description
void close()	This method is used to release the resources held by the table object.

 NOTE

The table [org.apache.hadoop.hbase.client.Admin](#) and [org.apache.hadoop.hbase.client.Table](#) list only some common methods.

### 12.5.3.3 SQLLine

You can run `sqlline.py` on the server to perform SQL operations on HBase. The SQLLine APIs of Phoenix are the same as those of the open-source community. For details, see <http://phoenix.apache.org/>.

**Table 12-11** lists common SQLLine syntax, **Table 12-12** lists common functions, and **Phoenix Command Line** describes how to use commands.

**Table 12-11** Common syntax

Command	Description	Example
<b>CREATE TABLE</b>	Creates a table.	CREATE TABLE MY_TABLE(id BIGINT not null primary key, name VARCHAR);
<b>ALTER</b>	Alters a table or a view.	ALTER TABLE MY_TABLE DROP COLUMN name;
<b>DROP TABLE</b>	Deletes a table.	DROP TABLE MY_TABLE;
<b>UPSERT VALUES</b>	Inserts or changes data.	UPSERT INTO MY_TABLE VALUES(1,'abc');
<b>SELECT</b>	Queries data.	SELECT * FROM MY_TABLE;
<b>CREATE INDEX</b>	Creates a global index.	CREATE INDEX MY_IDX ON MY_TABLE(name);
<b>CREATE LOCAL INDEX</b>	Creates a local index.	CREATE LOCAL INDEX MY_LOCAL_IDX ON MY_TABLE(id,name);
<b>ALTER INDEX</b>	Changes the index status.	ALTER INDEX MY_IDX ON MY_TABLE DISABLE;
<b>DROP INDEX</b>	Deletes an index.	DROP INDEX MY_IDX ON MY_TABLE;
<b>EXPLAIN</b>	Displays an execution plan.	EXPLAIN SELECT name FROM MY_TABLE;
<b>CREATE SEQUENCE</b>	Creates a sequence.	CREATE SEQUENCE MY_SEQUENCE;

Command	Description	Example
<b>DROP SEQUENCE</b>	Deletes a sequence.	DROP SEQUENCE MY_SEQUENCE;
<b>CREATE VIEW</b>	Creates a view.	CREATE VIEW MY_VIEW AS SELECT * FROM MY_TABLE;
<b>DROP VIEW</b>	Deletes a view.	DROP VIEW MY_VIEW;
<b>CREATE SCHEMA</b>	Creates a schema.	CREATE SCHEMA MY_SCHEMA;
<b>USE</b>	Modifies the default schema.	USE MY_SCHEMA;
<b>DROP SCHEMA</b>	Deletes a schema.	DROP SCHEMA MY_SCHEMA;

**Table 12-12** Common functions

Function Type	Function	Description	Example
String functions	<b>SUBSTR</b>	Extracts a part of a string.	SUBSTR('Hello', 2, 5)
	<b>INSTR</b>	Queries the position of the first occurrence of a string in another string.	INSTR('Hello World', 'World')
	<b>TRIM</b>	Removes whitespaces from both ends of a string.	TRIM(' Hello ')
	<b>LTRIM</b>	Removes whitespaces found on the left-hand side of the string.	LTRIM(' Hello')
	<b>RTRIM</b>	Removes whitespaces found on the right-hand side of the string.	RTRIM('Hello ')
	<b>LPAD</b>	Left-pads a string with another string.	LPAD('John',30,'*')



Function Type	Function	Description	Example
	<b>LENGTH</b>	Gets the length of a given string.	LENGTH('Hello')
	<b>UPPER</b>	Converts all the characters in a string to uppercase characters.	UPPER('Hello')
	<b>LOWER</b>	Converts all the characters in a string to lowercase characters.	LOWER('HELLO')
	<b>REVERSE</b>	Reverses a string.	REVERSE('Hello')
	<b>REGEXP_SPLIT</b>	Splits a string using a regular expression.	REGEXP_SPLIT('ONE,TWO,THREE', ',')
	<b>REGEXP_REPLACE</b>	Returns a string by applying a regular expression and replacing the matches with the replacement string.	REGEXP_REPLACE('abc123ABC', '[0-9]+', '#')
	<b>REGEXP_SUBSTR</b>	Returns a substring from a string based on a regular expression pattern.	REGEXP_SUBSTR('na1-appsrv35-sj35', '[^-]+')
Aggregate functions	<b>AVG</b>	Gets the average value.	AVG(X)
	<b>COUNT</b>	Gets the number of data records.	COUNT(*)
	<b>MAX</b>	Gets the maximum value.	MAX(NAME)
	<b>MIN</b>	Gets the minimum value.	MIN(NAME)
	<b>SUM</b>	Gets the sum of all values.	SUM(X)

Function Type	Function	Description	Example
	<b>STDDEV_POP</b>	Gets the population standard deviation of all values.	STDDEV_POP( X )
	<b>STDDEV_SAMP</b>	Gets the sample standard deviation of all values.	STDDEV_SAMP( X )
	<b>NTH_VALUE</b>	Gets the Nth value in each distinct group ordered according to the <b>ORDER BY</b> specification.	NTH_VALUE( name, 2 ) WITHIN GROUP (ORDER BY salary DESC)
Time and date functions	<b>NOW</b>	Returns the current date.	NOW()
	<b>CURRENT_TIME</b>	Returns the current time.	CURRENT_TIME()
	<b>CURRENT_DATE</b>	Returns the current date.	CURRENT_DATE()
	<b>TO_DATE</b>	Parses a string and returns a date.	TO_DATE('1970-01-01', 'yyyy-MM-dd', 'GMT+1')
	<b>TO_TIME</b>	Converts the given string into a <b>TIME</b> instance.	TO_TIME('1970-01-01', 'yyyy-MM-dd', 'GMT+1')
	<b>TO_TIMESTAMP</b>	Converts the given string into a <b>TIMESTAMP</b> instance.	TO_TIMESTAMP('1970-01-01', 'yyyy-MM-dd', 'GMT+1')
	<b>YEAR</b>	Returns the year of the specified date.	YEAR(TO_DATE('2015-6-05'))
	<b>MONTH</b>	Returns the month of the specified date.	MONTH(TO_TIMESTAMP('2015-6-05'))
	<b>WEEK</b>	Returns the week of the specified date.	WEEK(TO_TIME('2010-6-15'))

Function Type	Function	Description	Example
	<b>HOUR</b>	Returns the hour of the specified date.	HOUR(TO_TIMESTAMP('2015-6-05'))
	<b>MINUTE</b>	Returns the minute of the specified date.	MINUTE(TO_TIME('2015-6-05'))
	<b>SECOND</b>	Returns the second of the specified date.	SECOND(TO_DATE('2015-6-05'))
Numeric functions	<b>ROUND</b>	Rounds the numeric or timestamp expression to the nearest scale or time unit specified.	ROUND(2.56)
	<b>CEIL</b>	Rounds any fractional value up to the next even multiple.	CEIL(2.34)
	<b>FLOOR</b>	Rounds any fractional value down to the previous even multiple.	FLOOR(2.34)
	<b>TRUNC</b>	Rounds any fractional value down to the previous even multiple (same as <b>FLOOR</b> ).	TRUNC(2.34)
	<b>TO_NUMBER</b>	Formats a string as a number.	TO_NUMBER('-123.33')
	<b>RAND</b>	Returns a random number.	RAND()
Math functions	<b>ABS</b>	Returns the absolute value of the given numeric expression.	ABS(-1)

Function Type	Function	Description	Example
	<b>SQRT</b>	$\sqrt{x}$	SQRT(1.1)
	<b>EXP</b>	$e^x$	EXP(-1)
	<b>POWER</b>	$x^y$	POWER(2, 3)
	<b>LN</b>	$\ln(x)$	LN(3)
	<b>LOG</b>	$\log_x(y)$	LOG(2, 3)
Array functions	<b>ARRAY_ELEM</b>	Uses the array subscript notation to access an array element.	ARRAY_ELEM(ARRAY[1,2,3], 1)
	<b>ARRAY_PREPEND</b>	Appends the given element to the beginning of the array.	ARRAY_APPEND(ARRAY[1,2,3], 4)
	<b>ARRAY_CAT</b>	Concatenates the input arrays and returns the result.	ARRAY_CAT(ARRAY[1,2], ARRAY[3,4])
	<b>ARRAY_FILL</b>	Fills an array with values.	ARRAY_FILL(1, 3)
	<b>ARRAY_TO_STRING</b>	Converts the elements of an array to a string and joins them using the specified delimiter.	ARRAY_TO_STRING(ARRAY['a','b','c'], ',')

Function Type	Function	Description	Example
	<b>ANY</b>	Used on the right-hand side of a comparison expression to test that any array element satisfies the comparison expression against the left-hand side.	10 > ANY(my_array)
	<b>ALL</b>	Used on the right-hand side of a comparison expression to test that all array elements satisfy the comparison expression against the left-hand side.	10 > ALL(my_array)
Other functions	<b>MD5</b>	Computes the MD5 hash of the argument.	MD5(my_column)
	<b>ENCODE</b>	Encodes the expression according to the encoding format provided.	ENCODE(myNumber, 'BASE62')
	<b>DECODE</b>	Decodes the expression according to the encoding format provided.	DECODE('000000008512af277fff8', 'HEX')

### 12.5.3.4 JDBC APIs

Phoenix implements most **java.sql** interfaces. The SQL syntax follows the ANSI SQL standard.

For details about the functions that are supported, visit:

<http://phoenix.apache.org/language/functions.html>

For details about the syntax that is supported, visit:

<http://phoenix.apache.org/language/index.html>

### 12.5.3.5 WebUI

#### Scenario

The web user interface (WebUI) displays the HBase cluster status, including summary of a cluster and information about RegionServer, Master, snapshots, and running processes. You can determine the HBase cluster status based on the information displayed on the WebUI.

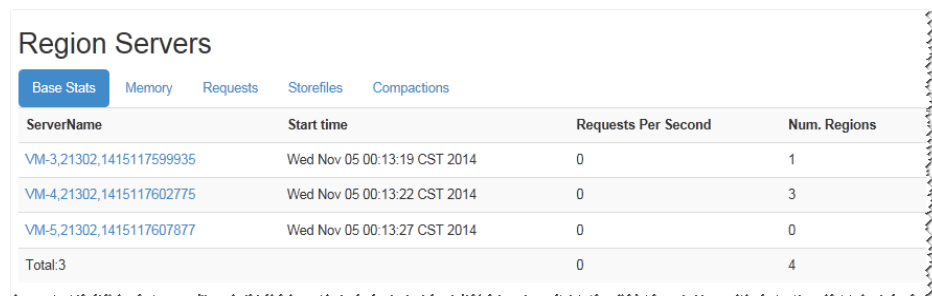
#### NOTE

Contact the administrator to obtain a service account that has the permission to access the WebUI and obtain its password.

#### Procedure

1. **Log in to the FusionInsight Manager portal.** Choose **Cluster > Name of the desired cluster > Services > HBase > HMaster(Active)** to open the HBase WebUI.
2. On the home page of the HBase WebUI, view the HBase summary. The following information is included.
  - a. On the **Region Servers** page, view basic information about the RegionServer, as shown in **Figure 12-25**.

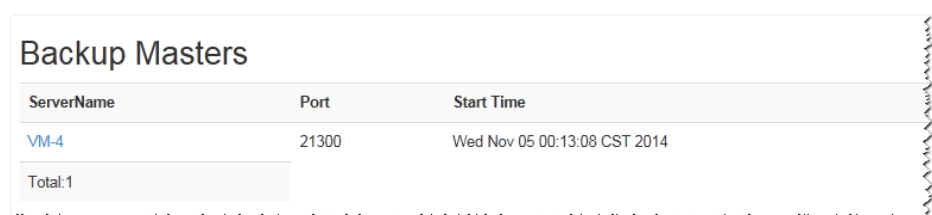
**Figure 12-25** Region Servers



ServerName	Start time	Requests Per Second	Num. Regions
VM-3,21302,1415117599935	Wed Nov 05 00:13:19 CST 2014	0	1
VM-4,21302,1415117602775	Wed Nov 05 00:13:22 CST 2014	0	3
VM-5,21302,1415117607877	Wed Nov 05 00:13:27 CST 2014	0	0
Total:3		0	4

- b. On the **Backup Masters** page, view information about the Backup Master, as shown in **Figure 12-26**.

**Figure 12-26** Backup Masters



ServerName	Port	Start Time
VM-4	21300	Wed Nov 05 00:13:08 CST 2014
Total:1		

- c. On the **Tables** page, view HBase table information, including **User Tables, Catalog Tables, and Snapshots**, as shown in **Figure 12-27**.

Figure 12-27 Tables

Table Name	Online Regions	Description
t	1	t, {NAME => 'd'}

- d. On the **Tasks** page, view information about tasks running on HBase, including the start time and status, as shown in [Figure 12-28](#).

Figure 12-28 Tasks

Start Time	Description	State	Status
Wed Nov 05 03:06:35 CST 2014	Closing region availabilityCheck_VM-5_1415127943,,1415127994683.1d82d088fbf4ba672ee2235fd98ae695.	COMPLETE (since 44sec ago)	Closed (since 44sec ago)
Wed Nov 05 00:20:13 CST 2014	RpcServer.reader=0,port=21300	WAITING (since 2mins, 36sec ago)	Waiting for a call (since 2mins, 36sec ago)
Wed Nov 05 00:19:49 CST 2014	RpcServer.reader=9,port=21300	WAITING (since 2mins, 53sec ago)	Waiting for a call (since 2mins, 53sec ago)
Wed Nov 05 00:19:17 CST 2014	RpcServer.reader=8,port=21300	WAITING (since 3mins, 8sec ago)	Waiting for a call (since 3mins, 8sec ago)
Wed Nov 05 00:15:10 CST 2014	RpcServer.reader=7,port=21300	WAITING (since 3mins, 46sec ago)	Waiting for a call (since 3mins, 46sec ago)
Wed Nov 05 00:14:52 CST 2014	RpcServer.reader=6,port=21300	WAITING (since 5mins, 2sec ago)	Waiting for a call (since 5mins, 2sec ago)
Wed Nov 05 00:14:36 CST 2014	RpcServer.reader=5,port=21300	WAITING (since 10sec ago)	Waiting for a call (since 10sec ago)
Wed Nov 05 00:14:01 CST 2014	RpcServer.reader=4,port=21300	WAITING (since 0sec ago)	Waiting for a call (since 0sec ago)

- 3. On the **Table Details** page of the HBase WebUI, view the summary of HBase storage tables, as shown in [Figure 12-29](#).

Figure 12-29 Table Details

Table	Description
t	t, {NAME => 'd', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}





Figure 12-31 HBase Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
- <configuration>
 - <property>
 <name>dfs.journalnode.rpc-address</name>
 <value>0.0.0.0:8485</value>
 <source>hdfs-default.xml</source>
 </property>
 - <property>
 <name>io.storefile.bloom.block.size</name>
 <value>131072</value>
 <source>hbase-default.xml</source>
 </property>
 - <property>
 <name>hbase.sessioncontrol.maxSessions</name>
 <value>65535</value>
 <source>hbase-site.xml</source>
 </property>
 - <property>
 <name>yarn.ipc.rpc.class</name>
 <value>org.apache.hadoop.yarn.ipc.HadoopYarnProtoRPC</value>
 <source>yarn-default.xml</source>
 </property>
 - <property>
 <name>mapreduce.job.maxtaskfailures.per.tracker</name>
 <value>3</value>
 <source>mapred-default.xml</source>
 </property>
 - <property>
 <name>hbase.rest.threads.min</name>
 <value>2</value>
 <source>hbase-default.xml</source>
 </property>
 - <property>
 <name>hbase.rs.cacheblocksonwrite</name>
 <value>false</value>
 <source>hbase-default.xml</source>
 </property>
 - <property>
 <name>ha.health-monitor.connect-retry-interval.ms</name>
 <value>1000</value>
 <source>core-default.xml</source>
 </property>
 - <property>
 <name>yarn.resourcemanager.work-preserving-recovery.enabled</name>
 <value>false</value>
 <source>yarn-default.xml</source>
 </property>
 - <property>
 <name>dfs.client.mmap.cache.size</name>
 <value>256</value>
 <source>hdfs-default.xml</source>
 </property>
 - <property>
```

## 12.5.4 HBase Access Configuration on Windows Using EIPs

### Scenario

This section describes how to bind Elastic IP addresses (EIPs) to a cluster and configure HBase files so that sample files can be compiled locally.

This section uses hbase-example as an example.

### Procedure

- Step 1** Apply for an EIP for each node in the cluster and add public IP addresses and corresponding host domain names of all nodes to the Windows local **hosts** file. (If a host name contains uppercase letters, change them to lowercase letters.)

1. On the VPC console, apply for EIPs (the number of EIPs you buy should be equal to the number of nodes in the cluster), click the name of each node in the MRS cluster, and bind an EIP to each node on the **EIPs** page.  
For details, see **Virtual Private Cloud > User Guide > EIP > Assigning an EIP and Binding It to an ECS**.
2. Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.

```

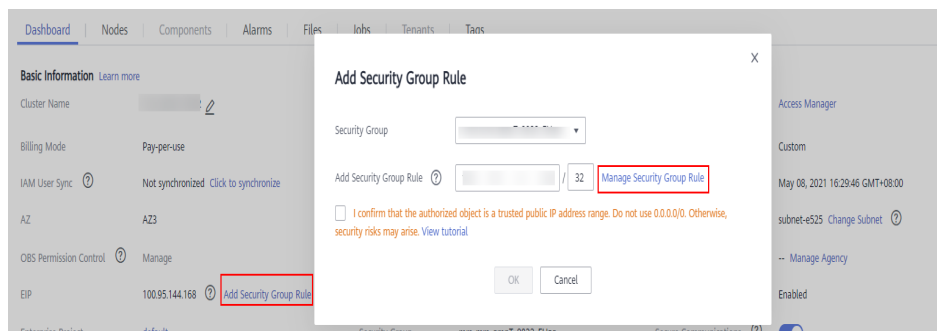
1 Mapping between public IP addresses and private IP addresses
2 100.95.144.168 172.16.0.120
3 100.95.144.168 172.16.0.42
4 100.93.144.168 172.16.0.62
5 100.95.144.168 172.16.0.200
6 100.93.144.168 172.16.0.139
7 100.93.144.168 172.16.0.214
8
9 hosts file in the cluster
10 172.16.0.120 node-group-1XZI00002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI00002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
11 172.16.0.42 node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
12 172.16.0.62 node-group-1XZI00003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI00003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
13 172.16.0.200 node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
14 172.16.0.139 node-group-1XZI00001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI00001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
15 172.16.0.214 node-master2pVNu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVNu.ead64699-185a-4290-bbef-1a07e2f0459b.com.
16
17 hosts file that should be added to Windows
18 100.95.144.168 node-group-1xzi00002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI00002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
19 100.95.144.168 node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
20 100.93.144.168 node-group-1xzi00003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI00003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
21 100.95.144.168 node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
22 100.93.144.168 node-group-1xzi00001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI00001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
23 100.93.144.168 node-master2pVNu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVNu.ead64699-185a-4290-bbef-1a07e2f0459b.com.

```

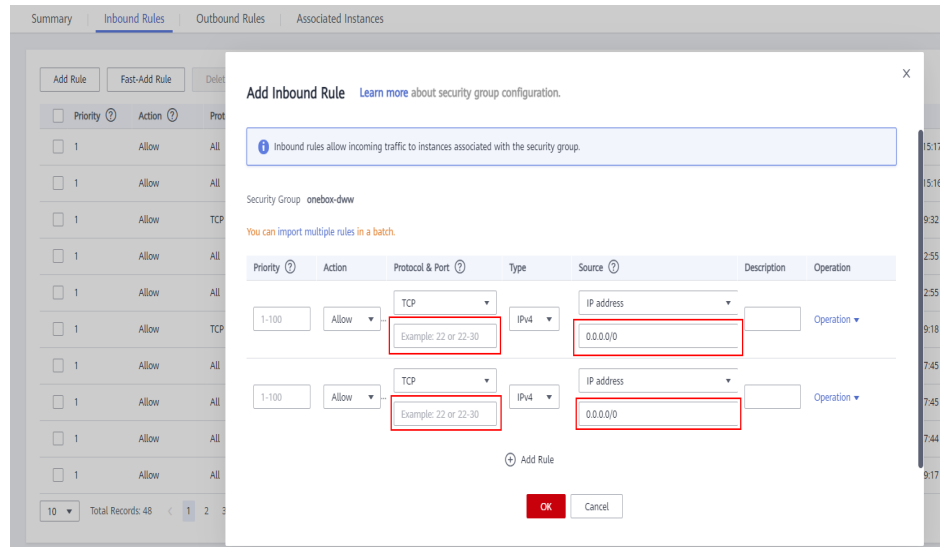
**Step 2** Change the IP addresses in the **krb5.conf** file to the corresponding host names.

**Step 3** Configure security group rules for the cluster.

1. On the **Dashboard** page, choose **Add Security Group Rule > Manage Security Group Rule**.



2. On the **Inbound Rules** page, click **Add Rule**. On the displayed **Add Inbound Rule** page, configure Windows IP addresses and ports **21730TCP**, **21731TCP/UDP**, and **21732TCP/UDP**.



**Step 4** On Manager, choose **Cluster > Services > HBase > More > Download Client**, and copy the **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml** files on the client to the **resources** directory of the sample project.

**Step 5** Before running the sample code, change the value of **userName** in the sample code to the username for security authentication.

----End

## 12.5.5 Phoenix Command Line

Phoenix supports SQL statements to operate HBase. The following describes how to use SQL statements to create tables, insert data, query data, and delete tables.

### Prerequisites

The HBase client has been installed. For example, the client has been installed in the **/opt/client** directory. The client directory in the following operations is only an example. Change it based on the actual installation directory onsite. Before using the client, download and update the client configuration file, and ensure that the active management node of Manager is available.

### Procedure

**Step 1 Optional:** Log in to the node where the client is installed as the client installation user.

Access the HBase client installation directory:

```
cd /opt/client
```

**Step 2** Run the following command to configure environment variables:

```
source bigdata_env
```

**Step 3** If Kerberos authentication is enabled for the current cluster, run the following command to authenticate the current user. The current user must have the permission to create HBase tables. If Kerberos authentication is disabled for the current cluster, skip this step:

**kinit** *MRS cluster user*

For example, **kinit hbaseuser**.

**Step 4** Running Commands on the Phoenix Client.

**sqlline.py**

**Step 5** Create a table:

```
CREATE TABLE TEST (id VARCHAR PRIMARY KEY, name VARCHAR);
```

**Step 6** Insert data:

```
UPSERT INTO TEST(id,name) VALUES ('1','jamee');
```

**Step 7** Query data:

```
SELECT * FROM TEST;
```

**Step 8** Deleting a table:

```
DROP TABLE TEST;
```

**Step 9** Exit the Phoenix CLI:

**!quit**

----End

## 12.5.6 FAQs

### 12.5.6.1 How to Rectify the Fault When an Exception Occurs During the Running of an HBase-developed Application and "org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory" Is Displayed in the Error Information?

**Step 1** Check whether the **hbase.rpc.controllerfactory.class** configuration item is included in the **hbase-site.xml** configuration file.

```
<name>hbase.rpc.controllerfactory.class</name>
<value>org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory</value>
```

**Step 2** If this configuration item is included in the current application development project, you need to import the **phoenix-core-\*.jar** package. You can obtain this package from **HBase/hbase/lib** in the HBase client installation directory.

**Step 3** If you do not import this JAR package, you need to delete **hbase.rpc.controllerfactory.class** from the **hbase-site.xml** configuration file.

----End

### 12.5.6.2 What Are the Application Scenarios of the Bulkload and put Data-loading Modes?

#### Question

Both the bulkload and put data-loading modes can be used to load data to HBase. Though the bulkload mode loads data faster than the put mode, the bulkload mode has its own disadvantages. The following describes the application scenarios of these two data-loading modes.

## Answer

The bulkload starts MapReduce tasks to generate HFile files, and then registers HFile files with HBase. Incorrect use of the bulkload mode will consume more cluster memory and CPU resources due to started MapReduce tasks. The large number of HFile files may frequently trigger Compaction, decreasing the query speed drastically.

Incorrect use of the put mode may cause a slow data loading rate. If the memory allocated to RegionServer is not sufficient, the process may exit.

The application scenarios of the bulkload and put modes are as follows:

- bulkload:
  - Load a large amount of data to HBase in the one-off manner.
  - Load data to HBase with low reliability requirements and without generating WAL files.
  - Low loading and query speed if the put mode is used.
  - The size of the HFile generated after data loading is similar to the size of HDFS block.
- put:
  - The size of the data loaded to one Region at a time is smaller than half the size of an HDFS block.
  - Load data to HBase in real time.
  - The query speed does not decrease wildly during data loading.

### 12.5.6.3 An Error Occurred When Building a JAR Package

#### Question

When the sample code is compiled using Maven to build a JAR package, the error message "Could not transfer artifact org.apache.commons:commons-crypto:pom:\${commons-crypto.version}" is displayed and the package building fails.

#### Answer

The hbase-common module depends on commons-crypto. In the **pom.xml** file of hbase-common, the **`\${commons-crypto.version}`** variable is used to introduce commons-crypto. The parsing logic of this variable is as follows: If the OS is AArch64, the value is **1.0.0-hw-aarch64**. If the OS is x86\_64, the value is **1.0.0**. If Maven fails to parse the variable using the OS due to incorrect configuration in the compilation environment, you can manually modify the **pom.xml** file to prevent correct compilation.

In the **pom.xml** file, manually change the dependency of the hbase-common module to the following to exclude the commons-crypto dependency:

```
<dependency>
 <groupId>org.apache.hbase</groupId>
 <artifactId>hbase-common</artifactId>
 <version>${hbase.version}</version>
 <exclusions>
 <exclusion>
 <groupId>org.apache.commons</groupId>
```

```
<artifactId>commons-crypto</artifactId>
</exclusion>
</exclusions>
</dependency>
```

Manually add the commons-crypto dependency of the specified version. Enter a correct version based on the OS architecture (x86\_64 or AArch64).

```
<dependency>
<groupId>org.apache.commons</groupId>
<artifactId>commons-crypto</artifactId>
<version>1.0.0</version>
</dependency>
```

# 13 HBase Development Guide (Normal Mode)

---

## 13.1 Overview

### 13.1.1 Application Development Overview

#### HBase Introduction

HBase is a column-oriented scalable distributed storage system featuring high reliability and high performance. HBase is designed to break through the limitation when relational databases are used to process massive data.

HBase applies to the following application scenarios:

- Massive data processing (higher than the TB or PB level).
- Scenarios that require high throughput.
- Scenarios that require efficient random read of massive data.
- Scenarios that require good scalability.
- Structured and unstructured data is concurrently processed.
- The Atomicity, Consistency, Isolation, Durability (ACID) feature supported by traditional relational databases is not required.
- HBase tables provide the following features:
  - Large: One table contains a hundred million rows and one million columns.
  - Column-oriented: Storage and rights control is implemented based on columns (families), and columns (families) are independently retrieved.
  - Sparse: Null columns do not occupy storage space, so a table is sparse.

#### Interface Type Introduction

The Java language is recommended for HBase application development because HBase is developed based on Java and Java is a concise, universal, and easy-to-understand language.

HBase adopts the same interfaces as those of Apache HBase.

**Table 13-1** describes the functions that HBase can provide by invoking interfaces.

**Table 13-1** Functions provided by HBase interfaces

Function	Description
Data CRUD function	Data creating, retrieving, updating, and deleting
Advanced feature	Filter, secondary index, and coprocessor
Management function	Table management and cluster management

## 13.1.2 Common Concepts

- Filter**  
 Filters provide powerful features to help users improve the table data processing efficiency of HBase. Users can use the filters predefined in HBase and customized filters.
- Coprocessor**  
 Coprocessors enable users to perform region-level operations and provide functions similar to those of triggers in relational database management systems (RDBMSs).
- Client**  
 Users can access the server from the client through the Java API, HBase Shell or WebUI to read and write HBase tables. The HBase client in this document indicates the HBase client installation package, see [External Interfaces](#).

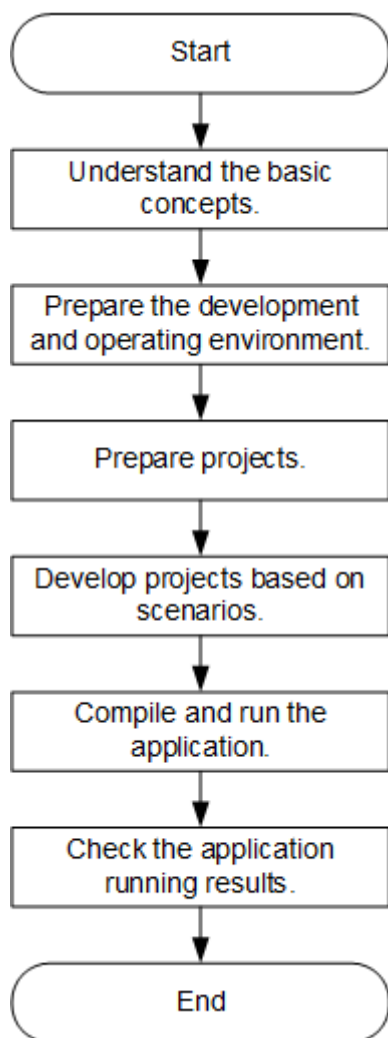
## 13.1.3 Development Process

This document describes HBase application development based on the Java API.

For information about each phase in the development process, see [Figure 13-1](#) and [Table 13-2](#).



**Figure 13-1** HBase application development process



**Table 13-2** HBase application development process description

Phase	Description	Reference Document
Understand the basic concepts.	Before developing an application, learn basic concepts of HBase and understand the scenario requirements and design tables.	<a href="#">Common Concepts</a>

Phase	Description	Reference Document
Prepare the development and operating environment.	The Java language is recommended for the development of HBase applications. The IntelliJ IDEA tool can be used.  The HBase running environment is the HBase client. Install and configure the client according to the guide.	<a href="#">Preparing for Development and Operating Environment</a>
Prepare projects.	HBase provides example projects for different scenarios. You can import an example project to learn the application.	<a href="#">Configuring and Importing Sample Projects</a>
Develop projects based on scenarios.	An example project based on the Java language is provided, including creating a table, writing data into the table, and deleting the table.	<a href="#">Developing an Application</a>
Compile and run the application.	Compile the developed application and submit it for running.	<a href="#">Application Commissioning</a>
View application running results.	Application running results are written into a specified path. You can also view the application running status on the UI.	<a href="#">Application Commissioning</a>

## 13.2 Environment Preparation

### 13.2.1 Preparing for Development and Operating Environment

#### Preparing the Development Environment

[Table 13-3](#) describes the environment required for secondary development.

**Table 13-3** Development environment

Item	Description
OS	<ul style="list-style-type: none"> <li>Development environment: Windows OS. Windows 7 or later is supported.</li> <li>Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li> </ul>
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> <li>For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> <li>Oracle JDK versions: 1.8</li> <li>Supported IBM JDK versions: 1.8.5.11</li> </ul> </li> <li>For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> <li>OpenJDK: 1.8.0_272</li> </ul> </li> </ul> <p><b>NOTE</b> The server supports only TLS V1.2 or later to ensure security. By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls">https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls</a>.</p>
IntelliJ IDEA installation and configuration	<p>Tool used for developing HBase applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li> <li>If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li> <li>If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li> <li>Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li> </ul>
JUnit plug-in installation	Basic configuration for the development environment
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.

Item	Description
7-zip	Used to decompress <b>.zip</b> and <b>.rar</b> packages. 7-Zip 16.04 is supported.

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. **Log in to the FusionInsight Manager portal** and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Configuration Files Only**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles\HBase\config**, obtain the **HBase-related configuration file**. The configuration file will be imported to the configuration file directory (usually the **conf** folder) of the HBase sample project.
  - c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, the client installation directory is **/opt/client**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. **Log in to the FusionInsight Manager portal.** Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig\HBase\config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/client/conf**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp HBase/config/* root@IP address of the client node:/opt/client/conf
```

- c. Check the network connection of the client node.  
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

## 13.2.2 Configuring and Importing Sample Projects

### Background

Obtain the HBase development example project . Import the project to IntelliJ IDEA for learning.

### Prerequisites

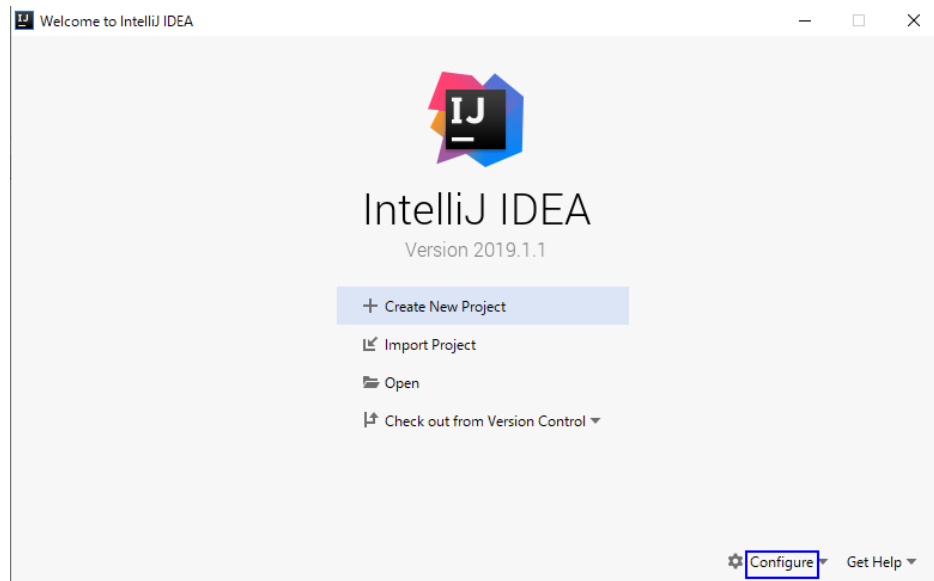
Ensure that the difference between the local PC time and the MRS cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the MRS cluster time in the bottom-right corner on FusionInsight Manager.

### Procedure

- Step 1** Obtain the sample project folder **hbase-example** in the **src/hbase-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

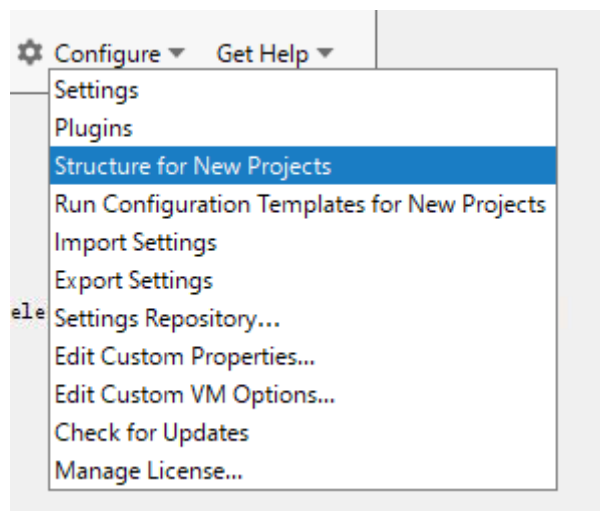
- Step 2** The cluster configuration file obtained in section [Preparing an Operating Environment](#) to the `hbase-example\src\main\resources\conf` directory of the sample project. For details about how to place configuration files and precautions for executing the sample code for other sample projects, see the `README.md` file of the corresponding sample project.
- Step 3** After the IntelliJ IDEA and JDK are installed, configure the JDK in IntelliJ IDEA.
1. Start the IntelliJ IDEA and click **Configure**.

**Figure 13-2** Quick Start



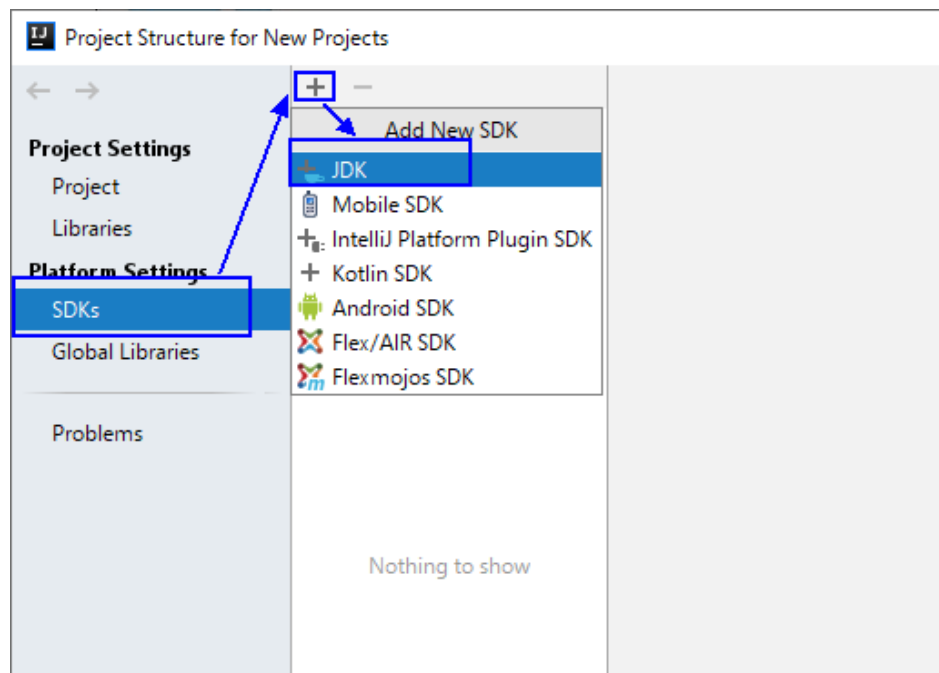
2. Click **Structure for New Projects** from the drop-down list.

**Figure 13-3** Configure



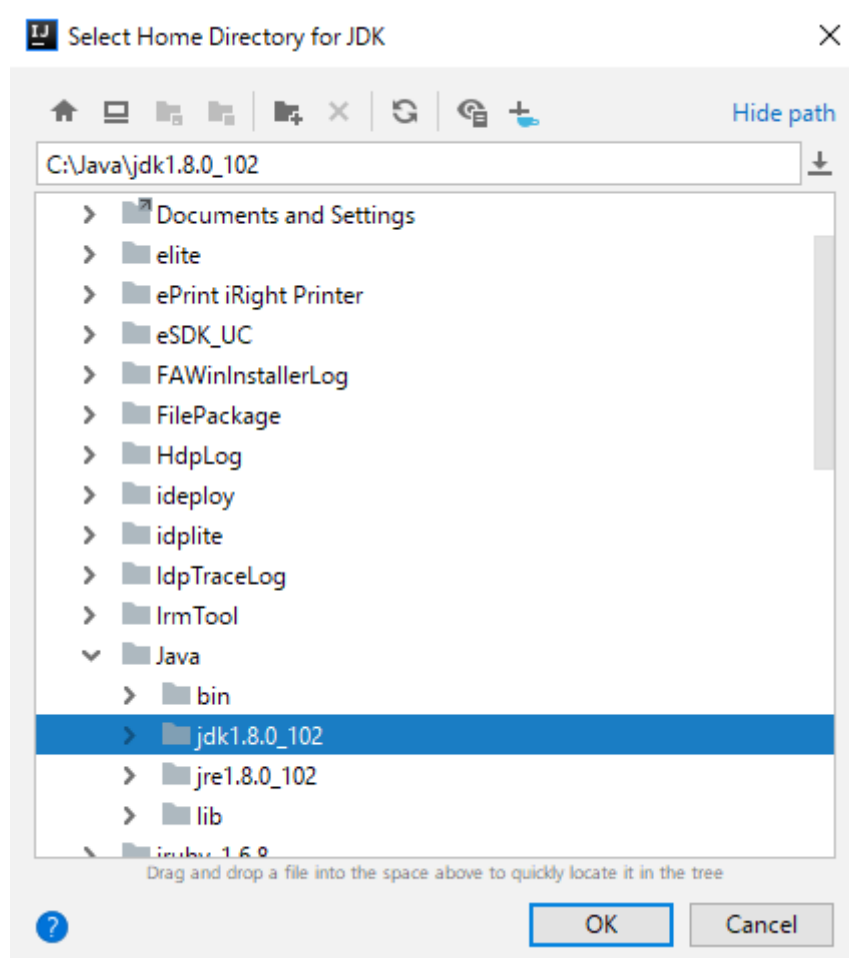
3. On the displayed **Project Structure for New Projects** page, select **SDKs** and click the plus sign (+) to add the **JDK**.

**Figure 13-4** Project Structure for New Projects



4. In the **Select Home Directory for JDK** window that is displayed, select the JDK directory, and click **OK**.

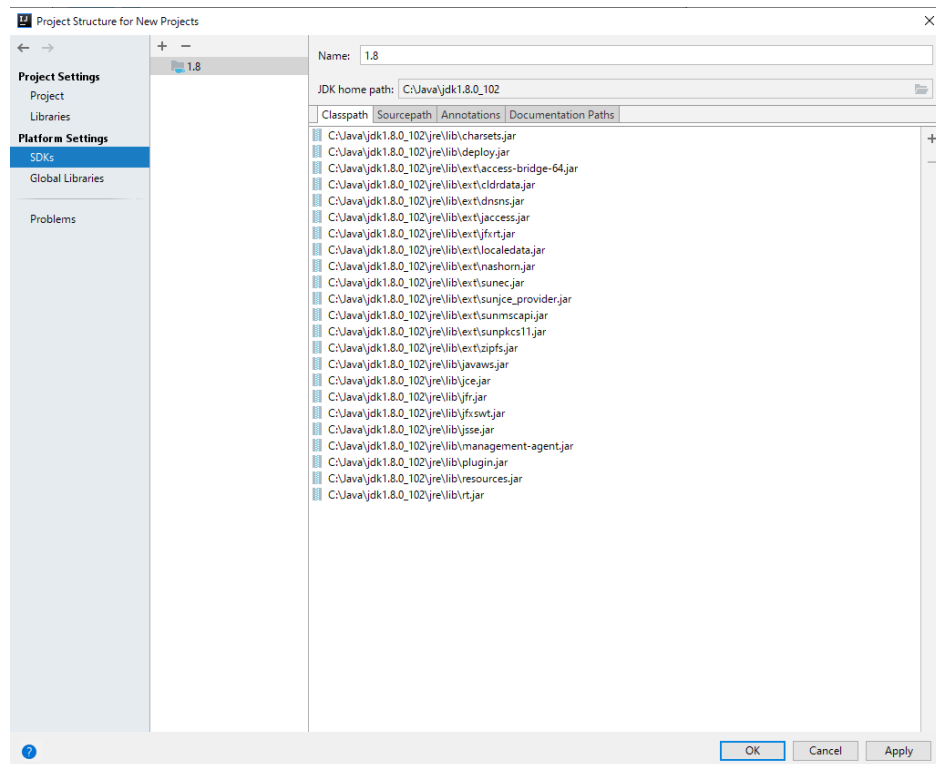
**Figure 13-5** Select Home Directory for JDK



5. After selecting the JDK, click **OK** to complete the configuration.



Figure 13-6 Complete JDK configuration



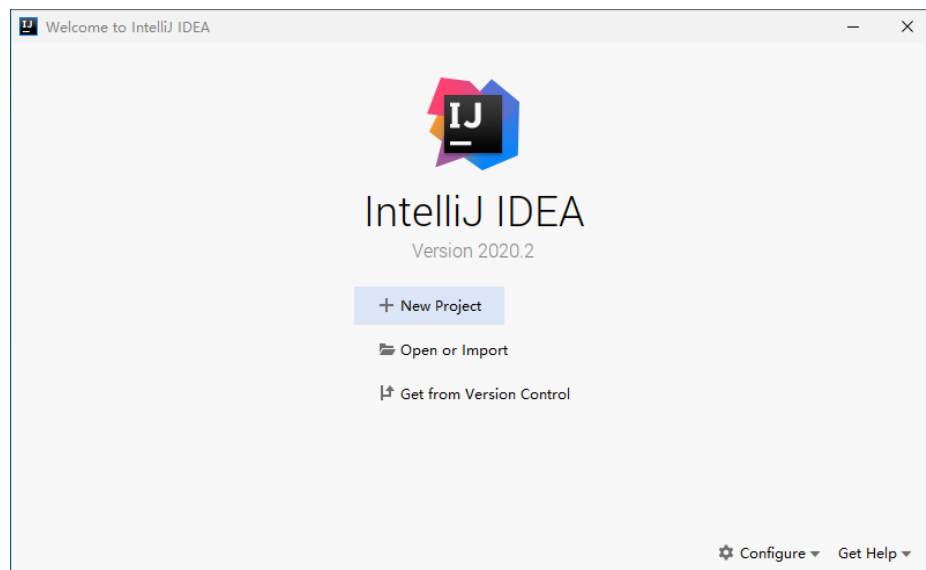
**NOTE**

The operations vary by the IDEA version. The operation procedure varies according to the actual version.

**Step 4** Import the example project to the IntelliJ IDEA development environment.

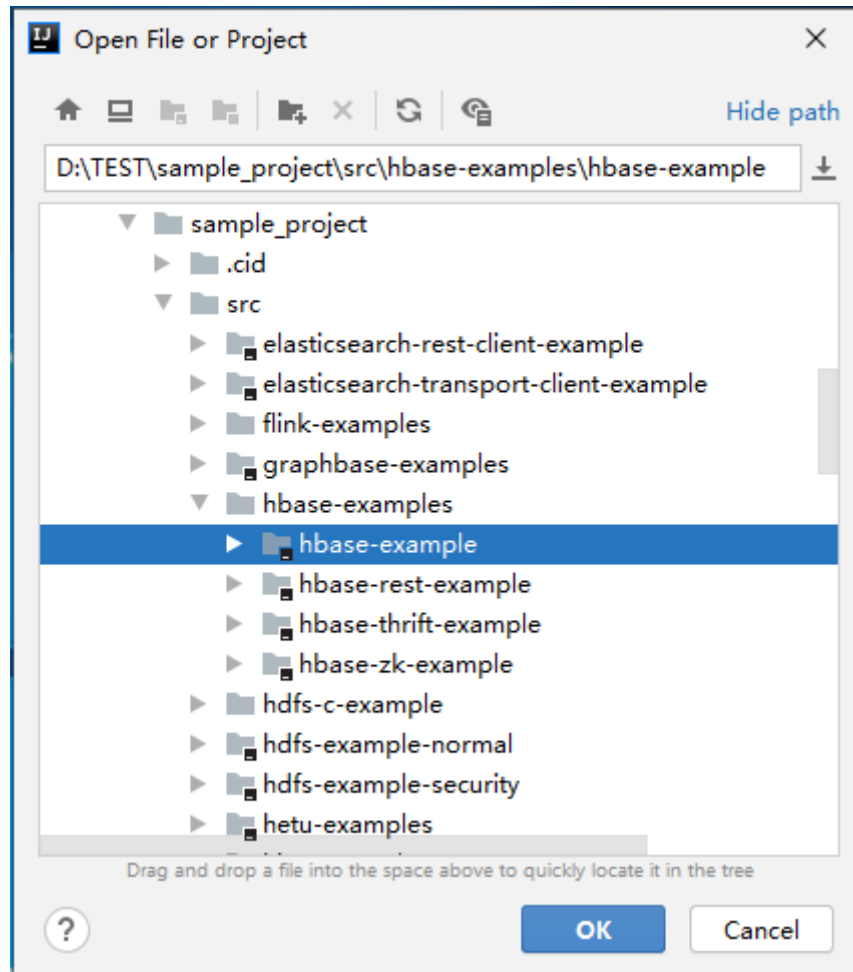
1. Start the IntelliJ IDEA, and click **Import Project** on the Open or Import. For the IDEA tool that has been used, you can choose **File > Import project...** on the main interface to import the sample project.

Figure 13-7 Open or Import (Quick Start page)



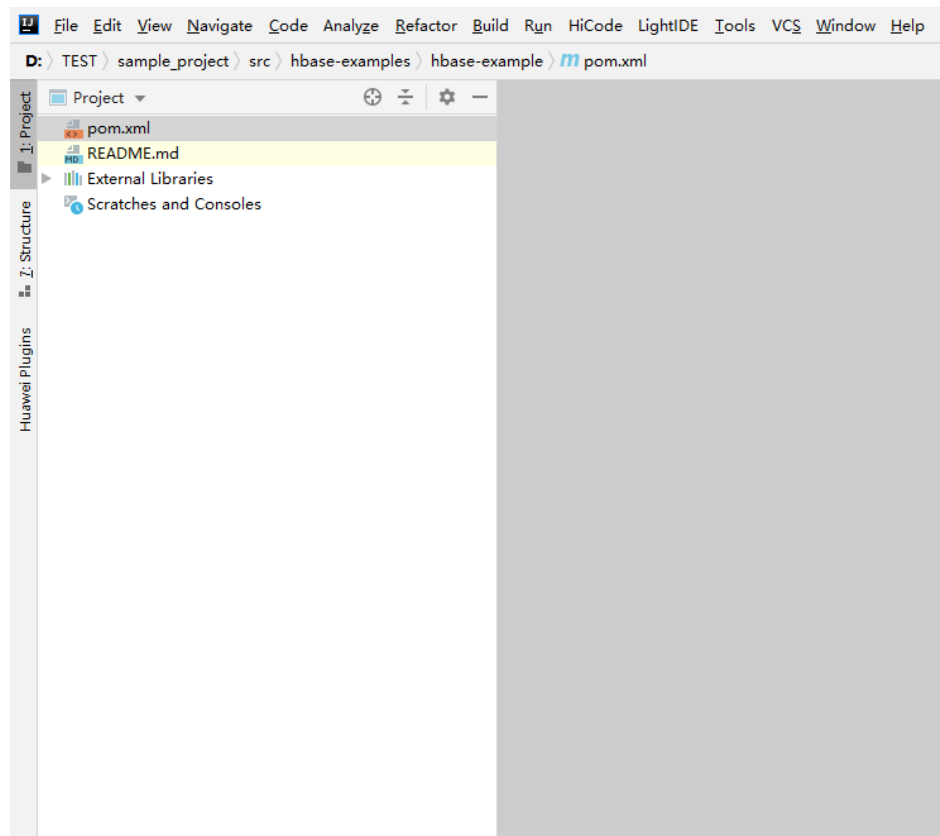
2. Select the example project folder **hbase-example**, and click **OK**.

**Figure 13-8** Select File or Directory to Import



3. After the import, the imported sample project is displayed on the IDEA home page.

**Figure 13-9** Successfully importing the sample project



4. Right-click **pom.xml** and choose **Add as Maven Project** from the shortcut menu to add the project as a Maven Project. If the **pom.xml** icon is as shown in [Figure 13-10](#), go to the next step.

**Figure 13-10** Sample project imported as a maven project

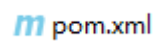
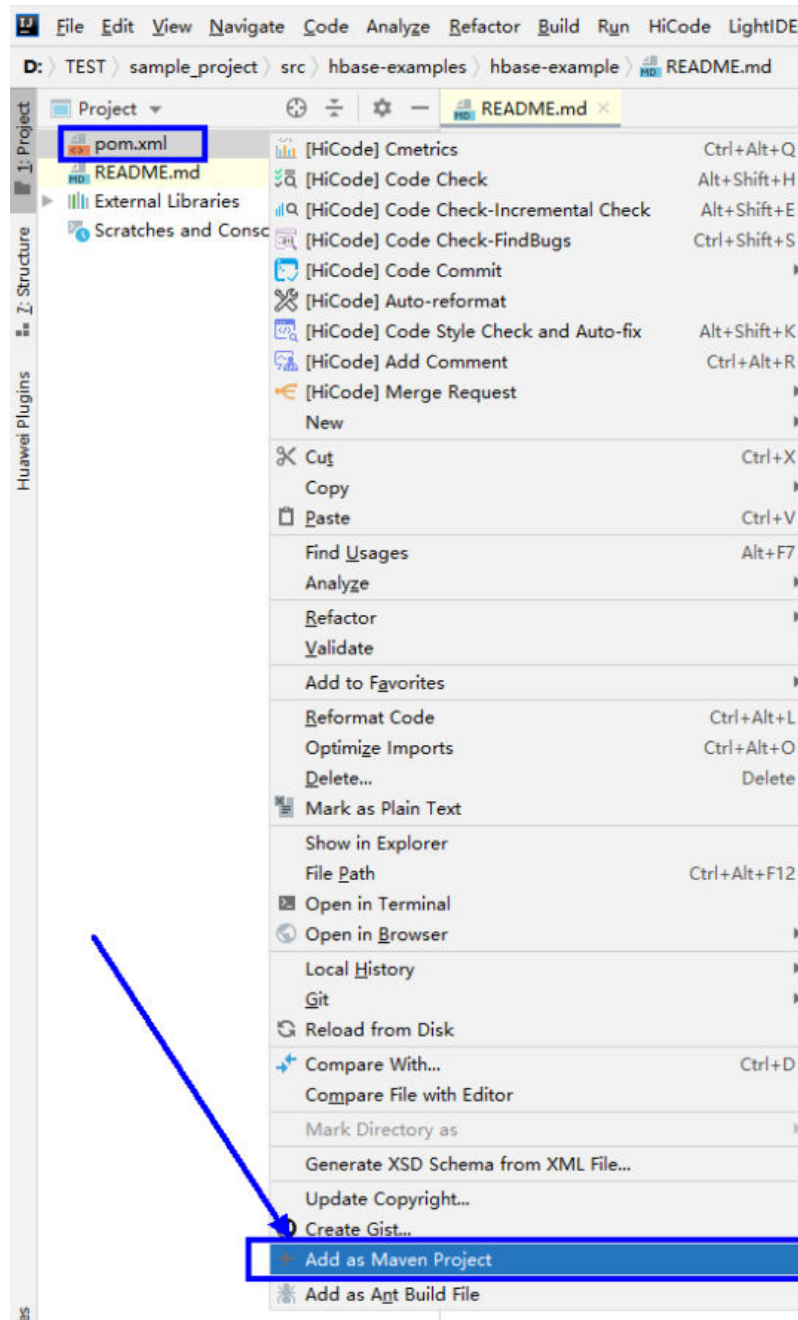
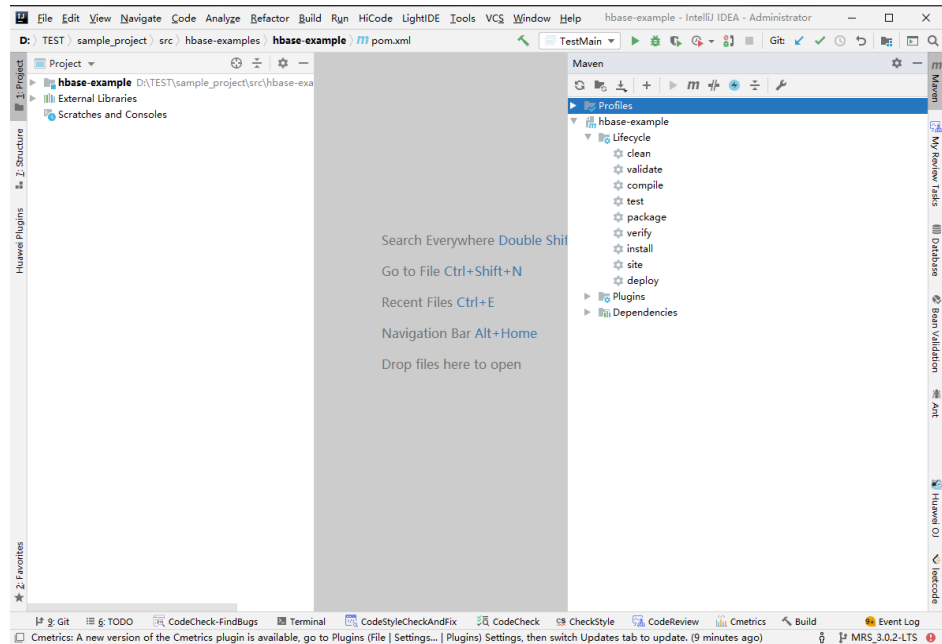


Figure 13-11 Add as Maven Project



In this case, the IDEA can identify the project as a Maven project.

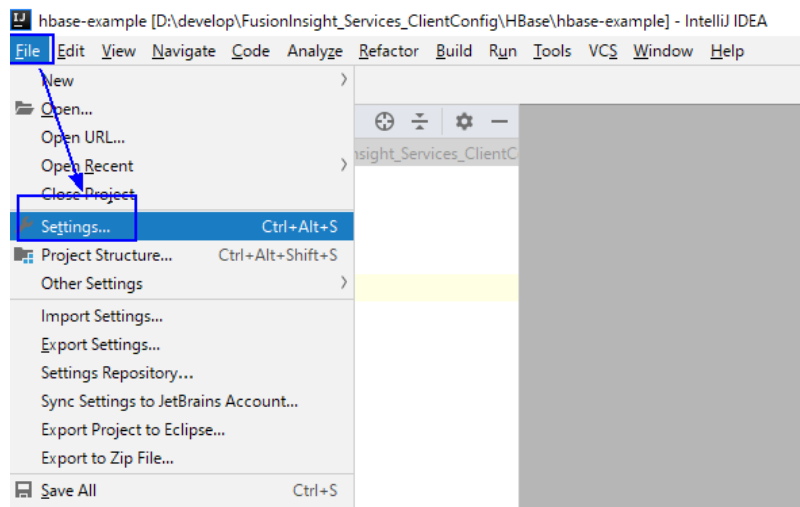
Figure 13-12 Sample project displayed in IDEA as a Maven project



**Step 5** Set the Maven version used by the project.

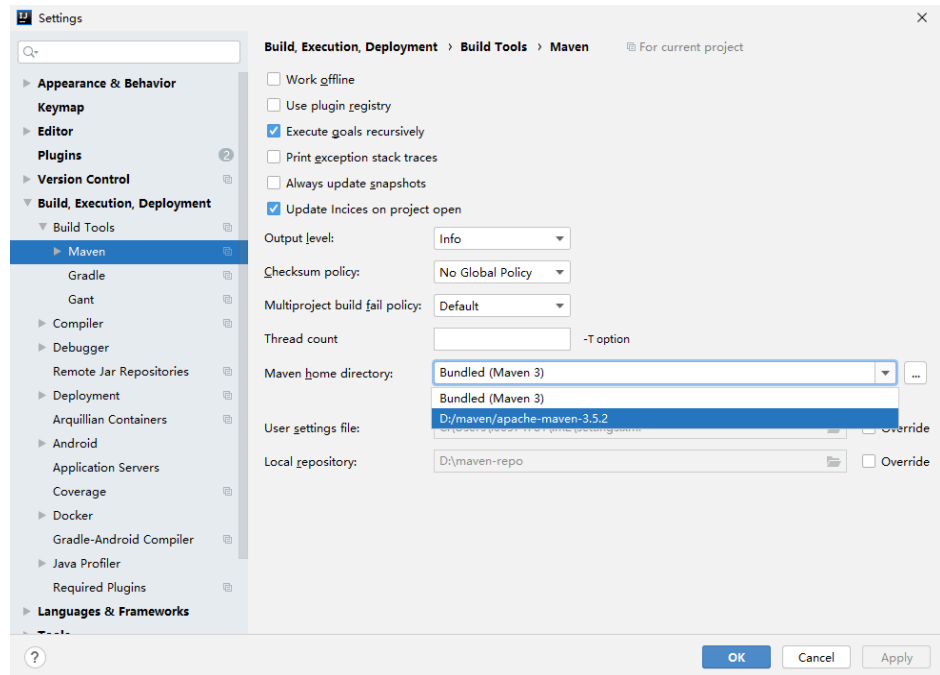
1. Choose **File > Settings...** from the main menu of IntelliJ IDEA.

Figure 13-13 Settings



2. Choose **Build, Execution, Deployment > Maven** and set **Maven home directory** to the Maven version installed on the local host.  
Set **User settings file** and **Local repository** parameters based on the site requirements, and choose **Apply > OK**.

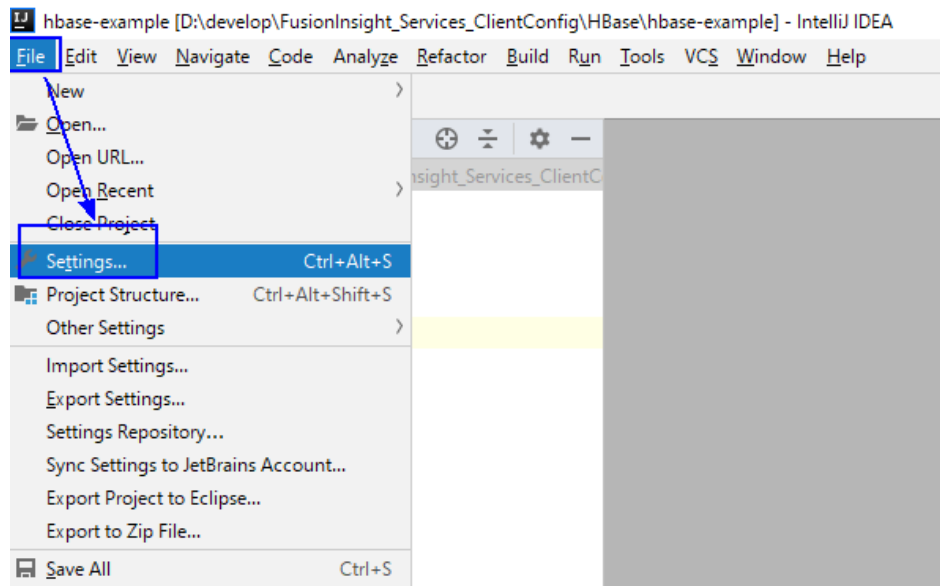
**Figure 13-14** Selecting the local Maven installation directory



**Step 6** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

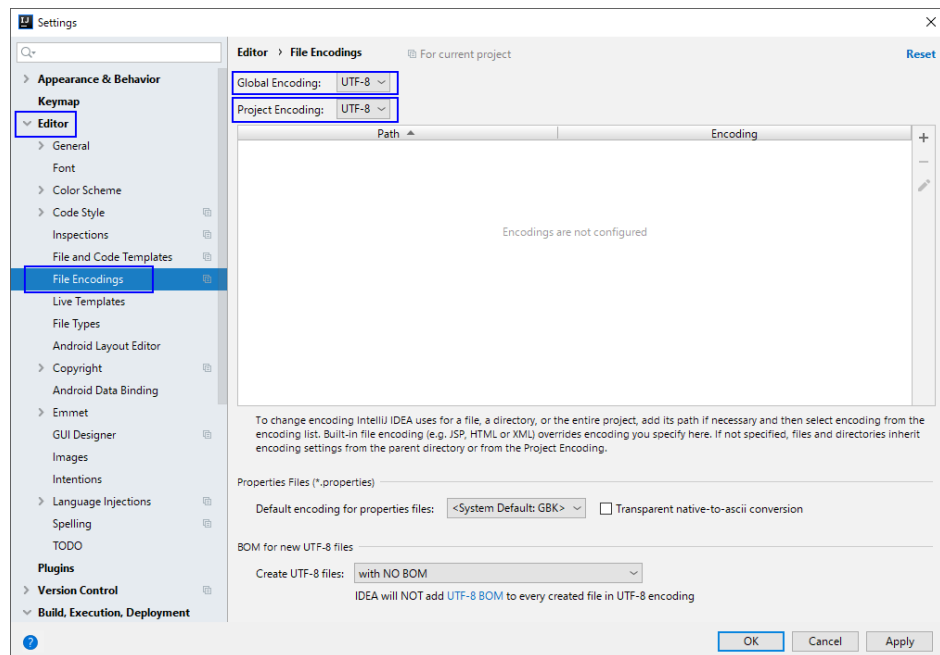
1. On the IntelliJ IDEA menu bar, choose **File > Settings...**

**Figure 13-15** Settings



2. In the navigation tree of the **Settings** page, choose **Editor > File Encodings**, and select **UTF-8** for **Global Encoding** and **Project Encoding**.

Figure 13-16 File Encodings



3. Click **Apply** and **OK** to complete the encoding configuration.

----End

## 13.3 Developing an Application

### 13.3.1 Typical Scenario Description

Based on typical scenarios, users can quickly learn and master the HBase development process and know important interface functions.

#### Scenario

Develop an application to manage information about users who use service A in an enterprise. [Table 13-4](#) provides the user information. The operation process of service A is described as follows:

- Create a user information table.
- Add diplomas and titles to the user information table.
- Query user names and addresses by user ID.
- Query information by user name.
- Query information about users whose ages range from 20 to 29.
- Collect statistics on the number and the maximum, minimum, and average ages of users.
- Deregister users, and delete user data.
- Delete the user information table after service A ends.

**Table 13-4** User information

ID	Name	Gender	Age	Address
1200500020 1	Zhang San	Male	19	Shenzhen, Guangdong
1200500020 2	Li Wanting	Female	23	Shijiazhuang, Hebei
1200500020 3	Wang Ming	Male	26	Ningbo, Zhejiang
1200500020 4	Li Gang	Male	18	Xiangyang, Hubei
1200500020 5	Zhao Enru	Female	21	Shangrao, Jiangxi
1200500020 6	Chen Long	Male	32	Zhuzhou, Hunan
1200500020 7	Zhou Wei	Female	29	Nanyang, Henan
1200500020 8	Yang Yiwen	Female	30	Kaixian, Chongqing
1200500020 9	Xu Bing	Male	26	Weinan, Shaanxi
1200500021 0	Xiao Kai	Male	25	Dalian, Liaoning

## Data Planning

Proper design of the table structure, RowKeys, and column names can give full play to the advantages of HBase. In this example, the unique ID is used as the RowKey, and columns are stored in the **info** column family.

## 13.3.2 Development Idea

### Function Decomposition

Functions are decomposed based on the preceding service scenario. [Table 13-5](#) provides the functions to be developed.

**Table 13-5** Functions to be developed in HBase

No.	Procedure	Code Implementation
1	Create a table based on <a href="#">Table 13-4</a> .	For details, see <a href="#">Creating a Table</a> .



No.	Procedure	Code Implementation
2	Import user data.	For details, see <a href="#">Inserting Data</a> .
3	Add the <b>Education</b> column family, and add diplomas and titles to the user information table.	For details, see <a href="#">Modifying a Table</a> .
4	Query user names and addresses by user ID.	For details, see <a href="#">Reading Data Using Get</a> .
5	Query information by user name.	For details, see <a href="#">Filtering Data</a> .
6	To improve query performance, create or delete secondary indexes.	For details, see <a href="#">Creating a Secondary Index</a> and <a href="#">Secondary Index-based Query</a> .
7	Deregister users, and delete user data.	For details, see <a href="#">Deleting Data</a> .
8	Delete the user information table after service A ends.	For details, see <a href="#">Deleting a Table</a> .

## Key Design Principle

HBase is a distributed database system based on the lexicographic order of RowKeys. The RowKey design has great impact on performance, so the RowKeys must be designed based on specific services.

## 13.3.3 Example Code Description

### 13.3.3.1 Configuring Log4j Log Output

#### Function Description

This section describes how to output HBase client logs to a specified log file separately from service logs to facilitate HBase problem analyzing and locating. If the **log4j** configuration exists in the process, copy the RFA and RFAS configurations in **hbase-example\src\main\resources\log4j.properties** to the existing **log4j** configuration.

#### Sample Code

```
hbase.root.logger=INFO,console,RFA //HBase client log output configuration. console: outputs to
the console; RFA: outputs to the log files.
hbase.security.logger=DEBUG,console,RFAS //HBase client security logs output configuration. console:
outputs to the console; RFAS: outputs the log files.
hbase.log.dir=/var/log/Bigdata/hbase/client/ //Log directory. Modify based on the actual directory. Ensure
that the directory has the write permission.
hbase.log.file=hbase-client.log //Log file name
hbase.log.level=INFO //Log level. If detailed logs are required for fault locating, change it
```

```
to DEBUG. The modification takes effect after restart.
hbase.log.maxbackupindex=20 //Maximum number of log files that can be saved.
Security audit appender
hbase.security.log.file=hbase-client-audit.log //Command of the audit log file
```

### 13.3.3.2 Creating Configuration

#### Function

HBase obtain configuration items by using the login method. The configuration items include user login information and security authentication information.

#### Example Codes

The following code snippet belongs to the **init** method in the **TestMain** class of the **com.huawei.bigdata.hbase.examples** package.

```
private static void init() throws IOException {
 // Default load from conf directory
 conf = HBaseConfiguration.create();
 //In Windows environment
 String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() + File.separator;[1]
 //In Linux environment
 //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
 conf.addResource(new Path(userdir + "core-site.xml"), false);
 conf.addResource(new Path(userdir + "hdfs-site.xml"), false);
 conf.addResource(new Path(userdir + "hbase-site.xml"), false);
}
```

[1]The value of **userdir** is the path of the **conf** directory in the resource path after compilation. Place the **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml** configuration files to the **src/main/resources/conf** directory.

### 13.3.3.3 Creating Connection

#### Function

HBase creates a Connection object using the **ConnectionFactory.createConnection(configuration)** method. The transferred parameter is the Configuration created in the last step.

Connection encapsulates the connections between underlying applications and servers and ZooKeeper. Connection is instantiated using the ConnectionFactory class. Creating Connection is a heavyweight operation. Connection is thread-safe. Therefore, multiple client threads can share one Connection.

In a typical scenario, a client program uses a Connection, and each thread obtains its own Admin or Table instance and invokes the operation interface provided by the Admin or Table object. You are not advised to cache or pool Table and Admin. The lifecycle of Connection is maintained by invokers that frees up resources by invoking **close()**.

#### Example Code

The following code snippet exemplifies login, creating Connection, and creating a table. It belongs to the **HBaseSample** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
private TableName tableName = null;
private Connection conn = null;

public HBaseSample(Configuration conf) throws IOException {
 this.tableName = TableName.valueOf("hbase_sample_table");
 this.conn = ConnectionFactory.createConnection(conf);
}
```

 **NOTE**

Avoid invoking login code repeatedly.

### 13.3.3.4 Creating a Table

#### Function

Create a table using the **createTable** method of the **org.apache.hadoop.hbase.client.Admin** object and specify the table name and column family name. Tables can be created in two modes. (The mode of creating a table using preassigned regions is strongly recommended.)

- Quickly create a table. A newly created table contains only one region which will be split into multiple new regions as data increases.
- Create a table using preassigned regions. Preassign multiple regions before creating a table. This mode accelerates data write at the beginning of massive data write.

 **NOTE**

The column name and column family name of an HBase table consists of letters, digits, and underscores and cannot contain any special characters.

#### Example Code

The following code snippet belongs to the **testCreateTable** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testCreateTable() {
 LOG.info("Entering testCreateTable.");
 // Specify the table descriptor.
 TableDescriptorBuilder htd = TableDescriptorBuilder.newBuilder(tableName);(1)

 // Set the column family name to info.
 ColumnFamilyDescriptorBuilder hcd =
 ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("info")); (2)

 // Set data encoding methods, HBase provides DIFF,FAST_DIFF,PREFIX

 hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);
 // Set compression methods, HBase provides two default compression
 // methods:GZ and SNAPPY
 // GZ has the highest compression rate,but low compression and
 // decompression efficiency,fit for cold data
 // SNAPPY has low compression rate, but high compression and
 // decompression efficiency,fit for hot data.
 // it is advised to use SNAANPPY
 hcd.setCompressionType(Compression.Algorithm.SNAPPY);//Note [1]
 htd.setColumnFamily(hcd.build()); (3)
 Admin admin = null;
 try {
 // Instantiate an Admin object.
 admin = conn.getAdmin(); (4)
 }
```

```
if (!admin.tableExists(tableName)) {
 LOG.info("Creating table...");
 admin.createTable(htd.build()); //Note [2] (5)
 LOG.info(admin.getClusterMetrics().toString());
 LOG.info(admin.listNamespaceDescriptors().toString());
 LOG.info("Table created successfully.");
} else {
 LOG.warn("table already exists");
}
} catch (IOException e) {
 LOG.error("Create table failed " ,e);
} finally {
 if (admin != null) {
 try {
 // Close the Admin object.
 admin.close();
 } catch (IOException e) {
 LOG.error("Failed to close admin " ,e);
 }
 }
}
LOG.info("Exiting testCreateTable.");
}
```

## Explanation

1. Create a table descriptor.
2. Create a column family descriptor.
3. Add the column family descriptor to the table descriptor.
4. Obtain an Admin object. The Admin object provides functions for creating a table, creating a column family, checking whether a table exists, modifying the table structure, modifying the column family structure, and deleting a table.
5. Invoke the table creation method of Admin.

## Precautions

- 1. The compression mode of a column family can be set. The code snippets are as follows:

```
// Set the encoding algorithm. HBase supports the DIFF, FAST_DIFF, PREFIX encoding algorithms.
hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);

// Set the file compression mode. HBase provides the GZ and SNAPPY compression algorithms by
default.
// GZ provides a high compression rate but low compression and decompression performance. GZ is
suitable for cold data.
// SNAPPY provides a low compression rate but high compression and decompression performance.
SNAPPY is suitable for hot data.
// It is recommended that SNAPPY be enabled by default.
hcd.setCompressionType(Compression.Algorithm.SNAPPY);
```

- 2. A table can be created by specifying the start and end RowKeys or preassigning regions using RowKey arrays. The code snippets are as follows:

```
// Create a table whose regions are preassigned.
byte[][] splits = new byte[4][];
splits[0] = Bytes.toBytes("A");
splits[1] = Bytes.toBytes("H");
splits[2] = Bytes.toBytes("O");
splits[3] = Bytes.toBytes("U");
admin.createTable(htd, splits);
```

### 13.3.3.5 Deleting a Table

#### Function

Delete a table using the **deleteTable** method of **org.apache.hadoop.hbase.client.Admin**.

#### Example Code

The following code snippet belongs to the **dropTable** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void dropTable() {
 LOG.info("Entering dropTable.");
 Admin admin = null;
 try {
 admin = conn.getAdmin();
 if (admin.tableExists(tableName)) {
 // Disable the table before deleting it.
 admin.disableTable(tableName);
 // Delete table.
 admin.deleteTable(tableName);// Note[1]
 }
 LOG.info("Drop table successfully.");
 } catch (IOException e) {
 LOG.error("Drop table failed " ,e);
 } finally {
 if (admin != null) {
 try {
 // Close the Admin object.
 admin.close();
 } catch (IOException e) {
 LOG.error("Close admin failed " ,e);
 }
 }
 }
 LOG.info("Exiting dropTable.");
}
```

#### Precautions

A table can be deleted only when the table is disabled. Therefore, **deleteTable** is used together with **disableTable**, **enableTable**, **tableExists**, **isTableEnabled**, and **isTableDisabled**.

### 13.3.3.6 Modifying a Table

#### Function

Modify table information using the **modifyTable** method of **org.apache.hadoop.hbase.client.Admin**.

#### Example Code

The following code snippet belongs to the **testModifyTable** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testModifyTable() {
 LOG.info("Entering testModifyTable.");
```

```
// Specify the column family name.
byte[] familyName = Bytes.toBytes("education");
Admin admin = null;
try {
 // Instantiate an Admin object.
 admin = conn.getAdmin();
 // Obtain the table descriptor.
 TableDescriptor htd = admin.getDescriptor(tableName);
 // Check whether the column family is specified before modification.
 if (!htd.hasFamily(familyName)) {
 // Create the column descriptor.

TableDescriptor tableBuilder = TableDescriptorBuilder.newBuilder(htd)
 .setColumnFamily(ColumnFamilyDescriptorBuilder.newBuilder(familyName).build()).build();

 // Disable the table to get the table offline before modifying
 // the table.
 admin.disableTable(tableName); // Note[1]
 // Submit a modifyTable request.
 admin.modifyTable(tableBuilder);
 // Enable the table to get the table online after modifying the
 // table.
 admin.enableTable(tableName);
}
LOG.info("Modify table successfully.");
} catch (IOException e) {
 LOG.error("Modify table failed " ,e);
} finally {
 if (admin != null) {
 try {
 // Close the Admin object.
 admin.close();
 } catch (IOException e) {
 LOG.error("Close admin failed " ,e);
 }
 }
}
LOG.info("Exiting testModifyTable.");
}
```

## Precautions

**modifyTable** takes effect only when a table is disabled.

### 13.3.3.7 Inserting Data

## Function

HBase is a column-oriented database. A row of data may have multiple column families, and a column family may contain multiple columns. When writing data, you must specify the columns (including the column family names and column names) to which data is written. In HBase, data (a row of data or data sets) is inserted using the **put** method of HTable.

## Example Code

The following code snippet belongs to the **testPut** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testPut() {
 LOG.info("Entering testPut.");

 // Specify the column family name.
```

```
byte[] familyName = Bytes.toBytes("info");
// Specify the column name.
byte[][] qualifiers = {
 Bytes.toBytes("name"), Bytes.toBytes("gender"), Bytes.toBytes("age"), Bytes.toBytes("address")
};

Table table = null;
try {
 // Instantiate an HTable object.
 table = conn.getTable(tableName);
 List<Put> puts = new ArrayList<Put>();

 // Instantiate a Put object.
 Put put = putData(familyName, qualifiers,
 Arrays.asList("012005000201", "Zhang San", "Male", "19", "Shenzhen, Guangdong"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000202", "Li Wanting", "Female", "23", "Shijiazhuang, Hebei"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000203", "Wang Ming", "Male", "26", "Ningbo, Zhejiang"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000204", "Li Gang", "Male", "18", "Xiangyang, Hubei"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000205", "Zhao Enru", "Female", "21", "Shangrao, Jiangxi"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000206", "Chen Long", "Male", "32", "Zhuzhou, Hunan"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000207", "Zhou Wei", "Female", "29", "Nanyang, Henan"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000208", "Yang Yiwen", "Female", "30", "Kaixian, Chongqing"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000209", "Xu Bing", "Male", "26", "Weinan, Shaanxi"));
 puts.add(put);

 put = putData(familyName, qualifiers,
 Arrays.asList("012005000210", "Xiao Kai", "Male", "25", "Dalian, Liaoning"));
 puts.add(put);

 // Submit a put request.
 table.put(puts);

 LOG.info("Put successfully.");
} catch (IOException e) {
 LOG.error("Put failed ", e);
} finally {
 if (table != null) {
 try {
 // Close the HTable object.
 table.close();
 } catch (IOException e) {
 LOG.error("Close table failed ", e);
 }
 }
}
```

```
LOG.info("Exiting testPut.");
}
```

## Precautions

Multiple threads are not allowed to use the same HTable instance at the same time. HTable is a non-thread safe class. If an HTable instance is used by multiple threads at the same time, concurrency problems will occur.

### 13.3.3.8 Deleting Data

#### Function

Delete data (a row of data or data sets) using the **delete** method of a Table instance.

#### Example Code

The following code snippet belongs to the **testDelete** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testDelete() {
 LOG.info("Entering testDelete.");

 byte[] rowKey = Bytes.toBytes("012005000201");

 Table table = null;
 try {
 // Instantiate an HTable object.
 table = conn.getTable(tableName);

 // Instantiate a Delete object.
 Delete delete = new Delete(rowKey);

 // Submit a delete request.
 table.delete(delete);

 LOG.info("Delete table successfully.");
 } catch (IOException e) {
 LOG.error("Delete table failed " ,e);
 } finally {
 if (table != null) {
 try {
 // Close the HTable object.
 table.close();
 } catch (IOException e) {
 LOG.error("Close table failed " ,e);
 }
 }
 }
 LOG.info("Exiting testDelete.");
}
```

#### NOTE

If secondary index is created in the family of the column where the deleted cell is, the index data is synchronously deleted.



### 13.3.3.9 Reading Data Using Get

#### Function

Before reading data from a table, instantiate the Table instance of the table, and then create a Get object. You can also set parameters for the Get object, such as the column family name and column name. Query results are stored in the Result object that stores multiple Cells.

#### Example Code

The following code snippet belongs to the `testGet` method in the `HBaseSample` class of the `com.huawei.bigdata.hbase.examples` package.

```
public void testGet() {
 LOG.info("Entering testGet.");
 // Specify the column family name.
 byte[] familyName = Bytes.toBytes("info");
 // Specify the column name.
 byte[][] qualifier = { Bytes.toBytes("name"), Bytes.toBytes("address") };
 // Specify RowKey.
 byte[] rowKey = Bytes.toBytes("012005000201");
 Table table = null;
 try {
 // Create the Table instance.
 table = conn.getTable(tableName);
 // Instantiate a Get object.
 Get get = new Get(rowKey);
 // Set the column family name and column name.
 get.addColumn(familyName, qualifier[0]);
 get.addColumn(familyName, qualifier[1]);
 // Submit a get request.
 Result result = table.get(get);
 // Print query results.
 for (Cell cell : result.rawCells()) {
 LOG.info("{}:{}", Bytes.toString(CellUtil.cloneRow(cell)),
 Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
 Bytes.toString(CellUtil.cloneValue(cell)));
 }
 LOG.info("Get data successfully.");
 } catch (IOException e) {
 LOG.error("Get data failed " ,e);
 } finally {
 if (table != null) {
 try {
 // Close the HTable object.
 table.close();
 } catch (IOException e) {
 LOG.error("Close table failed " ,e);
 }
 }
 }
 LOG.info("Exiting testGet.");
}
```

### 13.3.3.10 Reading Data Using Scan

#### Function

Before reading data from a table, instantiate the Table instance of the table, create a Scan object, and set parameters for the Scan object based on search criteria. To improve query efficiency, you are advised to specify **StartRow** and

**StopRow.** Query results are stored in the ResultScanner object where each row of data is stored as a Result object that stores multiple Cells.

## Example Code

The following code snippet belongs to the **testScanData** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testScanData() {
 LOG.info("Entering testScanData.");
 Table table = null;
 // Instantiate a ResultScanner object.
 ResultScanner rScanner = null;
 try {
 // Create the Configuration instance.
 table = conn.getTable(tableName);
 // Instantiate a Get object.
 Scan scan = new Scan();
 scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));
 // Set the cache size.
 scan.setCaching(1000);

 // Submit a scan request.
 rScanner = table.getScanner(scan);
 // Print query results.
 for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
 for (Cell cell : r.rawCells()) {
 LOG.info("{}:{}", Bytes.toString(CellUtil.cloneRow(cell)),
 Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
 Bytes.toString(CellUtil.cloneValue(cell)));
 }
 }
 LOG.info("Scan data successfully.");
 } catch (IOException e) {
 LOG.error("Scan data failed " ,e);
 } finally {
 if (rScanner != null) {
 // Close the scanner object.
 rScanner.close();
 }
 if (table != null) {
 try {
 // Close the HTable object.
 table.close();
 } catch (IOException e) {
 LOG.error("Close table failed " ,e);
 }
 }
 }
 LOG.info("Exiting testScanData.");
}
```

## Precautions

1. You are advised to specify **StartRow** and **StopRow** to ensure good performance with a specified Scan scope.
2. You can set **Batch** and **Caching**.
  - **Batch**  
Indicates the maximum number of records returned each time when the **next** interface is invoked using Scan. This parameter is related to the number of columns read each time.
  - **Caching**

Indicates the maximum number of **next** records returned for a remote procedure call (RPC) request. This parameter is related to the number of rows read by an RPC.

### 13.3.3.11 Filtering Data

#### Function

HBase Filter is used to filter data during Scan and Get. You can specify the filter criteria, such as filtering by RowKey, column name, or column value.

#### Example Code

The following code snippet belongs to the **testSingleColumnValueFilter** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testSingleColumnValueFilter() {
 LOG.info("Entering testSingleColumnValueFilter.");
 Table table = null;

 ResultScanner rScanner = null;

 try {

 table = conn.getTable(tableName);

 Scan scan = new Scan();
 scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));
 // Set the filter criteria.
 SingleColumnValueFilter filter = new SingleColumnValueFilter(
 Bytes.toBytes("info"), Bytes.toBytes("name"), CompareOperator.EQUAL,
 Bytes.toBytes("Xu Bing"));
 scan.setFilter(filter);
 // Submit a scan request.
 rScanner = table.getScanner(scan);
 // Print query results.
 for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
 for (Cell cell : r.rawCells()) {
 LOG.info("{}:{}", Bytes.toString(CellUtil.cloneRow(cell)),
 Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
 Bytes.toString(CellUtil.cloneValue(cell)));
 }
 }
 LOG.info("Single column value filter successfully.");
 } catch (IOException e) {
 LOG.error("Single column value filter failed " ,e);
 } finally {
 if (rScanner != null) {
 // Close the scanner object.
 rScanner.close();
 }
 if (table != null) {
 try {
 // Close the HTable object.
 table.close();
 } catch (IOException e) {
 LOG.error("Close table failed " ,e);
 }
 }
 }
 LOG.info("Exiting testSingleColumnValueFilter.");
}
```

## Precautions

Currently, secondary indexes do not support the comparators that use objects of the `SubstringComparator` class as filters.

For example, the following sample code is not supported:

```
Scan scan = new Scan();
filterList = new FilterList(FilterList.Operator.MUST_PASS_ALL);
filterList.addFilter(new SingleColumnValueFilter(Bytes
.toBytes(columnFamily), Bytes.toBytes(qualifier),
CompareOperator.EQUAL, new SubstringComparator(substring)));
scan.setFilter(filterList);
```

### 13.3.3.12 Creating a Secondary Index

#### Function

You can manage HBase secondary indexes using methods provided in **`org.apache.hadoop.hbase.hindex.client.HIndexAdmin`**. This class provides methods of creating an index.

#### NOTE

Secondary indexes cannot be modified. If you need to modify them, delete old indexes and create new ones.

#### Example Code

The following code snippet belongs to the `createIndex` method in the **`HBaseSample`** class of the **`com.huawei.bigdata.hbase.examples`** package.

```
public void createIndex() {
 LOG.info("Entering createIndex.");

 String indexName = "index_name";
 // Create hindex instance
 TableIndices tableIndices = new TableIndices();
 IndexSpecification iSpec = new IndexSpecification(indexName);
 iSpec.addIndexColumn(ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("info")).build(),
 "name", ValueType.STRING);// Note[1]
 tableIndices.addIndex(iSpec);

 HIndexAdmin iAdmin = null;
 Admin admin = null;
 try {

 admin = conn.getAdmin();
 iAdmin = HIndexClient.newHIndexAdmin(admin);

 // add index to the table
 iAdmin.addIndices(tableName, tableIndices);

 LOG.info("Create index successfully.");
 } catch (IOException e) {
 LOG.error("Create index failed " ,e);
 } finally {
 if (admin != null) {
 try {
 admin.close();
 } catch (IOException e) {
 LOG.error("Close admin failed " ,e);
 }
 }
 }
}
```

```
if (iAdmin != null) {
 try {
 // Close IndexAdmin Object
 iAdmin.close();
 } catch (IOException e) {
 LOG.error("Close admin failed ", e);
 }
}
}
LOG.info("Exiting createIndex.");
}
```

By default, newly created level-2 indexes are disabled. To enable a specified level-2 index, see the following code snippet. The following code snippet belongs to the `enableIndex` method in the `HBaseSample` class of the `com.huawei.bigdata.hbase.examples` packet.

```
public void enableIndex() {
 LOG.info("Entering createIndex.");

 // Name of the index to be enabled
 String indexName = "index_name";

 List<String> indexNameList = new ArrayList<String>();
 indexNameList.add(indexName);

 HIndexAdmin iAdmin = null;
 Admin admin = null;
 try {
 admin = conn.getAdmin();
 iAdmin = HIndexClient.newHIndexAdmin(admin);

 // Alternately, enable the specified indices
 iAdmin.enableIndices(tableName, indexNameList);
 LOG.info("Successfully enable indices {} of the table {}", indexNameList, tableName);
 } catch (IOException e) {
 LOG.error("Failed to enable indices {} of the table {} . {}", indexNameList, tableName, e);
 } finally {
 if (admin != null) {
 try {
 admin.close();
 } catch (IOException e) {
 LOG.error("Close admin failed ", e);
 }
 }
 if (iAdmin != null) {
 try {
 iAdmin.close();
 } catch (IOException e) {
 LOG.error("Close admin failed ", e);
 }
 }
 }
}
```

## Precautions

Create a combination index.

HBase supports creation of secondary indexes on multiple fields, for example, the name and age columns.

```
HIndexSpecification iSpecUnite = new HIndexSpecification(indexName);
iSpecUnite.addIndexColumn(new HColumnDescriptor("info"), "name", ValueType.String, 10);
iSpecUnite.addIndexColumn(new HColumnDescriptor("info"), "age", ValueType.String, 3);
```

## Related Operations

### Create an index table by running a command.

You can also use the TableIndexer tool to create an index in an existing user table.

#### NOTE

The **<table\_name>** user table must exist.

```
hbase org.apache.hadoop.hbase.index.mapreduce.TableIndexer -Dtablename.to.index=<table_name> -Dindexspecs.to.add='IDX1=>cf1:[q1->datatype];cf2:[q1->datatype],[q2->datatype],[q3->datatype]#IDX2=>cf1:[q5->datatype]' -Dindexnames.to.build='IDX1'
```

"#" is used to separate indexes. ";" is used to separate column families. "," is used to separate columns.

**tablename.to.index:** indicates the name of the table where the index is created.

**indexspecs.to.add:** indicates the user table columns corresponding to the index.

The parameters in the command are described as follows:

- **IDX1:** indicates the index name.
- **cf1:** indicates the column family name.
- **q1:** indicates the column name.
- **datatype:** indicates the data type. Only the Integer, String, Double, Float, Long, Short, Byte and Char formats are supported.

### 13.3.3.13 Deleting an Index

#### Function

You can manage HBase secondary indexes using methods provided in **org.apache.hadoop.hbase.index.client.HIndexAdmin**. This class provides methods of querying and deleting an index.

#### Example Code

The following code snippet belongs to the **dropIndex** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void dropIndex() {
 LOG.info("Entering dropIndex.");
 String indexName = "index_name";
 List<String> indexNameList = new ArrayList<String>();
 indexNameList.add(indexName);

 IndexAdmin iAdmin = null;
 try {
 // Instantiate HIndexAdmin Object
 iAdmin = HIndexClient.newHIndexAdmin(conn.getAdmin());
 // Delete Secondary Index
 iAdmin.dropIndex(tableName, indexNameList);

 LOG.info("Drop index successfully.");
 } catch (IOException e) {
 LOG.error("Drop index failed.");
 } finally {
 if (iAdmin != null) {
```

```
try {
 // Close Secondary Index
 iAdmin.close();
} catch (IOException e) {
 LOG.error("Close admin failed.");
}
}
}
LOG.info("Exiting dropIndex.");
}
```

## Precautions

- Use the dropIndex and dropIndexes interfaces to delete HBase level-2 indexes. Do not directly drop the index table, because it is an invalid operation and cannot update table information, leading to query failures at the time of index rebuilding.
- If a user table is deleted, the corresponding index table is also deleted.

### 13.3.3.14 Secondary Index-based Query

## Function

In user tables with secondary indexes, you can use Filter to query data. The data query performance is higher than that in user tables without secondary indexes.

#### NOTE

- HIndex supports three Filter types: SingleColumnValueFilter, SingleColumnValueExcludeFilter, and SingleColumnValuePartitionFilter.
- HIndex supports the following Comparator types: BinaryComparator, BitComparator, LongComparator, DecimalComparator, DoubleComparator, FloatComparator, IntComparator, and NullComparator.

The secondary index usage rules are as follows:

- For scenarios in which a single index is created for one or multiple columns:
  - When you use this column for AND or OR query filtering, the index is used to improve the query performance.  
For example, Filter\_Condition(IndexCol1) AND/OR Filter\_Condition(IndexCol2).
  - When you use "Index Column AND Non-Index Column" for filtering in the query, this index is used to improve the query performance.  
For example, Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol2) AND Filter\_Condition(NonIndexCol1).
  - When you use "Index Column OR Non-Index Column" for filtering in the query, the index is not used and the query performance is not improved.  
For example, Filter\_Condition(IndexCol1) AND/OR Filter\_Condition(IndexCol2) OR Filter\_Condition(NonIndexCol1).
- For scenarios in which a combination index is created for multiple columns:
  - When the columns used for query are all or part of the columns of the combination index and are in the same sequence with the combination index, the index is used to improve the query performance.

For example, a combination index is created for C1, C2, and C3. The index takes effect in the following scenarios:

```
Filter_Condition(IndexCol1) AND Filter_Condition(IndexCol2) AND
Filter_Condition(IndexCol3)
```

```
Filter_Condition(IndexCol1) AND Filter_Condition(IndexCol2)
```

```
Filter_Condition(IndexCol1)
```

The index does not take effect in the following scenarios:

```
Filter_Condition(IndexCol2) AND Filter_Condition(IndexCol3)
```

```
Filter_Condition(IndexCol1) AND Filter_Condition(IndexCol3)
```

```
Filter_Condition(IndexCol2)
```

```
Filter_Condition(IndexCol3)
```

- When you use "Index Column AND Non-Index Column" for filtering in the query, this index is used to improve the query performance.

For example:

```
Filter_Condition(IndexCol1) AND Filter_Condition(NonIndexCol1)
```

```
Filter_Condition(IndexCol1) AND Filter_Condition(IndexCol2) AND
Filter_Condition(NonIndexCol1)
```

- When you use "Index Column OR Non-Index Column" for filtering in the query, the index is not used and the query performance is not improved.

For example:

```
Filter_Condition(IndexCol1) OR Filter_Condition(NonIndexCol1)
```

```
(Filter_Condition(IndexCol1) AND Filter_Condition(IndexCol2))OR
(Filter_Condition(NonIndexCol1))
```

- When multiple columns are used for query, a value range can be specified only for the last column in the combination index and the other columns can only be set to a specified value.

For example, a combination index is created for C1, C2, and C3. In range query, a value range can be set only for C3 and the filter criterion is "C1 = XXX, C2 = XXX, and C3 = value range".

- For scenarios in which secondary index is created in a user table, you can use Filter to query data. The query results of the single and combination index with filter are the same as those in the table without secondary index. The data query performance is higher than that in user tables without secondary indexes.

## Example Code

The following code snippet belongs to the **testScanDataByIndex** method in the **HBaseSample** class of the **com.huawei.hadoop.hbase.example** package.

### Example: Query data using secondary indexes.

```
public void testScanDataByIndex() {
 LOG.info("Entering testScanDataByIndex.");
 Table table = null;
 ResultScanner scanner = null;
 try {
 table = conn.getTable(tableName);

 // Create a filter for indexed column.
```



```

Filter filter = new SingleColumnValueFilter(Bytes.toBytes("info"), Bytes.toBytes("name"),
 CompareOperator.EQUAL, "Li Gang".getBytes());
Scan scan = new Scan();
scan.setFilter(filter);
scanner = table.getScanner(scan);
LOG.info("Scan indexed data.");

for (Result result : scanner) {
 for (Cell cell : result.rawCells()) {
 LOG.info("{}:{}", Bytes.toString(CellUtil.cloneRow(cell)),
 Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
 Bytes.toString(CellUtil.cloneValue(cell)));
 }
}
LOG.info("Scan data by index successfully.");
} catch (IOException e) {
 LOG.error("Scan data by index failed.");
} finally {
 if (scanner != null) {
 // Close the scanner object.
 scanner.close();
 }
 try {
 if (table != null) {
 table.close();
 }
 } catch (IOException e) {
 LOG.error("Close table failed.");
 }
}

LOG.info("Exiting testScanDataByIndex.");
}

```

## Precaution

Create secondary indexes for the **name** field first.

## Related Operations

Query a table using a secondary index.

The following provides an example:

Add an index to the **name** column of the **info** column family in **hbase\_sample\_table**. Run the following command on the client:

```
hbase org.apache.hadoop.hbase.index.mapreduce.TableIndexer -Dtablename.to.index=hbase_sample_table -Dindexspecs.to.add='IDX1=>info:[name->String]' -Dindexnames.to.build='IDX1'
```

Query **info:name**. Run the following command on the HBase shell client:

```
>scan 'hbase_sample_table',
{FILTER=>"SingleColumnValueFilter(family,qualifier,compareOp,comparator,filterIfMissing,latestVersionOnly)"}
```

 NOTE

Use APIs to perform complex query on the HBase shell client.

The parameters are described as follows:

- **family**: indicates the column family where the column to be queried locates, such as **info**.
- **qualifier**: indicates the column to be queried, such as **name**.
- **compareOp**: indicates the comparison operator, such as = and >.
- **comparator**: indicates the target value to be queried, such as **binary:Zhang San**.
- **filterIfMissing**: indicates whether a row is filtered if the column does not exist in this row. The default value is **false**.
- **latestVersionOnly**: indicates whether only values of the latest version are to be queried. The default value is **false**.

For example:

```
>scan hbase_sample_table',{FILTER=>"SingleColumnValueFilter('info','name',=,'binary:Zhang San',true,true)"}
```

### 13.3.3.15 Multi-Point Region Division

#### Function

You can perform multi-point division by using **org.apache.hadoop.hbase.client.HBaseAdmin**. Note that the division operations take effect on empty regions only.

In this example, the multi-point division is performed on an HBase table by using **multiSplit**. The table will be split into 5 parts: "-∞~A", "A~D", "D~F", "F~H", and "H~+∞".

#### Example Code

The following code snippet belongs to the **testMultiSplit** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testMultiSplit() {
 LOG.info("Entering testMultiSplit.");

 Table table = null;
 Admin admin = null;
 try {
 admin = conn.getAdmin();

 // initialize a HTable object
 table = conn.getTable(tableName);
 Set<HRegionInfo> regionSet = new HashSet<HRegionInfo>();
 List<HRegionLocation> regionList = conn.getRegionLocator(tableName).getAllRegionLocations();
 for(HRegionLocation hrl : regionList){
 regionSet.add(hrl.getRegionInfo());
 }
 byte[][] sk = new byte[4][];
 sk[0] = "A".getBytes();
 sk[1] = "D".getBytes();
 sk[2] = "F".getBytes();
 sk[3] = "H".getBytes();
 for (RegionInfo regionInfo : regionSet) {
 admin.multiSplitSync(regionInfo.getRegionName(), sk);
 }
 LOG.info("MultiSplit successfully.");
 } catch (Exception e) {
```

```
LOG.error("MultiSplit failed.");
} finally {
 if (table != null) {
 try {
 // Close table object
 table.close();
 } catch (IOException e) {
 LOG.error("Close table failed " ,e);
 }
 }
 if (admin != null) {
 try {
 // Close the Admin object.
 admin.close();
 } catch (IOException e) {
 LOG.error("Close admin failed " ,e);
 }
 }
}
LOG.info("Exiting testMultiSplit.");
}
```

Note that the division operations take effect on empty regions only.

### 13.3.3.16 Creating a Phoenix Table

#### Function

Phoenix can be installed on HBase to enable it to support SQL and JDBC APIs. This way, SQL users can access the HBase cluster.

#### Example Code

The following code snippet belongs to the **testCreateTable** method in the **PhoenixSample** class of the **com.huawei.bigdata.hbase.examplespackage**.

```
/**
 * Create Table
 */
public void testCreateTable() {
 LOG.info("Entering testCreateTable.");
 String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
 // Create table
 String createTableSQL =
 "CREATE TABLE IF NOT EXISTS TEST (id integer not null primary key, name varchar, "
 + "account char(6), birth date)";
 try (Connection conn = DriverManager.getConnection(url, props);
 Statement stat = conn.createStatement()) {
 // Execute Create SQL
 stat.executeUpdate(createTableSQL);
 LOG.info("Create table successfully.");
 } catch (Exception e) {
 LOG.error("Create table failed.", e);
 }
 LOG.info("Exiting testCreateTable.");
}
/**
 * Drop Table
 */
public void testDrop() {
 LOG.info("Entering testDrop.");
 String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
 // Delete table
 String dropTableSQL = "DROP TABLE TEST";

 try (Connection conn = DriverManager.getConnection(url, props);
```

```
Statement stat = conn.createStatement() {
 stat.executeUpdate(dropTableSQL);
 LOG.info("Drop successfully.");
} catch (Exception e) {
 LOG.error("Drop failed.", e);
}
LOG.info("Exiting testDrop.");
}
```

### 13.3.3.17 Writing Data to the PhoenixTable

#### Function

The Phoenix table enables data writing in HBase.

#### Example Code

The following code snippet belongs to the `testPut` method in the `PhoenixSample` class of the `com.huawei.bigdata.hbase.examples` package.

```
/**
 * Put data
 */
public void testPut() {
 LOG.info("Entering testPut.");
 String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
 // Insert
 String upsertSQL =
 "UPSERT INTO TEST VALUES(1,'John','100000', TO_DATE('1980-01-01','yyyy-MM-dd'))";
 try (Connection conn = DriverManager.getConnection(url, props);
 Statement stat = conn.createStatement()){
 // Execute Update SQL
 stat.executeUpdate(upsertSQL);
 conn.commit();
 LOG.info("Put successfully.");
 } catch (Exception e) {
 LOG.error("Put failed.", e);
 }
 LOG.info("Exiting testPut.");
}
```

### 13.3.3.18 Reading the PhoenixTable

#### Function

The Phoenix table enables data reading.

#### Example Code

The following code snippet belongs to the `testSelect` method in the `PhoenixSample` class of the `com.huawei.bigdata.hbase.examples` package.

```
/**
 * Select Data
 */
public void testSelect() {
 LOG.info("Entering testSelect.");
 String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
 // Query
 String querySQL = "SELECT * FROM TEST WHERE id = ?";
 Connection conn = null;
 PreparedStatement preStat = null;
```

```
Statement stat = null;
ResultSet result = null;
try {
 // Create Connection
 conn = DriverManager.getConnection(url, props);
 // Create Statement
 stat = conn.createStatement();
 // Create PreparedStatement
 preStat = conn.prepareStatement(querySQL);
 // Execute query
 preStat.setInt(1, 1);
 result = preStat.executeQuery();
 // Get result
 while (result.next()) {
 int id = result.getInt("id");
 String name = result.getString(1);
 System.out.println("id: " + id);
 System.out.println("name: " + name);
 }
 LOG.info("Select successfully.");
} catch (Exception e) {
 LOG.error("Select failed.", e);
} finally {
 if (null != result) {
 try {
 result.close();
 } catch (Exception e2) {
 LOG.error("Result close failed.", e2);
 }
 }
 if (null != stat) {
 try {
 stat.close();
 } catch (Exception e2) {
 LOG.error("Stat close failed.", e2);
 }
 }
 if (null != conn) {
 try {
 conn.close();
 } catch (Exception e2) {
 LOG.error("Connection close failed.", e2);
 }
 }
}
LOG.info("Exiting testSelect.");
}
```

### 13.3.3.19 Accessing Multiple ZooKeepers

#### Function

This function allows simultaneous access to FusionInsight ZooKeeper from the HBase client and third-party ZooKeeper from the customer application in the same client process.

#### Example code

The following code snippet is in the **TestZKSample** class of the **hbase-zk-example** \src\main\java\com\huawei\hadoop\hbase\example. You need to pay attention to the **login** and **connectApacheZK** methods.

```
private static void login(String keytabFile, String principal) throws IOException {
 conf = HBaseConfiguration.create();
 //In Windows environment
```

```
[1] String confDirPath = TestZKSample.class.getClassLoader().getResource("").getPath() + File.separator;
//In Linux environment
//String confDirPath = System.getProperty("user.dir") + File.separator + "conf" + File.separator;

// Set zoo.cfg for hbase to connect to fi zookeeper.
conf.set("hbase.client.zookeeper.config.path", confDirPath + "zoo.cfg");
if (User.isHBaseSecurityEnabled(conf)) {
 // jaas.conf file, it is included in the client package file
 System.setProperty("java.security.auth.login.config", confDirPath + "jaas.conf");
 // set the kerberos server info,point to the kerberosclient
 System.setProperty("java.security.krb5.conf", confDirPath + "krb5.conf");
 // set the keytab file name
 conf.set("username.client.keytab.file", confDirPath + keytabFile);
 // set the user's principal
 try {
 conf.set("username.client.kerberos.principal", principal);
 User.login(conf, "username.client.keytab.file", "username.client.kerberos.principal",
 InetAddress.getLocalHost().getCanonicalHostName());
 } catch (IOException e) {
 throw new IOException("Login failed.", e);
 }
}
}
private void connectApacheZK() throws IOException, org.apache.zookeeper KeeperException {
 try {
 // Create apache zookeeper connection.
 ZooKeeper digestZk = new ZooKeeper("127.0.0.1:2181", 60000, null);
 LOG.info("digest directory: {}", digestZk.getChildren("/", null));
 LOG.info("Successfully connect to apache zookeeper.");
 } catch (InterruptedException e) {
 LOG.error("Found error when connect apache zookeeper ", e);
 }
}
}
```

- [1]The value of **userdir** is the path of the **conf** directory in the resource path after compilation. Place the **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml** configuration files to the **src/main/resources/conf** directory.
- The **jaas.conf** file specified by the **java.security.auth.login.config** parameter in the **login** method is used to set the authentication information for accessing ZooKeeper. The example code contains the Client\_new and Client configurations. The Client\_new configuration is used to access FusionInsight ZooKeeper and the Client configuration is used to access Apache ZooKeeper.
- The **hbase.client.zookeeper.config.path** parameter in the **login** method controls the access to the FusionInsight ZooKeeper client. The following parameters are involved:
  - **zookeeper.sasl.clientconfig**: Specifies the configuration in the **jaas.conf** file used for accessing FusionInsight ZooKeeper.
  - **zookeeper.server.principal**: Specifies the principle of the ZooKeeper server.
  - **zookeeper.sasl.client**: If the cluster works in the security mode, set this parameter to **true**. Otherwise, set this parameter to **false**. If this parameter is set to **false**, the **zookeeper.sasl.clientconfig** and **zookeeper.server.principal** parameters do not take effect.

### 13.3.3.20 Querying Cluster Information Using REST

#### Function

Use the REST service to transfer the URL consisting of the host and port to obtain the cluster version and status information through HTTP.

#### Example Code

- Connecting to the RestServer Service

In normal mode, users can connect to the RestServer service without logging in. Therefore, comment out the following code statements related to login in the **main** method of the **HBaseRestTest** class in the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** package:

```
//In Windows environment
//String userdir = HBaseRestTest.class.getClassLoader().getResource("conf").getPath() +
File.separator;
//In Linux environment
//String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
//userKeytabFile = userdir + "user.keytab";
//krb5File = userdir + "krb5.conf";
//String principal = "hbaseuser1";
//login(principal, userKeytabFile, krb5File);
```

The following code snippets are in the **main** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
// RESTServer's hostname.
String restHostName = "xxx.xxx.xxx.xxx";[1]
String securityModeUrl = new StringBuilder("https://").append(restHostName).append(":
21309").toString();
String nonSecurityModeUrl = new StringBuilder("http://").append(restHostName).append(":
21309").toString();
HBaseRestTest test = new HBaseRestTest();

//If cluster is non-security mode,use nonSecurityModeUrl as parameter.
test.test(nonSecurityModeUrl);[2]
```

[1] Change the value of **restHostName** to the IP address of the node where the RestServer instance to be accessed is located, and configure the node IP address in the hosts file on the local host where the sample code is run.

[2] In non-security mode, access the HBase REST service in HTTP mode and use **nonSecurityModeUrl** as the **test.test()** parameter.

- Obtaining the cluster version information

The following code snippets are in the **getClusterVersion** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getClusterVersion(String url) {
 String endpoint = "/version/cluster";
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
 handleNormalResult((Optional<ResultModel>) result);
}
```

- Obtaining the cluster status information

The following code snippets are in the **getClusterStatus** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getClusterStatus(String url) {
 String endpoint = "/status/cluster";
```

```
Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
handleNormalResult(result);
}
```

### 13.3.3.21 Obtaining All Tables Using REST

#### Function

Use the REST service and transfer the URL consisting of the host and port to obtain all tables using HTTP.

#### Example Code

The following code snippets are in the **getAllUserTables** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getAllUserTables(String url) {
 String endpoint = "/";
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
 handleNormalResult(result);
}
```

### 13.3.3.22 Operate Namespaces Using REST

#### Function

Use the REST service to import the URL consisting of the host and port and the specified namespace, Use HTTP to create, query, and delete namespaces, and obtain tables in the specified namespace.

#### Example Code

- Invoking methods

```
// Namespace operations.
createNamespace(url, "testNs");
getAllNamespace(url);
deleteNamespace(url, "testNs");
getAllNamespaceTables(url, "default");
```

- Creating a namespace

The following code snippets are in the **createNamespace** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void createNamespace(String url, String namespace) {
 String endpoint = "/namespaces/" + namespace;
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.POST, null);
 if (result.orElse(new ResultModel()).getStatusCode() == HttpStatus.SC_CREATED) {
 LOG.info("Create namespace '{}' success.", namespace);
 } else {
 LOG.error("Create namespace '{}' failed.", namespace);
 }
}
```

- Querying all namespaces

The following code snippets are in the **getAllNamespace** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.



```
private void getAllNamespace(String url) {
 String endpoint = "/namespaces";
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
 handleNormalResult(result);
}
```

- Deleting a specified namespace

The following code snippets are in the **deleteNamespace** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void deleteNamespace(String url, String namespace) {
 String endpoint = "/namespaces/" + namespace;
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.DELETE, null);
 if (result.orElse(new ResultModel()).getStatusCode() == HttpStatus.SC_OK) {
 LOG.info("Delete namespace '{}' success.", namespace);
 } else {
 LOG.error("Delete namespace '{}' failed.", namespace);
 }
}
```

- Obtain tables in a specified namespace.

The following code snippets are in the **getAllNamespaceTables** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getAllNamespaceTables(String url, String namespace) {
 String endpoint = "/namespaces/" + namespace + "/tables";
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
 handleNormalResult(result);
}
```

### 13.3.3.23 Operate Tables Using REST

#### Function

Use the REST service to transfer the URL consisting of the host and port as well as the specified **tableName** and **jsonHTD** to query, modify, create, and delete table information through HTTP.

#### Example Code

- Invoking methods

```
// Add a table with specified info.
createTable(url, "testRest",
 "{\"name\":\"default:testRest\",\"ColumnSchema\":{\"name\":\"cf1\"},\" + \"{\"name\":\"cf2\"}}");

// Add column family 'testCF1' if not exist, else update the 'VERSIONS' to 3.
// Notes: The unspecified property of this column family will be updated to default value.
modifyTable(url, "testRest",
 "{\"name\":\"testRest\",\"ColumnSchema\":{\"name\":\"testCF1\", \" + \"\"VERSIONS\":\"3\" + \"\"}}");

// Describe specific Table.
descTable(url, "default:testRest");

// delete a table with specified info.
deleteTable(url, "default:testRest",
 "{\"name\":\"default:testRest\",\"ColumnSchema\":{\"name\":\"testCF\", \" + \"\"VERSIONS\":\"3\"}}");
```

- Querying table information

The following code snippets are in the **descTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void descTable(String url, String tableName) {
 String endpoint = "/" + tableName + "/schema";
 Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
 handleNormalResult((Optional<ResultModel>) result);
}
```

- Modifying table information

The following code snippets are in the **modifyTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void modifyTable(String url, String tableName, String jsonHTD) {
 LOG.info("Start modify table.");
 String endpoint = "/" + tableName + "/schema";
 JsonElement tableDesc = new JsonParser().parse(jsonHTD);

 // Add a new column family or modify it.
 handleNormalResult(sendAction(url + endpoint, MethodType.POST, tableDesc));
}
```

- Creating a table

The following code snippets are in the **createTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void createTable(String url, String tableName, String jsonHTD) {
 LOG.info("Start create table.");
 String endpoint = "/" + tableName + "/schema";
 JsonElement tableDesc = new JsonParser().parse(jsonHTD);

 // Add a table.
 handleCreateTableResult(sendAction(url + endpoint, MethodType.PUT, tableDesc));
}
```

- Deleting a table

The following code snippets are in the **deleteTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void deleteTable(String url, String tableName, String jsonHTD) {
 LOG.info("Start delete table.");
 String endpoint = "/" + tableName + "/schema";
 JsonElement tableDesc = new JsonParser().parse(jsonHTD);

 // delete a table.
 handleNormalResult(sendAction(url + endpoint, MethodType.DELETE, tableDesc));
}
```

### 13.3.3.24 Accessing the ThriftServer Operation Table

#### Scenario

After importing the host where the ThriftServer instances are located and the port that provides services, you can create a Thrift client using the authentication credential and configuration file, access ThriftServer, and obtain table names, create a table, and delete a table based on the specified namespace.

#### Procedure

- Step 1** Log in to FusionInsight Manager, choose **Cluster > Service > HBase > Configuration** and click **All Configurations**, search for and modify the parameter **hbase.thrift.security.qop** of the ThriftServer instance. The value of this parameter

must be the same as that of **hbase.rpc.protection**. Save the configuration and restart the node service for the configuration to take effect.

 **NOTE**

The mapping between **hbase.rpc.protection** and **hbase.thrift.security.qop** is as follows:

- "privacy" - "auth-conf"
- "authentication" - "auth"
- "integrity" - "auth-int"

**Step 2** Obtain the configuration file of the ThriftServer instance in the cluster.

- Method 1: Choose **Cluster > Service > HBase > Instance**, click the ThriftServer instance to go to the details page, and obtain the configuration files **hdfs-site.xml**, **core-site.xml**, and **hbase-site.xml**.
- Method 2: Obtain the configuration files by decompressing the client file in [Preparing for Development and Operating Environment](#). Manually add the following configuration to **hbase-site.xml**. The value of **hbase.thrift.security.qop** must be the same as that in [Step 1](#).

```
<property>
<name>hbase.thrift.security.qop</name>
<value>auth</value>
</property>
<property>
<name>hbase.thrift.kerberos.principal</name>
<value>thrift/hadoop.hadoop.com@HADOOP.COM</value>
</property>
<property>
<name>hbase.thrift.keytab.file</name>
<value>/opt/huawei/Bigdata/FusionInsight_HD_8.1.0.1/install/FusionInsight-HBase-2.2.3/keytabs/
HBase/thrift.keytab</value>
</property>
```

----End

## Example Code

- Initializing configuration

The following code snippets belong to the **TestMain** class in the **com.huawei.bigdata.hbase.examples** package of the **hbase-thrift-example** sample project.

```
private static void init() throws IOException {
 // Default load from conf directory
 conf = HBaseConfiguration.create();

 String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() + File.separator;
[1]
 //In Linux environment
 //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
 conf.addResource(new Path(userdir + "core-site.xml"), false);
 conf.addResource(new Path(userdir + "hdfs-site.xml"), false);
 conf.addResource(new Path(userdir + "hbase-site.xml"), false);
}
```

[1] **userdir** obtains the **conf** directory in the resource path after compilation. The **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml** files used for initial configuration must be stored in the **src/main/resources/conf** directory.

- Connecting to a ThriftServer instance

The following code snippets belong to the **TestMain** class in the **com.huawei.bigdata.hbase.examples** package of the **hbase-thrift-example** sample project.

```
try {
 test = new ThriftSample();
 test.test("xxx.xxx.xxx.xxx", THRIFT_PORT, conf);[2]
} catch (TException | IOException e) {
 LOG.error("Test thrift error", e);
}
```

[2] The value of the input parameter **test.test()** is the IP address of the node where the ThriftServer instance to be accessed is located. Change the IP address to the actual one. The IP address of the node must be configured in the hosts file of the local host where the sample code is run.

**THRIFT\_PORT** is the value of **hbase.regionserver.thrift.port** configured for the ThriftServer instance.

- Invoking methods

```
// Get table of specified namespace.
getTableNamesByNamespace(client, "default");
// Create table.
createTable(client, TABLE_NAME);
// Delete specified table.
deleteTable(client, TABLE_NAME);
```

- Obtaining table names based on the specified namespace

The following code snippets are in the **getTableNamesByNamespace** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getTableNamesByNamespace(THBaseService.Iface client, String namespace) throws
TException {
 client.getTableNamesByNamespace(namespace)
 .forEach(
 tTableName -> LOGGER.info("{} ", TableName.valueOf(tTableName.getNs()),
 tTableName.getQualifier()));
}
```

- Creating a table

The following code snippets are in the **createTable** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void createTable(THBaseService.Iface client, String tableName) throws TException,
IOException {
 TTableName table = getTableName(tableName);
 TTableDescriptor descriptor = new TTableDescriptor(table);
 descriptor.setColumns(
 Collections.singletonList(new
 TColumnFamilyDescriptor().setName(COLUMN_FAMILY.getBytes())));
 if (client.tableExists(table)) {
 LOGGER.warn("Table {} is exists, delete it.", tableName);
 client.disableTable(table);
 client.deleteTable(table);
 }
 client.createTable(descriptor, null);
 if (client.tableExists(table)) {
 LOGGER.info("Created {}.", tableName);
 } else {
 LOGGER.error("Create {} failed.", tableName);
 }
}
```

- Deleting a table

The following code snippets are in the **deleteTable** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void deleteTable(THBaseService.Iface client, String tableName) throws TException,
IOException {
```

```
TTableName table = getTableName(tableName);
if (client.tableExists(table)) {
 client.disableTable(table);
 client.deleteTable(table);
 LOGGER.info("Deleted {}. ", tableName);
} else {
 LOGGER.warn("{} not exist.", tableName);
}
}
```

### 13.3.3.25 Accessing ThriftServer to Write Data

#### Function

After importing the host where the ThriftServer instances are located and the port that provides services, you can create a Thrift client using the authentication credential and configuration file, access the ThriftServer, and use Put and putMultiple to write data.

#### Example Code

- Invoking methods

```
// Write data with put.
putData(client, TABLE_NAME);

// Write data with putlist.
putDataList(client, TABLE_NAME);
```

- Using Put to write data.

The following code snippets are in the **putData** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void putData(THBaseService.Iface client, String tableName) throws TException {
 LOGGER.info("Test putData.");
 TPut put = new TPut();
 put.setRow("row1".getBytes());

 TColumnValue columnValue = new TColumnValue();
 columnValue.setFamily(COLUMN_FAMILY.getBytes());
 columnValue.setQualifier("q1".getBytes());
 columnValue.setValue("test value".getBytes());
 List<TColumnValue> columnValues = new ArrayList<>(1);
 columnValues.add(columnValue);
 put.setColumnValues(columnValues);

 ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
 client.put(table, put);
 LOGGER.info("Test putData done.");
}
```

- Using putMultiple to write data.

The following code snippets are in the **putDataList** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void putDataList(THBaseService.Iface client, String tableName) throws TException {
 LOGGER.info("Test putDataList.");
 TPut put1 = new TPut();
 put1.setRow("row2".getBytes());
 List<TPut> putList = new ArrayList<>();

 TColumnValue q1Value = new TColumnValue(ByteBuffer.wrap(COLUMN_FAMILY.getBytes()),
 ByteBuffer.wrap("q1".getBytes()), ByteBuffer.wrap("test value".getBytes()));
 TColumnValue q2Value = new TColumnValue(ByteBuffer.wrap(COLUMN_FAMILY.getBytes()),
```

```
 ByteBuffer.wrap("q2".getBytes()), ByteBuffer.wrap("test q2 value".getBytes()));
 List<TColumnValue> columnValues = new ArrayList<>(2);
 columnValues.add(q1Value);
 columnValues.add(q2Value);
 put1.setColumnValues(columnValues);
 putList.add(put1);

 TPut put2 = new TPut();
 put2.setRow("row3".getBytes());

 TColumnValue columnValue = new TColumnValue();
 columnValue.setFamily(COLUMN_FAMILY.getBytes());
 columnValue.setQualifier("q1".getBytes());
 columnValue.setValue("test q1 value".getBytes());
 put2.setColumnValues(Collections.singletonList(columnValue));
 putList.add(put2);

 ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
 client.putMultiple(table, putList);
 LOGGER.info("Test putDataList done.");
 }
}
```

### 13.3.3.26 Accessing ThriftServer to Read Data

#### Function

After importing the host where the ThriftServer instances are located and the port that provides services, you can create a Thrift client using the authentication credential and configuration file, access the ThriftServer, and use **get** and **scan** methods to read data.

#### Example Code

- Invoking methods

```
// Get data with single get.
getData(client, TABLE_NAME);

// Get data with getlist.
getDataList(client, TABLE_NAME);

// Scan data.
scanData(client, TABLE_NAME);
```

- Using the **get** method to write data.

The following code snippets are in the **getData** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getData(THBaseService.Iface client, String tableName) throws TException {
 LOGGER.info("Test getData.");
 TGet get = new TGet();
 get.setRow("row1".getBytes());

 ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
 TResult result = client.get(table, get);
 printResult(result);
 LOGGER.info("Test getData done.");
}
```

- Using the **getlist** method to write data.

The following code snippets are in the **getDataList** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getDataList(THBaseService.Iface client, String tableName) throws TException {
 LOGGER.info("Test getDataList.");
 List<TGet> getList = new ArrayList<>();
 TGet get1 = new TGet();
 get1.setRow("row1".getBytes());
 getList.add(get1);

 TGet get2 = new TGet();
 get2.setRow("row2".getBytes());
 getList.add(get2);

 ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
 List<TResult> resultList = client.getMultiple(table, getList);
 for (TResult tResult : resultList) {
 printResult(tResult);
 }
 LOGGER.info("Test getDataList done.");
}
```

- Using the **scan** method to write data.

The following code snippets are in the **scanData** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void scanData(THBaseService.Iface client, String tableName) throws TException {
 LOGGER.info("Test scanData.");
 int scannerId = -1;
 try {
 ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
 TScan scan = new TScan();
 scan.setLimit(500);
 scannerId = client.openScanner(table, scan);
 List<TResult> resultList = client.getScannerRows(scannerId, 100);
 while (resultList != null && !resultList.isEmpty()) {
 for (TResult tResult : resultList) {
 printResult(tResult);
 }
 resultList = client.getScannerRows(scannerId, 100);
 }
 } finally {
 if (scannerId != -1) {
 client.closeScanner(scannerId);
 LOGGER.info("Closed scanner {}.", scannerId);
 }
 }
 LOGGER.info("Test scanData done.");
}
```

### 13.3.3.27 Using HBase Dual-Read

#### Scenario

The HBase client application loads the configuration items of the active and standby clusters by customization to implement the dual-read capability. HBase dual-read is a key feature that improves the high availability of the HBase cluster system. It applies to four query scenarios: reading data using **Get**, reading data in batches using **Get**, reading data using **Scan**, and querying data using a secondary index. HBase can read data from the active and standby clusters at the same time, reducing the query glitch time. The advantages are as follows:

- High success rate: The concurrent dual-read mechanism ensures a high success rate of read requests.
- High availability: When a single cluster is faulty, the query service is not interrupted. A short network jitter does not prolong the query time.

- High generality: The dual-read feature does not support dual-write, but does not affect the original real-time write scenario.
- Ease-of-use: Client encapsulation is performed, which is not sensed by services.

 NOTE

Restrictions on HBase dual-read:

- The HBase dual-read feature is implemented based on replication. Data read from the standby cluster may be different from that from the active cluster. Therefore, only eventual consistency can be achieved.
- Currently, the HBase dual-read feature is used only for query. When the active cluster breaks down, the latest data cannot be synchronized. As a result, the latest data cannot be queried in the standby cluster.
- A **Scan** operation of HBase may be split into multiple RPC operations. Data may not be completely the same because related session information is not synchronized between different clusters. Therefore, the dual-read feature takes effect only when an RPC operation is performed for the first time. Requests before ResultScanner close access the cluster used for the first RPC operation.
- The HBase Admin API and real-time write API access only the active cluster. Therefore, after the active cluster breaks down, the Admin API and real-time write API are unavailable, and only the **Get** and **Scan** query services are available.

## Procedure

- Step 1** Obtain the client configuration files **core-site.xml**, **hbase-site.xml**, and **hdfs-site.xml** of the HBase active cluster and save them to the **src/main/resources/conf/active** directory. This directory needs to be created by yourself. For details, see [Preparing for Development and Operating Environment](#).
- Step 2** Obtain the client configuration files **core-site.xml**, **hbase-site.xml**, and **hdfs-site.xml** of the standby cluster and save them to the **src/main/resources/conf/standby** directory. This directory needs to be created by yourself. For details, see [Preparing for Development and Operating Environment](#).
- Step 3** Create the **hbase-dual.xml** configuration file and save it to the **src/main/resources/conf/** directory. For details about the configuration items in the configuration file, see [Table 13-6](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<!--Configuration file directory of the active cluster-->
 <property>
 <name>hbase.dualclient.active.cluster.configuration.path</name>
 <value>{Sample code directory}\src\main\resources\conf\active</value>
 </property>
<!--Configuration file directory of the standby cluster-->
 <property>
 <name>hbase.dualclient.standby.cluster.configuration.path</name>
 <value>{Sample code directory}\src\main\resources\standby</value>
 </property>
<!--Connection implementation of the dual-read mode-->
 <property>
 <name>hbase.client.connection.impl</name>
 <value>org.apache.hadoop.hbase.client.HBaseMultiClusterConnectionImpl</value>
 </property>
<!--Normal mode-->
 <property>
 <name>hbase.security.authentication</name>
 <value>Simple</value>
 </property>
```



```
<!--Normal mode-->
<property>
 <name>hadoop.security.authentication</name>
 <value>Simple</value>
</property>
```

----End

## Sample Code

- Creating a dual-read configuration. The following code snippet belongs to the **init** method in **TestMain** class of the **com.huawei.bigdata.hbase.examples** packet.

```
private static void init() throws IOException {
 // Default load from conf directory
 conf = HBaseConfiguration.create();
 //In Windows environment
 String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() + File.separator;
 //In Linux environment
 //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
 conf.addResource(new Path(userdir + "hbase-dual.xml"), false);
}
```

- Determining the data source cluster
  - GET request. The following code snippet belongs to the **testGet** method in **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
Result result = table.get(get);
if (result instanceof DualResult) {
 LOG.info(((DualResult)result).getClusterId());
}
```

- Scan request. The following code snippet belongs to the **testScanData** method in **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
ResultScanner rScanner = table.getScanner(scan);
if (rScanner instanceof HBaseMultiScanner) {
 LOG.info(((HBaseMultiScanner)rScanner).getClusterId());
}
```

- The client can print metric information.

Add the following content to the **log4j.properties** file so that the client can export metric information to the specified file: For details about the metrics, see [Printing Metric Information](#)

```
log4j.logger.DUAL=debug,DUAL
log4j.appender.DUAL=org.apache.log4j.RollingFileAppender
log4j.appender.DUAL.File=/var/log/dual.log //Local dual-read log path on the client. Change the value
to the actual directory, but ensure that the directory has the write permission.
log4j.additivity.DUAL=false
log4j.appender.DUAL.MaxFileSize=${hbase.log.maxfilesize}
log4j.appender.DUAL.MaxBackupIndex=${hbase.log.maxbackupindex}
log4j.appender.DUAL.layout=org.apache.log4j.PatternLayout
log4j.appender.DUAL.layout.ConversionPattern=%d{ISO8601} %-5p [%t] %c{2}: %m%n
```

## 13.4 Application Commissioning

### 13.4.1 Commissioning an Application in Windows

### 13.4.1.1 Compiling and Running an Application

#### Scenario

After the code development is complete, you can run the application in the Windows development environment.

If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

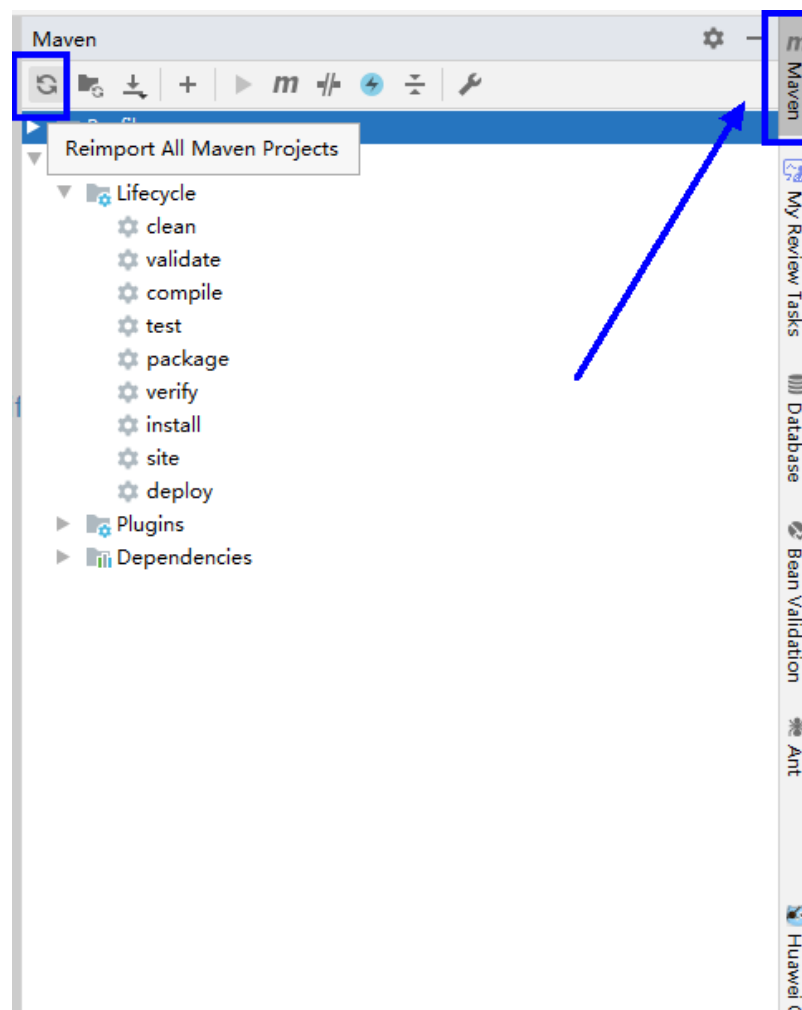
#### NOTE

- If IBM JDK is used in the Windows development environment, the application cannot be run in the Windows environment.
- You need to set the mapping between the host names and IP addresses of the access nodes in the hosts file on the local host where the sample code is run. The host names and IP addresses must be mapped one by one.

#### Procedure

- Step 1** Click **Reimport All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.

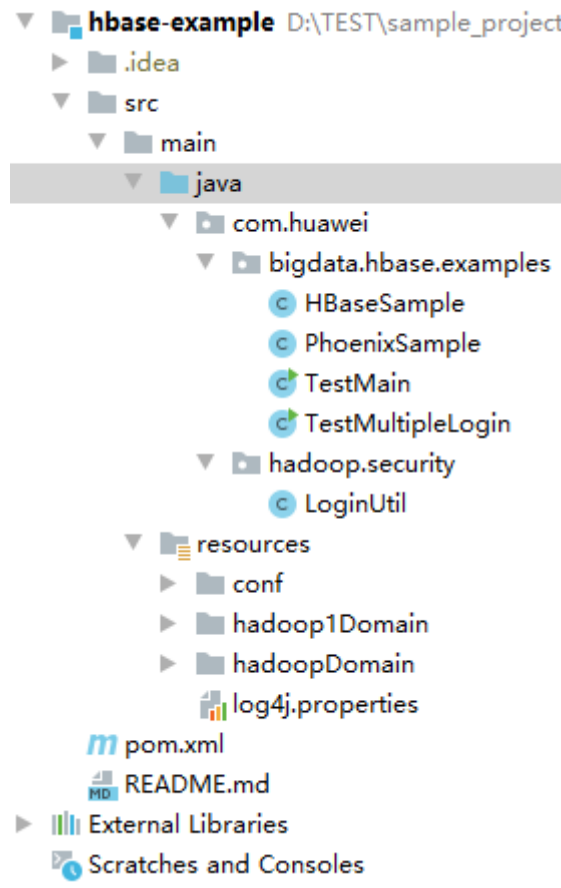
Figure 13-17 reimport projects



**Step 2** Compile and run an application.

Place the configuration file directory and modify the code to match the login user.  
See [Figure 13-18](#).

**Figure 13-18** Directory list of **hbase-example** to be compiled



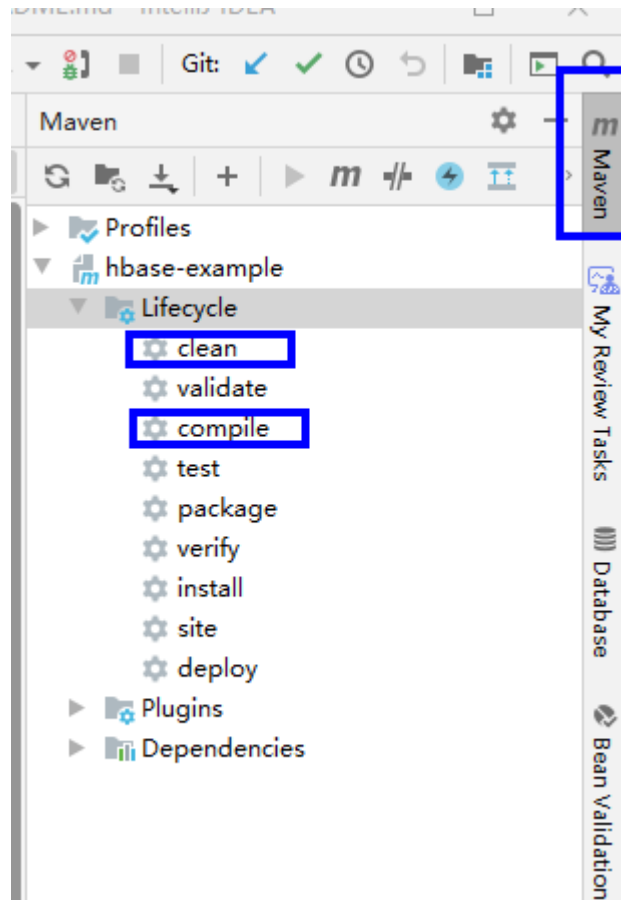
1. Compile an application.

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **compile**, double click **compile** to run the **compile** command of Maven.

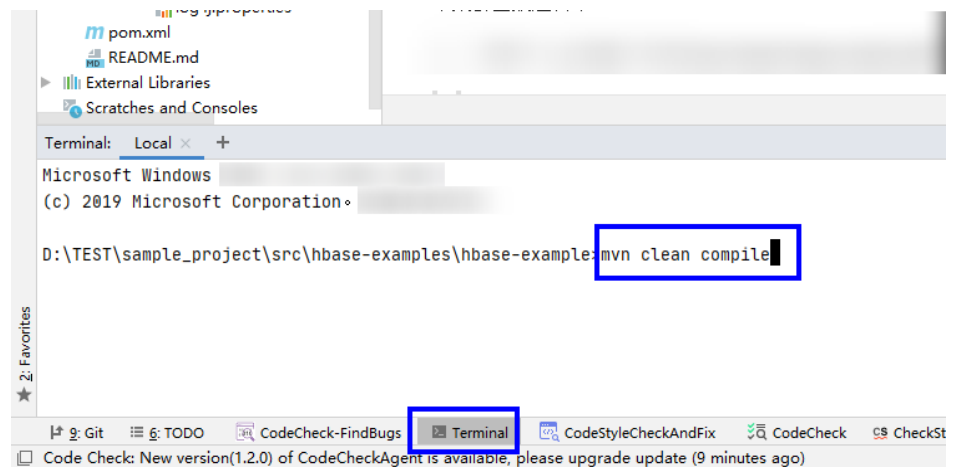
Figure 13-19 clean and compile tools of Maven



- Method 2:

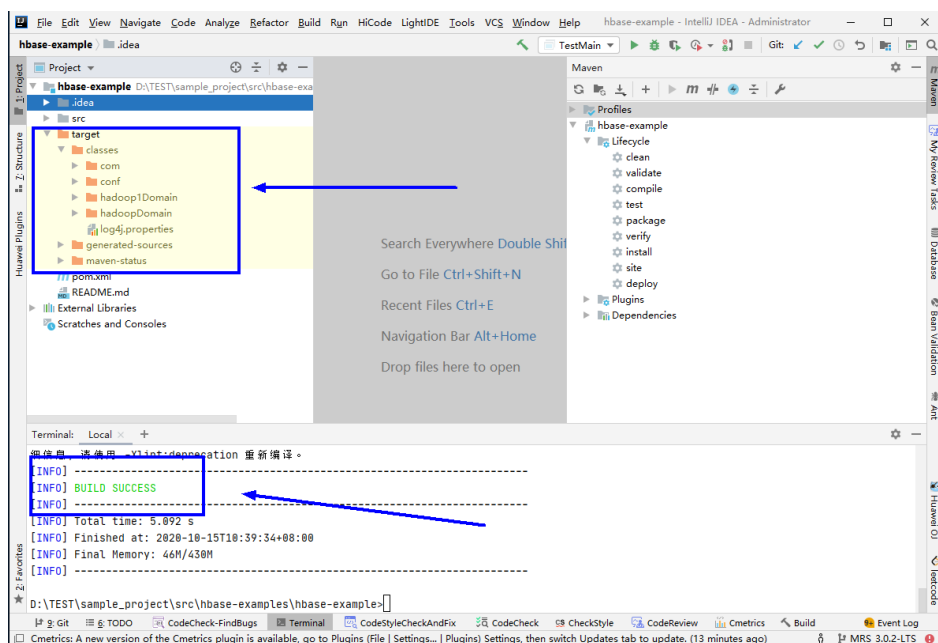
Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

Figure 13-20 Enter mvn clean compile in the IDEA Terminal text box.



After the compilation is complete, the message "Build Success" is displayed and the **target** directory is generated.

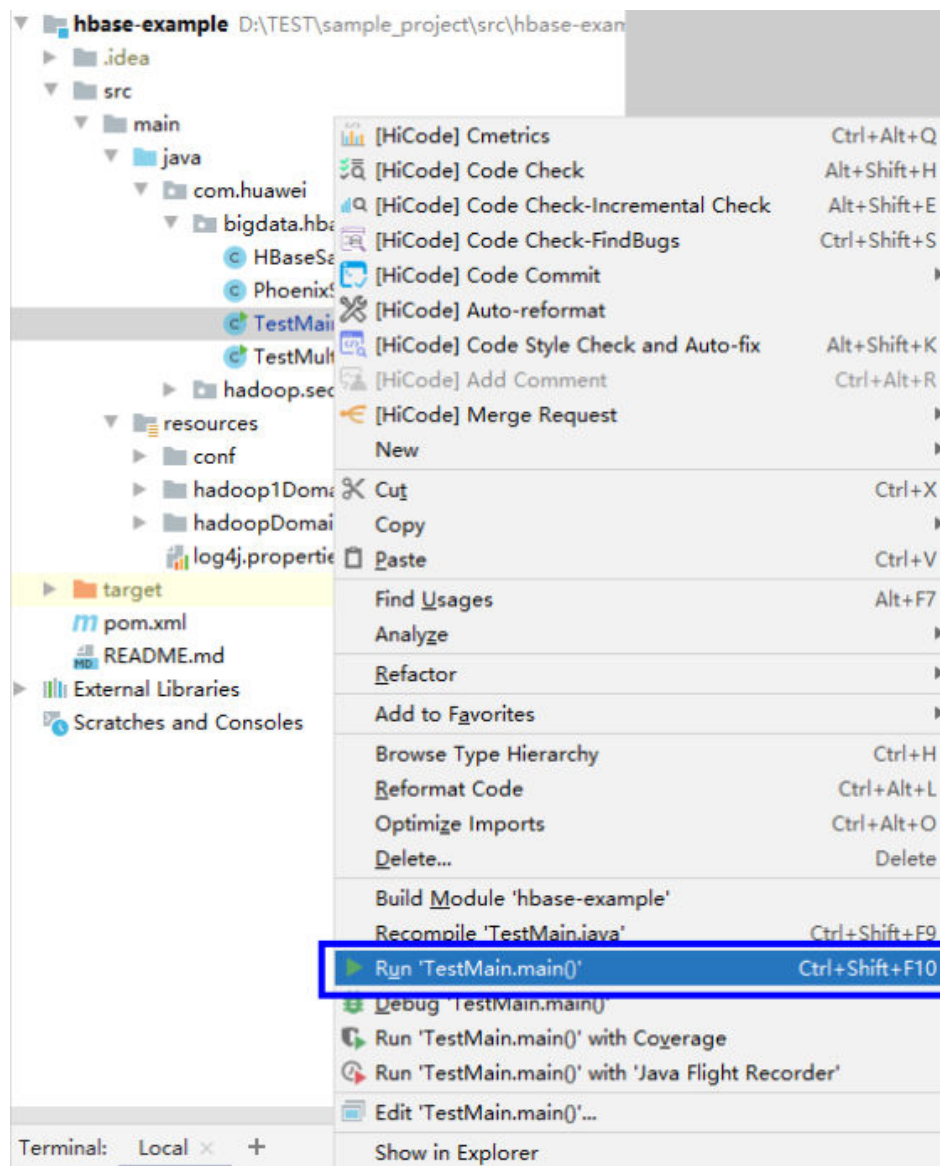
Figure 13-21 Compilation completed



2. Run the program.

Right-click the **TestMain.java** file and choose **RunTestMain.main()** from the shortcut menu.

Figure 13-22 Run the application.



----End

### 13.4.1.2 Viewing Windows Commissioning Results

#### Scenario

After an HBase application is run, you can check the running result through one of the following methods:

- Viewing the IntelliJ IDEA running result.
- Viewing HBase logs.
- Logging in to the HBase WebUI, see **More Information > External Interfaces > WebUI**.
- Using HBase Shell command, see **More Information > External Interfaces > Shell**.

## Procedure

- The following information is displayed in the running results:

```
2016-07-13 14:36:12,736 INFO [main] basic.CreateTableSample: Create table
sampleNameSpace:sampleTable successfully!
2016-07-13 14:36:15,426 INFO [main] basic.ModifyTableSample: Modify table
sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:16,708 INFO [main] basic.MultiSplitSample: Mmulti split table
sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:17,299 INFO [main] basic.PutDataSample: Successfully put 9 items data into
sampleNameSpace:sampleTable.
2016-07-13 14:36:18,992 INFO [main] basic.ScanSample: Scan data successfully.
2016-07-13 14:36:20,532 INFO [main] basic.DeletaDataSample: Successfully delete data from table
sampleNameSpace:sampleTable.
2016-07-13 14:36:21,006 INFO [main] acl.AclSample: Grant ACL for table
sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:27,836 INFO [main] index.CreateIndexSample: Successfully add index for table
sampleNameSpace:sampleTable.
```

### NOTE

In the Windows environment, the following exception occurs but does not affect services.

```
java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.
```

- Log description  
The log level is **INFO** by default and more detailed information can be viewed by changing the log level (**DEBUG**, **INFO**, **WARN**, **ERROR**, and **FATL**). You can modify the **log4j.properties** file to change log levels, for example:

```
hbase.root.logger=INFO,console
...
log4j.logger.org.apache.zookeeper=INFO
#log4j.logger.org.apache.hadoop.fs.FSNamesystem=DEBUG
log4j.logger.org.apache.hadoop.hbase=INFO
Make these two classes DEBUG-level. Make them DEBUG to see more zk debug.
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZKUtil=INFO
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZooKeeperWatcher=INFO
...
```

## 13.4.2 Commissioning an Application in Linux

### 13.4.2.1 Compiling and Running an Application When a Client Is Installed

#### Scenario

In a Linux environment where an HBase client is installed, you can upload the JAR package to the Linux and then run an application after the code development is complete.

#### Prerequisites

- You have installed the HBase client.
- If the host where the client is installed is not a node in the cluster, set the mapping between the host name and the IP address in the **hosts** file on the node where the client locates. The host name and IP address must be in one-to-one mapping.
- The JDK has been installed and Java environment variables have been correctly configured before Windows commissioning and compilation.

## Procedure

### Step 1 Export a JAR package.

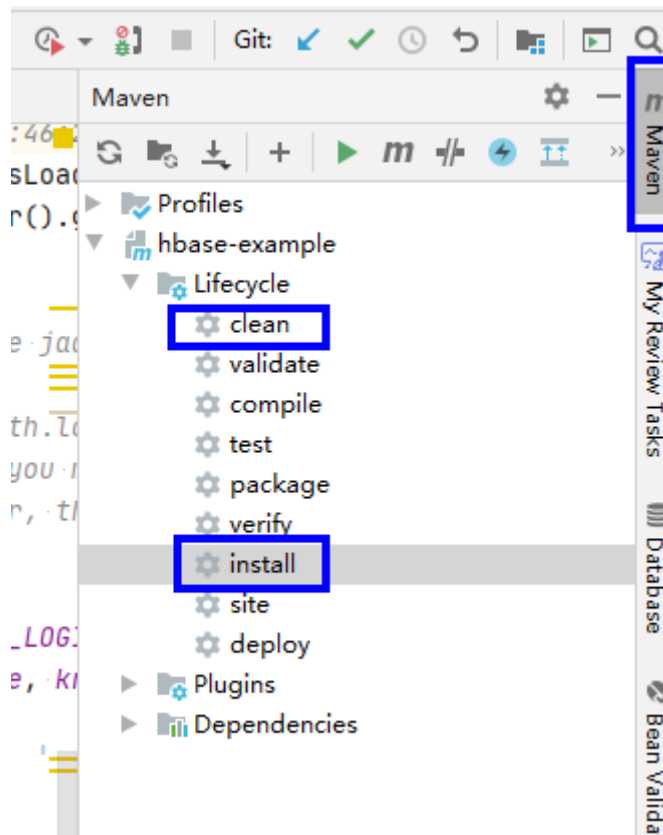
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven** > *Sample projectname* > **Lifecycle** > **clean**, double click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **install**, double click **install** to run the **install** command of Maven.

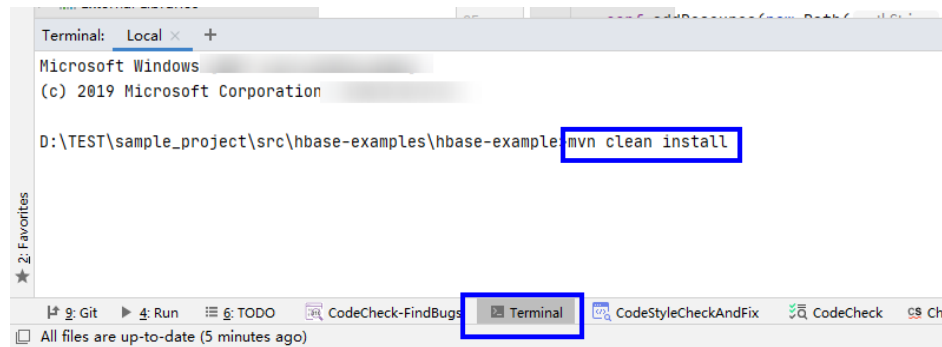
**Figure 13-23** Maven tools: **clean** and **install**



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the **pom.xml** file.

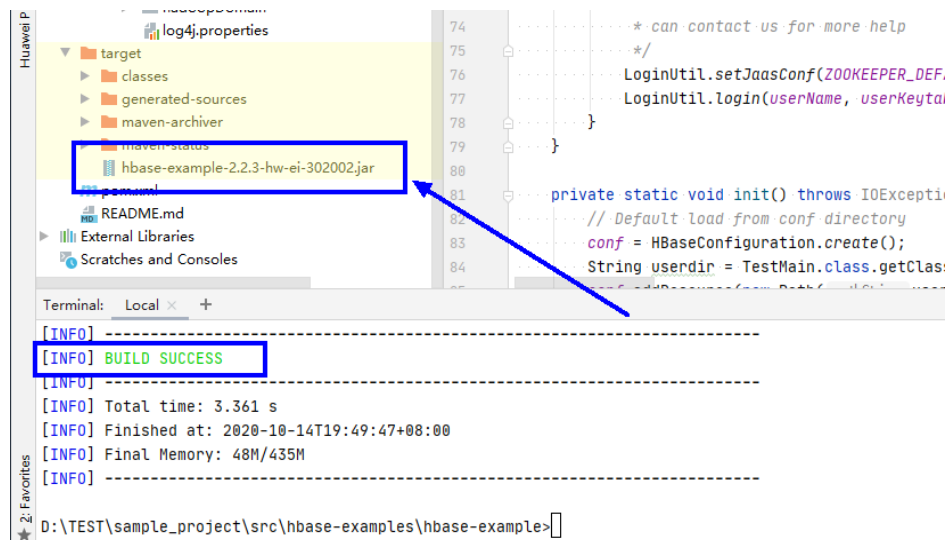


Figure 13-24 Enter `mvn clean compile` in the IDEA Terminal text box.



After the compilation is completed, the message "Build Success" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

Figure 13-25 When the compilation is completed, the JAR file is generated.



**Step 2** Export the JAR file on which the sample project depends.

Access the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of IDEA or by using other command line tools.

Run the command `mvn dependency:copy-dependencies -DoutputDirectory=lib`.

The **lib** folder is generated in the directory where the **pom.xml** file is located. The **lib** folder contains the JAR files on which the sample project depends.

**Step 3** Run the JAR package.

1. Before running the JAR package on the Linux client, run the following command to go to the client directory:

```
cd $BIGDATA_CLIENT_HOME
```

**NOTE**

**\$BIGDATA\_CLIENT\_HOME** is the installation directory of the HBase client.

2. Then run the following command:

```
source bigdata_env
```

 **NOTE**

After the multi-instance function is enabled, run the following command to switch to the client of the specified service instance before performing application development for the HBase service instance.

For example, for HBase2, run the **source /opt/client/HBase2/component\_env** command.

3. Upload the JAR file (non-dependent JAR file) of the sample project generated in the application development environment to the **\$BIGDATA\_CLIENT\_HOME/HBase/hbase/lib** directory in the client running environment. Check the **\$BIGDATA\_CLIENT\_HOME/HBase/hbase/conf** directory based on the sample project **README.md**, and copy the configuration file and authentication file in the sample project to this directory.
4. Go to the **\$BIGDATA\_CLIENT\_HOME/HBase/hbase** directory and run the following command to run the JAR package. The task is complete.

```
hbase com.huawei.bigdata.hbase.examples.TestMain
```

In the preceding command, *hbase com.huawei.bigdata.hbase.examples.TestMain* is used as an example.

----End

## 13.4.2.2 Compiling and Running an Application When No Client Is Installed

### Scenario

In a Linux environment where no HBase client is installed, you can upload the JAR package to the Linux and then run an application after the code development is complete.

### Prerequisites

- A JDK has been installed in the Linux environment. The version of the JDK must be consistent with that of the JDK used by the JAR package exported by IntelliJ IDEA.
- If the Linux host is not a node in the cluster, set the mapping between the host name and the IP address in the **hosts** file on the node. The host name and IP address must be in one-to-one mapping.

### Procedure

**Step 1** Export a JAR package. For details, see [Step 1](#) in section [Compiling and Running an Application When a Client Is Installed](#).

**Step 2** Prepare for the required JAR packages and configuration files.

1. In the Linux environment, create a directory, for example, **/opt/test**, and create subdirectories **lib** and **conf**. Export the JAR package that the sample project depends on. For details about how to export the JAR package, see [Step 2](#) in [1.4.2.1 Compiling and Running an Application When a Client Is](#)

**Installed.** Upload this JAR file and that exported in **Step 1** to the **lib** directory on the Linux server. Upload the **conf** configuration file and authentication file in the sample project to the **conf** directory on Linux.

2. In **/opt/test**, create the **run.sh** script, modify the following content, and save the file:

```
#!/bin/sh
BASEDIR=`cd $(dirname $0);pwd`
cd ${BASEDIR}
for file in ${BASEDIR}/lib/*.jar
do
i_cp=${i_cp}:${file}
echo "$file"
done
for file in ${BASEDIR}/conf/*
do
i_cp=${i_cp}:${file}
done
java -cp .${i_cp} com.huawei.bigdata.hbase.examples.TestMain
```

**Step 3** Go to **/opt/test** and run the following commands to run the JAR packages:

```
sh run.sh
```

```
----End
```

### 13.4.2.3 Viewing Linux Commissioning Results

#### Scenario

After an HBase application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Viewing HBase logs.
- Logging in to the HBase WebUI, see **More Information > External Interfaces > WebUI**.
- Using HBase Shell command, see **More Information > External Interfaces > Shell**.

#### Procedure

The following information is displayed in the running results:

```
2020-07-13 14:36:12,736 INFO [main] basic.CreateTableSample: Create table
sampleNameSpace:sampleTable successful!
2020-07-13 14:36:15,426 INFO [main] basic.ModifyTableSample: Modify table
sampleNameSpace:sampleTable successfully.
2020-07-13 14:36:16,708 INFO [main] basic.MultiSplitSample: Mmulti split table
sampleNameSpace:sampleTable successfully.
2020-07-13 14:36:17,299 INFO [main] basic.PutDataSample: Successfully put 9 items data into
sampleNameSpace:sampleTable.
2020-07-13 14:36:18,992 INFO [main] basic.ScanSample: Scan data successfully.
2020-07-13 14:36:20,532 INFO [main] basic.DeletaDataSample: Successfully delete data from table
sampleNameSpace:sampleTable.
2020-07-13 14:36:21,006 INFO [main] acl.AclSample: Grant ACL for table sampleNameSpace:sampleTable
successfully.
2020-07-13 14:36:27,836 INFO [main] index.CreateIndexSample: Successfully add index for table
sampleNameSpace:sampleTable.
```

## 13.5 More Information

### 13.5.1 SQL Query

#### Function

Phoenix is an intermediate structured query language (SQL) layer built on HBase. Phoenix provides a JDBC driver that can be embedded in a client. The Phoenix query engine converts input SQL statements to one or multiple HBase scans, and compiles and executes the scan tasks to generate a standard JDBC result set.

#### Example Code

- The **hbase-example/conf/hbase-site.xml** file on the client is used to configure the temporary directory for storing query results. If the client program configures the temporary directory in a Linux environment, configure a Linux path. If the client program configures the temporary directory in a Windows environment, configure a Windows path.

```
<property>
 <name>phoenix.spool.directory</name>
 <value>[1] Temporary directory for storing intermediate query results</value>
</property>
```

- JAVA Example: Use the JDBC interface to access HBase.

```
public String getURL(Configuration conf)
{
 String phoenix_jdbc = "jdbc:phoenix";
 String zkQuorum = conf.get("hbase.zookeeper.quorum");
 return phoenix_jdbc + ":" + zkQuorum;
}

public void testSQL()
{
 String tableName = "TEST";
 // Create table
 String createTableSQL = "CREATE TABLE IF NOT EXISTS TEST(id integer not null primary key,
name varchar, account char(6), birth date)";

 // Delete table
 String dropTableSQL = "DROP TABLE TEST";

 // Insert
 String upsertSQL = "UPSERT INTO TEST VALUES(1,'John','100000',
TO_DATE('1980-01-01','yyyy-MM-dd'))";

 // Query
 String querySQL = "SELECT * FROM TEST WHERE id = ?";

 // Create the Configuration instance
 Configuration conf = getConfiguration();

 // Get URL
 String URL = getURL(conf);

 Connection conn = null;
 PreparedStatement preStat = null;
 Statement stat = null;
 ResultSet result = null;

 try
 {
```

```
// Create Connection
conn = DriverManager.getConnection(URL);
// Create Statement
stat = conn.createStatement();
// Execute Create SQL
stat.executeUpdate(createTableSQL);
// Execute Update SQL
stat.executeUpdate(upsertSQL);
// Create PreparedStatement
preStat = conn.prepareStatement(querySQL);
conn.commit();
// Execute query
preStat.setInt(1,1);
result = preStat.executeQuery();
// Get result
while (result.next())
{
 int id = result.getInt("id");
 String name = result.getString(1);
}
}
catch (Exception e)
{
 // handler exception
}
finally
{
 if(null != result){
 try {
 result.close();
 } catch (Exception e2) {
 // handler exception
 }
 }
 if(null != stat){
 try {
 stat.close();
 } catch (Exception e2) {
 // handler exception
 }
 }
 if(null != conn){
 try {
 conn.close();
 } catch (Exception e2) {
 // handler exception
 }
 }
}
}
```

## Precaution

- You need to configure a temporary directory for storing intermediate query results in **hbase-site.xml**. The size of the query result set is restricted by the directory size.
- Phoenix provides most **java.sql** interfaces and follows the ANSI SQL standard.

## 13.5.2 HBase Dual-Read Configuration Items

This section provides the details of all the configurations required for the HBase dual-read feature.

## HBase Dual-Read Operations

**Table 13-6** Configuration items in hbase-dual.xml

Configuration Item	Description	Default Value	Level
hbase.dualclient.active.cluster.configuration.path	HBase client configuration directory of the active cluster	None	Mandatory
hbase.dualclient.standby.cluster.configuration.path	HBase client configuration directory of the standby cluster	None	Mandatory
dual.client.schedule.update.table.delay.second	DR table update interval	5	Optional
hbase.dualclient.glitchtimeout.ms	Maximum glitch time can be tolerated in the active cluster	50	Optional
hbase.dualclient.slow.query.timeout.ms	Slow query alarm log	180000	Optional
hbase.dualclient.active.cluster.id	Active cluster ID	ACTIVE	Optional
hbase.dualclient.standby.cluster.id	Standby cluster ID	STANDBY	Optional
hbase.dualclient.active.executor.thread.max	Maximum size of the thread pool for processing requests to the active cluster	100	Optional
hbase.dualclient.active.executor.thread.core	Core size of the thread pool for processing requests to the active cluster	100	Optional
hbase.dualclient.active.executor.queue	Queue size of the thread pool for processing requests to the active cluster	256	Optional

Configuration Item	Description	Default Value	Level
hbase.dualclient.standby.executor.thread.max	Maximum size of the thread pool for processing requests to the standby cluster	100	Optional
hbase.dualclient.standby.executor.thread.core	Core size of the thread pool for processing requests to the standby cluster	100	Optional
hbase.dualclient.standby.executor.queue	Queue size of the thread pool for processing requests to the standby cluster	256	Optional
hbase.dualclient.clear.executor.thread.max	Maximum size of the thread pool for clearing resources	30	Optional
hbase.dualclient.clear.executor.thread.core	Core size of the thread pool for clearing resources	30	Optional
hbase.dualclient.clear.executor.queue	Queue size of the thread pool for clearing resources	Integer. MAX_VALUE	Optional
dual.client.metrics.enable	Whether to print client metric information	true	Optional
dual.client.schedule.metrics.second	Interval for printing client metric information	300	Optional
dual.client.asynchronous.enable	Whether to asynchronously request the active and standby clusters	false	Optional

## Printing Metric Information

**Table 13-7** Basic specifications

Metric Name	Description	Log level
total_request_count	Total number of queries in a period	INFO
active_success_count	Number of successful queries in the active cluster in a period	INFO
active_error_count	Number of failed queries in the active cluster in a period	INFO
active_timeout_count	Number of query timeouts in the active cluster in a period	INFO
standby_success_count	Number of successful queries in the standby cluster in a period	INFO
standby_error_count	Number of failed queries in the standby cluster in a period	INFO
Active Thread pool	Periodically printed information about the thread pool for processing requests to the active cluster	DEBUG
Standby Thread pool	Periodically printed information about the thread pool for processing requests to the standby cluster	DEBUG
Clear Thread pool	Periodically printed information about the thread pool for releasing resources	DEBUG

**Table 13-8** Histogram indicators for **GET**, **BatchGET**, and **SCAN** requests

Metric Name	Description	Log level
averageLatency(ms)	Average latency	INFO
minLatency(ms)	Minimum latency	INFO



Metric Name	Description	Log level
maxLatency(ms)	Maximum latency	INFO
95thPercentileLatency(ms)	Maximum latency of 95% requests	INFO
99thPercentileLatency(ms)	Maximum latency of 99% requests	INFO
99.9PercentileLatency(ms)	Maximum latency of 99.9% requests	INFO
99.99PercentileLatency(ms)	Maximum latency of 99.99% requests	INFO

## 13.5.3 External Interfaces

### 13.5.3.1 Shell

You can directly perform operations on HBase using Shell on the server. Versions of Shell interfaces of HBase need to be consistent with those in the open-source community. For details, see <http://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/>.

Methods of running the Shell command:

Go to any directory on the HBase client and run the following command:

**hbase shell**

Go to the running mode of the HBase command (also called CLI client connection).

```
hbase(main):001:0>
```

Run the **help** command to obtain the help information of the HBase command parameters.

### Precautions

The **count** command does not support conditional statistics. It supports only full table statistics.

### Command to retrieve HBase replication metrics

All the required metrics will be added for the shell command "status".

- Command to view replication source metrics.

```
hbase(main):019:0> status 'replication', 'source'
```

The output is as follows:

```
version 2.2.3
1 live servers
```

```
BLR1000006595:
SOURCE: PeerID=1, SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0,
LogReadInBytes=1389, LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:44:42 CST 2016, Replication Lag=0 PeerID=3,
SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0, LogReadInBytes=1389,
LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:44:42 CST 2016, Replication Lag=0,
FailedReplicationAttempts=0
```

- Command to view replication sink metrics.

```
hbase(main):020:0> status 'replication', 'sink'
```

The output is as follows:

```
version 2.2.3
1 live servers
BLR1000006595:
SINK : AppliedBatches=0, AppliedOps=0, AppliedHFiles=0, AgeOfLastAppliedOp=0,
TimeStampsOfLastAppliedOp=Wed May 25 17:55:21 CST 2016
```

- Command to view both replication source and replication sink metrics.

```
hbase(main):018:0> status 'replication'
```

The output is as follows:

```
version 2.2.3
1 live servers
BLR1000006595:
SOURCE: PeerID=1, SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0,
LogReadInBytes=1389, LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:43:24 CST 2016, Replication Lag=0 PeerID=3,
SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0, LogReadInBytes=1389,
LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:43:24 CST 2016, Replication Lag=0,
FailedReplicationAttempts=0
SINK : AppliedBatches=0, AppliedOps=0, AppliedHFiles=0, AgeOfLastAppliedOp=0,
TimeStampsOfLastAppliedOp=Wed May 25 17:55:21 CST 2016
```

### 13.5.3.2 Java APIs

#### API Usage Suggestions

- org.apache.hadoop.hbase.Cell rather than org.apache.hadoop.hbase.KeyValue is recommended as the KV data object.
- It is recommended that Connection connection = ConnectionFactory.createConnection(conf) be used to create a connection. The HTablePool is abandoned.
- org.apache.hadoop.hbase.mapreduce rather than org.apache.hadoop.hbase.mapred is recommended.
- You are advised to obtain the HBase client operation object using the getAdmin() method of the constructed Connection object.

#### Common HBase APIs

Common HBase Java classes are as follows:

- Interface class Admin: It is the core class of HBase client applications, which encapsulates APIs for HBase management, such as table creation and table deletion. For details, see [Table 13-9](#).

- Interface class Table: It is a class for HBase read/write, which encapsulates APIs for reading and writing HBase tables. For details, see [Table 13-10](#).

**Table 13-9** org.apache.hadoop.hbase.client.Admin

Method	Description
boolean tableExists(final TableName tableName)	This method is used to check whether a specified table exists. If the table exists in the <b>hbase:meta</b> table, <b>true</b> is returned. Otherwise, <b>false</b> is returned.
HTableDescriptor[] listTables(String regex)	This method is used to view the user table that complies with the specified regular expression format. There are two other overload methods, one with the input parameter type of Pattern, and the other with the empty input parameter. When the input parameter is empty, all user tables are queried by default.
HTableDescriptor[] listTables(final Pattern pattern, final boolean includeSysTables)	The function of this method is similar to that of the previous method. Users can use this method to specify whether the returned result contains the system table. The previous method returns only the user table.
TableName[] listTableNames(String regex)	This method is used to view the user table that complies with the specified regular expression format. There are two other overload methods, one with the input parameter type of Pattern, and the other with the empty input parameter. When the input parameter is empty, all user tables are queried by default. The function of this method is similar to that of listTables. The only difference is that this method returns TableName[].
TableName[] listTableNames(final Pattern pattern, final boolean includeSysTables)	The function of this method is similar to that of the previous method. Users can use this method to specify whether the returned result contains the system table. The previous method returns only the user table.
void createTable(HTableDescriptor desc)	This method is used to create a table with only one region.

Method	Description
void createTable(HTableDescriptor desc, byte[] startKey, byte[] endKey, int numRegions)	This method is used to create a table with a specified number of regions. The endKey of the first region is the StartKey, and the StartKey of the last region is the endKey. If there are a large number of regions, the invoking of this method may time out.
void createTable(final HTableDescriptor desc, byte[][] splitKeys)	This method is used to create a table. The number of regions in the table and the StartKey of each region are determined by splitKeys. If there are a large number of regions, the invoking of this method may time out.
void createTable(final HTableDescriptor desc, final byte[][] splitKeys)	This method is used to create a table. The number of regions in the table and the StartKey of each region are determined by splitKeys. This method uses asynchronous invoking, and does not wait for the created table to be online.
void deleteTable(final TableName tableName)	This method is used to delete a specified table.
public void truncateTable(final TableName tableName, final boolean preserveSplits)	This method is used to re-create a specified table. If the second parameter is set to <b>true</b> , the region of the reconstructed table is the same as that of the previous table. Otherwise, there is only one region.
void enableTable(final TableName tableName)	This method is used to enable a specified table. If there are a large number of regions in a table, the invoking of this method may time out.
void enableTableAsync(final TableName tableName)	This method is used to enable a specified table. This method uses asynchronous invoking, and does not wait for all regions to be online.
void disableTable(final TableName tableName)	This method is used to disable a specified table. If there are a large number of regions in a table, the invoking of this method may time out.
void disableTableAsync(final TableName tableName)	This method is used to disable a specified table. This method uses asynchronous invoking, and does not wait for all regions to be offline.

Method	Description
boolean isTableEnabled(Table Name tableName)	This method is used to check whether a table is enabled. This method can be used with the enableTableAsync method to check whether the operation of enabling a table is complete.
boolean isTableDisabled(Table Name tableName)	This method is used to check whether a table is disabled. This method can be used with the disableTableAsync method to check whether the operation of disabling a table is complete.
void addColumn(final Table Name tableName, final HColumnDescriptor column)	This method is used to add a column family to a specified table.
void deleteColumn(final Table Name tableName, final HColumnDescriptor column)	This method is used to delete a specified column family from a specified table.
void modifyColumn(final Table Name tableName, final HColumnDescriptor column)	This method is used to modify a specified column family.

**Table 13-10** org.apache.hadoop.hbase.client.Table

Method	Description
boolean exists(Get get)	This method is used to check whether the specified rowkey exists in the table.
boolean[] existsAll(List<Get> gets)	This method is used to check whether the specified rowkey exists in the table. The returned Boolean array result corresponds to the input parameter position.
Result get(Get get)	This method is used to read data based on the specified RowKey.
Result[] get(List<Get> gets)	This method is used to read data in batches by specifying a batch of RowKeys.
ResultScanner getScanner(Scan scan)	This method is used to obtain a scanner object in the table. The query parameters can be specified by the input parameter scan, including <b>StartRow</b> , <b>StopRow</b> , and <b>caching</b> .

Method	Description
void put(Put put)	This method is used to write a data record to the table.
void put(List<Put> puts)	This method is used to write a batch of data records to the table.
void close()	This method is used to release the resources held by the table object.

 NOTE

[Table 13-9](#) and [Table 13-10](#) list only some common methods. F

### 13.5.3.3 SQLLine

You can run **sqlline.py** on the server to perform SQL operations on HBase. The SQLLine APIs of Phoenix are the same as those of the open-source community. For details, see <http://phoenix.apache.org/>.

[Table 13-11](#) lists common SQLLine syntax, [Table 13-12](#) lists common functions, and [Phoenix Command Line](#) describes how to use commands.

**Table 13-11** Common syntax

Command	Description	Example
<b>CREATE TABLE</b>	Creates a table.	CREATE TABLE MY_TABLE(id BIGINT not null primary key, name VARCHAR);
<b>ALTER</b>	Alters a table or a view.	ALTER TABLE MY_TABLE DROP COLUMN name;
<b>DROP TABLE</b>	Deletes a table.	DROP TABLE MY_TABLE;
<b>UPSERT VALUES</b>	Inserts or changes data.	UPSERT INTO MY_TABLE VALUES(1,'abc');
<b>SELECT</b>	Queries data.	SELECT * FROM MY_TABLE;
<b>CREATE INDEX</b>	Creates a global index.	CREATE INDEX MY_IDX ON MY_TABLE(name);
<b>CREATE LOCAL INDEX</b>	Creates a local index.	CREATE LOCAL INDEX MY_LOCAL_IDX ON MY_TABLE(id,name);
<b>ALTER INDEX</b>	Changes the index status.	ALTER INDEX MY_IDX ON MY_TABLE DISABLE;
<b>DROP INDEX</b>	Deletes an index.	DROP INDEX MY_IDX ON MY_TABLE;

Command	Description	Example
<b>EXPLAIN</b>	Displays an execution plan.	EXPLAIN SELECT name FROM MY_TABLE;
<b>CREATE SEQUENCE</b>	Creates a sequence.	CREATE SEQUENCE MY_SEQUENCE;
<b>DROP SEQUENCE</b>	Deletes a sequence.	DROP SEQUENCE MY_SEQUENCE;
<b>CREATE VIEW</b>	Creates a view.	CREATE VIEW MY_VIEW AS SELECT * FROM MY_TABLE;
<b>DROP VIEW</b>	Deletes a view.	DROP VIEW MY_VIEW;
<b>CREATE SCHEMA</b>	Creates a schema.	CREATE SCHEMA MY_SCHEMA;
<b>USE</b>	Modifies the default schema.	USE MY_SCHEMA;
<b>DROP SCHEMA</b>	Deletes a schema.	DROP SCHEMA MY_SCHEMA;

**Table 13-12** Common functions

Function Type	Function	Description	Example
String functions	<b>SUBSTR</b>	Extracts a part of a string.	SUBSTR(['Hello'], 2, 5)
	<b>INSTR</b>	Queries the position of the first occurrence of a string in another string.	INSTR('Hello World', 'World')
	<b>TRIM</b>	Removes whitespaces from both ends of a string.	TRIM(' Hello ')
	<b>LTRIM</b>	Removes whitespaces found on the left-hand side of the string.	LTRIM(' Hello')

Function Type	Function	Description	Example
	<b>RTRIM</b>	Removes whitespaces found on the right-hand side of the string.	RTRIM('Hello ')
	<b>LPAD</b>	Left-pads a string with another string.	LPAD('John',30,'*')
	<b>LENGTH</b>	Gets the length of a given string.	LENGTH('Hello')
	<b>UPPER</b>	Converts all the characters in a string to uppercase characters.	UPPER('Hello')
	<b>LOWER</b>	Converts all the characters in a string to lowercase characters.	LOWER('HELLO')
	<b>REVERSE</b>	Reverses a string.	REVERSE('Hello')
	<b>REGEXP_SPLIT</b>	Splits a string using a regular expression.	REGEXP_SPLIT('ONE,TWO,THREE',',')
	<b>REGEXP_REPLACE</b>	Returns a string by applying a regular expression and replacing the matches with the replacement string.	REGEXP_REPLACE('abc123ABC','[0-9]+',' #')
	<b>REGEXP_SUBSTR</b>	Returns a substring from a string based on a regular expression pattern.	REGEXP_SUBSTR('na1-appsrv35-sj35','[^-]+')
Aggregate functions	<b>AVG</b>	Gets the average value.	AVG(X)



Function Type	Function	Description	Example
	<b>COUNT</b>	Gets the number of data records.	COUNT(*)
	<b>MAX</b>	Gets the maximum value.	MAX(NAME)
	<b>MIN</b>	Gets the minimum value.	MIN(NAME)
	<b>SUM</b>	Gets the sum of all values.	SUM(X)
	<b>STDDEV_POP</b>	Gets the population standard deviation of all values.	STDDEV_POP( X )
	<b>STDDEV_SAMP</b>	Gets the sample standard deviation of all values.	STDDEV_SAMP( X )
	<b>NTH_VALUE</b>	Gets the Nth value in each distinct group ordered according to the <b>ORDER BY</b> specification.	NTH_VALUE( name, 2 ) WITHIN GROUP (ORDER BY salary DESC)
Time and date functions	<b>NOW</b>	Returns the current date.	NOW()
	<b>CURRENT_TIME</b>	Returns the current time.	CURRENT_TIME()
	<b>CURRENT_DATE</b>	Returns the current date.	CURRENT_DATE()
	<b>TO_DATE</b>	Parses a string and returns a date.	TO_DATE('1970-01-01', 'yyyy-MM-dd', 'GMT+1')
	<b>TO_TIME</b>	Converts the given string into a <b>TIME</b> instance.	TO_TIME('1970-01-01', 'yyyy-MM-dd', 'GMT+1')
	<b>TO_TIMESTAMP</b>	Converts the given string into a <b>TIMESTAMP</b> instance.	TO_TIMESTAMP('1970-01-01', 'yyyy-MM-dd', 'GMT+1')

Function Type	Function	Description	Example
	<b>YEAR</b>	Returns the year of the specified date.	YEAR(TO_DATE('2015-6-05'))
	<b>MONTH</b>	Returns the month of the specified date.	MONTH(TO_TIMESTAMP('2015-6-05'))
	<b>WEEK</b>	Returns the week of the specified date.	WEEK(TO_TIME('2010-6-15'))
	<b>HOUR</b>	Returns the hour of the specified date.	HOUR(TO_TIMESTAMP('2015-6-05'))
	<b>MINUTE</b>	Returns the minute of the specified date.	MINUTE(TO_TIME('2015-6-05'))
	<b>SECOND</b>	Returns the second of the specified date.	SECOND(TO_DATE('2015-6-05'))
Numeric functions	<b>ROUND</b>	Rounds the numeric or timestamp expression to the nearest scale or time unit specified.	ROUND(2.56)
	<b>CEIL</b>	Rounds any fractional value up to the next even multiple.	CEIL(2.34)
	<b>FLOOR</b>	Rounds any fractional value down to the previous even multiple.	FLOOR(2.34)
	<b>TRUNC</b>	Rounds any fractional value down to the previous even multiple (same as <b>FLOOR</b> ).	TRUNC(2.34)

Function Type	Function	Description	Example
	<b>TO_NUMBER</b>	Formats a string as a number.	TO_NUMBER('-123.33')
	<b>RAND</b>	Returns a random number.	RAND()
Math functions	<b>ABS</b>	Returns the absolute value of the given numeric expression.	ABS(-1)
	<b>SQRT</b>	$\sqrt{x}$	SQRT(1.1)
	<b>EXP</b>	$e^x$	EXP(-1)
	<b>POWER</b>	$x^y$	POWER(2, 3)
	<b>LN</b>	$\ln(x)$	LN(3)
	<b>LOG</b>	$\log_x(y)$	LOG(2, 3)
Array functions	<b>ARRAY_ELEM</b>	Uses the array subscript notation to access an array element.	ARRAY_ELEM(ARRAY[1,2,3], 1)
	<b>ARRAY_PREPEND</b>	Appends the given element to the beginning of the array.	ARRAY_APPEND(ARRAY[1,2,3], 4)
	<b>ARRAY_CAT</b>	Concatenates the input arrays and returns the result.	ARRAY_CAT(ARRAY[1,2], ARRAY[3,4])
	<b>ARRAY_FILL</b>	Fills an array with values.	ARRAY_FILL(1, 3)

Function Type	Function	Description	Example
	<b>ARRAY_TO_STRING</b>	Converts the elements of an array to a string and joins them using the specified delimiter.	ARRAY_TO_STRING(ARRAY['a','b','c'], ',')
	<b>ANY</b>	Used on the right-hand side of a comparison expression to test that any array element satisfies the comparison expression against the left-hand side.	10 > ANY(my_array)
	<b>ALL</b>	Used on the right-hand side of a comparison expression to test that all array elements satisfy the comparison expression against the left-hand side.	10 > ALL(my_array)
Other functions	<b>MD5</b>	Computes the MD5 hash of the argument.	MD5(my_column)
	<b>ENCODE</b>	Encodes the expression according to the encoding format provided.	ENCODE(myNumber, 'BASE62')
	<b>DECODE</b>	Decodes the expression according to the encoding format provided.	DECODE('000000008512af277fff8', 'HEX')

### 13.5.3.4 JDBC APIs

Phoenix implements most **java.sql** interfaces. The SQL syntax follows the ANSI SQL standard.

For details about the functions that are supported, visit:

<http://phoenix.apache.org/language/functions.html>

For details about the syntax that is supported, visit:

<http://phoenix.apache.org/language/index.html>

### 13.5.3.5 WebUI

#### Scenario

The web user interface (WebUI) displays the HBase cluster status, including summary of a cluster and information about RegionServer, Master, snapshots, and running processes. You can determine the HBase cluster status based on the information displayed on the WebUI.

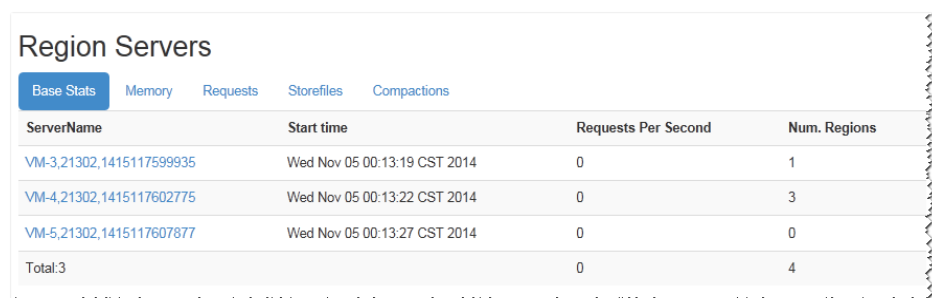
#### NOTE

Contact the administrator to obtain a service account that has the permission to access the WebUI and obtain its password.

#### Procedure

1. **Log in to the FusionInsight Manager portal.** Choose **Cluster > Name of the desired cluster > Services > HBase > HMaster(Active)** to open the HBase WebUI.
2. On the home page of the HBase WebUI, view the HBase summary. The following information is included.
  - a. On the **Region Servers** page, view basic information about the RegionServer, as shown in **Figure 13-26**.

**Figure 13-26** Region Servers



Region Servers				
Base Stats	Memory	Requests	Storefiles	Compactions
ServerName	Start time	Requests Per Second	Num. Regions	
VM-3,21302,1415117599935	Wed Nov 05 00:13:19 CST 2014	0	1	
VM-4,21302,1415117602775	Wed Nov 05 00:13:22 CST 2014	0	3	
VM-5,21302,1415117607877	Wed Nov 05 00:13:27 CST 2014	0	0	
Total:3		0	4	

- b. On the **Backup Masters** page, view information about the Backup Master, as shown in **Figure 13-27**.

**Figure 13-27 Backup Masters**

ServerName	Port	Start Time
VM-4	21300	Wed Nov 05 00:13:08 CST 2014
Total: 1		

- c. On the **Tables** page, view HBase table information, including **User Tables**, **Catalog Tables**, and **Snapshots**, as shown in **Figure 13-28**.

**Figure 13-28 Tables**

Table Name	Online Regions	Description
t	1	t', {NAME => 'd'}

- d. On the **Tasks** page, view information about tasks running on HBase, including the start time and status, as shown in **Figure 13-29**.

**Figure 13-29 Tasks**

Start Time	Description	State	Status
Wed Nov 05 03:06:35 CST 2014	Closing region availabilityCheck_VM-5_1415127943,,1415127994683.1d82d088fb4ba672ee2235fd98ae695.	COMPLETE (since 44sec ago)	Closed (since 44sec ago)
Wed Nov 05 00:20:13 CST 2014	RpcServer.reader=0,port=21300	WAITING (since 2mins, 36sec ago)	Waiting for a call (since 2mins, 36sec ago)
Wed Nov 05 00:19:49 CST 2014	RpcServer.reader=9,port=21300	WAITING (since 2mins, 53sec ago)	Waiting for a call (since 2mins, 53sec ago)
Wed Nov 05 00:19:17 CST 2014	RpcServer.reader=8,port=21300	WAITING (since 3mins, 8sec ago)	Waiting for a call (since 3mins, 8sec ago)
Wed Nov 05 00:15:10 CST 2014	RpcServer.reader=7,port=21300	WAITING (since 3mins, 46sec ago)	Waiting for a call (since 3mins, 46sec ago)
Wed Nov 05 00:14:52 CST 2014	RpcServer.reader=6,port=21300	WAITING (since 5mins, 2sec ago)	Waiting for a call (since 5mins, 2sec ago)
Wed Nov 05 00:14:36 CST 2014	RpcServer.reader=5,port=21300	WAITING (since 10sec ago)	Waiting for a call (since 10sec ago)
Wed Nov 05 00:14:01 CST 2014	RpcServer.reader=4,port=21300	WAITING (since 0sec ago)	Waiting for a call (since 0sec ago)

- 3. On the **Table Details** page of the HBase WebUI, view the summary of HBase storage tables, as shown in **Figure 13-30**.



Figure 13-32 HBase Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
- <configuration>
 - <property>
 <name>dfs.journalnode.rpc-address</name>
 <value>0.0.0.0:8485</value>
 <source>hdfs-default.xml</source>
 </property>
 - <property>
 <name>io.storefile.bloom.block.size</name>
 <value>131072</value>
 <source>hbase-default.xml</source>
 </property>
 - <property>
 <name>hbase.sessioncontrol.maxSessions</name>
 <value>65535</value>
 <source>hbase-site.xml</source>
 </property>
 - <property>
 <name>yarn.ipc.rpc.class</name>
 <value>org.apache.hadoop.yarn.ipc.HadoopYarnProtoRPC</value>
 <source>yarn-default.xml</source>
 </property>
 - <property>
 <name>mapreduce.job.maxtaskfailures.per.tracker</name>
 <value>3</value>
 <source>mapred-default.xml</source>
 </property>
 - <property>
 <name>hbase.rest.threads.min</name>
 <value>2</value>
 <source>hbase-default.xml</source>
 </property>
 - <property>
 <name>hbase.rs.cacheblocksonwrite</name>
 <value>>false</value>
 <source>hbase-default.xml</source>
 </property>
 - <property>
 <name>ha.health-monitor.connect-retry-interval.ms</name>
 <value>1000</value>
 <source>core-default.xml</source>
 </property>
 - <property>
 <name>yarn.resource-manager.work-preserving-recovery.enabled</name>
 <value>>false</value>
 <source>yarn-default.xml</source>
 </property>
 - <property>
 <name>dfs.client.mmap.cache.size</name>
 <value>256</value>
 <source>hdfs-default.xml</source>
 </property>
 - <property>
```

## 13.5.4 HBase Access Configuration on Windows Using EIPs

### Scenario

This section describes how to bind Elastic IP addresses (EIPs) to a cluster and configure HBase files so that sample files can be compiled locally.

This section uses hbase-example as an example.

### Procedure

- Step 1** Apply for an EIP for each node in the cluster and add public IP addresses and corresponding host domain names of all nodes to the Windows local **hosts** file. (If a host name contains uppercase letters, change them to lowercase letters.)



1. On the VPC console, apply for EIPs (the number of EIPs you buy should be equal to the number of nodes in the cluster), click the name of each node in the MRS cluster, and bind an EIP to each node on the **EIPs** page.  
For details, see **Virtual Private Cloud > User Guide > EIP > Assigning an EIP and Binding It to an ECS**.
2. Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.

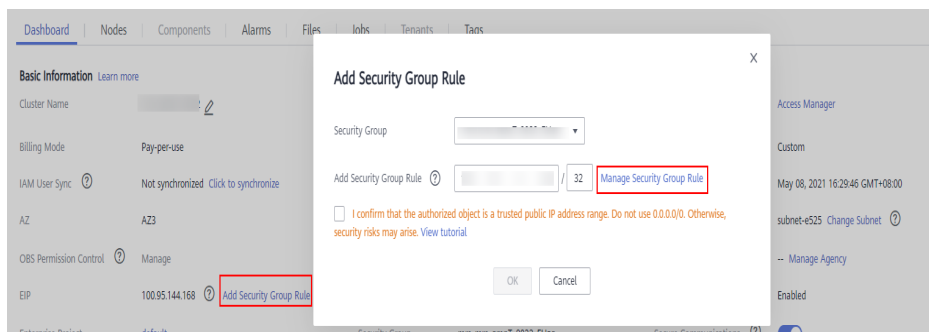
```

1 Mapping between public IP addresses and private IP addresses
2 100.95.144.168 172.16.0.120
3 100.95.144.168 172.16.0.42
4 100.93.144.168 172.16.0.62
5 100.95.144.168 172.16.0.200
6 100.93.144.168 172.16.0.139
7 100.93.144.168 172.16.0.214
8
9 hosts file in the cluster
10 172.16.0.120 node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
11 172.16.0.42 node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
12 172.16.0.62 node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
13 172.16.0.200 node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
14 172.16.0.139 node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
15 172.16.0.214 node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.
16
17 hosts file that should be added to Windows
18 100.95.144.168 node-group-1xzi00002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
19 100.95.144.168 node-master3vint.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
20 100.93.144.168 node-group-1xzi00003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
21 100.95.144.168 node-mastericeip.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
22 100.93.144.168 node-group-1xzi00001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
23 100.93.144.168 node-master2pvnuead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.

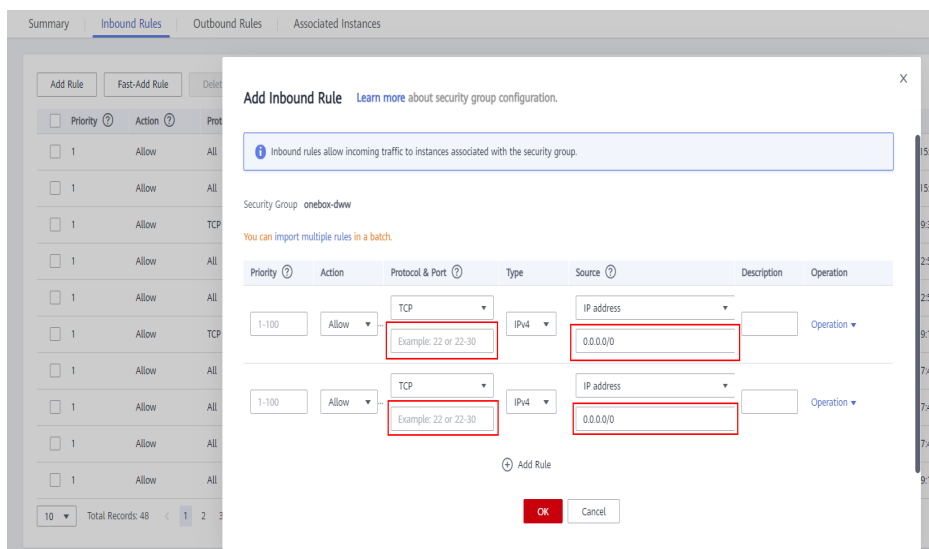
```

**Step 2** Configure security group rules for the cluster.

1. On the **Dashboard** page, choose **Add Security Group Rule > Manage Security Group Rule**.



2. On the **Inbound Rules** page, click **Add Rule**. On the displayed **Add Inbound Rule** page, configure Windows IP addresses and ports **21730TCP**, **21731TCP/UDP**, and **21732TCP/UDP**.



- Step 3** On Manager, choose **Cluster > Services > HBase > More > Download Client**, and copy the **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml** files on the client to the **resources** directory of the sample project.

----End

## 13.5.5 Phoenix Command Line

Phoenix supports SQL statements to operate HBase. The following describes how to use SQL statements to create tables, insert data, query data, and delete tables.

### Prerequisites

The HBase client has been installed. For example, the client has been installed in the **/opt/client** directory. The client directory in the following operations is only an example. Change it based on the actual installation directory onsite. Before using the client, download and update the client configuration file, and ensure that the active management node of Manager is available.

### Procedure

- Step 1 Optional:** Log in to the node where the client is installed as the client installation user.

Access the HBase client installation directory:

```
cd /opt/client
```

- Step 2** Run the following command to configure environment variables:

```
source bigdata_env
```

- Step 3** If Kerberos authentication is enabled for the current cluster, run the following command to authenticate the current user. The current user must have the permission to create HBase tables. If Kerberos authentication is disabled for the current cluster, skip this step:

```
kinit MRS cluster user
```

For example, **kinit hbaseuser**.

- Step 4** Running Commands on the Phoenix Client.

```
sqlline.py
```

- Step 5** Create a table:

```
CREATE TABLE TEST (id VARCHAR PRIMARY KEY, name VARCHAR);
```

- Step 6** Insert data:

```
UPSERT INTO TEST(id,name) VALUES ('1','jamee');
```

- Step 7** Query data:

```
SELECT * FROM TEST;
```

- Step 8** Deleting a table:

```
DROP TABLE TEST;
```

- Step 9** Exit the Phoenix CLI:

**!quit**

**----End**

## 13.5.6 FAQs

### 13.5.6.1 How to Rectify the Fault When an Exception Occurs During the Running of an HBase-developed Application and "org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory" Is Displayed in the Error Information?

**Step 1** Check whether the **hbase.rpc.controllerfactory.class** configuration item is included in the **hbase-site.xml** configuration file.

```
<name>hbase.rpc.controllerfactory.class</name>
<value>org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory</value>
```

**Step 2** If this configuration item is included in the current application development project, you need to import the **phoenix-core-\*.jar** package. You can obtain this package from **HBase/hbase/lib** in the HBase client installation directory.

**Step 3** If you do not import this JAR package, you need to delete **hbase.rpc.controllerfactory.class** from the **hbase-site.xml** configuration file.

**----End**

### 13.5.6.2 What Are the Application Scenarios of the bulkload and put Data-loading Modes?

#### Question

Both the bulkload and put data-loading modes can be used to load data to HBase. Though the bulkload mode loads data faster than the put mode, the bulkload mode has its own disadvantages. The following describes the application scenarios of these two data-loading modes.

#### Answer

The bulkload starts MapReduce tasks to generate HFile files, and then registers HFile files with HBase. Incorrect use of the bulkload mode will consume more cluster memory and CPU resources due to started MapReduce tasks. The large number of HFile files may frequently trigger Compaction, decreasing the query speed drastically.

Incorrect use of the put mode may cause a slow data loading rate. If the memory allocated to RegionServer is not sufficient, the process may exit.

The application scenarios of the bulkload and put modes are as follows:

- bulkload:
  - Load a large amount of data to HBase in the one-off manner.
  - Load data to HBase with low reliability requirements and without generating WAL files.

- Low loading and query speed if the put mode is used.
- The size of the HFile generated after data loading is similar to the size of HDFS block.
- put:
  - The size of the data loaded to one Region at a time is smaller than half the size of an HDFS block.
  - Load data to HBase in real time.
  - The query speed does not decrease wildly during data loading.

### 13.5.6.3 An Error Occurred When Building a JAR Package

#### Question

When the sample code is compiled using Maven to build a JAR package, the error message "Could not transfer artifact org.apache.commons:commons-crypto:pom:\${commons-crypto.version}" is displayed and the package building fails.

#### Answer

The hbase-common module depends on commons-crypto. In the **pom.xml** file of hbase-common, the **`\${commons-crypto.version}`** variable is used to introduce commons-crypto. The parsing logic of this variable is as follows: If the OS is AArch64, the value is **1.0.0-hw-aarch64**. If the OS is x86\_64, the value is **1.0.0**. If Maven fails to parse the variable using the OS due to incorrect configuration in the compilation environment, you can manually modify the **pom.xml** file to prevent correct compilation.

In the **pom.xml** file, manually change the dependency of the hbase-common module to the following to exclude the commons-crypto dependency:

```
<dependency>
 <groupId>org.apache.hbase</groupId>
 <artifactId>hbase-common</artifactId>
 <version>${hbase.version}</version>
 <exclusions>
 <exclusion>
 <groupId>org.apache.commons</groupId>
 <artifactId>commons-crypto</artifactId>
 </exclusion>
 </exclusions>
</dependency>
```

Manually add the commons-crypto dependency of the specified version. Enter a correct version based on the OS architecture (x86\_64 or AArch64).

```
<dependency>
 <groupId>org.apache.commons</groupId>
 <artifactId>commons-crypto</artifactId>
 <version>1.0.0</version>
</dependency>
```

# 14 HDFS Development Guide (Security Mode)

---

## 14.1 Overview

### 14.1.1 Introduction to HDFS

#### Introduction to HDFS

Hadoop distribute file system (HDFS) is a distributed file system with high fault tolerance. HDFS supports data access with high throughput and applies to processing of large data sets.

HDFS applies to the following application scenarios:

- Massive data processing (higher than the TB or PB level).
- Scenarios that require high throughput.
- Scenarios that require high reliability.
- Scenarios that require good scalability.

#### Introduction to HDFS Interface

HDFS can be developed by using Java language. For details of API Reference, see [Java API](#).

### 14.1.2 Basic Concepts

#### DataNode

A DataNode is used to store data blocks of each file and periodically report the storage status to the NameNode.

## NameNode

A NameNode is used to manage the namespace, directory structure, and metadata information of a file system and provide the backup mechanism. NameNodes are classified into the following two types:

- Active NameNode: manages the file system namespace, maintains the directory structure tree and metadata information of a file system, and records the relationship between each data block and the file to which the data block belongs.
- Standby NameNode: Data in a standby NameNode is synchronous with those in an active NameNode. A standby NameNode takes over services from the active NameNode if the active NameNode is exception.

## JournalNode

A JournalNode synchronizes metadata between the active and standby NameNodes in the High Availability (HA) cluster.

## ZKFC

ZKFC must be deployed for each NameNode. It is responsible for monitoring NameNode status and writing status information to the ZooKeeper. ZKFC also has permission to select the active NameNode.

## Colocation

Colocation is used to store associated data or the data to be associated on the same storage node. The HDFS Colocation stores files to be associated on a same data node so that data can be obtained from the same data node during associated operations. This greatly reduces network bandwidth consumption.

## Client

The HDFS can be accessed from the Java application programming interface (API), C API, Shell, HTTP REST API and web user interface (WebUI). For details, see [Common API Introduction](#) and [Shell Command Introduce](#).

- JAVA API  
Provides an application interface for the HDFS. This guide describes how to use the Java API to develop HDFS applications.
- C API  
Provides an application interface for the HDFS. This guide describes how to use the C API to develop HDFS applications.
- Shell  
Provides shell commands to perform operations on the HDFS.
- HTTP REST API  
Additional interfaces except Shell, Java API and C API. You can use the interfaces to monitor HDFS status.
- WEB UI  
Provides a visualized management web page.

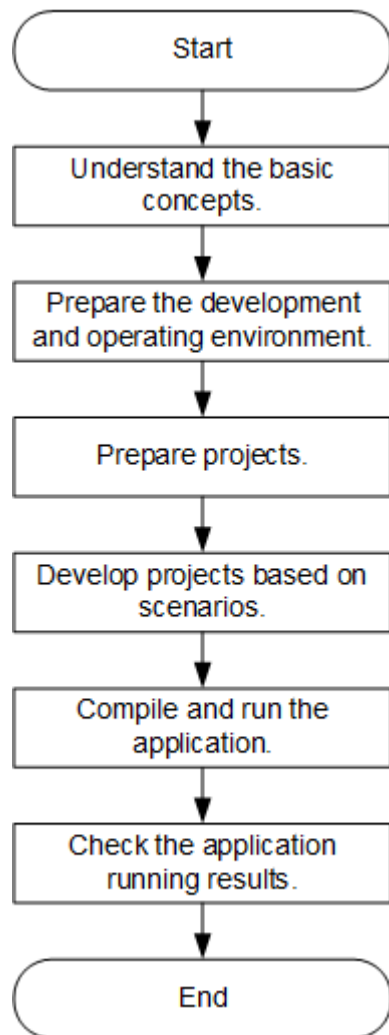
## keytab file

The keytab file is a key file that stores user information. Applications use the key file for API authentication on MRS.

### 14.1.3 Development Process

All stages of the development process are shown and described in [Figure 14-1](#) and [Table 14-1](#).

**Figure 14-1** HDFS development process



**Table 14-1** Description of HDFS development process

Stage	Description	Reference
Understand the basic concepts.	Before the application development, the basic concepts of HDFS are required to be understood.	<a href="#">Basic Concepts</a>

Stage	Description	Reference
Prepare the development and operating environment.	Use IntelliJ IDEA and configure the development environment based on the reference. The running environment of HDFS is the HDFS client. Install and configure the client based on the reference.	<a href="#">Preparing Development and Operating Environment</a>
Prepare projects.	HDFS provides sample projects in various scenarios. Sample projects can be imported for studying.	<a href="#">Configuring and Importing Sample Projects</a>
Prepare for security authentication.	If a safe cluster is used, the safety certification must be performed.	<a href="#">Preparing the Authentication Mechanism</a>
Develop projects based on scenarios.	Provide the sample project. This helps users to learn about the programming interfaces of all HDFS components quickly.	<a href="#">Developing the Project</a>
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	<a href="#">Commissioning the Application</a>
Check the application running results.	Application running results are stored in the directory specified by users. Users can also check the running results through the UI.	<a href="#">Commissioning the Application</a>

## 14.2 Environment Preparation

### 14.2.1 Preparing Development and Operating Environment

#### Preparing Development Environment

[Table 14-2](#) describes the environment required for application development.



**Table 14-2** Development Environment

Preparation	Description
OS	<ul style="list-style-type: none"> <li>Development environment: Windows OS. Windows 7 or later is supported.</li> <li>Operating environment: Windows OS or Linux OS.</li> </ul> <p>If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</p>
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> <li>For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> <li>Oracle JDK versions: 1.8</li> <li>IBM JDK versions: 1.8.5.11</li> </ul> </li> <li>For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> <li>OpenJDK: 1.8.0_272</li> </ul> </li> </ul> <p><b>NOTE</b> The server supports only TLS V1.2 or later to ensure security. By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls">https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls</a>.</p>
IntelliJ IDEA installation and configuration	<p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li> <li>If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li> <li>If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li> <li>Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li> </ul>
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
User development preparation	See <a href="#">Preparing a Developer Account</a> for configuration.

Preparation	Description
7-zip	Used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-Zip 16.04 is supported.

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. **Log in to the FusionInsight Manager portal** and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\HDFS\config** and manually import the configuration file to the configuration file directory (usually the **conf** folder) of the HDFS sample project.

The keytab file obtained during the **Preparing a Developer Account** is also stored in this directory.
  - c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, the client installation directory is **/opt/client**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. **Log in to the FusionInsight Manager portal.** Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/HDFS/config** directory and save them to the **conf** folder of the project code.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp HDFS/config/* root@IP address of the client node:/opt/client/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 14-3](#) describes the main configuration files.

**Table 14-3** Configuration files

File	Function
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.
user.keytab	Provides HDFS user information for Kerberos security authentication.
krb5.conf	Contains Kerberos server configuration information.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

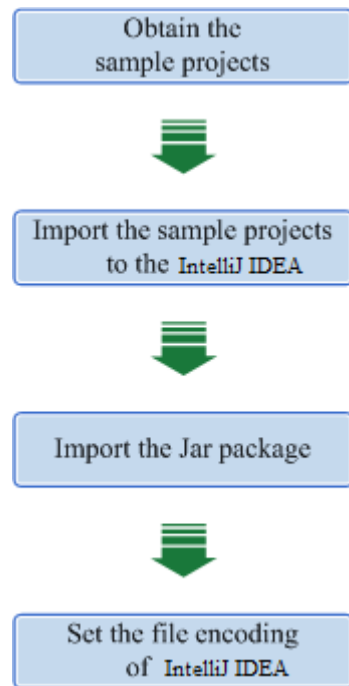
## 14.2.2 Configuring and Importing Sample Projects

### Scenario

HDFS provides sample projects for multiple scenarios to help you quickly learn HDFS projects.

The procedure for importing HDFS example codes is described as follows: [Figure 14-2](#) shows the procedure.

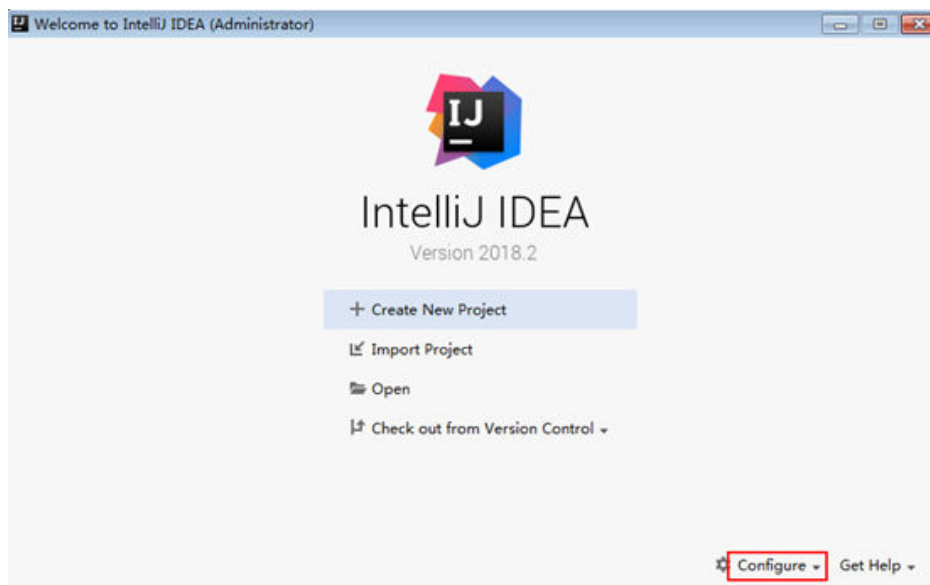
**Figure 14-2** Sample project importing procedure



### Procedure

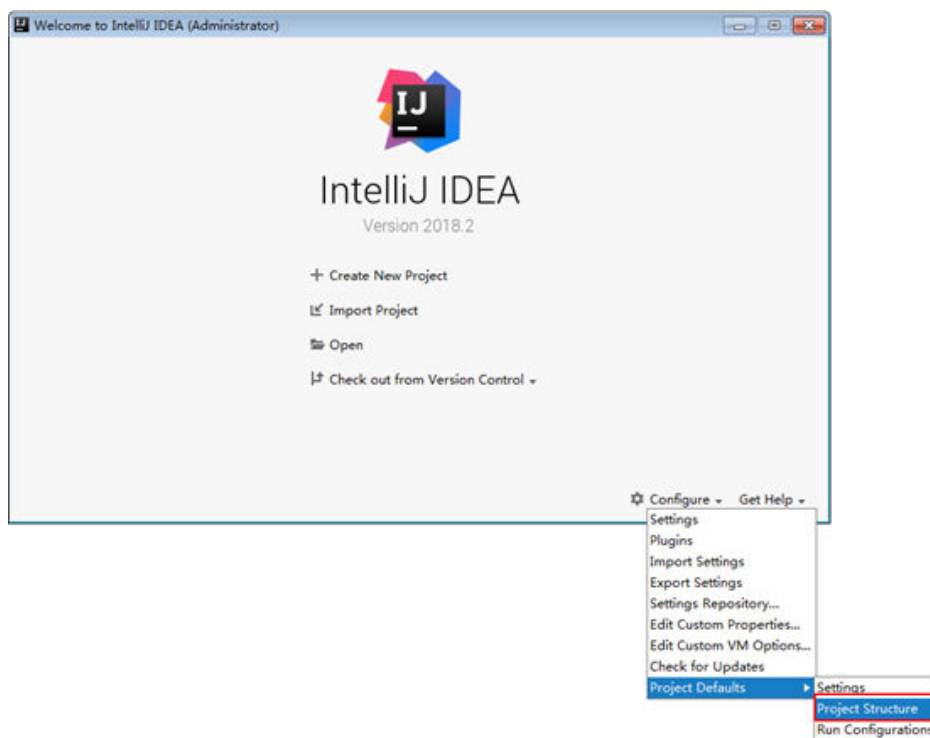
- Step 1** Obtain the sample project folder **hdfs-example-security** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the **user.keytab** and **krb5.conf** files obtained in [Preparing a Developer Account](#) section to the **conf** directory of the sample project.
- Step 3** After installing the IntelliJ IDEA and JDK tools, configure JDK in IntelliJ IDEA.
  1. Open IntelliJ IDEA and click **Configure**.

Figure 14-3 Quick Start



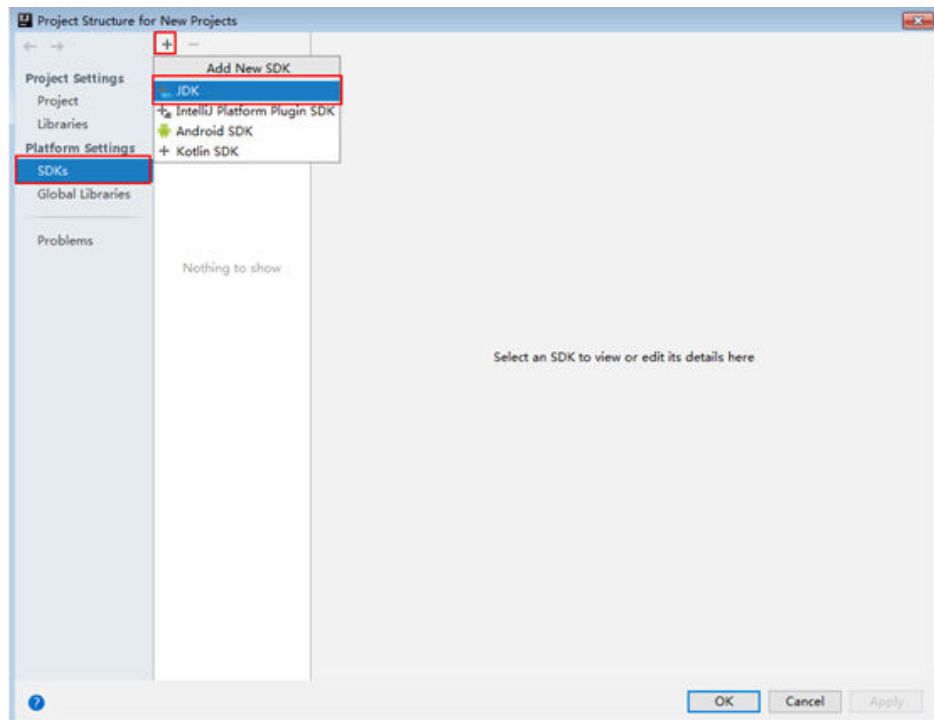
2. Choose **Project Defaults > Project Structure**.

Figure 14-4 Configure



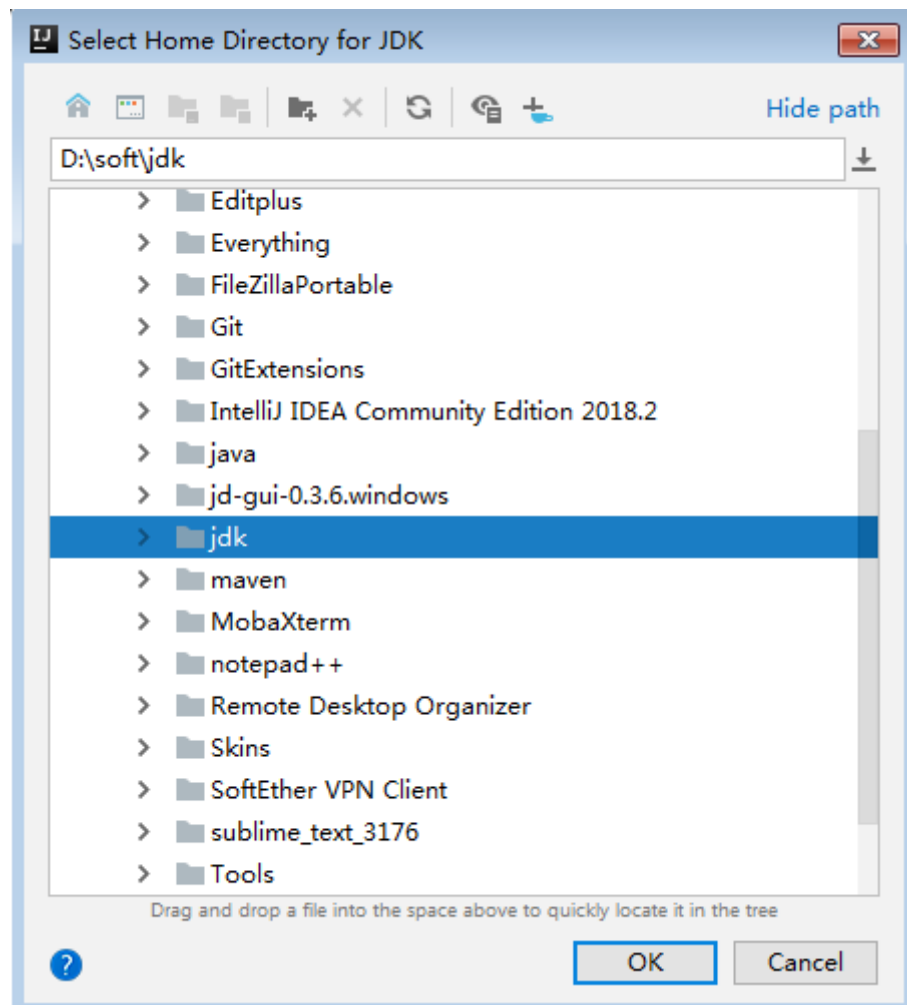
3. In the displayed **Project Structure for New Projects** interface, click **SDKs**. Then click the plus sign (+) and choose **JDK**.

**Figure 14-5** Project Structure for New Projects



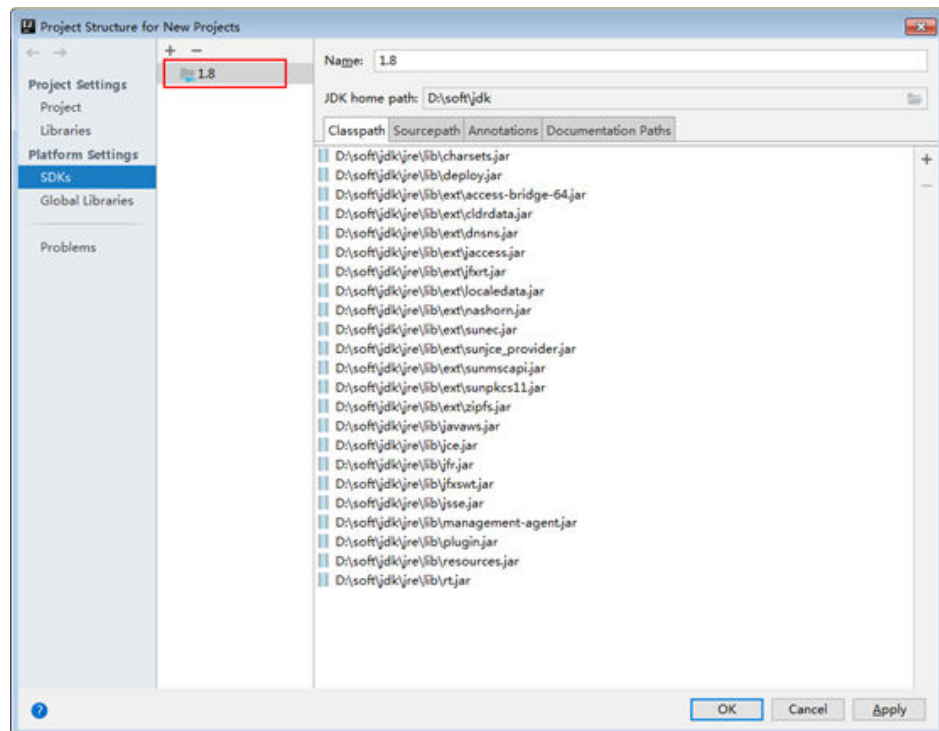
4. In the displayed **Select Home Directory for JDK** interface, select the JDK directory, and click **OK**.

**Figure 14-6** Select Home Directory for JDK



5. Click **OK**.

**Figure 14-7** Completing the JDK configuration



**Step 4** Import the example project to the IntelliJ IDEA development environment.

1. Open IntelliJ IDEA and choose **File > Open**.
2. Choose the directory of the example project **hdfs-example-security**. Click **OK**.

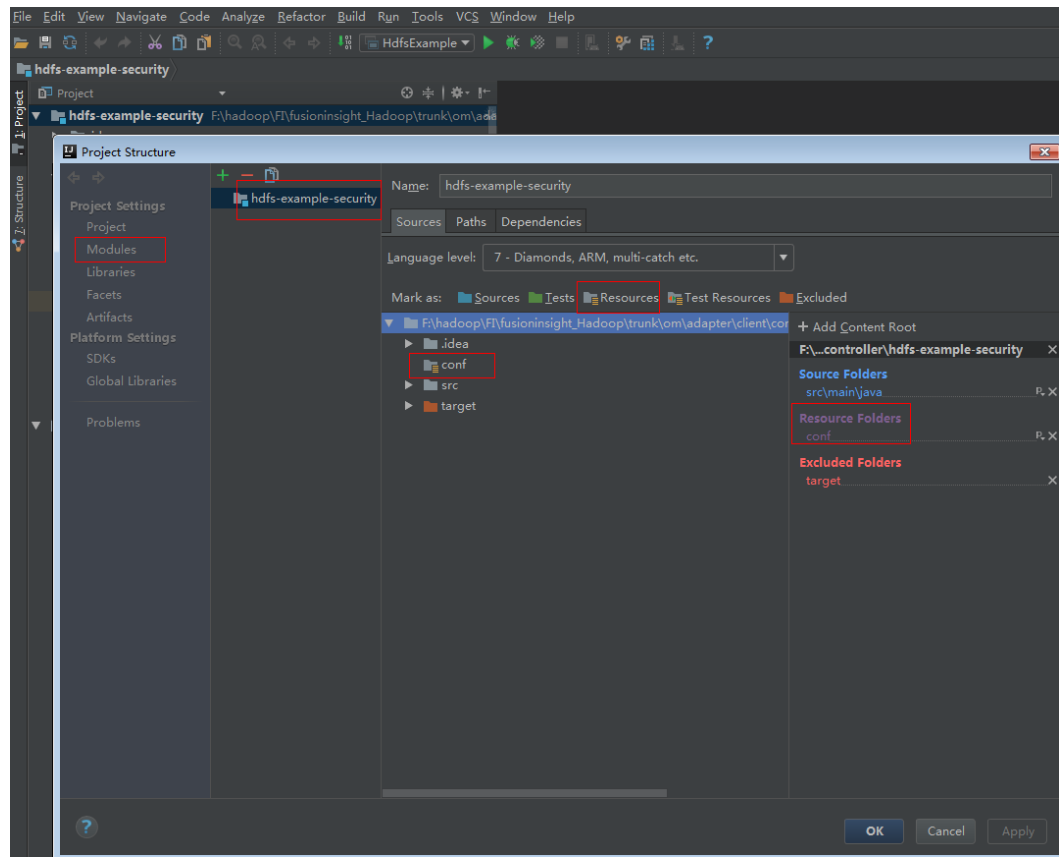
**Step 5** Add the related jar files the example project depending on into classpath.

If the sample project code is obtained from an open-source image site, dependency JAR files are automatically downloaded after Maven is configured (for details, see [Configuring Huawei Open-Source Mirrors](#)).

**Step 6** Add the conf directory in the project to the resource path. On the menu bar of the IntelliJ IDEA, choose **File > Project Structure**. In the dialog box that is displayed, click **Modules**, select the current project and click **Resources > conf > OK** to set the resource directory. [Figure 14-8](#) shows the directory for setting the project resource.



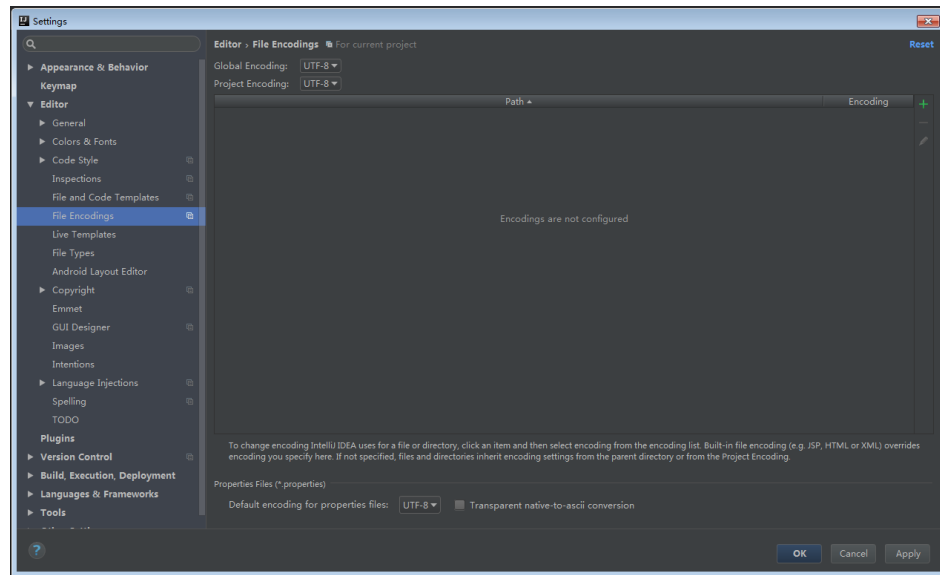
**Figure 14-8** Setting the Project Resource Directory



**Step 7** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.
2. Choose **Editor > File Encodings** from the navigation tree of the **Settings** window. In the "Global Encoding" and "Project Encodings" area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 14-9](#)

Figure 14-9 Setting the IntelliJ IDEA coding format



----End

## 14.2.3 Preparing the Authentication Mechanism

### Scenario

Before accessing services in the secure cluster environment, you must be authorized by Kerberos. Codes for security authentication need to be written into the HDFS applications to ensure that the applications can work properly.

Two security authentication methods are described as follows:

- Authentication by running command lines:  
Before submitting the HDFS application for running, run the following command in the HDFS client to obtain authentication:

```
kinit component service user
```

#### NOTE

This method applies only to the Linux OS that is installed with the HDFS client.

- Authentication by adding codes:  
Authenticate by obtaining the principal and keytab files of the client.  
Change the value of **PRINCIPAL\_NAME** in the code to the actual value.

```
private static final String PRINCIPAL_NAME = "hdfsDeveloper";
```

### Safety Security Code

The safety authentication of the example codes is completed by invoking the LoginUtil class.

In the HDFS sample project code, different sample projects use different authentication codes which are basic safety authentication and the basic safety authentication with the ZooKeeper authentication.

- Basic safety authentication:

Sample projects of the **HdfsExample** class in the **com.huawei.bigdata.hdfs.examples** package need only the basic safety authentication codes because these sample projects do not need to access the HBase or ZooKeeper. Add the following codes in the program:

```
...
private static final String PATH_TO_HDFS_SITE_XML =
HdfsExample.class.getClassLoader().getResource("hdfs-site.xml").getPath();
private static final String PATH_TO_CORE_SITE_XML =
HdfsExample.class.getClassLoader().getResource("core-site.xml").getPath();
private static final String PRNCIPAL_NAME = "hdfsDeveloper";
private static final String PATH_TO_KEYTAB =
HdfsExample.class.getClassLoader().getResource("user.keytab").getPath();
private static final String PATH_TO_KRB5_CONF =
HdfsExample.class.getClassLoader().getResource("krb5.conf").getPath();
private static Configuration conf = null;
//private static String PATH_TO_SMALL_SITE_XML =
HdfsExample.class.getClassLoader().getResource("smallfs-site.xml").getPath();
...
private static void confLoad() throws IOException {
 System.setProperty("java.security.krb5.conf", PATH_TO_KRB5_CONF);
 conf = new Configuration();
 // conf file
 conf.addResource(new Path(PATH_TO_HDFS_SITE_XML));
 conf.addResource(new Path(PATH_TO_CORE_SITE_XML));
 // conf.addResource(new Path(PATH_TO_SMALL_SITE_XML));
}
...
private static void authentication() throws IOException {
 // security mode
 if ("kerberos".equalsIgnoreCase(conf.get("hadoop.security.authentication"))) {
 System.setProperty("java.security.krb5.conf", PATH_TO_KRB5_CONF);
 LoginUtil.login(PRNCIPAL_NAME, PATH_TO_KEYTAB, PATH_TO_KRB5_CONF, conf);
 }
}
}
```

- Basic safety authentication with ZooKeeper Authentication:

Sample projects of the **ColocationExample** class in the **com.huawei.bigdata.hdfs.examples** package require not only the basic safety authentication, but also the Principal of the server in ZooKeeper to complete the safety authentication. Add the following codes in the program:

```
...
private static final String ZOOKEEPER_SERVER_PRINCIPAL_KEY = "zookeeper.server.principal";
private static final String PRINCIPAL = "username.client.kerberos.principal";
private static final String KEYTAB = "username.client.keytab.file";
private static final String PRNCIPAL_NAME = "hdfsDeveloper";
private static final String LOGIN_CONTEXT_NAME = "Client";
private static final String PATH_TO_KEYTAB = System.getProperty("user.dir") + File.separator +
"conf" + File.separator + "user.keytab";
private static final String PATH_TO_KRB5_CONF =
ColocationExample.class.getClassLoader().getResource("krb5.conf").getPath();
private static String zookeeperDefaultServerPrincipal = null;
private static Configuration conf = new Configuration();
private static DFSColocationAdmin dfsAdmin;
private static DFSColocationClient dfs;
private static void init() throws IOException {
 LoginUtil.login(PRNCIPAL_NAME, PATH_TO_KEYTAB, PATH_TO_KRB5_CONF, conf);
 LoginUtil.setJaasConf(LOGIN_CONTEXT_NAME, PRNCIPAL_NAME, PATH_TO_KEYTAB);
 zookeeperDefaultServerPrincipal = "zookeeper/hadoop." +
KerberosUtil.getKrb5DomainRealm().toLowerCase();
 LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY,
zookeeperDefaultServerPrincipal);
}
...
}
```

 NOTE

- The **HdfsDeveloper** user and the user's **user.keytab** and **krb5.conf** in the safety authentication codes are used as an example. In practical operations, contact the administrator to obtain the corresponding account and the keytab and krb5 files related to the account.
- You can log in to FusionInsight Manager, choose **System > Permission > Domain and Mutual Trust**, and check the value of **Local Domain**, which is the current system domain name.
- **zookeeper/hadoop.<system domain name>** is the user name. All letters in the system domain name contained in the user name of the system are lowercase letters. For example, if **Local domain** is set to **9427068F-6EFA-4833-B43E-60CB641E5B6C.COM**, the user name is **zookeeper/hadoop.9427068f-6efa-4833-b43e-60cb641e5b6c.com**.

## 14.3 Developing the Project

### 14.3.1 Scenario

#### Typical Scenario Description

Based on typical scenarios, users can quickly learn and master the HDFS development process and know important interface functions.

#### Scenario

Service operation objects of HDFS are files. File operations in example codes include creating a folder, writing data into a file, appending data to a file, reading data from a file, and deleting a file or folder. HDFS also supports other services including setting file permission. You can learn how to perform other operations on HDFS after learning the example codes in this chapter.

The example codes are described in the following order:

1. Initializing the HDFS.
2. Creating directories.
3. Writing data into a file.
4. Appending data to a file.
5. Reading data from a file.
6. Deleting a file.
7. Deleting directories.
8. Multi-thread tasks.
9. Setting storage policies.
10. Colocation.

## 14.3.2 Development Idea

### Development Idea

According to the previous scenario description, the following provides the basic operations for HDFS files with read, write, and delete operations on the `/user/hdfs-examples/test.txt` file as an example:

1. Pass the security certification.
2. Create a FileSystem object: `fSystem`.
3. Call the `mkdir` interface in `fSystem` to create a directory.
4. Call the `create` interface in `fSystem` to create an `FSDataOutputStream` object: `out`. Use the `write` method to write data into the object `out`.
5. Call the `append` interface in `fSystem` to create an `FSDataOutputStream` object: `out`. Use the `write` method to append data into the object `out`.
6. Call the `open` interface in `fSystem` to create an `FSDataInputStream` object: `in`. Use the `read` method to read files of the object `in`.
7. Call the `delete` interface in `fSystem` to delete the file.
8. Call the `delete` interface in `fSystem` to delete the folder.

## 14.3.3 Declare the Example Codes

### 14.3.3.1 Initializing the HDFS

#### Function

Hadoop distributed file system (HDFS) initialization is a prerequisite for using application programming interfaces (APIs) provided by the HDFS. The process of initializing the HDFS is

1. Load the HDFS service configuration file.
2. Instantiate a `FileSystem`.

#### NOTE

Obtain the keytab file for Kerberos security authentication in advance.

### Configuration File Description

**Table 14-4** lists the configuration files to be used during the login to the HDFS. These files already imported to the `conf` directory of the `hdfs-example-security` project.

**Table 14-4** Configuration files

File	Function
<code>core-site.xml</code>	Configures HDFS parameters.
<code>hdfs-site.xml</code>	Configures HDFS parameters.

File	Function
user.keytab	Provides HDFS user information for Kerberos security authentication.
krb5.conf	Contains Kerberos server configuration information.

 NOTE

- Different clusters cannot share the same **user.keytab** and **krb5.conf** files.
- The **log4j.properties** file under the **conf** directory can be configured based on your needs.

## Example Codes

The following is code snippets. For complete codes, see the `HdfsExample` class in `com.huawei.bigdata.hdfs.examples`.

The initialization codes used when applications are run in Linux and the codes used when applications are run in Windows are the same. The example codes are as follows:

```
// Complete initialization and authentication.
confLoad();
authentication();
// Creating a sample project
HdfsExample hdfs_examples = new HdfsExample("/user/hdfs-examples", "test.txt");

/**
 *
 * If the application is running in the Linux OS, the path of core-site.xml, hdfs-site.xml must be modified to
 * the absolute path of the client file in the Linux OS.
 *
 */
private static void confLoad() throws IOException {
 conf = new Configuration();
 // conf file
 conf.addResource(new Path(PATH_TO_HDFS_SITE_XML));
 conf.addResource(new Path(PATH_TO_CORE_SITE_XML));
 // conf.addResource(new Path(PATH_TO_SMALL_SITE_XML));
}

/**
 *Safety authentication
 *
 */
private static void authentication() throws IOException {
 // security mode
 if ("kerberos".equalsIgnoreCase(conf.get("hadoop.security.authentication"))) {
 System.setProperty("java.security.krb5.conf", PATH_TO_KRB5_CONF);
 LoginUtil.login(PRNCIPAL_NAME, PATH_TO_KEYTAB, PATH_TO_KRB5_CONF, conf);
 }
}

/**
 *Create a sample project.
 */
public HdfsExample(String path, String fileName) throws IOException {
 this.DEST_PATH = path;
 this.FILE_NAME = fileName;
}
```

```
instanceBuild();
}

private void instanceBuild() throws IOException {
 FileSystem = FileSystem.get(conf);
}
```

The login example codes need to be added for the first login when applications are run in both Windows and Linux. For details on the example codes, see the `LoginUtil` class in `com.huawei.hadoop.security`.

```
public synchronized static void login(String userPrincipal, String userKeytabPath, String krb5ConfPath,
Configuration conf)
 throws IOException
{
 // 1.Check the input parameters.

 if ((userPrincipal == null) || (userPrincipal.length() <= 0))
 {
 LOG.error("input userPrincipal is invalid.");
 throw new IOException("input userPrincipal is invalid.");
 }

 if ((userKeytabPath == null) || (userKeytabPath.length() <= 0))
 {
 LOG.error("input userKeytabPath is invalid.");
 throw new IOException("input userKeytabPath is invalid.");
 }

 if ((krb5ConfPath == null) || (krb5ConfPath.length() <= 0))
 {
 LOG.error("input krb5ConfPath is invalid.");
 throw new IOException("input krb5ConfPath is invalid.");
 }

 if ((conf == null))
 {
 LOG.error("input conf is invalid.");
 throw new IOException("input conf is invalid.");
 }

 // 2.Check whether the file exists.

 File userKeytabFile = new File(userKeytabPath);
 if (!userKeytabFile.exists())
 {
 LOG.error("userKeytabFile(" + userKeytabFile.getAbsolutePath() + ") does not exist.");
 throw new IOException("userKeytabFile(" + userKeytabFile.getAbsolutePath() + ") does not exist.");
 }
 if (!userKeytabFile.isFile())
 {
 LOG.error("userKeytabFile(" + userKeytabFile.getAbsolutePath() + ") is not a file.");
 throw new IOException("userKeytabFile(" + userKeytabFile.getAbsolutePath() + ") is not a file.");
 }

 File krb5ConfFile = new File(krb5ConfPath);
 if (!krb5ConfFile.exists())
 {
 LOG.error("krb5ConfFile(" + krb5ConfFile.getAbsolutePath() + ") does not exist.");
 throw new IOException("krb5ConfFile(" + krb5ConfFile.getAbsolutePath() + ") does not exist.");
 }
 if (!krb5ConfFile.isFile())
 {
 LOG.error("krb5ConfFile(" + krb5ConfFile.getAbsolutePath() + ") is not a file.");
 throw new IOException("krb5ConfFile(" + krb5ConfFile.getAbsolutePath() + ") is not a file.");
 }

 // 3.Set and check krb5config.

 setKrb5Config(krb5ConfFile.getAbsolutePath());
 setConfiguration(conf);
}
```

```
// 4.Log in to Hadoop to check items.

loginHadoop(userPrincipal, userKeytabFile.getAbsolutePath());
LOG.info("Login success!!!!!!!!!!!!!!");
}
```

### 14.3.3.2 Creating Directories

#### Function

Process of creating a directory:

1. Call the exists method of the FileSystem instance to check whether the directory exists.
2. If yes, the method stops.
3. If no, call the mkdirs method in the FileSystem instance to create a directory.

#### Example Codes

The following is a code snippet. For complete codes, see the **HdfsExample** class in **com.huawei.bigdata.hdfs.examples**.

```
/**
 * Create a directory.
 *
 * @throws java.io.IOException
 */
private void mkdir() throws IOException {
 Path destPath = new Path(DEST_PATH);
 if (!createPath(destPath)) {LOG.error("failed to create destPath " + DEST_PATH);
 return;
 }

 LOG.info("success to create path " + DEST_PATH);
}

/**
 * create file path
 *
 * @param filePath
 * @return
 * @throws java.io.IOException
 */
private boolean createPath(final Path filePath) throws IOException {
 if (!FileSystem.exists(filePath)) {
 fSystem.mkdirs(filePath);
 }
 return true;
}
```

### 14.3.3.3 Writing Data into a File

#### Function

The process of writing data into a file is

1. Use the create method in the FileSystem instance to obtain the output stream of writing files.
2. Uses this data stream to write content into a specified file in the HDFS.



 NOTE

Close all requested resources after writing files.

## Example Codes

The following is code snippets. For complete codes, see HdfsExample class in **com.huawei.bigdata.hdfs.examples**.

```
/**
 * Create a file and write data into the file.
 *
 * @throws java.io.IOException
 * @throws com.huawei.bigdata.hdfs.examples.ParameterException
 */
private void write() throws IOException {
 final String content = "hi, I am bigdata. It is successful if you can see me.";
 FSDataOutputStream out = null;
 try {
 out = fSystem.create(new Path(DEST_PATH + File.separator + FILE_NAME));
 out.write(content.getBytes());
 out.hsync();
 LOG.info("success to write.");
 } finally {
 // make sure the stream is closed finally.
 IOUtils.closeStream(out);
 }
}
```

### 14.3.3.4 Appending Data to a File

#### Function

Append data to a specified file in the Hadoop distributed file system (HDFS). The process of appending data to a file is

1. Use the append method in the FileSystem instance to obtain the output stream of the appended data.
2. Use the output stream to add the content to be appended behind the specified file in the HDFS.

 NOTE

Close all requested resources after appending data.

## Example Codes

The following is code snippets. For complete codes, see HdfsExample class in **com.huawei.bigdata.hdfs.examples**.

```
/**
 * Append data to a file
 *
 * @throws java.io.IOException
 */
private void append() throws IOException {
 final String content = "I append this content.";
 FSDataOutputStream out = null;
 try {
```

```
 out = fSystem.append(new Path(DEST_PATH + File.separator + FILE_NAME));
 out.write(content.getBytes());
 out.hsync();
 LOG.info("success to append.");
 } finally {
 // make sure the stream is closed finally.
 IOUtils.closeStream(out);
 }
}
```

### 14.3.3.5 Reading Data from a File

#### Function

Read data from a specified file in the Hadoop distributed file system (HDFS). The process is:

1. Use the open method in the FileSystem instance to obtain the input stream of writing files.
2. Use the input stream to read the content of the specified file in the HDFS.

#### NOTE

Close all requested resources after reading files.

#### Example Codes

The following is code snippets. For complete codes, see the HdfsExample class in **com.huawei.bigdata.hdfs.examples**.

```
/**
 * Read s file.
 *
 * @throws java.io.IOException
 */
private void read() throws IOException {
 String strPath = DEST_PATH + File.separator + FILE_NAME;
 Path path = new Path(strPath);
 FSDataInputStream in = null;
 BufferedReader reader = null;
 StringBuffer strBuffer = new StringBuffer();
 try {
 in = fSystem.open(path);
 reader = new BufferedReader(new InputStreamReader(in));
 String sTempOneLine;
 // write file
 while ((sTempOneLine = reader.readLine()) != null) {
 strBuffer.append(sTempOneLine);
 }
 LOG.info("result is : " + strBuffer.toString());
 LOG.info("success to read.");
 } finally {
 // make sure the streams are closed finally.
 IOUtils.closeStream(reader);
 IOUtils.closeStream(in);
 }
}
```

### 14.3.3.6 Deleting a File

#### Function

Delete a file from the Hadoop distributed file system (HDFS).

#### NOTE

Exercise caution when you delete files because the deletion is irreversible.

#### Example Codes

The following is code snippets. For complete codes, see the `HdfsExample` class in **`com.huawei.bigdata.hdfs.examples`**.

```
/**
 * Delete a file.
 *
 * @throws java.io.IOException
 */
private void delete() throws IOException {
 Path beDeletedPath = new Path(DEST_PATH + File.separator + FILE_NAME);
 if (fSystem.delete(beDeletedPath, true)) {
 LOG.info("success to delete the file " + DEST_PATH + File.separator + FILE_NAME);
 } else {
 LOG.warn("failed to delete the file " + DEST_PATH + File.separator + FILE_NAME);
 }
}
```

### 14.3.3.7 Deleting Directories

#### Function

Delete a specified directory from the HDFS.

#### NOTE

Files in the deleted directory will be stored in the **Trash/Current** folder of the directory of the current user. In the event of mis-deletion, you can restore the folder from the directory.

#### Example Codes

The following is a code snippet of file deletion. For complete codes, see the **`HdfsExample`** class in **`com.huawei.bigdata.hdfs.examples`**.

```
/**
 * delete the directory
 *
 * @throws java.io.IOException
 */
private void rmdir() throws IOException {
 Path destPath = new Path(DEST_PATH);
 if (!deletePath(destPath)) {
 LOG.error("failed to delete destPath " + DEST_PATH);
 return;
 }
 LOG.info("success to delete path " + DEST_PATH);
}
```

```
/**
 * delete file path
 *
 * @param filePath
 * @return
 * @throws java.io.IOException
 */
private boolean deletePath(final Path filePath) throws IOException {
 if (!FileSystem.exists(filePath)) {
 return false;
 }
 // FileSystem.delete(filePath, true);
 return FileSystem.delete(filePath, true);
}
```

### 14.3.3.8 Multi-Thread Tasks

#### Function

Create multi-threaded tasks and initiate multiple instances to perform file operations.

#### Example Codes

The following is a code snippet of file deletion. For complete codes, see the **HdfsExample** class in **com.huawei.bigdata.hdfs.examples**.

```
// Service example 2: multi-thread tasks
final int THREAD_COUNT = 2;
for (int threadNum = 0; threadNum < THREAD_COUNT; threadNum++) {
 HdfsExampleThread example_thread = new HdfsExampleThread("hdfs_example_" + threadNum);
 example_thread.start();
}

class HdfsExampleThread extends Thread {
 private final static Log LOG = LogFactory.getLog(HdfsExampleThread.class.getName());
 /**
 *
 * @param threadName
 */
 public HdfsExampleThread(String threadName) {
 super(threadName);
 }
 public void run() {
 HdfsExample example;
 try {
 example = new HdfsExample("/user/hdfs-examples/" + getName(), "test.txt");
 example.test();
 } catch (IOException e) {
 LOG.error(e);
 }
 }
}
```

The **example.test()** method is the operation on files. The code is as follows:

```
/**
 * HDFS operation instance
 *
 * @throws IOException
 * @throws ParameterException
 *
 * @throws Exception
 */
```

```
public void test() throws IOException {
 // Create a directory.
 mkdir();

 // Write data into a file.
 write();

 // Append data to a file.
 append();

 // Read a file.
 read();

 // Delete a file.
 delete();

 // Delete a directory.
 rmdir();
}
```

### 14.3.3.9 Setting Storage Policies

#### Function

Specify storage policies for a file or folder in the HDFS.

#### Example Code

1. [Log in to the FusionInsight Manager portal](#), and choose **Cluster > Name of the desired cluster > Services > HDFS > Configurations > All Configurations**.
2. Check whether the value of **dfs.storage.policy.enabled** is the default value **true**. If not, modify the value to **true**, click **Save**, and restart HDFS.
3. Check the code.

The following code segment is only an example. For details, see the `HdfsExample` class in `com.huawei.bigdata.hdfs.examples`.

```
/**
 * set storage policy to path
 * @param policyName
 * Policy Name can be accepted:
 * HOT
 * WARM
 * COLD
 * LAZY_PERSIST
 * ALL_SSD
 * ONE_SSD
 * @throws IOException
 */
private void setStoragePolicy(String policyName) throws IOException {
 if (fSystem instanceof DistributedFileSystem) {
 DistributedFileSystem dfs = (DistributedFileSystem) fSystem;
 Path destPath = new Path(DEST_PATH);
 Boolean flag = false;
 mkdir();
 BlockStoragePolicySpi[] storage = dfs.getStoragePolicies();
 for (BlockStoragePolicySpi bs : storage) {
 if (bs.getName().equals(policyName)) {
 flag = true;
 }
 LOG.info("StoragePolicy:" + bs.getName());
 }
 if (!flag) {
 policyName = storage[0].getName();
 }
 }
}
```

```
dfs.setStoragePolicy(destPath, policyName);
LOG.info("success to set Storage Policy path " + DEST_PATH);
rmdir();
} else {
 LOG.info("SmallFile not support to set Storage Policy !!!");
}
}
```

### 14.3.3.10 Colocation

#### Function Description

Colocation is to store associated data or data on which associated operations are performed on the same storage node. The HDFS Colocation feature stores files on which associated operations are performed on the same data node so that data can be obtained from the same data node during associated operations. This greatly reduces network bandwidth consumption.

Before using the Colocation function, you are advised to be familiar with the internal mechanisms of Colocation, including:

##### Colocation node allocation principle

##### Capacity expansion and Colocation allocation

##### Colocation and data node capacity

- **Colocation node allocation principle**

Colocation allocates data nodes to locators evenly according to the allocation node quantity.

##### NOTE

The allocation algorithm principle is as follows: Colocation queries all locators, reads the data nodes allocated to the locators, and records the number of times. Based on the number of times, Colocation sorts the data nodes. The data nodes that are rarely used are placed at the beginning and selected first. The count increase by 1 each time after a node is selected. The nodes are shorted again, and the subsequent node will be selected.

- **Capacity expansion and Colocation allocation**

After cluster capacity expansion, you can select one of the following two policies shown in [Table 14-5](#) to balance the usage of data nodes and ensure that the allocation frequency of the newly added nodes is consistent with that of the old data nodes.

**Table 14-5** Allocation policies

No.	Policy	Description
1	Delete the original locators and create new locators for all data nodes in the cluster.	<ol style="list-style-type: none"> <li>1. The original locator before the capacity expansion evenly uses all data nodes. After the capacity expansion, the newly added nodes are not allocated to existing locators, so Colocation stores data only to the old data nodes.</li> <li>2. Data nodes are allocated to specific locators. Therefore, after capacity expansion, Colocation needs to reallocated data nodes to locators.</li> </ol>
2	Create new locators and plan the data storage mode again.	The old locators use the old data nodes, while the newly created locators mainly use the new data nodes. Therefore, locators need to be planned again based on the actual service requirements on data.

 **NOTE**

Generally, you are advised to use the policy to reallocate data nodes to locators after capacity expansion to prevent data from being stored only to the new data nodes.

- **Colocation and data node capacity**

When Colocation is used to store data, the data is stored to the data node of a specified locator. If no locator planning is performed, the data node capacity will be uneven. [Table 14-6](#) summarizes the two usage principles to ensure even data node capacity.

**Table 14-6** Usage Principle

No.	Usage Principle	Description
1	All the data nodes are used in the same frequency in locators.	Assume that there are N data nodes, the number of locators must an integral multiple of N (N, 2 N, ...).
2	A proper data storage plan must be made for all locators to ensure that data is evenly stored in the locators.	None

During HDFS secondary development, you can obtain the DFSColocationAdmin and DFSColocationClient instances to create groups, delete groups, write files, and delete files in or from the location.

 NOTE

- When the Colocation function is enabled and users specify DataNodes, the data volume will be large on some nodes. Serious data skew will result in HDFS data write failures.
- Because of data skew, MapReduce accesses only several nodes. In this case, the load is heavy on these nodes, while other nodes are idle.
- For a single application task, the DFSColocationAdmin and DFSColocationClient instances can be used only once. If the instances are used for many time, excessive HDFS links will be created and use up HDFS resources.
- If you need to perform the balance operation for a file uploaded by colocation, you can set the `oi.dfs.colocation.file.pattern` parameter on FusionInsight Manager to the file path to avoid invalid colocation. If there are multiple files, use commas (,) to separate the file paths, for example, `/test1,/test2`.
- Colocation stores associated data or data on which associated operations are performed on the same storage. When Balancer- or Mover-related operations are performed, data blocks will be moved. As a result, the Colocation function becomes unavailable. Therefore, when using the Colocation function, you are advised to set the HDFS configuration item **`dfs.datanode.block-pinning.enabled`** to **`true`**. In this case, files written by Colocation will not be moved when Balancer- or Mover-related operations are performed in the cluster, ensuring file colocation.

## Example Codes

For complete example codes, see `com.huawei.bigdata.hdfs.examples.ColocationExample`.

 NOTE

- Before using the Colocation function, add HDFS users to the supergroup group.
- When the Colocation project is run, the HDFS parameter **`fs.defaultFS`** cannot be set to **`viewfs://ClusterX`**.

### 1. Initialization

Kerberos security authentication is required before using Colocation.

```
private static void init() throws IOException {
 conf.set(KEYTAB, PATH_TO_KEYTAB);
 conf.set(PRINCIPAL, PRNCIPAL_NAME);

 LoginUtil.setJaasConf(LOGIN_CONTEXT_NAME, PRNCIPAL_NAME, PATH_TO_KEYTAB);
 LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY,
 ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);
 LoginUtil.login(PRNCIPAL_NAME, PATH_TO_KEYTAB, PATH_TO_KRB5_CONF, conf);
}
```

### 2. Obtain instances.

Example: Colocation operations require the DFSColocationAdmin and DFSColocationClient instances. Therefore, the instances must be obtained before operations, such as creating a group.

```
dfsAdmin = new DFSColocationAdmin(conf);
dfs = new DFSColocationClient();
dfs.initialize(URI.create(conf.get("fs.defaultFS")), conf);
```

### 3. Create a group.

Example: Create a `gid01` group, which contains three locators.

```
/**
 * create group
 *
 * @throws java.io.IOException
```



```
*/
private static void createGroup() throws IOException {
 dfsAdmin.createColocationGroup(COLOCATION_GROUP_GROUP01,
 Arrays.asList(new String[] { "lid01", "lid02", "lid03" }));
}
}
```

4. Write data into a file. The related group must be created before writing data into the file.

Example: Write data into the testfile.txt file.

```
/**
 * create and write file
 *
 * @throws java.io.IOException
 */
private static void put() throws IOException {
 FSDataOutputStream out = dfs.create(new Path(TESTFILE_TXT), true,
 COLOCATION_GROUP_GROUP01, "lid01");
 // the data to be written to the hdfs.
 byte[] readBuf = "Hello World".getBytes("UTF-8");
 out.write(readBuf, 0, readBuf.length);
 out.close();
}
}
```

5. Delete a file.

Example: Delete the testfile.txt file.

```
/**
 * delete file
 *
 * @throws java.io.IOException
 */
@SuppressWarnings("deprecation")
private static void delete() throws IOException {
 dfs.delete(new Path(TESTFILE_TXT));
}
}
```

6. Delete a group.

Example: Delete gid01.

```
/**
 * delete group
 *
 * @throws java.io.IOException
 */
private static void deleteGroup() throws IOException {
 dfsAdmin.deleteColocationGroup(COLOCATION_GROUP_GROUP01);
}
}
```

## 14.4 Commissioning the Application

### 14.4.1 Commissioning an Application in the Windows Environment

#### 14.4.1.1 Compiling and Running an Application

##### Scenario

After the code development is complete, you can run an application in the Windows development environment.

If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

## Procedure

**Step 1** In a development environment (such as IntelliJ IDEA), choose the following two projects separately and run the projects:

- Choose **HdfsExample.java** and right-click the project and choose **Run 'HdfsExample.main()'** from the shortcut menu to run the project.
- Choose **ColocationExample.java** and right-click the project and choose **Run 'ColocationExample.main()'** from the shortcut menu to run the project.

### NOTE

- Do not restart HDFS service while HDFS application is in running status, otherwise the application will fail.
- When the Colocation project is run, the HDFS parameter **fs.defaultFS** cannot be set to **viewfs://ClusterX**.

----End

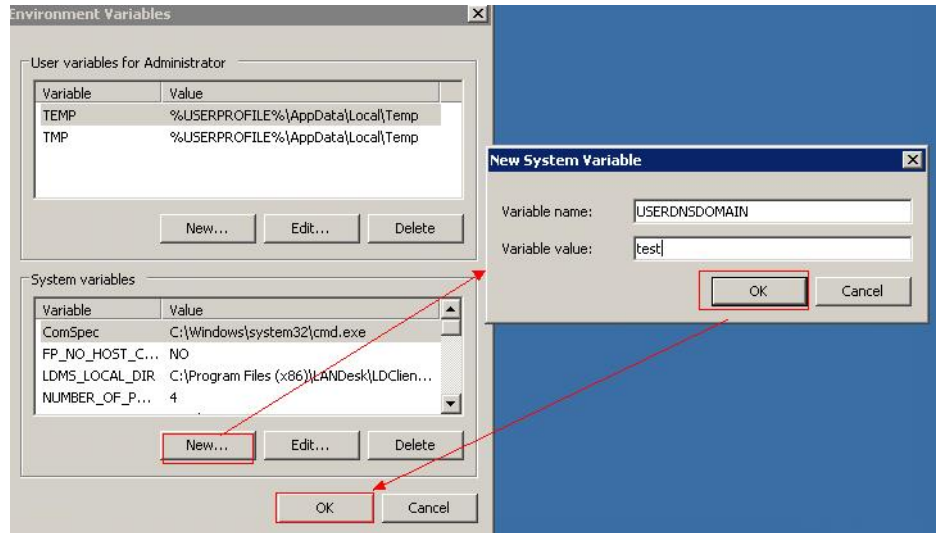
## Note

During security authentication, Hadoop needs to obtain the domain name of the host where the client is located (**Default Realm**, which is obtained from environment variable **USERDNSDOMAIN**). If the host does not have a domain name, the following error message is displayed when you run the sample program:

```
Exception in thread "main" java.lang.IllegalArgumentException: Can't get Kerberos realm
 at org.apache.hadoop.security.HadoopKerberosName.setConfiguration(HadoopKerberosName.java:65)
 at org.apache.hadoop.security.UserGroupInformation.initialize(UserGroupInformation.java:288)
 at org.apache.hadoop.security.UserGroupInformation.ensureInitialized(UserGroupInformation.java:273)
 at org.apache.hadoop.security.UserGroupInformation.loginUserFromSubject(UserGroupInformation.java:832)
 at org.apache.hadoop.security.UserGroupInformation.getLoginUser(UserGroupInformation.java:802)
 at org.apache.hadoop.security.UserGroupInformation.getCurrentUser(UserGroupInformation.java:675)
 at org.apache.hadoop.conf.Configuration$Resource.getRestrictParserDefault(Configuration.java:243)
 at org.apache.hadoop.conf.Configuration$Resource.<init>(Configuration.java:211)
 at org.apache.hadoop.conf.Configuration$Resource.<init>(Configuration.java:203)
 at org.apache.hadoop.conf.Configuration.addResource(Configuration.java:851)
 at com.huawei.bigdata.hdfs.examples.HdfsExample.confLoad(HdfsExample.java:307)
 at com.huawei.bigdata.hdfs.examples.HdfsExample.main(HdfsExample.java:277)
Caused by: java.lang.reflect.InvocationTargetException
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
 at java.lang.reflect.Method.invoke(Unknown Source)
 at org.apache.hadoop.security.authentication.util.KerberosUtil.getDefaultRealm(KerberosUtil.java:88)
 at org.apache.hadoop.security.HadoopKerberosName.setConfiguration(HadoopKerberosName.java:63)
 ... 11 more
Caused by: KrbException: Cannot locate default realm
 at sun.security.krb5.Config.getDefaultRealm(Unknown Source)
 ... 17 more
```

You can set environment variable **USERDNSDOMAIN** of the system to prevent this problem. The details are as follows:

1. Right-click **Computer** and choose **Properties** from the shortcut menu. The dialog box shown in the following figure is displayed. Click **Advanced system settings > Advanced > Environment Variables**.
2. Set the system environment variable and click "New". The New System Variable dialog box is displayed. Enter **USERDNSDOMAIN** in Variable name and set the Variable value to a non-null character string, for example, test. Click OK twice. The system environment variable is set.



3. Close the sample project, open it again, and run it.

### 14.4.1.2 Checking the Commissioning Result

#### Scenario

After an HDFS application is run, you can learn the application running conditions by viewing the running result or HDFS logs.

#### Procedure

- **Learn the application running conditions by viewing the running result.**
  - The running result of the HDFS windows example application is shown as follows:

```

1308 [main] INFO org.apache.hadoop.security.UserGroupInformation - Login successful for
user hdfsDevelop using keytab file
1308 [main] INFO com.huawei.hadoop.security.LoginUtil - Login success!!!!!!!!!!!!!!
2040 [main] WARN org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory - The short-
circuit local reads feature cannot be used because UNIX Domain sockets are not available on
Windows.
3006 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to create path /
user/hdfs-examples
3131 [main] WARN org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
3598 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to write.
4408 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to append.
5015 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I am
bigdata. It is successful if you can see me.I append this content.
5015 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to read.
5077 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete the file /
user/hdfs-examples/test.txt
5186 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete path /
user/hdfs-examples
5311 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
create path /user/hdfs-examples/hdfs_example_0
5311 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
create path /user/hdfs-examples/hdfs_example_1
5669 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
write.
5669 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
write.
7258 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
append.

```

```
7741 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
append.
7896 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I
am bigdata. It is successful if you can see me.I append this content.
7896 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
read.
7959 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
delete the file /user/hdfs-examples/hdfs_example_1/test.txt
8068 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
delete path /user/hdfs-examples/hdfs_example_1
8364 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I
am bigdata. It is successful if you can see me.I append this content.
8364 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
read.
8426 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
delete the file /user/hdfs-examples/hdfs_example_0/test.txt
8535 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
delete path /user/hdfs-examples/hdfs_example_0
```

### NOTE

In the Windows environment, the following exception occurs but does not affect services.

```
java.io.IOException: Could not locate executable null\bin\winutils.exe in the
Hadoop binaries.
```

- The running result of the Colocation windows example application is shown as follows:

```
...
945 [main] INFO org.apache.hadoop.security.UserGroupInformation - Login successful for user
hdfsDeveloper using keytab file user.keytab
945 [main] INFO com.huawei.hadoop.security.LoginUtil - Login success!!!!!!!!!!!!!!
945 [main] INFO com.huawei.hadoop.security.LoginUtil - JaasConfiguration
loginContextName=Client principal=hdfsDeveloper useTicketCache=false keytabFile=XXX
\sample_project\src\hdfs-example-security\conf\user.keytab
946 [main] INFO com.huawei.hadoop.security.KerberosUtil - Get default realm successfully,
the realm is : HADOOP.COM
...
Create Group has finished.
Put file is running...
Put file has finished.
Delete file is running...
Delete file has finished.
Delete Group is running...
Delete Group has finished.
4946 [main-EventThread] INFO org.apache.zookeeper.ClientCnxn - EventThread shut down for
connection: 0x440751cb41a4d415
4946 [main] INFO org.apache.zookeeper.ZooKeeper - Connection: 0x440751cb41a4d415 closed
...
```

- **Learn the application running conditions by viewing HDFS logs.**

The NameNode logs of HDFS offer immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 14.4.2 Commissioning an Application in the Linux Environment

### 14.4.2.1 Compiling and Running an Application With the Client Installed

#### Scenario

The Hadoop distributed file system (HDFS) application can run in the Linux operating system (OS) with the HDFS client installed. After the application code

has been developed, you can upload the jar packages to the HDFS client and run the application.

## Prerequisite

- The HDFS client has been installed.
- When the host where the Linux OS runs is not a node of the cluster, you are required to set the mapping between the host name and IP address in the **hosts** file of the node where the client is installed. The host name must be correctly mapped to the IP address.

## Procedure

**Step 1** Go to the local root directory of the project, copy the required configuration file to the conf folder of the local project, and run the following command in Windows cmd to compress the package:

```
mvn -s "{maven_setting_path}" clean package
```

### NOTE

- In the preceding command, {maven\_setting\_path} is the path of the **settings.xml** file of the local Maven.
- After the package is successfully packed, obtain the JAR package, for example, *HDFSTest-XXX.jar*, from the **target** subdirectory in the root directory of the project. The name of the JAR package varies according to the actual package.

**Step 2** Upload the exported jarpackages to any directory in the running environment of the client, for example, **/opt/client**.

**Step 3** Configure the environment variables:

```
cd /opt/client
```

```
source bigdata_env
```

**Step 4** Run the following commands to execute the jar packages.

```
hadoop jar HDFSTest-XXX.jar com.huawei.bigdata.hdfs.examples.HdfsExample
```

```
hadoop jar HDFSTest-XXX.jar
com.huawei.bigdata.hdfs.examples.ColocationExample
```

### NOTE

When **com.huawei.bigdata.hdfs.examples.ColocationExample** is run, the HDFSparameter **fs.defaultFS** cannot be set to **viewfs://ClusterX**.

----End

## 14.4.2.2 Compiling and Running an Application With the Client Not Installed

### Scenario

The Hadoop distributed file system (HDFS) application can run in the Linux operating system (OS) with the HDFS client not installed. After the application code has been developed, you can upload the jar packages to the Linux OS and run the application.

## Prerequisite

- A JDK has been installed in the Linux environment. The version of the JDK must be consistent with that of the JDK used by IDEA to export the JAR package.
- When the host where the Linux OS runs is not a node of the cluster, you are required to set the mapping between the host name and IP address in the **hosts** file of the node where the Linux OS runs. The host name must be correctly mapped to the IP address.

## Procedure

- Step 1** Go to the local root directory of the project, copy the required configuration file to the conf folder of the local project and run the following command in Windows cmd to compress the package:

```
mvn -s "{maven_setting_path}" clean package
```

### NOTE

- In the preceding command, {maven\_setting\_path} is the path of the **settings.xml** file of the local Maven.
- After the package is successfully packed, obtain the JAR package from the target subdirectory in the root directory of the project.

- Step 2** Upload the exported jar packages to any directory in the running environment of the Linux OS, for example, **/opt/hadoop\_client**.
- Step 3** Upload the **lib** and **conf** folders of the project to the same directory that stores the jar packages (the **lib** Create a **lib** folder in the Linux operating environment directory (for example, **/opt/client**) and upload the required **JAR** packages. The **lib** folder contains all the **JAR** packages that the project depends on. For details, see section [Preparing an Operating Environment](#).
- Step 4** (Optional) To run the SmallFS example code, modify the value of the **fs.defaultFS** configuration item in the **core-site.xml** file from **hdfs://hacluster** to **sfs://hacluster**.
- Step 5** Run the following commands to execute the jar packages.

```
java -cp HDFSTest-XXX.jar:conf:/lib/*
com.huawei.bigdata.hdfs.examples.HdfsExample

java -cp HDFSTest-XXX.jar:conf:/lib/*
com.huawei.bigdata.hdfs.examples.ColocationExample
```

### NOTE

When **com.huawei.bigdata.hdfs.examples.ColocationExample** is run, the HDFSparameter **fs.defaultFS** cannot be set to **viewfs://ClusterX**.

----End

### 14.4.2.3 Checking the Commissioning Result

#### Scenario

After an HDFS application is run, you can learn the application running conditions by viewing the running result or HDFS logs.

#### Procedure

- **Learn the application running conditions by viewing the running result.**

- The running result of the HDFS example application is shown as follows:

```
0 [main] INFO org.apache.hadoop.security.UserGroupInformation - Login successful for user
hdfsDevelop using keytab file user.keytab
1 [main] INFO com.huawei.hadoop.security.LoginUtil - Login success!!!!!!!!!!!!!!
568 [main] WARN org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
582 [main] WARN org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory - The short-
circuit local reads feature cannot be used because libhadoop cannot be loaded.
793 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to create path /
user/hdfs-examples
969 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to write.
1068 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to append.
1191 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I am
bigdata. It is successful if you can see me.I append this content.
1191 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to read.
1202 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete the file /
user/hdfs-examples/test.txt
1210 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete path /
user/hdfs-examples
1223 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
create path /user/hdfs-examples/hdfs_example_0
1224 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
create path /user/hdfs-examples/hdfs_example_1
1261 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
write.
1264 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
write.
2807 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
append.
2810 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
append.
2861 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I
am bigdata. It is successful if you can see me.I append this content.
2861 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
read.
2866 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
delete the file /user/hdfs-examples/hdfs_example_0/test.txt
2874 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
delete path /user/hdfs-examples/hdfs_example_0
2874 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I
am bigdata. It is successful if you can see me.I append this content.
2874 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
read.
2879 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
delete the file /user/hdfs-examples/hdfs_example_1/test.txt
2885 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
delete path /user/hdfs-examples/hdfs_example_1
```

- The running result of the Colocation example application is shown as follows:

```
0 [main] INFO com.huawei.hadoop.security.LoginUtil - JaasConfiguration
loginContextName=Client principal=hdfsDevelop useTicketCache=false keytabFile=/opt/
hdfsDemo/conf/user.keytab
817 [main] INFO org.apache.hadoop.security.UserGroupInformation - Login successful for user
hdfsDevelop using keytab file user.keytab
817 [main] INFO com.huawei.hadoop.security.LoginUtil - Login success!!!!!!!!!!!!!!
```

```
1380 [main] WARN org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
...
Create Group has finished.
Put file is running...
Put file has finished.
Delete file is running...
Delete file has finished.
Delete Group is running...
Delete Group has finished.
...
```

- **Learn the application running conditions by viewing HDFS logs.**  
The namenode logs of HDFS offer immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 14.5 More Information

### 14.5.1 Common API Introduction

#### 14.5.1.1 Java API

For details about Hadoop distributed file system (HDFS) APIs, see:

<http://hadoop.apache.org/docs/r3.1.1/api/index.html>.

### HDFS Common API

Common HDFS Java classes are as follows:

- **FileSystem**: the core class of client applications. For details about common APIs, see [Table 14-7](#).
- **FileStatus**: record the status of files and directories. For details about common APIs, see [Table 14-8](#).
- **DFSColocationAdmin**: API used to manage colocation group information. For details about common APIs, see [Table 14-9](#).
- **DFSColocationClient**: API used to manage colocation files. For details about common APIs, see [Table 14-10](#).

#### NOTE

- The system reserves only the mapping between nodes and locator IDs, but does not reserve the mapping between files and locator IDs. When a file is created using a Colocation interface, the file is created on the node that corresponds to a locator ID. File creation and writing must be performed using Colocation interfaces.
- After the file is written, subsequent operations on the file can use other open-source interfaces in addition to Colocation interfaces.
- The **DFSColocationClient** class inherits from the open-source **DistributedFileSystem** class and contains common file operation functions. If a user uses the **DFSColocationClient** class to create a Colocation file, the user is advanced to use the functions of this class in file operations.



**Table 14-7** Common FileSystem APIs

API	Description
public static FileSystem get(Configuration conf)	FileSystem is the API class provided for users in the Hadoop class library. FileSystem is an abstract class. Concrete classes can be obtained only using the get method. The get method has multiple overload versions and is commonly used.
public FSDataOutputStream create(Path f)	This API is used to create files in the HDFS. <i>f</i> indicates a complete file path.
public void copyFromLocalFile(Path src, Path dst)	This API is used to upload local files to a specified directory in the HDFS. <i>src</i> and <i>dst</i> indicate complete file paths.
public boolean mkdirs(Path f)	This API is used to create folders in the HDFS. <i>f</i> indicates a complete folder path.
public abstract boolean rename(Path src, Path dst)	This API is used to rename a specified HDFS file. <i>src</i> and <i>dst</i> indicate complete file paths.
public abstract boolean delete(Path f, boolean recursive)	This API is used to delete a specified HDFS file. <i>f</i> indicates the complete path of the file to be deleted, and <b>recursive</b> specifies recursive deletion.
public boolean exists(Path f)	This API is used to query a specified HDFS file. <i>f</i> indicates a complete file path.
public FileStatus getFileStatus(Path f)	This API is used to obtain the FileStatus object of a file or folder. The FileStatus object records status information of the file or folder, including the modification time and file directory.
public BlockLocation[] getFileBlockLocations(FileStatus file, long start, long len)	This API is used to query the block location of a specified file in an HDFS cluster. <i>file</i> indicates a complete file path, and <i>start</i> and <i>len</i> specify the block scope.
public FSDataInputStream open(Path f)	This API is used to open the output stream of a specified file in the HDFS and read the file using the API provided by the FSDataInputStream class. <i>f</i> indicates a complete file path.
public FSDataOutputStream create(Path f, boolean overwrite)	This API is used to create the input stream of a specified file in the HDFS and write the file using the API provided by the FSDataOutputStream class. <i>f</i> indicates a complete file path. If <b>overwrite</b> is <b>true</b> , the file is rewritten if it exists; if <b>overwrite</b> is <b>false</b> , an error is reported if the file exists.

API	Description
public FSDataOutputStream append(Path f)	This API is used to open the input stream of a specified file in the HDFS and write the file using the API provided by the FSDataOutputStream class <i>f</i> indicates a complete file path.

**Table 14-8** Common FileStatus APIs

API	Description
public long getModificationTime()	This API is used to query the modification time of a specified HDFS file.
public Path getPath()	This API is used to query all files in an HDFS directory.

**Table 14-9** Common DFSColocationAdmin APIs

API	Description
public Map<String, List<DatanodeInfo>> createColocationGroup(String groupId,String file)	This API is used to create a group based on the locatorIds information in the file. file indicates the file path.
public Map<String, List<DatanodeInfo>> createColocationGroup(String groupId,List<String> locators)	This API is used to create a group based on the locatorIds information in the list in the memory.
public void deleteColocationGroup(String groupId)	This API is used to delete a group.
public List<String> listColocationGroups()	This API is used to return all group information of Colocation. The returned group ID arrays are sorted by the creation time.
public List<DatanodeInfo> getNodesForLocator(String groupId, String locatorId)	This API is used to obtain the list of all nodes in the locator.

**Table 14-10** Common DFSColocationAdmin APIs

API	Description
public FSDataOutputStream create(Path f, boolean overwrite, String groupId,String locatorId)	This API is used to create a FSDataOutputStream in colocation mode to allow users to write files in f. f is the HDFS path. overwrite indicates whether an existing file can be overwritten. groupId and locatorId of the file specified by a user must exist.
public FSDataOutputStream create(final Path f, final FsPermission permission, final EnumSet<CreateFlag> cflags, final int bufferSize, final short replication, final long blockSize, final Progressable progress, final ChecksumOpt checksumOpt, final String groupId, final String locatorId)	The function of this API is the same as that of FSDataOutputStream create(Path f, boolean overwrite, String groupId, String locatorId), except that users are allowed to customize checksum.
public void close()	This API is used to close the connection.

**Table 14-11** HDFS client WebHdfsFileSystem API

API	Description
public RemoteIterator<FileStatus> listStatusIterator(final Path)	This API will help in fetching the child files and folders information through multiple requests using remote iterator, thus avoiding the user interface from becoming slow when there is a large amount of child information to be fetched.

## SmallFS Common API

The Java class SmallFileSystem common interfaces of the SmallFS are shown in [Table 14-12](#).

**Table 14-12** Description of Class SmallFileSystem Common Interfaces

Interface	Description
public void close()	Closes the connection after use.

Interface	Description
public void copyFromLocalFile(boolean delSrc, boolean overwrite, Path src, Path dst) throws IOException	This interface is used to upload the local file to the given location of the SmallFileSystem. src and dst indicate complete file paths.
public void copyFromLocalFile(boolean delSrc, boolean overwrite, Path[] srcs, Path dst) throws IOException	This interface is used to upload multiple local files to the given location of the SmallFileSystem. src and dst indicate complete file paths.
public void copyToLocalFile(boolean delSrc, Path src, Path dst, boolean useRawLocalFileSystem) throws IOException	This interface is used to download specified files of the SmallFileSystem to the local folder. src and dst indicate complete file paths.
public FSDataOutputStream create(Path path, FsPermission permission, boolean overwrite, int bufferSize, short replication, long blockSize, Progressable progress) throws IOException	This interface is used to create files in the given path in the SmallFileSystem.
public FileSystem[] getChildFileSystems()	This interface is used to obtain all sub-file systems of the SmallFileSystem.
public long getDefaultBlockSize()	This interface is used to obtain the default block size of the SmallFileSystem.
public short getDefaultReplication()	This interface is used to obtain the default backup counts of the SmallFileSystem.
public BlockLocation[] getFileBlockLocations(Path path, long start, long len) throws IOException	This interface is used to obtain the block location of the given file path.
public Path getHomeDirectory()	This interface is used to obtain the original path.
public String getScheme()	This interface is used to obtain the Schema of the SmallFileSystem.
public FsServerDefaults getServerDefaults() throws IOException	This interface is used to obtain the default configuration of the SmallFileSystem.
public void initialize(URI name, Configuration conf) throws IOException	This interface is used to initialize the SmallFileSystem.

Interface	Description
<p>public void setOwner(Path path, String username, String groupname) throws IOException</p>	<p>This interface is used to set the owner of the given path (file or path). The parameters username and groupname cannot both be null. <b>NOTE</b> The merged files do not support this interface.</p>
<p>public void setPermission(Path p, FsPermission permission) throws IOException</p>	<p>This interface is used to set the file permission of the given path (file or path). <b>NOTE</b> The merged files do not support this interface.</p>
<p>public boolean setReplication(Path path, short replication) throws IOException</p>	<p>This interface is used to set the backup counts of the given path (file or path). <b>NOTE</b> The merged files do not support this interface.</p>
<p>public void setTimes(Path path, long mtime, long atime) throws IOException</p>	<p>This interface is used to set the modification time and access time of the given path (file or path). <b>NOTE</b> The merged files do not support this interface.</p>
<p>public boolean delete(Path path, boolean recursive) throws IOException</p>	<p>This interface is used to delete the given file path (file or path) in the SmallFileSystem.</p>
<p>public FileStatus getFileStatus(Path path) throws IOException</p>	<p>This interface is used to obtain the FsStatus object in the designated partition of the SmallFileSystem. The object records information such as the total capacity of the partition, used capacity, and remaining capacity.</p>
<p>public URI getUri()</p>	<p>This interface is used to return the default URI of the SmallFileSystem.</p>
<p>public Path getWorkingDirectory()</p>	<p>This interface is used to obtain the current working directory for the SmallFileSystem.</p>
<p>public FileStatus[] listStatus(Path path) throws IOException</p>	<p>This interface is used to list the statuses of files/directories in the given path if the path is a directory.</p>
<p>public boolean mkdirs(Path path, FsPermission permission) throws IOException</p>	<p>This interface is used to create files or folders for the given path. <b>NOTE</b> The merged files do not support this interface.</p>

Interface	Description
<p>public FSDataInputStream open(Path path, int bufferSize) throws IOException</p>	<p>This interface is used to open the output stream of the specified file in the SmallFileSystem and read files through the interface provided by the FSDataInputStream class. path indicates a complete file path.</p>
<p>public boolean rename(Path src, Path dst) throws IOException</p>	<p>This interface is used to rename the specified files of the SmallFileSystem. src and dst indicate complete file paths.</p>
<p>public void setWorkingDirectory(Path path)</p>	<p>The method is rewritten in the SmallFileSystem and cannot be set to other work directories.</p>
<p>public Configuration getConf()</p>	<p>This interface is used to obtain the configuration of the SmallFileSystem.</p>
<p>public Path getInitialWorkingDirectory()</p>	<p>This interface is used to obtain the initial working directory of the SmallFileSystem.</p>
<p>public FsStatus getStatus(Path path) throws IOException</p>	<p>This interface is used to return a status object describing the usage and capacity of the file system. If the file system has multiple partitions, the usage and capacity of the partition pointed by the specified path is reflected.</p>
<p>public FSDataOutputStream createNonRecursive(Path path, boolean overwrite, int bufferSize, short replication, long blockSize, Progressable progress) throws IOException</p>	<p>This interface is used to open an FSDataOutputStream at the indicated path with write-progress reporting. Same as create(), except fails if the parent directory does not exist.</p> <p><b>NOTE</b> This interface is not recommended. Use the create interface instead.</p>
<p>public FSDataOutputStream append(Path path) throws IOException</p>	<p>This interface is used to add the additional content to the specified file path in the SmallFileSystem.</p> <p><b>NOTE</b> The merged files do not support this interface.</p>
<p>public boolean truncate(Path path, long newLength) throws IOException</p>	<p>This interface is used to cut relevant content of the specified file path in the SmallFileSystem.</p> <p><b>NOTE</b> The merged files do not support this interface.</p>

Interface	Description
public FsServerDefaults getServerDefaults(Path path) throws IOException	This interface is used to obtain the FsServerDefaults object of the target file system in the designated path. The object records information such as block size, backup count, and garbage retention time.
public long getUsed() throws IOException	This interface is used to return the total size of all files in the filesystem.
public long getDefaultBlock- Size(Path path)	This interface is used to obtain the default block size of the specified file path.
public short getDefaultReplica- tion(Path path)	This interface is used to obtain the default backup counts of the specified file path.

## Glob path pattern based API to get LocatedFileStatus and Open file from FileStatus

Following APIs are added in DistributedFileSystem to get the FileStatus with block location and open file from FileStatus object. These APIs will reduce the number of RPC calls from client to Namenodes.

**Table 14-13** FileSystem APIs

Interface	Description
public LocatedFileStatus[] globLocatedStatus(Path, PathFilter, boolean) throws IOException	Return an array of LocatedFileStatus objects whose path names match pathPattern and pass the in path filter.
public FSDatInputStream open(FileStatus stat) throws IOException	If the stat is an instance of LocatedFileStatusHdfs that already have the location information, the InputStream is created without contacting NameNode.

### 14.5.1.2 C API

#### Function Description

Users can use the C application programming interface (API) to create, read and write, append, and delete files. For details of the C API, see the following official guidelines:

<http://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-hdfs/LibHdfs.html>

## Code Sample

The following code snippets are used as an example. For complete code, see the HDFS C sample code **HDFS/hdfs-c-example/hdfs\_test.c** in the HDFS sample code decompression directory.

1. Configure the HDFS NameNode parameter and create the link connecting to the HDFS files.

```
hdfsFS fs = hdfsConnect("default", 0);
fprintf(stderr, "hdfsConnect- SUCCESS!\n");
```

2. Create the HDFS directory.

```
const char* dir = "/tmp/nativeTest";
int exitCode = hdfsCreateDirectory(fs, dir);
if(exitCode == -1){
 fprintf(stderr, "Failed to create directory %s \n", dir);
 exit(-1);
}
fprintf(stderr, "hdfsCreateDirectory- SUCCESS! : %s\n", dir);
```

3. Write files.

```
const char* file = "/tmp/nativeTest/testfile.txt";
hdfsFile writeFile = openFile(fs, (char*)file, O_WRONLY | O_CREAT, 0, 0, 0);
fprintf(stderr, "hdfsOpenFile- SUCCESS! for write : %s\n", file);
```

```
if(!hdfsFilesOpenForWrite(writeFile)){
 fprintf(stderr, "Failed to open %s for writing.\n", file);
 exit(-1);
}
```

```
char* buffer = "Hadoop HDFS Native file write!";
```

```
hdfsWrite(fs, writeFile, (void*)buffer, strlen(buffer)+1);
fprintf(stderr, "hdfsWrite- SUCCESS! : %s\n", file);
```

```
printf("Flushing file data\n");
if (hdfsFlush(fs, writeFile) {
 fprintf(stderr, "Failed to 'flush' %s\n", file);
 exit(-1);
}
```

```
hdfsCloseFile(fs, writeFile);
fprintf(stderr, "hdfsCloseFile- SUCCESS! : %s\n", file);
```

4. Read files.

```
hdfsFile readFile = openFile(fs, (char*)file, O_RDONLY, 100, 0, 0);
fprintf(stderr, "hdfsOpenFile- SUCCESS! for read : %s\n", file);
```

```
if(!hdfsFilesOpenForRead(readFile)){
 fprintf(stderr, "Failed to open %s for reading.\n", file);
 exit(-1);
}
```

```
buffer = (char *) malloc(100);
tSize num_read = hdfsRead(fs, readFile, (void*)buffer, 100);
fprintf(stderr, "hdfsRead- SUCCESS!, Byte read : %d, File content : %s \n", num_read ,buffer);
hdfsCloseFile(fs, readFile);
```

5. From the specified location to read the file.

```
buffer = (char *) malloc(100);
readFile = openFile(fs, file, O_RDONLY, 100, 0, 0);
if (hdfsSeek(fs, readFile, 10) {
 fprintf(stderr, "Failed to 'seek' %s\n", file);
 exit(-1);
}
num_read = hdfsRead(fs, readFile, (void*)buffer, 100);
fprintf(stderr, "hdfsSeek- SUCCESS!, Byte read : %d, File seek content : %s \n", num_read ,buffer);
hdfsCloseFile(fs, readFile);
```

6. Copy the file.



```
const char* destfile = "/tmp/nativeTest/testfile1.txt";
if (hdfsCopy(fs, file, fs, destfile)) {
 fprintf(stderr, "File copy failed, src : %s, des : %s \n", file, destfile);
 exit(-1);
}
fprintf(stderr, "hdfsCopy- SUCCESS!, File copied, src : %s, des : %s \n", file, destfile);
```

7. Move the file.

```
const char* mvfile = "/tmp/nativeTest/testfile2.txt";
if (hdfsMove(fs, destfile, fs, mvfile)) {
 fprintf(stderr, "File move failed, src : %s, des : %s \n", destfile, mvfile);
 exit(-1);
}
fprintf(stderr, "hdfsMove- SUCCESS!, File moved, src : %s, des : %s \n", destfile, mvfile);
```

8. Rename the file.

```
const char* renamefile = "/tmp/nativeTest/testfile3.txt";
if (hdfsRename(fs, mvfile, renamefile)) {
 fprintf(stderr, "File rename failed, Old name : %s, New name : %s \n", mvfile, renamefile);
 exit(-1);
}
fprintf(stderr, "hdfsRename- SUCCESS!, File renamed, Old name : %s, New name : %s \n", mvfile, renamefile);
```

9. Delete Files.

```
if (hdfsDelete(fs, renamefile, 0)) {
 fprintf(stderr, "File delete failed : %s \n", renamefile);
 exit(-1);
}
fprintf(stderr, "hdfsDelete- SUCCESS!, File deleted : %s\n", renamefile);
```

10. Set the number of replications.

```
if (hdfsSetReplication(fs, file, 10)) {
 fprintf(stderr, "Failed to set replication : %s \n", file);
 exit(-1);
}
fprintf(stderr, "hdfsSetReplication- SUCCESS!, Set replication 10 for %s\n", file);
```

11. Set users, user groups.

```
if (hdfsChown(fs, file, "root", "root")) {
 fprintf(stderr, "Failed to set chown : %s \n", file);
 exit(-1);
}
fprintf(stderr, "hdfsChown- SUCCESS!, Chown success for %s\n", file);
```

12. Set permissions.

```
if (hdfsChmod(fs, file, S_IRWXU | S_IRWXG | S_IRWXO)) {
 fprintf(stderr, "Failed to set chmod: %s \n", file);
 exit(-1);
}
fprintf(stderr, "hdfsChmod- SUCCESS!, Chmod success for %s\n", file);
```

13. Set the file time.

```
struct timeval now;
gettimeofday(&now, NULL);
if (hdfsUtime(fs, file, now.tv_sec, now.tv_sec)) {
 fprintf(stderr, "Failed to set time: %s \n", file);
 exit(-1);
}
fprintf(stderr, "hdfsUtime- SUCCESS!, Set time success for %s\n", file);
```

14. Get file information.

```
hdfsFileInfo *fileInfo = NULL;
if((fileInfo = hdfsGetPathInfo(fs, file)) != NULL) {
 printFileInfo(fileInfo);
 hdfsFreeFileInfo(fileInfo, 1);
 fprintf(stderr, "hdfsGetPathInfo - SUCCESS!\n");
}
```

15. Variable directory.

```
hdfsFileInfo *fileList = 0;
int numEntries = 0;
```

```
if((fileList = hdfsListDirectory(fs, dir, &numEntries)) != NULL) {
 int i = 0;
 for(i=0; i < numEntries; ++i) {
 printFileInfo(fileList+i);
 }
 hdfsFreeFileInfo(fileList, numEntries);
}
fprintf(stderr, "hdfsListDirectory- SUCCESS!, %s\n", dir);
```

#### 16. Stream builder interfaces.

```
buffer = (char *) malloc(100);
struct hdfsStreamBuilder *builder= hdfsStreamBuilderAlloc(fs, (char*)file, O_RDONLY);
hdfsStreamBuilderSetBufferSize(builder,100);
hdfsStreamBuilderSetReplication(builder,20);
hdfsStreamBuilderSetDefaultBlockSize(builder,10485760);
readFile = hdfsStreamBuilderBuild(builder);
num_read = hdfsRead(fs, readFile, (void*)buffer, 100);
fprintf(stderr, "hdfsStreamBuilderBuild- SUCCESS! File read success. Byte read : %d, File content : %s\n", num_read ,buffer);
free(buffer);

struct hdfsReadStatistics *stats = NULL;
hdfsFileGetReadStatistics(readFile, &stats);
fprintf(stderr, "hdfsFileGetReadStatistics- SUCCESS! totalBytesRead : %" PRIu64 ",
totalLocalBytesRead : %" PRIu64 ", totalShortCircuitBytesRead : %" PRIu64 ",
totalZeroCopyBytesRead : %" PRIu64 "\n", stats->totalBytesRead , stats->totalLocalBytesRead, stats->totalShortCircuitBytesRead, stats->totalZeroCopyBytesRead);
hdfsFileFreeReadStatistics(stats);
```

#### 17. Disconnect the HDFS links.

```
hdfsDisconnect(fs);
```

## Preparing Running Environment

Install a client on the node. For example, install a client in the `/opt/client` directory.

## Compiling and Running Applications in Linux

1. Go to the `/opt/client` directory and run the following command to import the environment variables of the C client:

```
cd /opt/client
source bigdata_env
```

2. In the directory, run the following command as the hdfs user. For the user password, contact the cluster administrator.

```
kinit hdfs
```

#### NOTE

The validity duration of kinit authentication is 24 hours. After 24 hours, you need to re-authenticate the sample with the kinit to restart the sample.

3. Go to the `/opt/client/HDFS/hadoop/hdfs-c-example` directory and run the following command to import the environment variables of the C client:

```
cd /opt/client/HDFS/hadoop/hdfs-c-example
source component_env_C_example
```

4. Run the following command to clean the object files and executable files that are generated before:

```
make clean
```

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make clean
rm -f hdfs_test.o
rm -f hdfs_test
```

5. Run the following command to compile the new object files and executable files:

**make (or make all)**

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make
cc -c -I/opt/client/HDFS/hadoop/include -Wall -o hdfs_test.o hdfs_test.c
cc -o hdfs_test hdfs_test.o -lhdfs
```

6. Run the following command to create, write, read, append, and delete files:

**make run**

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make run
./hdfs_test
hdfsConnect- SUCCESS!
hdfsCreateDirectory- SUCCESS! : /tmp/nativeTest
hdfsOpenFile- SUCCESS! for write : /tmp/nativeTest/testfile.txt
hdfsWrite- SUCCESS! : /tmp/nativeTest/testfile.txt
Flushing file data
hdfsCloseFile- SUCCESS! : /tmp/nativeTest/testfile.txt
hdfsOpenFile- SUCCESS! for read : /tmp/nativeTest/testfile.txt
hdfsRead- SUCCESS!, Byte read : 31, File content : Hadoop HDFS Native file write!
hdfsSeek- SUCCESS!, Byte read : 21, File seek content : S Native file write!
hdfsPread- SUCCESS!, Byte read : 10, File read content : S Native f
hdfsCopy- SUCCESS!, File copied, src : /tmp/nativeTest/testfile.txt, des : /tmp/nativeTest/testfile1.txt
hdfsMove- SUCCESS!, File moved, src : /tmp/nativeTest/testfile1.txt, des : /tmp/nativeTest/testfile2.txt
hdfsRename- SUCCESS!, File renamed, Old name : /tmp/nativeTest/testfile2.txt, New name : /tmp/
nativeTest/testfile3.txt
hdfsDelete- SUCCESS!, File deleted : /tmp/nativeTest/testfile3.txt
hdfsSetReplication- SUCCESS!, Set replication 10 for /tmp/nativeTest/testfile.txt
hdfsChown- SUCCESS!, Chown success for /tmp/nativeTest/testfile.txt
hdfsChmod- SUCCESS!, Chmod success for /tmp/nativeTest/testfile.txt
hdfsUtime- SUCCESS!, Set time success for /tmp/nativeTest/testfile.txt

Name: hdfs://hacluster/tmp/nativeTest/testfile.txt, Type: F, Replication: 10, BlockSize: 134217728, Size:
31, LastMod: 1500345260, Owner: root, Group: root, Permissions: 511 (rwxrwxrwx)
hdfsGetPathInfo - SUCCESS!

Name: hdfs://hacluster/tmp/nativeTest/testfile.txt, Type: F, Replication: 10, BlockSize: 134217728, Size:
31, LastMod: 1500345260, Owner: root, Group: root, Permissions: 511 (rwxrwxrwx)
hdfsListDirectory- SUCCESS!, /tmp/nativeTest
hdfsTruncateFile- SUCCESS!, /tmp/nativeTest/testfile.txt
Block Size : 134217728
hdfsGetDefaultBlockSize- SUCCESS!
Block Size : 134217728 for file /tmp/nativeTest/testfile.txt
hdfsGetDefaultBlockSizeAtPath- SUCCESS!
HDFS Capacity : 102726873909
hdfsGetCapacity- SUCCESS!
HDFS Used : 4767076324
hdfsGetCapacity- SUCCESS!
hdfsExists- SUCCESS! /tmp/nativeTest/testfile.txt
hdfsConfGetStr- SUCCESS : hdfs://hacluster
hdfsStreamBuilderBuild- SUCCESS! File read success. Byte read : 31, File content : Hadoop HDFS
Native file write!
hdfsFileGetReadStatistics- SUCCESS! totalBytesRead : 31, totalLocalBytesRead : 0,
totalShortCircuitBytesRead : 0, totalZeroCopyBytesRead : 0
```

7. Enter the debug mode (optional)

**make gdb**

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make gdb
gdb hdfs_test
```

```
GNU gdb (GDB) SUSE (7.5.1-0.7.29)
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-suse-linux".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /opt/client/HDFS/hadoop/hdfs-c-example/hdfs_test...done.
(gdb)
```

### 14.5.1.3 HTTP REST API

#### Function Description

Users can use the application programming interface (API) of Representational State Transfer (REST) to create, read and write, append, and delete files. For details of the REST API, see the following official guidelines:

<http://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-hdfs/WebHDFS.html>.

#### Preparing Running Environment

**Step 1** Install the client. Install the client on the node. For example, install the client in the `/opt/client` directory.

1. Run the following command to for the user authentication. In the command, the `hdfs` is taken as an example and can be defined by users.

```
kinit hdfs
```

#### NOTE

The validity duration of `kinit` authentication is 24 hours. After 24 hours, you need to re-authenticate the sample with the `kinit` to restart the sample.

2. Prepare files `testFile` and `testFileAppend` and write content 'Hello, webhdfs user!' and 'Welcome back to webhdfs!'. Run the following command to prepare `testFile` and `testFileAppend` files:

```
touch testFile
```

```
vi testFile
```

```
Hello, webhdfs user!
```

```
touch testFileAppend
```

```
vi testFileAppend
```

```
Welcome back to webhdfs!
```

**Step 2** The MRS cluster supports only the HTTPS access by default. If access by using the HTTPS service, perform [Step 3](#). If access by using the HTTP service, perform [Step 4](#).

**Step 3** HTTPS-based access is different from HTTP-based access. When you access HDFS using HTTPS, you must ensure that the SSL protocol supported by the `curl` command is supported by the cluster because SSL security encryption is used. If the cluster does not support the SSL protocol, change the SSL protocol in the cluster. For example, if the `Curl` command only supports the TLSv1 protocol, modify the protocol configuration performing following measures:

Log in to FusionInsight Manager and choose **Cluster** > *Name of the desired cluster* > **Services** > **HDFS** > **Configurations** > **All Configurations**. Type **hadoop.ssl.enabled.protocols** in the research box, check whether the parameter value contains **TLSv1**. If the parameter value does not contain **TLSv1**, add **TLSv1** in the **hadoop.ssl.enabled.protocols** configuration item, and clear the value of **ssl.server.exclude.cipher.list**. Otherwise, HDFS cannot be accessed by using HTTPS. Then click **Save** and click **More** > **Restart Service** to restart the HDFS service.

 **NOTE**

TLSv1 has security vulnerabilities. Exercise caution when using it.

**Step 4** **Log in to the FusionInsight Manager portal**, choose **Cluster** > *Name of the desired cluster* > **Services** > **HDFS** > **Configurations** > **All Configurations**. Type **dfs.http.policy** in the research box, select **HTTP\_AND\_HTTPS**, click **Save**, and select **More** > **Restart Service** to restart the HDFS service.

----End

## Procedure

**Step 1** **Log in to the FusionInsight Manager portal**, click **Cluster** > *Name of the desired cluster* > **Services**, and then select **HDFS**. The HDFS page is displayed.

 **NOTE**

Because webhdfs is accessed through HTTP/HTTPS, you need to obtain the IP address of the active NameNode and the HTTP/HTTPS port.

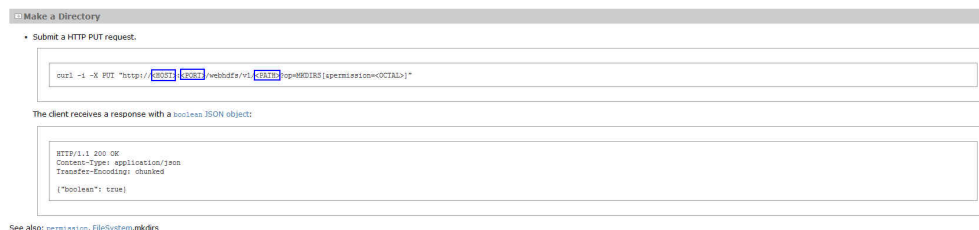
1. Click **Instance**, and in HDFS Instances page, view the host name and IP address of the active NameNode.
2. Click **Configurations**, and in HDFS Service Configuration page, find **namenode.https.port** (9870) and **namenode.https.port** (9871).

**Step 2** Create a directory by referring to the following link:

[http://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-hdfs/WebHDFS.html#Make\\_a\\_Directory](http://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-hdfs/WebHDFS.html#Make_a_Directory)

Click the link, **Figure 14-10** is displayed:

**Figure 14-10** Example code of creating a directory



Go to the **/opt/client** directory, the installation directory of the client, and create the **huawei** directory.

1. Run the following command to check whether the **huawei** directory exists in the current path.

### hdfs dfs -ls /

The running results are as follows:

```
linux1:/opt/client # hdfs dfs -ls /
16/04/22 16:10:02 INFO hdfs.PeerCache: SocketCache disabled.
Found 7 items
-rw-r--r-- 3 hdfs supergroup 0 2016-04-20 18:03 /PRE_CREATE_DIR.SUCCESS
drwxr-x--- - flume hadoop 0 2016-04-20 18:02 /flume
drwx----- - hbase hadoop 0 2016-04-22 15:19 /hbase
drwxrwxrwx - mapred hadoop 0 2016-04-20 18:02 /mr-history
drwxrwxrwx - spark supergroup 0 2016-04-22 15:19 /sparkJobHistory
drwxrwxrwx - hdfs hadoop 0 2016-04-22 14:51 /tmp
drwxrwxrwx - hdfs hadoop 0 2016-04-22 14:50 /user
```

The **huawei** directory does not exist in the current path.

2. Run the command in [Figure 14-10](#) that is named with **huawei**. Replace the <HOST> and <PORT> in the command with the host name or IP address and port number that are obtained in [Step 1](#). Type the **huawei** as the directory in the <PATH>.

#### NOTE

The <HOST> can be replaced by the host name or IP address. It is noted that the port of HTTP is different from the port of HTTPS.

- Run the following command to access HTTP:

```
linux1:/opt/client # curl -i -X PUT --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei?op=MKDIRS"
```

In the command, the <HOST> is replaced by linux 1 and the <PORT> is replaced by 9870.

- The running result is displayed as follows:

```
HTTP/1.1 401 Authentication required
Date: Thu, 05 May 2016 03:10:09 GMT
Pragma: no-cache
Date: Thu, 05 May 2016 03:10:09 GMT
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; HttpOnly
Content-Length: 0
HTTP/1.1 200 OK
Cache-Control: no-cache
Expires: Thu, 05 May 2016 03:10:09 GMT
Date: Thu, 05 May 2016 03:10:09 GMT
Pragma: no-cache
Expires: Thu, 05 May 2016 03:10:09 GMT
Date: Thu, 05 May 2016 03:10:09 GMT
Pragma: no-cache
Content-Type: application/json
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate YGoGCSqGS1b3EgECAglAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKiRARCArhuv39Ttp6lhBlG3B0JAmFjv9weLp+SGFI+t2HSEHN6p4UVWKKy/
kd9dKEgNMLyDu/o7ytzs0cqMxNsl69WbN5H
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1462453809395&s=wiRF4rdTWpm3tDST+a/Sy0lwgA4="; Path=/;
Expires=Thu, 05-May-2016 13:10:09 GMT; HttpOnly
Transfer-Encoding: chunked
{"boolean":true}linux1:/opt/client #
```

If {"boolean":true} returns, the **huawei** directory is successfully created.

- Run the following command to access HTTPS:

```
linux1:/opt/client # curl -i -k -X PUT --negotiate -u: "https://10.120.172.109:9871/webhdfs/v1/huawei?op=MKDIRS"
```

In the command, the <HOST> is replaced by IP address (10.120.172.109) and the <PORT> is replaced by 9871.

- The running result is displayed as follows:

```
HTTP/1.1 401 Authentication required
Date: Fri, 22 Apr 2016 08:13:37 GMT
Pragma: no-cache
Date: Fri, 22 Apr 2016 08:13:37 GMT
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; Secure; HttpOnly
Content-Length: 0
HTTP/1.1 200 OK
Cache-Control: no-cache
Expires: Fri, 22 Apr 2016 08:13:37 GMT
Date: Fri, 22 Apr 2016 08:13:37 GMT
Pragma: no-cache
Expires: Fri, 22 Apr 2016 08:13:37 GMT
Date: Fri, 22 Apr 2016 08:13:37 GMT
Pragma: no-cache
Content-Type: application/json
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate YGoGCSqGS1b3EgECAglAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKiRARCugB+yT3Y+z8YCRMYJHXF84o1cyCflq157+NZN1gu7D7yhMULnjr
+7BuUdEcZKewFR7uD+DRiMY3akg3OgU45xQ9R
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1461348817963&s=sh57G7iVccX/Aknoz410yJPTLHg="; Path=/;
Expires=Fri, 22-Apr-2016 18:13:37 GMT; Secure; HttpOnly
Transfer-Encoding: chunked

{"boolean":true}linux1:/opt/client #
```

If {"boolean":true} returns, the **huawei** directory is successfully created.

3. Run the following command to check the **huawei** directory in the path.

```
linux1:/opt/client # hdfs dfs -ls /
16/04/22 16:14:25 INFO hdfs.PeerCache: SocketCache disabled.
Found 8 items
-rw-r--r-- 3 hdfs supergroup 0 2016-04-20 18:03 /PRE_CREATE_DIR.SUCCESS
drwxr-x--- - flume hadoop 0 2016-04-20 18:02 /flume
drwx----- - hbase hadoop 0 2016-04-22 15:19 /hbase
drwxr-xr-x - hdfs supergroup 0 2016-04-22 16:13 /huawei
drwxrwxrwx - mapred hadoop 0 2016-04-20 18:02 /mr-history
drwxrwxrwx - spark supergroup 0 2016-04-22 16:12 /sparkJobHistory
drwxrwxrwx - hdfs hadoop 0 2016-04-22 14:51 /tmp
drwxrwxrwx - hdfs hadoop 0 2016-04-22 16:10 /user
```

### Step 3 Create a command of the upload request to obtain the information about Location where the DataNode IP address is written in.

- Run the following command to access HTTP:

```
linux1:/opt/client # curl -i -X PUT --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs?op=CREATE"
```

- The running result is displayed as follows:

```
HTTP/1.1 401 Authentication required
Date: Thu, 05 May 2016 06:09:48 GMT
Pragma: no-cache
Date: Thu, 05 May 2016 06:09:48 GMT
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; HttpOnly
Content-Length: 0

HTTP/1.1 307 TEMPORARY_REDIRECT
Cache-Control: no-cache
Expires: Thu, 05 May 2016 06:09:48 GMT
Date: Thu, 05 May 2016 06:09:48 GMT
Pragma: no-cache
Expires: Thu, 05 May 2016 06:09:48 GMT
Date: Thu, 05 May 2016 06:09:48 GMT
```

```

Pragma: no-cache
Content-Type: application/octet-stream
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate YGoGCSqGSib3EgECAglAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKIRARCzQ6w
+9pNzWCTJEdoU3z9xKEyg1JQNka0nYaB9TndvrL5S0neAoK2usnictTfnqIncajwB6SnTtht8Q16WDIHJX/
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1462464588403&s=qry87vAyYzSn9VsS6Rm6vKLhKeU="; Path=/; Expires=Thu,
05-May-2016 16:09:48 GMT; HttpOnly
Location: http://linux1:25010/webhdfs/v1/huawei/testHdfs?
op=CREATE&delegation=HgAFYWRtaW4FYWRtaW4AigFUf4lZdl0BVKOV3XQOCBSyXvFAP92alcRs4j-
KNuInN6wUoBJXRUIREZTIGRlbgVnYXRpb24UMTAuMTIwLjE3Mi4xMDk6MjUwMDA&namenoderpcaddr
ess=hacluster&overwrite=false
Content-Length: 0

```

- Run the following command to access HTTPS:

```
linux1:/opt/client # curl -i -k -X PUT --negotiate -u: "https://linux1:25003/webhdfs/v1/huawei/
testHdfs?op=CREATE"
```

- The running result is displayed as follows:

```

HTTP/1.1 401 Authentication required
Date: Thu, 05 May 2016 03:46:18 GMT
Pragma: no-cache
Date: Thu, 05 May 2016 03:46:18 GMT
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; Secure; HttpOnly
Content-Length: 0

```

```

HTTP/1.1 307 TEMPORARY_REDIRECT
Cache-Control: no-cache
Expires: Thu, 05 May 2016 03:46:18 GMT
Date: Thu, 05 May 2016 03:46:18 GMT
Pragma: no-cache
Expires: Thu, 05 May 2016 03:46:18 GMT
Date: Thu, 05 May 2016 03:46:18 GMT
Pragma: no-cache
Content-Type: application/octet-stream
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate YGoGCSqGSib3EgECAglAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKIRARCZMYR8GGUkn7pPZaoOYZD5HxzLTRZ71angUHKubW2wC/18m9/
OOZstGQ6M1wH2pGriipuCNsKlIwP93eO2Co0fQF3
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1462455978166&s=F4rXUwEevHZze3PR8TxkzcV7RQQ="; Path=/; Expires=Thu,
05-May-2016 13:46:18 GMT; Secure; HttpOnly
Location: https://linux1:9865/webhdfs/v1/huawei/testHdfs?
op=CREATE&delegation=HgAFYWRtaW4FYWRtaW4AigFUfwX3t4oBVKMSe7cCCBSFJT9j7X64QwnS59T
GFPKFF7GhNTV0VCSERGUyBkZWxlZ2F0aW9uFDEwLjEyMC4xNzluMTA5Oj11MDAw&namenoderpcaddr
ess=hacluster&overwrite=false
Content-Length: 0

```

**Step 4** According to the Location information, create the **testHdfs** file in the **/huawei/** **testHdfs** file on the HDFS and upload the content in the local **testFile** file into the **testHdfs** file.

- Run the following command to access HTTP:

```
linux1:/opt/client # curl -i -X PUT -T testFile --negotiate -u: "http://linux1:9864/webhdfs/v1/huawei/
testHdfs?
op=CREATE&delegation=HgAFYWRtaW4FYWRtaW4AigFUf4lZdl0BVKOV3XQOCBSyXvFAP92alcRs4j-
KNuInN6wUoBJXRUIREZTIGRlbgVnYXRpb24UMTAuMTIwLjE3Mi4xMDk6MjUwMDA&namenoderpcaddr
ess=hacluster&overwrite=false"
```

- The running result is displayed as follows:

```

HTTP/1.1 100 Continue
HTTP/1.1 201 Created
Location: hdfs://hacluster/huawei/testHdfs
Content-Length: 0
Connection: close

```



- Run the following command to access HTTPS:  

```
linux1:/opt/client # curl -i -k -X PUT -T testFile --negotiate -u: "https://linux1:9865/webhdfs/v1/huawei/testHdfs?op=CREATE&delegation=HgAFYWRtaW4FYWRtaW4AigFUfwX3t4oBVKMSe7cCCBSFJT9j7X64QwnSz59TGFPKFf7GhNTV0VCSERGUyBkZWxlZ2F0aW9uFEwLjEyMC4xNzluMTA5OjI1MDAw&namenoderpcaddress=hacluster&overwrite=false"
```
- The running result is displayed as follows:  

```
HTTP/1.1 100 Continue
HTTP/1.1 201 Created
Location: hdfs://hacluster/huawei/testHdfs
Content-Length: 0
Connection: close
```

**Step 5** Go to the **/huawei/testHdfs** directory and read the content of **testHdfs** file.

- Run the following command to access HTTP:  

```
linux1:/opt/client # curl -L --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs?op=OPEN"
```
- The running result is displayed as follows:  

```
Hello, webhdfs user!
```
- Run the following command to access HTTPS:  

```
linux1:/opt/client # curl -k -L --negotiate -u: "https://linux1:9871/webhdfs/v1/huawei/testHdfs?op=OPEN"
```
- The running result is displayed as follows:  

```
Hello, webhdfs user!
```

**Step 6** Create a command of the upload request to obtain the information about Location where the DataNode IP address of **testHdfs** file is written in.

- Run the following command to access HTTP:  

```
linux1:/opt/client # curl -i -X POST --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs?op=APPEND"
```
- The running result is displayed as follows:  

```
HTTP/1.1 401 Authentication required
Cache-Control: must-revalidate,no-cache,no-store
Date: Thu, 05 May 2016 05:35:02 GMT
Pragma: no-cache
Date: Thu, 05 May 2016 05:35:02 GMT
Pragma: no-cache
Content-Type: text/html; charset=iso-8859-1
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; HttpOnly
Content-Length: 1349

HTTP/1.1 307 TEMPORARY_REDIRECT
Cache-Control: no-cache
Expires: Thu, 05 May 2016 05:35:02 GMT
Date: Thu, 05 May 2016 05:35:02 GMT
Pragma: no-cache
Expires: Thu, 05 May 2016 05:35:02 GMT
Date: Thu, 05 May 2016 05:35:02 GMT
Pragma: no-cache
Content-Type: application/octet-stream
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate YGoGCSqGSib3EgECAGlAb1swWaADAgEFoQMCAQ+iTTBLoAMCARKiRARTYvNX/2JMXhZsVPTw3Sluox6s/gEroHH980xMBkkYlCnO3W+0fM32c4/F98U5bl5dzgoolQoBvqq/EYXivvR12WX
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain name>&t=kerberos&e=1462462502626&s=et1okVIOd7DWJ/LdhzNeS2wQEEY="; Path=/; Expires=Thu, 05-May-2016 15:35:02 GMT; HttpOnly
Location: http://linux1:9864/webhdfs/v1/huawei/testHdfs?op=APPEND&delegation=HgAFYWRtaW4FYWRtaW4AigFUf2mGHooBVKN2Ch4KCBRzjM3jwSMIAowXb4dhqfKB5rT-8hJXRUIJIREZTIGRlbgVnYXRpb24UMTAuMTIwLjE3Mi4xMDk6MjUwMDA&namenoderpcaddress=hacluster
Content-Length: 0
```

- Run the following command to access HTTPS:  
linux1:/opt/client # curl -i -k -X POST --negotiate -u: "https://linux1:9871/webhdfs/v1/huawei/testHdfs?op=APPEND"
- The running result is displayed as follows:  
HTTP/1.1 401 Authentication required  
Cache-Control: must-revalidate,no-cache,no-store  
Date: Thu, 05 May 2016 05:20:41 GMT  
Pragma: no-cache  
Date: Thu, 05 May 2016 05:20:41 GMT  
Pragma: no-cache  
Content-Type: text/html; charset=iso-8859-1  
X-Frame-Options: SAMEORIGIN  
WWW-Authenticate: Negotiate  
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; Secure; HttpOnly  
Content-Length: 1349  
  
HTTP/1.1 307 TEMPORARY\_REDIRECT  
Cache-Control: no-cache  
Expires: Thu, 05 May 2016 05:20:41 GMT  
Date: Thu, 05 May 2016 05:20:41 GMT  
Pragma: no-cache  
Expires: Thu, 05 May 2016 05:20:41 GMT  
Date: Thu, 05 May 2016 05:20:41 GMT  
Pragma: no-cache  
Content-Type: application/octet-stream  
X-Frame-Options: SAMEORIGIN  
WWW-Authenticate: Negotiate YGoGCSqGSib3EgECAglAb1swWaADAgEFoQMCAQ  
+iTTBLoAMCARKiRARCXgdjZuoxLHGtM1oyrPcXk95/  
Y869eMfXIQV5UdEwBZ0iQiYaOdf5+Vk7a7FezhmzCABOWYXPxEQPNugbZ/yD5VLT  
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain  
name>&t=kerberos&e=1462461641713&s=tGwwOH9scmnNtxPjlnu28SFtex0="; Path=/; Expires=Thu,  
05-May-2016 15:20:41 GMT; Secure; HttpOnly  
Location: https://linux1:9865/webhdfs/v1/huawei/testHdfs?  
op=APPEND&delegation=HgAFYWRtaW4FYWRtaW4AigFUf1xi\_4oBVKN05v8HCBSE3Fg0f\_EwtFKKIODK  
QSM2t32CjhNTV0VCSERGUyBkZWxlZ2F0aW9uFDEwLjEyMC4xNzluMTA5OjI1MDAw&namenoderpcadd  
ress=hacluster

**Step 7** According to the Location information, add the content in the local **testFileAppend** file to the **testHdfs** file that is in the **/huawei/testHdfs** directory of HDFS.

- Run the following command to access HTTP:  
linux1:/opt/client # curl -i -X POST -T testFileAppend --negotiate -u: "http://linux1:9864/webhdfs/v1/huawei/testHdfs?  
op=APPEND&delegation=HgAFYWRtaW4FYWRtaW4AigFUf2mGHooBVKN2Ch4KCBRzjM3jwSMIAowXb  
4dhqfKB5rT-8hXRUIJIREZTIGRlbgVnYXRpb24UMTAuMTIwLjE3Mi4xMDk6MjUwMDA&namenoderpcadd  
ress=hacluster"
- The running result is displayed as follows:  
HTTP/1.1 100 Continue  
HTTP/1.1 200 OK  
Content-Length: 0  
Connection: close
- Run the following command to access HTTPS:  
linux1:/opt/client # curl -i -k -X POST -T testFileAppend --negotiate -u: "https://linux1:9865/  
webhdfs/v1/huawei/testHdfs?  
op=APPEND&delegation=HgAFYWRtaW4FYWRtaW4AigFUf1xi\_4oBVKN05v8HCBSE3Fg0f\_EwtFKKIODK  
QSM2t32CjhNTV0VCSERGUyBkZWxlZ2F0aW9uFDEwLjEyMC4xNzluMTA5OjI1MDAw&namenoderpcadd  
ress=hacluster"
- The running result is displayed as follows:  
HTTP/1.1 100 Continue  
HTTP/1.1 200 OK  
Content-Length: 0  
Connection: close

**Step 8** Go to the **/huawei/testHdfs** directory and read all content in the **testHdfs** file.

- Run the following command to access HTTP:  
linux1:/opt/client # curl -L --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs?op=OPEN"
- The running result is displayed as follows:  
Hello, webhdfs user!  
Welcome back to webhdfs!
- Run the following command to access HTTPS:  
linux1:/opt/client # curl -k -L --negotiate -u: "https://linux1:9871/webhdfs/v1/huawei/testHdfs?op=OPEN"
- The running result is displayed as follows:  
Hello, webhdfs user!  
Welcome back to webhdfs!

**Step 9** List details of all directory and file information in the **huawei** directory of the HDFS.

**LISTSTATUS** will return all child files and folders information in a single request.

- Run the following command to access HTTP:  
linux1:/opt/client # curl --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs?op=LISTSTATUS"
- The result is displayed as follows:  
{"FileStatuses":{"FileStatus":[{"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergroup","length":70,"modificationTime":1462426678379,"owner":"hdfs","pathSuffix":"","permission":"755","replication":3,"storagePolicy":0,"type":"FILE"}]}}
- Run the following command to access HTTPS.  
linux1:/opt/client # curl -k --negotiate -u: "https://linux1:9871/webhdfs/v1/huawei/testHdfs?op=LISTSTATUS"
- The result is displayed as follows:  
{"FileStatuses":{"FileStatus":[{"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergroup","length":70,"modificationTime":1462426678379,"owner":"hdfs","pathSuffix":"","permission":"755","replication":3,"storagePolicy":0,"type":"FILE"}]}}

**LISTSTATUS** along with **size** and **startafter** param will help in fetching the child files and folders information through multiple requests, thus avoiding the user interface from becoming slow when there is a large amount of child information to be fetched.

- Run the following command to access HTTP.  
linux1:/opt/client # curl --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/?op=LISTSTATUS&startafter=sparkJobHistory&size=1"
- The result is displayed as follows:  
{"FileStatuses":{"FileStatus":[{"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergroup","length":70,"modificationTime":1462426678379,"owner":"hdfs","pathSuffix":"testHdfs","permission":"755","replication":3,"storagePolicy":0,"type":"FILE"}]}}
- Run the following command to access HTTPS.  
linux1:/opt/client # curl -k --negotiate -u: "https://linux1:9871/webhdfs/v1/huawei/?op=LISTSTATUS&startafter=sparkJobHistory&size=1"
- The result is displayed as follows:  
{"FileStatuses":{"FileStatus":[{"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergroup","length":70,"modificationTime":1462426678379,"owner":"hdfs","pathSuffix":"testHdfs","permission":"755","replication":3,"storagePolicy":0,"type":"FILE"}]}}

```
3,"storagePolicy":0,"type":"FILE"}
]}]}
```

**Step 10** Delete the **testHdfs** file that is in the **/huawei/testHdfs** directory of HDFS.

- Run the following command to access HTTP:

```
linux1:/opt/client # curl -i -X DELETE --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs?op=DELETE"
```

- The running result is displayed as follows:

```
HTTP/1.1 401 Authentication required
Date: Thu, 05 May 2016 05:54:37 GMT
Pragma: no-cache
Date: Thu, 05 May 2016 05:54:37 GMT
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; HttpOnly
Content-Length: 0
HTTP/1.1 200 OK
Cache-Control: no-cache
Expires: Thu, 05 May 2016 05:54:37 GMT
Date: Thu, 05 May 2016 05:54:37 GMT
Pragma: no-cache
Expires: Thu, 05 May 2016 05:54:37 GMT
Date: Thu, 05 May 2016 05:54:37 GMT
Pragma: no-cache
Content-Type: application/json
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate YGoGCSqGS1b3EgECAglAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKiRARC9k0/v6Ed8VIUBy3kuT0b4RkqkNMCRDevsLGQOUQRORkzWI3Wu
+XLJUMKlmZaWpP+bPzpx8O2Od81mLBgdi8sOkLw
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1462463677153&s=Pwx5UIqaUlJFb9R6ZwISX85Gol="; Path=/; Expires=Thu,
05-May-2016 15:54:37 GMT; HttpOnly
Transfer-Encoding: chunked
{"boolean":true}linux1:/opt/client #
```

- Run the following command to access HTTPS:

```
linux1:/opt/client # curl -i -k -X DELETE --negotiate -u: "https://linux1:9871/webhdfs/v1/huawei/testHdfs?op=DELETE"
```

- The running result is displayed as follows:

```
HTTP/1.1 401 Authentication required
Date: Thu, 05 May 2016 06:20:10 GMT
Pragma: no-cache
Date: Thu, 05 May 2016 06:20:10 GMT
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; Secure; HttpOnly
Content-Length: 0
HTTP/1.1 200 OK
Cache-Control: no-cache
Expires: Thu, 05 May 2016 06:20:10 GMT
Date: Thu, 05 May 2016 06:20:10 GMT
Pragma: no-cache
Expires: Thu, 05 May 2016 06:20:10 GMT
Date: Thu, 05 May 2016 06:20:10 GMT
Pragma: no-cache
Content-Type: application/json
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate YGoGCSqGS1b3EgECAglAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKiRARCLY5vrVmgsiH2VWRypc30iZGffRUf4nXNaHCWni3TIDUOTl+S+hffatSbo/+uayQI/
6k9jAfaJrvFlfxqppFtofpP
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1462465210180&s=KGd2SbH/EUSaaeVKCb5zPzGBRko="; Path=/; Expires=Thu,
05-May-2016 16:20:10 GMT; Secure; HttpOnly
```

```
Transfer-Encoding: chunked
{"boolean":true}linux1:/opt/client #
```

----End

The Key Management Server (KMS) uses the HTTP REST API to provide key management services for external systems. For details about the API, see:

<http://hadoop.apache.org/docs/r3.1.1/hadoop-kms/index.html>.

 NOTE

As REST API reference has done security hardening to prevent script injection attack. Through REST API reference, it cannot create directory and file name which contain those key words "<script ", "<iframe", "<frame", "javascript:".

## 14.5.2 Shell Command Introduce

### HDFS Shell

You can use the Hadoop Distributed File System (HDFS) Shell command to perform operations on the HDFS, such as reading and writing files.

To run the HDFS Shell:

Go to the directory of HDFS client and enter the command. An example is shown as follows:

```
cd /opt/client/HDFS/hadoop/bin
```

```
hdfs dfs -mkdir /tmp/input
```

You can run the following command to seek help about HDFS commands:

```
hdfs --help
```

For details about the shell, see

<http://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-common/FileSystemShell.html>

**Table 14-14** Transparent encryption-related commands

Scenario	Operation	Command	Description
Hadoop shell command management key	Create keys.	<b><i>hadoop key create</i></b> <keyname> [-cipher <cipher>] [-size <size>] [-description <description>] [-attr <attribute=value>] [-provider <provider>] [-help]	The create subcommand creates a key for the name specified by the <keyname> argument within the provider specified by the -provider argument. You may specify a cipher with the -cipher argument. The default cipher is "AES/CTR/NoPadding" currently. The default keysize is 128. You may specify the requested key length using the -size argument. Arbitrary attribute=value style attributes may be specified using the -attr argument. The -attr may be specified for multiple times, once per attribute.
	Rollback	<b><i>hadoop key roll</i></b> <keyname> [-provider <provider>] [-help]	The roll subcommand creates a new version for the specified key within the provider indicated using the -provider argument.
	Delete keys	<b><i>hadoop key delete</i></b> <keyname> [-provider <provider>] [-f] [-help]	The delete subcommand deletes all versions of the key specified by the <keyname> argument within the provider specified by the -provider argument. The command asks for user confirmation unless -f is specified.
	View keys	<b><i>hadoop key list</i></b> [-provider <provider>] [-metadata] [-help]	The list subcommand displays the keynames contained in a particular provider as configured in core-site.xml or specified with the -provider argument. The -metadata argument displays the metadata.

**Table 14-15** Shell commands of Colocation client

Operation	Command	Description
Group creation	hdfs colocationadmin - createGroup -groupId <groupId> -locatorIds <comma separated locatorIDs> or -file <path of the file contains all of locatorIDs>	Used to create a group. In the command, groupId is the group name and locatorID is the locator name. You can enter comma-separated locator IDs using command lines. You can also write locator IDs into a file so that the system can obtain the locator IDs by reading the file.
Group deletion	hdfs colocationadmin - deleteGroup <groupId>	Used to delete the specified group.
Group query	hdfs colocationadmin - queryGroup <groupId>	Used to query details about a specified group, including locators in the group and information about each locator and its corresponding DataNode.
Viewing all groups	hdfs colocationadmin - listGroups	Used to list all groups and their creation time.
Setting ACL permissions on Colocation directories	hdfs colocationadmin - setAcl	Used to set ACL permissions on Colocation directories in ZooKeeper. The default root directory of Colocation in ZooKeeper is <b>/hadoop/colocationDetails</b> .

### 14.5.3 HDFS Access Configuration on Windows Using EIPs

#### Scenario

This section describes how to bind Elastic IP addresses (EIPs) to a cluster and configure HDFS files so that sample files can be compiled locally.

This section uses HdfsExample as an example.

## Procedure

**Step 1** Apply for an EIP for each node in the cluster and add public IP addresses and corresponding host domain names of all nodes to the Windows local **hosts** file. (If a host name contains uppercase letters, change them to lowercase letters.)

1. On the VPC console, apply for EIPs (the number of EIPs you buy should be equal to the number of nodes in the cluster), click the name of each node in the MRS cluster, and bind an EIP to each node on the **EIPs** page.

For details, see **Virtual Private Cloud > User Guide > EIP > Assigning an EIP and Binding It to an ECS**.

2. Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.

```

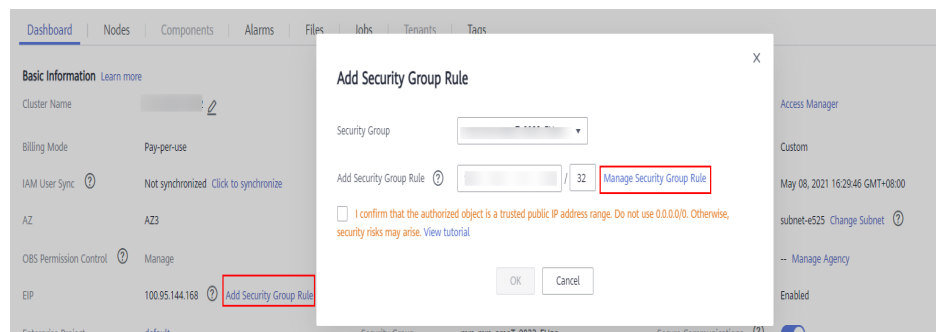
1 Mapping between public IP addresses and private IP addresses
2 100.95.144.168 172.16.0.120
3 100.95.144.168 172.16.0.42
4 100.95.144.168 172.16.0.64
5 100.95.144.168 172.16.0.200
6 100.93.144.168 172.16.0.139
7 100.93.144.168 172.16.0.214
8
9 hosts file in the cluster
10 172.16.0.120 node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
11 172.16.0.42 node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
12 172.16.0.64 node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
13 172.16.0.200 node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
14 172.16.0.139 node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
15 172.16.0.214 node-master2pVNu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVNu.ead64699-185a-4290-bbef-1a07e2f0459b.com.
16
17 hosts file that should be added to Windows
18 100.95.144.168 node-group-1xzi00002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
19 100.95.144.168 node-master3vint.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
20 100.93.144.168 node-group-1xzi00003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
21 100.95.144.168 node-master1ceip.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
22 100.93.144.168 node-group-1xzi00001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
23 100.93.144.168 node-master2pvnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVNu.ead64699-185a-4290-bbef-1a07e2f0459b.com.

```

**Step 2** Change the IP addresses in the **krb5.conf** file to the corresponding host names.

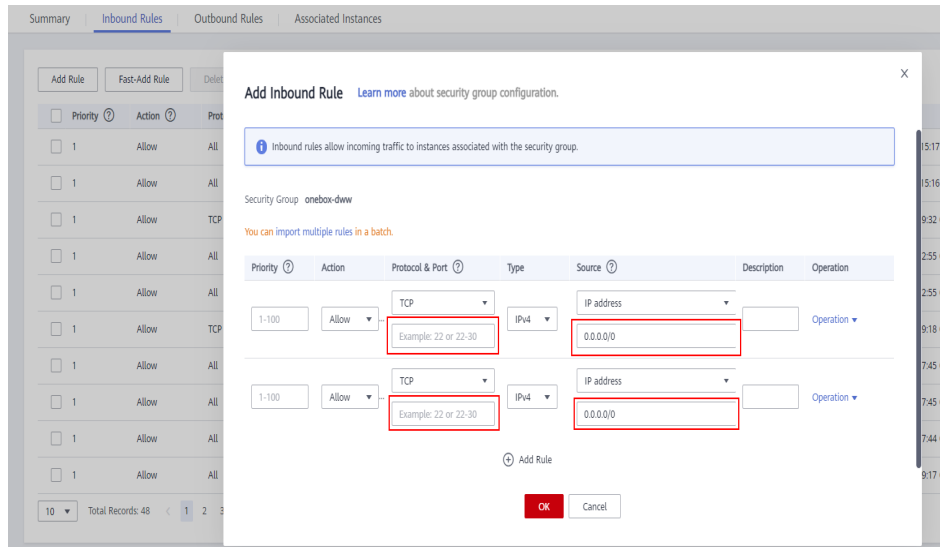
**Step 3** Configure security group rules for the cluster.

1. On the **Dashboard** page, choose **Add Security Group Rule > Manage Security Group Rule**.



2. On the **Inbound Rules** tab page, click **Add Rule**. In the **Add Inbound Rule** dialog box, configure Windows IP addresses and ports 821730TCP, 21731TCP/UDP, and 21732TCP/UDP.

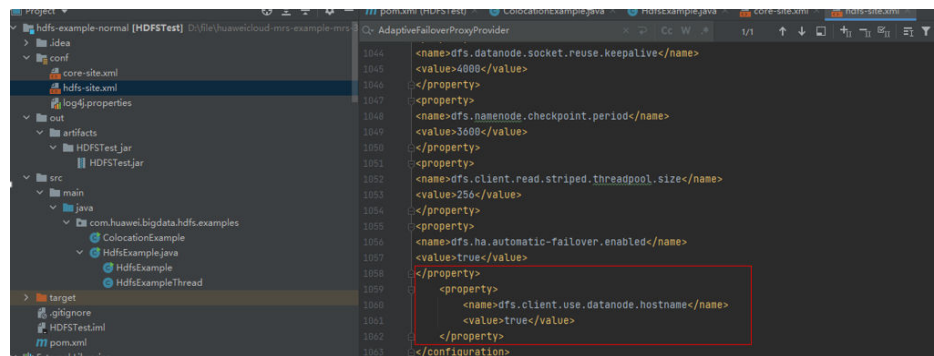




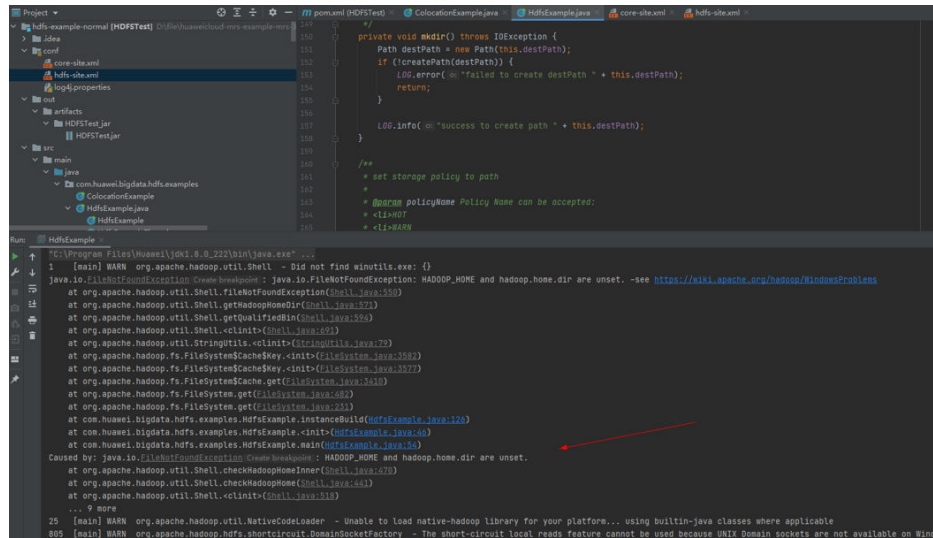
**Step 4** On Manager, choose **Cluster > Services > HDFS > More > Download Client**, copy the **core-site.xml** and **hdfs-site.xml** files on the client to the **conf** directory of the sample project, and add the following content to the **hdfs-site.xml** file.

```
<property>
<name>dfs.client.use.datanode.hostname</name>
<value>true</value>
</property>
```

(Change the DataNode communication mode to hostname.)



After the modification, an error message indicating that **hadoop\_home** does not exist may be displayed when you run the sample project. You can ignore the error because it does not affect the use.



**Step 5** Before running the sample code, change the value of **PRINCIPAL\_NAME** in the sample code to the username for security authentication.

----End

# 15 HDFS Development Guide (Normal Mode)

---

## 15.1 Overview

### 15.1.1 Introduction to HDFS

#### Introduction to HDFS

Hadoop distribute file system (HDFS) is a distributed file system with high fault tolerance. HDFS supports data access with high throughput and applies to processing of large data sets.

HDFS applies to the following application scenarios:

- Massive data processing (higher than the TB or PB level).
- Scenarios that require high throughput.
- Scenarios that require high reliability.
- Scenarios that require good scalability.

#### Introduction to HDFS Interface

HDFS can be developed by using Java language. For details of API reference, see [Java API](#).

### 15.1.2 Basic Concepts

#### DataNode

A DataNode is used to store data blocks of each file and periodically report the storage status to the NameNode.

## NameNode

A NameNode is used to manage the namespace, directory structure, and metadata information of a file system and provide the backup mechanism. NameNodes are classified into the following two types:

- **Active NameNode:** manages the file system namespace, maintains the directory structure tree and metadata information of a file system, and records the relationship between each data block and the file to which the data block belongs.
- **Standby NameNode:** Data in a standby NameNode is synchronous with those in an active NameNode. A standby NameNode takes over services from the active NameNode if the active NameNode is exception.

## JournalNode

A JournalNode synchronizes metadata between the active and standby NameNodes in the High Availability (HA) cluster.

## ZKFC

ZKFC must be deployed for each NameNode. It is responsible for monitoring NameNode status and writing status information to the ZooKeeper. ZKFC also has permission to select the active NameNode.

## Colocation

Colocation is used to store associated data or the data to be associated on the same storage node. The HDFS Colocation stores files to be associated on a same data node so that data can be obtained from the same data node during associated operations. This greatly reduces network bandwidth consumption.

## Client

The HDFS can be accessed from the Java application programming interface (API), C API, Shell, HTTP REST API and web user interface (WebUI).

For details, see [Common API Introduction](#) and [Shell Command Introduce](#).

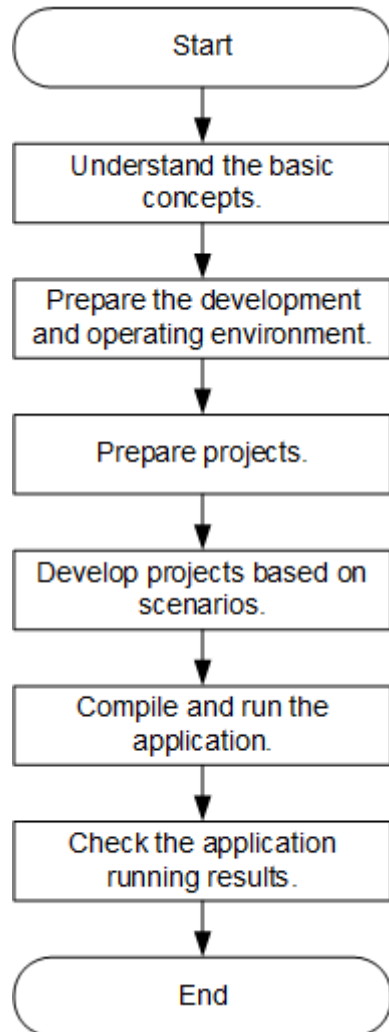
- **JAVA API**  
Provides an application interface for the HDFS. This guide describes how to use the Java API to develop HDFS applications.
- **C API**  
Provides an application interface for the HDFS. This guide describes how to use the C API to develop HDFS applications.
- **Shell**  
Provides shell commands to perform operations on the HDFS.
- **HTTP REST API**  
Additional interfaces except Shell, Java API and C API. You can use the interfaces to monitor HDFS status.
- **WEB UI**

Provides a visualized management web page.

### 15.1.3 Development Process

All stages of the development process are shown and described in [Figure 15-1](#) and [Table 15-1](#).

**Figure 15-1** HDFS development process



**Table 15-1** Description of HDFS development process

Stage	Description	Reference
Understand the basic concepts.	Before the application development, the basic concepts of HDFS are required to be understood.	<a href="#">Basic Concepts</a>

Stage	Description	Reference
Prepare the development and operating environment.	Use IntelliJ IDEA and configure the development environment based on the reference. The running environment of HDFS is the HDFS client. Install and configure the client based on the reference.	<a href="#">Development and Operating Environment</a>
Prepare projects.	HDFS provides sample projects in various scenarios. Sample projects can be imported for studying.	<a href="#">Configuring and Importing Sample Projects</a>
Develop projects based on scenarios.	Provide the sample project. This helps users to learn about the programming interfaces of all HDFS components quickly.	<a href="#">Developing the Project</a>
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	<a href="#">Commissioning the Application</a>
Check the application running results.	Application running results are stored in the directory specified by users. Users can also check the running results through the UI.	<a href="#">Commissioning the Application</a>

## 15.2 Environment Preparation

### 15.2.1 Development and Operating Environment

#### Preparing Development Environment

[Table 15-2](#) describes the environment required for application development.

**Table 15-2** Development Environment

Preparation	Description
OS	<ul style="list-style-type: none"> <li>Development environment: Windows OS. Windows 7 or later is supported.</li> <li>Operating environment: Windows OS or Linux OS.</li> </ul> <p>If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</p>

Preparation	Description
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> <li>For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> <li>Oracle JDK versions: 1.8</li> <li>Supported IBM JDK versions: 1.8.5.11</li> </ul> </li> <li>For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> <li>OpenJDK: 1.8.0_272</li> </ul> </li> </ul> <p><b>NOTE</b> The server supports only TLS V1.2 or later to ensure security. By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls">https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls</a>.</p>
IntelliJ IDEA installation and configuration	<p>It is the basic configuration for the development environment. IntelliJ IDEA 2019.1 or other compatible versions are used.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li> <li>If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li> <li>If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li> <li>Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li> </ul>
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
7-zip	Used to decompress <b>.zip</b> and <b>.rar</b> packages. 7-Zip 16.04 is supported.

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host;

obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.

- a. [Log in to the FusionInsight Manager portal](#) and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is

**FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\HDFS\config** and manually import the configuration file to the configuration file directory (usually the **conf** folder) of the HDFS sample project.
- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

#### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client on the node. For example, the client installation directory is **/opt/client**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. [Log in to the FusionInsight Manager portal](#). Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/HDFS/config** directory and save them to the **conf** folder of the project code.

For example, if the client software package is

**FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path



is `/tmp/FusionInsight-Client` on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp HDFS/config/* root@IP address of the client node:/opt/client/conf
```

**Table 15-3** Configuration files

File	Function
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.

- c. Check the network connection of the client node.  
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the `/etc/hosts` file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

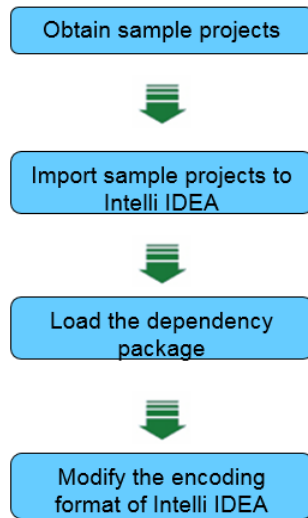
## 15.2.2 Configuring and Importing Sample Projects

### Scenario

HDFS provides sample projects for multiple scenarios to help you quickly learn HDFS projects.

The procedure for importing HDFS example codes is described as follows: [Figure 15-2](#) shows the procedure.

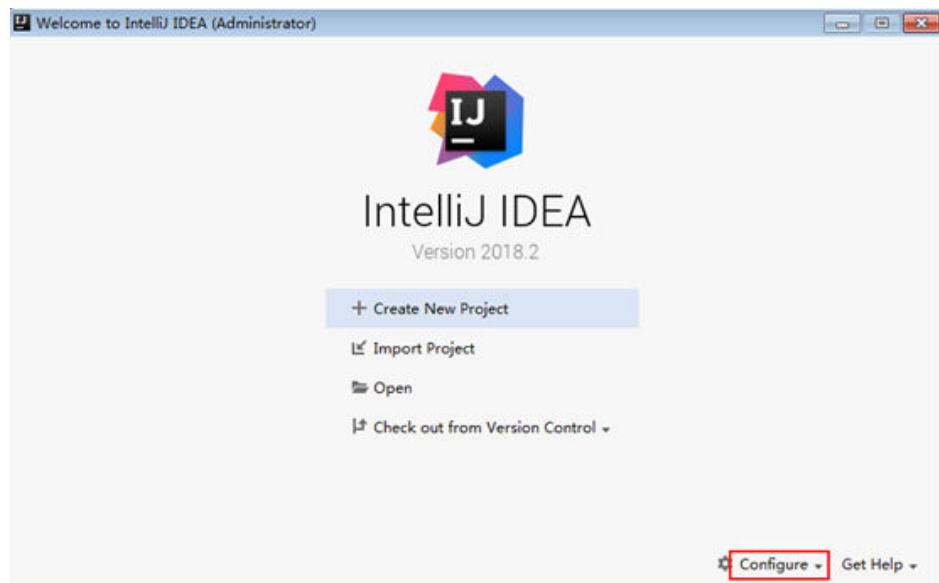
**Figure 15-2** Sample project importing procedure



## Procedure

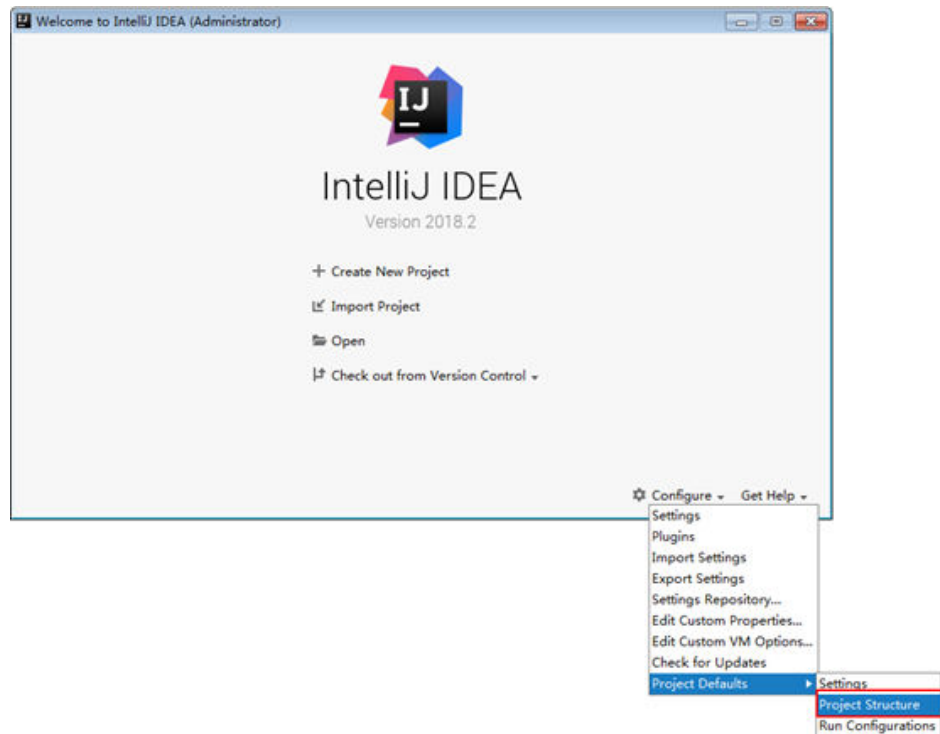
- Step 1** Obtain the sample project folder **hdfs-example-normal** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Import the example project to the IntelliJ IDEA development environment.
1. Open IntelliJ IDEA and choose **File > Open**.
  2. Choose the directory of the example project **hdfs-example-normal**. Click **OK**.
- Step 3** After installing the IntelliJ IDEA and JDK tools, configure JDK in IntelliJ IDEA.
1. Open IntelliJ IDEA and click **Configure**.

**Figure 15-3** Quick Start



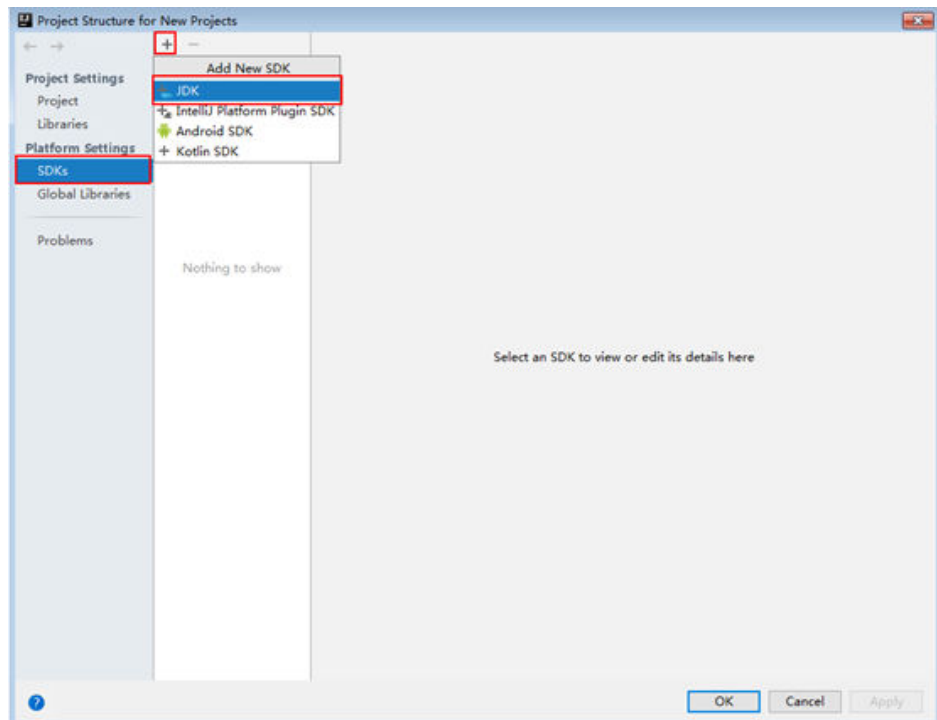
2. Choose **Project Defaults > Project Structure**.

**Figure 15-4** Configure



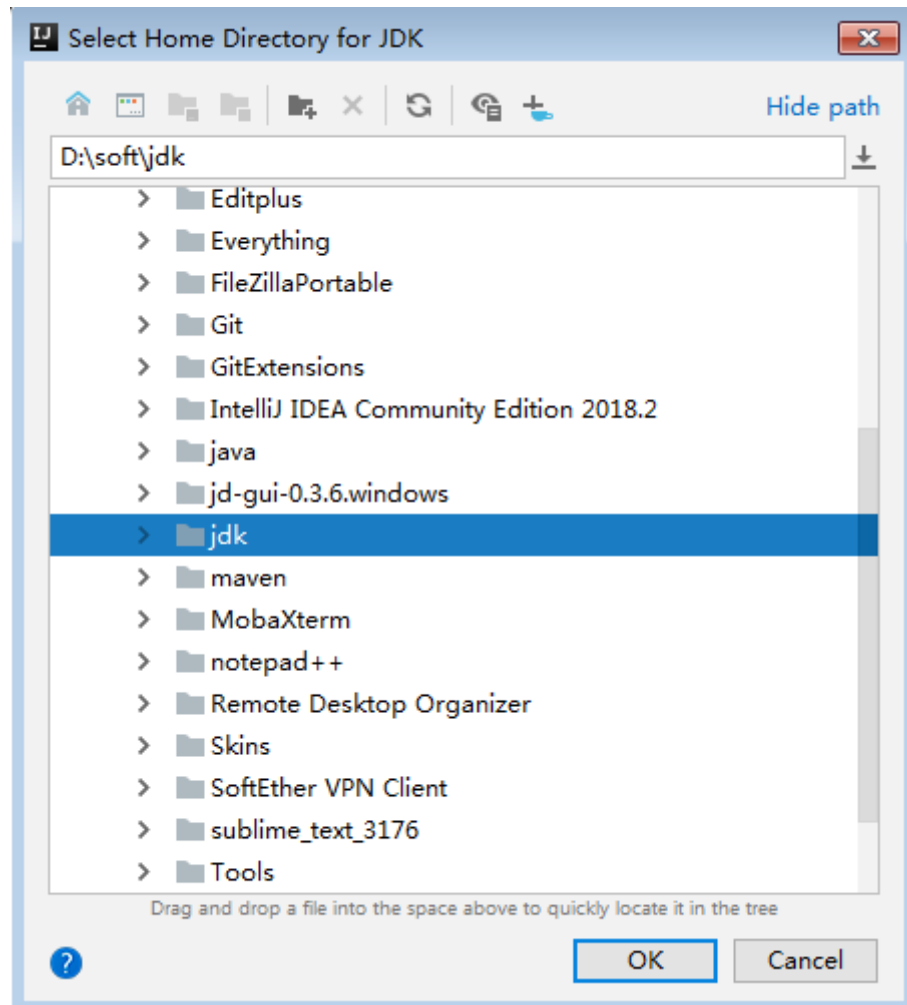
3. In the displayed **Project Structure for New Projects** interface, click **SDKs**. Then click the plus sign (+) and choose **JDK**.

**Figure 15-5** Project Structure for New Projects



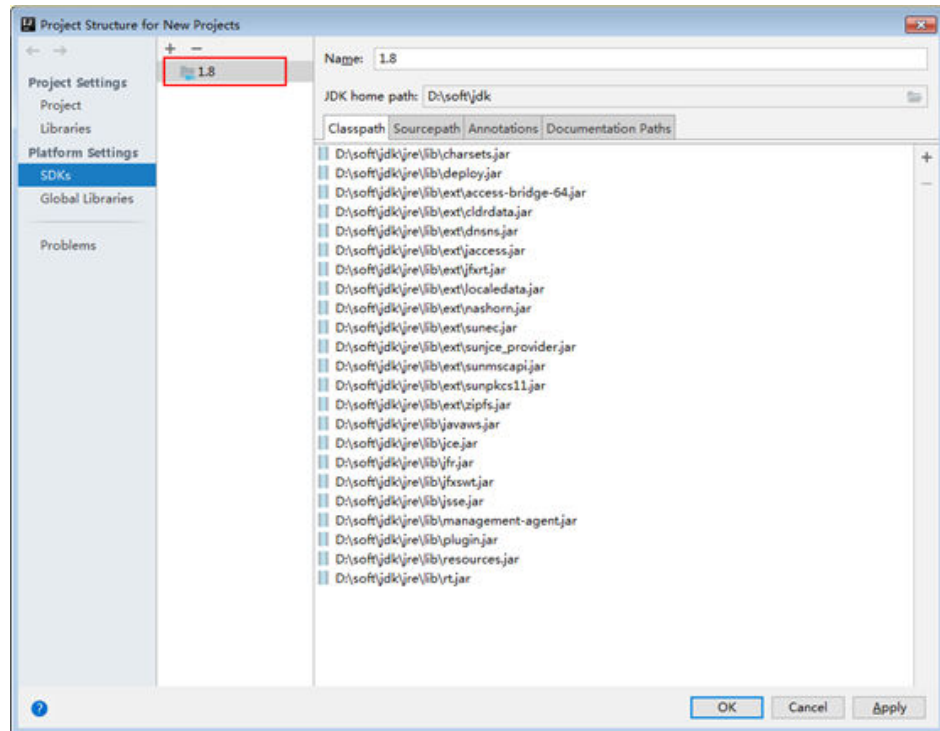
4. In the displayed **Select Home Directory for JDK** interface, select the JDK directory, and click **OK**.

**Figure 15-6** Select Home Directory for JDK



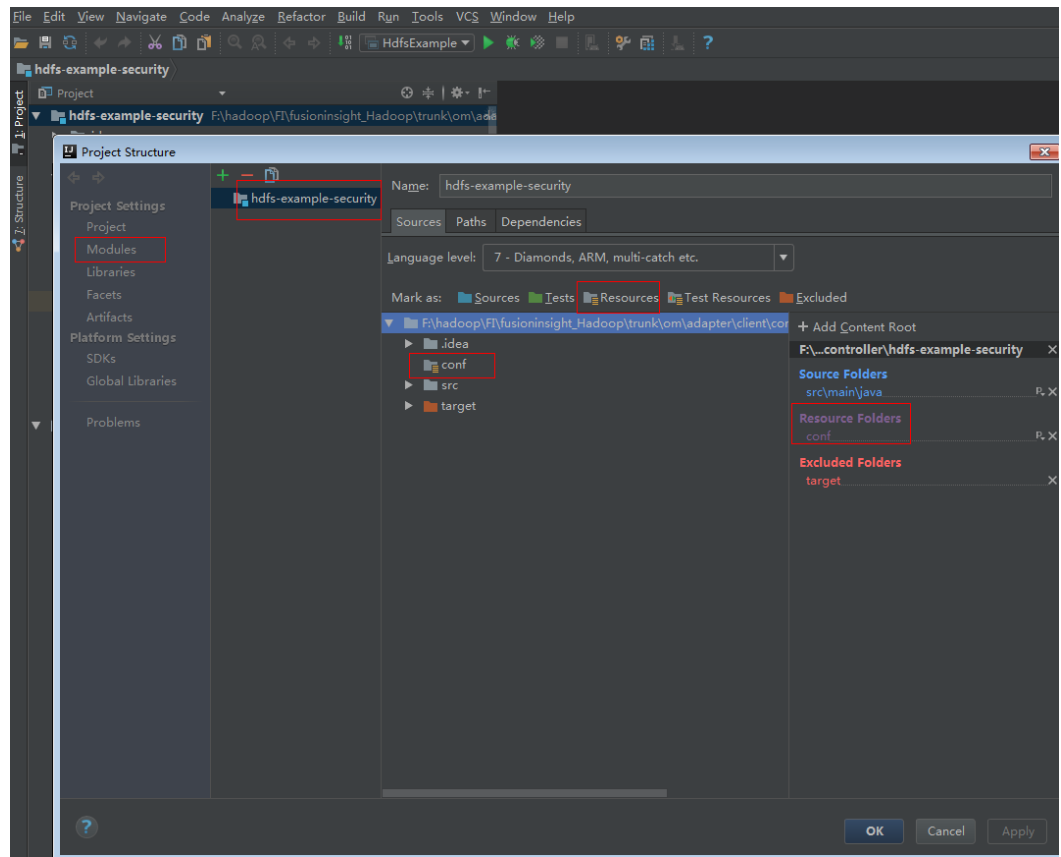
5. Click **OK**.

**Figure 15-7** Completing the JDK configuration



**Step 4** Add the conf directory in the project to the resource path. On the menu bar of the IntelliJ IDEA, choose **File > Project Structure**. In the dialog box that is displayed, select the current project and click **Resources > conf > OK** to set the resource directory. The figure shows the directory for setting the project resource.

**Figure 15-8** Setting the Project Resource Directory



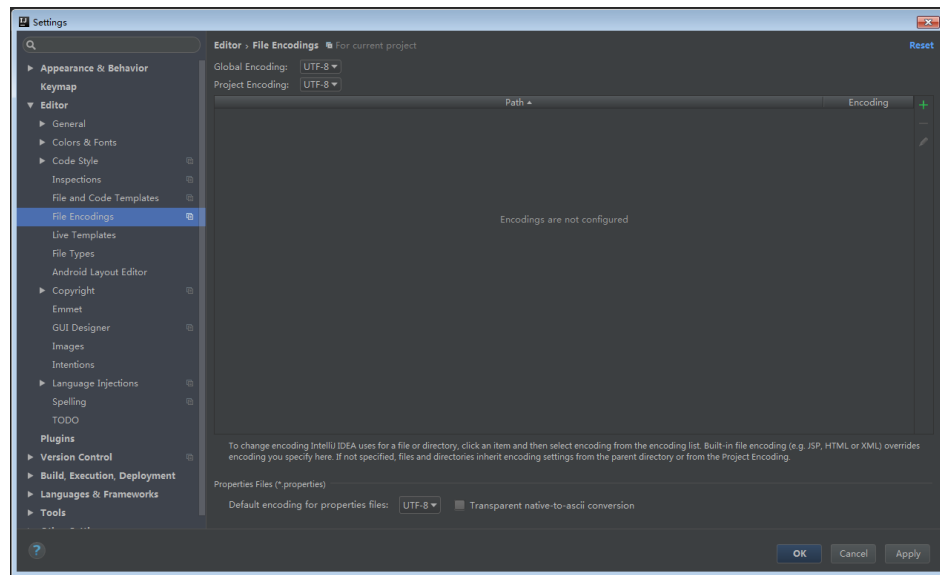
**Step 5** Add the related jar files the example project depending on into classpath.

If the sample project code is obtained from an open-source image site, dependency JAR files are automatically downloaded after Maven is configured (for details, see [Configuring Huawei Open-Source Mirrors](#)).

**Step 6** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.
2. Choose **Editor > File Encodings** from the navigation tree of the **Settings** window. In the "Global Encoding" and "Project Encodings" area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 15-9](#)

**Figure 15-9** Setting the IntelliJ IDEA coding format



----End

## 15.3 Developing the Project

### 15.3.1 Scenario

#### Typical Scenario Description

Based on typical scenarios, users can quickly learn and master the HDFS development process and know important interface functions.

#### Scenario

Service operation objects of HDFS are files. File operations in example codes include creating a folder, writing data into a file, appending data to a file, reading data from a file, and deleting a file or folder. HDFS also supports other services including setting file permission. You can learn how to perform other operations on HDFS after learning the example codes in this chapter.

The example codes are described in the following order:

1. Initializing the HDFS.
2. Creating directories.
3. Writing data into a file.
4. Appending data to a file.
5. Reading data from a file.
6. Deleting a file.
7. Deleting directories.
8. Multi-thread tasks



9. Setting storage policies.
10. Colocation.

## 15.3.2 Development Idea

### Development Idea

According to the previous scenario description, the following provides the basic operations for HDFS files with read, write, and delete operations on the `/user/hdfs-examples/test.txt` file as an example:

1. Create a FileSystem object: `fSystem`.
2. Call the `mkdir` interface in `fSystem` to create a directory.
3. Call the `create` interface in `fSystem` to create an `FSDataOutputStream` object: `out`. Use the `write` method to write data into the object `out`.
4. Call the `append` interface in `fSystem` to create an `FSDataOutputStream` object: `out`. Use the `write` method to append data into the object `out`.
5. Call the `open` interface in `fSystem` to create an `FSDataInputStream` object: `in`. Use the `read` method to read files of the object `in`.
6. Call the `delete` interface in `fSystem` to delete the file.
7. Call the `delete` interface in `fSystem` to delete the folder.

## 15.3.3 Declare the Example Codes

### 15.3.3.1 Initializing the HDFS

#### Function

Hadoop distributed file system (HDFS) initialization is a prerequisite for using application programming interfaces (APIs) provided by the HDFS. The process of initializing the HDFS is

1. Load the HDFS service configuration file.
2. Instantiate a FileSystem.

#### Configuration File Description

**Table 15-4** lists the configuration files to be used during the login to the HDFS. These files already imported to the `conf` directory of the `hadoop-examples` project.

**Table 15-4** Configuration files

File	Function	Path
core-site.xml	Configures HDFS parameters.	FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\config\core-site.xml

File	Function	Path
hdfs-site.xml	Configures HDFS parameters.	FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\config\hdfs-site.xml

 NOTE

The **log4j.properties** file under the **conf** directory can be configured based on your needs.

## Example Codes

The following is code snippets. For complete codes, see the HdfsExample class in com.huawei.bigdata.hdfs.examples.

The initialization codes used when applications are run in Linux and the codes used when applications are run in Windows are the same. The example codes are as follows:

```
//initialization
/**confLoad();

// Creating a sample project
HdfsExample hdfs_examples = new HdfsExample("/user/hdfs-examples", "test.txt");
/**
 *
 * If the application is running in the Linux OS, the path of core-site.xml, hdfs-site.xml must be modified to
the absolute path of the client file in the Linux OS.
 *
 */
private static void confLoad() throws IOException {
 conf = new Configuration();
 // conf file
 conf.addResource(new Path(PATH_TO_HDFS_SITE_XML));
 conf.addResource(new Path(PATH_TO_CORE_SITE_XML));
 // conf.addResource(new Path(PATH_TO_SMALL_SITE_XML));
}

/**
 *Create a sample project.
 */
public HdfsExample(String path, String fileName) throws IOException {
 this.DEST_PATH = path;
 this.FILE_NAME = fileName;
 instanceBuild();
}
private void instanceBuild() throws IOException {
 fSystem = FileSystem.get(conf);
}
```

 NOTE

- (Optional) Specify a user to run the example code. To run the example code related to the Colocation operation, the user must be a member of the supergroup group. The following describes two ways to specify the user who runs the example code:

Add the environment variable **HADOOP\_USER\_NAME**: For operation details in a Windows-based environment, see [Step 1](#) in section **Compiling and Running an Application** For operation details in a Linux-based environment, see [Step 4](#) in section **Compiling and Running an Application with the Client Installed** or [Step 4](#).in section **Compiling and Running an Application with the Client Not Installed**.

Modify the code: If **HADOOP\_USER\_NAME** is not specified, modify "USER" in the code to the actual user name:

```
System.setProperty("HADOOP_USER_NAME", USER);
```

### 15.3.3.2 Creating Directories

#### Function

Process of creating a directory:

1. Call the exists method of the FileSystem instance to check whether the directory exists.
2. If yes, the method stops.
3. If no, call the mkdirs method in the FileSystem instance to create a directory.

#### Example Codes

The following is a code snippet. For complete codes, see the **HdfsExample** class in **com.huawei.bigdata.hdfs.examples**.

```
/**
 * Create a directory.
 *
 * @throws java.io.IOException
 */
private void mkdir() throws IOException {
 Path destPath = new Path(DEST_PATH);
 if (!createPath(destPath)) {
 LOG.error("failed to create destPath " + DEST_PATH);
 return;
 }

 LOG.info("success to create path " + DEST_PATH);
}

/**
 * create file path
 *
 * @param filePath
 * @return
 * @throws java.io.IOException
 */
private boolean createPath(final Path filePath) throws IOException {
 if (!System.exists(filePath)) {
 fSystem.mkdirs(filePath);
 }
 return true;
}
```

### 15.3.3.3 Writing Data into a File

#### Function

The process of writing data into a file is

1. Use the create method in the FileSystem instance to obtain the output stream of writing files.
2. Uses this data stream to write content into a specified file in the HDFS.

#### NOTE

Close all requested resources after writing files.

#### Example Codes

The following is code snippets. For complete codes, see HdfsMain and HdfsWriter classes in **com.huawei.bigdata.hdfs.examples**.

```
/**
 * Write a file, write the data
 *
 * @throws IOException
 * @throws ParameterException
 */
private void write() throws IOException {
 final String content = "hi, I am bigdata. It is successful if you can see me.";
 FSDataOutputStream out = null;
 try {
 out = fSystem.create(new Path(DEST_PATH + File.separator + FILE_NAME));
 out.write(content.getBytes());
 out.hsync();
 LOG.info("success to write.");
 } finally {
 // make sure the stream is closed finally.
 IOUtils.closeStream(out);
 }
}
```

### 15.3.3.4 Appending Data to a File

#### Function

Append data to a specified file in the Hadoop distributed file system (HDFS). The process of appending data to a file is

1. Use the append method in the FileSystem instance to obtain the output stream of the appended data.
2. Use the output stream to add the content to be appended behind the specified file in the HDFS.

#### NOTE

Close all requested resources after appending data.

#### Example Codes

The following is code snippets. For complete codes, see HdfsMain and HdfsWriter classes in **com.huawei.bigdata.hdfs.examples**.

```
/**
 * Append data to a file.
 *
 * @throws IOException
 */
private void append() throws IOException {
 final String content = "I append this content.";
 FSDataOutputStream out = null;
 try {
 out = fSystem.append(new Path(DEST_PATH + File.separator + FILE_NAME));
 out.write(content.getBytes());
 out.hsync();
 LOG.info("success to append.");
 } finally {
 // make sure the stream is closed finally.
 IOUtils.closeStream(out);
 }
}
```

### 15.3.3.5 Reading Data from a File

#### Function

Read data from a specified file in the Hadoop distributed file system (HDFS). The process is:

1. Use the open method in the FileSystem instance to obtain the input stream of writing files.
2. Use the input stream to read the content of the specified file in the HDFS.

#### NOTE

Close all requested resources after reading files.

#### Example Codes

The following is code snippets. For complete codes, see the HdfsMain class in **com.huawei.bigdata.hdfs.examples**.

```
/**
 * Read a file.
 *
 * @throws IOException
 */
private void read() throws IOException {
 String strPath = DEST_PATH + File.separator + FILE_NAME;
 Path path = new Path(strPath);
 FSDataInputStream in = null;
 BufferedReader reader = null;
 StringBuffer strBuffer = new StringBuffer();
 try {
 in = fSystem.open(path);
 reader = new BufferedReader(new InputStreamReader(in));
 String sTempOneLine;
 // write file
 while ((sTempOneLine = reader.readLine()) != null) {
 strBuffer.append(sTempOneLine);
 }
 LOG.info("result is : " + strBuffer.toString());
 LOG.info("success to read.");
 } finally {
 // make sure the streams are closed finally.
 IOUtils.closeStream(reader);
 IOUtils.closeStream(in);
 }
}
```

```
}
}
```

### 15.3.3.6 Deleting a File

#### Function

Delete a file from the Hadoop distributed file system (HDFS).

#### NOTE

Exercise caution when you delete files because the deletion is irreversible.

#### Example Codes

The following is code snippets. For complete codes, see the HdfsMain class in **com.huawei.bigdata.hdfs.examples**.

```
/**
 * Delete a file.
 *
 * @throws IOException
 */
private void delete() throws IOException {
 Path beDeletedPath = new Path(DEST_PATH + File.separator + FILE_NAME);
 if (fSystem.delete(beDeletedPath, true)) {
 LOG.info("success to delete the file " + DEST_PATH + File.separator + FILE_NAME);
 } else {
 LOG.warn("failed to delete the file " + DEST_PATH + File.separator + FILE_NAME);
 }
}
```

### 15.3.3.7 Deleting Directories

#### Function

Delete a specified directory from the HDFS.

#### NOTE

Files in the deleted directory will be stored in the **Trash/Current** folder of the directory of the current user. In the event of mis-deletion, you can restore the folder from the directory.

#### Example Codes

The following is a code snippet of file deletion. For complete codes, see the **HdfsExample** class in **com.huawei.bigdata.hdfs.examples**.

```
/**
 * delete the directory
 *
 * @throws java.io.IOException
 */
private void rmdir() throws IOException {
 Path destPath = new Path(DEST_PATH);
 if (!deletePath(destPath)) {
 LOG.error("failed to delete destPath " + DEST_PATH);
 return;
 }
 LOG.info("success to delete path " + DEST_PATH);
}
```

```
}

/**
 * delete file path
 *
 * @param filePath
 * @return
 * @throws java.io.IOException
 */
private boolean deletePath(final Path filePath) throws IOException {
 if (!FileSystem.exists(filePath)) {
 return false;
 }
 // fSystem.delete(filePath, true);
 return fSystem.delete(filePath, true);
}
```

### 15.3.3.8 Multi-Thread Tasks

#### Function

Create multi-threaded tasks and initiate multiple instances to perform file operations.

#### Example Codes

The following is a code snippet of file deletion. For complete codes, see the **HdfsExample** class in **com.huawei.bigdata.hdfs.examples**.

```
// Service example 2: multi-thread tasks
final int THREAD_COUNT = 2;
for (int threadNum = 0; threadNum < THREAD_COUNT; threadNum++) {
 HdfsExampleThread example_thread = new HdfsExampleThread("hdfs_example_" + threadNum);
 example_thread.start();
}

class HdfsExampleThread extends Thread {
 private final static Log LOG = LogFactory.getLog(HdfsExampleThread.class.getName());
 /**
 *
 * @param threadName
 */
 public HdfsExampleThread(String threadName) {
 super(threadName);
 }
 public void run() {
 HdfsExample example;
 try {
 example = new HdfsExample("/user/hdfs-examples/" + getName(), "test.txt");
 example.test();
 } catch (IOException e) {
 LOG.error(e);
 }
 }
}
```

The **example.test()** method is the operation on files. The code is as follows:

```
/**
 * HDFS operation instance
 *
 * @throws IOException
 * @throws ParameterException
 *
 * @throws Exception
```

```
*
*/
public void test() throws IOException {
 // Create a directory.
 mkdir();

 // Write data into a file.
 write();

 // Append data to a file.
 append();

 // Read a file.
 read();

 // Delete a file.
 delete();

 // Delete a directory.
 rmdir();
}
```

### 15.3.3.9 Setting Storage Policies

#### Function

Specify storage policies for a file or folder in the HDFS.

#### Example Code

1. [Log in to the FusionInsight Manager portal](#), choose **Cluster** > *Name of the desired cluster* > **Services** > **HDFS** > **Configurations** > **All Configurations**.
2. Check whether the value of **dfs.storage.policy.enabled** is the default value **true**. If not, modify the value to **true**, click **Save**, and restart HDFS.
3. Check the code.

The following code segment is only an example. For details, see the HdfsMain class in com.huawei.bigdata.hdfs.examples.

```
/**
 * set storage policy to path
 * @param policyName
 * Policy Name can be accepted:
 * HOT
 * WARM
 * COLD
 * LAZY_PERSIST
 * ALL_SSD
 * ONE_SSD
 * @throws IOException
 */

private void setStoragePolicy(String policyName) throws IOException {
 if (fSystem instanceof DistributedFileSystem) {
 DistributedFileSystem dfs = (DistributedFileSystem) fSystem;
 Path destPath = new Path(DEST_PATH);
 Boolean flag = false;
 mkdir();
 BlockStoragePolicySpi[] storage = dfs.getStoragePolicies();
 for (BlockStoragePolicySpi bs : storage) {
 if (bs.getName().equals(policyName)) {
 flag = true;
 }
 }
 LOG.info("StoragePolicy:" + bs.getName());
 }
}
```



```
 if (!flag) {
 policyName = storage[0].getName();
 }
 dfs.setStoragePolicy(destPath, policyName);
 LOG.info("success to set Storage Policy path " + DEST_PATH);
 rmdir();
} else {
 LOG.info("SmallFile not support to set Storage Policy !!!");
}
}
```

### 15.3.3.10 Colocation

#### Function Description

Colocation is to store associated data or data on which associated operations are performed on the same storage node. The HDFS Colocation feature stores files on which associated operations are performed on the same data node so that data can be obtained from the same data node during associated operations. This greatly reduces network bandwidth consumption.

Before using the Colocation function, you are advised to be familiar with the internal mechanisms of Colocation, including:

#### Colocation node allocation principle

#### Capacity expansion and Colocation allocation

#### Colocation and data node capacity

- **Colocation node allocation principle**

Colocation allocates data nodes to locators evenly according to the allocation node quantity.

#### NOTE

The allocation algorithm principle is as follows: Colocation queries all locators, reads the data nodes allocated to the locators, and records the number of times. Based on the number of times, Colocation sorts the data nodes. The data nodes that are rarely used are placed at the beginning and selected first. The count increase by 1 each time after a node is selected. The nodes are sorted again, and the subsequent node will be selected.

- **Capacity expansion and Colocation allocation**

After cluster capacity expansion, you can select one of the following two policies shown in [Table 15-5](#) to balance the usage of data nodes and ensure that the allocation frequency of the newly added nodes is consistent with that of the old data nodes.

**Table 15-5** Allocation policies

No.	Policy	Description
1	Delete the original locators and create new locators for all data nodes in the cluster.	<ol style="list-style-type: none"> <li>1. The original locator before the capacity expansion evenly uses all data nodes. After the capacity expansion, the newly added nodes are not allocated to existing locators, so Colocation stores data only to the old data nodes.</li> <li>2. Data nodes are allocated to specific locators. Therefore, after capacity expansion, Colocation needs to reallocated data nodes to locators.</li> </ol>
2	Create new locators and plan the data storage mode again.	The old locators use the old data nodes, while the newly created locators mainly use the new data nodes. Therefore, locators need to be planned again based on the actual service requirements on data.

 **NOTE**

Generally, you are advised to use the policy to reallocate data nodes to locators after capacity expansion to prevent data from being stored only to the new data nodes.

- **Colocation and data node capacity**

When Colocation is used to store data, the data is stored to the data node of a specified locator. If no locator planning is performed, the data node capacity will be uneven. [Table 15-6](#) summarizes the two usage principles to ensure even data node capacity.

**Table 15-6** Usage Principle

No.	Usage Principle	Description
1	All the data nodes are used in the same frequency in locators.	Assume that there are N data nodes, the number of locators must an integral multiple of N (N, 2 N, ...).
2	A proper data storage plan must be made for all locators to ensure that data is evenly stored in the locators.	None

During HDFS secondary development, you can obtain the DFSColocationAdmin and DFSColocationClient instances to create groups, delete groups, write files, and delete files in or from the location.

 NOTE

- When the Colocation function is enabled and users specify DataNodes, the data volume will be large on some nodes. Serious data skew will result in HDFS data write failures.
- Because of data skew, MapReduce accesses only several nodes. In this case, the load is heavy on these nodes, while other nodes are idle.
- For a single application task, the DFSColocationAdmin and DFSColocationClient instances can be used only once. If the instances are used for many time, excessive HDFS links will be created and use up HDFS resources.
- If you need to perform the balance operation for a file uploaded by colocation, you can set the `oi.dfs.colocation.file.pattern` parameter on FusionInsight Manager to the file path to avoid invalid colocation. If there are multiple files, use commas (,) to separate the file paths, for example, `/test1,/test2`.
- Colocation stores associated data or data on which associated operations are performed on the same storage. When Balancer- or Mover-related operations are performed, data blocks will be moved. As a result, the Colocation function becomes unavailable. Therefore, when using the Colocation function, you are advised to set the HDFS configuration item **`dfs.datanode.block-pinning.enabled`** to **`true`**. In this case, files written by Colocation will not be moved when Balancer- or Mover-related operations are performed in the cluster, ensuring file colocation.

## Example Codes

For complete example codes, see `com.huawei.bigdata.hdfs.examples.ColocationExample`.

 NOTE

- Specify a user to run a Colocation project. This user must be a member of the supergroup group.
- When the Colocation project is run, the HDFS parameter `fs.defaultFS` cannot be set to `viewfs://ClusterX`.

### 1. Initialization

The user to run the Colocation project must be specified before the running.

```
private static void init() throws IOException {
 // //Set the user. If HADOOP_USER_NAME is not set for the use, use USER.
 if (System.getenv("HADOOP_USER_NAME") == null &&
 System.getProperty("HADOOP_USER_NAME") == null) {
 System.setProperty("HADOOP_USER_NAME", USER);
 }
}
```

### 2. Obtain instances.

Example: Colocation operations require the DFSColocationAdmin and DFSColocationClient instances. Therefore, the instances must be obtained before operations, such as creating a group.

```
dfsAdmin = new DFSColocationAdmin(conf);
dfs = new DFSColocationClient();
dfs.initialize(URI.create(conf.get("fs.defaultFS")), conf);
```

### 3. Create a group.

Example: Create a `gid01` group, which contains three locators.

```
/**
 * create group
 *
 * @throws java.io.IOException
 */
private static void createGroup() throws IOException {
```

```
dfsAdmin.createColocationGroup(COLOCATION_GROUP_GROUP01,
 Arrays.asList(new String[] { "lid01", "lid02", "lid03" }));
}
```

4. Write data into a file. The related group must be created before writing data into the file.

Example: Write data into the testfile.txt file.

```
/**
 * create and write file
 *
 * @throws java.io.IOException
 */
private static void put() throws IOException {
 FSDataOutputStream out = dfs.create(new Path(TESTFILE_TXT), true,
 COLOCATION_GROUP_GROUP01, "lid01");
 // the data to be written to the hdfs.
 byte[] readBuf = "Hello World".getBytes("UTF-8");
 out.write(readBuf, 0, readBuf.length);
 out.close();
}
```

5. Delete a file.

Example: Delete the testfile.txt file.

```
/**
 * delete file
 *
 * @throws java.io.IOException
 */
@SuppressWarnings("deprecation")
private static void delete() throws IOException {
 dfs.delete(new Path(TESTFILE_TXT));
}
```

6. Delete a group.

Example: Delete gid01.

```
/**
 * delete group
 *
 * @throws java.io.IOException
 */
private static void deleteGroup() throws IOException {
 dfsAdmin.deleteColocationGroup(COLOCATION_GROUP_GROUP01);
}
```

## 15.4 Commissioning the Application

### 15.4.1 Commissioning an Application in the Windows Environment

#### 15.4.1.1 Compiling and Running an Application

##### Scenario

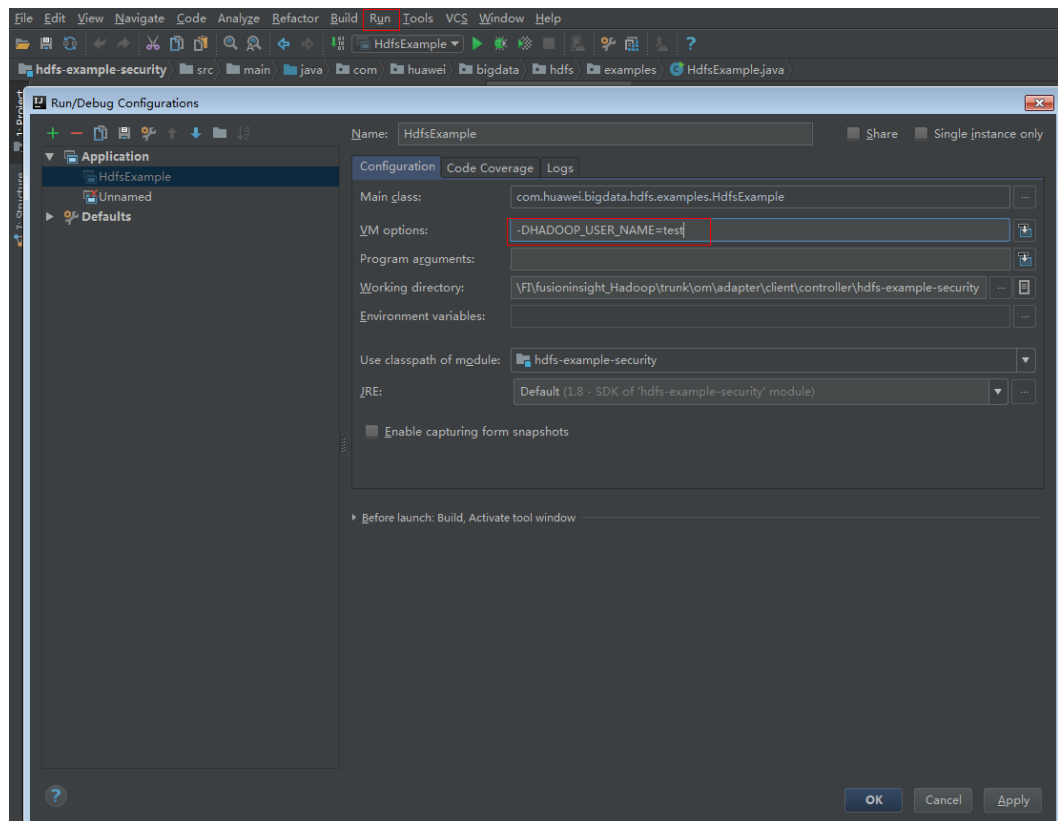
After the code development is complete, you can run an application in the Windows development environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

## Procedure

**Step 1** (Optional) In a development environment (for example, IntelliJ IDEA), a user must be specified to run the example code. There are two ways to specify the user:

Select the sample program `HdfsExample.java` or `ColocationExample.java` to be run, right-click the project, and choose `Run Configurations` from the shortcut menu. In the dialog box that is displayed, select `JavaApplication > HdfsExample` to set the running parameters. On the menu bar of the IntelliJ IDEA, choose `Run > Edit Configurations`. In the dialog box that is displayed, set the running user.

```
-DHADOOP_USER_NAME=test
```



### NOTE

The **test** user here is an example. To run the example code related to the Colocation operation, the user must be a member of the supergroup group.

**Step 2** If the environment variable is configured according to [Step 1](#) click **Run** to run the application. If not, choose the following two projects separately and run the projects:

- Choose `HdfsExample.java`, right-click the project and choose **Run 'HdfsExample.main()'** from the shortcut menu to run the project.
- Choose `ColocationExample.java`, right-click the project and choose **Run 'ColocationExample.main()'** from the shortcut menu to run the project.

 NOTE

- It is forbidden to restart HDFS service while HDFS application is in running status, otherwise the application will fail.
- When the Colocation project is run, the HDFS parameter `fs.defaultFS` cannot be set to `viewfs://ClusterX`.

----End

## 15.4.1.2 Checking the Commissioning Result

### Scenario

After an HDFS application is run, you can learn the application running conditions by viewing the running result or HDFS logs.

### Procedure

- **Learn the application running conditions by viewing the running result.**
  - The running result of the HDFS windows example application is shown as follows:

```
1654 [main] WARN org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory - The short-
circuit local reads feature cannot be used because UNIX Domain sockets are not available on
Windows.
2013 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to create path /
user/hdfs-examples
2137 [main] WARN org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
2590 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to write.
3245 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to append.
4447 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I am
bigdata. It is successful if you can see me.I append this content.
4447 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to read.
4509 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete the file /
user/hdfs-examples/test.txt
4618 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete path /
user/hdfs-examples
4743 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
create path /user/hdfs-examples/hdfs_example_1
4743 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
create path /user/hdfs-examples/hdfs_example_0
5087 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
write.
5087 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
write.
6507 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
append.
6553 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
append.
7505 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I
am bigdata. It is successful if you can see me.I append this content.
7505 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
read.
7568 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
delete the file /user/hdfs-examples/hdfs_example_1/test.txt
7583 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I
am bigdata. It is successful if you can see me.I append this content.
7583 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
read.
7630 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
delete the file /user/hdfs-examples/hdfs_example_0/test.txt
7677 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
delete path /user/hdfs-examples/hdfs_example_1
```

```
7739 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete path /user/hdfs-examples/hdfs_example_0
```

#### NOTE

In the Windows environment, the following exception occurs but does not affect services.

```
java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.
```

- The running result of the Colocation windows example application is shown as follows:

```
1623 [main] WARN org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory - The short-circuit local reads feature cannot be used because UNIX Domain sockets are not available on Windows.
1670 [main] INFO org.apache.zookeeper.ZooKeeper - Client environment:zookeeper.version=***, built on 10/19/2017 04:21 GMT
1670 [main] INFO org.apache.zookeeper.ZooKeeper - Client environment:host.name=siay7user1.china.huawei.com
1670 [main] INFO org.apache.zookeeper.ZooKeeper - Client environment:java.version=***
1670 [main] INFO org.apache.zookeeper.ZooKeeper - Client environment:java.vendor=Oracle Corporation
1670 [main] INFO org.apache.zookeeper.ZooKeeper - Client environment:java.home=D:\Program Files\Java\jre1.8.0_131
.....
Create Group has finished.
Put file is running...
5930 [main] WARN org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Put file has finished.
Delete file is running...
Delete file has finished.
Delete Group is running...
Delete Group has finished.
6866 [main] INFO org.apache.zookeeper.ZooKeeper - Session: 0x13000074b7e464b7 closed
6866 [main-EventThread] INFO org.apache.zookeeper.ClientCnxn - EventThread shut down for session: 0x13000074b7e464b7
6928 [main-EventThread] INFO org.apache.zookeeper.ClientCnxn - EventThread shut down for session: 0x14000073f13b657b
6928 [main] INFO org.apache.zookeeper.ZooKeeper - Session: 0x14000073f13b657b closed
```

- **Learn the application running conditions by viewing HDFS logs.**  
The NameNode logs of HDFS offer immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 15.4.2 Commissioning an Application in the Linux Environment

### 15.4.2.1 Compiling and Running an Application with the Client Installed

#### Scenario

The Hadoop distributed file system (HDFS) application can run in the Linux operating system (OS) with the HDFS client installed. After the application code has been developed, you can upload the jar packages to the prepared Linux client operating environment and run the application.

#### Prerequisite

- The HDFS client has been installed.

- When the host where the Linux OS runs is not a node of the cluster, you are required to set the mapping between the host name and IP address in the **hosts** file of the node where the client is installed. The host name must be correctly mapped to the IP address.

## Procedure

- Step 1** Go to the local root directory of the sample project, copy the required configuration file to the conf folder of the local project, and run the following command in Windows CLI to compress the package:

```
mvn -s "{maven_setting_path}" clean package
```

### NOTE

- In the preceding command, {maven\_setting\_path} is the path of the **settings.xml** file of the local Maven.
- After the package is successfully packed, obtain the JAR package, for example, *HDFSTest-XXX.jar*, from the **target** subdirectory in the root directory of the project. The name of the JAR package varies according to the actual package.

- Step 2** Upload the exported jarpackages to any directory in the running environment of the client, for example, **/opt/client**.

- Step 3** Configure the environment variables:

```
cd /opt/client
```

```
source bigdata_env
```

- Step 4** Specify a user to run the example. There are two ways to specify the user: add the environment variable **HADOOP\_USER\_NAME** and modify the code. If the code cannot be modified, run the following statement to add the environment variable:

```
export HADOOP_USER_NAME=test
```

### NOTE

The **test** user here is an example. To run the example code related to the Colocation operation, the user must be a member of the supergroup group.

- Step 5** Run the following commands to execute the jar packages.

```
hadoop jar HDFSTest-XXX.jar com.huawei.bigdata.hdfs.examples.HdfsExample
```

```
hadoop jar HDFSTest-XXX.jar
com.huawei.bigdata.hdfs.examples.ColocationExample
```

### NOTE

When **com.huawei.bigdata.hdfs.examples.ColocationExample** is run, the HDFSparameter **fs.defaultFS** cannot be set to **viewfs://ClusterX**.

----End

## 15.4.2.2 Compiling and Running an Application with the Client Not Installed

### Scenario

The Hadoop distributed file system (HDFS) application can run in the Linux operating system (OS) with the HDFS client not installed. After the application



code has been developed, you can upload the jar packages to the Linux OS and run the application.

## Prerequisite

- A JDK has been installed in the Linux environment. The version of the JDK must be consistent with that of the JDK used by IDEA to export the JAR package.
- When the host where the Linux OS runs is not a node of the cluster, you are required to set the mapping between the host name and IP address in the **hosts** file of the node where the Linux OS runs. The host name must be correctly mapped to the IP address.

## Procedure

- Step 1** Go to the local root directory of the project, copy the required configuration file to the conf folder of the local project and run the following command in Windows CLI to compress the package:

```
mvn -s "{maven_setting_path}" clean package
```

### NOTE

- In the preceding command, {maven\_setting\_path} is the path of the **settings.xml** file of the local Maven.
- After the package is successfully packed, obtain the JAR package from the target subdirectory in the root directory of the project.

- Step 2** Upload the exported jar packages to any directory in the running environment of the Linux OS, for example, **/opt/client**.

- Step 3** Create a **lib** folder in the Linux operating environment directory (for example, **/opt/client**) and upload the required **JAR** packages. The **lib** folder contains all the **JAR** packages that the project depends on. For details, see section [Preparing an Operating Environment](#).

- Step 4** Specify a user to run the example. There are two ways to specify the user: add the environment variable **HADOOP\_USER\_NAME** and modify the code. If the code cannot be modified, run the following statement to add the environment variable:
- ```
export HADOOP_USER_NAME=test
```

NOTE

The **test** user here is an example. To run the example code related to the Colocation operation, the user must be a member of the supergroup group.

- Step 5** Run the following commands to execute the jar packages.

```
java -cp HDFSTest-XXX.jar:lib/* com.huawei.bigdata.hdfs.examples.HdfsExample
```

```
java -cp HDFSTest-XXX.jar:lib/*  
com.huawei.bigdata.hdfs.examples.ColocationExample
```

NOTE

When **com.huawei.bigdata.hdfs.examples.ColocationExample** is run, the HDFSparameter **fs.defaultFS** cannot be set to **viewfs://ClusterX**.

----End

15.4.2.3 Checking the Commissioning Result

Scenario

After an HDFS application is run, you can learn the application running conditions by viewing the running result or HDFS logs.

Procedure

- **Learn the application running conditions by viewing the running result.**

- The running result of the HDFS example application is shown as follows:

```
[root@192-168-32-144 client]#hadoop jar HDFSTest-XXX.jar
com.huawei.bigdata.hdfs.examples.HdfsExample
WARNING: Use "yarn jar" to launch YARN applications.
17/10/26 19:11:44 INFO examples.HdfsExample: success to create path /user/hdfs-examples
17/10/26 19:11:44 INFO examples.HdfsExample: success to write.
17/10/26 19:11:45 INFO examples.HdfsExample: success to append.
17/10/26 19:11:45 INFO examples.HdfsExample: result is : hi, I am bigdata. It is successful if you
can see me.I append this content.
17/10/26 19:11:45 INFO examples.HdfsExample: success to read.
17/10/26 19:11:45 INFO examples.HdfsExample: success to delete the file /user/hdfs-examples/
test.txt
17/10/26 19:11:45 INFO examples.HdfsExample: success to delete path /user/hdfs-examples
17/10/26 19:11:45 INFO examples.HdfsExample: success to create path /user/hdfs-examples/
hdfs_example_1
17/10/26 19:11:45 INFO examples.HdfsExample: success to create path /user/hdfs-examples/
hdfs_example_0
17/10/26 19:11:45 INFO examples.HdfsExample: success to write.
17/10/26 19:11:45 INFO examples.HdfsExample: success to write.
17/10/26 19:11:46 INFO examples.HdfsExample: success to append.
17/10/26 19:11:46 INFO examples.HdfsExample: result is : hi, I am bigdata. It is successful if you
can see me.I append this content.
17/10/26 19:11:46 INFO examples.HdfsExample: success to read.
17/10/26 19:11:46 INFO examples.HdfsExample: success to delete the file /user/hdfs-examples/
hdfs_example_1/test.txt
17/10/26 19:11:46 INFO examples.HdfsExample: success to delete path /user/hdfs-examples/
hdfs_example_1
17/10/26 19:11:46 INFO examples.HdfsExample: success to append.
17/10/26 19:11:46 INFO examples.HdfsExample: result is : hi, I am bigdata. It is successful if you
can see me.I append this content.
17/10/26 19:11:46 INFO examples.HdfsExample: success to read.
17/10/26 19:11:46 INFO examples.HdfsExample: success to delete the file /user/hdfs-examples/
hdfs_example_0/test.txt
17/10/26 19:11:46 INFO examples.HdfsExample: success to delete path /user/hdfs-examples/
hdfs_example_0
```

- The running result of the Colocation example application is shown as follows:

```
[root@192-168-32-144 client]#hadoop jar HDFSTest-XXX.jar
com.huawei.bigdata.hdfs.examples.ColocationExample
WARNING: Use "yarn jar" to launch YARN applications.
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client
environment:zookeeper.version=V100R002C30, built on 10/19/2017 04:21 GMT
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:host.name=192-168-32-144
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:java.version=1.8.0_144
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:java.vendor=Oracle
Corporation
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:java.home=/opt/client/JDK/
jdk1.8.0_144/jre
.....
Create Group has finished.
Put file is running...
Put file has finished.
Delete file is running...
Delete file has finished.
Delete Group is running...
```

```
Delete Group has finished.  
17/10/26 19:12:39 INFO zookeeper.ZooKeeper: Session: 0x13000074b7e4687f closed  
17/10/26 19:12:39 INFO zookeeper.ClientCnxn: EventThread shut down for session:  
0x13000074b7e4687f  
17/10/26 19:12:39 INFO zookeeper.ZooKeeper: Session: 0x12000059699f69e1 closed  
17/10/26 19:12:39 INFO zookeeper.ClientCnxn: EventThread shut down for session:  
0x12000059699f69e1
```

- **Learn the application running conditions by viewing HDFS logs.**
The NameNode logs of HDFS offer immediate visibility into application running conditions. You can adjust application programs based on the logs.

15.5 More Information

15.5.1 Common API Introduction

15.5.1.1 Java API

For details about Hadoop distributed file system (HDFS) APIs, see:

<http://hadoop.apache.org/docs/r3.1.1/api/index.html>.

HDFS Common API

Common HDFS Java classes are as follows:

- **FileSystem**: the core class of client applications. For details about common APIs, see [Table 15-7](#).
- **FileStatus**: record the status of files and directories. For details about common APIs, see [Table 15-8](#).
- **DFSColocationAdmin**: API used to manage colocation group information. For details about common APIs, see [Table 15-9](#).
- **DFSColocationClient**: API used to manage colocation files. For details about common APIs, see [Table 15-10](#).

NOTE

- The system reserves only the mapping between nodes and locator IDs, but does not reserve the mapping between files and locator IDs. When a file is created using a Colocation interface, the file is created on the node that corresponds to a locator ID. File creation and writing must be performed using Colocation interfaces.
- After the file is written, subsequent operations on the file can use other open-source interfaces in addition to Colocation interfaces.
- The **DFSColocationClient** class inherits from the open-source **DistributedFileSystem** class and contains common file operation functions. If a user uses the **DFSColocationClient** class to create a Colocation file, the user is advanced to use the functions of this class in file operations.

Table 15-7 Common FileSystem APIs

| API | Description |
|---|---|
| public static
FileSystem
get(Configuration
conf) | FileSystem is the API class provided for users in the Hadoop class library. FileSystem is an abstract class. Concrete classes can be obtained only using the get method. The get method has multiple overload versions and is commonly used. |
| public
FSDataOutputStream
create(Path f) | This API is used to create files in the HDFS. <i>f</i> indicates a complete file path. |
| public void
copyFromLocalFile(Path
src, Path dst) | This API is used to upload local files to a specified directory in the HDFS. <i>src</i> and <i>dst</i> indicate complete file paths. |
| public boolean
mkdirs(Path f) | This API is used to create folders in the HDFS. <i>f</i> indicates a complete folder path. |
| public abstract
boolean rename(Path
src, Path dst) | This API is used to rename a specified HDFS file. <i>src</i> and <i>dst</i> indicate complete file paths. |
| public abstract
boolean delete(Path f,
boolean recursive) | This API is used to delete a specified HDFS file. <i>f</i> indicates the complete path of the file to be deleted, and recursive specifies recursive deletion. |
| public boolean
exists(Path f) | This API is used to query a specified HDFS file. <i>f</i> indicates a complete file path. |
| public FileStatus
getFileStatus(Path f) | This API is used to obtain the FileStatus object of a file or folder. The FileStatus object records status information of the file or folder, including the modification time and file directory. |
| public BlockLocation[]
getFileBlockLocations(FileStatus file,
long start, long len) | This API is used to query the block location of a specified file in an HDFS cluster. <i>file</i> indicates a complete file path, and <i>start</i> and <i>len</i> specify the block scope. |
| public
FSDataInputStream
open(Path f) | This API is used to open the output stream of a specified file in the HDFS and read the file using the API provided by the FSDataInputStream class. <i>f</i> indicates a complete file path. |
| public
FSDataOutputStream
create(Path f, boolean
overwrite) | This API is used to create the input stream of a specified file in the HDFS and write the file using the API provided by the FSDataOutputStream class. <i>f</i> indicates a complete file path. If overwrite is true , the file is rewritten if it exists; if overwrite is false , an error is reported if the file exists. |

| API | Description |
|--|---|
| public
FSDataOutputStream
append(Path f) | This API is used to open the input stream of a specified file in the HDFS and write the file using the API provided by the FSDataOutputStream class. <i>f</i> indicates a complete file path. |

Table 15-8 Common FileStatus APIs

| API | Description |
|--------------------------------------|---|
| public long
getModificationTime() | This API is used to query the modification time of a specified HDFS file. |
| public Path getPath() | This API is used to query all files in an HDFS directory. |

Table 15-9 Common DFSColocationAdmin APIs

| API | Description |
|---|---|
| public Map<String,
List<DatanodeInfo>>
createColocationGroup(String
groupId,String file) | This API is used to create a group based on the locatorIds information in the file. file indicates the file path. |
| public Map<String,
List<DatanodeInfo>>
createColocationGroup(String
groupId,List<String> locators) | This API is used to create a group based on the locatorIds information in the list in the memory. |
| public void
deleteColocationGroup(String
groupId) | This API is used to delete a group. |
| public List<String>
listColocationGroups() | This API is used to return all group information of Colocation. The returned group ID arrays are sorted by the creation time. |
| public List<DatanodeInfo>
getNodesForLocator(String
groupId, String locatorId) | This API is used to obtain the list of all nodes in the locator. |

Table 15-10 Common DFSColocationAdmin APIs

| API | Description |
|--|--|
| public FSDataOutputStream create(Path f, boolean overwrite, String groupId, String locatorId) | This API is used to create a FSDataOutputStream in colocation mode to allow users to write files in f.
f is the HDFS path.
overwrite indicates whether an existing file can be overwritten.
groupId and locatorId of the file specified by a user must exist. |
| public FSDataOutputStream create(final Path f, final FsPermission permission, final EnumSet<CreateFlag> cflags, final int bufferSize, final short replication, final long blockSize, final Progressable progress, final ChecksumOpt checksumOpt, final String groupId, final String locatorId) | The function of this API is the same as that of FSDataOutputStream create(Path f, boolean overwrite, String groupId, String locatorId), except that users are allowed to customize checksum. |
| public void close() | This API is used to close the connection. |

Table 15-11 HDFS client WebHdfsFileSystem API

| API | Description |
|--|--|
| public RemoteIterator<FileStatus> listStatusIterator(final Path) | This API will help in fetching the child files and folders information through multiple request using remote iterator, thus avoiding the user interface from becoming slow when there are millions of child information to be fetched. |

Glob path pattern based API to get LocatedFileStatus and Open file from FileStatus

Following APIs are added in DistributedFileSystem to get the FileStatus with block location and open file from FileStatus object. These APIs will reduce the number of RPC calls from client to Namenodes.

Table 15-12 FileSystem APIs

| Interface | Description |
|--|---|
| public LocatedFileStatus[] globLocatedStatus(Path, PathFilter, boolean) throws IOException | Return an array of LocatedFileStatus objects whose path names match pathPattern and pass the in path filter. |
| public FSDataInputStream open(FileStatus stat) throws IOException | If the stat is an instance of LocatedFileStatusHdfs that already have the location information, the InputStream is created without contacting NameNode. |

15.5.1.2 C API

Function Description

Users can use the C application programming interface (API) to create, read and write, append, and delete files. For details of the C API, see the following official guidelines:

<http://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-hdfs/LibHdfs.html>

Code Sample

The following code snippets are used as an example. For complete code, see the HDFS C sample code **HDFS/hdfs-c-example/hdfs_test.c** in the HDFS sample code decompression directory.

1. Configure the HDFS NameNode parameter and create the link connecting to the HDFS files.

```
hdfsFS fs = hdfsConnect("default", 0);
fprintf(stderr, "hdfsConnect- SUCCESS!\n");
```

2. Create the HDFS directory.

```
const char* dir = "/tmp/nativeTest";
int exitCode = hdfsCreateDirectory(fs, dir);
if( exitCode == -1 ){
    fprintf(stderr, "Failed to create directory %s \n", dir );
    exit(-1);
}
fprintf(stderr, "hdfsCreateDirectory- SUCCESS! : %s\n", dir);
```

3. Write files.

```
const char* file = "/tmp/nativeTest/testfile.txt";
hdfsFile writeFile = openFile(fs, (char*)file, O_WRONLY |O_CREAT, 0, 0, 0);
fprintf(stderr, "hdfsOpenFile- SUCCESS! for write : %s\n", file);

if(!hdfsFileOpenForWrite(writeFile)){
    fprintf(stderr, "Failed to open %s for writing.\n", file);
    exit(-1);
}

char* buffer = "Hadoop HDFS Native file write!";

hdfsWrite(fs, writeFile, (void*)buffer, strlen(buffer)+1);
fprintf(stderr, "hdfsWrite- SUCCESS! : %s\n", file);
```

```
printf("Flushing file data ....\n");
if (hdfsFlush(fs, writeFile)) {
    fprintf(stderr, "Failed to 'flush' %s\n", file);
    exit(-1);
}
hdfsCloseFile(fs, writeFile);
fprintf(stderr, "hdfsCloseFile- SUCCESS! : %s\n", file);
```

4. Read files.

```
hdfsFile readFile = openFile(fs, (char*)file, O_RDONLY, 100, 0, 0);
fprintf(stderr, "hdfsOpenFile- SUCCESS! for read : %s\n", file);

if(!hdfsFileIsOpenForRead(readFile)){
    fprintf(stderr, "Failed to open %s for reading.\n", file);
    exit(-1);
}

buffer = (char *) malloc(100);
tSize num_read = hdfsRead(fs, readFile, (void*)buffer, 100);
fprintf(stderr, "hdfsRead- SUCCESS!, Byte read : %d, File content : %s \n", num_read ,buffer);
hdfsCloseFile(fs, readFile);
```

5. From the specified location to read the file.

```
buffer = (char *) malloc(100);
readFile = openFile(fs, file, O_RDONLY, 100, 0, 0);
if (hdfsSeek(fs, readFile, 10)) {
    fprintf(stderr, "Failed to 'seek' %s\n", file);
    exit(-1);
}
num_read = hdfsRead(fs, readFile, (void*)buffer, 100);
fprintf(stderr, "hdfsSeek- SUCCESS!, Byte read : %d, File seek content : %s \n", num_read ,buffer);
hdfsCloseFile(fs, readFile);
```

6. Copy the file.

```
const char* destfile = "/tmp/nativeTest/testfile1.txt";
if (hdfsCopy(fs, file, fs, destfile)) {
    fprintf(stderr, "File copy failed, src : %s, des : %s \n", file, destfile);
    exit(-1);
}
fprintf(stderr, "hdfsCopy- SUCCESS!, File copied, src : %s, des : %s \n", file, destfile);
```

7. Move the file.

```
const char* mvfile = "/tmp/nativeTest/testfile2.txt";
if (hdfsMove(fs, destfile, fs, mvfile )) {
    fprintf(stderr, "File move failed, src : %s, des : %s \n", destfile , mvfile);
    exit(-1);
}
fprintf(stderr, "hdfsMove- SUCCESS!, File moved, src : %s, des : %s \n", destfile , mvfile);
```

8. Rename the file.

```
const char* renamefile = "/tmp/nativeTest/testfile3.txt";
if (hdfsRename(fs, mvfile, renamefile)) {
    fprintf(stderr, "File rename failed, Old name : %s, New name : %s \n", mvfile, renamefile);
    exit(-1);
}
fprintf(stderr, "hdfsRename- SUCCESS!, File renamed, Old name : %s, New name : %s \n", mvfile, renamefile);
```

9. Delete Files.

```
if (hdfsDelete(fs, renamefile, 0)) {
    fprintf(stderr, "File delete failed : %s \n", renamefile);
    exit(-1);
}
fprintf(stderr, "hdfsDelete- SUCCESS!, File deleted : %s\n",renamefile);
```

10. Set the number of replications.

```
if (hdfsSetReplication(fs, file, 10)) {
    fprintf(stderr, "Failed to set replication : %s \n", file );
    exit(-1);
}
fprintf(stderr, "hdfsSetReplication- SUCCESS!, Set replication 10 for %s\n",file);
```


11. Set users, user groups.

```
if (hdfsChown(fs, file, "root", "root")) {
    fprintf(stderr, "Failed to set chown : %s \n", file );
    exit(-1);
}
fprintf(stderr, "hdfsChown- SUCCESS!, Chown success for %s\n",file);
```

12. Set permissions.

```
if (hdfsChmod(fs, file, S_IRWXU | S_IRWXG | S_IRWXO)) {
    fprintf(stderr, "Failed to set chmod: %s \n", file );
    exit(-1);
}
fprintf(stderr, "hdfsChmod- SUCCESS!, Chmod success for %s\n",file);
```

13. Set the file time.

```
struct timeval now;
gettimeofday(&now, NULL);
if (hdfsUtime(fs, file, now.tv_sec, now.tv_sec)) {
    fprintf(stderr, "Failed to set time: %s \n", file );
    exit(-1);
}
fprintf(stderr, "hdfsUtime- SUCCESS!, Set time success for %s\n",file);
```

14. Get file information.

```
hdfsFileInfo *fileInfo = NULL;
if((fileInfo = hdfsGetPathInfo(fs, file)) != NULL) {
    printFileInfo(fileInfo);
    hdfsFreeFileInfo(fileInfo, 1);
    fprintf(stderr, "hdfsGetPathInfo - SUCCESS!\n");
}
```

15. Variable directory.

```
hdfsFileInfo *fileList = 0;
int numEntries = 0;
if((fileList = hdfsListDirectory(fs, dir, &numEntries)) != NULL) {
    int i = 0;
    for(i=0; i < numEntries; ++i) {
        printFileInfo(fileList+i);
    }
    hdfsFreeFileInfo(fileList, numEntries);
}
fprintf(stderr, "hdfsListDirectory- SUCCESS!, %s\n", dir);
```

16. Stream builder interfaces.

```
buffer = (char *) malloc(100);
struct hdfsStreamBuilder *builder= hdfsStreamBuilderAlloc(fs, (char*)file, O_RDONLY);
hdfsStreamBuilderSetBufferSize(builder,100);
hdfsStreamBuilderSetReplication(builder,20);
hdfsStreamBuilderSetDefaultBlockSize(builder,10485760);
readFile = hdfsStreamBuilderBuild(builder);
num_read = hdfsRead(fs, readFile, (void*)buffer, 100);
fprintf(stderr, "hdfsStreamBuilderBuild- SUCCESS! File read success. Byte read : %d, File content : %s\n", num_read ,buffer);
free(buffer);

struct hdfsReadStatistics *stats = NULL;
hdfsFileGetReadStatistics(readFile, &stats);
fprintf(stderr, "hdfsFileGetReadStatistics- SUCCESS! totalBytesRead : %" PRId64 ",
totalLocalBytesRead : %" PRId64 ", totalShortCircuitBytesRead : %" PRId64 ",
totalZeroCopyBytesRead : %" PRId64 "\n", stats->totalBytesRead , stats->totalLocalBytesRead, stats->totalShortCircuitBytesRead, stats->totalZeroCopyBytesRead);
hdfsFileFreeReadStatistics(stats);
```

17. Disconnect the HDFS links.

```
hdfsDisconnect(fs);
```

Preparing Running Environment

Install a client on the node. For example, install a client in the **/opt/client** directory.

Compiling and Running Applications in Linux

1. Go to the **/opt/client** directory and run the following command to import the environment variables of the C client:

```
cd /opt/client  
source bigdata_env
```

2. Go to the **/opt/client/HDFS/hadoop/hdfs-c-example** directory and run the following command to import the environment variables of the C client:

```
cd /opt/client/HDFS/hadoop/hdfs-c-example  
source component_env_C_example
```

3. Run the following command to clean the object files and executable files that are generated before:

make clean

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make clean  
rm -f hdfs_test.o  
rm -f hdfs_test
```

4. Run the following command to compile the new object files and executable files:

make (or make all)

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make  
cc -c -I/opt/client/HDFS/hadoop/include -Wall -o hdfs_test.o hdfs_test.c  
cc -o hdfs_test hdfs_test.o -lhdfs
```

5. Run the following command to create, write and read, append, and delete files:

make run

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make run  
./hdfs_test  
hdfsConnect- SUCCESS!  
hdfsCreateDirectory- SUCCESS! : /tmp/nativeTest  
hdfsOpenFile- SUCCESS! for write : /tmp/nativeTest/testfile.txt  
hdfsWrite- SUCCESS! : /tmp/nativeTest/testfile.txt  
Flushing file data ....  
hdfsCloseFile- SUCCESS! : /tmp/nativeTest/testfile.txt  
hdfsOpenFile- SUCCESS! for read : /tmp/nativeTest/testfile.txt  
hdfsRead- SUCCESS!, Byte read : 31, File content : Hadoop HDFS Native file write!  
hdfsSeek- SUCCESS!, Byte read : 21, File seek content : S Native file write!  
hdfsPread- SUCCESS!, Byte read : 10, File pread content : S Native f  
hdfsCopy- SUCCESS!, File copied, src : /tmp/nativeTest/testfile.txt, des : /tmp/nativeTest/testfile1.txt  
hdfsMove- SUCCESS!, File moved, src : /tmp/nativeTest/testfile1.txt, des : /tmp/nativeTest/testfile2.txt  
hdfsRename- SUCCESS!, File renamed, Old name : /tmp/nativeTest/testfile2.txt, New name : /tmp/  
nativeTest/testfile3.txt  
hdfsDelete- SUCCESS!, File deleted : /tmp/nativeTest/testfile3.txt  
hdfsSetReplication- SUCCESS!, Set replication 10 for /tmp/nativeTest/testfile.txt  
hdfsChown- SUCCESS!, Chown success for /tmp/nativeTest/testfile.txt  
hdfsChmod- SUCCESS!, Chmod success for /tmp/nativeTest/testfile.txt  
hdfsUtime- SUCCESS!, Set time success for /tmp/nativeTest/testfile.txt  
  
Name: hdfs://hacluster/tmp/nativeTest/testfile.txt, Type: F, Replication: 10, BlockSize: 134217728, Size:  
31, LastMod: 1500345260, Owner: root, Group: root, Permissions: 511 (rwxrwxrwx)  
hdfsGetPathInfo - SUCCESS!  
  
Name: hdfs://hacluster/tmp/nativeTest/testfile.txt, Type: F, Replication: 10, BlockSize: 134217728, Size:  
31, LastMod: 1500345260, Owner: root, Group: root, Permissions: 511 (rwxrwxrwx)
```

```
hdfsListDirectory- SUCCESS!, /tmp/nativeTest
hdfsTruncateFile- SUCCESS!, /tmp/nativeTest/testfile.txt
Block Size : 134217728
hdfsGetDefaultBlockSize- SUCCESS!
Block Size : 134217728 for file /tmp/nativeTest/testfile.txt
hdfsGetDefaultBlockSizeAtPath- SUCCESS!
HDFS Capacity : 102726873909
hdfsGetCapacity- SUCCESS!
HDFS Used : 4767076324
hdfsGetCapacity- SUCCESS!
hdfsExists- SUCCESS! /tmp/nativeTest/testfile.txt
hdfsConfGetStr- SUCCESS : hdfs://hacluster
hdfsStreamBuilderBuild- SUCCESS! File read success. Byte read : 31, File content : Hadoop HDFS
Native file write!
hdfsFileGetReadStatistics- SUCCESS! totalBytesRead : 31, totalLocalBytesRead : 0,
totalShortCircuitBytesRead : 0, totalZeroCopyBytesRead : 0
```

6. Enter the debug mode. (Optional)

make gdb

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make gdb
gdb hdfs_test
GNU gdb (GDB) SUSE (7.5.1-0.7.29)
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-suse-linux".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /opt/client/HDFS/hadoop/hdfs-c-example/hdfs_test...done.
(gdb)
```

15.5.1.3 HTTP REST API

Function Description

Users can use the application programming interface (API) of Representational State Transfer (REST) to create, read and write, append, and delete files. For details of the REST API, see the following official guidelines:

<http://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-hdfs/WebHDFS.html>.

Preparing Running Environment

- Step 1** Install the client. Install the client on the node. For example, install the client in the `/opt/client` directory. See details in "Installing the Client."

1. Prepare files `testFile` and `testFileAppend` and write content 'Hello, webhdfs user!' and 'Welcome back to webhdfs!'. Run the following command to prepare `testFile` and `testFileAppend` files:

```
touch testFile
```

```
vi testFile
```

Write 'Hello, webhdfs user!', save the files, and exit.

```
touch testFileAppend
```

```
vi testFileAppend
```

Write 'Welcome back to webhdfs!', save the files, and exit.

- Step 2** In normal mode, only the HTTP service is supported. **Log in to the FusionInsight Manager portal**, choose **Cluster > Name of the desired cluster > Services > HDFS > Configurations > All Configurations**. Type **dfs.http.policy** in the research box, select **HTTP_ONLY**, click **Save Configuration**, and select **Restart the affected services or instances**. Click **OK** to restart the HDFS service.

 **NOTE**

HTTP_ONLY is selected by default.

----End

Procedure

- Step 1** **Log in to the FusionInsight Manager portal**, click **Cluster > Name of the desired cluster > Services**, and then select **HDFS**. The HDFS page is displayed.

 **NOTE**

Because webhdfs is accessed through HTTP, you need to obtain the IP address of the active NameNode and the HTTP port.

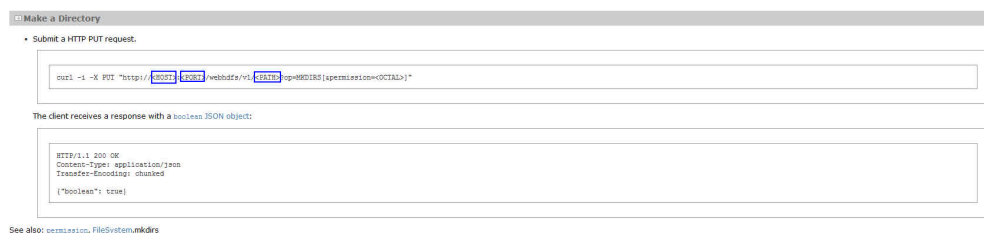
1. Click **Instances**, view the host name and IP address of the active NameNode.
2. Click **Configurations**, search **namenode.http.port** in the search box (9870).

- Step 2** Create a directory by referring to the following link:

http://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-hdfs/WebHDFS.html#Make_a_Directory

Click the link, **Figure 15-10** is displayed:

Figure 15-10 Example code of creating a directory



Go to the **/opt/client** directory, the installation directory of the client, and create the **huawei** directory.

1. Run the following command to check whether the **huawei** directory exists in the current path.

hdfs dfs -ls /

The running results are as follows:

```
linux1:/opt/client # hdfs dfs -ls /
16/04/22 16:10:02 INFO hdfs.PeerCache: SocketCache disabled.
Found 7 items
-rw-r--r--  3 hdfs  supergroup    0 2016-04-20 18:03 /PRE_CREATE_DIR.SUCCESS
drwxr-x---  - flume  hadoop      0 2016-04-20 18:02 /flume
drwx-----  - hbase  hadoop      0 2016-04-22 15:19 /hbase
drwxrwxrwx  - mapred hadoop      0 2016-04-20 18:02 /mr-history
```

```
drwxrwxrwx - spark supergroup 0 2016-04-22 15:19 /sparkJobHistory
drwxrwxrwx - hdfs hadoop 0 2016-04-22 14:51 /tmp
drwxrwxrwx - hdfs hadoop 0 2016-04-22 14:50 /user
```

The **huawei** directory does not exist in the current path.

2. Run the command in **Figure 15-10** that is named with **huawei**. Replace the <HOST> and <PORT> in the command with the host name or IP address and port number that are obtained in **Step 1**. Type the **huawei** as the directory in the <PATH>.

 **NOTE**

<HOST> can be replaced by the host name or IP address. It is noted that the port of HTTP is different from the port of HTTPS.

- Run the following command to access HTTP:

```
curl -i -X PUT --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei?
user.name=test&op=MKDIRS"
```

In the command, <HOST> is replaced by **linux1** and <PORT> is replaced by **9870**.

The test in the preceding command is the user who performs the operation. The user must confirm with the administrator for the permission.

- The running result is displayed as follows:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Expires: Thu, 14 Jul 2016 08:04:39 GMT
Date: Thu, 14 Jul 2016 08:04:39 GMT
Pragma: no-cache
Expires: Thu, 14 Jul 2016 08:04:39 GMT
Date: Thu, 14 Jul 2016 08:04:39 GMT
Pragma: no-cache
Content-Type: application/json
X-FRAME-OPTIONS: SAMEORIGIN
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs&t=simple&e=1468519479514&s=/j/
+ZnVrN7NSz1yKnB2JVlwkj0="; Path=/; Expires=Thu, 14-Jul-2016 18:04:39 GMT; HttpOnly
Transfer-Encoding: chunked
{"boolean":true}
```

If {"boolean":true} returns, the **huawei** directory is successfully created.

3. Run the following command to check the **huawei** directory in the path.

```
linux1:/opt/client # hdfs dfs -ls /
16/04/22 16:14:25 INFO hdfs.PeerCache: SocketCache disabled.
Found 8 items
-rw-r--r-- 3 hdfs supergroup 0 2016-04-20 18:03 /PRE_CREATE_DIR.SUCCESS
drwxr-x--- - flume hadoop 0 2016-04-20 18:02 /flume
drwx----- - hbase hadoop 0 2016-04-22 15:19 /hbase
drwxr-xr-x - hdfs supergroup 0 2016-04-22 16:13 /huawei
drwxrwxrwx - mapred hadoop 0 2016-04-20 18:02 /mr-history
drwxrwxrwx - spark supergroup 0 2016-04-22 16:12 /sparkJobHistory
drwxrwxrwx - hdfs hadoop 0 2016-04-22 14:51 /tmp
drwxrwxrwx - hdfs hadoop 0 2016-04-22 16:10 /user
```

Step 3 Create a command of the upload request to obtain the information about Location where the DataNode IP address is written in.

- Run the following command to access HTTP:

```
linux1:/opt/client # curl -i -X PUT --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs?
user.name=test&op=CREATE"
```
- The running result is displayed as follows:

```
HTTP/1.1 307 TEMPORARY_REDIRECT
Cache-Control: no-cache
Expires: Thu, 14 Jul 2016 08:53:07 GMT
Date: Thu, 14 Jul 2016 08:53:07 GMT
```

```
Pragma: no-cache
Expires: Thu, 14 Jul 2016 08:53:07 GMT
Date: Thu, 14 Jul 2016 08:53:07 GMT
Pragma: no-cache
Content-Type: application/octet-stream
X-FRAME-OPTIONS: SAMEORIGIN
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs&t=simple&e=1468522387880&s=OiksfRjvEkh/
Out9y2Ot2FvrXWk="; Path=/; Expires=Thu, 14-Jul-2016 18:53:07 GMT; HttpOnly
Location:
http://10-120-180-170:25010/webhdfs/v1/testHdfs?
op=CREATE&user.name=hdfs&namenoderpcaddress=hacluster&createflag=&createparent=true&overwr
ite=false
Content-Length: 0
```

Step 4 According to the Location information, create the **testHdfs** file in the **/huawei/** **testHdfs** file on the HDFS and upload the content in the local **testFile** file into the **testHdfs** file.

- Run the following command to access HTTP:
linux1:/opt/client # curl -i -X PUT -T testFile --negotiate -u: "http://10-120-180-170:25010/webhdfs/v1/testHdfs?
op=CREATE&user.name=test&namenoderpcaddress=hacluster&createflag=&createparent=true&overwri
te=false"
- The running result is displayed as follows:
HTTP/1.1 100 Continue
HTTP/1.1 201 Created
Location: hdfs://hacluster/testHdfs
Content-Length: 0
Connection: close

Step 5 Go to the **/huawei/testHdfs** directory and read the content of **testHdfs** file.

- Run the following command to access HTTP:
linux1:/opt/client # curl -L --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs??
user.name=test&op=OPEN"
- The running result is displayed as follows:
Hello, webhdfs user!

Step 6 Create a command of the upload request to obtain the information about Location where the DataNode IP address of **testHdfs** file is written in.

- Run the following command to access HTTP:
linux1:/opt/client # curl -i -X POST --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs??
user.name=test&op=APPEND"
- The running result is displayed as follows:
HTTP/1.1 307 TEMPORARY_REDIRECT
Cache-Control: no-cache
Expires: Thu, 14 Jul 2016 09:18:30 GMT
Date: Thu, 14 Jul 2016 09:18:30 GMT
Pragma: no-cache
Expires: Thu, 14 Jul 2016 09:18:30 GMT
Date: Thu, 14 Jul 2016 09:18:30 GMT
Pragma: no-cache
Content-Type: application/octet-stream
X-FRAME-OPTIONS: SAMEORIGIN
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs&t=simple&e=1468523910234&s=JGK
+6M6PsVMFdAw2cglHaKU1kBM="; Path=/; Expires=Thu, 14-Jul-2016 19:18:30 GMT; HttpOnly
Location:
http://10-120-180-170:25010/webhdfs/v1/testHdfs?
op=APPEND&user.name=hdfs&namenoderpcaddress=hacluster
Content-Length: 0

Step 7 According to the Location information, add the content in the local **testFileAppend** file to the **testHdfs** file that is in the **/huawei/testHdfs** directory of HDFS.

- Run the following command to access HTTP:
linux1:/opt/client # curl -i -X POST -T testFileAppend --negotiate -u: "http://linux1:25010/webhdfs/v1/huawei/testHdfs?user.name=test&op=APPEND&namenoderpcaddress=hacluster"
- The running result is displayed as follows:
HTTP/1.1 100 Continue
HTTP/1.1 200 OK
Content-Length: 0
Connection: close

Step 8 Go to the **/huawei/testHdfs** directory and read all content in the **testHdfs** file.

- Run the following command to access HTTP:
linux1:/opt/client # curl -L --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs?user.name=test&op=OPEN"
- The running result is displayed as follows:
Hello, webhdfs user!
Welcome back to webhdfs!

Step 9 List details of all directory and file information in the **huawei** directory of the HDFS.

LISTSTATUS will return all child files and folders information in a single request.

- Run the following command to access HTTP:
linux1:/opt/client # curl --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs?user.name=test&op=LISTSTATUS"
- The result is displayed as follows:
{"FileStatuses":[{"FileStatus":{"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergroup","length":70,"modificationTime":1462426678379,"owner":"test","pathSuffix":"","permission":"755","replication":3,"storagePolicy":0,"type":"FILE"}]}

LISTSTATUS along with size and startafter param will help in fetching the child files and folders information through multiple request, thus avoiding the user interface from becoming slow when there are millions of child information to be fetched.

- Run the following command to access HTTP:
linux1:/opt/client # curl --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/?user.name=test&op=LISTSTATUS&startafter=sparkJobHistory&size=1"
- The result is displayed as follows:
{"FileStatuses":[{"FileStatus":{"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergroup","length":70,"modificationTime":1462426678379,"owner":"test","pathSuffix":"testHdfs","permission":"755","replication":3,"storagePolicy":0,"type":"FILE"}]}

Step 10 Delete the **testHdfs** file that is in the **/huawei/testHdfs** directory of HDFS.

- Run the following command to access HTTP:
linux1:/opt/client # curl -i -X DELETE --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/testHdfs?user.name=test&op=DELETE"
- The running result is displayed as follows:
HTTP/1.1 200 OK
Cache-Control: no-cache
Expires: Thu, 14 Jul 2016 10:27:44 GMT
Date: Thu, 14 Jul 2016 10:27:44 GMT
Pragma: no-cache
Expires: Thu, 14 Jul 2016 10:27:44 GMT
Date: Thu, 14 Jul 2016 10:27:44 GMT
Pragma: no-cache
Content-Type: application/json

```
X-FRAME-OPTIONS: SAMEORIGIN
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs&t=simple&e=1468528064220&s=HrvUEd72+V5L4GwCLC/
sG3xTI0o="; Path=/; Expires=Thu, 14-Jul-2016 20:27:44 GMT; HttpOnly
Transfer-Encoding: chunked
{"boolean":true}
```

----End

The Key Management Server (KMS) uses the HTTP REST API to provide key management services for external systems. For details about the API, see

<http://hadoop.apache.org/docs/r3.1.1/hadoop-kms/index.html>.

NOTE

As REST API reference has done security hardening to prevent script injection attack. Through REST API reference, it cannot create directory and file name which contain those key words "<script ", "<iframe", "<frame", "javascript:".

15.5.2 Shell Command Introduce

HDFS Shell

You can use the Hadoop Distributed File System (HDFS) Shell command to perform operations on the HDFS, such as reading and writing files.

To run the HDFS Shell:

Go to the directory of HDFS client and enter the command. An example is shown as follows:

```
cd /opt/client/HDFS/hadoop/bin
```

```
hdfs dfs -mkdir /tmp/input
```

You can run the following command to seek help about HDFS commands:

```
hdfs --help
```

For details about the shell, see

<http://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-common/FileSystemShell.html>

Table 15-13 Transparent encryption-related commands

| Scenario | Operation | Command | Description |
|-------------------------------------|--------------|---|--|
| Hadoop shell command management key | Create keys. | <i>hadoop key create</i> <keyname> [-cipher <cipher>] [-size <size>] [-description <description>] [-attr <attribute=value>] [-provider <provider>] [-help] | The create subcommand creates a key for the name specified by the <keyname> argument within the provider specified by the -provider argument. You may specify a cipher with the -cipher argument. The default cipher is "AES/CTR/NoPadding" currently. The default keysize is 128. You may specify the requested key length using the -size argument. Arbitrary attribute=value style attributes may be specified using the -attr argument. The -attr may be specified for multiple times, once per attribute. |
| | Rollback | <i>hadoop key roll</i> <keyname> [-provider <provider>] [-help] | The roll subcommand creates a new version for the specified key within the provider indicated using the -provider argument. |
| | Delete keys | <i>hadoop key delete</i> <keyname> [-provider <provider>] [-f] [-help] | The delete subcommand deletes all versions of the key specified by the <keyname> argument within the provider specified by the -provider argument. The command asks for user confirmation unless -f is specified. |
| | View keys | <i>hadoop key list</i> [-provider <provider>] [-metadata] [-help] | The list subcommand displays the keynames contained in a particular provider as configured in core-site.xml or specified with the -provider argument. The -metadata argument displays the metadata. |

Table 15-14 Shell commands of Colocation client

| Operation | Command | Description |
|---|---|---|
| Group creation | hdfs colocationadmin - createGroup -groupId <groupId> -locatorIds <comma separated locatorIDs> or -file <path of the file contains all of locatorIDs> | Used to create a group. In the command, groupId is the group name and locatorID is the locator name. You can enter comma-separated locator IDs using command lines. You can also write locator IDs into a file so that the system can obtain the locator IDs by reading the file. |
| Group deletion | hdfs colocationadmin - deleteGroup <groupId> | Used to delete the specified group. |
| Group query | hdfs colocationadmin - queryGroup <groupId> | Used to query details about a specified group, including locators in the group and information about each locator and its corresponding DataNode. |
| Viewing all groups | hdfs colocationadmin - listGroups | Used to list all groups and their creation time. |
| Setting ACL permissions on Colocation directories | hdfs colocationadmin - setAcl | Used to set ACL permissions on Colocation directories in ZooKeeper.
The default root directory of Colocation in ZooKeeper is /hadoop/colocationDetails . |

15.5.3 HDFS Access Configuration on Windows Using EIPs

Scenario

This section describes how to bind Elastic IP addresses (EIPs) to a cluster and configure HDFS files so that sample files can be compiled locally.

This section uses HdfsExample as an example.

Procedure

Step 1 Apply for an EIP for each node in the cluster and add public IP addresses and corresponding host domain names of all nodes to the Windows local **hosts** file. (If a host name contains uppercase letters, change them to lowercase letters.)

1. On the VPC console, apply for EIPs (the number of EIPs you buy should be equal to the number of nodes in the cluster), click the name of each node in the MRS cluster, and bind an EIP to each node on the **EIPs** page.

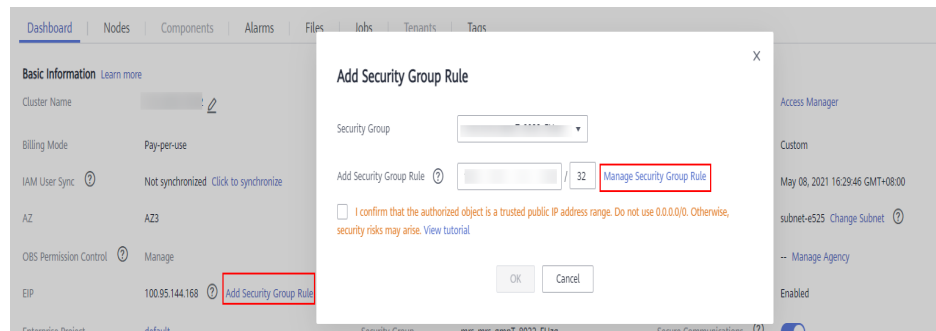
For details, see **Virtual Private Cloud > User Guide > EIP > Assigning an EIP and Binding It to an ECS**.

2. Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.

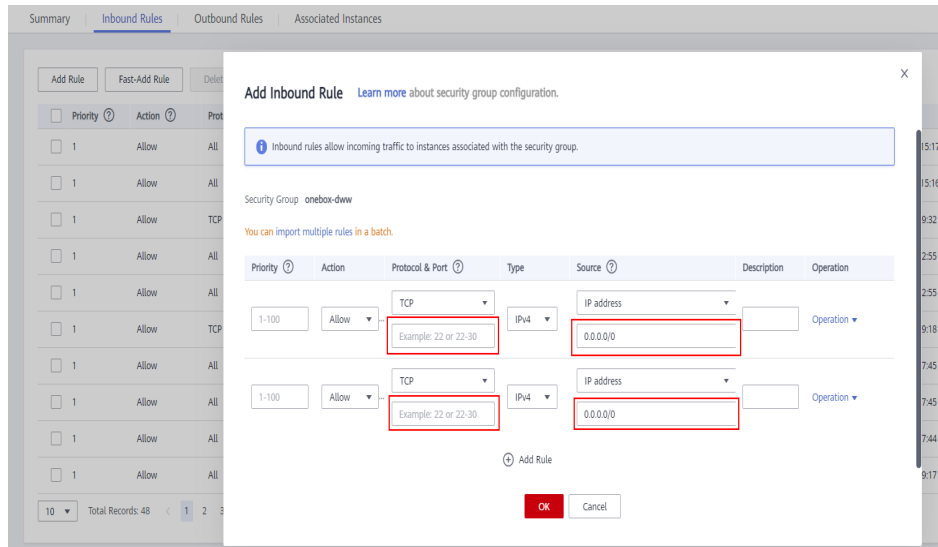
```
1 Mapping between public IP addresses and private IP addresses
2 100.95.144.168 172.16.0.120
3 100.95.144.168 172.16.0.42
4 100.95.144.168 172.16.0.64
5 100.95.144.168 172.16.0.200
6 100.93.144.168 172.16.0.139
7 100.93.144.168 172.16.0.214
8
9 hosts file in the cluster
10 172.16.0.120 node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
11 172.16.0.42 node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com.
12 172.16.0.62 node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
13 172.16.0.200 node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
14 172.16.0.139 node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
15 172.16.0.214 node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.
16
17 hosts file that should be added to Windows
18 100.95.144.168 node-group-1xzi00002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
19 100.95.144.168 node-master3vint.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com.
20 100.93.144.168 node-group-1xzi00003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
21 100.95.144.168 node-master1ceip.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
22 100.93.144.168 node-group-1xzi00001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
23 100.93.144.168 node-master2pvnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.
```

Step 2 Configure security group rules for the cluster.

1. On the **Dashboard** page, choose **Add Security Group Rule > Manage Security Group Rule**.



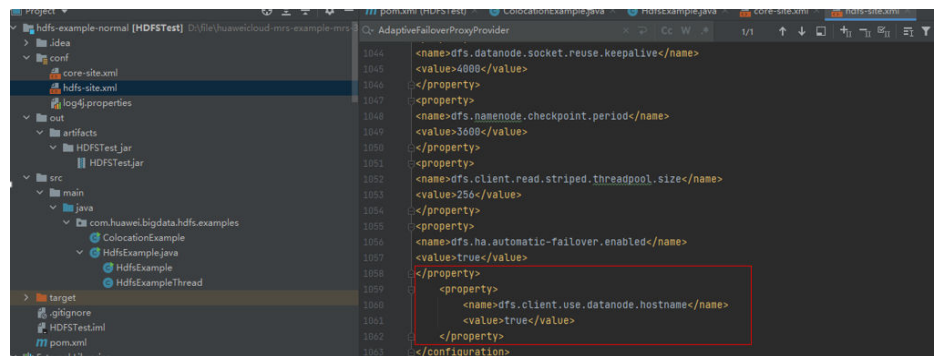
2. On the **Inbound Rules** tab page, click **Add Rule**. In the **Add Inbound Rule** dialog box, configure the Windows IP addresses and ports 8020 and 9866.



Step 3 On Manager, choose **Cluster > Services > HDFS > More > Download Client**, copy the **core-site.xml** and **hdfs-site.xml** files on the client to the **conf** directory of the sample project, and add the following content to the **hdfs-site.xml** file.

```
<property>
<name>dfs.client.use.datanode.hostname</name>
<value>>true</value>
</property>
```

(Change the DataNode communication mode to hostname.)



When you run the sample project, an error message indicating that **hadoop_home** does not exist may be displayed. You can ignore the error.

```
private void mkdir() throws IOException {
    Path destPath = new Path(this.destPath);
    if (!createPath(destPath)) {
        LOG.error("failed to create destPath " + this.destPath);
        return;
    }
    LOG.info("success to create path " + this.destPath);
}

/**
 * set storage policy to path
 * @param policyName Policy Name can be accepted:
 * + cli-WDT
 * + cli-WARN
 */
```

```
Run: HDFSExample
1 [main] WARN org.apache.hadoop.util.Shell - Did not find winutils.exe: {}
java.io.FileNotFoundException: Create breakpoint : java.io.FileNotFoundException: HADOOP_HOME and hadoop.home.dir are unset. -see https://wiki.apache.org/hadoop/WindowsProblems
at org.apache.hadoop.util.Shell.fileNotFoundException(Shell.java:350)
at org.apache.hadoop.util.Shell.getHadoopHomeDir(Shell.java:223)
at org.apache.hadoop.util.Shell.getQualifiedBin(Shell.java:354)
at org.apache.hadoop.util.Shell.<clinit>(Shell.java:621)
at org.apache.hadoop.util.StringUtils.<clinit>(StringUtils.java:72)
at org.apache.hadoop.fs.FileSystemCacheKey.<clinit>(FileSystem.java:352)
at org.apache.hadoop.fs.FileSystemCacheKey.<clinit>(FileSystem.java:322)
at org.apache.hadoop.fs.FileSystemCache.get(FileSystem.java:361)
at org.apache.hadoop.fs.FileSystem.get(FileSystem.java:483)
at org.apache.hadoop.fs.FileSystem.<clinit>(FileSystem.java:231)
at com.huawei.bigdata.hdfs.examples.HDFSExample.initializeFile(HDFSExample.java:126)
at com.huawei.bigdata.hdfs.examples.HDFSExample.<clinit>(HDFSExample.java:46)
at com.huawei.bigdata.hdfs.examples.HDFSExample.main(HDFSExample.java:14)
Caused by: java.io.FileNotFoundException: Create breakpoint : HADOOP_HOME and hadoop.home.dir are unset.
at org.apache.hadoop.util.Shell.checkHadoopHomeInner(Shell.java:276)
at org.apache.hadoop.util.Shell.checkHadoopHome(Shell.java:461)
at org.apache.hadoop.util.Shell.<clinit>(Shell.java:313)
... 9 more
25 [main] WARN org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
800 [main] WARN org.apache.hadoop.hdfs.ShortCircuitDomainSocketFactory - The short-circuit local reads feature cannot be used because UNIX domain sockets are not available on Windows
```

----End

16 Hive Development Guide (Security Mode)

16.1 Overview

16.1.1 Application Development Overview

Hive Introduction

Hive is an open-source data warehouse built on Hadoop. It stores structured data and provides basic data analysis services using the Hive query language (HQL), a language like the SQL. Hive converts HQL statements to Mapreduce or Spark jobs for querying and analyzing massive data stored in Hadoop clusters.

Hive provides the following features:

- Extracts, transforms, and loads (ETL) data using HQL.
- Analyzes massive structured data using HQL.
- Supports flexible data storage formats, including JavaScript object notation (JSON), comma separated values (CSV), TextFile, RCFile, ORCFIELD, and SequenceFile, and supports custom extensions.
- Multiple client connection modes. Interfaces, such as JDBC and Thrift interfaces are supported.

Hive applies to offline massive data analysis (such as log and cluster status analysis), large-scale data mining (such as user behavior analysis, interest region analysis, and region display), and other scenarios.

To ensure Hive high availability (HA), user data security, and service access security, MRS incorporates the following features based on Hive 3.1.0:

- Kerberos security authentication
- Data file encryption
- Complete rights management

For Hive features in the Open Source Community, see <https://cwiki.apache.org/confluence/display/hive/designdocs>.

16.1.2 Common Concepts

- **keytab file**
The keytab file is a key file that stores user information. Applications use the key file for API authentication on MRS.
- **Client**
Users can access the server from the client through the Java API and Thrift API to perform Hive-related operations.
- **HQL**
Similar to SQL
- **HCatalog**
HCatalog is a table information management layer created based on Hive metadata and absorbs DDL commands of Hive. HCatalog provides read/write interfaces for Mapreduce and provides Hive command line interfaces (CLIs) for defining data and querying metadata. Hive and Mapreduce development personnel can share metadata based on the HCatalog component of MRS, preventing intermediate conversion and adjustment and improving the data processing efficiency.
- **WebHCat**
WebHCat running users use Rest APIs to perform operations, such as running Hive DDL commands, submitting Mapreduce tasks, and querying Mapreduce task execution results.

16.1.3 Required Permissions

Before developing an application based on this guide, ensure that the basic permissions of the users belong to a Hive group and obtain additional operation permissions from the system administrator. **Table 16-1** describes the permission required for each operation. To run example programs, you must have the create permission for the default database.

Table 16-1 Required permissions

Operation Type/ Functional Object	Operation	Required Permission
DATABASE	CREATE DATABASE <i>dbname</i> [LOCATION "hdfs_path"]	If the HDFS path hdfs_path is specified, the ownership and RWX permission of hdfs_path are required.
	DROP DATABASE <i>dbname</i>	The database dbname ownership is required.
	ALTER DATABASE <i>dbname</i> SET OWNER <i>user_or_role</i>	The admin permission is required.

Operation Type/ Functional Object	Operation	Required Permission
TABLE	CREATE TABLE <i>table_a</i>	The create permission for the database is required.
	CREATE TABLE <i>table_a</i> AS SELECT <i>table_b</i>	The create permission for the database and the select permission for table_b are required.
	CREATE TABLE <i>table_a</i> LIKE <i>table_b</i>	The create permission for the database is required.
	CREATE [EXTERNAL] TABLE <i>table_a</i> LOCATION " <i>hdfs_path</i> "	The create permission for the database, and the ownership and RWX permission of hdfs_path on HDFS are required.
	DROP TABLE <i>table_a</i>	The ownership of table_a is required.
	ALTER TABLE <i>table_a</i> SET LOCATION " <i>hdfs_path</i> "	The ownership of table_a , and the ownership and RWX permission of hdfs_path on HDFS are required.
	ALTER TABLE <i>table_a</i> SET <i>FILEFORMAT</i>	The ownership of table_a is required.
	TRUNCATE TABLE <i>table_a</i>	The ownership of table_a is required.
	ANALYZE TABLE <i>table_a</i> COMPUTE STATISTICS	The select and insert permission for table_a is required.
	SHOW TBLPROPERTIES <i>table_a</i>	The select permission for table_a is required.
SHOW CREATE TABLE <i>table_a</i>	The select permission with grant option for table_a is required.	
Alter	ALTER TABLE <i>table_a</i> ADD COLUMN	The ownership of table_a is required.
	ALTER TABLE <i>table_a</i> REPLACE COLUMN	The ownership of table_a is required.
	ALTER TABLE <i>table_a</i> RENAME	The ownership of table_a is required.
	ALTER TABLE <i>table_a</i> SET SERDE	The ownership of table_a is required.

Operation Type/ Functional Object	Operation	Required Permission
	ALTER TABLE <i>table_a</i> CLUSTER BY	The ownership of table_a is required.
PARTITION	ALTER TABLE <i>table_a</i> ADD PARTITION <i>partition_spec</i> LOCATION " <i>hdfs_path</i> "	The insert permission for table_a , and the ownership and RWX permission of hdfs_path on HDFS are required.
	ALTER TABLE <i>table_a</i> DROP PARTITION <i>partition_spec</i>	The delete permission for table_a is required.
	ALTER TABLE <i>table_a</i> PARTITION <i>partition_spec</i> SET LOCATION " <i>hdfs_path</i> "	The ownership of table_a , and the ownership and RWX permission of hdfs_path on HDFS are required.
	ALTER TABLE <i>table_a</i> PARTITION <i>partition_spec</i> SET FILEFORMAT	The ownership of table_a is required.
LOAD	LOAD INPATH ' <i>hdfs_path</i> ' INTO TABLE <i>table_a</i>	The insert permission for table_a , and the ownership and RWX permission of hdfs_path on HDFS are required.
INSERT	INSERT TABLE <i>table_a</i> SELECT FROM <i>table_b</i>	The update permission for table_a and select permission for table_b are required.
SELECT	SELECT * FROM <i>table_a</i>	The select permission for table_a is required.
	SELECT FROM <i>table_a</i> JOIN <i>table_b</i>	The select permission for table_a and table_b , the Submit permission of the default Yarn queue is required.
	SELECT FROM (SELECT FROM <i>table_a</i> UNION ALL SELECT FROM <i>table_b</i>)	The select permission for table_a and table_b , the Submit permission of the default Yarn queue is required.

Operation Type/ Functional Object	Operation	Required Permission
EXPLAIN	EXPLAIN [EXTENDED] DEPENDENCY] <i>query</i>	The RX permissions for related table directory is required.
VIEW	CREATE VIEW <i>view_name</i> AS SELECT ...	The select permission with grant option for related tables is required.
	ALTER VIEW <i>view_name</i> RENAME TO <i>new_view_name</i>	The ownership of view_name is required.
	DROP VIEW <i>view_name</i>	The ownership of view_name is required.
INDEX	CREATE INDEX <i>index_name</i> ON TABLE <i>base_table_name</i> (<i>col_name</i> , ...) AS <i>index_type</i>	The ownership of table_a is required.
	DROP INDEX <i>index_name</i> ON <i>table_name</i>	The ownership of index_name is required.
	ALTER INDEX <i>index_name</i> ON <i>table_name</i> REBUILD	The ownership of index_name is required.
FUNCTION	CREATE [TEMPORARY] FUNCTION <i>function_name</i> AS ' <i>class_name</i> '	The admin permission is required.
	DROP [TEMPORARY] <i>function_name</i>	The admin permission is required.
MACRO	CREATE TEMPORARY MACRO <i>macro_name</i> ...	The admin permission is required.
	DROP TEMPORARY MACRO <i>macro_name</i>	The admin permission is required.

 NOTE

- You can perform all the previous operations when owning the **admin** permission of Hive and the corresponding directory permission of HDFS.
- If the current component uses Ranger for permission control, you need to configure permission management policies based on Ranger. For details, see [Adding a Ranger Access Permission Policy for Hive](#).

16.1.4 Development Process

Figure 16-1 and Table 16-2 illustrates each phase of the development process.

Figure 16-1 Hive application development process

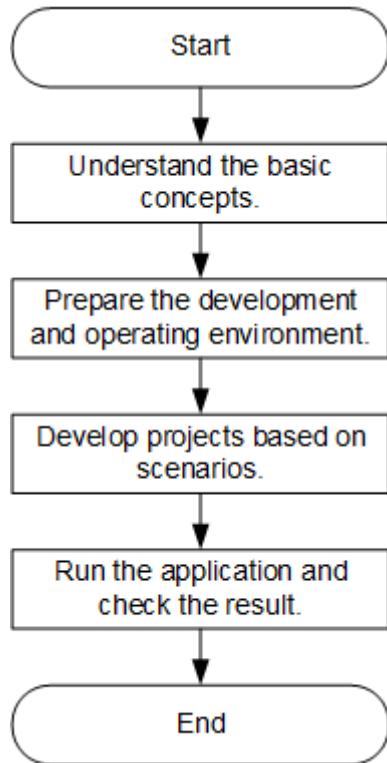


Table 16-2 Description of the Hive application development process

Phase	Description	Reference Document
Understand the basic concepts.	Before application development, understand common concepts about Hive.	Common Concepts
Prepare the development and operating environment.	Hive applications support Java and Python development languages. You are advised to use the IntelliJ IDEA tool to configure the development environment based on the language.	Preparing the Environment

Phase	Description	Reference Document
Develop projects based on scenarios.	Provides example projects of the Java and Python languages as well as example projects covering table creation, data load, and data query.	Developing an Application
Run the application and view results.	Describes how to submit a developed application for compiling and view the result.	Commissioning Applications

16.2 Preparing the Environment

16.2.1 Preparing Development and Operating Environment

Preparing Development Environment

Hive applications can be developed by using the JDBC/Python/Python3 API. The following table describes the development and operating environment to be prepared.

Table 16-3 JDBC development environment

Item	Description
OS	<ul style="list-style-type: none"> Development environment: Windows OS. Windows 7 or later is supported. Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.

Item	Description
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <p>For x86 nodes that run clients, the following JDKs can be used:</p> <ul style="list-style-type: none"> • Oracle JDK versions: 1.8 • IBM JDK versions: 1.8.5.11 <p>For nodes that run TaiShan and clients, the following JDK can be used:</p> <ul style="list-style-type: none"> • OpenJDK: 1.8.0_272 <p>NOTE</p> <p>The server supports only TLS V1.2 or later to ensure security.</p> <p>By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls.</p>
IntelliJ IDEA installation and configuration	<p>Tool used for developing Hive applications. Requirements are as follows:</p> <p>JDK 1.8 is used. IntelliJ IDEA 2019.1 or other compatible versions are used.</p> <p>NOTE</p> <ul style="list-style-type: none"> • If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK. • If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK. • If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.
Installation of Maven	<p>Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.</p>
Developer account preparation	<p>See Preparing a Developer Account for configuration.</p>
7-zip	<p>Used to decompress .zip and .rar packages. The 7-Zip 16.04 is supported.</p>

Table 16-4 Python development environment

Item	Description
OS	Development and operating environment: Linux OS
Python installation	Python is a tool used to develop Hive applications. Its version must be 2.6.6 or later but should not be later than 2.7.13.
setuptools installation	Basic configuration for the Python development environment. The version must be 5.0 or later.
Developer account preparation	See Preparing a Developer Account for configuration.

 **NOTE**

For details about how to install and configure the Python development tool, see the [Configuring the Python Sample Project](#).

Table 16-5 Python3 development environment

Item	Description
OS	Development and operating environment: Linux OS
Python3 installation	Python3 is a tool used to develop Hive applications. Its version must be 3.6 or later but should not be later than 3.8.
setuptools installation	Basic configuration for the Python3 development environment. The version must be 47.3.1.
Developer account preparation	See Preparing a Developer Account for configuration.

 **NOTE**

For details about how to install and configure the Python3 development tool, see the [Configuring the Python3 Sample Project](#).

Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.

- a. **Log in to the FusionInsight Manager portal** and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Configuration Files Only**. Select the platform type based on the type of the node where the client is to be installed (select **x86_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is

FusionInsight_Cluster_1_Services_Client.tar, decompress it to obtain **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar** file.

Then, decompress

FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar file to the **D:\FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles\Hive\config** and manually import the configuration file to the configuration file directory (usually the **resources** folder) of the Hive sample project. The keytab file obtained during the **Preparing a Developer Account** is also stored in this directory. **Table 16-6** describes the main configuration files.

Table 16-6 Configuration file

Document Name	Function
hivemetastore-site.xml	Configures Hive parameters.
hiveclient.properties	Configures Hive parameters.
user.keytab	Provides Hive user information for Kerberos security authentication.
krb5.conf	Contains Kerberos server configuration information.
core-site.xml	Configures Hive parameters.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client on the node. For example, the client installation directory is **/opt/client**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. **Log in to the FusionInsight Manager portal**. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight_Cluster_1_Services_ClientConfig/Hive/config** directory to the **src/main/resources** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/client/src/main/resources**.

For example, if the client software package is **FusionInsight_Cluster_1_Services_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Hive/config/* root@IP address of the client node:/opt/client/src/main/resources
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 16-7](#) describes the main configuration files.

Table 16-7 Configuration files

File	Function
hivemetastore-site.xml	Configures Hive parameters.
hiveclient.properties	Configures Hive parameters.
user.keytab	Provides Hive user information for Kerberos security authentication.
krb5.conf	Contains Kerberos server configuration information.
core-site.xml	Configures Hive parameters.

- c. Check the network connection of the client node.
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no,

manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

16.2.2 Configuring the JDBC Sample Project

Scenario

To run the JDBC API example codes of the Hive component of MRS, perform the following operations.

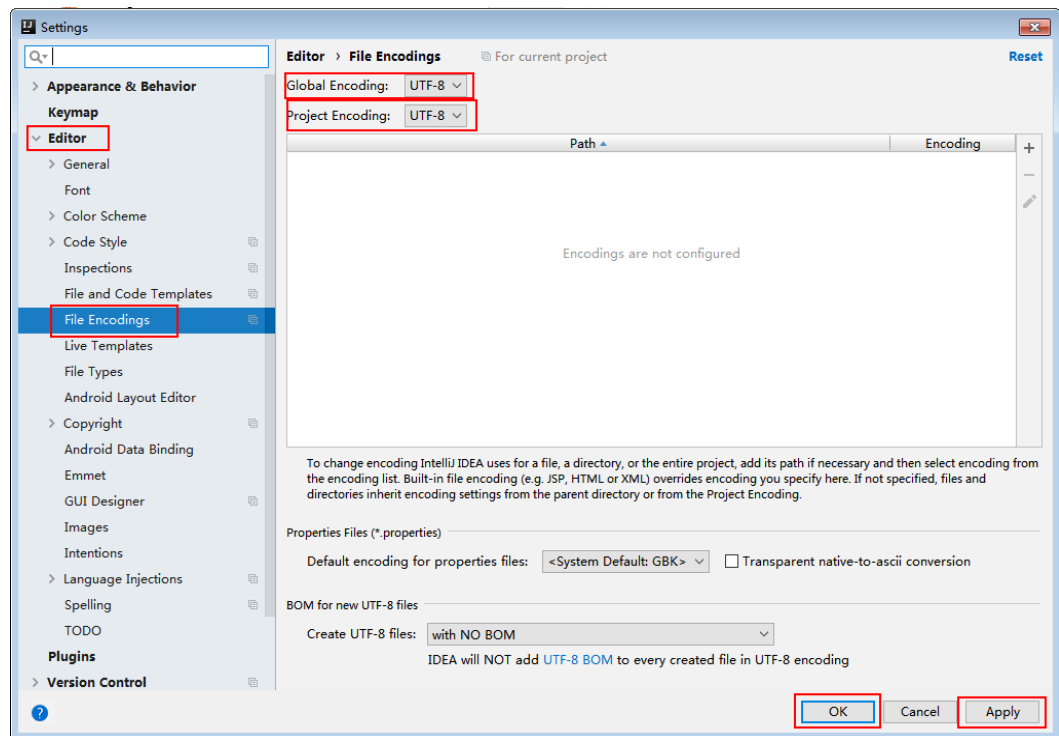
NOTE

- The following uses the development of an application for connecting the Hive service in JDBC mode on Windows as an example.
- After importing the jdbc-example sample project, you need to change xxx of USER_NAME = "xxx" in the code to the development user created in [Preparing a Developer Account](#).

Procedure

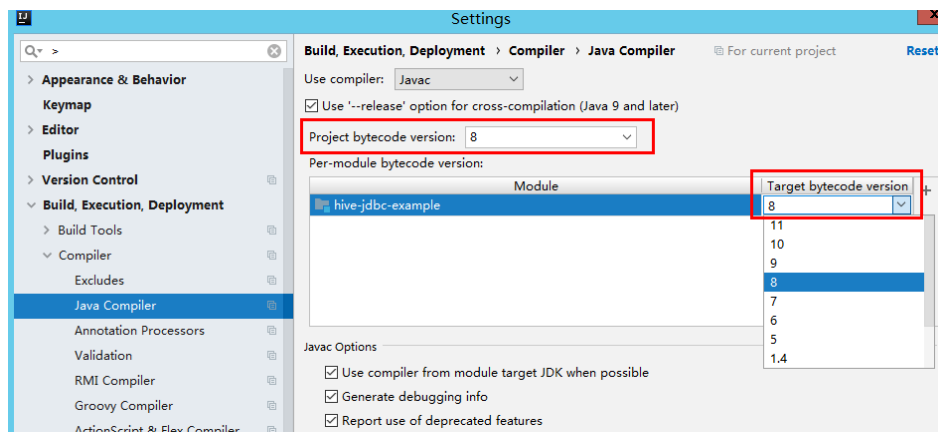
- Step 1** Obtain the sample project folder **hive-jdbc-example** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the **user.keytab** and **krb5.conf** files obtained in [Preparing a Developer Account](#) to the **hive-jdbc-example\src\main\resources** directory of the sample project.
- Step 3** Go to the client decompression path **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles\Hive\config** and manually import the **core-site.xml** and **hiveclient.properties** files to the **hive-jdbc-example\src\main\resources** directory of the sample project.
- Step 4** Import the example project to the IntelliJ IDEA development environment.
 1. On the IntelliJ IDEA menu bar, choose **File > Open....** The **Open File or Project** window is displayed.
 2. In the displayed window, select the folder **hive-jdbc-example**, and click **OK**. On Windows, the path cannot contain any space.
- Step 5** Set the IntelliJ IDEA text file coding format to prevent garbled characters.
 1. On the IntelliJ IDEA menu bar, choose **File > Settings**. The **Settings** window is displayed.
 2. In the navigating tree on the left, choose **Editor > File Encodings**. In the **Project Encoding** area and **Global Encoding** area, set the value to **UTF-8**, and click **Apply** and **OK**, as shown in [Figure 16-2](#).

Figure 16-2 Setting the IntelliJ IDEA coding format

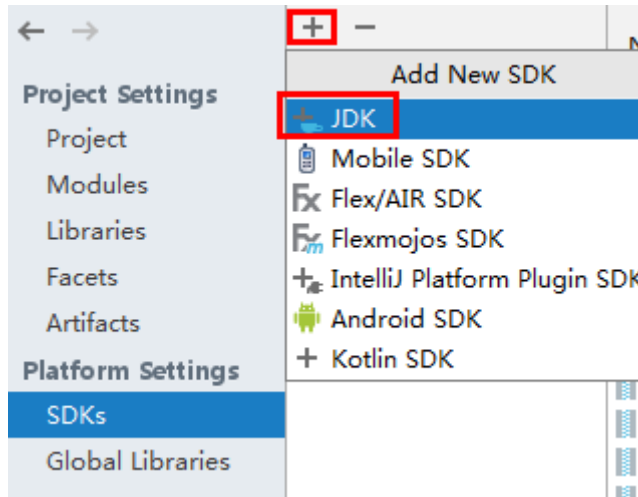


Step 6 Set the JDK of the project.

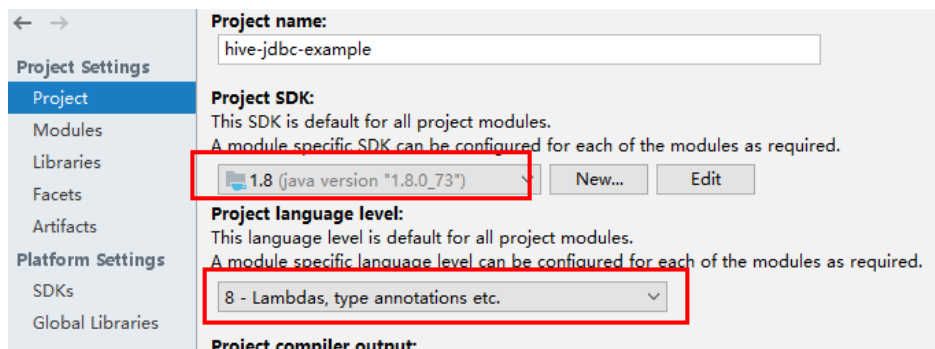
1. On the IntelliJ IDEA menu bar, choose **File > Settings...**. The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **hive-jdbc-example**.



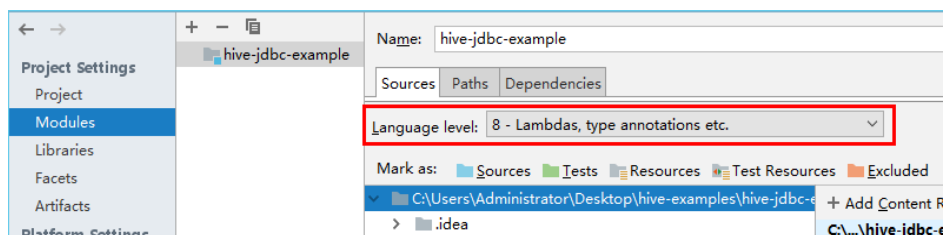
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File > Project Structure...**. The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



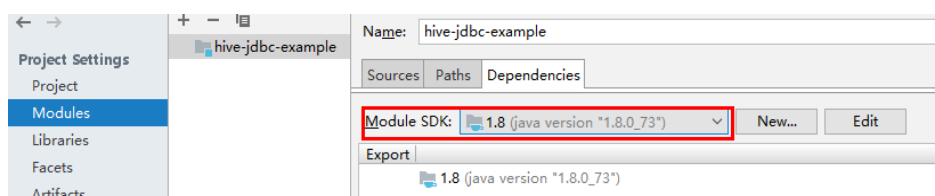
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Source** page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



On the **Dependencies** page, change the value of **Module SDK** to the JDK added in SDKs.

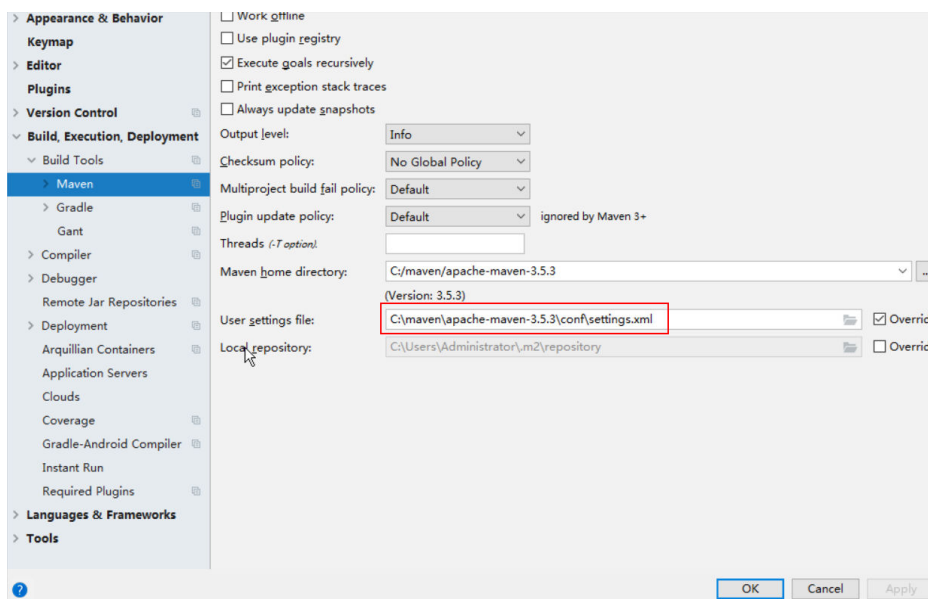


11. Click **Apply** and **OK**.

Step 7 Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is *<Local Maven installation directory>\conf\settings.xml*.

Figure 16-3 Directory for storing the **settings.xml** file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply**, and then click **OK**.

----End

16.2.3 Configuring the Hcatalog Sample Project

Scenario

To run the HCatalog interface example codes of the Hive component of MRS, perform the following operations.

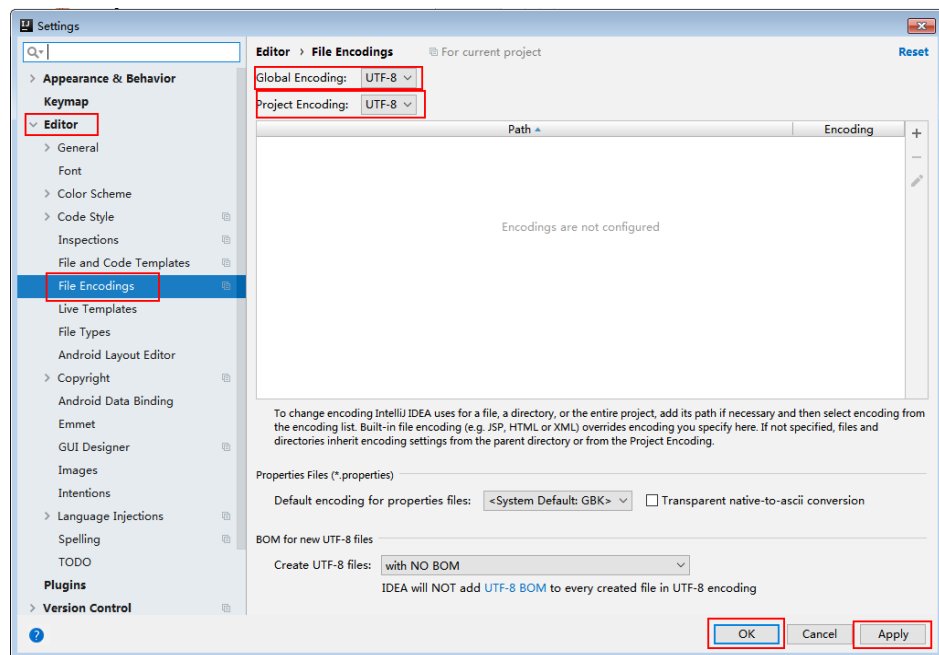
NOTE

The following uses the development of an application for connecting the Hive service in HCatalog mode on Windows as an example.

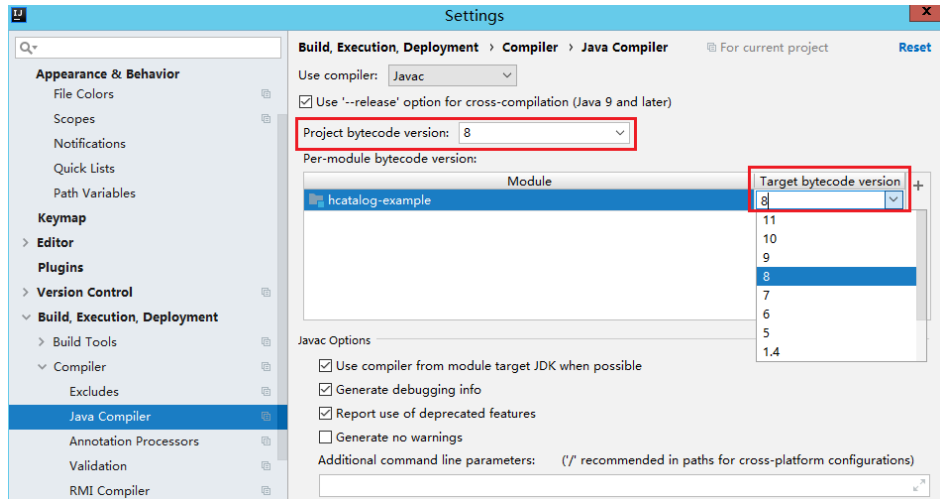
Procedure

- Step 1** Obtain the sample project folder **hcatalog-example** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Import the example project to the IntelliJ IDEA development environment.
1. On the IntelliJ IDEA menu bar, choose **File > Open....** The **Open File or Project** window is displayed.
 2. In the displayed window, select the folder **hcatalog-example**, and click **OK**. On Windows, the path cannot contain any space.
- Step 3** Set the IntelliJ IDEA text file coding format to prevent garbled characters.
1. On the IntelliJ IDEA menu bar, choose **File > Settings**. The **Settings** window is displayed.
 2. Choose **Editor > File Encodings** from the navigation tree. In the **Project Encoding** area and **Global Encoding** area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 16-4](#).

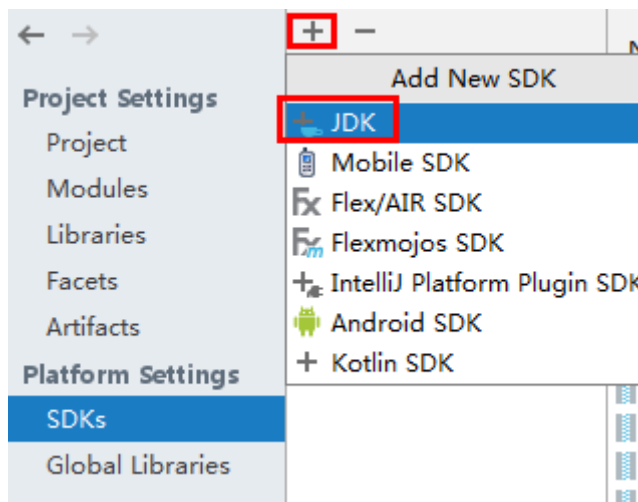
Figure 16-4 Setting the IntelliJ IDEA coding format



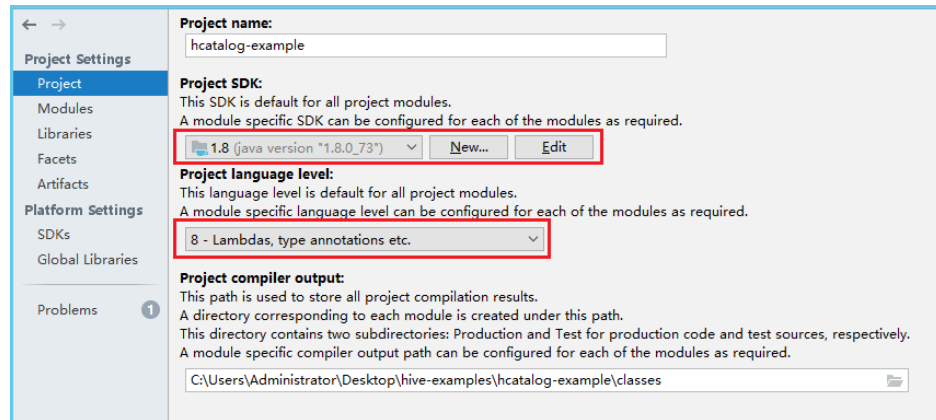
- Step 4** Set the JDK of the project.
1. On the IntelliJ IDEA menu bar, choose **File > Settings....** The **Settings** window is displayed.
 2. Choose **Build, Execution, Deployment > Compiler > Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **hive-jdbc-example**.



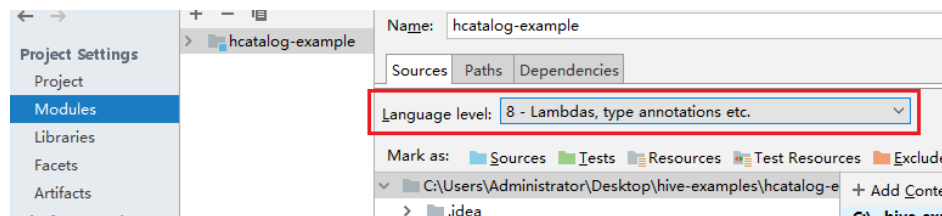
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File > Project Structure...** The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



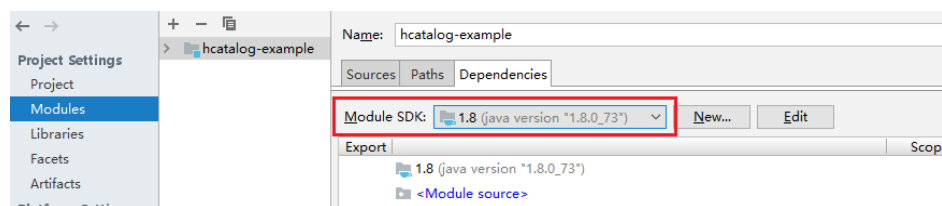
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Source** page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



11. On the **Dependencies** page, change the value of **Module SDK** to the JDK added in SDKs.

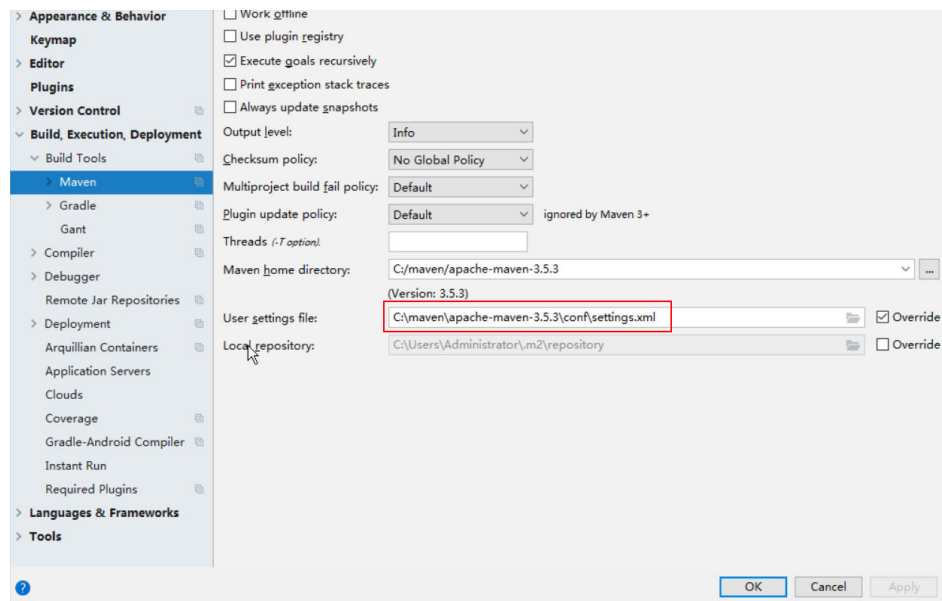


12. Click **Apply** and **OK**.

Step 5 Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is `<Local Maven installation directory>\conf\settings.xml`.

Figure 16-5 Directory for storing the settings.xml file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply**, and then click **OK**.

----End

16.2.4 Configuring the Python Sample Project

Scenario

To run the Python interface example codes of the Hive component of MRS, perform the following operations.

Procedure

- Step 1** Python of 2.6.6 or a higher version has been installed on a client. The Python version cannot be higher than 2.7.13.

The Python version can be viewed by running the **python** command on the command-line interface (CLI) of the client. The following information indicates that the Python version is 2.6.6.

```
Python 2.6.6 (r266:84292, Oct 12 2012, 14:23:48)
[GCC 4.4.6 20120305 (Red Hat 4.4.6-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

- Step 2** Setuptools of 5.0 or a higher version has been installed on a client. The setuptools version cannot be higher than 36.8.0.

To obtain the software, visit the official website.

<https://pypi.org/project/setuptools/#files>

Copy the downloaded setuptools package to the client, decompress the package, go to the decompressed directory, and run the **python setup.py install** command in the CLI of the client.

The following information indicates that `setuptools` 5.7 is installed successfully.

```
Finished processing dependencies for setuptools==5.7
```

Step 3 Install Python on the client.

1. Obtain the sample project folder **python-examples** in the `src\hive-examples` directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python-examples** folder.
3. Go to the **python-examples** folder.
4. Run the **python setup.py install** command in the CLI.

The following information indicates that Python is installed successfully.

```
Finished processing dependencies for pyhs2==0.5.0
```

Step 4 After the installation is successful, the following files are generated. **python-examples/pyCLI_sec.py** is the Python client example codes. **python-examples/pyhs2/haconnection.py** is the Python client API. Run `hive_python_client` scripts to execute the SQL functions, for example, **sh hive_python_client 'show tables'**.

 **NOTE**

This function applies to only simple SQL statements and depends on the ZooKeeper client.

----End

16.2.5 Configuring the Python3 Sample Project

Scenario

To run the Python3 interface example codes of the Hive component of MRS, perform the following operations.

Procedure

Step 1 Python3 of 3.6 or a higher version has been installed on a client. The Python3 version cannot be higher than 3.8.

The Python version can be viewed by running the **python3** command on the command-line interface (CLI) of the client. The following information indicates that the Python version is 3.8.2.

```
Python 3.8.2 (default, Jun 23 2020, 10:26:03)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

Step 2 `setuptools` of 47.3.1 version has been installed on a client.

To obtain the software, visit the official website.

<https://pypi.org/project/setuptools/#files>

Copy the downloaded `setuptools` package to the client, decompress the package, go to the decompressed directory, and run the **python3 setup.py install** command in the CLI of the client.

The following information indicates that `setuptools` 47.3.1 is installed successfully.

```
Finished processing dependencies for setuptools==47.3.1
```

 NOTE

If the system displays a message indicating that the installation of `setuptools` of 47.3.1 fails, check whether the environment is faulty or whether the problem is caused by Python.

Step 3 Install Python on the client.

1. Obtain the sample project folder **python3-examples** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python3-examples** folder.
3. Go to the **dependency_python3.6**, **dependency_python3.7**, or **dependency_python3.8** folder based on the Python3 version.
4. Run the **whereis easy_install** command to find the path of the **easy_install** program. If there are multiple paths, run the **easy_install --version** command to select the **easy_install** path corresponding to the **setuptools** version, for example, **/usr/local/bin/easy_install**.
5. Run the **easy_install** command to install the egg files in the **dependency_python3.x** folder. If the egg file has dependencies, you can use wildcards to install the egg file. For example:

– **dependency_python3.6** directory:

```
/usr/local/bin/easy_install future*egg six*egg python*egg sasl-*linux-$(uname -p).egg thrift-*egg thrift_sasl*egg
```

– **dependency_python3.7** directory:

```
/usr/local/bin/easy_install future*egg six*egg sasl-*linux-$(uname -p).egg thrift-*egg thrift_sasl*egg
```

– **dependency_python3.8** directory:

```
/usr/local/bin/easy_install future*egg six*egg python*egg sasl-*linux-$(uname -p).egg thrift-*linux-$(uname -p).egg thrift_sasl*egg
```

If the following information is displayed for each egg file, the installation is successful:

```
Finished processing dependencies for ***
```

- Step 4** After the installation is successful, the following files are generated. **python3-examples/pyCLI_sec.py** is the Python client example codes. **python3-examples/pyhive/hive.py** is the Python client API.

----End

16.3 Developing an Application

16.3.1 Typical Scenario Description

Scenarios

Assume that you need to develop a Hive data analysis application to manage the employee information described in [Table 16-8](#) and [Table 16-9](#).

Development Guidelines

Step 1 Prepare data.

1. Create three tables: employee information table **employees_info**, employee contact table **employees_contact**, and extended employee information table **employees_info_extended**.
 - Employee information table **employees_info** contains fields such as employee ID, name, salary currency, salary, tax category, work place, and hire date. In salary currency, **R** indicates RMB and **D** indicates USD.
 - Fields in the **employees_contact** table include the employee ID, phone number, and email address.
 - Fields in the **employees_info_extended** table include the employee ID, name, mobile phone number, e-mail address, salary currency, salary, tax category, and work place. The partition field is the hire date.

For table creation codes, see [Creating a Table](#).

2. Load employee information to **employees_info**.

For data loading codes, see [Loading Data](#).

[Table 16-8](#) describes employee information.

Table 16-8 Employee information

ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hiring Date
1	Wang	R	8000.01	personal income tax&0.05	Country 1:City1	2014
3	Tom	D	12000.02	personal income tax&0.09	Country 2:City2	2014
4	Jack	D	24000.03	personal income tax&0.09	Country 3:City3	2014
6	Linda	D	36000.04	personal income tax&0.09	Country 4:City4	2014
8	Zhang	R	9000.05	personal income tax&0.05	Country 5:City5	2014

3. Load employee contact information to **employees_contact**.

[Table 16-9](#) describes employee contact information.

Table 16-9 Employee contact information

ID	Mobile Phone Number	E-mail Address
1	135 XXXX XXXX	xxxx@xx.com
3	159 XXXX XXXX	xxxxx@xx.com.cn
4	186 XXXX XXXX	xxxx@xx.org
6	189 XXXX XXXX	xxxx@xxx.cn
8	134 XXXX XXXX	xxxx@xxxx.cn

4. Load extended employee information to **employees_info_extended**.

Table 16-10 describes the extended employee information.

Table 16-10 Extended employee information

ID	Name	Mobile Phone Number	E-mail Addresses	Salary Currency	Salary	Tax Category	Work Place	Hiring Date
1	Wang	135 XXXX XXXX	xxxx@xx.com	R	8000.01	personal income tax&0.05	Country1:City1	2014
3	Tom	159 XXXX XXXX	xxxxx@xx.com.cn	D	1200.02	personal income tax&0.09	Country2:City2	2014
4	Jack	186 XXXX XXXX	xxxx@xx.org	D	2400.03	personal income tax&0.09	Country3:City3	2014
6	Linda	189 XXXX XXXX	xxxx@xx.cn	D	3600.04	personal income tax&0.09	Country4:City4	2014
8	Zhang	134 XXXX XXXX	xxxx@xxx.cn	R	9000.05	personal income tax&0.05	Country5:City5	2014

Step 2 Analyze data.

For data analysis codes, see [Querying Data](#).

- Query contact information of employees whose salaries are paid in USD.

- Query the IDs and names of employees who were hired in 2014, and load the query results to the partition with the hiring date of 2014 in **employees_info_extended**.
- Collect the number of records in the **employees_info** table.
- Query information about employees whose email addresses end with "cn".

Step 3 Submit a data analysis task to collect the number of records in the **employees_info** table. For details about the implementation, see [Example Program Guide](#).

----End

16.3.2 Example Codes

16.3.2.1 Creating a Table

Function

This section describes how to use Hive Query Language (HQL) to create internal and external tables. You can create a table in three modes:

- Customize the table structure, and use the key word **EXTERNAL** to differentiate between internal and external tables.
 - If data is to be processed by Hive, create an internal table. When an internal table is deleted, the metadata and data in the table are deleted together.
 - If data is to be processed by multiple tools (such as Pig), create an external table. When an external table is deleted, only metadata is deleted.
- Create a table based on existing tables. Use **CREATE LIKE** to fully copy the original table structure, including the storage format.
- Create a table based on query results using **CREATE AS SELECT**.

In this mode, you can specify fields (except for the storage format) that you want to copy to the new table when copying the structure of the existing table.

NOTE

- To perform the following operations on a cluster enabled with the security service, you must have the **CREATE** permission for databases. To create a table using the **CREATE AS SELECT** statement, you must have the **SELECT** permission for tables. For details about the permission requirements, see [Overview](#).
- Both the table name and field name can contain a maximum of 128 bytes. Both the field comment and value can contain a maximum of 4,000 bytes. The key in **WITH SERDEPROPERTIES** can contain a maximum of 256 bytes.

Sample Codes

```
-- Create an external table employees_info.
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info
(
  id INT,
  name STRING,
  usd_flag STRING,
```

```
salary DOUBLE,
deductions MAP<STRING, DOUBLE>,
address STRING,
entrytime STRING
)
-- Specify the field delimiter.
-- "delimited fields terminated by" indicates that the column delimiter is ',' and "MAP KEYS TERMINATED
BY" indicates that the delimiter of key values in the MAP is '&'.
ROW FORMAT delimited fields terminated by ','; MAP KEYS TERMINATED BY '&'
-- Set the storage format to TEXTFILE.
STORED AS TEXTFILE;

-- Create an external table employees_contact.
CREATE EXTERNAL TABLE IF NOT EXISTS employees_contact
(
  id INT,
  tel_phone STRING,
  email STRING
)
ROW FORMAT delimited fields terminated by ','
STORED AS TEXTFILE;

-- Create an external table employees_info_extended.
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info_extended(id INT, name STRING, usd_flag STRING,
salary DOUBLE, deductions MAP<STRING, DOUBLE>, address STRING)
-- A table may have one or multiple partitions. Each partition is saved as an independent folder in the table
directory. Partitions help minimize the query scope, accelerate data query, and allow users to manage data
based on certain criteria.
-- Use PARTITIONED BY to specify the column name and data type of the partition.
PARTITIONED BY(entrytime STRING)
ROW FORMAT delimited fields terminated by ','; MAP KEYS TERMINATED BY '&'
STORED AS TEXTFILE;
-- After a table is created, you can use ALTER TABLE to add or delete fields to or from the table, modify
table attributes, and add partitions.
-- Add the tel_phone and email fields to the employees_info_extended table.
ALTER TABLE employees_info_extended ADD COLUMNS (tel_phone STRING, email STRING);
```

16.3.2.2 Loading Data

Function

This section describes how to use HQL to load data to the existing **employees_info** table. You can learn how to load data from a local file system and MRS cluster. LOCAL is used to differentiate between local and non-local data sources.

NOTE

To perform the following operations on a cluster enabled with the security service, you must have the UPDATE permission for databases, and owner permission and read/write permission for files to be loaded. For details about permission requirements, see [Overview](#).

If LOCAL is used in data loading statements, data is loaded from a local directory. In addition to the UPDATE permission for tables, you must have the read permission for the data path. It is also required that the data can be accessed by user **omm** on the active HiveServer.

If OVERWRITE is used in data loading statements, the existing data in a table will be overwritten by new data. If OVERWRITE does not exist, data will be added to the table.

Example Codes

```
-- Load the employee_info.txt file from the /opt/hive_examples_data/ directory of the local file system to
the employees_info table.
---- Overwrite the original data using the new data.
```

```
LOAD DATA LOCAL INPATH '/opt/hive_examples_data/employee_info.txt' OVERWRITE INTO TABLE
employees_info;
---- Retain the original data and add the new data to the table.
LOAD DATA LOCAL INPATH '/opt/hive_examples_data/employee_info.txt' INTO TABLE employees_info;

-- Load /user/hive_examples_data/employee_info.txt from HDFS to the employees_info table.
---- Overwrite the original data using the new data.
LOAD DATA INPATH '/user/hive_examples_data/employee_info.txt' OVERWRITE INTO TABLE
employees_info;
---- Retain the original data and add the new data to the table.
LOAD DATA INPATH '/user/hive_examples_data/employee_info.txt' INTO TABLE employees_info;
```

NOTE

The essence of loading data is to copy the data to the specified table directory in HDFS.

Sample Data

Data in **employees_info** is as follows:

```
1,Wang,R,8000.01,person&personal^Btype&income^Btax&0.05,Country1:City1,2014
3,Tom,D,12000.02,person&personal^Btype&income^Btax&0.09,Country2:City2,2014
4,Jack,D,24000.03,person&personal^Btype&income^Btax&0.05,Country3:City3,2014
6,Linda,D,36000.04,person&personal^Btype&income^Btax&0.05,Country4:City4,2014
8,Zhang,R,9000.05,person&personal^Btype&income^Btax&0.05,Country5:City5,2014
```

Data in **employees_contact** is as follows:

```
1,135 XXXX XXXX,xxxx@xx.com
3,159 XXXX XXXX,xxxxx@xx.com.cn
4,186 XXXX XXXX,xxxx@xx.org
6,189 XXXX XXXX,xxxx@xxx.cn
8,134 XXXX XXXX,xxxx@xxxx.cn
```

Data in **employees_info_extended** is as follows:

```
1,Wang,135 XXXX XXXX,xxxx@xx.com,R,
8000.01,person&personal^Btype&income^Btax&0.05,Country1:City1,2014
3,Tom,159 XXXX XXXX,xxxxx@xx.com.cn,D,
12000.02,person&personal^Btype&income^Btax&0.09,Country2:City2,2014
4,Jack,186 XXXX XXXX,xxxx@xx.org,D,
24000.03,person&personal^Btype&income^Btax&0.05,Country3:City3,2014
6,Linda,189 XXXX XXXX,xxxx@xxx.cn,D,
36000.04,person&personal^Btype&income^Btax&0.05,Country4:City4,2014
8,Zhang,134 XXXX XXXX,xxxx@xxxx.cn,R,
9000.05,person&personal^Btype&income^Btax&0.05,Country5:City5,2014
```

16.3.2.3 Querying Data

Function

This topic describes how to use Hive Query Language (HQL) to query and analyze data. You can query and analyze data using the following methods:

- Use common features for SELECT query, such as JOIN.
- Load data to a specified partition.
- Use Hive-provided functions.
- Query and analyze data using user-defined functions (UDFs). For details about how to create and define UDFs, see [UDF](#).

 NOTE

To perform the following operations on a cluster enabled with the security service, you must have related permissions for tables. For details about permission requirements, see [Overview](#).

Example Codes

```
-- Query the contact information of employees whose salaries are paid in USD.
SELECT
a.name,
b.tel_phone,
b.email
FROM employees_info a JOIN employees_contact b ON(a.id = b.id) WHERE usd_flag='D';

-- Query the IDs and names of employees who were hired in 2014, and load query results to the partition
with the hiring time of 2014 in the employees_info_extended table.
INSERT OVERWRITE TABLE employees_info_extended PARTITION (entrytime = '2014')
SELECT
a.id,
a.name,
a.usd_flag,
a.salary,
a.deductions,
a.address,
b.tel_phone,
b.email
FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE a.entrytime = '2014';

-- Use the existing Hive function COUNT() to count the number of records in the employees_info table.
SELECT COUNT(*) FROM employees_info;

-- Query information about employees whose email addresses end with "cn".
SELECT a.name, b.tel_phone FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE
b.email like '%cn';
```

Extensions

- Configure intermediate Hive data encryption.
Set the table format to RCFile (recommended) or SequenceFile, and the encryption algorithm to ARC4Codec. SequenceFile is a unique Hadoop file format, and RCFile is a Hive file format with optimized column storage. When a big table is queried, RCFile provides higher performance than SequenceFile.

```
set hive.exec.compress.output=true;
set hive.exec.compress.intermediate=true;
set hive.intermediate.compression.codec=org.apache.hadoop.io.encrypted.arc4.ARC4Codec;
```
- For details about UDFs, see [UDF](#).

16.3.2.4 UDF

When internal functions of Hive cannot meet requirements, you can compile user-defined functions (UDFs) and use them for query.

According to implementation methods, UDFs are classified as follows:

- Common UDFs: used to perform operations on a single data row and export a single data row.
- User-defined aggregating functions (UDAFs): used to input multiple data rows and export a single data row.
- User-defined table-generating functions (UDTFs): used to perform operations on a single data row and export multiple data rows.

According to usage methods, UDFs are classified as follows:

- Temporary functions: used only in the current session and must be recreated after a session restarts.
- Permanent function: used in multiple sessions and do not have to be created every time a session restarts.

The following uses AddDoublesUDF as an example to describe how to compile and use UDFs.

Function

AddDoublesUDF is used to add two or multiple floating point values. The following example describes how to compile and use UDFs.

NOTE

- The normal UDF must be originated from **org.apache.hadoop.hive.ql.exec.UDF**.
- The normal UDF must implement at least one evaluate(). The evaluate function supports overloading.
- To develop a customized function, you need to add the hive-exec-3.1.0.jar dependency package to the project. The package can be obtained from the Hive installation directory.

Example Codes

The following is a UDF example:

```
package com.huawei.bigdata.hive.example.udf;
import org.apache.hadoop.hive.ql.exec.UDF;

public class AddDoublesUDF extends UDF {
    public Double evaluate(Double... a) {
        Double total = 0.0;
        // Processing logic
        for (int i = 0; i < a.length; i++)
            if (a[i] != null)
                total += a[i];
        return total;
    }
}
```

How to Use

- Step 1** Package the preceding program into **AddDoublesUDF.jar**, and upload it to a directory on the HDFS (such as **/user/hive_examples_jars/**). The user who creates the UDF and the user who uses the UDF function must have read right on this JAR file. Example statements:

```
hdfs dfs -put ./hive_examples_jars /user/hive_examples_jars
```

```
hdfs dfs -chmod 777 /user/hive_examples_jars
```

- Step 2** Use a user with admin rights to log in to the Beeline client and run the following command:

```
kinit Hive service user
```

```
beeline
```

set role admin;

Step 3 Define the function in Hive Server. Run the following SQL statement to create a permanent function:

```
CREATE FUNCTION addDoubles AS  
'com.huawei.bigdata.hive.example.udf.AddDoublesUDF' using jar 'hdfs://  
hacluster/user/hive_examples_jars/AddDoublesUDF.jar';
```

addDoubles indicates the function alias that is used for SELECT query.

Run the following statement to create a temporary function:

```
CREATE TEMPORARY FUNCTION addDoubles AS  
'com.huawei.bigdata.hive.example.udf.AddDoublesUDF' using jar 'hdfs://  
hacluster/user/hive_examples_jars/AddDoublesUDF.jar';
```

- *addDoubles* indicates the function alias that is used for SELECT query.
- TEMPORARY indicates that the function is used only in the current session with the Hive server.

Step 4 Run the following SQL statement to use the function in the Hive server:

```
SELECT addDoubles(1,2,3);
```

 **NOTE**

*If an [Error 10011] error is displayed when you log in to the client again, run the **reload function;** command and then use this function.*

Step 5 Run the following SQL statement to delete the function from the Hive server:

```
DROP FUNCTION addDoubles;
```

----End

Extensions

None

16.3.2.5 Example Program Guide

Function

This section describes how to use an example program to complete an analysis task. An example program can submit a task by using the following methods:

- Submitting a data analysis task by using JDBC interfaces
- Submitting a data analysis task by using Python

Example Codes

- Submit a data analysis task using the Hive Java database connectivity (JDBC) interface, that is, JDBCExample.java.
 - a. Read the **property** file of the HiveServer client. The **hiveclient.properties** file is saved in the **resources** directory of the JDBC example program provided by Hive.

```

Properties clientInfo = null;
String userdir = System.getProperty("user.dir") + File.separator
+ "conf" + File.separator;
InputStream fileInputStream = null;
try{
clientInfo = new Properties();
//hiveclient.properties indicates the client configuration file. If the multiple-service feature is
used, the file must be replaced with the hiveclient.properties file on the instance client.
//hiveclient.properties is located under the config directory of the directory where the instance
client installation package is decompressed.
String hiveclientProp = userdir + "hiveclient.properties" ;
File propertiesFile = new File(hiveclientProp);
fileInputStream = new FileInputStream(propertiesFile);
clientInfo.load(fileInputStream);
}catch (Exception e) {
throw new IOException(e);
}finally{
if(fileInputStream != null){
fileInputStream.close();
fileInputStream = null;
}
}
}

```

- b. Obtain the IP address and port list of ZooKeeper, the cluster authentication mode, the SASL configuration of HiveServers, node names of HiveServers in ZooKeeper, the discovery mode from the client to the server, and the principal server process for user authentication. You can read all these configurations from the **hiveclient.properties** file.

```

//The format of zkQuorum is xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181";
//xxx.xxx.xxx.xxx is the IP address of the node where ZooKeeper resides. The default port is
24002.

```

```

zkQuorum = clientInfo.getProperty("zk.quorum");
auth = clientInfo.getProperty("auth");
sasL_qop = clientInfo.getProperty("sasL.qop");
zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
principal = clientInfo.getProperty("principal");

```

- c. In security mode, the kerberos user and keytab file path are required for login authentication. For details about how to obtain **USER_NAME**, **USER_KEYTAB_FILE**, and **KRB5_FILE**, see [Running JDBC and Viewing Results](#).

```

// Set the userName of new user.
USER_NAME = "xxx";
// Set the keytab and krb5 files location of client.
String userdir = System.getProperty("user.dir") + File.separator
+ "conf" + File.separator;
USER_KEYTAB_FILE = userdir + "user.keytab";
KRB5_FILE = userdir + "krb5.conf";

```

- d. Define HQL. HQL must be a single statement and cannot contain ";".

```

// Define HQL. HQL cannot contain ";"
String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT,name STRING)",
"SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};

```

- e. Build JDBC URL.

 NOTE

You can also implement pre-authentication without the need of providing the account and keytab file path. For details, see JDBC code example 2 in [Examples](#). If IBM JDK is used to run Hive applications, pre-authentication in JDBC sample code 2 must be implemented.

The following is an example of the JDBC URL composed of code snippets:

```
jdbc:hive2://xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181;/serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2;sasl.qop=auth-conf;auth=KERBEROS;principal=hive/hadoop.<system domain name>@<system domain name>;
```

```
// Concat JDBC URL
StringBuilder sBuilder = new StringBuilder(
    "jdbc:hive2://").append(zkQuorum).append("/");

if ("KERBEROS".equalsIgnoreCase(auth)) {
    sBuilder.append(";serviceDiscoveryMode=")
        .append(serviceDiscoveryMode)
        .append(";zooKeeperNamespace=")
        .append(zooKeeperNamespace)
        .append(";sasl.qop=")
        .append(sasl_qop)
        .append(";auth=")
        .append(auth)
        .append(";principal=")
        .append(principal)
        .append(";user.principal=")
        .append(USER_NAME)
        .append(";user.keytab=")
        .append(USER_KEYTAB_FILE)
        .append(";");
} else {
    // Normal mode
    sBuilder.append(";serviceDiscoveryMode=")
        .append(serviceDiscoveryMode)
        .append(";zooKeeperNamespace=")
        .append(zooKeeperNamespace)
        .append(";auth=none;");
}
String url = sBuilder.toString();
```

- f. Load the Hive JDBC driver.

```
// Load the Hive JDBC driver.
Class.forName(HIVE_DRIVER);
```

- g. Obtain the JDBC connection, confirm the HQL type (DDL/DML), call ports to run the HQL statement, return the queried column name and results to the console, and close the JDBC connection.

```
Connection connection = null;
try {
    // Obtain the JDBC connection.
    // If the normal mode is used, the second parameter needs to be set to a correct username.
    // Otherwise, the anonymous user will be used for login.
    connection = DriverManager.getConnection(url, "", "");

    // Create a table
    // To import data to a table after the table is created, you can use the LOAD statement. For
    // example, import data from the HDFS to the table.
    //load data inpath '/tmp/employees.txt' overwrite into table employees_info;
    execDDL(connection,sqls[0]);
    System.out.println("Create table success!");

    // Query
    execDML(connection,sqls[1]);
```

```

// Delete the table
execDDL(connection,sqls[2]);
System.out.println("Delete table success!");
}
finally {
// Close the JDBC connection.
if (null != connection) {
connection.close();
}
}

public static void execDDL(Connection connection, String sql)
throws SQLException {
PreparedStatement statement = null;
try {
statement = connection.prepareStatement(sql);
statement.execute();
}
finally {
if (null != statement) {
statement.close();
}
}
}

public static void execDML(Connection connection, String sql) throws SQLException {
PreparedStatement statement = null;
ResultSet resultSet = null;
ResultSetMetaData resultMetaData = null;

try {
// Run the HQL statement.
statement = connection.prepareStatement(sql);
resultSet = statement.executeQuery();

// Return the queried column name to the console.
resultMetaData = resultSet.getMetaData();
int columnCount = resultMetaData.getColumnCount();
for (int i = 1; i <= columnCount; i++) {
System.out.print(resultMetaData.getColumnLabel(i) + '\t');
}
System.out.println();

// Return the results to the console.
while (resultSet.next()) {
for (int i = 1; i <= columnCount; i++) {
System.out.print(resultSet.getString(i) + '\t');
}
System.out.println();
}
}
finally {
if (null != resultSet) {
resultSet.close();
}

if (null != statement) {
statement.close();
}
}
}
}

```

- Submit a data analysis task using the Python interface, that is, **python-examples/pyCLI_sec.py**. The authentication mode of the cluster to which the example program connects is the secure mode. Before running the example program, run the kinit command to authenticate the kerberos user with related rights.

- a. Import the HAConnection class.

```
from pyhs2.haconnection import HAConnection
```

- b. Declare the HiveServer IP address list. In this example, **hosts** indicate the nodes of HiveServer, and **xxx.xxx.xxx.xxx** indicates the service IP address.

```
hosts = ["xxx.xxx.xxx.xxx", "xxx.xxx.xxx.xxx"]
```

 **NOTE**

1. If the HiveServer instance is migrated, the original sample program is invalid. You need to update the IP address of the HiveServer after the migration of the HiveServer instance used in the sample program.
2. If multiple Hive instances are used, update the IP address based on the address of the instance that is actually connected.

- c. Configure the kerberos host name and service name. In this example, the kerberos host name is hadoop and service name is hive.

```
conf = {"krb_host": "hadoop.<system domain name>", "krb_service": "hive"}
```

 **NOTE**

If multiple Hive instances are used, change the service name based on the cluster that is actually connected. For example, if the Hive1 instance is connected, change the service name to hive1.

- d. Create a connection, run the HQL statement, and return the queried column name and results to the console.

```
try:
    with HAConnection(hosts = hosts,
                      port = 21066,
                      authMechanism = "KERBEROS",
                      configuration = conf) as haConn:
        with haConn.getConnection() as conn:
            with conn.cursor() as cur:
                # show databases
                print cur.getdatabases()

                # execute query
                cur.execute("show tables")

                # return column info from query
                print cur.getschema()

                # fetch table results
                for i in cur.fetch():
                    print i

except exception, e:
    print e
```

 **NOTE**

If multiple Hive instances are used, you need to modify hosts according to the description in **b** and change the port number based on to the actual port number. The default ports of Hive to Hive4 are 21066 to 21070, respectively.

16.3.2.6 Accessing Multiple ZooKeepers

Description

This section describes how to access both FusionInsight ZooKeeper and the third-party ZooKeeper in the same client process using the `testConnectHive` and `testConnectApacheZK` methods respectively.

In the `JDBCExample` class of the `hive-jdbc-example-multizk` package, the code structure of the main method is as follows:

```
public static void main(String[] args) throws InstantiationException, IllegalAccessException,
ClassNotFoundException, SQLException, IOException{
    testConnectHive();// Method of accessing FusionInsight ZooKeeper
    testConnectApacheZk();// Method of accessing the open-source ZooKeeper
}
```

Accessing FusionInsight ZooKeeper

If only the method of accessing FusionInsight ZooKeeper needs to be executed, comment out the `testConnectApacheZk` method in the `main` function.

Before using the `testConnectHive` method to access FusionInsight ZooKeeper, perform the following operations:

- Step 1** Change the value of `USER_NAME` in the `init` method in `JDBCExample`. `USER_NAME` indicates the user that is used to access FusionInsight ZooKeeper and has permissions of the FusionInsight Hive and Hadoop common user groups.
- Step 2** Go to the client decompression path `FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles\Hive\config` and manually import the `core-site.xml` and `hiveclient.properties` files to the `hive-jdbc-example-multizk\src\main\resources` directory of the sample project.
- Step 3** Download the `krb5.conf` and `user.keytab` files of the user to the `resources` directory in the `hive-jdbc-example-multizk` package.
- Step 4** Check and change the values of `zk.port` and `zk.quorum` in the `hiveclient.properties` file in the `resources` directory.
 - **zk.port**: indicates the port for accessing FusionInsight ZooKeeper. Generally, the default value is used. Change the value as required.
 - **zk.quorum**: indicates the IP address for accessing ZooKeeper quorumpeer. Set it to the IP address of the cluster deployed with the FusionInsight ZooKeeper service.

----End

Accessing Open-Source ZooKeeper

To use `testConnectApacheZk` to connect to the open-source ZooKeeper code, change `xxx.xxx.xxx.xxx` in the following code to the IP address of the open-source ZooKeeper to be connected. Change the port number as required. If only the sample for accessing the third-party ZooKeeper needs to be executed, comment out the `testConnectHive` method in the `main` function.

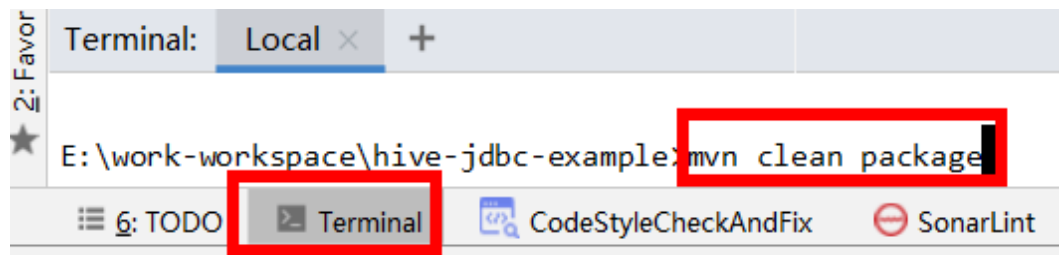
```
digestZK = new org.apache.zookeeper.ZooKeeper("xxx.xxx.xxx.xxx:2181", 60000, null);
```

16.4 Commissioning Applications

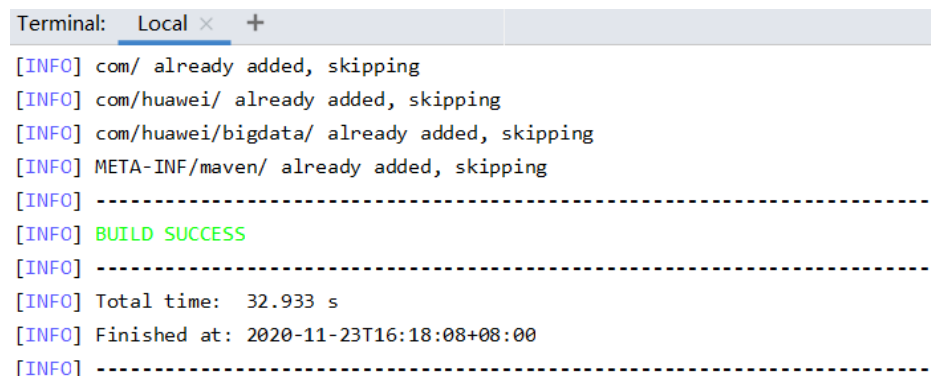
16.4.1 Running JDBC and Viewing Results

Running JDBC in CLI Mode

- Step 1** In the lower left corner of the IDEA page, click **Terminal** to access the terminal. Run the **mvn clean package** command to compile the package.



If **BUILD SUCCESS** is displayed, the compilation is successful, as shown in the following figure. A JAR file containing the **-with-dependencies** field is generated in the **target** directory of the sample project.



- Step 2** Create a directory on Windows or Linux as the running directory, for example, **D:\jdbc_example** (Windows) or **/opt/jdbc_example** (Linux). Place the JAR file whose name contains **-with-dependencies** in the **target** directory generated in **Step 1** to this directory. Create the **src/main/resources** subdirectory in the directory. Copy all files in the **resources** directory of the **jdbc-examples** project to the **resources** directory.

- Step 3** In Windows, run the following command:

```
cd /d d:\jdbc_example
```

```
java -jar hive-jdbc-example-1.0-SNAPSHOT-jar-with-dependencies.jar
```

In Linux, run the following command:

```
chmod +x /opt/jdbc_example -R
```

```
cd /opt/jdbc_example
```

```
java -jar hive-jdbc-example-1.0-SNAPSHOT-jar-with-dependencies.jar
```


 NOTE

The preceding JAR file names are for reference only. The actual names may vary.

Step 4 In the CLI, view the HQL query results in the example codes.

The following information is displayed if the sample project is successful in Windows:

```
Create table success!  
_c0  
0  
Delete table success!
```

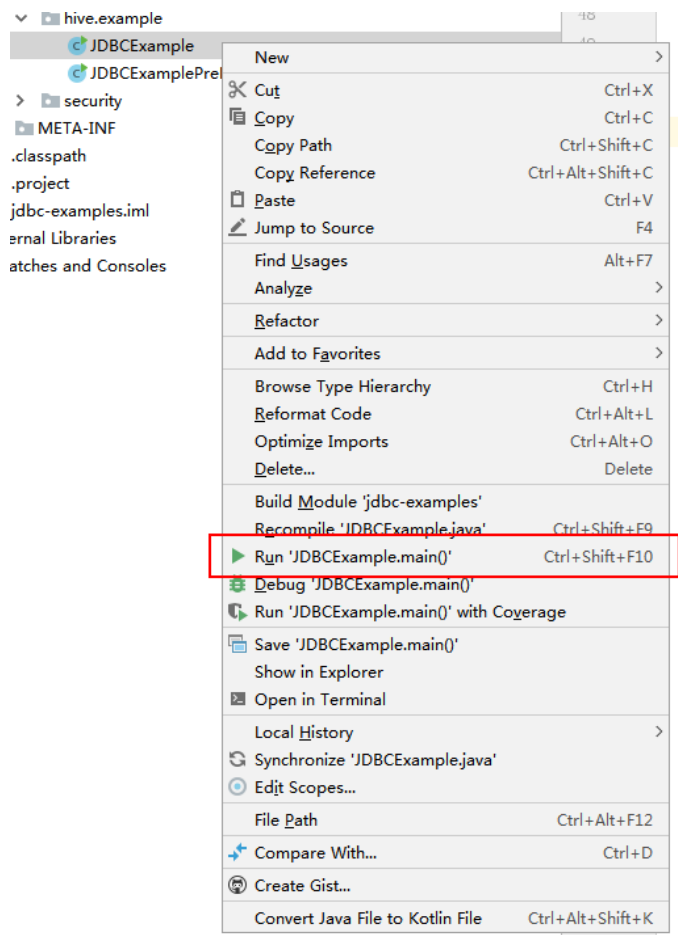
The following information is displayed if the sample project is successful in Linux:

```
Create table success!  
_c0  
0  
Delete table success!
```

----End

Running JDBC in IntelliJ IDEA Mode

Step 1 Right-click the JDBCExample class in the IntelliJ IDEA jdbc-examples project, and choose **Run JDBCExample.main()** from the shortcut menu. As shown in the following figure.



Step 2 In the IntelliJ IDEA output window, view the HQL query results in the example codes.

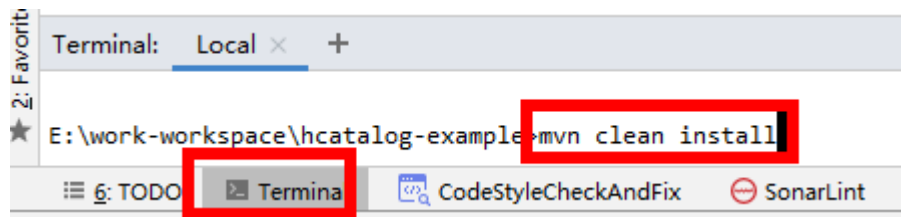
```
Create table success!  
_c0  
0  
Delete table success!
```

----End

16.4.2 Running HCatalog and Viewing Results

Running HCatalog Example Projects

- Step 1** In the lower left corner of the IDEA page, click **Terminal** to access the terminal. Run the **mvn clean install** command to compile the package.



If **BUILD SUCCESS** is displayed, the compilation is successful, as shown in the following figure. The **hcatalog-example-*.jar** package is generated in the **target** directory of the sample project.

```
Terminal: Local x +  
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ hcatalog-example ---  
[INFO] Building jar: E:\other-workspase\sample_project\src\hive-examples\hcata:  
[INFO]  
[INFO] --- maven-install-plugin:2.4:install (default-install) @ hcatalog-exi  
[INFO] Installing E:\other-workspase\sample_project\src\hive-examples\hcata:  
[INFO] Installing E:\other-workspase\sample_project\src\hive-examples\hcata:  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 4.916 s  
[INFO] Finished at: 2020-11-23T16:25:57+08:00  
[INFO] -----
```

NOTE

The preceding JAR file names are for reference only. The actual names may vary.

- Step 2** Upload the **hcatalog-example-*.jar** file generated in the **target** directory in the previous step to the specified directory on Linux, for example, **/opt/hive_client**, marked as **\$HCAT_CLIENT**, and ensure that the Hive and YARN clients have been installed. Execute environment variables for the HCAT_CLIENT to take effect.

```
export HCAT_CLIENT=/opt/hive_client
```

- Step 3** Run the following command to configure environment parameters (client installation path **/opt/client** is used as an example):

```
export HADOOP_HOME=/opt/client/HDFS/hadoop  
export HIVE_HOME=/opt/client/Hive/Beeline  
export HCAT_HOME=$HIVE_HOME/../HCatalog
```

```
export LIB_JARS=$HCAT_HOME/lib/hive-hcatalog-core-xxx.jar,$HCAT_HOME/lib/hive-metastore-xxx.jar,
$HCAT_HOME/lib/hive-standalone-metastore-xxx.jar,$HIVE_HOME/lib/hive-exec-xxx.jar,$HCAT_HOME/lib/
libfb303-xxx.jar,$HCAT_HOME/lib/slf4j-api-xxx.jar,$HCAT_HOME/lib/jdo-api-xxx.jar,$HCAT_HOME/lib/antlr-
runtime-xxx.jar,$HCAT_HOME/lib/datanucleus-api-jdo-xxx.jar,$HCAT_HOME/lib/datanucleus-core-xxx.jar,
$HCAT_HOME/lib/datanucleus-rdbms-fi-xxx.jar,$HCAT_HOME/lib/log4j-api-xxx.jar,$HCAT_HOME/lib/log4j-
core-xxx.jar,$HIVE_HOME/lib/commons-lang-xxx.jar
export HADOOP_CLASSPATH=$HCAT_HOME/lib/hive-hcatalog-core-xxx.jar:$HCAT_HOME/lib/hive-
metastore-xxx.jar:$HCAT_HOME/lib/hive-standalone-metastore-xxx.jar:$HIVE_HOME/lib/hive-exec-xxx.jar:
$HCAT_HOME/lib/libfb303-xxx.jar:$HADOOP_HOME/etc/hadoop:$HCAT_HOME/conf:$HCAT_HOME/lib/
slf4j-api-xxx.jar:$HCAT_HOME/lib/jdo-api-xxx.jar:$HCAT_HOME/lib/antlr-runtime-xxx.jar:$HCAT_HOME/lib/
datanucleus-api-jdo-xxx.jar:$HCAT_HOME/lib/datanucleus-core-xxx.jar:$HCAT_HOME/lib/datanucleus-
rdbms-fi-xxx.jar:$HCAT_HOME/lib/log4j-api-xxx.jar:$HCAT_HOME/lib/log4j-core-xxx.jar:$HIVE_HOME/lib/
commons-lang-xxx.jar
```

NOTE

- xxx: Indicates the version number of the JAR package. **Change the version numbers of the JAR files specified in LIB_JARS and HADOOP_CLASSPATH based on the actual environment.**
- If the multi-instance function is enabled for Hive, perform the configuration in **export HIVE_HOME=/opt/client/Hive/Beeline**. For example, if Hive 1 is used, ensure that the Hive 1 client has been installed before using it. Change the value of **export HIVE_HOME** to **/opt/client/Hive1/Beeline**.

Step 4 Prepare for the running:

1. Use the Hive client to create source table t1 in beeline: **create table t1(col1 int);**

Insert the following data into t1:

```
+-----+--+
| t1.col1 |
+-----+--+
| 1      |
| 1      |
| 1      |
| 2      |
| 2      |
| 3      |
```

2. Create destination table t2: **create table t2(col1 int,col2 int);**

NOTE

Tables created in this sample project use the default storage format of Hive. Currently, tables whose storage format is ORC are not supported.

Step 5 Use the Yarn client to submit tasks:

```
yarn --config $HADOOP_HOME/etc/hadoop jar $HCAT_CLIENT/hcatalog-
example-1.0-SNAPSHOT.jar com.huawei.bigdata.HCatalogExample -libjars
$LIB_JARS t1 t2
```

Step 6 View the running result. The data in t2 is as follows:

```
0: jdbc:hive2://192.168.1.18:2181,192.168.1.> select * from t2;
+-----+-----+--+
| t2.col1 | t2.col2 |
+-----+-----+--+
| 1      | 3      |
| 2      | 2      |
| 3      | 1      |
+-----+-----+--+
```

----End

16.4.3 Running Python and Viewing Results

Running Python in CLI Mode

Step 1 Grant the execution permission for the script of **python-examples** folder. Run the following command in the CLI:

```
chmod +x python-examples -R.
```

Step 2 Enter the service plane IP address of the node where HiveServer is installed in the hosts array of `python-examples/pyCLI_sec.py`.

NOTE

When the multi-instance is enabled: In `python-examples/pyCLI_sec.py`, the hosts array must be modified. In addition, the port must be set according to the used instance. The port (`hive.server2.thrift.port`) is used for Hive to provide the Thrift service.

Step 3 Change `hadoop.hadoop.com` in the `conf` array of `python-examples/pyCLI_sec.py` and `python-examples/pyline.py` to `hadoop.the actual domain name`. To view the actual domain name, log in to FusionInsight Manager and choose **System > Permission > Domain and Mutual Trust > Local Domain**.

Step 4 Run the **kinit** command to obtain the cache for Kerberos authentication.

Use the developer account created in [Preparing a Developer Account](#) to run the following commands to run the client program:

```
kinit -kt keytabstorage path username
```

```
cd python-examples
```

```
python pyCLI_sec.py
```

Step 5 In the CLI, view the HQL query results in the example codes. For example:

```
[['default', '']]  
[{'comment': 'from deserializer', 'columnName': 'tab_name', 'type': 'STRING_TYPE'}]  
['xx']
```

NOTE

If the following exception occurs:

```
importError: libsasl2.so.2: cannot open shared object file: No such file or directory
```

You can handle the problem as follows:

1. Run the following command to check the LibSASL version in the installed operating system.

```
ldconfig -p|grep sasl
```

If the following is displayed, the current operating system only has the 3.x version.

```
libsasl2.so.3 (libc6,x86-64) => /usr/lib64/libsasl2.so.3  
libsasl2.so.3 (libc6) => /usr/lib/libsasl2.so.3
```

2. If only the 3. x version exists, run the following command to create a soft link.

```
ln -s /usr/lib64/libsasl2.so.3.0.0 /usr/lib64/libsasl2.so.2
```

----End

16.4.4 Running Python3 and Viewing Results

Running Python in CLI Mode

Step 1 Grant the execution permission for the script of **python3-examples** folder. Run the following command in the CLI:

```
chmod +x python3-examples -R.
```

Step 2 In **python3-examples/pyCLI_nosec.py**, change the value of host to the service plane IP address of the node where HiveServer is installed, and change the value of port to the port (**hive.server2.thrift.port**) used by Hive to provide the Thrift service. The default value is 21066.

 **NOTE**

When the multi-instance is enabled: In **python3-examples/pyCLI_sec.py**, the hosts must be modified. In addition, the port must be set according to the used instance. The port (**hive.server2.thrift.port**) is used for Hive to provide the Thrift service.

Step 3 Change **hadoop.hadoop.com** in the conf of **python-examples/pyCLI_sec.py** to **hadoop.the actual domain name**. To view the actual domain name, log in to FusionInsight Manager and choose **System > Permission > Domain and Mutual Trust > Local Domain**.

Step 4 Run the **kinit** command to obtain the cache for Kerberos authentication.

Use the developer account created in [Preparing a Developer Account](#) to run the following commands to run the client program:

```
kinit -kt keytabstorage path username
```

```
cd python3-examples
```

```
python3 pyCLI_sec.py
```

Step 5 In the CLI, view the HQL query results in the example codes. For example:

```
('table_name1',)  
(table_name2',)  
(table_name3',)  
(table_name4',)  
(table_name5',)
```

In the preceding command, **table_nameX** indicates the actual table name.

----End

16.5 More Information

16.5.1 Interface Reference

16.5.1.1 JDBC

The Hive Java database connectivity (JDBC) interface complies with the Java JDBC driver standard.

 NOTE

As a data warehouse, Hive does not support all JDBC APIs. For example, if transactional operations, such as rollback and setAutoCommit, are performed, SQL exceptions like **Method not supported** will occur.

16.5.1.2 Hive SQL

Hive SQL supports all features of Hive-3.1.0. For details, see <https://cwiki.apache.org/confluence/display/hive/languagemanual>.

Table 16-11 describes the extended Hive statements provided by .

Table 16-11 Extended Hive statements

Extended Syntax	Syntax Description	Syntax Example	Example Description
<pre>CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_ name (col_name data_type [COMMENT col_comment], ...) [ROW FORMAT row_format] [STORED AS file_format] STORED BY 'storage.handler.cl ass.name' [WITH SERDEPROPERTIE S (...)] [TBLPROPERTIES ("groupId"=" group1 ","locatorId"="loc ator1")] ...;</pre>	<p>The statement is used to create a Hive table and specify locators on which table data files locate. For details, see Using HDFS Colocation to Store Hive Tables.</p>	<pre>CREATE TABLE tab1 (id INT, name STRING) row format delimited fields terminated by '\t' stored as RCFILE TBLPROPERTIES(" groupId"=" group1 ","locatorId"="loc ator1");</pre>	<p>The statement is used to create table tab1 and specify locator1 on which the table data of tab1 locates.</p>

Extended Syntax	Syntax Description	Syntax Example	Example Description
<pre>CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name (col_name data_type [COMMENT col_comment], ...) [ROW FORMAT row_format] [STORED AS file_format] STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)] ... [TBLPROPERTIES ('column.encode.columns'='col_name1,col_name2' 'column.encode.indices'='col_id1,col_id2', 'column.encode.classname'='encode_classname')]...;</pre>	<p>The statement is used to create a hive table and specify the table encryption column and encryption algorithm. For details, see Using the Hive Column Encryption.</p>	<pre>create table encode_test(id INT, name STRING, phone STRING, address STRING) ROW FORMAT SERDE 'org.apache.hadoop p.hive.serde2.lazy. LazySimpleSerDe' WITH SERDEPROPERTIES S ('column.encode.i ndices'='2,3', 'column.encode.cl assname'='org.apa che.hadoop.hive.s erde2.SMS4Rewrit er') STORED AS TEXTFILE;</pre>	<p>The statement is used to create table encode_test and specify that column 2 and column 3 will be encrypted using the org.apache.hadoop.hive.serde2.SMS4Rewriter encryption algorithm class during data insertion.</p>
<pre>REMOVE TABLE hbase_tablename [WHERE where_condition];</pre>	<p>The statement is used to delete data that meets criteria from the Hive on HBase table. For details, see Deleting Single-row Records from Hive on HBase.</p>	<pre>remove table hbase_table1 where id = 1;</pre>	<p>The statement is used to delete data that meets the criterion of "id = 1" from the table.</p>

Extended Syntax	Syntax Description	Syntax Example	Example Description
<pre>CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_ name (col_name data_type [COMMENT col_comment], ...) [ROW FORMAT row_format] STORED AS inputformat 'org.apache.hado op.hive.contrib.fil eformat.Specifie dDelimiterInput- Format' outputformat 'org.apache.hadoop p.hive ql.io.HiveIlg noreKeyTextOutpu tFormat';</pre>	<p>The statement is used to create a hive table and specify that the table supports customized row delimiters. For details, see Customizing Row Separators.</p>	<pre>create table blu(time string, num string, msg string) row format delimited fields terminated by ',' stored as inputformat 'org.apache.hado op.hive.contrib.fil eformat.Specifie dDelimiterInput- Format' outputformat 'org.apache.hadoop p.hive ql.io.HiveIlg noreKeyTextOutpu tFormat';</pre>	<p>The statement is used to create table blu and set inputformat to SpecifiedDelimiterInputFormat so that the query row delimiter can be specified during the query.</p>

16.5.1.3 WebHCat

 NOTE

- The following uses the service IP address of WebHCat and the WebHCat HTTP port configured during the installation as an example.
- You can perform the example operations only after **kin** authentication is implemented on the installed client.
- The examples of the HTTPS protocol are as follows. To use the HTTP protocol, you need to perform the following operations:
 1. Switch the RESTful API to the HTTP protocol. For details, see [Configuring HTTPS/ HTTP-based REST APIs](#).
 2. Delete **--insecure** from the example and replace HTTPS with HTTP, for example:

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:9111/templeton/v1/status'
```

is changed to

```
curl -i -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/status'
```
- Before performing the operation, ensure that the curl in use is later than 7.34.0. You can run the following command to view the curl version:

```
curl -V
```

 1. :version(GET)
 - Description

Queries a list of response types supported by WebHCat.

- URL
`https://www.myserver.com/templeton/:version`
- Parameter

Parameter	Description
<code>:version</code>	WebHCat version number. Currently, the version number must be <code>v1</code> .

- Returned result

Parameter	Description
<code>responseTypes</code>	List of response types supported by WebHCat.

- Example

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:9111/templeton/v1'
```

2. status (GET)

- Description
Obtains the status of the current server.
- URL
`https://www.myserver.com/templeton/v1/status`
- Parameter
None
- Returned result

Parameter	Description
<code>status</code>	If the WebHCat connection is normal, OK is returned.
<code>version</code>	Character string, including the version number, for example, <code>v1</code> .

- Example

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:9111/templeton/v1/status'
```

3. version (GET)

- Description
Obtains the WebHCat version of the server.
- URL
`https://www.myserver.com/templeton/v1/version`
- Parameter
None
- Returned result

Parameter	Description
supportedVersions	All supported versions.
version	WebHCat version of the server.

- Example

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:9111/templeton/v1/version'
```

4. version/hive (GET)

- Description

Obtains the Hive version of the server.

- URL

<https://www.myserver.com/templeton/v1/version/hive>

- Parameter

None

- Returned result

Parameter	Description
module	Hive.
version	Hive version.

- Example

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:9111/templeton/v1/version/hive'
```

5. version/hadoop (GET)

- Description

Obtains the Hadoop version of the server.

- URL

<https://www.myserver.com/templeton/v1/version/hadoop>

- Parameter

None

- Returned result

Parameter	Description
module	Hadoop.
version	Hadoop version.

- Example

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:9111/templeton/v1/version/hadoop'
```

6. ddl (POST)

- Description

Executes a DDL statement.

- URL

<https://www.myserver.com/templeton/v1/ddl>

- Parameter

Parameter	Description
exec	HCatalog DDL statement to be executed.
group	User group used when DDL is used to create a table.
permissions	Permission used when DDL is used to create a table. The format is rwxr-xr-x .

- Returned result

Parameter	Description
stdout	Standard output value during HCatalog execution. The value may be empty.
stderr	Error output during HCatalog execution. The value may be empty.
exitcode	Return value of HCatalog.

- Example

```
curl -i -u : --insecure --negotiate -d exec="show tables" 'https://10.64.35.144:9111/templeton/v1/ddl'
```

7. ddl/database (GET)

- Description

Lists all databases.

- URL

<https://www.myserver.com/templeton/v1/ddl/database>

- Parameter

Parameter	Description
like	Regular expression used to match the database name.

- Returned result

Parameter	Description
databases	Database name.

- Example

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:9111/templeton/v1/ddl/database'
```

8. ddl/database/:db (GET)

- Description
Obtains details about a specified database.
- URL
<https://www.myserver.com/templeton/v1/ddl/database/:db>
- Parameter

Parameter	Description
:db	Database name.

- Returned result

Parameter	Description
location	Database location.
comment	Database remarks. If there are no database remarks, the value is null.
database	Database name.
owner	Database owner.
owertype	Type of the database owner.

- Example
`curl -i -u : --insecure --negotiate 'https://10.64.35.144:9111/templeton/v1/ddl/database/default'`

9. ddl/database/:db (PUT)

- Description
Creates a database.
- URL
<https://www.myserver.com/templeton/v1/ddl/database/:db>
- Parameter

Parameter	Description
:db	Database name.
group	User group used for creating the database.
permission	Permission used for creating the database.
location	Database location.
comment	Database remarks, for example, description.
properties	Database properties.

- Returned result

Parameter	Description
database	Name of the newly created database.

- Example

```
curl -i -u : --insecure --negotiate -X PUT -HContent-type:application/json -d '{"location": "/tmp/a", "comment": "my db", "properties": {"a": "b"}}' https://10.64.35.144:9111/templeton/v1/ddl/database/db2'
```

10. ddl/database/:db (DELETE)

- Description

Deletes a database.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db>

- Parameter

Parameter	Description
:db	Database name.
ifExists	If the specified database does not exist, Hive returns an error unless ifExists is set to true .
option	Set the parameter to cascade or restrict . If you set it to cascade , all data and definitions are cleared. If you set it to restrict , the table content is empty and the mode does not exist.

- Returned result

Parameter	Description
database	Name of the deleted database.

- Example

```
curl -i -u : --insecure --negotiate -X DELETE 'https://10.64.35.144:9111/templeton/v1/ddl/database/db3?ifExists=true'
```

11. ddl/database/:db/table (GET)

- Description

Lists all tables in the database.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table>

- Parameter

Parameter	Description
:db	Database name.
like	Regular expression used to match a table name.

- Returned result

Parameter	Description
database	Database name.
tables	List of tables in the database.

- Example

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:9111/templeton/v1/ddl/database/default/table'
```

12. ddl/database/:db/table/:table (GET)

- Description

Obtains details about a table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table>

- Parameter

Parameter	Description
:db	Database name.
:table	Table name.
format	The format is "format=extended". If you want to see additional information, use "table extended like".

- Returned result

Parameter	Description
columns	Column name and type.
database	Database name.
table	Table name.
partitioned	Whether a table is a partition table. This parameter is available only when the table format is extended .

Parameter	Description
location	Table location. This parameter is available only when the table format is extended .
outputformat	Output format. This parameter is available only when the table format is extended .
inputformat	Input format. This parameter is available only when the table format is extended .
owner	Table owner. This parameter is available only when the table format is extended .
partitionColumns	Partition column. This parameter is available only when the table format is extended .

- Example

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1?format=extended'
```

13. ddl/database/:db/table/:table (PUT)

- Description

Creates a table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table>

- Parameter

Parameter	Description
:db	Database name.
:table	New table name.
group	User group used for creating the table.
permissions	Permission used for creating the table.
external	Allows you to specify a location so that Hive does not use the default location for this table.
ifNotExists	If this parameter is set to true , no error is reported if a table exists.
comment	Remarks.

Parameter	Description
columns	Column description, including the column name, type, and optional remarks.
partitionedBy	Partition column description, which is used to partition tables. The columns parameter is used to list the column name, type, and optional remarks.
clusteredBy	Bucket column description, including the columnNames , sortedBy , and numberOfBuckets parameters. The columnNames parameter includes columnName and sorting sequence (ASC indicates an ascending order, and DESC indicates a descending order).
format	Storage format. The parameters include rowFormat , storedAs , and storedBy .
location	HDFS path.
tableProperties	Table property names and values (name-value pairs).

- Returned result

Parameter	Description
database	Database name.
table	Table name.

- Example

```
curl -i -u : --insecure --negotiate -X PUT -HContent-type:application/json -d '{"columns": [{"name": "id", "type": "int"}, {"name": "name", "type": "string"}], "comment": "hello", "format": {"storedAs": "orc"} }' 'https://10.64.35.144:9111/templeton/v1/ddl/database/db3/table/tbl1'
```

14. ddl/database/:db/table/:table (POST)

- Description
Renames a table.
- URL
<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table>
- Parameter

Parameter	Description
:db	Database name.
:table	Existing table name.
rename	New table name.

- Returned result

Parameter	Description
database	Database name.
table	New table name.

- Example

```
curl -i -u : --insecure --negotiate -d rename=table1 'https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/tbl1'
```

15. ddl/database/:db/table/:table (DELETE)

- Description

Deletes a table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table>

- Parameter

Parameter	Description
:db	Database name.
:table	Table name.
ifExists	If this parameter is set to true , no error is reported.

- Returned result

Parameter	Description
database	Database name.
table	Table name.

- Example

```
curl -i -u : --insecure --negotiate -X DELETE 'https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/table2?ifExists=true'
```

16. ddl/database/:db/table/:existingtable/like/:newtable (PUT)

- Description

Creates a table that is the same as an existing table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:existingtable/like/:newtable>

- Parameter

Parameter	Description
:db	Database name.
:existingtable	Existing table name.
:newtable	New table name.
group	User group used for creating the table.
permissions	Permission used for creating the table.
external	Allows you to specify a location so that Hive does not use the default location for this table.
ifNotExists	If this parameter is set to true , the Hive does not report an error if a table already exists.
location	HDFS path.

- Returned result

Parameter	Description
database	Database name.
table	Table name.

- Example

```
curl -i -u : --insecure --negotiate -X PUT -HContent-type:application/json -d '{"ifNotExists": "true"}' 'https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/like/tt1'
```

17. ddl/database/:db/table/:table/partition(GET)

- Description

Lists information about all partitions of a table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition>

- Parameter

Parameter	Description
:db	Database name.
:table	Table name.

- Returned result

Parameter	Description
database	Database name.
table	Table name.
partitions	List of partition attribute values and partition names.

- Example

```
curl -i -u : --insecure --negotiate https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/x1/partition
```

18. ddl/database/:db/table/:table/partition/:partition(GET)

- Description

Lists information about a specific partition of a table.

- URL

https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition/:partition

- Parameter

Parameter	Description
:db	Database name.
:table	Table name.
:partition	Partition name. Exercise caution when decoding HTTP quote, for example, country=%27algeria%27 .

- Returned result

Parameter	Description
database	Database name.
table	Table name.
partition	Partition name.
partitioned	If this parameter is set to true , the table is a partitioned table.
location	Storage path of the table.
outputFormat	Output format.
columns	Column name, type, and remarks.
owner	Owner.
partitionColumns	Partition column.

Parameter	Description
inputFormat	Input format.
totalNumberFiles	Number of files in a partition.
totalFileSize	Total size of files in a partition.
maxFileSize	Maximum file size.
minFileSize	Minimum file size.
lastAccessTime	Last access time.
lastUpdateTime	Last update time.

- Example

```
curl -i -u : --insecure --negotiate https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/x1/partition/dt=1
```

19. ddl/database/:db/table/:table/partition/:partition(PUT)

- Description

Adds a table partition.

- URL

https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition/:partition

- Parameter

Parameter	Description
:db	Database name.
:table	Table name.
group	User group used for creating a partition.
permissions	User permission used for creating a partition.
location	Storage location of the new partition.
ifNotExists	If this parameter is set to true , the system reports an error when the partition already exists.

- Returned result

Parameter	Description
database	Database name.
table	Table name.

Parameter	Description
partitions	Partition name.

- Example

```
curl -i -u : --insecure --negotiate -X PUT -HContent-type:application/json -d '{}' https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/x1/partition/dt=10
```

20. ddl/database/:db/table/:table/partition/:partition(DELETE)

- Description

Deletes a table partition.

- URL

https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition/:partition

- Parameter

Parameter	Description
:db	Database name.
:table	Table name.
group	User group used for deleting a new partition.
permissions	User permission used for deleting a new partition. The format is rwXrw-r-x .
ifExists	If the specified partition does not exist, the Hive reports an error, unless this parameter is set to true .

- Returned result

Parameter	Description
database	Database name.
table	Table name.
partitions	Partition name.

- Example

```
curl -i -u : --insecure --negotiate -X DELETE -HContent-type:application/json -d '{}' https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/x1/partition/dt=10
```

21. ddl/database/:db/table/:table/column(GET)

- Description

Obtains a column list of a table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/column>

- Parameter

Parameter	Description
:db	Database name.
:table	Table name.

- Returned result

Parameter	Description
database	Database name.
table	Table name.
columns	Column name and type.

- Example

```
curl -i -u : --insecure --negotiate https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/column
```

22. ddl/database/:db/table/:table/column/:column(GET)

- Description

Obtains details about a column in a table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/column/:column>

- Parameter

Parameter	Description
:db	Database name.
:table	Table name.
:column	Column name.

- Returned result

Parameter	Description
database	Database name.
table	Table name.
column	Column name and type.

- Example

```
curl -i -u : --insecure --negotiate https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/column/id
```

23. ddl/database/:db/table/:table/column/:column(PUT)

- Description
Adds a column to a table.
- URL
<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/column/:column>
- Parameter

Parameter	Description
:db	Database name.
:table	Table name.
:column	Column name.
type	Column type, for example, string and int.
comment	Column remarks, for example, description.

- Returned result

Parameter	Description
database	Database name.
table	Table name.
column	Column name.

- Example

```
curl -i -u : --insecure --negotiate -X PUT -HContent-type:application/json -d '{"type": "string", "comment": "new column"}' https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/column/name
```

24. ddl/database/:db/table/:table/property(GET)

- Description
Obtains properties of a table.
- URL
<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/property>
- Parameter

Parameter	Description
:db	Database name.
:table	Table name.

- Returned result

Parameter	Description
database	Database name.
table	Table name.
properties	Property.

- Example

```
curl -i -u : --insecure --negotiate https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/property
```

25. ddl/database/:db/table/:table/property/:property(GET)

- Description

Obtains a specific property value of a table.

- URL

https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/property/:property

- Parameter

Parameter	Description
:db	Database name.
:table	Table name.
:property	Property name.

- Returned result

Parameter	Description
database	Database name.
table	Table name.
property	Property list.

- Example

```
curl -i -u : --insecure --negotiate https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/property/last_modified_by
```

26. ddl/database/:db/table/:table/property/:property(PUT)

- Description

Adds a property value to a table.

- URL

https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/property/:property

- Parameter

Parameter	Description
:db	Database name.
:table	Table name.
:property	Property name.
value	Property value.

- Returned result

Parameter	Description
database	Database name.
table	Table name.
property	Property name.

- Example

```
curl -i -u : --insecure --negotiate -X PUT -HContent-type:application/json -d '{"value": "my value"}' https://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/property/mykey
```

27. mapreduce/jar(POST)

- Description

Executes a MapReduce job. Before executing a MapReduce job, upload the JAR file of the MapReduce job to HDFS.

- URL

<https://www.myserver.com/templeton/v1/mapreduce/jar>

- Parameter

Parameter	Description
jar	JAR file of the MapReduce job to be executed.
class	Class of the MapReduce job to be executed.
libjars	JAR file names of classpath to be added, separated by commas (,).
files	Names of files to be copied to the MapReduce cluster, separated by commas (,).
arg	Input parameter received by the Main class.
define	This parameter is used to configure Hadoop in the define=NAME=VALUE format.

Parameter	Description
statusdir	WebHCat writes the status of the MapReduce task to statusdir . If this parameter is set, you need to manually delete it.
enablelog	If statusdir is set and enablelog is set to true , Hadoop task configurations and logs are collected to \$statusdir/logs . Then, successful and failed attempts are recorded in the logs. The layout of the subdirectories in \$statusdir/logs is as follows: logs/\$job_id (directory for \$job_id) logs/\$job_id/job.xml.html logs/\$job_id/\$attempt_id (directory for \$attempt_id) logs/\$job_id/\$attempt_id/stderr logs/\$job_id/\$attempt_id/stdout logs/\$job_id/\$attempt_id/syslog Only Hadoop 1.X is supported.
callback	Callback address after MapReduce job execution. Use \$jobId to embed the job ID in the callback address. In the callback address, replace the \$jobId with the job ID.

– Returned result

Parameter	Description
id	Job ID, similar to job_201110132141_0001 .

– Example

```
curl -i -u : --insecure --negotiate -d jar="/tmp/word.count-0.0.1-SNAPSHOT.jar" -d class=com.huawei.word.count.WD -d statusdir="/output" -d enablelog=true "https://10.64.35.144:9111/templeton/v1/mapreduce/jar"
```

28. mapreduce/streaming(POST)

– Description

Submits a MapReduce job in Streaming mode.

– URL

<https://www.myserver.com/templeton/v1/mapreduce/streaming>

– Parameter

Parameter	Description
input	Input path of Hadoop.
output	Output save path. If this parameter is not specified, WebHCat will store the output in a path that can be found by using queue resources.
mapper	Location of the mapper program.
reducer	Location of the reducer program.
files	Add HDFS files to the distributed cache.
arg	Set an argument.
define	Set Hadoop configuration variables in the define=NAME=VALUE format.
cmdenv	Set environment variables in the cmdenv=NAME=VALUE format.
statusdir	WebHCat writes the status of the MapReduce task to statusdir . If this parameter is set, you need to manually delete it.
enablelog	<p>If statusdir is set and enablelog is set to true, Hadoop task configurations and logs are collected to \$statusdir/logs. Then, successful and failed attempts are recorded in the logs. The layout of the subdirectories in \$statusdir/logs is as follows:</p> <p>logs/\$job_id (directory for \$job_id) logs/\$job_id/job.xml.html logs/\$job_id/\$attempt_id (directory for \$attempt_id) logs/\$job_id/\$attempt_id/stderr logs/\$job_id/\$attempt_id/stdout logs/\$job_id/\$attempt_id/syslog Only Hadoop 1.X is supported.</p>
callback	Callback address after MapReduce job execution. Use \$jobId to embed the job ID in the callback address. In the callback address, replace the \$jobId with the job ID.

- Returned result

Parameter	Description
id	Job ID, similar to job_201110132141_0001 .

- Example

```
curl -i -u : --insecure --negotiate -d input=/input -d output=/oooo -d mapper=/bin/cat -d reducer="/usr/bin/wc -w" -d statusdir="/output" 'https://10.64.35.144:9111/templeton/v1/mapreduce/streaming'
```

 **NOTE**

Before using this API, ensure that the prerequisites are met. For details, see [Rules](#).

29. /hive(POST)

- Description

Runs Hive commands.

- URL

<https://www.myserver.com/templeton/v1/hive>

- Parameter

Parameter	Description
execute	Hive commands, including entire and short Hive commands.
file	HDFS file containing Hive commands.
files	Names of files to be copied to the MapReduce cluster, separated by commas (,).
arg	Set an argument.
define	Hive configuration. The format is define=key=value . When using the post statement, you need to configure the scratch dir of the instance. The WebHCat instance uses define=hive.exec.scratchdir=/tmp/hive-scratch , and the WebHCat1 instance uses define=hive.exec.scratchdir=/tmp/hive1-scratch . The same rule applies to other instances.
statusdir	WebHCat writes the status of the MapReduce task to statusdir . If this parameter is set, you need to manually delete it.

Parameter	Description
enablelog	<p>If statusdir is set and enablelog is set to true, Hadoop task configurations and logs are collected to \$statusdir/logs. Then, successful and failed attempts are recorded in the logs. The layout of the subdirectories in \$statusdir/logs is as follows:</p> <ul style="list-style-type: none"> logs/\$job_id (directory for \$job_id) logs/\$job_id/job.xml.html logs/\$job_id/\$attempt_id (directory for \$attempt_id) logs/\$job_id/\$attempt_id/stderr logs/\$job_id/\$attempt_id/stdout logs/\$job_id/\$attempt_id/syslog
callback	<p>Callback address after MapReduce job execution. Use \$jobid to embed the job ID in the callback address. In the callback address, replace the \$jobid with the job ID.</p>

- Returned result

Parameter	Description
id	Job ID, similar to job_201110132141_0001 .

- Example

```
curl -i -u : --insecure --negotiate -d execute="select count(*) from t1" -d
define=hive.exec.scratchdir=/tmp/hive-scratch -d statusdir="/output" "https://10.64.35.144:9111/
templeton/v1/hive"
```

30. jobs(GET)

- Description
Obtains all job IDs.
- URL
<https://www.myserver.com/templeton/v1/jobs>
- Parameter

Parameter	Description
fields	If this parameter is set to *, details about each job are returned. If this parameter is not set, only a job ID is returned. The parameter can only be set to *. If the parameter is set to another value, an exception occurs.
jobid	If jobid is set, only jobs whose lexicographic order is greater than jobid are returned. For example, if the value of jobid is job_201312091733_0001 , only the job whose value is greater than the value can be returned. The number of returned jobs depends on the value of numrecords .
numrecords	If numrecords and jobid are set, the jobid list is sorted lexicographically. After jobid is returned, the maximum value of numrecords can be obtained. If jobid is not set but numrecords is set, the maximum value of numrecords can be obtained after the jobid list is sorted lexicographically. In contrast, if numrecords is not set but jobid is set, all jobs whose lexicographic orders are greater than jobid will be returned.
showall	If showall is set to true , the request will return all jobs the user has permission to view, not only the jobs belonging to the user.

- Returned result

Parameter	Description
id	Job id
detail	If the value of showall is true , details are displayed. Otherwise, the value is null.

- Example

```
curl -i -u : --insecure --negotiate "https://10.64.35.144:9111/templeton/v1/jobs"
```

31. jobs/:jobid(GET)

- Description
Obtains information about a specified job.
- URL
<https://www.myserver.com/templeton/v1/jobs/:jobid>
- Parameter

Parameter	Description
jobid	Job ID, received after a job is created.

- Returned result

Parameter	Description
status	JSON object containing job status information.
profile	JSON object containing job information. WebHCat parses information in the JobProfile object. The object varies according to the Hadoop version.
id	Job ID.
percentComplete	Job completion percentage, for example, 75%. If the job is complete, the value is null.
user	User who created the job.
callback	Callback URL (if any).
userargs	Parameter argument and parameter value when a user submits a job.
exitValue	Exit value of the job.

- Example

```
curl -i -u : --insecure --negotiate "https://10.64.35.144:9111/templeton/v1/jobs/job_1440386556001_0255"
```

32. jobs/:jobid(DELETE)

- Description
Kills a job.
- URL
<https://www.myserver.com/templeton/v1/jobs/:jobid>
- Parameter

Parameter	Description
:jobid	ID of the job to be deleted.

- Returned result

Parameter	Description
user	User who submits a job.
status	JSON object containing job status information.
profile	JSON object containing job information. WebHCat parses information in the JobProfile object. The object varies according to the Hadoop version.
id	Job ID.
callback	Callback URL (if any).

- Example

```
curl -i -u : --insecure --negotiate -X DELETE "https://10.64.35.143:9111/templeton/v1/jobs/job_1440386556001_0265"
```

16.5.2 Hive Access Configuration on Windows Using EIPs

Scenario

This section describes how to bind Elastic IP addresses (EIPs) to a cluster and configure Hive files so that sample files can be compiled locally.

This section uses hive-jdbc-example as an example.

Procedure

- Step 1** Apply for an EIP for each node in the cluster and add public IP addresses and corresponding host domain names of all nodes to the Windows local **hosts** file. (If a host name contains uppercase letters, change them to lowercase letters.)

1. On the VPC console, apply for EIPs (the number of EIPs you buy should be equal to the number of nodes in the cluster), click the name of each node in the MRS cluster, and bind an EIP to each node on the **EIPs** page.

For details, see **Virtual Private Cloud > User Guide > EIP > Assigning an EIP and Binding It to an ECS**.

2. Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.


```

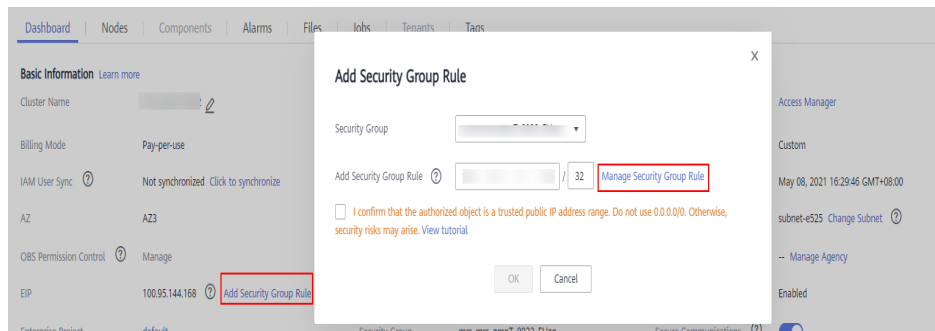
1 Mapping between public IP addresses and private IP addresses
2 100.95.144.120 172.16.0.120
3 100.95.144.139 172.16.0.42
4 100.93.144.139 172.16.0.62
5 100.95.144.139 172.16.0.200
6 100.93.144.139 172.16.0.139
7 100.93.144.139 172.16.0.214
8
9 hosts file in the cluster
10 172.16.0.120 node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
11 172.16.0.42 node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
12 172.16.0.62 node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
13 172.16.0.200 node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
14 172.16.0.139 node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
15 172.16.0.214 node-master2pVNu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVNu.ead64699-185a-4290-bbef-1a07e2f0459b.com.
16
17 hosts file that should be added to Windows
18 100.95.144.139 node-group-1xzi00002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
19 100.95.144.139 node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
20 100.93.144.139 node-group-1xzi00003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
21 100.95.144.139 node-master1ceip.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1ceIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
22 100.93.144.139 node-group-1xzi00001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
23 100.93.144.139 node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVNu.ead64699-185a-4290-bbef-1a07e2f0459b.com.

```

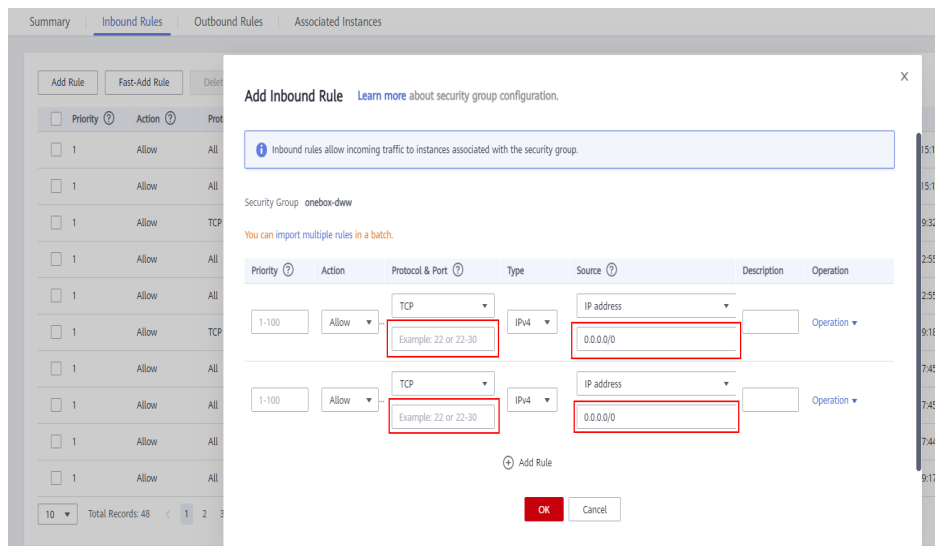
Step 2 Change the IP addresses in the `krb5.conf` file to the corresponding host names.

Step 3 Configure security group rules for the cluster.

1. On the **Dashboard** page, choose **Add Security Group Rule > Manage Security Group Rule**.



2. On the **Inbound Rules** tab page, click **Add Rule**. In the **Add Inbound Rule** dialog box, configure Windows IP addresses and ports 21730TCP, 21731TCP/UDP, and 21732TCP/UDP.



Step 4 On Manager, choose **Cluster > Services > Hive > More > Download Client**, and copy the `core-site.xml` and `hiveclient.properties` files on the client to the `resources` directory of the sample project.

Step 5 In the sample code, change the ZooKeeper IP addresses in the JDBC URL to the HiveServer2 host name for connection. Change the URL to `jdbc:hive2://HiveServer host name:10000/`.

 NOTE

- The IP addresses in the **/hiveserver2** directory of ZooKeeper are private IP addresses and cannot be used to connect to Hive from Windows. Therefore, you need to replace ZooKeeper IP addresses with the HiveServer2 host name.
- To obtain the HiveServer2 host name, choose **Cluster > Services > Hive > Instance** on Manager and view **Host Name of HiveServer** on the **Instance** page.

```

// Build JDBC URL
StringBuilder strBuilder = new StringBuilder("jdbc:hive2://").append(zkQuorum).append("/");
StringBuilder strBuilder = new StringBuilder("jdbc:hive2://node-master110ks-0bc17d76-481f-4b24-83af-159ed415ad95.com:10000/");

if ("KERBEROS".equalsIgnoreCase(auth)) {
    strBuilder
        .append(";serviceDiscoveryMode=")
        .append(serviceDiscoveryMode)
        .append(";zooKeeperNamespace=")
        .append(zooKeeperNamespace)
        .append(";sasL_qop=")
        .append(sasL_qop)
        .append(";auth=")
        .append(auth)
        .append(";principal=")
        .append(principal)
        .append(";user.principal=")
        .append(USER_NAME)
        .append(";user.keytab=")
        .append(USER_KEYTAB_FILE)
        .append(";");
} else {
    /* Normal mode */
    strBuilder
        .append(";serviceDiscoveryMode=")
        .append(serviceDiscoveryMode)
        .append(";zooKeeperNamespace=")
        .append(zooKeeperNamespace)
        .append(";auth=none");
}
    
```

Step 6 Before running the sample code, change **PRINCIPAL_NAME** in the sample code to the username for security authentication.

----End

16.5.3 FAQ

16.5.3.1 A Message Is Displayed Stating "Unable to read HiveServer2 configs from ZooKeeper" During the Use of the Secondary Development Program

Question

A Message Is Displayed Stating "Unable to read HiveServer2 configs from ZooKeeper" During the Use of the Secondary Development Program.

Answer

- Possible Causes
 - The used **krb5.conf** and **user.keytab** may not be the latest ones, or the user names in the files do not match with those in example codes.
 - The difference between the time of the client and that of the connected cluster is greater than 5 minutes.
- Possible Causes
 - Check the code and download the latest credential file for user authentication.

- Check whether the difference between the time of the client and that of the connected cluster is greater than 5 minutes. If yes, adjust the time of the client.

16.5.3.2 Problem performing GSS wrap Message Is Displayed Due to IBM JDK Exceptions

Question

IBM JDK is abnormal, and the **Problem performing GSS wrap** message is displayed.

Answer

Possible Causes

The Hive connection duration exceeds the user authentication timeout duration (one day by default), causing authentication failure.

NOTE

The IBM JDK mechanism is different from the Oracle JDK mechanism. IBM JDK executes time check after authentication and login, but does not check external time update. This results in refreshing failure even Hive relogin is invoked explicitly.

Solution

When one Hive connection fails, disable this connection, and create a connection to continue performing previous operations.

16.5.3.3 Hive SQL Is Incompatible with SQL2003 Standards

This document describes the incompatibility issues between Hive SQL and SQL2003.

- **The view cannot be written in having.**

For example:

```
select c_last_name
       ,c_first_name
       ,s_store_name
       ,sum(netpaid) paid
from ssales
where i_color = 'chiffon'
group by c_last_name
        ,c_first_name
        ,s_store_name
having sum(netpaid) > (select 0.05*avg(netpaid) from ssales);
```

Error message:

Error: Error while compiling statement: FAILED:ParseException line 46:23 cannot recognize input near 'select' '0.05' '*' in expression specification (state=42000,code=40000)

- **The exists query entity association is not supported.**

For example:

```
select
       o_orderpriority,
       count(*) as order_count
from
```

```

orders
where
o_orderdate >= date '1996-09-01'
and o_orderdate < date '1996-09-01' + interval '3' month
and exists (
  select
  *
  from
  lineitem
  where
  l_orderkey = o_orderkey
  and l_commitdate < l_receiptdate
)
group by
o_orderpriority
order by
o_orderpriority;

```

Error message:

Error: Error while compiling statement: FAILED: SemanticException [Error 10250]: Line 1:181 Invalid SubQuery expression 'l_receiptdate': For Exists/Not Exists operator SubQuery must be Correlated. (state=42000,code=10250)

- **The Having does not support sub-query.**

For example:

```

select
ps_partkey,
sum(ps_supplycost * ps_availqty) as value
from
partsupp,
supplier,
nation
where
ps_suppkey = s_suppkey
and s_nationkey = n_nationkey
and n_name = 'SAUDI ARABIA'
group by
ps_partkey having
sum(ps_supplycost * ps_availqty) > (
  select
  sum(ps_supplycost * ps_availqty) * 0.0001000000
  from
  partsupp,
  supplier,
  nation
  where
  ps_suppkey = s_suppkey
  and s_nationkey = n_nationkey
  and n_name = 'SAUDI ARABIA'
)
order by
value desc;

```

Error message:

Error: Error while compiling statement: FAILED: SemanticException Line 0:-1 Unsupported SubQuery Expression "SAUDI ARABIA": Only SubQuery expressions that are top level conjuncts are allowed (state=42000,code=40000)

- **Multiple query results cannot be displayed as multiple fields.**

For example:

```

select
c_count,
count(*) as custdist
from
(
  select
  c_custkey,
  count(o_orderkey)

```

```

        from
            customer left outer join orders on
                c_custkey = o_custkey
                and o_comment not like '%pending%requests%'
        group by
            c_custkey
    ) as c_orders (c_custkey, c_count)
group by
    c_count
order by
    custdist desc,
    c_count desc;

```

Error message:

Error: Error while compiling statement: FAILED: ParseException line 1:213 missing EOF at '(' near 'c_orders' (state=42000,code=40000)

- **The query results cannot be compared as fields.**

For example:

```

select
    sum(L_extendedprice) / 7.0 as avg_yearly
from
    lineitem,
    part
where
    p_partkey = L_partkey
    and p_brand = 'Brand#25'
    and p_container = 'MED JAR'
    and L_quantity < (
        select
            0.2 * avg(L_quantity)
        from
            lineitem
        where
            L_partkey = p_partkey
    );

```

Error message:

Error: Error while compiling statement: FAILED: SemanticException [Error 10249]: Line 14:4 Unsupported SubQuery Expression 'ps_suppkey': Correlating expression contains ambiguous column references. (state=42000,code=10249)

- **The multi-table association query does not support the sub-query filter by not in or in.**

For example:

```

select
    p_brand,
    p_type,
    p_size,
    count(distinct ps_suppkey) as supplier_cnt
from
    partsupp,
    part
where
    p_partkey = ps_partkey
    and p_brand <> 'Brand#12'
    and p_type not like 'PROMO PLATED%'
    and p_size in (25, 2, 43, 9, 35, 36, 48, 24)
    and ps_suppkey in (
        select
            s_suppkey as ps_suppkey
        from
            supplier
        where
            s_comment like '%Customer%Complaints%'
    )
group by
    p_brand,

```

```
p_type,  
p_size  
order by  
supplier_cnt desc,  
p_brand,  
p_type,  
p_size;
```

Error message:

Error: Error while compiling statement: FAILED: SemanticException [Error 10249]: Line 14:4
Unsupported SubQuery Expression 'ps_suppkey': Correlating expression contains ambiguous column
references. (state=42000,code=10249)

- **The multi-table association does not support filtering of not in and in sub-queries.**

For example:

```
select  
  c_name,  
  c_custkey,  
  o_orderkey,  
  o_orderdate,  
  o_totalprice,  
  sum(l_quantity)  
from  
  customer,  
  orders,  
  lineitem  
where  
  o_orderkey in (  
    select  
      l_orderkey  
    from  
      lineitem  
    group by  
      l_orderkey having  
        sum(l_quantity) > 315  
  )  
  and c_custkey = o_custkey  
  and o_orderkey = l_orderkey  
group by  
  c_name,  
  c_custkey,  
  o_orderkey,  
  o_orderdate,  
  o_totalprice  
order by  
  o_totalprice desc,  
  o_orderdate  
limit 100;
```

Error message:

Error: Error while compiling statement: FAILED: SemanticException [Error 10249]: Line 13:0
Unsupported SubQuery Expression 'o_orderkey': Correlating expression contains ambiguous column
references. (state=42000,code=10249)

- **The association conditions do not support multiple exists query entities.**

For example:

```
select  
  s_name,  
  count(*) as numwait  
from  
  supplier,  
  lineitem l1,  
  orders,  
  nation  
where  
  s_suppkey = l1.l_suppkey  
  and o_orderkey = l1.l_orderkey
```

```

and o_orderstatus = 'F'
and l1.l_receiptdate > l1.l_commitdate
and exists (
  select
    *
  from
    lineitem l2
  where
    l2.l_orderkey = l1.l_orderkey
    and l2.l_suppkey <> l1.l_suppkey
)
and not exists (
  select
    *
  from
    lineitem l3
  where
    l3.l_orderkey = l1.l_orderkey
    and l3.l_suppkey <> l1.l_suppkey
    and l3.l_receiptdate > l3.l_commitdate
)
and s_nationkey = n_nationkey
and n_name = 'VIETNAM'
group by
  s_name
order by
  numwait desc,
  s_name
limit 100;

```

Error message:

Error: Error while compiling statement: FAILED: SemanticException [Error 10249]: Line 23:8
Unsupported SubQuery Expression 'l_commitdate': Only 1 SubQuery expression is supported.
(state=42000,code=10249)

- **The multi-level in-nested sub-query is not supported.**

For example:

```

select i_item_id item_id,
       sum(sr_return_quantity) sr_item_qty
from store_returns,
     item,
     date_dim
where sr_item_sk = i_item_sk
and d_date in
  (select d_date
   from date_dim
   where d_week_seq in
     (select d_week_seq
      from date_dim
      where d_date in ('1998-01-02','1998-10-15','1998-11-10')))
and sr_returned_date_sk = d_date_sk
group by i_item_id,
         cr_items as
(select i_item_id item_id,
       sum(cr_return_quantity) cr_item_qty
 from catalog_returns,
     item,
     date_dim
 where cr_item_sk = i_item_sk);

```

Error message:

Unsupported SubQuery Expression 'd_week_seq': SubQuery cannot use the table alias: date_dim; this
is also an alias in the Outer Query and SubQuery contains a unqualified column reference
(state=42000,code=10249)

17 Hive Development Guide (Normal Mode)

17.1 Overview

17.1.1 Application Development Overview

Hive Introduction

Hive is an open-source data warehouse built on Hadoop. It stores structured data and provides basic data analysis services using the Hive query language (HQL), a language like the SQL. Hive converts HQL statements to MapReduce or Spark jobs for querying and analyzing massive data stored in Hadoop clusters.

Hive provides the following features:

- Extracts, transforms, and loads (ETL) data using HQL.
- Analyzes massive structured data using HQL.
- Supports flexible data storage formats, including JavaScript object notation (JSON), comma separated values (CSV), TextFile, RCFile, ORCFIELD, and SequenceFile, and supports custom extensions.
- Multiple client connection modes. Interfaces, such as JDBC and Thrift interfaces are supported.

Hive applies to offline massive data analysis (such as log and cluster status analysis), large-scale data mining (such as user behavior analysis, interest region analysis, and region display), and other scenarios.

To ensure Hive high availability (HA), user data security, and service access security, MRS incorporates the following features based on Hive 3.1.0:

- Data file encryption

For Hive features in the Open Source Community, see <https://cwiki.apache.org/confluence/display/hive/designdocs>.

17.1.2 Common Concepts

- **Client**
Users can access the server from the client through the Java API and Thrift API to perform Hive-related operations.
- **HQL**
Similar to SQL
- **HCatalog**
HCatalog is a table information management layer created based on Hive metadata and absorbs DDL commands of Hive. HCatalog provides read/write interfaces for MapReduce and provides Hive command line interfaces (CLIs) for defining data and querying metadata. Hive and MapReduce development personnel can share metadata based on the HCatalog component of MRS, preventing intermediate conversion and adjustment and improving the data processing efficiency.
- **WebHCat**
WebHCat running users use Rest APIs to perform operations, such as running Hive DDL commands, submitting MapReduce tasks, and querying MapReduce task execution results.

17.1.3 Development Process

[Figure 17-1](#) and [Table 17-1](#) illustrates each phase of the development process.

Figure 17-1 Hive application development process

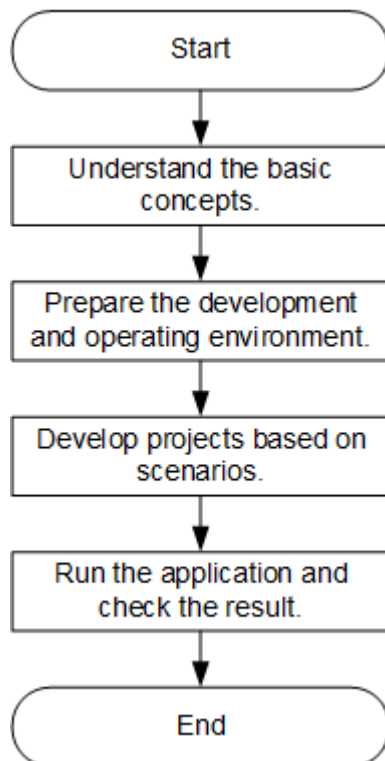


Table 17-1 Description of the Hive application development process

Phase	Description	Reference Document
Understand the basic concepts.	Before application development, understand common concepts about Hive.	Common Concepts
Prepare the development and operating environment.	Hive applications support Java and Python development languages. You are advised to use the IntelliJ IDEA tool to configure the development environment based on the language.	Preparing the Environment
Develop projects based on scenarios.	Provides example projects of the Java and Python languages as well as example projects covering table creation, data load, and data query.	Developing an Application
Run the application and view results.	Describes how to submit a developed application for compiling and view the result.	Commissioning Applications

17.2 Preparing the Environment

17.2.1 Preparing Development and Operating Environment

Preparing Development Environment

Hive applications can be developed by using the JDBC/Python/Python3 API. The following table describes the development and operating environment to be prepared.

Table 17-2 JDBC development environment

Item	Description
OS	<ul style="list-style-type: none"> • Development environment: Windows OS. Windows 7 or later is supported. • Operating environment: Windows OS or Linux OS If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows: The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed. Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> • For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> - Oracle JDK versions: 1.8 - Supported IBM JDK versions: 1.8.5.11 • For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> - OpenJDK: 1.8.0_272 <p>NOTE The server supports only TLS V1.2 or later to ensure security. By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls.</p>
IntelliJ IDEA installation and configuration	<p>Tool used for developing Hive applications. Requirements are as follows: JDK 1.8 is used. IntelliJ IDEA 2019.1 or other compatible versions are used.</p> <p>NOTE</p> <ul style="list-style-type: none"> • If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK. • If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK. • If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.
Installation of Maven	<p>Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.</p>
7-zip	<p>Used to decompress .zip and .rar packages. 7-Zip 16.04 is supported.</p>

Table 17-3 Python development environment

Item	Description
OS	Development and operating environment: Linux OS
Python installation	Python is a tool used to develop Hive applications. Its version must be 2.6.6 or later but should not be later than 2.7.13.
setuptools installation	Basic configuration for the Python development environment. The version must be 5.0 or later.

 **NOTE**

For details about how to install and configure the Python development tool, see the [Configuring the Python Sample Project](#).

Table 17-4 Python3 development environment

Item	Description
OS	Development and operating environment: Linux OS
Python3 installation	Python3 is a tool used to develop Hive applications. Its version must be 3.6 or later but should not be later than 3.8.
setuptools installation	Basic configuration for the Python3 development environment. The version must be 47.3.1.

 **NOTE**

For details about how to install and configure the Python3 development tool, see the [Configuring the Python3 Sample Project](#).

Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
 - a. [Log in to the FusionInsight Manager portal](#) and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Configuration Files Only**. Select the platform type based on the type of the node where the client is to be installed (select **x86_64** for the x86

architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight_Cluster_1_Services_Client.tar**, decompress it to obtain **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar** file. Then, decompress

FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar file to the **D:\FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight_Cluster_1_Services_ClientConfig\Hive\config** and manually import the configuration file to the configuration file directory (usually the **resources** folder) of the Hive sample project.

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 17-5](#) describes the main configuration files.

Table 17-5 Configuration file

Document Name	Function
hivemetastore-site.xml	Configures Hive parameters.
hiveclient.properties	Configures Hive parameters.
user.keytab	Provides Hive user information for Kerberos security authentication.
krb5.conf	Contains Kerberos server configuration information.
core-site.xml	Configures Hive parameters.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
 - a. Install the client on the node. For example, the client installation directory is **/opt/client**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. **Log in to the FusionInsight Manager portal.** Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight_Cluster_1_Services_ClientConfig/Hive/config** directory to the **src/main/resources** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/client/src/main/resources**.

For example, if the client software package is **FusionInsight_Cluster_1_Services_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Hive/config/* root@IP address of the client node:/opt/client/src/main/resources
```

Table 17-6 Configuration files

File	Function
hivemetastore-site.xml	Configures Hive parameters.
hive-site.xml	Configures Hive parameters.

- c. Check the network connection of the client node.
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

17.2.2 Configuring the JDBC Sample Project

Scenario

To run the JDBC API example codes of the Hive component of MRS, perform the following operations.

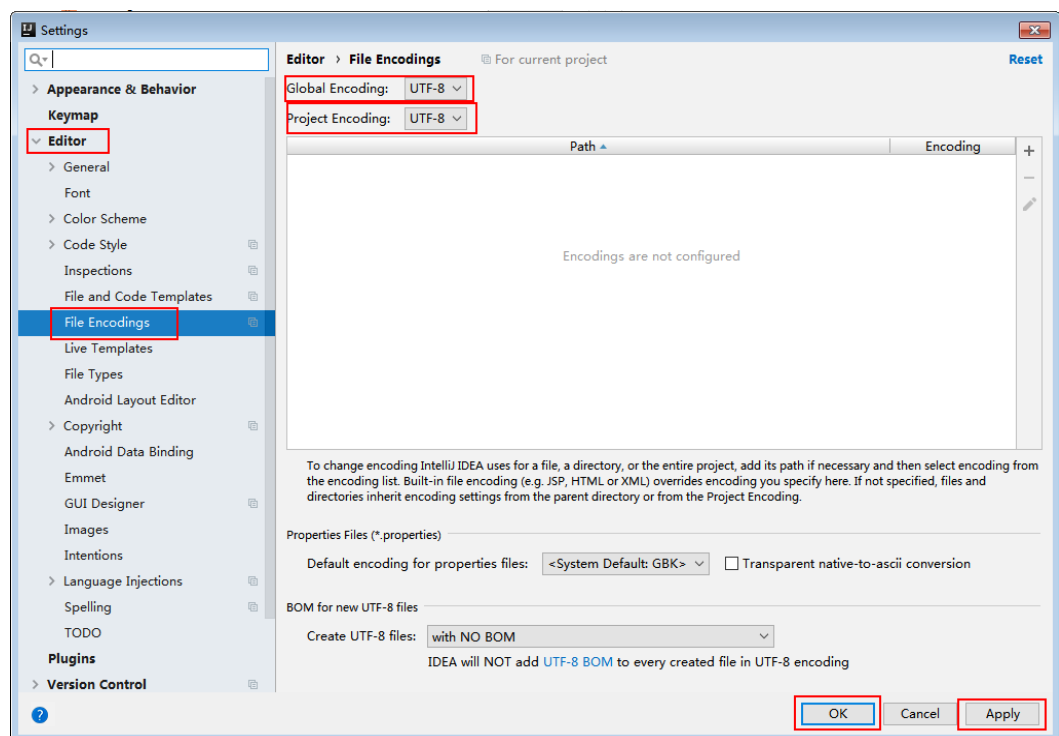
 NOTE

The following uses the development of an application for connecting the Hive service in JDBC mode on Windows as an example.

Procedure

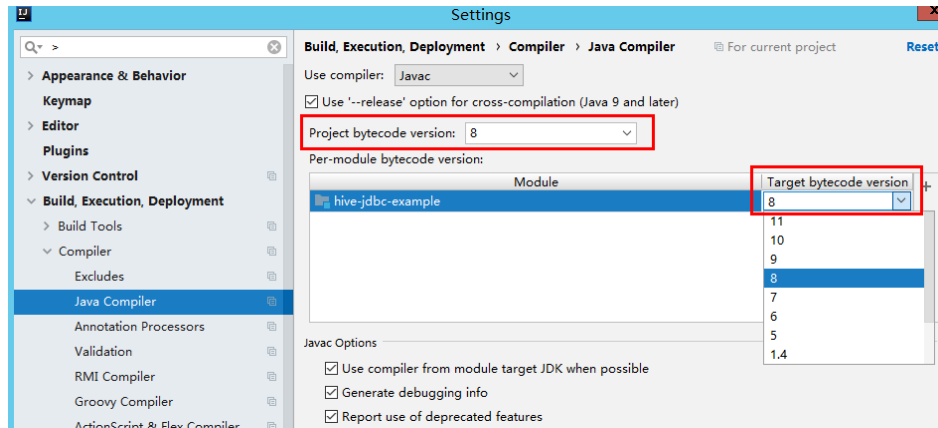
- Step 1** Obtain the sample project folder **hive-jdbc-example** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Go to the client decompression path **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles\Hive\config** and manually import the **core-site.xml** and **hiveclient.properties** files to the **hive-jdbc-example\src\main\resources** directory of the sample project.
- Step 3** Import the example project to the IntelliJ IDEA development environment.
 1. On the IntelliJ IDEA menu bar, choose **File > Open....** The **Open File or Project** window is displayed.
 2. In the displayed window, select the folder **hive-jdbc-example**, and click **OK**. On Windows, the path cannot contain any space.
- Step 4** Set the IntelliJ IDEA text file coding format to prevent garbled characters.
 1. On the IntelliJ IDEA menu bar, choose **File > Settings**. The **Settings** window is displayed.
 2. In the navigating tree on the left, choose **Editor > File Encodings**. In the **Project Encoding** area and **Global Encoding** area, set the value to **UTF-8**, and click **Apply** and **OK**, as shown in [Figure 17-2](#).

Figure 17-2 Setting the IntelliJ IDEA coding format

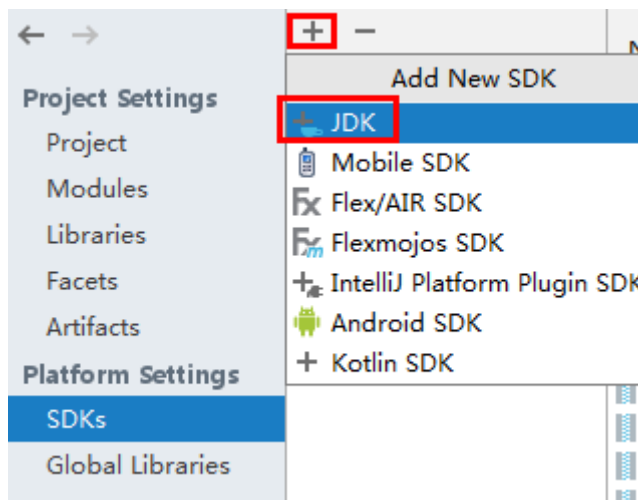


Step 5 Set the JDK of the project.

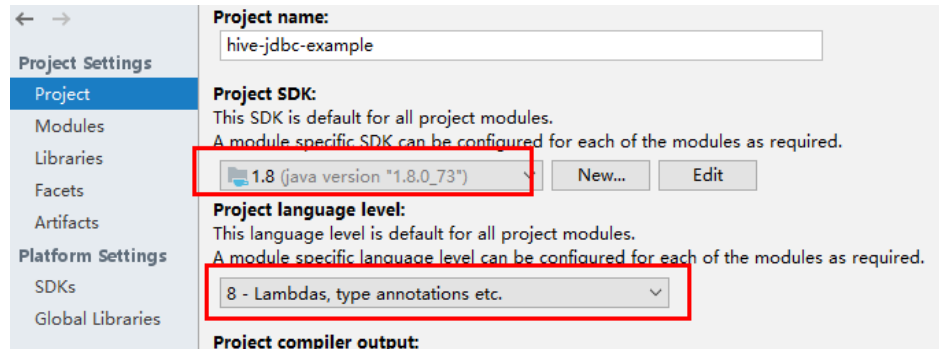
1. On the IntelliJ IDEA menu bar, choose **File > Settings...** The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **hive-jdbc-example**.



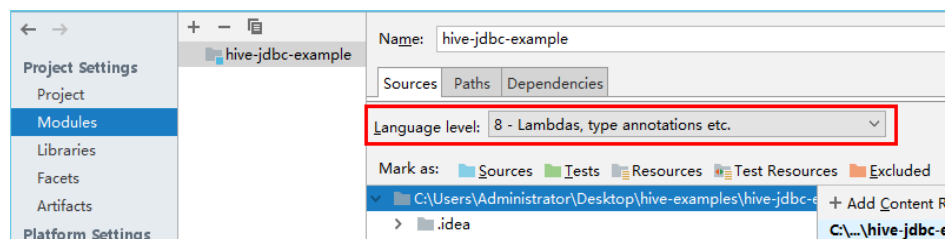
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File > Project Structure...** The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



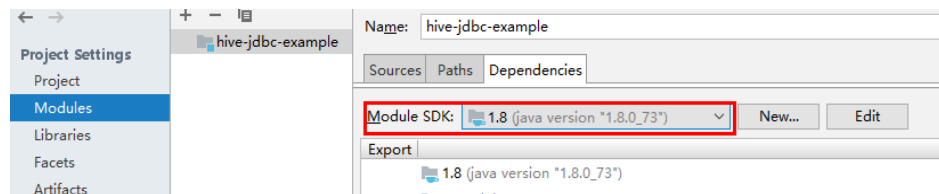
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Source** page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



On the **Dependencies** page, change the value of **Module SDK** to the JDK added in SDKs.

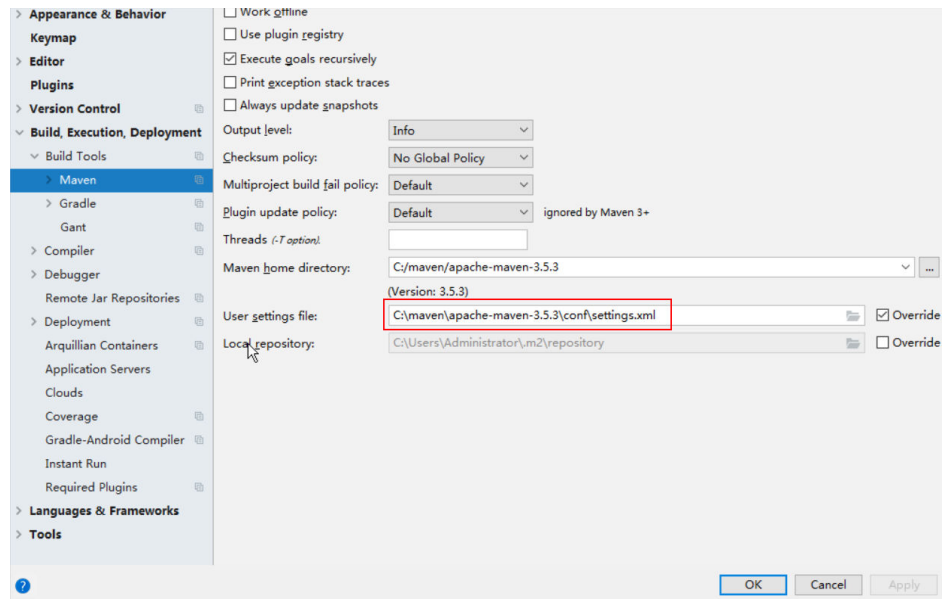


11. Click **Apply** and **OK**.

Step 6 Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is `<Local Maven installation directory>\conf\settings.xml`.

Figure 17-3 Directory for storing the settings.xml file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply**, and then click **OK**.

----End

17.2.3 Configuring the Hcatalog Sample Project

Scenario

To run the HCatalog interface example codes of the Hive component of MRS, perform the following operations.

NOTE

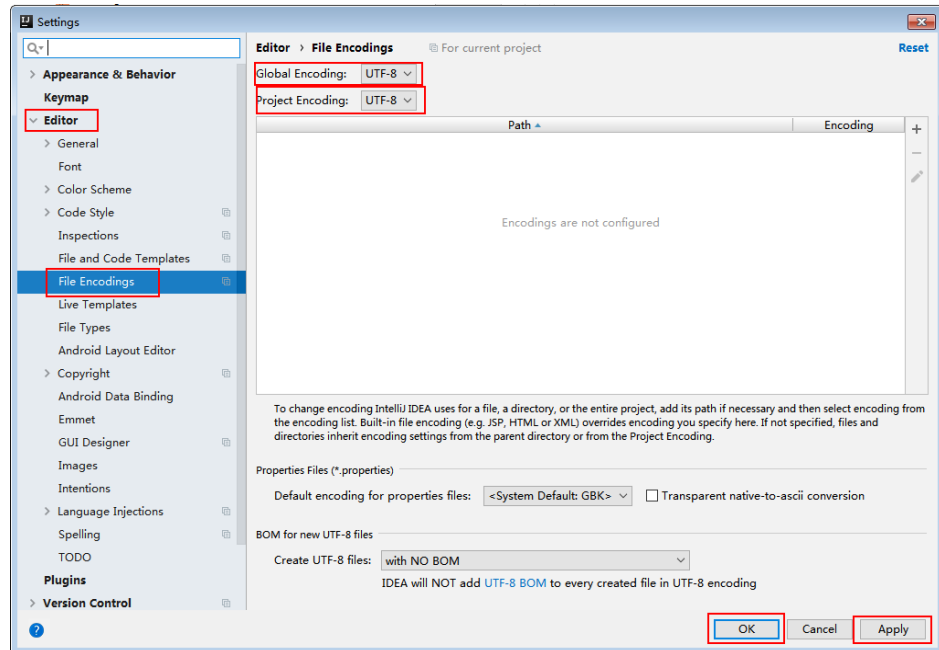
The following uses the development of an application for connecting the Hive service in HCatalog mode on Windows as an example.

Procedure

- Step 1** Obtain the sample project folder **hcatalog-example** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Import the example project to the IntelliJ IDEA development environment.
 1. On the IntelliJ IDEA menu bar, choose **File > Open....** The **Open File or Project** window is displayed.
 2. In the displayed window, select the folder **hcatalog-example**, and click **OK**. On Windows, the path cannot contain any space.
- Step 3** Set the IntelliJ IDEA text file coding format to prevent garbled characters.
 1. On the IntelliJ IDEA menu bar, choose **File > Settings**. The **Settings** window is displayed.

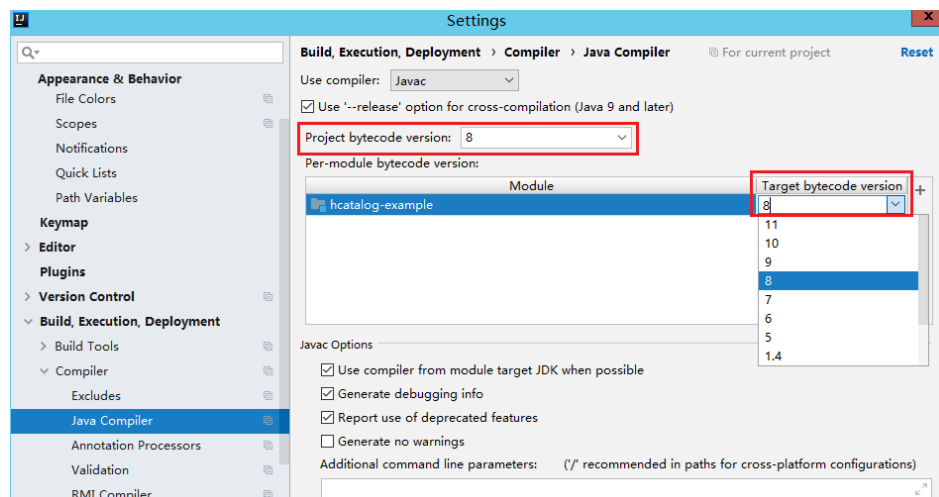
2. Choose **Editor > File Encodings** from the navigation tree. In the **Project Encoding** area and **Global Encoding** area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in **Figure 17-4**.

Figure 17-4 Setting the IntelliJ IDEA coding format

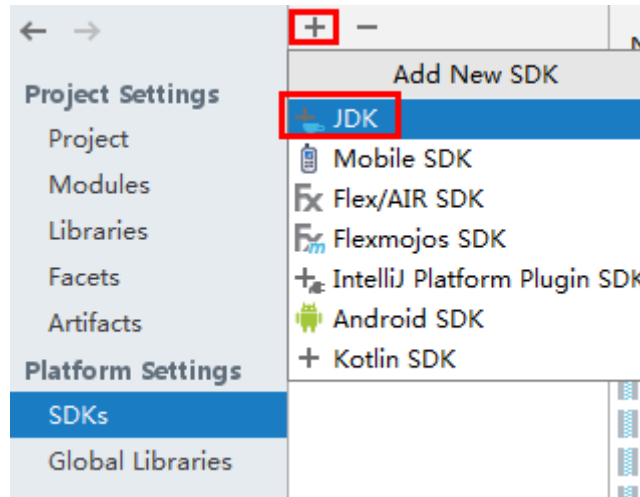


Step 4 Set the JDK of the project.

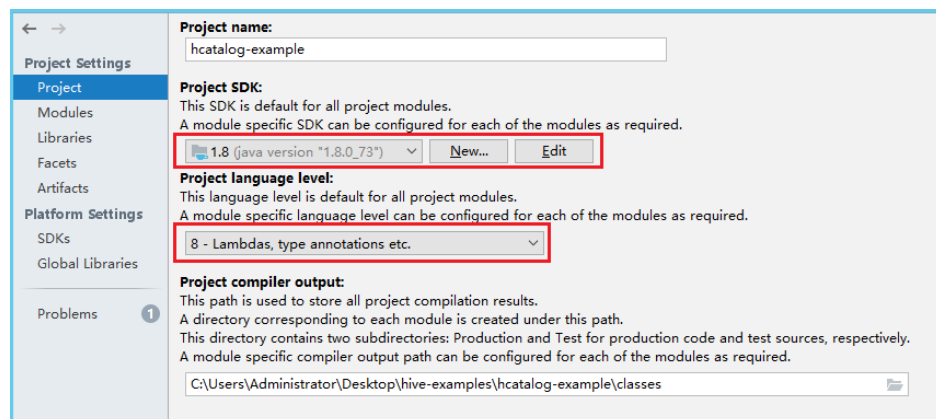
1. On the IntelliJ IDEA menu bar, choose **File > Settings...** The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **hive-jdbc-example**.



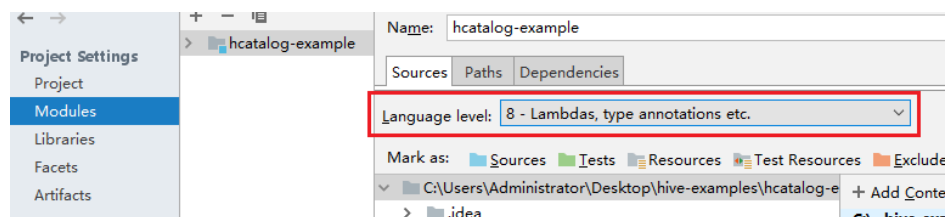
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File > Project Structure...** The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



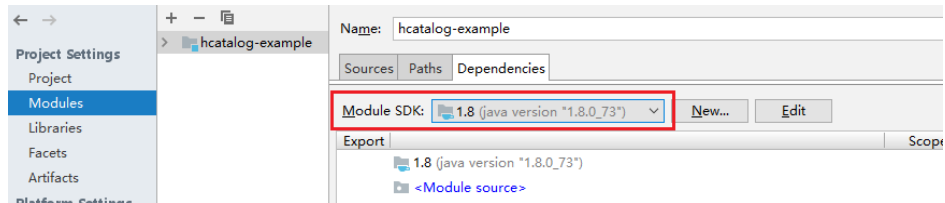
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Source** page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



11. On the **Dependencies** page, change the value of **Module SDK** to the JDK added in SDKs.

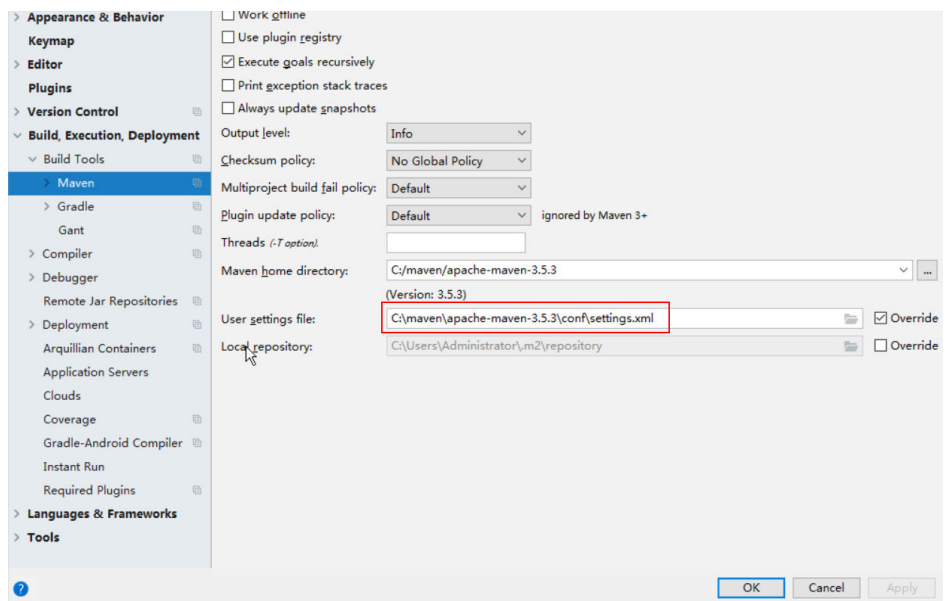


12. Click **Apply** and **OK**.

Step 5 Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is *<Local Maven installation directory>\conf\settings.xml*.

Figure 17-5 Directory for storing the **settings.xml** file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply**, and then click **OK**.

----End

17.2.4 Configuring the Python Sample Project

Scenario

To run the Python interface example codes of the Hive component of MRS, perform the following operations.

Procedure

- Step 1** Python of 2.6.6 or a higher version has been installed on a client. The Python version cannot be higher than 2.7.13.

The Python version can be viewed by running the **python** command on the command-line interface (CLI) of the client. The following information indicates that the Python version is 2.6.6.

```
Python 2.6.6 (r266:84292, Oct 12 2012, 14:23:48)
[GCC 4.4.6 20120305 (Red Hat 4.4.6-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

- Step 2** Setuptools of 5.0 or a higher version has been installed on a client. The setuptools version cannot be higher than 36.8.0.

To obtain the software, visit the official website.

<https://pypi.org/project/setuptools/#files>

Copy the downloaded setuptools package to the client, decompress the package, go to the decompressed directory, and run the **python setup.py install** command in the CLI of the client.

The following information indicates that setuptools 5.7 is installed successfully.

```
Finished processing dependencies for setuptools==5.7
```

- Step 3** Install Python on the client.

1. Obtain the sample project folder **python-examples** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python-examples** folder.
3. Run the **python setup.py install** command in the CLI.

The following information indicates that Python is installed successfully.

```
Finished processing dependencies for pyhs2==0.5.0
```

- Step 4** After the installation is successful, the following files are generated. **python-examples/pyCLI_nosec.py** is the Python client example codes. **python-examples/pyhs2/haconnection.py** is the Python client API. Run **hive_python_client** scripts to execute the SQL functions, for example, **sh hive_python_client 'show tables'**. This function applies to only simple SQL statements and depends on the ZooKeeper client.

----End

17.2.5 Configuring the Python3 Sample Project

Scenario

To run the Python3 interface example codes of the Hive component of FusionInsight MRS, perform the following operations.

Procedure

- Step 1** Python3 of 3.6 or a higher version has been installed on a client. The Python3 version cannot be higher than 3.8.

The Python version can be viewed by running the **python3** command on the command-line interface (CLI) of the client. The following information indicates that the Python version is 3.8.2.

```
Python 3.8.2 (default, Jun 23 2020, 10:26:03)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

Step 2 Setuptools of 47.3.1 version has been installed on a client.

To obtain the software, visit the official website.

<https://pypi.org/project/setuptools/#files>

Copy the downloaded setuptools package to the client, decompress the package, go to the decompressed directory, and run the **python3 setup.py install** command in the CLI of the client.

The following information indicates that setuptools 47.3.1 is installed successfully.

```
Finished processing dependencies for setuptools==47.3.1
```

 **NOTE**

If the system displays a message indicating that the installation of setuptools of 47.3.1 fails, check whether the environment is faulty or whether the problem is caused by Python.

Step 3 Install Python on the client.

1. Obtain the sample project folder **python3-examples** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python3-examples** folder.
3. Go to the **dependency_python3.6**, **dependency_python3.7**, or **dependency_python3.8** folder based on the Python3 version.
4. Run the **whereis easy_install** command to find the path of the **easy_install** program. If there are multiple paths, run the **easy_install --version** command to select the **easy_install** path corresponding to the **setuptools** version, for example, **/usr/local/bin/easy_install**.
5. Run the **easy_install** command to install the egg files in the **dependency_python3.x** folder. If the egg file has dependencies, you can use wildcards to install the egg file. For example:

- **dependency_python3.6** directory:
/usr/local/bin/easy_install future*egg six*egg python*egg sasl-*linux-\$(uname -p).egg thrift-*egg thrift_sasl*egg
- **dependency_python3.7** directory:
/usr/local/bin/easy_install future*egg six*egg sasl-*linux-\$(uname -p).egg thrift-*egg thrift_sasl*egg
- **dependency_python3.8** directory:
/usr/local/bin/easy_install future*egg six*egg python*egg sasl-*linux-\$(uname -p).egg thrift-*linux-\$(uname -p).egg thrift_sasl*egg

If the following information is displayed for each egg file, the installation is successful:

```
Finished processing dependencies for ***
```

Step 4 After the installation is successful, the following files are generated. **python3-examples/pyCLI_sec.py** is the Python client example codes. **python3-examples/pyhive/hive.py** is the Python client API.

----End

17.3 Developing an Application

17.3.1 Typical Scenario Description

Scenarios

A user develops a Hive data analysis application for managing employee information described in [Table 17-7](#) and [Table 17-8](#).

Procedure

Step 1 Prepare data.

1. Create three tables: employee information table **employees_info**, contact table **employees_contact**, and extended employee information table **employees_info_extended**.
 - Fields in the **employees_info** table include the employee ID, name, salary currency, salary, tax category, work place, and hiring date. **R** indicates RMB, and **D** indicates USD.
 - Fields in the **employees_contact** table include the employee ID, mobile phone number, and e-mail address.
 - Fields in the **employees_info_extended** table include the employee ID, name, mobile phone number, e-mail address, salary currency, salary, tax category, and work place. The partition field is the hiring date.

For table creation codes, see [Creating a Table](#).

2. Load employee information to **employees_info**.

For data loading codes, see [Loading Data](#).

[Table 17-7](#) describes employee information.

Table 17-7 Employee information

ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hiring Date
1	Wang	R	8000.01	personal income tax&0.05	Country 1:City1	2014
3	Tom	D	12000.02	personal income tax&0.09	Country 2:City2	2014

ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hiring Date
4	Jack	D	24000.03	personal income tax&0.09	Country3:City3	2014
6	Linda	D	36000.04	personal income tax&0.09	Country4:City4	2014
8	Zhang	R	9000.05	personal income tax&0.05	Country5:City5	2014

- Load employee contact information to **employees_contact**.

Table 17-8 describes employee contact information.

Table 17-8 Employee contact information

ID	Mobile Phone Number	E-mail Address
1	135 XXXX XXXX	xxxx@xx.com
3	159 XXXX XXXX	xxxxx@xx.com.cn
4	186 XXXX XXXX	xxxx@xx.org
6	189 XXXX XXXX	xxxx@xxx.cn
8	134 XXXX XXXX	xxxx@xxxx.cn

- Load extended employee information to **employees_info_extended**.

Table 17-9 describes the extended employee information.

Table 17-9 Extended employee information

ID	Name	Mobile Phone Number	E-mail Addresses	Salary Currency	Salary	Tax Category	Work Place	Hiring Date
1	Wang	135 XXXX XXXX	xxxx@xx.com	R	8000.01	personal income tax&0.05	Country1:City1	2014
3	Tom	159 XXXX XXXX	xxxxx@xx.com.cn	D	1200.02	personal income tax&0.09	Country2:City2	2014

ID	Name	Mobile Phone Number	E-mail Addresses	Salary Currency	Salary	Tax Category	Work Place	Hiring Date
4	Jack	186 XXXX XXXX	xxxx@x x.org	D	2400 0.03	personal income tax&0.0 9	Country3 :City3	201 4
6	Linda	189 XXXX XXXX	xxxx@x xx.cn	D	3600 0.04	personal income tax&0.0 9	Country4 :City4	201 4
8	Zhang	134 XXXX XXXX	xxxx@x xxx.cn	R	9000 .05	personal income tax&0.0 5	Country5 :City5	201 4

Step 2 Analyze data.

For data analysis codes, see [Querying Data](#).

- Query contact information of employees whose salaries are paid in USD.
- Query the IDs and names of employees who were hired in 2014, and load query results to the partition with the hiring time of 2014 in **employees_info_extended**.
- Collect statistics for the number of records in the employees_info table.
- Query information about employees whose email addresses end with "cn".

Step 3 Submit a data analysis task to collect statistics for the number of records in the employees_info table.

For details about the implementation, see [Example Program Guide](#).

----End

17.3.2 Example Codes

17.3.2.1 Creating a Table

Function

This topic describes how to use Hive Query Language (HQL) to create internal and external tables. You can create a table in three modes:

- Define the table structure, and use the key word EXTERNAL to differentiate between internal and external tables.
 - If all data is to be processed by Hive, create an internal table. When an internal table is deleted, the metadata and data in the table are deleted.

- If data is to be processed by multiple tools (such as Pig), create an external table. When an external table is deleted, only the metadata is deleted.
- Create a table based on existing tables. Use CREATE LIKE to fully copy the original table structure, including the storage format.
- Create a table based on query results using CREATE AS SELECT.
In this mode, you can specify which fields are to be copied when copying the original table structure. The storage format is not copied.

 **NOTE**

Both the table name and field name can contain a maximum of 128 bytes. Both the field comment and value can contain a maximum of 4000 bytes. The key in **WITH SERDEPROPERTIES** can contain a maximum of 256 bytes.

Example Codes

```
-- Create an external table employees_info.
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info
(
  id INT,
  name STRING,
  usd_flag STRING,
  salary DOUBLE,
  deductions MAP<STRING, DOUBLE>,
  address STRING,
  entrytime STRING
)
-- Specify the field delimiter. Use delimited fields terminated by to specify the delimiter between
columns to a comma (.).
-- Use MAP KEYS TERMINATED BY to specify the delimiter between map keys to &.
ROW FORMAT delimited fields terminated by ',' MAP KEYS TERMINATED BY '&'
-- Set the storage format to TEXTFILE.
STORED AS TEXTFILE;

-- Create an external table employees_contact.
CREATE EXTERNAL TABLE IF NOT EXISTS employees_contact
(
  id INT,
  tel_phone STRING,
  email STRING
)
ROW FORMAT delimited fields terminated by ','
STORED AS TEXTFILE;

-- Create an external table employees_info_extended.
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info_extended(id INT, name STRING, usd_flag STRING,
salary DOUBLE, deductions MAP<STRING, DOUBLE>, address STRING)
-- A table may have one or multiple partitions. Each partition is saved as an independent folder in the table
directory. Partitions help minimize the query scope, accelerate data query, and allow users to manage data
based on certain criteria.
-- Use PARTITIONED BY to specify the column name and data type of the partition.
PARTITIONED BY(entrytime STRING)
ROW FORMAT delimited fields terminated by ',' MAP KEYS TERMINATED BY '&'
STORED AS TEXTFILE;
-- After a table is created, you can use ALTER TABLE to add or delete fields to or from the table, modify
table attributes, and add partitions.
-- Add the tel_phone and email fields to the employees_info_extended table.
ALTER TABLE employees_info_extended ADD COLUMNS (tel_phone STRING, email STRING);
```

17.3.2.2 Loading Data

Function

This topic describes how to use Hive Query Language (HQL) to load data to the existing **employees_info** table. You can learn how to load data from a local file system and MRS cluster. LOCAL is used to differentiate between local and non-local data sources.

Example Codes

```
-- Load the employee_info.txt file from the /opt/hive_examples_data/ directory of the local file system to
the employees_info table.
---- Overwrite the original data using the new data
LOAD DATA LOCAL INPATH '/opt/hive_examples_data/employee_info.txt' OVERWRITE INTO TABLE
employees_info;
---- Retain the original data and add the new data to the table.
LOAD DATA LOCAL INPATH '/opt/hive_examples_data/employee_info.txt' INTO TABLE employees_info;

-- Load /user/hive_examples_data/employee_info.txt from HDFS to the employees_info table.
---- Overwrite the original data using the new data
LOAD DATA INPATH '/user/hive_examples_data/employee_info.txt' OVERWRITE INTO TABLE
employees_info;
---- Retain the original data and add the new data to the table.
LOAD DATA INPATH '/user/hive_examples_data/employee_info.txt' INTO TABLE employees_info;
```

NOTE

Loading data is to copy data to a specified table in the Hadoop distributed file system (HDFS).

Sample Data

Data in **employees_info** is as follows:

```
1,Wang,R,8000.01,person&personal^Btype&income^Btax&0.05,Country1:City1,2014
3,Tom,D,12000.02,person&personal^Btype&income^Btax&0.09,Country2:City2,2014
4,Jack,D,24000.03,person&personal^Btype&income^Btax&0.05,Country3:City3,2014
6,Linda,D,36000.04,person&personal^Btype&income^Btax&0.05,Country4:City4,2014
8,Zhang,R,9000.05,person&personal^Btype&income^Btax&0.05,Country5:City5,2014
```

Data in **employees_contact** is as follows:

```
1,135 XXXX XXXX,xxxx@xx.com
3,159 XXXX XXXX,xxxxx@xx.com.cn
4,186 XXXX XXXX,xxxx@xx.org
6,189 XXXX XXXX,xxxx@xxx.cn
8,134 XXXX XXXX,xxxx@xxxx.cn
```

Data in **employees_info_extended** is as follows:

```
1,Wang,135 XXXX XXXX,xxxx@xx.com,R,
8000.01,person&personal^Btype&income^Btax&0.05,Country1:City1,2014
3,Tom,159 XXXX XXXX,xxxxx@xx.com.cn,D,
12000.02,person&personal^Btype&income^Btax&0.09,Country2:City2,2014
4,Jack,186 XXXX XXXX,xxxx@xx.org,D,
24000.03,person&personal^Btype&income^Btax&0.05,Country3:City3,2014
6,Linda,189 XXXX XXXX,xxxx@xxx.cn,D,
36000.04,person&personal^Btype&income^Btax&0.05,Country4:City4,2014
8,Zhang,134 XXXX XXXX,xxxx@xxxx.cn,R,
9000.05,person&personal^Btype&income^Btax&0.05,Country5:City5,2014
```

17.3.2.3 Querying Data

Function

This topic describes how to use Hive Query Language (HQL) to query and analyze data. You can query and analyze data using the following methods:

- Use common features for SELECT query, such as JOIN.
- Load data to a specified partition.
- Use Hive-provided functions.
- Query and analyze data using user-defined functions (UDFs). For details about how to create and define UDFs, see [UDF](#).

Example Codes

```
-- Query the contact information of employees whose salaries are paid in USD.
SELECT
a.name,
b.tel_phone,
b.email
FROM employees_info a JOIN employees_contact b ON(a.id = b.id) WHERE usd_flag='D';

-- Query the IDs and names of employees who were hired in 2014, and load query results to the partition
with the hiring time of 2014 in the employees_info_extended table.
INSERT OVERWRITE TABLE employees_info_extended PARTITION (entrytime = '2014')
SELECT
a.id,
a.name,
a.usd_flag,
a.salary,
a.deductions,
a.address,
b.tel_phone,
b.email
FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE a.entrytime = '2014';

-- Use the existing Hive function COUNT() to count the number of records in the employees_info table.
SELECT COUNT(*) FROM employees_info;

-- Query information about employees whose email addresses end with "cn".
SELECT a.name, b.tel_phone FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE
b.email like '%cn';
```

Extensions

- Configure intermediate Hive data encryption.
Set the table format to RCFile (recommended) or SequenceFile, and the encryption algorithm to ARC4Codec. SequenceFile is a unique Hadoop file format, and RCFile is a Hive file format with optimized column storage. When a big table is queried, RCFile provides higher performance than SequenceFile.

```
set hive.exec.compress.output=true;
set hive.exec.compress.intermediate=true;
set hive.intermediate.compression.codec=org.apache.hadoop.io.encrypted.arc4.ARC4Codec;
```

- For details about UDFs, see [UDF](#).

17.3.2.4 UDF

When internal functions of Hive cannot meet requirements, you can compile user-defined functions (UDFs) and use them for query.

According to implementation methods, UDFs are classified as follows:

- Common UDFs: used to perform operations on a single data row and export a single data row.
- User-defined aggregating functions (UDAFs): used to input multiple data rows and export a single data row.
- User-defined table-generating functions (UDTFs): used to perform operations on a single data row and export multiple data rows.

According to usage methods, UDFs are classified as follows:

- Temporary functions: used only in the current session and must be recreated after a session restarts.
- Permanent function: used in multiple sessions and do not have to be created every time a session restarts.

The following uses AddDoublesUDF as an example to describe how to compile and use UDFs.

Function

AddDoublesUDF is used to add two or multiple floating point values. The following example describes how to compile and use UDFs.

NOTE

- The normal UDF must be originated from **org.apache.hadoop.hive.ql.exec.UDF**.
- The normal UDF must implement at least one evaluate(). The evaluate function supports overloading.
- To develop a customized function, you need to add the **hive-exec-3.1.0.jar** dependency package to the project. The package can be obtained from the Hive installation directory.

Example Codes

The following is a UDF example:

```
package com.huawei.bigdata.hive.example.udf;
import org.apache.hadoop.hive.ql.exec.UDF;

public class AddDoublesUDF extends UDF {
    public Double evaluate(Double... a) {
        Double total = 0.0;
        // Processing logic
        for (int i = 0; i < a.length; i++)
            if (a[i] != null)
                total += a[i];
        return total;
    }
}
```

How to Use

- Step 1** Package the preceding program into **AddDoublesUDF.jar**, and upload it to a directory on the HDFS (such as **/user/hive_examples_jars/**). The user who creates the UDF and the user who uses the UDF function must have read right on this JAR file. Example statements:

```
hdfs dfs -put ./hive_examples_jars /user/hive_examples_jars
```

```
hdfs dfs -chmod 777 /user/hive_examples_jars
```

Step 2 Run the following command:

```
beeline -n Hive service user
```

Step 3 Define the function in Hive Server. Run the following SQL statement to create a permanent function:

```
CREATE FUNCTION addDoubles AS  
'com.huawei.bigdata.hive.example.udf.AddDoublesUDF' using jar 'hdfs://  
hacluster/user/hive_examples_jars/AddDoublesUDF.jar';
```

addDoubles indicates the function alias that is used for SELECT query.

Run the following statement to create a temporary function:

```
CREATE TEMPORARY FUNCTION addDoubles AS  
'com.huawei.bigdata.hive.example.udf.AddDoublesUDF' using jar 'hdfs://  
hacluster/user/hive_examples_jars/AddDoublesUDF.jar';
```

- *addDoubles* indicates the function alias that is used for SELECT query.
- TEMPORARY indicates that the function is used only in the current session with the Hive server.

Step 4 Run the following SQL statement to use the function in the Hive server:

```
SELECT addDoubles(1,2,3);
```

NOTE

*If an [Error 10011] error is displayed when you log in to the client again, run the **reload function;** command and then use this function.*

Step 5 Run the following SQL statement to delete the function from the Hive server:

```
DROP FUNCTION addDoubles;
```

```
----End
```

Extensions

None

17.3.2.5 Example Program Guide

Function

This section describes how to use an example program to complete an analysis task. An example program can submit a task by using the following methods:

- Submitting a data analysis task by using JDBC interfaces
- Submitting a data analysis task by using Python

Example Codes

- Submit a data analysis task using the Hive Java database connectivity (JDBC) interface, that is, JDBCExample.java.
 - a. Read the **property** file of the HiveServer client. The **hiveclient.properties** file is saved in the **resources** directory of the JDBC example program provided by Hive.

```
Properties clientInfo = null;
String userdir = System.getProperty("user.dir") + File.separator
+ "conf" + File.separator;
InputStream fileInputStream = null;
try{
clientInfo = new Properties();
//hiveclient.properties indicates the client configuration file. If the multiple instances feature is
used, the file must be replaced with the hiveclient.properties file on the instance client.
//hiveclient.properties is located under the config directory of the directory where the instance
client installation package is decompressed.
String hiveclientProp = userdir + "hiveclient.properties" ;
File propertiesFile = new File(hiveclientProp);
fileInputStream = new FileInputStream(propertiesFile);
clientInfo.load(fileInputStream);
}catch (Exception e) {
throw new IOException(e);
}finally{
if(fileInputStream != null){
fileInputStream.close();
fileInputStream = null;
}
}
```

- b. Obtain the IP address and port list of ZooKeeper, the cluster authentication mode, the SASL configuration of HiveServers, node names of HiveServers in ZooKeeper, the discovery mode from the client to the server, and the principal server process for user authentication. You can read all these configurations from the **hiveclient.properties** file.

```
//The format of zkQuorum is xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181";
//xxx.xxx.xxx.xxx is the IP address of the node where ZooKeeper resides. The default port is
2181.
zkQuorum = clientInfo.getProperty("zk.quorum");
auth = clientInfo.getProperty("auth");
sasL_qop = clientInfo.getProperty("sasL.qop");
zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
principal = clientInfo.getProperty("principal");
```

- c. In security mode, the kerberos user and keytab file path are required for login authentication. For details about how to obtain **USER_NAME**, **USER_KEYTAB_FILE**, and **KRB5_FILE**, see [Running JDBC and Viewing Results](#).

```
// Set the userName of new user.
USER_NAME = "xxx";
// Set the keytab and krb5 files location of client.
String userdir = System.getProperty("user.dir") + File.separator
+ "conf" + File.separator;
USER_KEYTAB_FILE = userdir + "user.keytab";
KRB5_FILE = userdir + "krb5.conf";
```

- d. Define HQL. HQL must be a single statement and cannot contain semicolons (;).

```
// Define HQL. HQL cannot contain ";"
String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT,name STRING)",
"SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};
```

- e. Build JDBC URL.

 NOTE

You can also implement pre-authentication without the need of providing the account and keytab file path. For details, see JDBC code example 2 in [Examples](#). If IBM JDK is used to run Hive applications, pre-authentication in JDBC sample code 2 must be implemented.

The following is an example of the JDBC URL composed of code snippets:

```
jdbc:hive2://xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181;/serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2;sasl.qop=auth-conf;auth=KERBEROS;principal=hive/hadoop.<system domain name>@<system domain name>;
```

You can login in to FusionInsight Manager, choose **System > Permission > Domain and Mutual Trust**, and check the value of **Local Domain**, which is the current system domain name.

hive/hadoop.<system domain name> is the user name. All letters in the system domain name contained in the user name of the system are lowercase letters. For example, if **Local domain** is set to **9427068F-6EFA-4833-B43E-60CB641E5B6C.COM**, the user is **hive/hadoop.9427068f-6efa-4833-b43e-60cb641e5b6c.com**.

```
// Concat JDBC URL
StringBuilder sBuilder = new StringBuilder(
    "jdbc:hive2://").append(zkQuorum).append("/");

if ("KERBEROS".equalsIgnoreCase(auth)) {
    sBuilder.append(";serviceDiscoveryMode=")
        .append(serviceDiscoveryMode)
        .append(";zooKeeperNamespace=")
        .append(zooKeeperNamespace)
        .append(";sasl.qop=")
        .append(sasl_qop)
        .append(";auth=")
        .append(auth)
        .append(";principal=")
        .append(principal)
        .append(";user.principal=")
        .append(USER_NAME)
        .append(";user.keytab=")
        .append(USER_KEYTAB_FILE)
        .append(";");
} else {
    // Normal mode
    sBuilder.append(";serviceDiscoveryMode=")
        .append(serviceDiscoveryMode)
        .append(";zooKeeperNamespace=")
        .append(zooKeeperNamespace)
        .append(";auth=none;");
}
String url = sBuilder.toString();
```

- f. Load the Hive JDBC driver.

```
// Load the Hive JDBC driver.
Class.forName(HIVE_DRIVER);
```

- g. Obtain the JDBC connection, confirm the HQL type (DDL/DML), call ports to run the HQL statement, return the queried column name and results to the console, and close the JDBC connection.

```
Connection connection = null;
try {
    // Obtain the JDBC connection.
    // If the normal mode is used, the second parameter needs to be set to a correct username.
    // Otherwise, the anonymous user will be used for login.
    connection = DriverManager.getConnection(url, "", "");
```

```

// Create a table
// To import data to a table after the table is created, you can use the LOAD statement. For
example, import data from the HDFS to the table.
//load data inpath '/tmp/employees.txt' overwrite into table employees_info;
execDDL(connection,sqls[0]);
System.out.println("Create table success!");

// Query
execDML(connection,sqls[1]);

// Delete the table
execDDL(connection,sqls[2]);
System.out.println("Delete table success!");
}
finally {
// Close the JDBC connection.
if (null != connection) {
connection.close();
}
}

public static void execDDL(Connection connection, String sql)
throws SQLException {
PreparedStatement statement = null;
try {
statement = connection.prepareStatement(sql);
statement.execute();
}
finally {
if (null != statement) {
statement.close();
}
}
}

public static void execDML(Connection connection, String sql) throws SQLException {
PreparedStatement statement = null;
ResultSet resultSet = null;
ResultSetMetaData resultMetaData = null;

try {
// Run the HQL statement.
statement = connection.prepareStatement(sql);
resultSet = statement.executeQuery();

// Return the queried column name to the console.
resultMetaData = resultSet.getMetaData();
int columnCount = resultMetaData.getColumnCount();
for (int i = 1; i <= columnCount; i++) {
System.out.print(resultMetaData.getColumnLabel(i) + '\t');
}
System.out.println();

// Return the results to the console.
while (resultSet.next()) {
for (int i = 1; i <= columnCount; i++) {
System.out.print(resultSet.getString(i) + '\t');
}
System.out.println();
}
}
finally {
if (null != resultSet) {
resultSet.close();
}

if (null != statement) {
statement.close();
}
}
}

```

```
}
}
}
```

- Submit a data analysis task using the Python interface, that is, **python-examples/pyCLI_nosec.py**.

- a. Import the HAConnection class.

```
from pyhs2.haconnection import HAConnection
```

- b. Declare the HiveServer IP address list. In this example, **hosts** indicate the nodes of HiveServer, and **xxx.xxx.xxx.xxx** indicates the business IP address.

```
hosts = ["xxx.xxx.xxx.xxx", "xxx.xxx.xxx.xxx"]
```

NOTE

1. If the HiveServer instance is migrated, the original sample program is invalid. You need to update the IP address of the HiveServer after the migration of the HiveServer instance used in the sample program.
 2. If multiple Hive instances are used, update the IP address based on the address of the instance that is actually connected.
- c. Set the third parameter of the **HAConnection** constructor to a correct username. The password can be left empty. Create a connection, run the HQL statement, and return the queried column name and results to the console.

```
try:
    with HAConnection(hosts = hosts,
                      port = 21066,
                      authMechanism = "PLAIN",
                      user='user1',
                      password='*****') as haConn:
        with haConn.getConnection() as conn:
            with conn.cursor() as cur:
                # Show databases
                print cur.getDatabases()
                # Execute query
                cur.execute("show tables")
                # Return column info from query
                print cur.getSchema()
                # Fetch table results
                for i in cur.fetch():
                    print i
except Exception, e:
    print e
```

NOTE

If multiple Hive instances are used, you need to modify hosts according to the description in **b** and change the port number based on to the actual port number. The default ports of Hive to Hive4 are 21066 to 21070, respectively.

17.3.2.6 Accessing Multiple ZooKeepers

Description

This section describes how to access both FusionInsight ZooKeeper and the third-party ZooKeeper in the same client process using the `testConnectHive` and `testConnectApacheZk` methods respectively.

In the `JDBCExample` class of the `hive-jdbc-example-multizk` package, the code structure of the main method is as follows:

```
public static void main(String[] args) throws InstantiationException, IllegalAccessException,
ClassNotFoundException, SQLException, IOException{
    testConnectHive();// Method of accessing FusionInsight ZooKeeper
    testConnectApacheZk();// Method of accessing the open-source ZooKeeper
}
```

Accessing FusionInsight ZooKeeper

If only the method of accessing FusionInsight ZooKeeper needs to be executed, comment out the `testConnectApacheZk` method in the `main` function.

Before using the `testConnectHive` method to access FusionInsight ZooKeeper, perform the following operations:

- Step 1** Change the value of `USER_NAME` in the `init` method in `JDBCExample`. `USER_NAME` indicates the user that is used to access FusionInsight ZooKeeper and has permissions of the FusionInsight Hive and Hadoop common user groups.
- Step 2** Go to the client decompression path `FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles\Hive\config` and manually import the `core-site.xml` and `hiveclient.properties` files to the `hive-jdbc-example-multizk\src\main\resources` directory of the sample project.
- Step 3** Check and change the values of `zk.port` and `zk.quorum` in the `hiveclient.properties` file in the `resources` directory.
 - **zk.port**: indicates the port for accessing FusionInsight ZooKeeper. Generally, the default value is used. Change the value as required.
 - **zk.quorum**: indicates the IP address for accessing ZooKeeper quorumpeer. Set it to the IP address of the cluster deployed with the FusionInsight ZooKeeper service.

----End

Accessing Open-Source ZooKeeper

To use `testConnectApacheZk` to connect to the open-source ZooKeeper code, change `xxx.xxx.xxx.xxx` in the following code to the IP address of the open-source ZooKeeper to be connected. Change the port number as required. If only the sample for accessing the third-party ZooKeeper needs to be executed, comment out the `testConnectHive` method in the `main` function.

```
digestZK = new org.apache.zookeeper.ZooKeeper("xxx.xxx.xxx.xxx:2181", 60000, null);
```

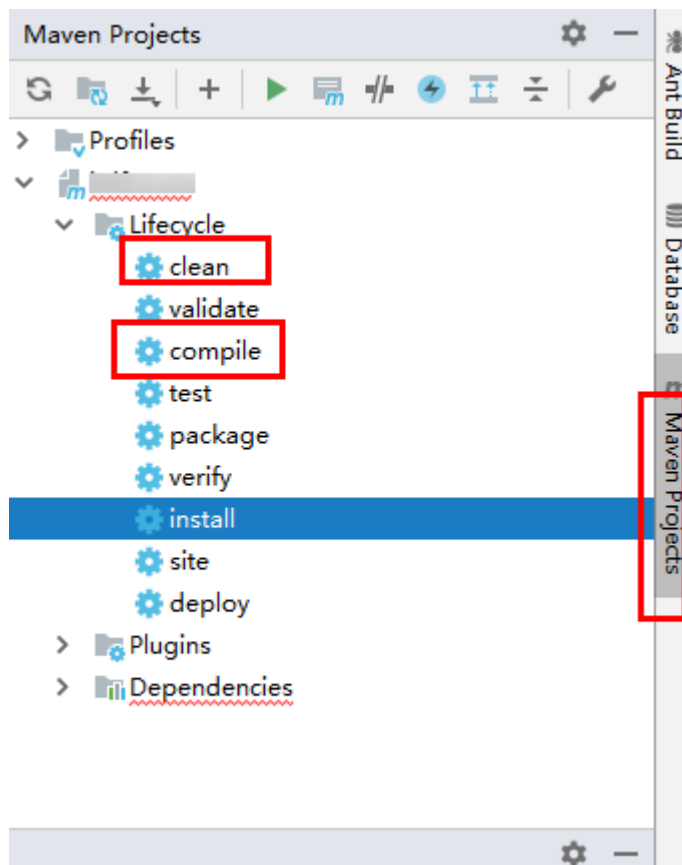
17.4 Commissioning Applications

17.4.1 Running JDBC and Viewing Results

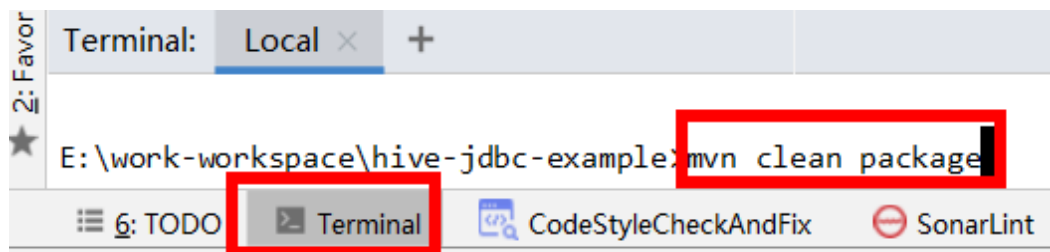
Running JDBC in CLI Mode

- Step 1** On the right of the IntelliJ IDEA home page, click **Maven Projects**. On the **Maven Projects** page, choose *Project name* > **Lifecycle** and run the **clean** and **compile** scripts.

Figure 17-6 Maven Projects page



- Step 2** In the lower left corner of the IDEA page, click **Terminal** to access the terminal. Run the **mvn clean package** command to compile the package.



If **BUILD SUCCESS** is displayed, the compilation is successful, as shown in the following figure. A JAR file containing the **-with-dependencies** field is generated in the **target** directory of the sample project.

```
Terminal: Local x +
[INFO] com/ already added, skipping
[INFO] com/huawei/ already added, skipping
[INFO] com/huawei/bigdata/ already added, skipping
[INFO] META-INF/maven/ already added, skipping
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 32.933 s
[INFO] Finished at: 2020-11-23T16:18:08+08:00
[INFO] -----
```

Step 3 Create a directory on Windows or Linux as the running directory, for example, **D:\jdbc_example** (Windows) or **/opt/jdbc_example** (Linux). Place the JAR file whose name contains **-with-dependencies** in the **target** directory generated in **Step 2** to this directory. Create the **src/main/resources** subdirectory in the directory. Copy all files in the **resources** directory of the **jdbc-examples** project to the **resources** directory.

Step 4 In Windows, run the following command:

```
cd /d d:\jdbc_example
java -jar hive-jdbc-example-1.0-SNAPSHOT-jar-with-dependencies.jar
```

In Linux, run the following command:

```
chmod +x /opt/jdbc_example -R
cd /opt/jdbc_example
java -jar hive-jdbc-example-1.0-SNAPSHOT-jar-with-dependencies.jar
```

 **NOTE**

The preceding JAR file names are for reference only. The actual names may vary.

Step 5 In the CLI, view the HQL query results in the example codes.

The following information is displayed if the sample project is successful in Windows:

```
Create table success!
_c0
0
Delete table success!
```

The following information is displayed if the sample project is successful in Linux:

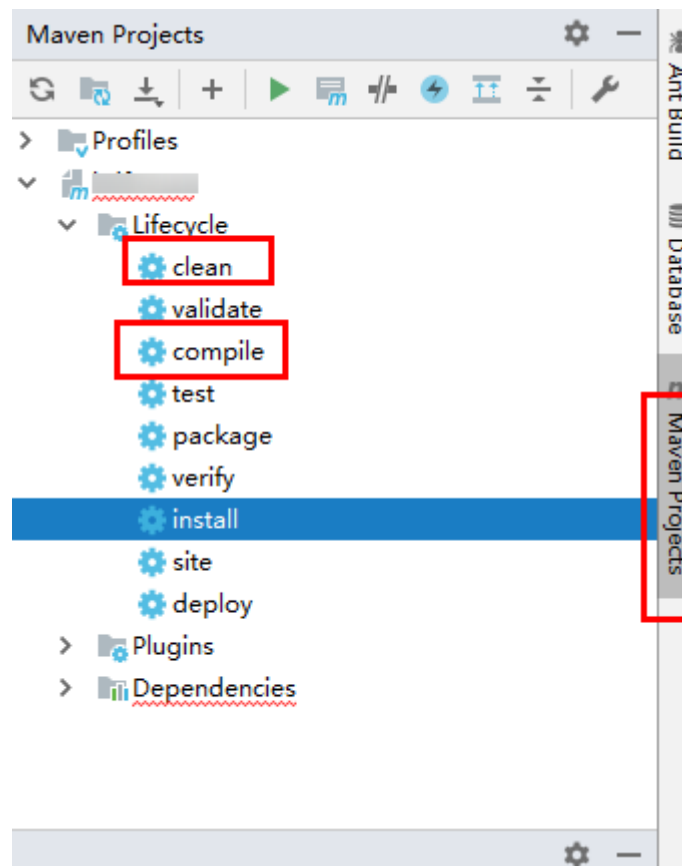
```
Create table success!
_c0
0
Delete table success!
```

----End

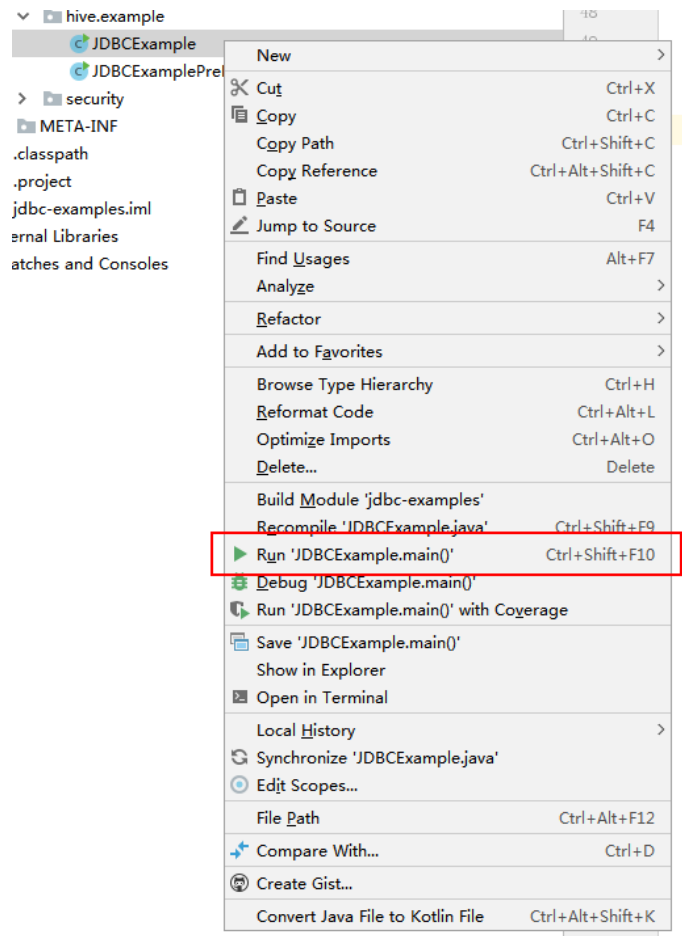
Running JDBC in IntelliJ IDEA Mode

Step 1 On the right of the IntelliJ IDEA home page, click **Maven Projects**. On the **Maven Projects** page, choose *Project name* > **Lifecycle** and run the **clean** and **compile** scripts.

Figure 17-7 Maven Projects page



Step 2 Right-click the JDBCExample class in the IntelliJ IDEA jdbc-examples project, and choose **Run JDBCExample.main()** from the shortcut menu. As shown in the following figure.



Step 3 In the IntelliJ IDEA output window, view the HQL query results in the example codes.

```
Create table success!  
_c0  
0  
Delete table success!
```

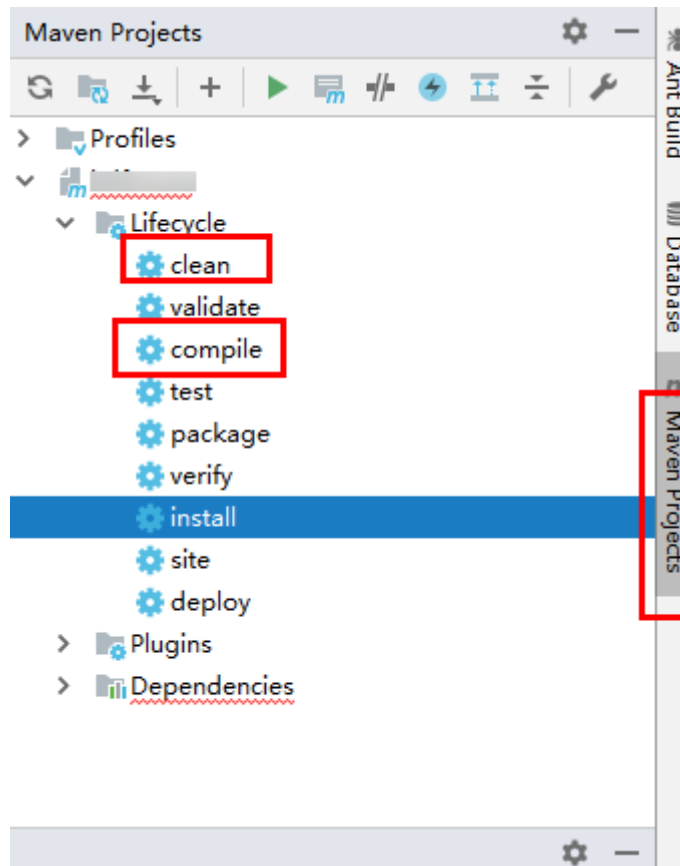
----End

17.4.2 Running HCatalog and Viewing Results

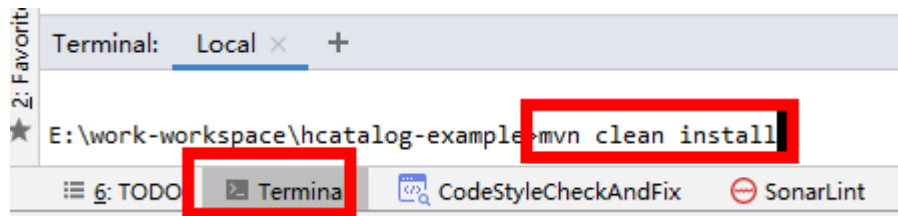
Running HCatalog Example Projects

Step 1 On the right of the IntelliJ IDEA home page, click **Maven Projects**. On the **Maven Projects** page, choose *Project name* > **Lifecycle** and run the **clean** and **compile** scripts.

Figure 17-8 Maven Projects page



Step 2 In the lower left corner of the IDEA page, click **Terminal** to access the terminal. Run the **mvn clean install** command to compile the package.



If **BUILD SUCCESS** is displayed, the compilation is successful, as shown in the following figure. The **hcatalog-example-*.jar** package is generated in the **target** directory of the sample project.

```
Terminal: Local x +
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ hcatalog-example ---
[INFO] Building jar: E:\other-workspase\sample_project\src\hive-examples\hca
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ hcatalog-exi
[INFO] Installing E:\other-workspase\sample_project\src\hive-examples\hcata:
[INFO] Installing E:\other-workspase\sample_project\src\hive-examples\hcata:
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.916 s
[INFO] Finished at: 2020-11-23T16:25:57+08:00
[INFO] -----
```

 **NOTE**

The preceding JAR file names are for reference only. The actual names may vary.

- Step 3** Upload the **hcatalog-example-*.jar** file generated in the **target** directory in the previous step to the specified directory on Linux, for example, **/opt/hive_client**, marked as **\$HCAT_CLIENT**, and ensure that the Hive and YARN clients have been installed. Execute environment variables for the HCAT_CLIENT to take effect.

```
export HCAT_CLIENT=/opt/hive_client
```

- Step 4** Run the following command to configure environment parameters (client installation path **/opt/client** is used as an example):

```
export HADOOP_HOME=/opt/client/HDFS/hadoop
export HIVE_HOME=/opt/client/Hive/Beeline
export HCAT_HOME=$HIVE_HOME/./HCatalog
export LIB_JARS=$HCAT_HOME/lib/hive-hcatalog-core-xxx.jar,$HCAT_HOME/lib/hive-metastore-xxx.jar,
$HCAT_HOME/lib/hive-standalone-metastore-xxx.jar,$HIVE_HOME/lib/hive-exec-xxx.jar,$HCAT_HOME/lib/
libfb303-xxx.jar,$HCAT_HOME/lib/slf4j-api-xxx.jar,$HCAT_HOME/lib/jdo-api-xxx.jar,$HCAT_HOME/lib/antlr-
runtime-xxx.jar,$HCAT_HOME/lib/datanucleus-api-jdo-xxx.jar,$HCAT_HOME/lib/datanucleus-core-xxx.jar,
$HCAT_HOME/lib/datanucleus-rdbms-fi-xxx.jar,$HCAT_HOME/lib/log4j-api-xxx.jar,$HCAT_HOME/lib/log4j-
core-xxx.jar,$HIVE_HOME/lib/commons-lang-xxx.jar
export HADOOP_CLASSPATH=$HCAT_HOME/lib/hive-hcatalog-core-xxx.jar:$HCAT_HOME/lib/hive-
metastore-xxx.jar:$HCAT_HOME/lib/hive-standalone-metastore-xxx.jar:$HIVE_HOME/lib/hive-exec-xxx.jar:
$HCAT_HOME/lib/libfb303-xxx.jar:$HADOOP_HOME/etc/hadoop:$HCAT_HOME/conf:$HCAT_HOME/lib/
slf4j-api-xxx.jar:$HCAT_HOME/lib/jdo-api-xxx.jar:$HCAT_HOME/lib/antlr-runtime-xxx.jar:$HCAT_HOME/lib/
datanucleus-api-jdo-xxx.jar:$HCAT_HOME/lib/datanucleus-core-xxx.jar:$HCAT_HOME/lib/datanucleus-
rdbms-fi-xxx.jar:$HCAT_HOME/lib/log4j-api-xxx.jar:$HCAT_HOME/lib/log4j-core-xxx.jar:$HIVE_HOME/lib/
commons-lang-xxx.jar
```

 **NOTE**

- xxx: Indicates the version number of the JAR package. **Change the version numbers of the JAR files specified in LIB_JARS and HADOOP_CLASSPATH based on the actual environment.**
- If the multi-instance function is enabled for Hive, perform the configuration in **export HIVE_HOME=/opt/client/Hive/Beeline**. For example, if Hive 1 is used, ensure that the Hive 1 client has been installed before using it. Change the value of **export HIVE_HOME** to **/opt/client/Hive1/Beeline**.

- Step 5** Prepare for the running:

1. Use the Hive client to create source table t1 in beeline: **create table t1(col1 int);**

Insert the following data into t1:

```
+-----+
| t1.col1 |
+-----+
| 1      |
| 1      |
| 1      |
| 2      |
| 2      |
| 3      |
```

2. Create destination table t2: **create table t2(col1 int,col2 int);**

 **NOTE**

Tables created in this sample project use the default storage format of Hive. Currently, tables whose storage format is ORC are not supported.

- Step 6** Use the Yarn client to submit tasks:

```
yarn --config $HADOOP_HOME/etc/hadoop jar $HCAT_CLIENT/hcatalog-
example-1.0-SNAPSHOT.jar com.huawei.bigdata.HCatalogExample -libjars
$LIB_JARS t1 t2
```

- Step 7** View the running result. The data in t2 is as follows:

```
0: jdbc:hive2://192.168.1.18:2181,192.168.1.> select * from t2;
+-----+-----+
| t2.col1 | t2.col2 |
+-----+-----+
| 1      | 3      |
| 2      | 2      |
| 3      | 1      |
+-----+-----+
```

----End

17.4.3 Running Python and Viewing Results

Running Python in CLI Mode

- Step 1** Grant the execution permission for the script of **python-examples** folder. Run the following command in the CLI:

```
chmod +x python-examples -R
```

- Step 2** Enter the service plane IP address of the node where HiveServer is installed in the hosts array of **python-examples/pyCLI_nosec.py**.

 **NOTE**

When the multi-instance is enabled: In **python-examples/pyCLI_nosec.py** the hosts array must be modified. In addition, the port must be set according to the used instance. The port (**hive.server2.thrift.port**) is used for Hive to provide the Thrift service.

- Step 3** Run the following command:

```
cd python-examples
python pyCLI_nosec.py
```

- Step 4** In the CLI, view the HQL query results in the example codes.

For example:

```
[[['default', "]]  
[{'comment': 'from deserializer', 'columnName': 'tab_name', 'type': 'STRING_TYPE'}]  
['xx']
```

NOTE

If the following exception occurs:

```
importError: libsas2.so.2: cannot open shared object file: No such file or directory
```

You can handle the problem as follows:

1. Run the following command to check the LibSASL version in the installed operating system.

```
ldconfig -p|grep sasl
```

If the following is displayed, the current operating system only has the 3.x version.

```
libsas2.so.3 (libc6,x86-64) => /usr/lib64/libsas2.so.3  
libsas2.so.3 (libc6) => /usr/lib/libsas2.so.3
```

2. If only the 3. x version exists, run the following command to create a soft link.

```
ln -s /usr/lib64/libsas2.so.3.0.0 /usr/lib64/libsas2.so.2
```

----End

17.4.4 Running Python3 and Viewing Results

Running Python in CLI Mode

- Step 1** Grant the execution permission for the script of **python3-examples** folder. Run the following command in the CLI:

```
chmod +x python3-examples -R.
```

- Step 2** In **python3-examples/pyCLI_nosec.py**, change the value of host to the service plane IP address of the node where HiveServer is installed, and change the value of port to the port (**hive.server2.thrift.port**) used by Hive to provide the Thrift service. The default value is 21066.

NOTE

When the multi-instance is enabled: In **python3-examples/pyCLI_nosec.py**, the hosts must be modified. In addition, the port must be set according to the used instance. The port (**hive.server2.thrift.port**) is used for Hive to provide the Thrift service.

- Step 3** Run the following commands to start the Python3 client:

```
cd python3-examples
```

```
python pyCLI_nosec.py
```

- Step 4** In the CLI, view the HQL query results in the example codes. For example:

```
('table_name1',)  
('table_name2',)  
('table_name3',)  
('table_name4',)  
('table_name5',)
```

In the preceding command, **table_nameX** indicates the actual table name.

----End

17.5 More Information

17.5.1 Interface Reference

17.5.1.1 JDBC

The Hive Java database connectivity (JDBC) interface complies with the Java JDBC driver standard.

 **NOTE**

As a data warehouse, Hive does not support all JDBC APIs. For example, if transactional operations, such as rollback and setAutoCommit, are performed, SQL exceptions like **Method not supported** will occur.

17.5.1.2 Hive SQL

Hive SQL supports all features of Hive-3.1.0. For details, see <https://cwiki.apache.org/confluence/display/hive/languagemanual>.

Table 17-10 describes the extended Hive statements provided by FusionInsight.

Table 17-10 Extended Hive statements

Extended Syntax	Syntax Description	Syntax Example	Example Description
<pre>CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_ name (col_name data_type [COMMENT col_comment], ...) [ROW FORMAT row_format] [STORED AS file_format] STORED BY 'storage.handler.cl ass.name' [WITH SERDEPROPERTIE S (...)] [TBLPROPERTIES ("group1d"=" group1 ","locator1d"="loc ator1")] ...;</pre>	<p>The statement is used to create a Hive table and specify locators on which table data files locate. For details, see Using HDFS Colocation to Store Hive Tables.</p>	<pre>CREATE TABLE tab1 (id INT, name STRING) row format delimited fields terminated by '\t' stored as RCFILE TBLPROPERTIES(" group1d"=" group1 ","locator1d"="loc ator1");</pre>	<p>The statement is used to create table tab1 and specify locator1 on which the table data of tab1 locates.</p>

Extended Syntax	Syntax Description	Syntax Example	Example Description
<pre>CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name (col_name data_type [COMMENT col_comment], ...) [ROW FORMAT row_format] [STORED AS file_format] STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)] ... [TBLPROPERTIES ('column.encode.columns'='col_name1,col_name2' 'column.encode.indices'='col_id1,col_id2', 'column.encode.classname'='encode_classname')]...;</pre>	<p>The statement is used to create a hive table and specify the table encryption column and encryption algorithm. For details, see Using the Hive Column Encryption.</p>	<pre>create table encode_test(id INT, name STRING, phone STRING, address STRING) ROW FORMAT SERDE 'org.apache.hadoop p.hive.serde2.lazy. LazySimpleSerDe' WITH SERDEPROPERTIES S ('column.encode.i ndices'='2,3', 'column.encode.cl assname'='org.apa che.hadoop.hive.s erde2.SMS4Rewrit er') STORED AS TEXTFILE;</pre>	<p>The statement is used to create table encode_test and specify that column 2 and column 3 will be encrypted using the org.apache.hadoop.hive.serde2.SMS4Rewriter encryption algorithm class during data insertion.</p>
<pre>REMOVE TABLE hbase_tablename [WHERE where_condition];</pre>	<p>The statement is used to delete data that meets criteria from the Hive on HBase table. For details, see Deleting Single-row Records from Hive on HBase.</p>	<pre>remove table hbase_table1 where id = 1;</pre>	<p>The statement is used to delete data that meets the criterion of "id = 1" from the table.</p>

Extended Syntax	Syntax Description	Syntax Example	Example Description
<pre>CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_ name (col_name data_type [COMMENT col_comment], ...) [ROW FORMAT row_format] STORED AS inputformat 'org.apache.hado op.hive.contrib.fil eformat.Specifie dDelimiterInput- Format' outputformat 'org.apache.hadoo p.hive ql.io.HiveIlg noreKeyTextOutpu tFormat';</pre>	<p>The statement is used to create a hive table and specify that the table supports customized row delimiters. For details, see Customizing Row Separators.</p>	<pre>create table blu(time string, num string, msg string) row format delimited fields terminated by ',' stored as inputformat 'org.apache.hado op.hive.contrib.fil eformat.Specifie dDelimiterInput- Format' outputformat 'org.apache.hadoo p.hive ql.io.HiveIlg noreKeyTextOutpu tFormat';</pre>	<p>The statement is used to create table blu and set inputformat to SpecifiedDelimiterInputFormat so that the query row delimiter can be specified during the query.</p>

17.5.1.3 WebHCat

 NOTE

- The following uses the service IP address of WebHCat and the WebHCat HTTP port configured during the installation as an example.
- The **user.name** parameter needs to be added to APIs except **:version**, **status**, **version**, **version/hive**, and **version/hadoop**.

1. :version(GET)

- Description

Obtain the list of the response types supported by WebHCat.

- URL

http://www.myserver.com/templeton/:version

- Parameter

Parameter	Description
:version	WebHCat version number (Currently this must be v1.)

- Returned result

Parameter	Description
responseTypes	A list of all supported response types

- Example
`curl -i -u : --negotiate 'http://10.64.35.144:9111/templeton/v1'`

2. status (GET)

- Description
Obtain the status of the current server.
- URL
`http://www.myserver.com/templeton/v1/status`
- Parameter
None
- Returned result

Parameter	Description
status	OK is returned if the WebHCat server was contacted.
version	String containing the version number similar to "v1"

- Example
`curl -i -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/status'`

3. version (GET)

- Description
Obtain the WebHCat version of the server.
- URL
`http://www.myserver.com/templeton/v1/version`
- Parameter
None
- Returned result

Parameter	Description
supportedVersions	A list of all supported versions
version	The current version

- Example
`curl -i -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/version'`

4. version/hive (GET)

- Description
Obtain the Hive version of the server.

- URL
http://www.myserver.com/templeton/v1/version/hive
- Parameter
None
- Returned result

Parameter	Description
module	hive
version	Hive version

- Example
curl -i -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/version/hive'

5. version/hadoop (GET)

- Description
Obtain the Hadoop version of the server.
- URL
http://www.myserver.com/templeton/v1/version/hadoop
- Parameter
None
- Returned result

Parameter	Description
module	hadoop
version	Hadoop version

- Example
curl -i -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/version/hadoop'

6. ddl (POST)

- Description
Execute a DDL statement.
- URL
http://www.myserver.com/templeton/v1/ddl
- Parameter

Parameter	Description
exec	The HCatalog ddl string to execute
group	The user group to use when creating a table

Parameter	Description
permissions	The permissions string to use when creating a table. The format is "rwxrw-r-x".

- Returned result

Parameter	Description
stdout	A string containing the result HCatalog sent to standard out (possibly empty)
stderr	A string containing the result HCatalog sent to standard error (possibly empty)
exitcode	The exitcode HCatalog returned

- Example

```
curl -i -u : --negotiate -d exec="show tables" 'http://10.64.35.144:9111/templeton/v1/ddl?user.name=user1'
```

7. ddl/database (GET)

- Description

List all databases.

- URL

<http://www.myserver.com/templeton/v1/ddl/database>

- Parameter

Parameter	Description
like	List only databases whose names match the specified pattern.

- Returned result

Parameter	Description
databases	A list of database names

- Example

```
curl -i -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/ddl/database?user.name=user1'
```

8. ddl/database/:db (GET)

- Description

Obtain details about a specified database.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db>

- Parameter

Parameter	Description
:db	The database name

- Returned result

Parameter	Description
location	The database location
comment	The database comment. If there is no database comment, the value is null.
database	The database name
owner	The database owner
owertype	The type of the database owner

- Example

```
curl -i -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/ddl/database/default?user.name=user1'
```

9. ddl/database/:db (PUT)

- Description

Create a database.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db>

- Parameter

Parameter	Description
:db	The database name
group	The user group to use
permission	The permissions string to use
location	The database location
comment	A comment for the database, like a description
properties	The database properties

- Returned result

Parameter	Description
database	The database name

- Example

```
curl -i -u : --negotiate -X PUT -HContent-type:application/json -d '{"location": "/tmp/a", "comment": "my db", "properties": {"a": "b"}}' 'http://10.64.35.144:9111/templeton/v1/ddl/database/db?user.name=user1'
```

10. ddl/database/:db (DELETE)

- Description

Delete a database.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db>

- Parameter

Parameter	Description
:db	The database name
ifExists	Hive returns an error if the database specified does not exist, unless ifExists is set to true.
option	Parameter set to either cascade or restrict . restrict will remove the schema if all the tables are empty. cascade removes everything including data and definitions.

- Returned result

Parameter	Description
database	The database name

- Example

```
curl -i -u : --negotiate -X DELETE 'http://10.64.35.144:9111/templeton/v1/ddl/database/db3?ifExists=true?user.name=user1'
```

11. ddl/database/:db/table (GET)

- Description

List all tables in a database.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table>

- Parameter

Parameter	Description
:db	The database name
like	List only tables whose names match the specified pattern.

- Returned result

Parameter	Description
database	A list of table names
tables	The database name

- Example

```
curl -i -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/ddl/database/default/table?user.name=user1'
```

12. ddl/database/:db/table/:table (GET)

- Description

Obtain details of a table.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table>

- Parameter

Parameter	Description
:db	The database name
:table	The table name
format	Set "format=extended" to see additional information (using "show table extended like").

- Returned result

Parameter	Description
columns	A list of column names and types
database	The database name
table	The table name
partitioned	Indicates whether a table is a partition table. This parameter is available only when the table format is extended .
location	Location of a table. This parameter is available only when the table format is extended .
outputformat	Output format. This parameter is available only when the table format is extended .
inputformat	Input format. This parameter is available only when the table format is extended.

Parameter	Description
owner	Owner of a table. This parameter is available only when the table format is extended.
partitionColumns	Partition column. This parameter is available only when the table format is extended.

- Example

```
curl -i -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1?format=extended&user.name=user1'
```

13. ddl/database/:db/table/:table (PUT)

- Description

Create a table.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table>

- Parameter

Parameter	Description
:db	The database name
:table	The new table name
group	The user group to use when creating a table
permissions	The permissions string to use when creating a table
external	Allows you to specify a location so that Hive does not use the default location for this table.
ifNotExists	If the value true , you will not receive an error if the table already exists.
comment	Comment for the table
columns	A list of column descriptions, including name, type, and an optional comment
partitionedBy	A list of column descriptions used to partition the table. Like the columns parameter this is a list of name, type, and comment fie.

Parameter	Description
clusteredBy	Bucket column description, including the columnNames , sortedBy , and numberOfBuckets parameters. The columnNames parameter includes columnName and sorting sequence (ASC for ascending or DESC for descending).
format	Storage format description including parameters for rowFormat , storedAs , and storedBy .
location	The HDFS path
tableProperties	A list of table property names and values (key/value pairs).

- Returned result

Parameter	Description
database	The new table name
table	The database name

- Example

```
curl -i -u : --negotiate -X PUT -HContent-type:application/json -d '{"columns": [{"name": "id", "type": "int"}, {"name": "name", "type": "string"}], "comment": "hello", "format": {"storedAs": "orc"} }' 'http://10.64.35.144:9111/templeton/v1/ddl/database/db3/table/tbl1?user.name=user1'
```

14. ddl/database/:db/table/:table (POST)

- Description

Rename a table.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table>

- Parameter

Parameter	Description
:db	The database name
:table	The existing (old) table name
rename	The new table name

- Returned result

Parameter	Description
database	The database name
table	The new table name

– Example

```
curl -i -u : --negotiate -d rename=table1 'http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/tbl1?user.name=user1'
```

15. ddl/database/:db/table/:table (DELETE)

– Description

Delete a table.

– URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table>

– Parameter

Parameter	Description
:db	The database name
:table	The table name
ifExists	If the value is true , you will not receive an error.

– Returned result

Parameter	Description
database	The database name
table	The table name

– Example

```
curl -i -u : --negotiate -X DELETE 'http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/table2?ifExists=true&user.name=user1'
```

16. ddl/database/:db/table/:existingtable/like/:newtable (PUT)

– Description

Create a table that is the same as an existing one.

– URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:existingtable/like/:newtable>

– Parameter

Parameter	Description
:db	The database name
:existingtable	The existing table name

Parameter	Description
:newtable	The new table name
group	The user group to use when creating a table
permissions	The permissions string to use when creating a table
external	Allows you to specify a location so that Hive does not use the default location for this table.
ifNotExists	If the value is true , you will not receive an error if the table already exists.
location	The HDFS path

- Returned result

Parameter	Description
database	The database name
table	The new table name

- Example

```
curl -i -u : --negotiate -X PUT -HContent-type:application/json -d '{"ifNotExists": "true"}' 'http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/like/tt1?user.name=user1'
```

17. ddl/database/:db/table/:table/partition(GET)

- Description

List partition information of a table.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition>

- Parameter

Parameter	Description
:db	The database name
:table	The table name

- Returned result

Parameter	Description
database	The database name
table	The table name

Parameter	Description
partitions	A list of partition values and of partition names

- Example

```
curl -i -u : --negotiate http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/x1/partition?user.name=user1
```

18. ddl/database/:db/table/:table/partition/:partition(GET)

- Description

List information about a partition of a table.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition/:partition>

- Parameter

Parameter	Description
:db	The database name
:table	The table name
:partition	The partition name, col_name='value' list. Be careful to properly encode the quote for http, for example, "country=%27algeria%27".

- Returned result

Parameter	Description
database	The database name
table	The table name
partition	The partition name
partitioned	True if the table is partitioned
location	Location of table
outputFormat	Output format
columns	List of column names, types, and comments
owner	The owner's user name
partitionColumns	List of the partition columns
inputFormat	Input format
totalNumberFiles	Number of files in a partition

Parameter	Description
totalFileSize	Total file size in a partition
maxFileSize	Maximum file size
minFileSize	Minimum file size
lastAccessTime	Last access time
lastUpdateTime	Last update time

- Example

```
curl -i -u : --negotiate http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/x1/partition/dt=1?user.name=user1
```

19. ddl/database/db/table/:table/partition/:partition(PUT)

- Description

Add a table partition.

- URL

http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition/:partition

- Parameter

Parameter	Description
:db	The database name
:table	The table name
group	The user group to use
permissions	The permissions string to use
location	The location for partition creation
ifNotExists	If the value is true , return an error if the partition already exists.

- Returned result

Parameter	Description
database	The database name
table	The table name
partitions	The partition name

- Example

```
curl -i -u : --negotiate -X PUT -HContent-type:application/json -d '{} ' http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/x1/partition/dt=10?user.name=user1
```

20. ddl/database/db/table/:table/partition/:partition(DELETE)

- Description

Delete a table partition.

- URL

`http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition/:partition`

- Parameter

Parameter	Description
:db	The database name
:table	The table name
group	The user group to use
permissions	The permissions string to use. The format is "rwxrw-r-x".
ifExists	Hive returns an error if the partition specified does not exist, unless ifExists is set to true.

- Returned result

Parameter	Description
database	The database name
table	The table name
partitions	The partition name

- Example

```
curl -i -u : --negotiate -X DELETE -HContent-type:application/json -d '{} ' http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/x1/partition/dt=10?user.name=user1
```

21. ddl/database/:db/table/:table/column(GET)

- Description

Obtain the column list of a table.

- URL

`http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/column`

- Parameter

Parameter	Description
:db	The database name
:table	The table name

- Returned result

Parameter	Description
database	The database name
table	The table name
columns	A list of column names and types

- Example

```
curl -i -u : --negotiate http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/column?user.name=user1
```

22. ddl/database/:db/table/:table/column/:column(GET)

- Description

Obtain details about a column in a table.

- URL

http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/column/:column

- Parameter

Parameter	Description
:db	The database name
:table	The table name
:column	The column name

- Returned result

Parameter	Description
database	The database name
table	The table name
column	Column name and type

- Example

```
curl -i -u : --negotiate http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/column/id?user.name=user1
```

23. ddl/database/:db/table/:table/column/:column(PUT)

- Description

Add a column to a table.

- URL

http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/column/:column

- Parameter

Parameter	Description
:db	The database name
:table	The table name
:column	The column name
type	The type of column to add, like "string" or "int"
comment	The column comment, like a description

- Returned result

Parameter	Description
database	The database name
table	The table name
column	The column name

- Example

```
curl -i -u : --negotiate -X PUT -HContent-type:application/json -d '{"type": "string", "comment": "new column"}' http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/column/name?user.name=user1
```

24. ddl/database/:db/table/:table/property(GET)

- Description

Obtain table properties.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/property>

- Parameter

Parameter	Description
:db	The database name
:table	The table name

- Returned result

Parameter	Description
database	The database name
table	The table name
properties	Property list

- Example

```
curl -i -u : --negotiate http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/property?user.name=user1
```

25. ddl/database/:db/table/:table/property/:property(GET)

- Description

Obtain a specified property value of a table.

- URL

http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/property/:property

- Parameter

Parameter	Description
:db	The database name
:table	The table name
:property	The property name

- Returned result

Parameter	Description
database	The database name
table	The table name
property	Property list

- Example

```
curl -i -u : --negotiate http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/property/last_modified_by?user.name=user1
```

26. ddl/database/:db/table/:table/property/:property(PUT)

- Description

Add a property value for a table.

- URL

http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/property/:property

- Parameter

Parameter	Description
:db	The database name
:table	The table name
:property	The property name
value	The property value

- Returned result

Parameter	Description
database	The database name
table	The table name
property	The property name

- Example

```
curl -i -u : --negotiate -X PUT -HContent-type:application/json -d '{"value": "my value"}' http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/property/mykey?user.name=user1
```

27. mapreduce/jar(POST)

- Description

Execute an MR task. Before the execution, upload MR JAR packages to HDFS.

- URL

<http://www.myserver.com/templeton/v1/mapreduce/jar>

- Parameter

Parameter	Description
jar	Name of the jar file for Map Reduce to use
class	Name of the class for Map Reduce to use
libjars	Comma-separated jar files to include in the classpath
files	Comma-separated files to be copied to the map reduce cluster
arg	Main class input parameter
define	Set a Hadoop configuration variable using the syntax define=NAME=VALUE.
statusdir	A directory where WebHCat will write the status of the Map Reduce job. If provided, it is the caller's responsibility to remove this directory when done.

Parameter	Description
enablelog	<p>If statusdir is set and enablelog is true, collect Hadoop job configuration and logs in to a directory named \$statusdir/logs after the job finishes. Both completed and failed attempts are logged. The layout of subdirectories in \$statusdir/logs is:</p> <ul style="list-style-type: none"> logs/\$job_id (directory for \$job_id) logs/\$job_id/job.xml.html logs/\$job_id/\$attempt_id (directory for \$attempt_id) logs/\$job_id/\$attempt_id/stderr logs/\$job_id/\$attempt_id/stdout logs/\$job_id/\$attempt_id/syslog <p>This parameter supports only Hadoop 1.X.</p>
callback	<p>Define a URL to be called upon job completion. You may embed a specific job ID into this URL using \$jobId. This tag will be replaced in the callback URL with this job's job ID.</p>

- Returned result

Parameter	Description
id	<p>A string containing the job ID similar to "job_201110132141_0001"</p>

- Example

```
curl -i -u : --negotiate -d jar="/tmp/word.count-0.0.1-SNAPSHOT.jar" -d class=com.huawei.word.count.WD -d statusdir="/output" "http://10.64.35.144:210559111/templeton/v1/mapreduce/jar?user.name=user1"
```

28. mapreduce/streaming(POST)

- Description

Submit an MR task in Streaming mode.

- URL

<http://www.myserver.com/templeton/v1/mapreduce/streaming>

- Parameter

Parameter	Description
input	Location of the input data in Hadoop
output	Location in which to store the output data. If not specified, WebHCat will store the output in a location that can be discovered using the queue resource.
mapper	Location of the mapper program in Hadoop
reducer	Location of the reducer program in Hadoop
files	Add an HDFS file to the distributed cache.
arg	Set a program argument.
define	Set a Hadoop configuration variable using the syntax <code>define=NAME=VALUE</code> .
cmdenv	Set an environment variable using the syntax <code>cmdenv=NAME=VALUE</code> .
statusdir	A directory where WebHCat will write the status of the Map Reduce job. If provided, it is the caller's responsibility to remove this directory when done.
enablelog	<p>If statusdir is set and enablelog is true, collect Hadoop job configuration and logs in to a directory named \$statusdir/logs after the job finishes. Both completed and failed attempts are logged. The layout of subdirectories in \$statusdir/logs is:</p> <ul style="list-style-type: none"> logs/\$job_id (directory for \$job_id) logs/\$job_id/job.xml.html logs/\$job_id/\$attempt_id (directory for \$attempt_id) logs/\$job_id/\$attempt_id/stderr logs/\$job_id/\$attempt_id/stdout logs/\$job_id/\$attempt_id/syslog <p>This parameter supports only Hadoop 1.X.</p>

Parameter	Description
callback	Define a URL to be called upon job completion. You may embed a specific job ID into this URL using \$jobId. This tag will be replaced in the callback URL with this job's job ID.

- Returned result

Parameter	Description
id	A string containing the job ID similar to "job_201110132141_0001"

- Example

```
curl -i -u : --negotiate -d input=/input -d output=/oooo -d mapper=/bin/cat -d reducer="/usr/bin/wc -w" -d statusdir="/output" 'http://10.64.35.144:9111/templeton/v1/mapreduce/streaming?user.name=user1'
```

 **NOTE**

For details about prerequisites for using this interface, see [Rules](#).

29. /hive(POST)

- Description

Run Hive commands.

- URL

http://www.myserver.com/templeton/v1/hive

- Parameter

Parameter	Description
execute	String containing an entire, short Hive program to run
file	HDFS file name of a Hive program to run
files	Comma-separated files to be copied to the map reduce cluster
arg	Set a program argument.

Parameter	Description
define	Hive configuration. The format is define=key=value. If multiple instances are used, you need to configure the scratch dir for them. For example, the WebHCat instance uses define=hive.exec.scratchdir= /tmp/hive-scratch . The WebHCat1 instance uses define=hive.exec.scratchdir= /tmp/hive1-scratch . The same rule applies to other instances.
statusdir	A directory where WebHCat will write the status of the Hive job. If provided, it is the caller's responsibility to remove this directory when done.
enablelog	If statusdir is set and enablelog is true , collect Hadoop job configuration and logs in to a directory named \$statusdir/logs after the job finishes. Both completed and failed attempts are logged. The layout of subdirectories in \$statusdir/logs is: logs/\$job_id (directory for \$job_id) logs/\$job_id/job.xml.html logs/\$job_id/\$attempt_id (directory for \$attempt_id) logs/\$job_id/\$attempt_id/stderr logs/\$job_id/\$attempt_id/stdout logs/\$job_id/\$attempt_id/syslog This parameter supports only Hadoop 1.X.
callback	Define a URL to be called upon job completion. You may embed a specific job ID into this URL using \$jobId. This tag will be replaced in the callback URL with this job's job ID.

- Returned result

Parameter	Description
id	A string containing the job ID similar to "job_201110132141_0001"

- Example

```
curl -i -u : --negotiate -d execute="select count(*) from t1" -d statusdir="/output" -d define=hive.exec.scratchdir=/tmp/hive-scratch "http://10.64.35.144:9111/templeton/v1/hive?user.name=user1"
```

30. jobs(GET)

- Indicates IDs of all obtained jobs.
- URL <http://www.myserver.com/templeton/v1/jobs>
- Parameter

Parameter	Description
fields	If fields set to *, the request will return full details of the job. If fields is missing, will only return the job ID. Currently the value can only be *, other values are not allowed and will throw exception.
jobid	If jobid is present, only the records whose job ID is lexicographically greater than jobid are returned. For example, if jobid = "job_201312091733_0001", the jobs whose job ID is greater than "job_201312091733_0001" are returned. The number of records returned depends on the value of numrecords .

Parameter	Description
numrecords	If the jobid and numrecords parameters are present, the top numrecords records appearing after jobid will be returned after sorting the job ID list lexicographically. If the jobid parameter is missing and numrecords is present, the top numrecords will be returned after lexicographically sorting the job ID list. If the jobid parameter is present and numrecords is missing, all the records whose job ID is greater than jobid are returned.
showall	If this parameter is set to true , all jobs can be obtained. If this parameter is set to false , only jobs submitted by the current user can be obtained. This parameter is set to false by default.

- Returned result

Parameter	Description
id	Job id
detail	Job details if showall is set to true ; otherwise null .

- Example

```
curl -i -u : --negotiate "http://10.64.35.144:9111/templeton/v1/jobs?user.name=user1"
```

31. jobs/:jobid(GET)

- Indicates the information of the specified job.
- URL
`http://www.myserver.com/templeton/v1/jobs/:jobid`
- Parameter

Parameter	Description
jobid	The job ID to check. This is the ID received when the job was created.

- Returned result

Parameter	Description
status	A JSON object containing the job status information
profile	A JSON object containing the job profile information. WebHCat passes along the information in the JobProfile object, which is subject to change from one Hadoop version to another.
id	The job ID
percentComplete	The job completion percentage, for example "75% complete". If the job is completed, the value is null.
user	User name of the job creator
callback	The callback URL, if any
userargs	A JSON object representing the argument names and values for the job submission request
exitValue	The job's exit value

- Example

```
curl -i -u : --negotiate "http://10.64.35.144:9111/templeton/v1/jobs/job_1440386556001_0255?user.name=user1"
```

32. jobs/:jobid(DELETE)

- Indicates kill jobs.
- URL
`http://www.myserver.com/templeton/v1/jobs/:jobid`
- Parameter

Parameter	Description
jobid	The job ID to delete. This is the ID received when the job was created.

- Returned result

Parameter	Description
user	Indicates the user that submits jobs.

Parameter	Description
status	A JSON object containing the job status information
profile	A JSON object containing the job profile information. WebHCat passes along the information in the JobProfile object, which is subject to change from one Hadoop version to another.
id	The job ID
callback	The callback URL, if any

– Example

```
curl -i -u : --negotiate -X DELETE "http://10.64.35.143:9111/templeton/v1/jobs/job_1440386556001_0265?user.name=user1"
```

17.5.2 Hive Access Configuration on Windows Using EIPs

Scenario

This section describes how to bind Elastic IP addresses (EIPs) to a cluster and configure Hive files so that sample files can be compiled locally.

This section uses hive-jdbc-example as an example.

Procedure

Step 1 Apply for an EIP for each node in the cluster and add public IP addresses and corresponding host domain names of all nodes to the Windows local **hosts** file. (If a host name contains uppercase letters, change them to lowercase letters.)

1. On the VPC console, apply for EIPs (the number of EIPs you buy should be equal to the number of nodes in the cluster), click the name of each node in the MRS cluster, and bind an EIP to each node on the **EIPs** page.

For details, see **Virtual Private Cloud > User Guide > EIP > Assigning an EIP and Binding It to an ECS**.

2. Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.

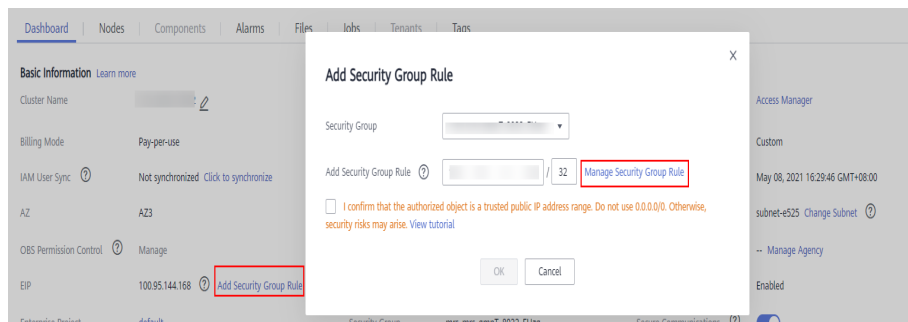
```

0 10 20 30 40 50 60 70 80 90 100 110 120 130 140
1 Mapping between public IP addresses and private IP addresses
2 100.95.10.115 172.16.0.120
3 100.95.10.116 172.16.0.42
4 100.93.10.115 172.16.0.62
5 100.95.10.116 172.16.0.200
6 100.93.10.115 172.16.0.139
7 100.93.10.116 172.16.0.214
8
9 hosts file in the cluster
10 172.16.0.120 node-group-1XZI0002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
11 172.16.0.42 node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com.
12 172.16.0.62 node-group-1XZI0003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
13 172.16.0.200 node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
14 172.16.0.139 node-group-1XZI0001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
15 172.16.0.214 node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.
16
17 hosts file that should be added to Windows
18 100.95.10.115 node-group-1xzi0002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
19 100.95.10.116 node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com.
20 100.93.10.115 node-group-1xzi0003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
21 100.95.10.116 node-master1ceip.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1ceip.ead64699-185a-4290-bbef-1a07e2f0459b.com.
22 100.93.10.115 node-group-1xzi0001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
23 100.93.10.116 node-master2pvnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pvnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.

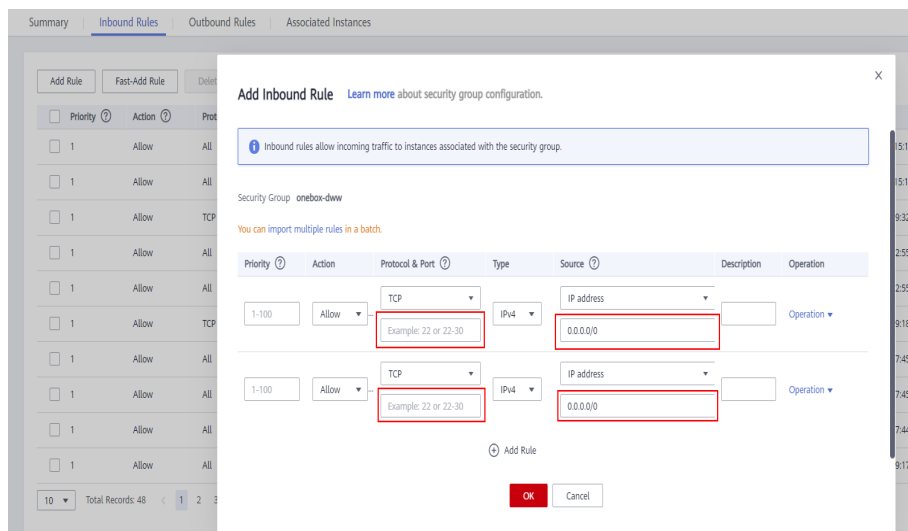
```

Step 2 Configure security group rules for the cluster.

1. On the **Dashboard** page, choose **Add Security Group Rule > Manage Security Group Rule**.



2. On the **Inbound Rules** tab page, click **Add Rule**. In the **Add Inbound Rule** dialog box, configure the Windows IP address and port 10000.



- Step 3** On Manager, choose **Cluster > Services > Hive > More > Download Client**, and copy the **core-site.xml** and **hiveclient.properties** files on the client to the **resources** directory of the sample project.

- Step 4** In the sample code, change the ZooKeeper IP addresses in the JDBC URL to the HiveServer2 host name for connection. Change the URL to **jdbc:hive2://HiveServer host name:10000/**.

NOTE

- The IP addresses in the **/hiveserver2** directory of ZooKeeper are private IP addresses and cannot be used to connect to Hive from Windows. Therefore, you need to replace ZooKeeper IP addresses with the HiveServer2 host name.
- To obtain the HiveServer2 host name, choose **Cluster > Services > Hive > Instance** on Manager and view **Host Name** of **HiveServer** on the **Instance** page.

18 Impala Development Guide (Security Mode)

18.1 Overview

18.1.1 Application Development Overview

Impala provides fast, interactive SQL queries directly on your Apache Hadoop data stored in HDFS, HBase, or Object Storage Service (OBS). In addition to using the same unified storage platform, Impala also uses the same metadata, SQL syntax (Hive SQL), ODBC driver, and user interface (Impala query UI in Hue) as Apache Hive. This provides a familiar and unified platform for real-time or batch-oriented queries. Impala is an addition to tools available for querying big data. It does not replace the batch processing frameworks built on MapReduce such as Hive. Hive and other frameworks built on MapReduce are best suited for long running batch jobs.

Impala provides the following features:

- Most common SQL-92 features of Hive Query Language (HiveQL) including SELECT, JOIN, and aggregate functions
- HDFS, HBase, and OBS storage, including:
 - HDFS file formats: delimited text files, Parquet, Avro, SequenceFile, and RCFile
 - Compression codecs: Snappy, GZIP, Deflate, BZIP
- Common data access interfaces, including:
 - JDBC driver
 - ODBC driver
 - Hue Beeswax and the Impala query UI
- **impala-shell** command line interface
- Kerberos authentication

Impala applies to offline analysis (such as log and cluster status analysis) of real-time data queries, large-scale data mining (such as user behavior analysis, interest region analysis, and region display), and other scenarios.

18.1.2 Basic Concepts

- **Client**
Users can access the server from the client through Java APIs and Thrift APIs to perform Impala-related operations. The Impala client in this document refers to the ImpalaImpala client installation directory, which contains sample codes for Impala access using Java APIs.
- **HiveQL**
Hive Query Language (SQL-like statement, similar to Hive)
- **Statestore**
The StateStore checks on the health of all Impalad instance in an Impala cluster, and continuously relays its findings to each of those Impalad instances. If an Impalad instance goes offline due to node failure, network error, or other reasons, the StateStore informs all the other Impalad instances so that future queries can avoid making requests to the unreachable Impalad instances.
- **Catalog**
The Catalog Service relays the metadata changes from each Impalad instance to other Impalad instances in a cluster. The catalog service avoids the need to issue the REFRESH statement on other instances when the metadata changes on an Impalad instance. When you create or modify a table in Hive, you need to issue REFRESH or INVALIDATE METADATA.

18.1.3 Application Development Process

[Figure 18-1](#) and [Table 18-1](#) describe the phases in the development process.

Figure 18-1 Impala application development process

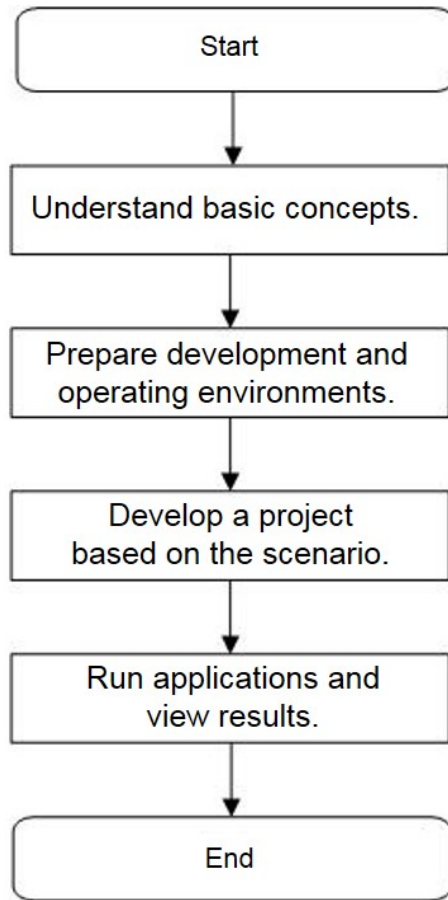


Table 18-1 Description of the Impala application development process

Phase	Description	Reference
Understand basic concepts.	Before application development, learn basic concepts of Impala.	Basic Concepts
Prepare development and operating environments.	You can use Java and Python to develop Impala applications. You are advised to use the IntelliJ IDEA tool to configure the development environment based on the language.	Preparing Development and Operating Environment
Develop a project based on the scenario.	Impala provides Java and Python sample projects, covering table creation, data load, and data queries.	Typical Application Scenario

Phase	Description	Reference
Run applications and view results.	This phase provides guidance for users to submit a developed application for running and view the result.	Application Commissioning

18.2 Environment Preparation

18.2.1 Preparing Development and Operating Environment

Preparing Development Environment

[Table 18-2](#) describes the local environment required for application development. You also need to prepare an environment for verifying whether the application is running properly.

Table 18-2 Development environment

Item	Description
OS	<ul style="list-style-type: none"> Development environment: Windows 7 or later version is recommended. Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.
JDK installation	<p>Basic configurations of the development and operating environments. The version requirements are as follows:</p> <ul style="list-style-type: none"> The server and client of an MRS cluster support only built-in Oracle JDK 1.8, which cannot be replaced. If users' applications need to reference the JAR files of the SDK class in the user application processes, Oracle JDK and IBM JDK are supported. <ul style="list-style-type: none"> Oracle JDK versions: 1.7 and 1.8 IBM JDK versions: 1.7.8.10, 1.7.9.40, and 1.8.3.0

Item	Description
IntelliJ IDEA installation and configuration	<p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p>NOTE</p> <ul style="list-style-type: none"> • If you use IBM JDK, ensure that the JDK configured in IntelliJ IDEA is IBM JDK. • If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK. • Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
User development preparation	See Preparing a Developer Account for configuration.
7-zip	Used to decompress .zip and .rar packages. The 7-zip 16.04 is supported.

Preparing an Operating Environment

During application development, you need to prepare a environment for verifying whether the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, configure the network connection, and commission the program in Windows.
- If you use the Linux environment for commissioning, you need to prepare the Linux node.

18.3 Application Development

18.3.1 Typical Application Scenario

Scenario Description

Assume that you need to develop an Impala data analysis application to manage the employee information of an enterprise. [Table 18-3](#) and [Table 18-4](#) provide employee information.

Development Guidelines

Step 1 Prepare data.

1. Create three tables: employee information table **employees_info**, employee contact table **employees_contact**, and extended employee information table **employees_info_extended**.
 - Employee information table **employees_info** contains fields such as employee ID, name, salary currency, salary, tax category, work place, and hire date. In salary currency, R indicates RMB and D indicates USD.
 - Fields in the **employees_contact** table include the employee ID, phone number, and email address.
 - Fields in the **employees_info_extended** table include the employee ID, name, mobile phone number, e-mail address, salary currency, salary, tax category, and work place. The partition field is the hire date.

For details about table creation codes, see [Creating a Table](#).
2. Load employee information to **employees_info**.
For details about data loading codes, see [Loading Data](#).

[Table 18-3](#) provides employee information.

Table 18-3 Employee information

Employee ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hire Date
1	Wang	R	8000.01	personal income tax&0.05	China:Shenzhen	2014
3	Tom	D	12000.02	personal income tax&0.09	America:NewYork	2014
4	Jack	D	24000.03	personal income tax&0.09	America:Manhattan	2014
6	Linda	D	36000.04	personal income tax&0.09	America:NewYork	2014
8	Zhang	R	9000.05	personal income tax&0.05	China:Shanghai	2014

3. Load employee contact information to **employees_contact**.
[Table 18-4](#) provides employee contact information.

Table 18-4 Employee contact information

Employee ID	Phone Number	Email
1	135 XXXX XXXX	xxxx@xx.com
3	159 XXXX XXXX	xxxxx@xx.com.cn
4	186 XXXX XXXX	xxxx@xx.org
6	189 XXXX XXXX	xxxx@xxx.cn
8	134 XXXX XXXX	xxxx@xxxx.cn

Step 2 Analyze data.

For details about data analysis codes, see [Querying Data](#).

- Query contact information of employees whose salaries are paid in USD.
- Query the IDs and names of employees who were hired in 2014, and load the query results to the partition with the hire date of 2014 in the **employees_info_extended** table.
- Collect the number of records in the **employees_info** table.
- Query information about employees whose email addresses end with "cn".

Step 3 Submit a data analysis task to collect the number of records in the **employees_info** table. For details, see [Sample Program Guide](#).

----End

18.3.2 Creating a Table

Function Description

This section describes how to use Impala SQL to create internal and external tables. You can create a table by using any of the following methods:

- Customize the table structure, and use the key word **EXTERNAL** to differentiate between internal and external tables.
 - If all data is to be processed by Impala, create an internal table. When an internal table is deleted, the metadata and data in the table are deleted together.
 - If data is to be processed by multiple tools (such as Pig), create an external table. When an external table is deleted, only metadata is deleted.
- Create a table based on existing tables. Use CREATE LIKE to fully copy the original table structure, including the storage format.
- Create a table based on query results using CREATE AS SELECT.

This method is flexible. By using this method, you can specify fields (except for the storage format) that you want to copy to the new table when copying the structure of the existing table.

Sample Code

```
-- Create the external table employees_info.
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info
(
  id INT,
  name STRING,
  usd_flag STRING,
  salary DOUBLE,
  deductions MAP<STRING, DOUBLE>,
  address STRING,
  entrytime STRING
)
-- Specify the field delimiter.
-- "delimited fields terminated by" indicates that a column delimiter is a comma (.).
ROW FORMAT delimited fields terminated by ',';
-- Specify the table storage format to TEXTFILE.
STORED AS TEXTFILE;
-- Use CREATE Like to create a table.
CREATE TABLE employees_like LIKE employees_info;
-- Run the DESCRIBE command to check the structures of the employees_info and employees_like tables.
DESCRIBE employees_info;
DESCRIBE employees_like;

-- Create the internal table employees_info.
CREATE TABLE IF NOT EXISTS employees_info
(
  id INT,
  name STRING,
  usd_flag STRING,
  salary DOUBLE,
  deductions MAP<STRING, DOUBLE>,
  address STRING,
  entrytime STRING
)
-- Specify the field delimiter.
-- "delimited fields terminated by" indicates that a column delimiter is a comma (.).
ROW FORMAT delimited fields terminated by ',';
-- Specify the table storage format to TEXTFILE.
STORED AS TEXTFILE;
```

Extended Applications

- Create a partition table.

A table may have one or more partitions. Each partition is saved as an independent folder in the table directory. Partitions help minimize the query scope, accelerate data query, and allow users to manage data based on certain criteria.

A partition is defined using the **PARTITIONED BY** clause during table creation.

```
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info_extended
(
  id INT,
  name STRING,
  usd_flag STRING,
  salary DOUBLE,
  deductions MAP<STRING, DOUBLE>,
  address STRING
)
-- Use "PARTITIONED BY" to specify the column name and data type of the partition.
PARTITIONED BY (entrytime STRING)
STORED AS TEXTFILE;
```

- Update the table structure.

After a table is created, you can use **ALTER TABLE** to add or delete fields, modify table attributes, and add partitions.

```
-- Add the tel_phone and email fields to the employees_info_extended table.
ALTER TABLE employees_info_extended ADD COLUMNS (tel_phone STRING, email STRING);
```

- Enable Impala to use OBS.

You need to set the specified parameters for **core-site.xml** on the cluster management page of Manager. You can log in to OBS console and access the **My Credential** page to obtain the AK/SK.

```
fs.s3a.access.key=AK;
fs.s3a.secret.key=SK;
fs.s3a.endpoint=endpoint;
```

Set the storage type of the new table to obs.

```
create table obs(c1 string, c2 string) stored as parquet location 's3a://obs-lmm/hive/orctest'
tblproperties('orc.compress'='SNAPPY');
```

NOTE

When Impala uses OBS to store data, partition and table storage locations in the same table cannot be stored in different buckets.

For example, create a partition table and set its storage location to the folder in OBS bucket 1. In this case, modifying the storage location of the table partition does not take effect. When data is inserted, the storage location of the table is used.

1. Create a partition table and specify the path for storing the table.

```
create table table_name(id int,name string,company string) partitioned by(dt date) row
format delimited fields terminated by ',' stored as textfile location "obs://OBS bucket 1/
Folder in the bucket";
```
2. Modifying the storage location of the table partition to another bucket does not take effect.

```
alter table table_name partition(dt date) set location "obs://OBS bucket 2/Folder in the
bucket";
```

18.3.3 Loading Data

Function Description

This section describes how to use Impala SQL to load data to the existing **employees_info** table. You can learn how to load data from a cluster.

Sample Code

```
--Load the employee_info.txt file from the /opt/impala_examples_data/ directory of the local file system
to the employees_info table.
LOAD DATA LOCAL INPATH '/opt/impala_examples_data/employee_info.txt' OVERWRITE INTO TABLE
employees_info;

-- Load /user/impala_examples_data/employee_info.txt from HDFS to the employees_info table.
LOAD DATA INPATH '/user/impala_examples_data/employee_info.txt' OVERWRITE INTO TABLE
employees_info;
```

NOTE

The essence of loading data is to copy the data to the specified table directory in HDFS.

The **LOAD DATA LOCAL INPATH** command can be used to load files from a local file system to Impala. If **LOCAL** is specified, the path refers to the path of the local file system of the currently connected **Impalad**.

18.3.4 Querying Data

Function Description

This section describes how to use Impala SQL to query and analyze data. You can query and analyze data using the following methods:

- Use common features of a SELECT query, such as JOIN.
- Load data to a specified partition.
- Use built-in functions of Impala.
- Query and analyze data using user-defined functions. For details about how to create and define functions, see [User-defined Functions](#).

Sample Code

```
-- Query contact information of employees whose salaries are paid in USD.
SELECT
a.name,
b.tel_phone,
b.email
FROM employees_info a JOIN employees_contact b ON(a.id = b.id) WHERE usd_flag='D';
-- Query the IDs and names of employees who were hired in 2014, and load the query results to the
partition with the hire date of 2014 in the employees_info_extended table.
INSERT OVERWRITE TABLE employees_info_extended PARTITION (entrytime = '2014')
SELECT
a.id,
a.name,
a.usd_flag,
a.salary,
a.deductions,
a.address,
b.tel_phone,
b.email
FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE a.entrytime = '2014';
-- Use the existing function COUNT() in Impala to calculate the number of records in the employees_info
table.
SELECT COUNT(*) FROM employees_info;
-- Query information about employees whose email addresses end with "cn".
SELECT a.name, b.tel_phone FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE
b.email like '%cn';
```

Extended Applications

For details about user-defined functions, see [User-defined Functions](#).

18.3.5 User-defined Functions

When built-in functions of Impala cannot meet requirements, you can compile user-defined functions (UDFs) and use them for query.

According to implementation methods, UDFs are classified as follows:

- Common UDFs: used to perform operations on a single data row and export a single data row.
- User-defined aggregating functions (UDAFs): used to input multiple data rows and export a single data row.
- User-defined table-generating functions (UDTFs): used to perform operations on a single data row and export multiple data rows. **Impala does not support this type of UDFs.**

According to use methods, UDFs are classified as follows:

- Temporary functions: used only in the current session and must be recreated after a session restarts.
- Permanent functions: used in multiple sessions. You do not need to create them every time a session restarts.

Impala supports development of Java UDFs and reuse of UDFs developed by Hive. The prerequisite is that the data types supported by Impala are used. For details about the data types supported by Impala, visit http://impala.apache.org/docs/build3x/html/topics/impala_datatypes.html.

In addition, Impala supports UDFs written in C++, which provides better performance than Java UDFs.

Example

The following is an example of reusing the **lower()** function:

```
[localhost:21000] > create database udfs;
[localhost:21000] > use udfs;
[localhost:21000] > create function my_lower(string)
returns string location '/user/hive/udfs/hive.jar'
symbol='org.apache.hadoop.hive.ql.udf.UDFLower';
[localhost:21000] > select my_lower('Some String NOT ALREADY LOWERCASE');
+-----+
| udfs.my_lower('some string not already lowercase') |
+-----+
| some string not already lowercase |
+-----+
Returned 1 row(s) in 0.11s
[localhost:21000] > create table t2 (s string);
[localhost:21000] > insert into t2 values ('lower'),('UPPER'),('Init cap'),('CamelCase');
Inserted 4 rows in 2.28s
[localhost:21000] > select * from t2;
+-----+
| s |
+-----+
| lower |
| UPPER |
| Init cap |
| CamelCase |
+-----+
Returned 4 row(s) in 0.47s
[localhost:21000] > select my_lower(s) from t2;
+-----+
| udfs.my_lower(s) |
+-----+
| lower |
| upper |
| init cap |
| camelcase |
+-----+
Returned 4 row(s) in 0.54s
```

18.3.6 Sample Program Guide

Function Description

This section describes how to use a sample program to complete an analysis task. This section uses JDBC APIs as an example to describe how to submit a data analysis task.

Sample Code

If you submit a data analysis task using Impala JDBC APIs, refer to **JDBCExample.java** in the sample program.

1. Change the value of the following variable to **false**, specifying that the authentication mode for the connected clusters is normal mode.
2. Define Impala SQL. Impala SQL must be a single statement and cannot contain ";".

```
// Indicates whether the authentication mode of the connected cluster is security mode.
boolean isSecureVer = false;
```

```
// Define HQL, which cannot contain the semicolon (;).
String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT,name STRING)",
"SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};
```

3. Build JDBC URL.

```
// Build JDBC URL.
StringBuilder sBuilder = new StringBuilder(
"jdbc:hive2://").append("impalad_ip").append("/");

if (isSecurityMode) {
    // Security mode
    sBuilder.append(";auth=")
        .append(clientInfo.getAuth())
        .append(";principal=")
        .append(clientInfo.getPrincipal())
        .append(";");
} else {
    // Normal mode
    sBuilder.append(";auth=none");
}
String url = sBuilder.toString();
```

NOTE

If an Impalad instance is directly connected, an Impalad instance fault will cause Impala access failure.

4. Load the Hive JDBC driver.

```
// Load the Hive JDBC driver.
Class.forName(HIVE_DRIVER);
```

5. Enter a correct username, obtain the JDBC connection, confirm the Impala SQL type (DDL/DML), call APIs to run Impala SQL, return the queried column name and result to the console, and close the JDBC connection.

```
Connection connection = null;
try {
    // Obtain the JDBC connection.
    // If you set the second parameter to an incorrect username, the anonymous user will be used
for login.
    connection = DriverManager.getConnection(url, "userName", "");
    // Create a table.
    // To import data to a table after the table is created, you can use the LOAD statement. For
example, import data from HDFS to the table.
    //load data inpath '/tmp/employees.txt' overwrite into table employees_info;
    execDDL(connection,sqls[0]);
    System.out.println("Create table success!");
    // Query
    execDML(connection,sqls[1]);
    // Delete the table.
    execDDL(connection,sqls[2]);
    System.out.println("Delete table success!");
}
finally {
    // Close the JDBC connection.
    if (null != connection) {
```

```
        connection.close();
    }

    public static void execDDL(Connection connection, String sql)
        throws SQLException {
        PreparedStatement statement = null;
        try {
            statement = connection.prepareStatement(sql);
            statement.execute();
        }
        finally {
            if (null != statement) {
                statement.close();
            }
        }
    }

    public static void execDML(Connection connection, String sql) throws SQLException {
        PreparedStatement statement = null;
        ResultSet resultSet = null;
        ResultSetMetaData resultMetaData = null;

        try {
            // Run the Impala SQL statement.
            statement = connection.prepareStatement(sql);
            resultSet = statement.executeQuery();
            // Output the queried column name to the console.
            resultMetaData = resultSet.getMetaData();
            int columnCount = resultMetaData.getColumnCount();
            for (int i = 1; i <= columnCount; i++) {
                System.out.print(resultMetaData.getColumnLabel(i) + '\t');
            }
            System.out.println();
            // Output the query result to the console.
            while (resultSet.next()) {
                for (int i = 1; i <= columnCount; i++) {
                    System.out.print(resultSet.getString(i) + '\t');
                }
                System.out.println();
            }
        }
        finally {
            if (null != resultSet) {
                resultSet.close();
            }

            if (null != statement) {
                statement.close();
            }
        }
    }
}
```

18.4 Application Commissioning

18.4.1 Commissioning Applications on Windows

Running the JDBC Client Using CLI

Step 1 Run the sample project.

After importing and modifying the sample project in the development environment, right-click **JDBCExample.java** and choose **Run 'JDBCExample.main()'** from the shortcut menu to run the corresponding application project.

 **NOTE**

You can use either of the following method to access an MRS cluster to operate Impala on Windows.

Method 1: Apply for a Windows ECS to access the MRS cluster to operate Impala. Run the sample code after the development environment is installed. To apply for ECS to access the MRS cluster, perform the following steps:

1. On the **Active Clusters** page, click the name of an existing cluster.
On the cluster details page, record the **AZ**, **VPC**, and **Default Security Group** of the Master node.
2. On the ECS management console, create a new ECS.
The **AZ**, **VPC**, and **security group** of ECS must be the same as those of the cluster to be accessed.
Select a Windows public image.
For details about other configuration parameters, see [Purchasing an ECS](#).

Method 2: Use the local host to access the MRS cluster to operate Impala. After installing the development environment and completing the following steps, run the sample code.

1. Bind an EIP address to any Core node. Then, configure the IP address in the **impala-server** configuration item in the **client.properties** file of the development sample to access the Impala service and submit SQL statements. To bind an EIP, perform the following steps:
 1. On the ECS page, apply for an EIP and bind it to the ECS. For details, see [Assigning an EIP and Binding It to an ECS](#).
 2. Open security group rules for the MRS cluster.
Add security group rules to the security groups of Master and Core nodes in the cluster so that ECS can access the cluster. For details, see [Configuring Security Group Rules](#).
2. Modify the IP addresses of the **kdc**, **admin_server**, and **kpasswd_server** parameters in the **krb5.conf** file of the imported sample to the EIPs corresponding to KrbServer. (The Kerberos service is deployed on the master node by default. Therefore, the public IP address of the master node is used.)

The **client.properties** configuration in the sample is as follows:

```
auth = KERBEROS ## Kerberos mode
principal = impala/node-ana-corexphm@10530B19_8446_4846_97BD_87880A2535DF.COM ##
Principal used by the Impalad instance to be connected
impala-server = XX.XX.XX.XX:21050 ## Specifies the service address bound to the Core node where the
Impalad instance to be connected is located. If method 2 is used, enter the EIP bound in step 1.
```

Step 2 View the execution result.

View the Impala SQL query results in the sample code. If the following information is displayed, the execution is successful.

Result:

```
Create table success!
_c0
0
Delete table success!
```

----End

18.4.2 Commissioning Applications on Linux

- Step 1** Create a directory as the running directory in the running and commissioning environment, for example, **/opt/impala_examples**, and create the **conf** subdirectory in the directory.

Step 2 Run the `mvn package` command to obtain a JAR package, for example, `impala-examples-mrs-2.1-jar-with-dependencies.jar` obtained from the `target` directory of the project and copy the package to the `/opt/impala_examples` directory.

Step 3 For a security cluster with Kerberos authentication enabled, copy the `user.keytab` and `krb5.conf` files obtained in [Preparing a Developer Account](#) to the `/opt/impala_examples/conf` directory.

Step 4 In Linux, run the following command to run the sample program:

```
chmod +x /opt/impala_examples -R
cd /opt/impala_examples
java -cp impala-examples-mrs-2.1-jar-with-dependencies.jar
com.huawei.bigdata.impala.example.ExampleMain
```

Step 5 In the CLI, view the Impala SQL query results in the example code.

If the following information is displayed, the sample project execution is successful on Linux.

```
Create table success!
_c0
0
Delete table success!
```

----End

18.5 Impala APIs

18.5.1 JDBC

Impala uses Hive JDBC APIs that comply with the Java JDBC driver standard. For details, see [JDK 1.7 API](#).

NOTE

Impala does not support all standard Hive JDBC APIs. `SQLException` "**Method not supported**" is generated when some operations are performed.

18.5.2 Impala SQL

Impala SQL is highly compatible with HiveQL. For details, visit https://impala.apache.org/docs/build/html/topics/impala_langref.html.

18.6 Development Specifications

18.6.1 Rules

Load the Hive JDBC Driver

A client program connects to Impalad using JDBC. You need to load the JDBC driver class `org.apache.hive.jdbc.HiveDriver` of Hive first.

Therefore, you must load the driver class using the current class loader before starting the client program.

If there is no JAR file in **classpath**, the client program throws "Class Not Found" and exits.

Example:

```
Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance();
```

Obtain a Database Connection

Use the driver management class **java.sql.DriverManager** of the JDK to obtain an Impalad database connection.

The URL of the Impalad database is as follows: url="jdbc:hive2://xxx.xxx.xxx.xxx:21050;auth=KERBEROS;principal=impala/hadoop.hadoop.com@HADOOP.COM;user.principal=impala/hadoop.hadoop.com;user.keytab=conf/impala.keytab";

The username and password are null or empty because authentication has been performed successfully.

```
//Set up a connection.
```

```
connection = DriverManager.getConnection(url, "", "");
```

// Execute Impala SQL

Execute Impala SQL. Note that Impala SQL cannot end with a semicolon (;).

Correct example:

```
String sql = "SELECT COUNT(*) FROM employees_info";  
Connection connection = DriverManager.getConnection(url, "", "");  
PreparedStatement statement = connection.prepareStatement(sql);  
resultSet = statement.executeQuery();
```

Incorrect example:

```
String sql = "SELECT COUNT(*) FROM employees_info";  
Connection connection = DriverManager.getConnection(url, "", "");  
PreparedStatement statement = connection.prepareStatement(sql);  
resultSet = statement.executeQuery();
```

Use Impala SQL Syntax to Check Whether a Field is Null

Use **is null** to check whether a field is empty, that is, the field has no value. Use **is not null** to check whether a field is not null, that is, the field has a value.

If you use **is null** to check a field whose type is String and length is 0, **False** is returned. In this case, you need to use **col = ""** to check null character strings, and **col != ""** to check non-null character strings.

Correct example:

```
select * from default.tbl_src where id is null;  
select * from default.tbl_src where id is not null;  
select * from default.tbl_src where name = "";  
select * from default.tbl_src where name != "";
```

Incorrect example:

```
select * from default.tbl_src where id = null;
select * from default.tbl_src where id != null;
select * from default.tbl_src where name is null;
select * from default.tbl_src where name is not null; Note: The id field in the tbl_src table is of the Int type,
and the name field is of the String type.
```

Multithread Security Login Mode

If multiple threads are performing login operations, the relogin mode must be used for the subsequent logins of all threads after the first successful login of an application.

Login sample code:

```
private Boolean login(Configuration conf){
    boolean flag = false;
    UserGroupInformation.setConfiguration(conf);

    try {
        UserGroupInformation.loginUserFromKeytab(conf.get(PRINCIPAL), conf.get(KEYTAB));
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "
+UserGroupInformation.isLoginKeytabBased());
        flag = true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return flag;
}
```

Relogin sample code:

```
public Boolean relogin(){
    boolean flag = false;
    try {

        UserGroupInformation.getLoginUser().reloginFromKeytab();
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "
+UserGroupInformation.isLoginKeytabBased());
        flag = true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return flag;
}
```

Do Not Perform Read and Write Operations on the Same Table at the Same Time

In the current version, Hive does not support concurrent operations. Therefore, do not perform read and write operations on the same table at the same time. Otherwise, query results may be inaccurate or even tasks fail.

18.6.2 Suggestions

Impala SQL Compilation - Implicit Type Conversion Not Supported

If the query statements use the field value for filtering, the implicit type conversion similar to Hive cannot be used to compile Impala SQL.

Impala example:

```
select * from default.tbl_src where id = 10001;
select * from default.tbl_src where name = 'TestName';
```

Hive example (implicit type conversion supported):

```
select * from default.tbl_src where id = '10001';  
select * from default.tbl_src where name = TestName;
```

 **NOTE**

In the `tbl_src` table, the `id` field is of the `Int` type and the `name` field is of the `String` type.

JDBC Timeout Limit

Impala uses the Hive JDBC whose implementation has a timeout limit. The default value is 5 minutes. You can set the value using **`java.sql.DriverManager.setLoginTimeout (int seconds)`**. The unit of **seconds** is second.

18.6.3 Examples

Sample Code of JDBC Secondary Development

The following sample code provides the following functions:

1. In normal (non-Kerberos) mode, uses the username and password for login. If no user is specified, anonymous login is used.
2. Provides the principal for logging in to the Kerberos user in the JDBC URL so that programs can automatically perform secure logins and create Impala connections.
3. Runs Impala SQL statements for creating, querying, and deleting tables.

```
package com.huawei.bigdata.impala.example;  
  
import java.io.IOException;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.ResultSetMetaData;  
import java.sql.SQLException;  
  
/**  
 * Simple example for hive jdbc.  
 */  
public class JDBCExample {  
    private static final String HIVE_DRIVER = "org.apache.hive.jdbc.HiveDriver";  
    private ClientInfo clientInfo;  
    private boolean isSecurityMode;  
    public JDBCExample(ClientInfo clientInfo, boolean isSecurityMode){  
        this.clientInfo = clientInfo;  
        this.isSecurityMode = isSecurityMode;  
    }  
  
    /**  
     *  
     * @throws ClassNotFoundException  
     * @throws SQLException  
     */  
    public void run() throws ClassNotFoundException, SQLException {  
  
        //Define hive sql, the sql can not include ";"  
        String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT,name STRING)",  
            "SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};  
  
        StringBuilder sBuilder = new StringBuilder(  

```

```

        "jdbc:hive2://").append(clientInfo.getImpalaServer()).append("/");

    if (isSecurityMode) {
        sBuilder.append(";auth=")
            .append(clientInfo.getAuth())
            .append(";principal=")
            .append(clientInfo.getPrincipal())
            .append(";");
    } else {
        sBuilder.append(";auth=noSasl");
    }
    String url = sBuilder.toString();
    Class.forName(HIVE_DRIVER);
    Connection connection = null;
    try {
        /**
         * Get JDBC connection, If not use security mode, need input correct username,
         * otherwise, wil login as "anonymous" user
         */
        //connection = DriverManager.getConnection(url, "", "");
        connection = DriverManager.getConnection(url);
        /**
         * Run the create table sql, then can load the data if needed. eg.
         * "load data inpath '/tmp/employees.txt' overwrite into table employees_info;"
         */
        execDDL(connection,sqls[0]);
        System.out.println("Create table success!");

        execDML(connection,sqls[1]);

        execDDL(connection,sqls[2]);
        System.out.println("Delete table success!");
    }
    finally {
        if (null != connection) {
            connection.close();
        }
    }
}

public static void execDDL(Connection connection, String sql)
    throws SQLException {
    PreparedStatement statement = null;
    try {
        statement = connection.prepareStatement(sql);
        statement.execute();
    }
    finally {
        if (null != statement) {
            statement.close();
        }
    }
}

public static void execDML(Connection connection, String sql) throws SQLException {
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    ResultSetMetaData resultMetaData = null;

    try {
        statement = connection.prepareStatement(sql);
        resultSet = statement.executeQuery();

        /**
         * Print the column name to console
         */
        resultMetaData = resultSet.getMetaData();
        int columnCount = resultMetaData.getColumnCount();
        for (int i = 1; i <= columnCount; i++) {

```

```
        System.out.print(resultMetaData.getColumnLabel(i) + '\t');
    }
    System.out.println();

    /**
     * Print the query result to console
     */
    while (resultSet.next()) {
        for (int i = 1; i <= columnCount; i++) {
            System.out.print(resultSet.getString(i) + '\t');
        }
        System.out.println();
    }
}
finally {
    if (null != resultSet) {
        resultSet.close();
    }

    if (null != statement) {
        statement.close();
    }
}
}
```

19 Impala Development Guide (Normal Mode)

19.1 Overview

19.1.1 Application Development Overview

Impala provides fast, interactive SQL queries directly on your Apache Hadoop data stored in HDFS, HBase, or Object Storage Service (OBS). In addition to using the same unified storage platform, Impala also uses the same metadata, SQL syntax (Hive SQL), ODBC driver, and user interface (Impala query UI in Hue) as Apache Hive. This provides a familiar and unified platform for real-time or batch-oriented queries. Impala is an addition to tools available for querying big data. It does not replace the batch processing frameworks built on MapReduce such as Hive. Hive and other frameworks built on MapReduce are best suited for long running batch jobs.

Impala provides the following features:

- Most common SQL-92 features of Hive Query Language (HiveQL) including SELECT, JOIN, and aggregate functions
- HDFS, HBase, and OBS storage, including:
 - HDFS file formats: delimited text files, Parquet, Avro, SequenceFile, and RCFile
 - Compression codecs: Snappy, GZIP, Deflate, BZIP
- Common data access interfaces, including:
 - JDBC driver
 - ODBC driver
 - Hue Beeswax and the Impala query UI
- **impala-shell** command line interface
- Kerberos authentication

Impala applies to offline analysis (such as log and cluster status analysis) of real-time data queries, large-scale data mining (such as user behavior analysis, interest region analysis, and region display), and other scenarios.

19.1.2 Basic Concepts

- **Client**

Users can access the server from the client through Java APIs and Thrift APIs to perform Impala-related operations. The Impala client in this document refers to the ImpalaImpala client installation directory, which contains sample codes for Impala access using Java APIs.

- **HiveQL**

Hive Query Language (SQL-like statement, similar to Hive)

- **Statestore**

The StateStore checks on the health of all Impalad instance in an Impala cluster, and continuously relays its findings to each of those Impalad instances. If an Impalad instance goes offline due to node failure, network error, or other reasons, the StateStore informs all the other Impalad instances so that future queries can avoid making requests to the unreachable Impalad instances.

- **Catalog**

The Catalog Service relays the metadata changes from each Impalad instance to other Impalad instances in a cluster. The catalog service avoids the need to issue the REFRESH statement on other instances when the metadata changes on an Impalad instance. When you create or modify a table in Hive, you need to issue REFRESH or INVALIDATE METADATA.

19.1.3 Application Development Process

[Figure 19-1](#) and [Table 19-1](#) describe the phases in the development process.

Figure 19-1 Impala application development process

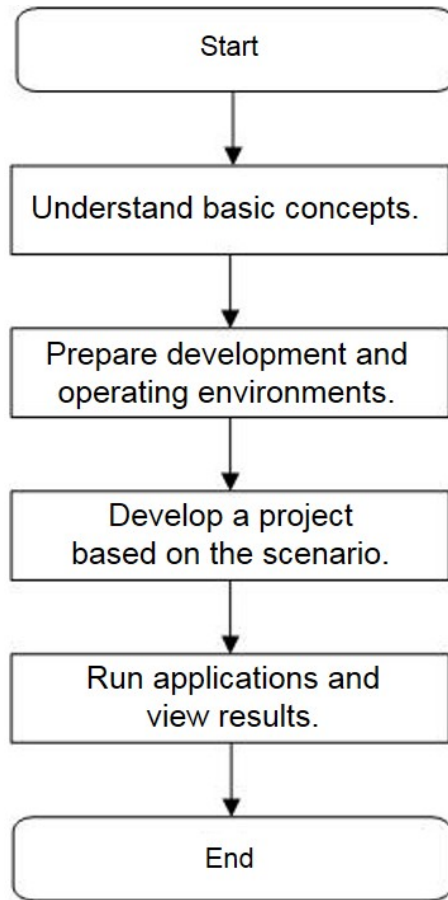


Table 19-1 Description of the Impala application development process

Phase	Description	Reference
Understand basic concepts.	Before application development, learn basic concepts of Impala.	Basic Concepts
Prepare development and operating environments.	You can use Java and Python to develop Impala applications. You are advised to use the IntelliJ IDEA tool to configure the development environment based on the language.	Preparing Development and Operating Environment
Develop a project based on the scenario.	Impala provides Java and Python sample projects, covering table creation, data load, and data queries.	Typical Application Scenario

Phase	Description	Reference
Run applications and view results.	This phase provides guidance for users to submit a developed application for running and view the result.	Application Commissioning

19.2 Environment Preparation

19.2.1 Preparing Development and Operating Environment

Preparing Development Environment

[Table 19-2](#) describes the local environment required for application development.

Table 19-2 Development environment

Item	Description
OS	<ul style="list-style-type: none"> Development environment: Windows 7 or later version is recommended. Operating environment: Windows or Linux OS If the program needs to be commissioned locally, the operating environment must be able to communicate with the cluster service plane network.
JDK installation	<p>Basic configurations of the development and operating environments. The version requirements are as follows:</p> <ul style="list-style-type: none"> The server and client of an MRS cluster support only built-in Oracle JDK 1.8, which cannot be replaced. If users' applications need to reference the JAR files of the SDK class in the user application processes, Oracle JDK and IBM JDK are supported. <ul style="list-style-type: none"> Oracle JDK versions: 1.7 and 1.8 IBM JDK versions: 1.7.8.10, 1.7.9.40, and 1.8.3.0

Item	Description
IntelliJ IDEA installation and configuration	<p>Basic configuration of the development environment. The version must be 2019.1 or other compatible versions.</p> <p>NOTE</p> <ul style="list-style-type: none"> • If you use IBM JDK, ensure that the JDK configured in IntelliJ IDEA is IBM JDK. • If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK. • Do not use the same workspace or the sample project in the same path for different IntelliJ IDEA programs.
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
7-zip	This tool is used to decompress *.zip and *.rar packages. 7-zip 16.04 is supported.

Preparing Operating Environment

During application development, prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, configure the network connection, and commission the program in Windows.
- If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

For details about how to use a client on the Master or Core nodes inside a cluster, see [Installing a Client](#). For details about how to use a client outside a cluster, see [Installing a Client](#).

19.2.2 Configuring and Importing a Sample Project

Scenario

To run the JDBC API sample code of Impala, you need to perform the following operations.

NOTE

The following example develops an application that uses JDBC to connect to Impala in the Windows environment.

Procedure

Step 1 Obtain the Impala sample project. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

Step 2 In the `impala-examples-normal` folder of the root directory of the Impala example project, open the `cmd` window and run the `mvn install` command to perform compilation.

Step 3 In the `impala-examples-normal` folder of the root directory of the Impala example project, open the `cmd` window and run the `mvn idea:idea` command to create an IntelliJ IDEA project.

Step 4 Import the sample project to the IntelliJ IDEA development environment.

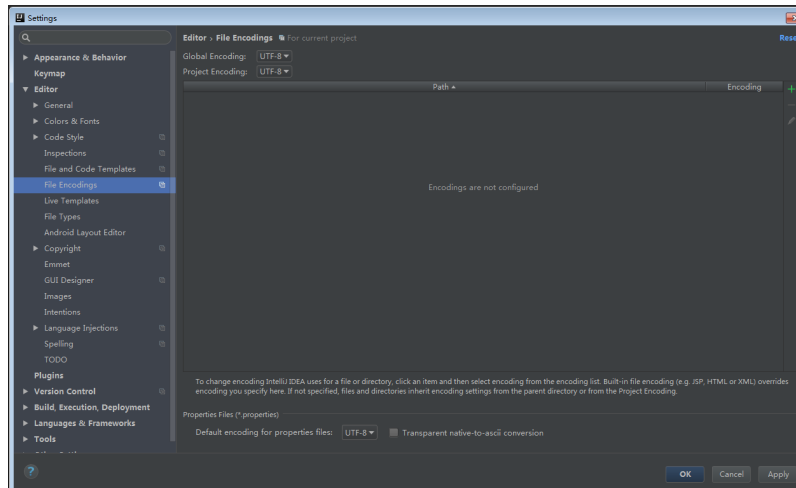
1. Run IntelliJ IDEA and choose **File > Open**.
2. In the **Open File or Project** dialog box, select the Impala sample project folder and click **OK**.

After the folder is imported, the `JDBCExample` class in the `com.huawei.bigdata.impala.example` folder is the JDBC API sample code.

Step 5 Set the IntelliJ IDEA text file coding format to prevent garbled characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.
2. Choose **Editor > File Encodings** from the navigation tree of the **Settings** window. In the "Global Encoding" and "Project Encodings" area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 19-2](#).

Figure 19-2 Setting the IntelliJ IDEA coding format



----End

19.3 Application Development

19.3.1 Typical Application Scenario

Scenario Description

Assume that you need to develop an Impala data analysis application to manage the employee information of an enterprise. [Table 19-3](#) and [Table 19-4](#) provide employee information.

Development Guidelines

Step 1 Prepare data.

1. Create three tables: employee information table **employees_info**, employee contact table **employees_contact**, and extended employee information table **employees_info_extended**.
 - Employee information table **employees_info** contains fields such as employee ID, name, salary currency, salary, tax category, work place, and hire date. In salary currency, R indicates RMB and D indicates USD.
 - Fields in the **employees_contact** table include the employee ID, phone number, and email address.
 - Fields in the **employees_info_extended** table include the employee ID, name, mobile phone number, e-mail address, salary currency, salary, tax category, and work place. The partition field is the hire date.

For details about table creation codes, see [Creating a Table](#).
2. Load employee information to **employees_info**.
For details about data loading codes, see [Loading Data](#).
[Table 19-3](#) provides employee information.

Table 19-3 Employee information

Employee ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hire Date
1	Wang	R	8000.01	personal income tax&0.05	China:Shenzhen	2014
3	Tom	D	12000.02	personal income tax&0.09	America:NewYork	2014
4	Jack	D	24000.03	personal income tax&0.09	America:Manhattan	2014
6	Linda	D	36000.04	personal income tax&0.09	America:NewYork	2014

Employee ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hire Date
8	Zhang	R	9000.05	personal income tax&0.05	China:Shanghai	2014

3. Load employee contact information to **employees_contact**.

Table 19-4 provides employee contact information.

Table 19-4 Employee contact information

Employee ID	Phone Number	Email
1	135 XXXX XXXX	xxxx@xx.com
3	159 XXXX XXXX	xxxxx@xx.com.cn
4	186 XXXX XXXX	xxxx@xx.org
6	189 XXXX XXXX	xxxx@xxx.cn
8	134 XXXX XXXX	xxxx@xxxx.cn

Step 2 Analyze data.

For details about data analysis codes, see [Querying Data](#).

- Query contact information of employees whose salaries are paid in USD.
- Query the IDs and names of employees who were hired in 2014, and load the query results to the partition with the hire date of 2014 in the **employees_info_extended** table.
- Collect the number of records in the **employees_info** table.
- Query information about employees whose email addresses end with "cn".

- Step 3 Submit a data analysis task to collect the number of records in the **employees_info** table. For details, see [Sample Program Guide](#).

----End

19.3.2 Creating a Table

Function Description

This section describes how to use Impala SQL to create internal and external tables. You can create a table by using any of the following methods:

- Customize the table structure, and use the key word **EXTERNAL** to differentiate between internal and external tables.
 - If all data is to be processed by Impala, create an internal table. When an internal table is deleted, the metadata and data in the table are deleted together.

- If data is to be processed by multiple tools (such as Pig), create an external table. When an external table is deleted, only metadata is deleted.
- Create a table based on existing tables. Use CREATE LIKE to fully copy the original table structure, including the storage format.
- Create a table based on query results using CREATE AS SELECT.
This method is flexible. By using this method, you can specify fields (except for the storage format) that you want to copy to the new table when copying the structure of the existing table.

Sample Code

```
-- Create the external table employees_info.
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info
(
  id INT,
  name STRING,
  usd_flag STRING,
  salary DOUBLE,
  deductions MAP<STRING, DOUBLE>,
  address STRING,
  entrytime STRING
)
-- Specify the field delimiter.
-- "delimited fields terminated by" indicates that a column delimiter is a comma (,).
ROW FORMAT delimited fields terminated by ',';
-- Specify the table storage format to TEXTFILE.
STORED AS TEXTFILE;

-- Use CREATE Like to create a table.
CREATE TABLE employees_like LIKE employees_info;

-- Run the DESCRIBE command to check the structures of the employees_info and employees_like tables.
DESCRIBE employees_info;
DESCRIBE employees_like;

-- Create the internal table employees_info.
CREATE TABLE IF NOT EXISTS employees_info
(
  id INT,
  name STRING,
  usd_flag STRING,
  salary DOUBLE,
  deductions MAP<STRING, DOUBLE>,
  address STRING,
  entrytime STRING
)
-- Specify the field delimiter.
-- "delimited fields terminated by" indicates that a column delimiter is a comma (,).
ROW FORMAT delimited fields terminated by ',';
-- Specify the table storage format to TEXTFILE.
STORED AS TEXTFILE;
```

Extended Applications

- Create a partitioned table.
A table may have one or more partitions. Each partition is saved as an independent folder in the table directory. Partitions help minimize the query scope, accelerate data query, and allow users to manage data based on certain criteria.
A partition is defined using the PARTITIONED BY clause during table creation.


```
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info_extended
(
  id INT,
  name STRING,
  usd_flag STRING,
  salary DOUBLE,
  deductions MAP<STRING, DOUBLE>,
  address STRING
)
-- Use "PARTITIONED BY" to specify the column name and data type of the partition.
PARTITIONED BY (entrytime STRING)
STORED AS TEXTFILE;
```

- Update the table structure.

After a table is created, you can use ALTER TABLE to add or delete fields, modify table attributes, and add partitions.

```
-- Add the tel_phone and email fields to the employees_info_extended table.
ALTER TABLE employees_info_extended ADD COLUMNS (tel_phone STRING, email STRING);
```

- Enable Impala to use OBS.

You need to set the specified parameters for **core-site.xml** on the cluster management page of Manager. You can log in to OBS console and access the **My Credential** page to obtain the AK/SK.

```
fs.obs.access.key=AK;
fs.obs.secret.key=SK;
fs.obs.endpoint=endpoint;
```

Set the storage type of the new table to **obs**.

```
create table obs(c1 string, c2 string) stored as parquet location 'obs://obs-lmm/hive/orctest'
tblproperties('orc.compress='SNAPPY');
```

NOTE

When Impala uses OBS to store data, partition and table storage locations in the same table cannot be stored in different buckets.

For example, create a partition table and set its storage location to the folder in OBS bucket 1. In this case, modifying the storage location of the table partition does not take effect. When data is inserted, the storage location of the table is used.

1. Create a partition table and specify the path for storing the table.

```
create table table_name(id int,name string,company string) partitioned by(dt date) row
format delimited fields terminated by ',' stored as textfile location "obs://OBS bucket 1/
Folder in the bucket";
```
2. Modifying the storage location of the table partition to another bucket does not take effect.

```
alter table table_name partition(dt date) set location "obs://OBS bucket 2/Folder in the
bucket";
```

19.3.3 Loading Data

Function Description

This section describes how to use Impala SQL to load data to the existing **employees_info** table. You can learn how to load data from a cluster.

Sample Code

```
--Load the employee_info.txt file from the /opt/impala_examples_data/ directory of the local file system
to the employees_info table.
LOAD DATA LOCAL INPATH '/opt/impala_examples_data/employee_info.txt' OVERWRITE INTO TABLE
employees_info;
-- Load /user/impala_examples_data/employee_info.txt from HDFS to the employees_info table.
```

```
LOAD DATA INPATH '/user/impala_examples_data/employee_info.txt' OVERWRITE INTO TABLE
employees_info;
```

NOTE

The essence of loading data is to copy the data to the specified table directory in HDFS. The **LOAD DATA LOCAL INPATH** command can be used to load files from a local file system to Impala. If **LOCAL** is specified, the path refers to the path of the local file system of the currently connected **Impalad**.

19.3.4 Querying Data

Function Description

This section describes how to use Impala SQL to query and analyze data. You can query and analyze data using the following methods:

- Use common features of a SELECT query, such as JOIN.
- Load data to a specified partition.
- Use built-in functions of Impala.
- Query and analyze data using user-defined functions. For details about how to create and define functions, see [User-defined Functions](#).

Sample Code

```
-- Query contact information of employees whose salaries are paid in USD.
SELECT
a.name,
b.tel_phone,
b.email
FROM employees_info a JOIN employees_contact b ON(a.id = b.id) WHERE usd_flag='D';
-- Query the IDs and names of employees who were hired in 2014, and load the query results to the
partition with the hire date of 2014 in the employees_info_extended table.
INSERT OVERWRITE TABLE employees_info_extended PARTITION (entrytime = '2014')
SELECT
a.id,
a.name,
a.usd_flag,
a.salary,
a.deductions,
a.address,
b.tel_phone,
b.email
FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE a.entrytime = '2014';
-- Use the existing function COUNT() in Impala to calculate the number of records in the employees_info
table.
SELECT COUNT(*) FROM employees_info;
-- Query information about employees whose email addresses end with "cn".
SELECT a.name, b.tel_phone FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE
b.email like '%cn';
```

Extended Applications

For details about user-defined functions, see [User-defined Functions](#).

19.3.5 User-defined Functions

When built-in functions of Impala cannot meet requirements, you can compile user-defined functions (UDFs) and use them for query.

According to implementation methods, UDFs are classified as follows:

- Common UDFs: used to perform operations on a single data row and export a single data row.
- User-defined aggregating functions (UDAFs): used to input multiple data rows and export a single data row.
- User-defined table-generating functions (UDTFs): used to perform operations on a single data row and export multiple data rows. **Impala does not support this type of UDFs.**

According to use methods, UDFs are classified as follows:

- Temporary functions: used only in the current session and must be recreated after a session restarts.
- Permanent functions: used in multiple sessions. You do not need to create them every time a session restarts.

Impala supports development of Java UDFs and reuse of UDFs developed by Hive. The prerequisite is that the data types supported by Impala are used. For details about the data types supported by Impala, visit http://impala.apache.org/docs/build3x/html/topics/impala_datatypes.html.

In addition, Impala supports UDFs written in C++, which provides better performance than Java UDFs.

Example

The following is an example of reusing the **lower()** function:

```
[localhost:21000] > create database udfs;
[localhost:21000] > use udfs;
[localhost:21000] > create function my_lower(string)
returns string location '/user/hive/udfs/hive.jar'
symbol='org.apache.hadoop.hive.ql.udf.UDFLower';
[localhost:21000] > select my_lower('Some String NOT ALREADY LOWERCASE');
+-----+
| udfs.my_lower('some string not already lowercase') |
+-----+
| some string not already lowercase |
+-----+
Returned 1 row(s) in 0.11s
[localhost:21000] > create table t2 (s string);
[localhost:21000] > insert into t2 values ('lower'),('UPPER'),('Init cap'),('CamelCase');
Inserted 4 rows in 2.28s
[localhost:21000] > select * from t2;
+-----+
| s |
+-----+
| lower |
| UPPER |
| Init cap |
| CamelCase |
+-----+
Returned 4 row(s) in 0.47s
[localhost:21000] > select my_lower(s) from t2;
+-----+
| udfs.my_lower(s) |
+-----+
| lower |
| upper |
| init cap |
| camelcase |
+-----+
Returned 4 row(s) in 0.54s
```

19.3.6 Sample Program Guide

Function Description

This section describes how to use a sample program to complete an analysis task. This section uses JDBC APIs as an example to describe how to submit a data analysis task.

Sample Code

If you submit a data analysis task using Impala JDBC APIs, refer to **JDBCExample.java** in the sample program.

1. Change the value of the following variable to **false**, specifying that the authentication mode for the connected clusters is normal mode.
2. Define Impala SQL. Impala SQL must be a single statement and cannot contain ";".

```
// Indicates whether the authentication mode of the connected cluster is security mode.
boolean isSecureVer = false;
```

```
// Define HQL, which cannot contain the semicolon (;).
String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT,name STRING)",
"SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};
```

3. Build JDBC URL.

```
// Build JDBC URL.
StringBuilder sBuilder = new StringBuilder(
"jdbc:hive2://").append("impalad_ip").append("/");

if (isSecurityMode) {
    // Security mode
    sBuilder.append(";auth=")
        .append(clientInfo.getAuth())
        .append(";principal=")
        .append(clientInfo.getPrincipal())
        .append(";");
} else {
    // Normal mode
    sBuilder.append(";auth=none");
}
String url = sBuilder.toString();
```

NOTE

If an Impalad instance is directly connected, an Impalad instance fault will cause Impala access failure.

4. Load the Hive JDBC driver.

```
// Load the Hive JDBC driver.
Class.forName(HIVE_DRIVER);
```

5. Enter a correct username, obtain the JDBC connection, confirm the Impala SQL type (DDL/DML), call APIs to run Impala SQL, return the queried column name and result to the console, and close the JDBC connection.

```
Connection connection = null;
try {
    // Obtain the JDBC connection.
    // If you set the second parameter to an incorrect username, the anonymous user will be used
for login.
    connection = DriverManager.getConnection(url, "userName", "");
    // Create a table.
    // To import data to a table after the table is created, you can use the LOAD statement. For
example, import data from HDFS to the table.
    //load data inpath '/tmp/employees.txt' overwrite into table employees_info;
```

```
    execDDL(connection,sqls[0]);
    System.out.println("Create table success!");
    // Query
    execDML(connection,sqls[1]);
    // Delete the table.
    execDDL(connection,sqls[2]);
    System.out.println("Delete table success!");
}
finally {
    // Close the JDBC connection.
    if (null != connection) {
        connection.close();
    }
}

public static void execDDL(Connection connection, String sql)
throws SQLException {
    PreparedStatement statement = null;
    try {
        statement = connection.prepareStatement(sql);
        statement.execute();
    }
    finally {
        if (null != statement) {
            statement.close();
        }
    }
}

public static void execDML(Connection connection, String sql) throws SQLException {
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    ResultSetMetaData resultMetaData = null;

    try {
        // Run the Impala SQL statement.
        statement = connection.prepareStatement(sql);
        resultSet = statement.executeQuery();
        // Output the queried column name to the console.
        resultMetaData = resultSet.getMetaData();
        int columnCount = resultMetaData.getColumnCount();
        for (int i = 1; i <= columnCount; i++) {
            System.out.print(resultMetaData.getColumnLabel(i) + '\t');
        }
        System.out.println();
        // Output the query result to the console.
        while (resultSet.next()) {
            for (int i = 1; i <= columnCount; i++) {
                System.out.print(resultSet.getString(i) + '\t');
            }
            System.out.println();
        }
    }
    finally {
        if (null != resultSet) {
            resultSet.close();
        }

        if (null != statement) {
            statement.close();
        }
    }
}
```

19.4 Application Commissioning

19.4.1 Commissioning Applications on Windows

Step 1 Run the sample project.

After importing and modifying the sample, right-click **ExampleMain.java** in the development environment and choose **ExampleMain.main()** to run the corresponding application project.

NOTE

You can use either of the following method to access an MRS cluster to operate Impala on Windows.

Method 1: Apply for a Windows ECS to access the MRS cluster to operate Impala. Run the sample code after the development environment is installed.

1. On the **Active Clusters** page, click the name of an existing cluster.
On the cluster details page, record the **AZ**, **VPC**, and **Default Security Group** of the Master node.
2. On the ECS management console, create a ECS.
The **AZ**, **VPC**, and **security group** of ECS must be the same as those of the cluster to be accessed.
Select a Windows public image.
For details about other configuration parameters, see [Purchasing an ECS](#).

Method 2: Use the local host to access the MRS cluster to operate Impala. After installing the development environment and completing the following steps, run the sample code.

Bind an EIP address to any Core node. Then, configure the IP address in the **impala-server** configuration item in the **client.properties** file of the development sample to access the Impala service and submit SQL statements.

1. On the ECS page, apply for an EIP and bind it to the ECS. For details, see [Assigning an EIP and Binding It to an ECS](#).
2. Open security group rules for the MRS cluster. Add security group rules to the security groups of Master and Core nodes in the cluster so that the ECS can access the cluster. For details, see [Configuring Security Group Rules](#).

The **client.properties** configuration in the sample is as follows:

```
impala-server = XX.XX.XX.XX:21050 ## Specifies the service address bound to the Core node where the Impalad instance to be connected is located. If method 2 is used, enter the EIP bound in step 1.
```

Step 2 View the execution result.

View the Impala SQL query results in the sample code. If the following information is displayed, the execution is successful.

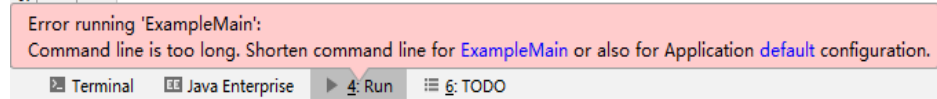
Running ExampleMain and viewing results

```
Create table success!  
_c0  
0  
Delete table success!
```

NOTE

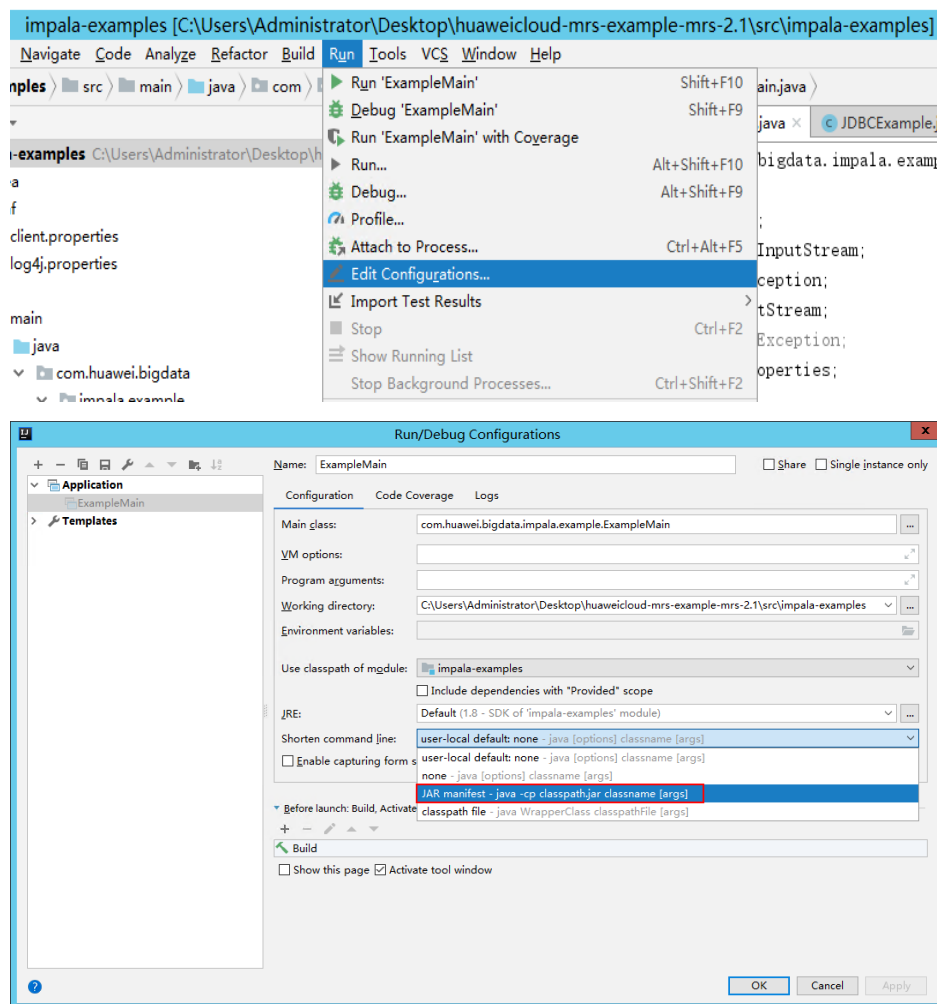
If the sample running fails, the following information is displayed:

Error running 'ExampleMain': Command line is too long. Shorten command line for ServiceStarter or also for Application default configuration.



Solution:

In IntelliJ, set **shorten command line** in **Edit Configurations**.



----End

19.4.2 Commissioning Applications on Linux

Prerequisites

The MRS client has been installed. For details, see:

- For how to install the MRS client on the Master or Core node in a cluster, see [Installing a Client](#).

- For how to install the client outside the MRS cluster, see [Installing a Client](#).

Procedure

- Step 1** Create a directory as the running directory in the running and commissioning environment, for example, `/opt/impala_examples`, and create the `conf` subdirectory in the directory. Upload `client.properties` in the sample code folder `impala-examples-normal` to the `/opt/impala_examples/conf` folder in the sample code folder on the Linux OS, and enter the IP address of Impalad in `client.properties`.
- Step 2** Run the `mvn package` command in the `cmd` window or IntelliJ to obtain a JAR package, for example, `impala-examples-mrs-2.1-jar-with-dependencies.jar` obtained from the `target` directory of the project and copy the package to the `/opt/impala_examples` directory.
- Step 3** Run the following commands in the Linux environment to run the sample program in `bigdata_env` on the `source` client:

```
chmod +x /opt/impala_examples -R
cd /opt/impala_examples
java -cp impala-examples-mrs-2.1-jar-with-dependencies.jar
com.huawei.bigdata.impala.example.ExampleMain
```

- Step 4** In the CLI, view the Impala SQL query results in the example code.

If the following information is displayed, the sample project execution is successful on Linux.

```
Create table success!
_c0
0
Delete table success!
```


19.5.1 JDBC

Impala uses Hive JDBC APIs that comply with the Java JDBC driver standard. For details, see JDK 1.7 API.

NOTE

Impala does not support all standard Hive JDBC APIs. SQLException "**Method not supported**" is generated when some operations are performed.

19.5.2 Impala SQL

Impala SQL is highly compatible with HiveQL. For details, visit https://impala.apache.org/docs/build/html/topics/impala_langref.html.

19.6 Development Specifications

19.6.1 Rules

Load the Hive JDBC Driver

A client program connects to Impalad using JDBC. You need to load the JDBC driver class **org.apache.hive.jdbc.HiveDriver** of Hive first.

Therefore, you must load the driver class using the current class loader before starting the client program.

If there is no JAR file in **classpath**, the client program throws "Class Not Found" and exits.

Example:

```
Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance();
```

Obtain a Database Connection

Use the driver management class **java.sql.DriverManager** of the JDK to obtain an Impalad database connection.

The URL of the Impalad database is as follows: `url="jdbc:hive2://xxx.xxx.xxx.xxx:21050;auth=KERBEROS;principal=impala/hadoop.hadoop.com@HADOOP.COM;user.principal=impala/hadoop.hadoop.com;user.keytab=conf/impala.keytab";`

The username and password are null or empty because authentication has been performed successfully.

//Set up a connection.

```
connection = DriverManager.getConnection(url, "", "");
```

// Execute Impala SQL

Execute Impala SQL. Note that Impala SQL cannot end with a semicolon (;).

Correct example:

```
String sql = "SELECT COUNT(*) FROM employees_info";  
Connection connection = DriverManager.getConnection(url, "", "");  
PreparedStatement statement = connection.prepareStatement(sql);  
resultSet = statement.executeQuery();
```

Incorrect example:

```
String sql = "SELECT COUNT(*) FROM employees_info";  
Connection connection = DriverManager.getConnection(url, "", "");  
PreparedStatement statement = connection.prepareStatement(sql);  
resultSet = statement.executeQuery();
```

Use Impala SQL Syntax to Check Whether a Field is Null

Use **is null** to check whether a field is empty, that is, the field has no value. Use **is not null** to check whether a field is not null, that is, the field has a value.

If you use **is null** to check a field whose type is String and length is 0, **False** is returned. In this case, you need to use **col = ""** to check null character strings, and **col != ""** to check non-null character strings.

Correct example:

```
select * from default.tbl_src where id is null;  
select * from default.tbl_src where id is not null;  
select * from default.tbl_src where name = "";  
select * from default.tbl_src where name != "";
```

Incorrect example:

```
select * from default.tbl_src where id = null;  
select * from default.tbl_src where id != null;  
select * from default.tbl_src where name is null;  
select * from default.tbl_src where name is not null; Note: The id field in the tbl_src table is of the Int type,  
and the name field is of the String type.
```

Multithread Security Login Mode

If multiple threads are performing login operations, the relogin mode must be used for the subsequent logins of all threads after the first successful login of an application.

Login sample code:

```
private Boolean login(Configuration conf){  
    boolean flag = false;  
    UserGroupInformation.setConfiguration(conf);  
  
    try {  
        UserGroupInformation.loginUserFromKeytab(conf.get(PRINCIPAL), conf.get(KEYTAB));  
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "  
+UserGroupInformation.isLoginKeytabBased());  
        flag = true;  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
    return flag;  
}
```

Relogin sample code:

```
public Boolean relogin(){  
    boolean flag = false;  
    try {
```

```

        UserGroupInformation.getLoginUser().reloginFromKeytab();
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "
+UserGroupInformation.isLoginKeytabBased());
        flag = true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return flag;
}

```

Do Not Perform Read and Write Operations on the Same Table at the Same Time

In the current version, Hive does not support concurrent operations. Therefore, do not perform read and write operations on the same table at the same time. Otherwise, query results may be inaccurate or even tasks fail.

19.6.2 Suggestions

Impala SQL Compilation - Implicit Type Conversion Not Supported

If the query statements use the field value for filtering, the implicit type conversion similar to Hive cannot be used to compile Impala SQL.

Impala example:

```

select * from default.tbl_src where id = 10001;
select * from default.tbl_src where name = 'TestName';

```

Hive example (implicit type conversion supported):

```

select * from default.tbl_src where id = '10001';
select * from default.tbl_src where name = TestName;

```

NOTE

In the **tbl_src** table, the **id** field is of the **Int** type and the **name** field is of the **String** type.

JDBC Timeout Limit

Impala uses the Hive JDBC whose implementation has a timeout limit. The default value is 5 minutes. You can set the value using **java.sql.DriverManager.setLoginTimeout (int seconds)**. The unit of **seconds** is second.

19.6.3 Examples

Sample Code of JDBC Secondary Development

The following sample code provides the following functions:

1. In normal (non-Kerberos) mode, uses the username and password for login. If no user is specified, anonymous login is used.
2. Provides the principal for logging in to the Kerberos user in the JDBC URL so that programs can automatically perform secure logins and create Impala connections.

3. Runs Impala SQL statements for creating, querying, and deleting tables.

```
package com.huawei.bigdata.impala.example;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;

/**
 * Simple example for hive jdbc.
 */
public class JDBCExample {
    private static final String HIVE_DRIVER = "org.apache.hive.jdbc.HiveDriver";
    private ClientInfo clientInfo;
    private boolean isSecurityMode;
    public JDBCExample(ClientInfo clientInfo, boolean isSecurityMode){
        this.clientInfo = clientInfo;
        this.isSecurityMode = isSecurityMode;
    }

    /**
     *
     * @throws ClassNotFoundException
     * @throws SQLException
     */
    public void run() throws ClassNotFoundException, SQLException {

        //Define hive sql, the sql can not include ";"
        String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT,name STRING)",
            "SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};

        StringBuilder sBuilder = new StringBuilder(
            "jdbc:hive2://").append(clientInfo.getImpalaServer()).append("/");

        if (isSecurityMode) {
            sBuilder.append(";auth=")
                .append(clientInfo.getAuth())
                .append(";principal=")
                .append(clientInfo.getPrincipal())
                .append(";");
        } else {
            sBuilder.append(";auth=noSasl");
        }
        String url = sBuilder.toString();
        Class.forName(HIVE_DRIVER);
        Connection connection = null;
        try {
            /**
             * Get JDBC connection, If not use security mode, need input correct username,
             * otherwise, wil login as "anonymous" user
             */
            //connection = DriverManager.getConnection(url, "", "");
            connection = DriverManager.getConnection(url);
            /**
             * Run the create table sql, then can load the data if needed. eg.
             * "load data inpath '/tmp/employees.txt' overwrite into table employees_info;"
             */
            execDDL(connection,sqls[0]);
            System.out.println("Create table success!");

            execDML(connection,sqls[1]);

            execDDL(connection,sqls[2]);
            System.out.println("Delete table success!");
        }
        finally {
```

```

        if (null != connection) {
            connection.close();
        }
    }
}

public static void execDDL(Connection connection, String sql)
    throws SQLException {
    PreparedStatement statement = null;
    try {
        statement = connection.prepareStatement(sql);
        statement.execute();
    }
    finally {
        if (null != statement) {
            statement.close();
        }
    }
}

public static void execDML(Connection connection, String sql) throws SQLException {
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    ResultSetMetaData resultMetaData = null;

    try {
        statement = connection.prepareStatement(sql);
        resultSet = statement.executeQuery();

        /**
         * Print the column name to console
         */
        resultMetaData = resultSet.getMetaData();
        int columnCount = resultMetaData.getColumnCount();
        for (int i = 1; i <= columnCount; i++) {
            System.out.print(resultMetaData.getColumnLabel(i) + '\t');
        }
        System.out.println();

        /**
         * Print the query result to console
         */
        while (resultSet.next()) {
            for (int i = 1; i <= columnCount; i++) {
                System.out.print(resultSet.getString(i) + '\t');
            }
            System.out.println();
        }
    }
    finally {
        if (null != resultSet) {
            resultSet.close();
        }

        if (null != statement) {
            statement.close();
        }
    }
}
}
}

```

20 Kafka Development Guide (Security Mode)

20.1 Overview

20.1.1 Development Environment Preparation

Kafka Introduction

Kafka is a distributed message release and subscription system. With features similar to JMS, Kafka processes active streaming data.

Kafka is applicable to message queuing, behavior tracing, operation & maintenance (O&M) data monitoring, log collection, streaming processing, event tracing, and log persistence.

Kafka features:

- High throughput
- Message persistence to disks
- Scalable distributed system
- Fault-tolerant
- Support for online and offline scenarios

Interface Type Introduction

APIs provided by Kafka can be divided into two types: Producer API and Consumer API. Both the types of APIs contain Java API. For details, see section [Java API](#).

20.1.2 Common Concepts

- **Topic**
A same type of messages maintained by Kafka.
- **Partition**

One topic can be divided into multiple partitions, and each partition corresponds to an appendant and log file whose sequence is fixed.

- **Producer**
Role in a Kafka topic to which messages are sent.
- **Consumer**
Role that obtains messages from Kafka topics.
- **Broker**
A node server in a Kafka cluster.
- **keytab file**
A key file for storing user information. Applications use the key file for API authentication in the cluster.

20.1.3 Development Process

Kafka client roles include Producer and Consumer, which share the same application development process.

[Figure 20-1](#) and [Table 20-1](#) show each phase of the development process.

Figure 20-1 Kafka client application development process

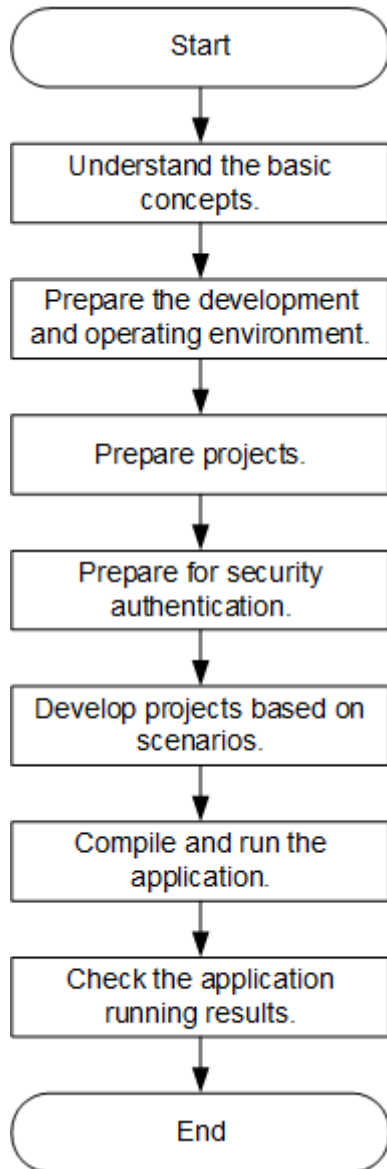


Table 20-1 Kafka client application development process description

Phase	Description	Reference Document
Understand the basic concepts.	Before developing an application, learn basic concepts of Kafka, and determine whether the desired role is Producer or Consumer based on the actual scenario.	Common Concepts

Phase	Description	Reference Document
Prepare the development and operating environment.	The Java language is recommended for the development of Java applications. The IntelliJ IDEA tool can be used. The Kafka running environment is the Kafka client. Install and configure the client according to the guide.	Preparing for Development and Operating Environment
Prepare projects	Kafka provides program samples for different scenarios. You can import the program samples for learning.	Configuring and Importing a Sample Project
Prepare for security authentication.	If you use a security cluster, you need to perform security authentication.	Preparing for Security Authentication
Develop projects based on scenarios.	Producer and Consumer API usage samples are provided and cover API, and multi-thread usage scenarios, helping you quickly know Kafka interfaces well.	Developing an Application
Compile and run the application.	Compile the developed application and submit it for running.	Application Commissioning
View application running results.	Program running results will be written to and printed on the console. You can use a Linux client to consume topic data to check whether data has been successfully written.	Application Commissioning

20.2 Environment Preparation

20.2.1 Preparing for Development and Operating Environment

Preparing Development Environment

Table 20-2 describes the environment required for secondary development.

Table 20-2 Development environment

Item	Description
OS	<ul style="list-style-type: none"> Development environment: Windows OS. Windows 7 or later is supported. Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.
IntelliJ IDEA installation and configuration	<p>It is the basic configuration for the development environment. JDK 1.8 is used. IntelliJ IDEA 2019.1 or other compatible versions are used.</p> <p>NOTE</p> <ul style="list-style-type: none"> If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK. If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK. If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> Oracle JDK versions: 1.8 IBM JDK versions: 1.8.5.11 For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> OpenJDK: 1.8.0_272 <p>NOTE</p> <p>The server supports only TLS V1.2 or later to ensure security.</p> <p>By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls.</p>

Item	Description
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
Developer account preparation	See Preparing a Developer Account for configuration.
7-zip	It is a tool used to decompress .zip and .rar packages. The 7-Zip 16.04 is supported.

Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.

- a. [Log in to the FusionInsight Manager portal](#) and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight_Cluster_1_Services_Client.tar**, decompress it to obtain **FusionInsight_Cluster_1_Services_ClientConfig.tar** file. Then, decompress **FusionInsight_Cluster_1_Services_ClientConfig.tar** file to the **D:\FusionInsight_Cluster_1_Services_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight_Cluster_1_Services_ClientConfig\Kafka\config**. Obtain the Kafka-related configuration file.

[Table 20-3](#) describes the main configuration files.

Table 20-3 Configuration file

Document Name	Function
server.properties	Configures the Kafka server.
client.properties	Configures the Kafka client.
producer.properties	Configures producer of kafka parameters.
consumer.properties	Configures consumer of kafka parameters.

Document Name	Function
user.keytab	Provides user information for Kerberos security authentication.
krb5.conf	Kerberos server configuration information
kafkaSecurityMode	Indicates whether the security mode is enabled for the Kafka service.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
 - a. Install the client on the node. For example, the client installation directory is **/opt/client**.
Ensure that the difference between the client time and the cluster time is less than 5 minutes.
For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).
 - b. [Using an MRS Client on Nodes Outside a Cluster](#). Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all JAR package in the **FusionInsight_Cluster_1_Services_ClientConfig/Kafka/install_files/kafka/libs** directory to the **lib** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/client/lib**.
For example, if the client software package is **FusionInsight_Cluster_1_Services_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

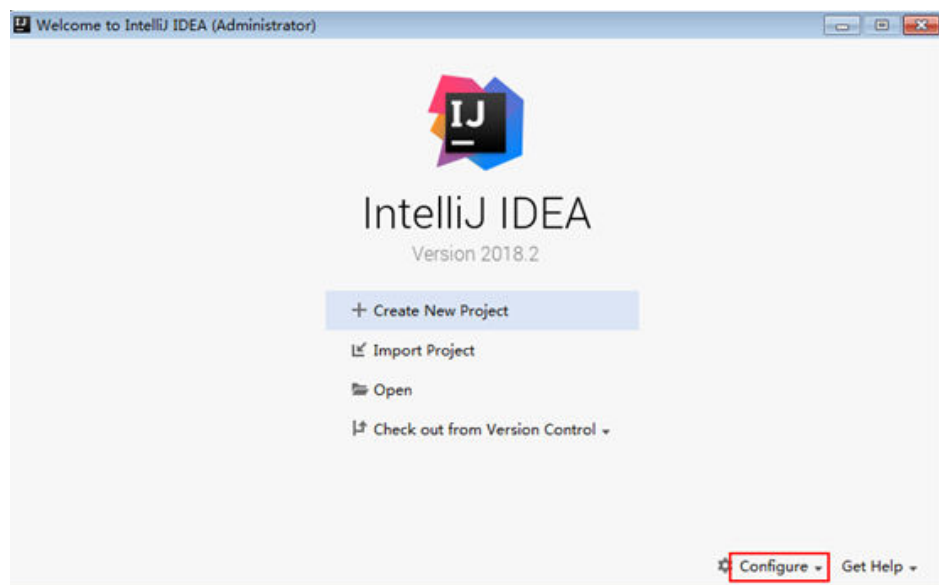
```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Kafka/install_files/kafka/libs/* root@IP address of the client node:/opt/client/lib
```

- c. Check the network connection of the client node.
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

20.2.2 Configuring and Importing a Sample Project

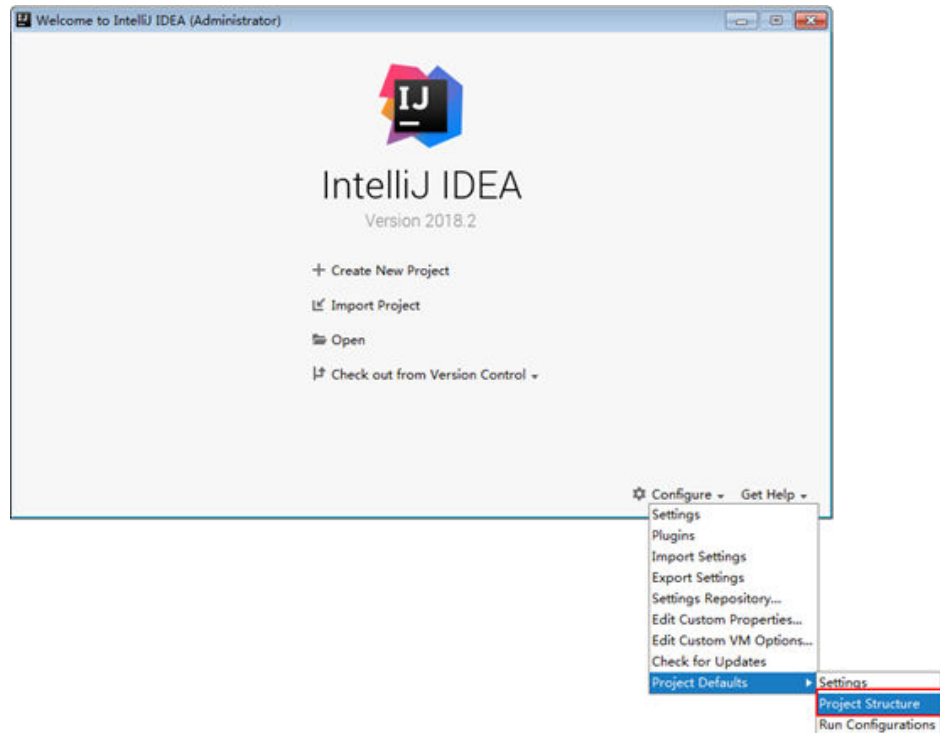
- Step 1** Obtain the sample project folder **kafka-examples** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Configure the **krb5.conf** and **user.keytab** files.
Copy the **krb5.conf** and **keytab** files obtained in [Preparing a Developer Account](#) section and all the cluster configuration file obtained in section [Preparing an Operating Environment](#) to the **kafka-examples\src\main\resources** directory of the sample project.
- Step 3** After installing the IntelliJ IDEA and JDK tools, configure JDK in IntelliJ IDEA.
1. Open IntelliJ IDEA and click **Configure**.

Figure 20-2 Quick Start



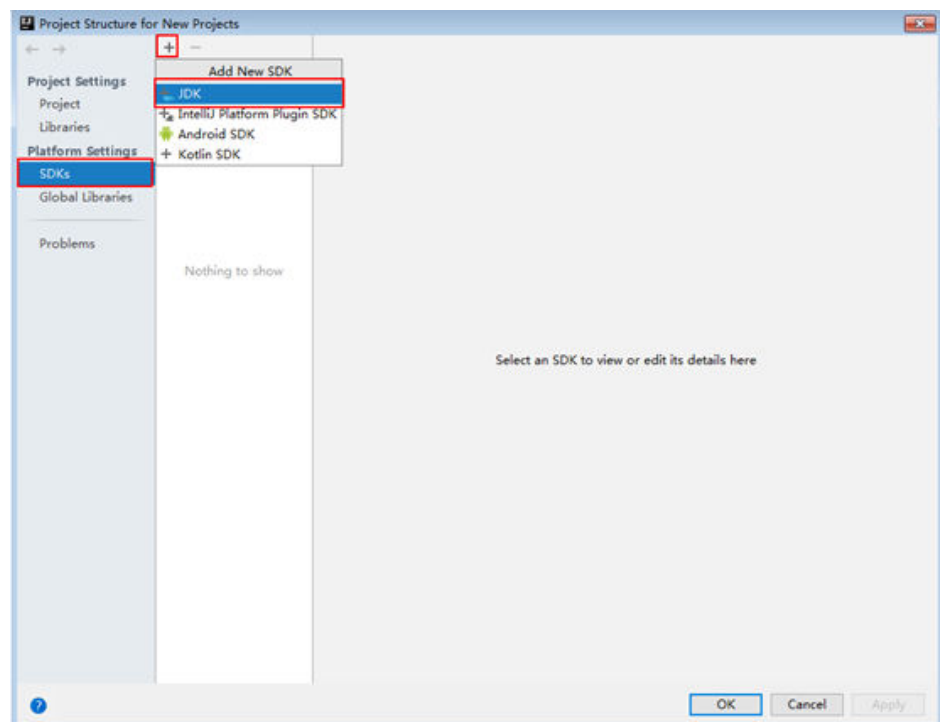
2. Choose **Project Defaults > Project Structure**.

Figure 20-3 Configure



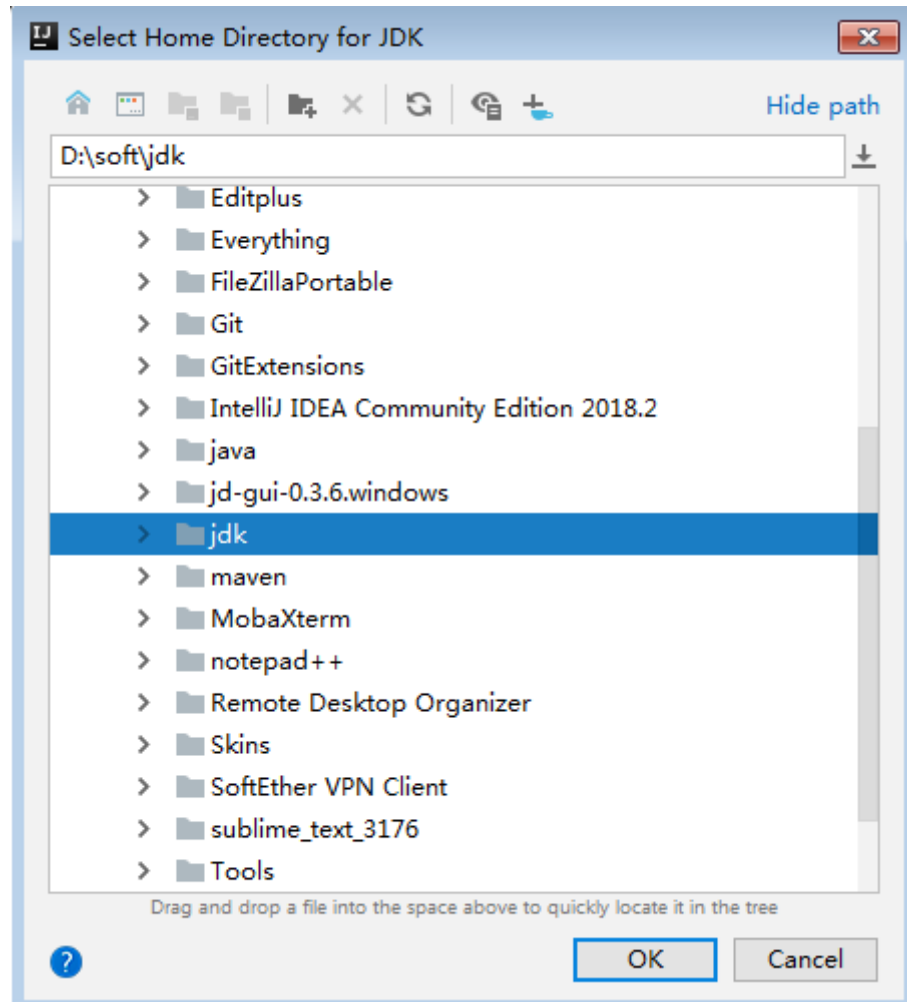
3. In the displayed **Project Structure for New Projects** interface, click **SDKs**. Then click the plus sign (+) and choose **JDK**.

Figure 20-4 Project Structure for New Projects



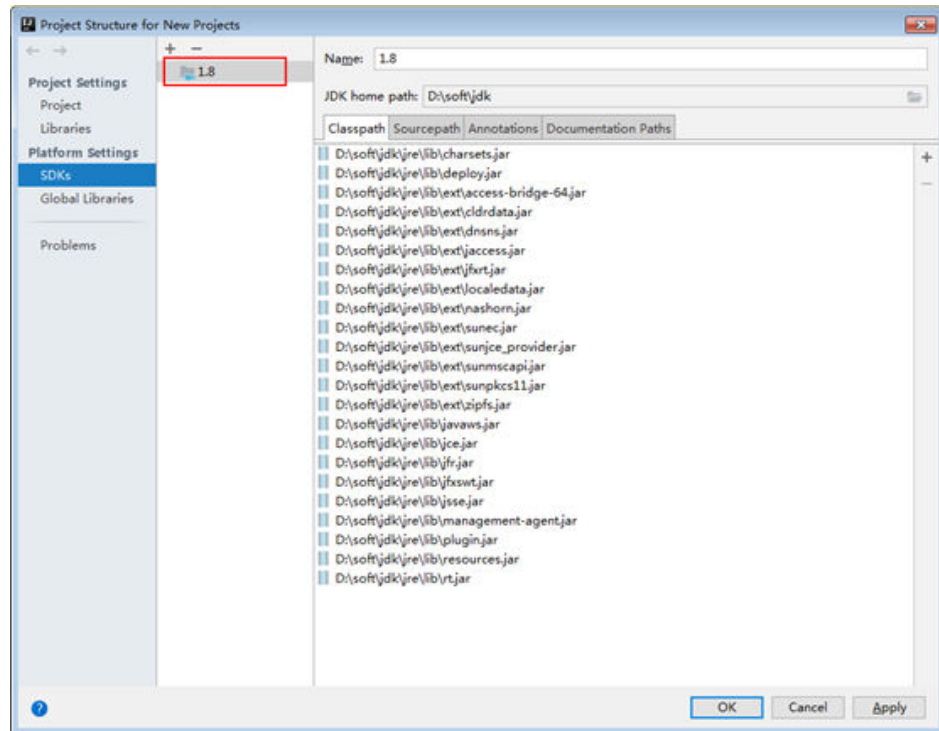
4. In the displayed **Select Home Directory for JDK** interface, select the JDK directory, and click **OK**.

Figure 20-5 Select Home Directory for JDK



5. Click **OK**.

Figure 20-6 Completing the JDK configuration



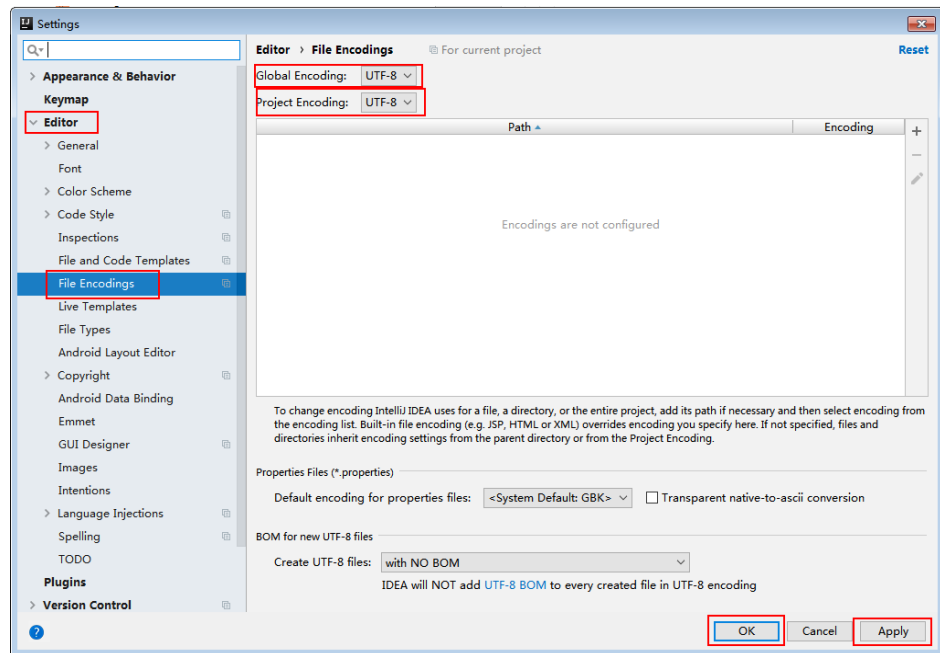
Step 4 Import the sample project into the IntelliJ IDEA development environment.

1. Choose **Open**.
The dialog box for browsing directories is displayed.
2. Select the sample project folder and click **OK**.

Step 5 Set the IntelliJ IDEA text file coding format to prevent invalid characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.
The **Settings** window is displayed.
2. Choose **Editor > File Encodings** in the navigation tree. In the **Project Encoding** area and **Global Encoding** area, set the parameter to **UTF-8**, click **Apply**, and then click **OK**, as shown in [Figure 20-7](#).

Figure 20-7 Setting the IntelliJ IDEA coding format



----End

20.2.3 Preparing for Security Authentication

20.2.3.1 Sasl Kerberos authentication

In a secure cluster environment, components must perform mutual authentication before communicating with each other to ensure communication security. Kafka application development requires Kafka, ZooKeeper and Kerberos security authentication. To perform the security authentication, a **jaas** file needs to be generated, and related environment variables need to be configured. **LoginUtil** related interfaces are provided for the configuration. As shown in the following code sample, only the machine-machine user account applied by the user and the keytab file name need to be configured. For details about the code, reference may be made to the **LoginUtil** class in **com.huawei.bigdata.kafka.example.security** of the sample project.

Code sample:

```
/**
 * keytab file name of the machine-machine account that a user applies for
 */
private static final String USER_KEYTAB_FILE = "keytab file name of the machine-machine account that a
user applies for";

/**
 * Machine-machine account that a user applies for
 */
private static final String USER_PRINCIPAL = "Machine-machine account that a user applies for ";

public static void securityPrepare() throws IOException
{
    String filePath = System.getProperty("user.dir") + File.separator + "src" + File.separator + "main" +
File.separator + "resources" + File.separator;
    String krbFile = filePath + "krb5.conf";
}
```

```
String userKeyTableFile = filePath + USER_KEYTAB_FILE;

//Replace separators in Windows.
userKeyTableFile = userKeyTableFile.replace("\\", "\\\\");
krbFile = krbFile.replace("\\", "\\\\");

LoginUtil.setKrb5Config(krbFile);
LoginUtil.setZookeeperServerPrincipal("zookeeper/hadoop.<system domain name>");
LoginUtil.setJaasFile(USER_PRINCIPAL, userKeyTableFile);
}
```

NOTE

You can log in to the FusionInsight Manager, choose **System > Permission > Domain and Mutual Trust**, and check the value of **Local Domain**, which is the current system domain name.

20.3 Developing an Application

20.3.1 Typical Scenario Description

Scenario

Kafka is a distributed message system, in which messages can be publicized or subscribed. A Producer is to be developed to send a message to a topic of a Kafka cluster every second, and a Consumer is to be implemented to ensure that the topic is subscribed and that messages of the topic are consumed in real time.

Development Idea

1. Use a Linux client to create a topic. For details, see [Shell](#).
2. Develop a Producer to produce data to the topic.
3. Develop a Consumer to consume the data of the topic.

Suggestions on Performance Tuning

1. Create a topic and plan its partitions based on service requirements. The number of partitions limits the number of concurrent consumers.
2. The key value of a message must be variable to ensure even message distribution.
3. Consumers are advised to proactively submit the offset to avoid repeated consumption.

20.3.2 Typical Scenario Sample Code Description

20.3.2.1 Producer API Usage Sample

Function Description

The following code snippet belongs to the **run** method of the **com.huawei.bigdata.kafka.example.Producer** class. It is used by the Producer APIs to produce messages for the security topic.

Code Sample

```
/**
 * The Producer thread executes a function to send messages periodically.
 */
public void run() {
    LOG.info("New Producer: start.");
    int messageNo = 1;

    while (messageNo <= MESSAGE_NUM) {
        String messageStr = "Message_" + messageNo;
        long startTime = System.currentTimeMillis();

        // Construct message records.
        ProducerRecord<Integer, String> record = new ProducerRecord<Integer, String>(topic, messageNo,
messageStr);

        if (isAsync) {
            // Sending in asynchronous mode
            producer.send(record, new DemoCallBack(startTime, messageNo, messageStr));
        } else {
            try {
                // Sending in synchronous mode
                producer.send(record).get();
                long elapsedTime = System.currentTimeMillis() - startTime;
                LOG.info("message(" + messageNo + ", " + messageStr + ") sent to topic(" + topic + ") in " +
elapsedTime + " ms.");
            } catch (InterruptedException ie) {
                LOG.info("The InterruptedException occurred : { }.", ie);
            } catch (ExecutionException ee) {
                LOG.info("The ExecutionException occurred : { }.", ee);
            }
        }
        messageNo++;
    }
}
```

20.3.2.2 Consumer API Usage Sample

Function Description

The following code sample belongs to the **com.huawei.bigdata.kafka.example.Consumer** class. It is used to enable the Consumer API to subscribe a secure topic and consume messages.

Code Sample

```
/**
 * Consumer constructor.
 * @param topic Name of the subscribed topic
 */
public Consumer(String topic) {
    super("KafkaConsumerExample", false);
    // Initializes the configuration parameters required for starting the consumer. For details, see the code.
    Properties props = initProperties();
    consumer = new KafkaConsumer<Integer, String>(props);
    this.topic = topic;
}

public void doWork() {
    // Subscribe
    consumer.subscribe(Collections.singletonList(this.topic));
    // Message consumption request
    ConsumerRecords<Integer, String> records = consumer.poll(waitTime);
    // Message Processing
}
```

```
    for (ConsumerRecord<Integer, String> record : records) {
        LOG.info("[ConsumerExample], Received message: (" + record.key() + ", " + record.value() + ") at
offset " + record.offset());
    }
}
```

20.3.2.3 Multi-thread Producer Sample

Function Description

The multi-thread producer function is implemented based on the code sample described in section [Producer API Usage Sample](#). Multiple producer threads can be started. Each thread sends messages to the partition whose key is the same as the thread ID.

The following code snippet belongs to the run method in the **com.huawei.bigdata.kafka.example.ProducerMultThread** class, and these code snippets are used to enable multiple threads to produce data.

Code Sample

```
/**
 * Specify the current ThreadID as the key value and send data.
 */
public void run()
{
    LOG.info("Producer: start.");

    // Record the number of messages.
    int messageCount = 1;

    // Specify the number of messages sent by each thread.
    int messagesPerThread = 5;
    while (messageCount <= messagesPerThread)
    {

        // Specify the content of messages to be sent.
        String messageStr = new String("Message_" + sendThreadId + "_" + messageCount);

        // Specify a key value for each thread to enable the thread to send messages to only a specified
        partition.
        String key = String.valueOf(sendThreadId);

        // Send messages.
        producer.send(new KeyedMessage<String, String>(sendTopic, key, messageStr));
        LOG.info("Producer: send " + messageStr + " to " + sendTopic + " with key: " + key);
        messageCount++;
    }
}
```

20.3.2.4 Multi-thread Consumer Sample

Function Description

The multi-thread consumer function is implemented based on the code sample described in section [Consumer API Usage Sample](#). The number of consumer threads that can be started to consume the messages in partitions is the same as the number of partitions in the topic.

The following code snippet belongs to the run method in the **com.huawei.bigdata.kafka.example.ConsumerMultThread** class, and these code

snippets are used to implement concurrent consumption of messages in a specified topic.

Code Sample

```
/**
 * Start the concurrent multi-thread consumer.
 */
public void run() {
    LOG.info("Consumer: start.");
    Properties props = Consumer.initProperties();
    // Start a specified number of consumer threads to consume.
    // Note: When this parameter is larger than the number of partitions of the topic to be consumed, the
    extra threads fails to consume data.
    for (int threadNum = 0; threadNum < CONCURRENCY_THREAD_NUM; threadNum++) {
        new ConsumerThread(threadNum, topic, props).start();
        LOG.info("Consumer Thread " + threadNum + " Start.");
    }
}

private class ConsumerThread extends ShutdownableThread {
    private int threadNum = 0;
    private String topic;
    private Properties props;
    private KafkaConsumer<String, String> consumer = null;

    /**
     * Construction method of the consumer thread class
     *
     * @param threadNum Thread number
     * @param topic topic
     */
    public ConsumerThread(int threadNum, String topic, Properties props) {
        super("ConsumerThread" + threadNum, true);
        this.threadNum = threadNum;
        this.topic = topic;
        this.props = props;
        this.consumer = new KafkaConsumer<String, String>(props);
    }

    public void doWork() {
        consumer.subscribe(Collections.singleton(this.topic));
        ConsumerRecords<String, String> records = consumer.poll(waitTime);
        for (ConsumerRecord<String, String> record : records) {
            LOG.info("Consumer Thread-" + this.threadNum + " partitions:" + record.partition() + " record: "
                + record.value() + " offsets: " + record.offset());
        }
    }
}
```

20.3.3 Kafka Streams Scenario Description

Scenario Description

Kafka Streams is a lightweight stream processing framework provided by Apache Kafka. The input and output of Kafka Streams are stored in the Kafka cluster.

The following describes the most common WordCount samples.

Development Guideline

1. Create two topics on the Linux client to serve as the input and output topics.
2. Develop a Kafka Streams to implement the word count function. The system collects statistics on the number of words in each message by reading the

message in the input topic, consumes data from the output topic, and outputs the statistical result in the form of a key-value pair.

20.3.4 Kafka Streams Sample Code Description

20.3.4.1 High level KafkaStreams API Usage Sample

Function Description

The following code snippets are used in the **createWordCountStream** method of the **com.huawei.bigdata.kafka.example.WordCountDemo** class to implement the following function:

Collects statistics on input records. Same words are divided into a group, which is used as a key value. The occurrence times of each word are calculated as a value and are output in the form of a key-value pair.

Code Sample

```
static void createWordCountStream(final StreamBuilder builder) {
    // Receive the input records from input-topic.
    final KStream<String, String> source = builder.stream(INPUT_TOPIC_NAME);

    // Aggregate the calculation result of the key-value pair.
    final KTable<String, Long> counts = source
        // Process the received records and split according to the regular expression REGEX_STRING.
        .flatMapValues(value ->
Arrays.asList(value.toLowerCase(Locale.getDefault()).split(REGEX_STRING)))
        // Aggregate the calculation result of the key-value pair.
        .groupBy((key, value) -> value)
        // The final calculation result
        .count();

    // Output the key-value pair of the calculation result from the output topic.
    counts.toStream().to(OUTPUT_TOPIC_NAME, Produced.with(Serdes.String(), Serdes.Long()));
}
```

20.3.4.2 Low level KafkaStreams API Usage Sample

Function Description

The following code snippets are used in the **com.huawei.bigdata.kafka.example.WordCountProcessorDemo** class to implement the following function:

Collects statistics on input records. Same words are divided into a group, which is used as a key value. The occurrence times of each word are calculated as a value and are output in the form of a key-value pair.

Code Sample

```
private static class MyProcessorSupplier implements ProcessorSupplier<String, String> {
    @Override
    public Processor<String, String> get() {
        return new Processor<String, String>() {
            // ProcessorContext instance, which provides the access of the metadata of the records being
            processed
            private ProcessorContext context;
```

```
private KeyValueStore<String, Integer> kvStore;

@Override
@SuppressWarnings("unchecked")
public void init(ProcessorContext context) {
    // Save processor context in the local host, because it will be used for punctuate() and commit().
    this.context = context;
    // Execute punctuate() once every second.
    this.context.schedule(Duration.ofSeconds(1), PunctuationType.STREAM_TIME, timestamp -> {
        try (final KeyValueIterator<String, Integer> iter = kvStore.all()) {
            System.out.println("----- " + timestamp + " ----- ");
            while (iter.hasNext()) {
                final KeyValue<String, Integer> entry = iter.next();
                System.out.println "[" + entry.key + ", " + entry.value + "]";
                // Send the new records to the downstream processor as key-value pairs.
                context.forward(entry.key, entry.value.toString());
            }
        }
    });
    // Search for the key-value states storage area named KEY_VALUE_STATE_STORE_NAME to
    memorize the recently received input records.
    this.kvStore = (KeyValueStore<String, Integer>)
    context.getStateStore(KEY_VALUE_STATE_STORE_NAME);
}

// Process the receiving records of input topic. Split the records into words, and count the words.
@Override
public void process(String dummy, String line) {
    String[] words = line.toLowerCase(Locale.getDefault()).split(REGEX_STRING);

    for (String word : words) {
        Integer oldValue = this.kvStore.get(word);

        if (oldValue == null) {
            this.kvStore.put(word, 1);
        } else {
            this.kvStore.put(word, oldValue + 1);
        }
    }
}

@Override
public void close() {
}
};
}
```

20.3.5 Kafka Token Authentication Mechanism Scenario

Scenario

The token authentication mechanism is a lightweight authentication mechanism that does not require Kerberos authentication. It can be used in APIs.

Development Guideline

1. Use a Linux client to create a token. For details, see [Kafka Token Authentication Mechanism Tool Usage](#).
2. Use the token mechanism for authentication in the secondary development sample.

20.3.6 Sample Code of the Kafka Token Authentication Mechanism

Function

The token authentication mechanism can be used for authentication.

Sample Code

Add the token configuration to the **Producer()** method.

```
public static Properties initProperties() {
    Properties props = new Properties();
    KafkaProperties kafkaProc = KafkaProperties.getInstance();

    // Broker address list
    props.put(BOOTSTRAP_SERVER, kafkaProc.getValues(BOOTSTRAP_SERVER, "localhost:21007"));
    // Client ID
    props.put(CLIENT_ID, kafkaProc.getValues(CLIENT_ID, "DemoProducer"));
    // Key serialization class
    props.put(KEY_SERIALIZER,
        kafkaProc.getValues(KEY_SERIALIZER, "org.apache.kafka.common.serialization.StringSerializer"));
    // Value serialization class
    props.put(VALUE_SERIALIZER,
        kafkaProc.getValues(VALUE_SERIALIZER, "org.apache.kafka.common.serialization.StringSerializer"));
    // Protocol type: Currently, the SASL_PLAINTEXT or PLAINTEXT protocol types can be used.
    props.put(SEcurity_PROTOCOL, kafkaProc.getValues(SEcurity_PROTOCOL, "SASL_PLAINTEXT"));
    // Service name
    props.put(SASL_KERBEROS_SERVICE_NAME, "kafka");
    // Domain name
    props.put(KERBEROS_DOMAIN_NAME, kafkaProc.getValues(KERBEROS_DOMAIN_NAME,
        "hadoop.hadoop.com"));
    // Partition class name
    props.put(PARTITIONER_NAME,
        kafkaProc.getValues(PARTITIONER_NAME, "com.huawei.bigdata.kafka.example.SimplePartitioner"));
    // Generate token configurations.
    StringBuilder token = new StringBuilder();
    String LINE_SEPARATOR = System.getProperty("line.separator");
    token.append("org.apache.kafka.common.security.scram.ScramLoginModule
required").append(LINE_SEPARATOR);
    /**
     * TOKENID generated by the user.
     */
    token.append("username=\\"PPVz2cxuQC-okwJVZnFKFg\\").append(LINE_SEPARATOR);
    /**
     * Token HMAC generated by the user.
     */
    token.append("password=\\"pL5nHsIUODg5u0dRM+o62cOIf/j6yATSt6uaPBYfIb29dj/
jbpiAnRGSWDJ6tL4KXo89dot0axcRIDsMagyN4g=\\"").append(LINE_SEPARATOR);
    token.append("tokenauth=true;");
    // Use SCRAM-SHA-512 as the SASL mechanism for the user.
    props.put("sasl.mechanism", "SCRAM-SHA-512");
    props.put("sasl.jaas.config", token.toString());

    return props;
}
```

 NOTE

When using the token authentication mechanism, you need to comment out the Kerberos authentication mechanism to ensure that only one authentication mechanism is used during code running, as shown in the following:

```
public static void main(String[] args)
{
    if (isSecurityModel())
    {
        // try
        // {
        //     LOG.info("Securitymode start.");
        //     //!!!Note: When using security authentication, you need to manually change the account
        //     //to a machine-machine one.
        //     securityPrepare();
        // }
        // catch (IOException e)
        // {
        //     LOG.error("Security prepare failure.");
        //     LOG.error("The IOException occurred.", e);
        //     return;
        // }
        LOG.info("Security prepare success.");
    }

    // Specify whether to use the asynchronous sending mode.
    final boolean asyncEnable = false;
    Producer producerThread = new Producer(KafkaProperties.TOPIC, asyncEnable);
    producerThread.start();
}
```

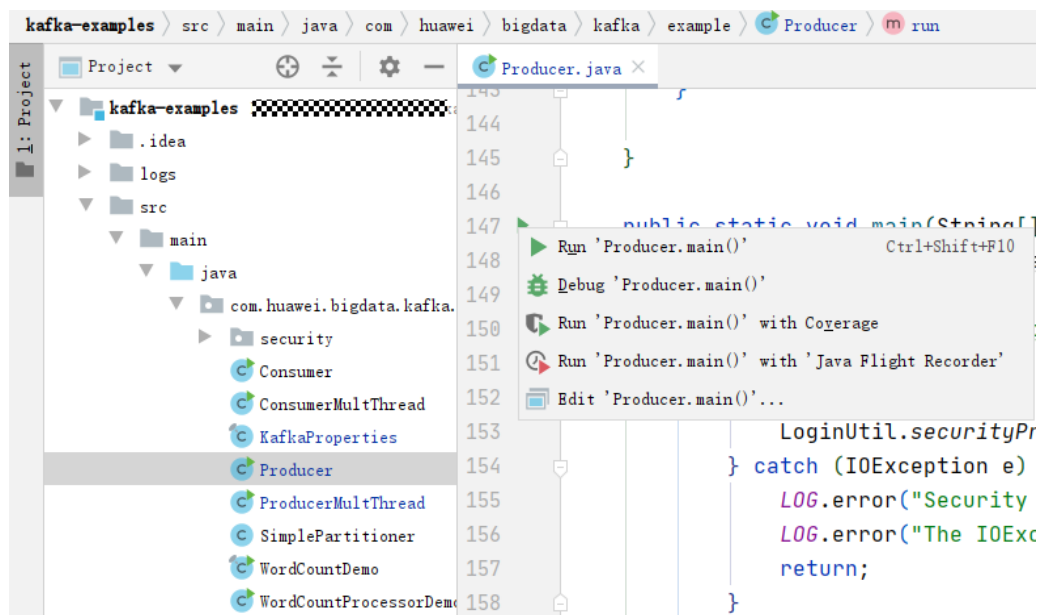
20.4 Application Commissioning

20.4.1 Commissioning an Application in Windows

Starting Producer

- Step 1** Ensure that the mappings between the host names and service IP addresses of all the hosts in the remote cluster are configured in the local **hosts** file.
- Step 2** Run **Producer.java** on IntelliJ IDEA, as shown in [Figure 20-8](#).

Figure 20-8 Run Producer.java on IntelliJ IDEA



Step 3 The console window appears. You can find that Producer is sending messages to the default topic (example-metric1). One piece of log is printed when every 10 messages are sent.

Figure 20-9 Procduder running window

```
[2019-06-12 10:31:37,865] INFO Updated cluster metadata version 2 to Cluster(id = 1cPugjnSQAernMH8RS_-jA, nodes = [187.
[2019-06-12 10:31:51,729] INFO Updated cluster metadata version 3 to Cluster(id = 1cPugjnSQAernMH8RS_-jA, nodes = [187.
[2019-06-12 10:31:53,140] INFO The Producer have send 10 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:31:54,516] INFO The Producer have send 20 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:31:55,906] INFO The Producer have send 30 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:31:57,299] INFO The Producer have send 40 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:31:58,686] INFO The Producer have send 50 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:32:00,070] INFO The Producer have send 60 messages. (com.huawei.bigdata.kafka.example.NewProducer)
```

----End

Starting Consumer

Step 1 Run the **Consumer.java** file.

Step 2 Click **Run**. In the console window that appears, you can find that Producer starts after Consumer successfully starts, and then you can view messages received in real time.

Figure 20-10 Consumer.java running window

```
[2019-06-23 17:49:48,609] INFO [Consumer clientId=consumer-1, groupId=DemoConsumer] (Re-)joining group (org.apache.kafka.clients.consumer.internals.AbstractCoordinator)
[2019-06-23 17:49:51,865] INFO [Consumer clientId=consumer-1, groupId=DemoConsumer] Successfully joined group with generation 5 (org.apache.kafka.clients.consumer.interr
[2019-06-23 17:49:51,866] INFO [Consumer clientId=consumer-1, groupId=DemoConsumer] Setting newly assigned partitions [example-metric1-0, example-metric1-1] (org.apache
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (1, Message_1) at offset 188 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (5, Message_5) at offset 189 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (9, Message_9) at offset 190 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (9, Message_9) at offset 191 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (15, Message_15) at offset 192 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (18, Message_18) at offset 193 (com.huawei.bigdata.kafka.example.NewConsumer)
```

----End

Starting Other Code Samples

The procedures for starting other code samples are similar to the procedures for starting Producer and Consumer in this section.

20.4.2 Commissioning an Application in Linux

Scenario

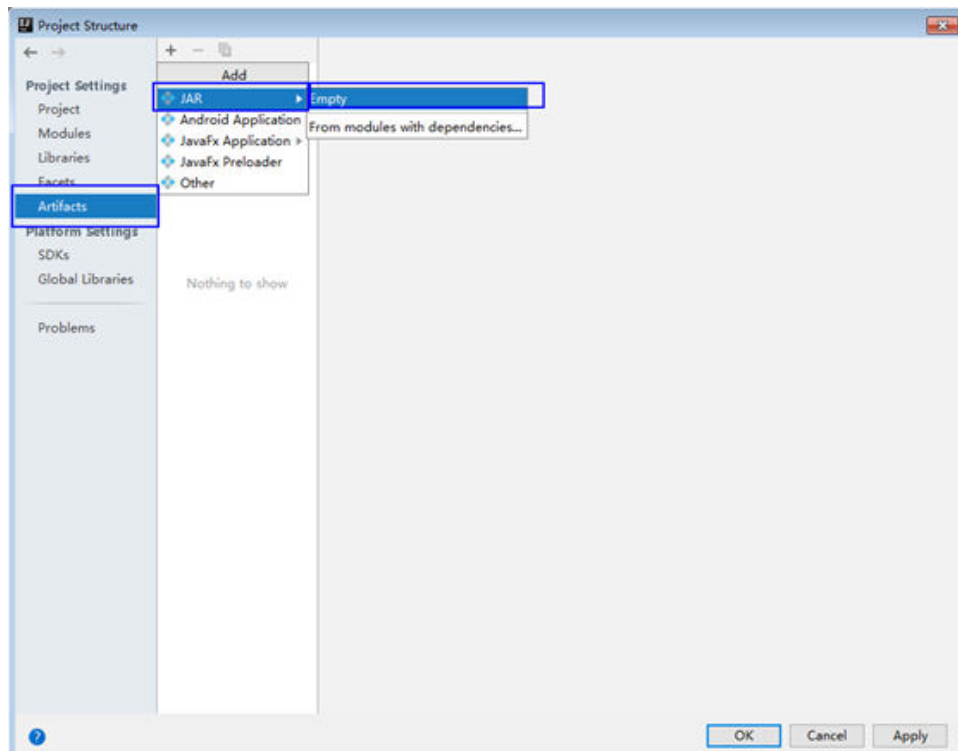
Run a sample program in Linux after code development is complete.

Procedure

Step 1 In IntelliJ IDEA, configure the Artifacts information for the project before the JAR package is generated.

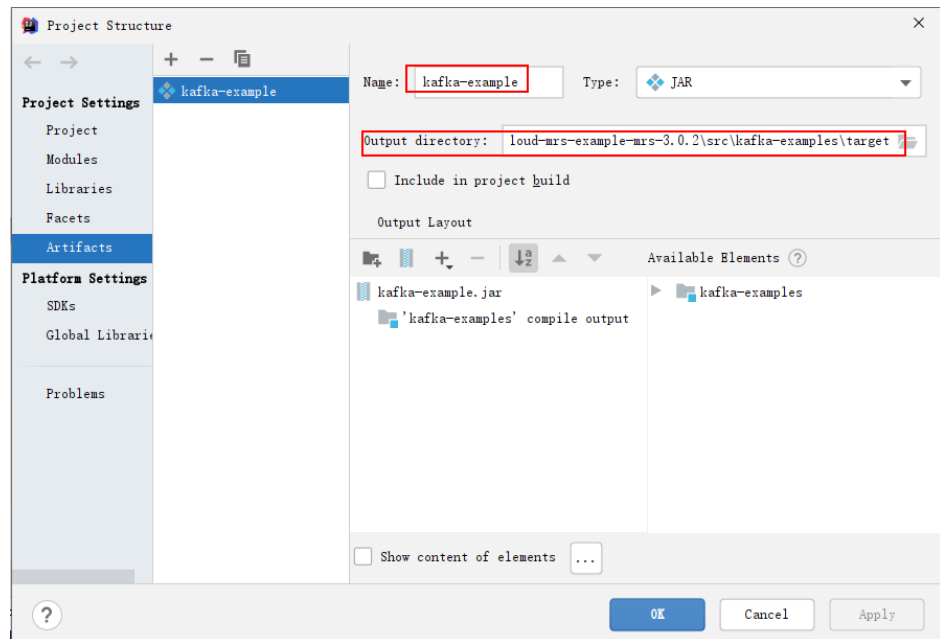
1. Enter IntelliJ IDEA and choose **File > Project Structure**.
2. On the **Project Structure** page, select **Artifacts**, click the plus sign (+), and choose **JAR > Empty**.

Figure 20-11 Add Artifacts



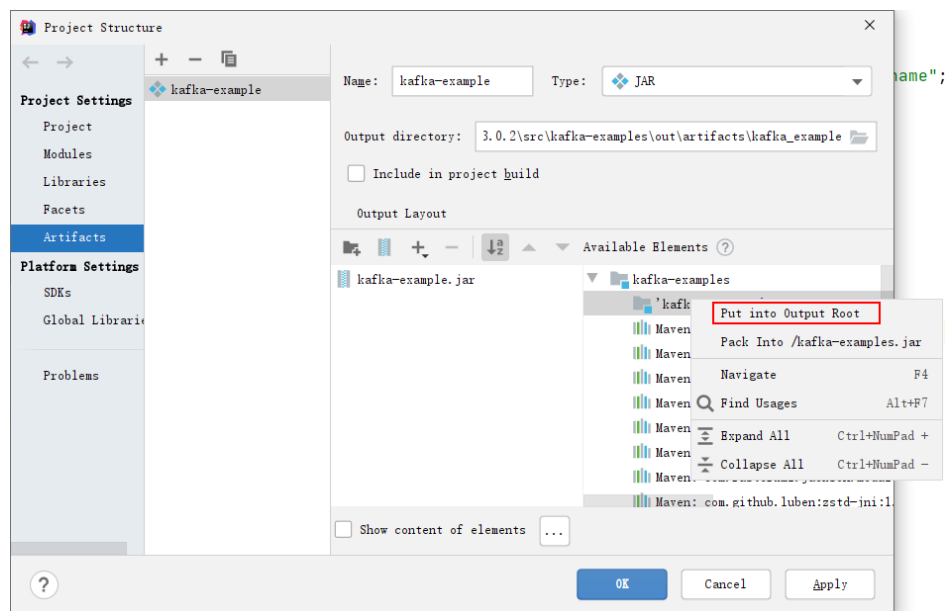
3. Set the name, type, and output path of the JAR package based on site requirements.

Figure 20-12 Configuring basic information



4. Choose '**kafka-examples**' compile output, right-click **Put into Output Root**, and click **Apply**.

Figure 20-13 Put into Output Root

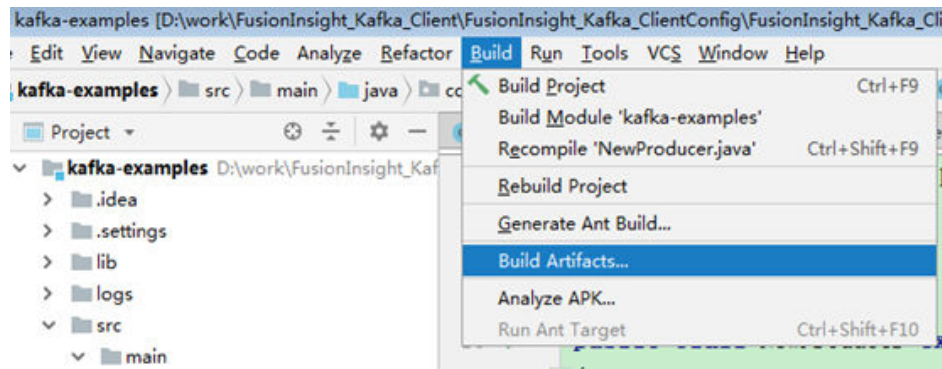


5. Click **OK** to complete the configuration.

Step 2 Generate the JAR package.

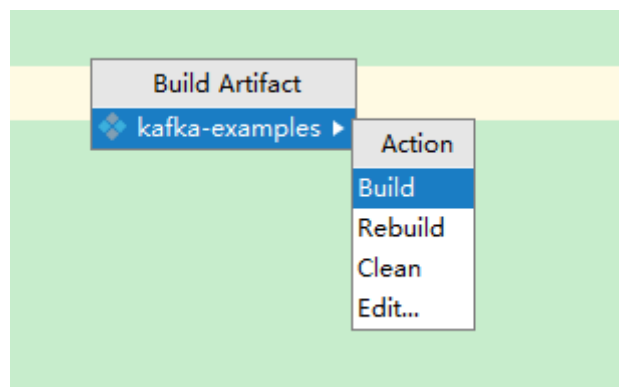
1. Enter IntelliJ IDEA and choose **Build > Build Artifacts**.

Figure 20-14 Build Artifacts



2. In the displayed interface, choose **kafka-example:jar** > **Build** to generate the JAR package.

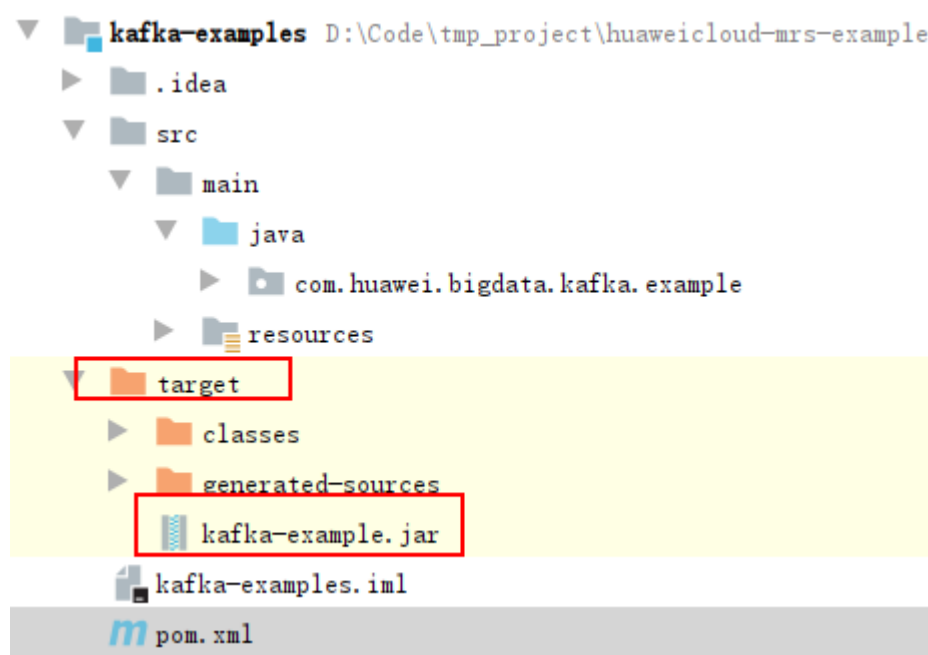
Figure 20-15 Build



3. If log similar to the following is displayed in event logs, the JAR package is successfully generated. Obtain the JAR package from the **Output directory** configured shown in [Figure 20-12](#)
14:37 Compilation completed successfully in 25 s 991 ms

Step 3 Copy the generated jar package to **/opt/client/lib**.

Figure 20-16 File location



Step 4 Copy all files in the **src/main/resources** directory of the IntelliJ IDEA project to the **src/main/resources** directory at the same level as the **lib** folder, that is, **/opt/client/src/main/resources**.

Step 5 Ensure that the current user has read permission of all the files in the **src/main/resources** and **lib** folders in **/opt/client**, jdk has been installed, and java environment variables are set. Then, run the command, for example, **java -cp /opt/client/lib/*:/opt/client/src/main/resources com.huawei.bigdata.kafka.example.Producer** to run the example project.

----End

20.4.3 Kafka Streams Sample Running Guide

20.4.3.1 High level Streams API Sample Usage Guide

1. Open sample code **WordCountDemo.java** in the IDE and change the following two variables to the machine-machine account name and keytab file that you apply for.

```
// Use the machine-machine account keytab file that you apply for.  
private static final String USER_KEYTAB_FILE = "Change it to the real-world keytab file name.;"  
// Use the machine-machine account name that you apply for.  
private static final String USER_PRINCIPAL = "Change it to the real-world username.;"
```

2. Use the Linux client to create the input and output topics. Ensure that the topic names are the same as those in the sample code, set the policy for clearing the output topic to compact, and run the sample code.

```
// source-topic name created by the user, that is, input topic  
private static final String INPUT_TOPIC_NAME = "streams-wordcount-input";  
// sink-topic name created by the user, that is, output topic  
private static final String OUTPUT_TOPIC_NAME = "streams-wordcount-output";
```

- Create the input topic:

- ```
kafka-topics.sh --create --zookeeper
192.168.0.11:2181,192.168.0.12:2181,192.168.0.13:2181/kafka --
replication-factor 1 --partitions 1 --topic streams-wordcount-input
```
- Create the output topic:

```
kafka-topics.sh --create --zookeeper
192.168.0.11:2181,192.168.0.12:2181,192.168.0.13:2181/kafka --
replication-factor 1 --partitions 1 --topic streams-wordcount-output
--config cleanup.policy=compact
```
  - Run sample code **WordCountDemo.java**. For details, see [Commissioning an Application in Windows](#) and [Commissioning an Application in Linux](#).
3. Use the Linux client to write messages and view the statistics result.
- Run the **kafka-console-producer.sh** command to write messages to the input topic.
- ```
# kafka-console-producer.sh --broker-list 192.168.0.11:21007,192.168.0.12:21007,192.168.0.13:21007 --
topic streams-wordcount-input --producer.config /opt/client/Kafka/kafka/config/producer.properties
>This is Kafka Streams test
>test starting
>now Kafka Streams is running
>test end
>
```

Run the **kafka-console-consumer.sh** command to consume data from the output topic and view the statistics result.

```
# kafka-console-consumer.sh --topic streams-wordcount-output --bootstrap-server
192.168.0.11:21007,192.168.0.12:21007,192.168.0.13:21007 --consumer.config /opt/client/Kafka/kafka/
config/consumer.properties --from-beginning --property print.key=true --property print.value=true --
property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer --property
value.deserializer=org.apache.kafka.common.serialization.LongDeserializer --formatter
kafka.tools.DefaultMessageFormatter
this 1
is 6
kafka 12
streams 8
test 8
test 9
starting 1
now 1
kafka 13
streams 9
is 7
running 1
test 10
end 1
```

20.4.3.2 Low level Streams API Sample Usage Guide

1. Open sample code **WordCountProcessorDemo.java** in the IDE and change the following two variables to the machine-machine account name and keytab file that you apply for.

```
// Use the machine-machine account keytab file that you apply for.
private static final String USER_KEYTAB_FILE = "Change it to the real-world keytab file name";
// Use the machine-machine account name that you apply for.
private static final String USER_PRINCIPAL = "Change it to the real-world username. ";
```
2. Use the Linux client to create the input and output topics. Ensure that the topic names are the same as those in the sample code, set the policy for clearing the output topic to compact, and run the sample code.

```
// source-topic name created by the user, that is, input topic
private static final String INPUT_TOPIC_NAME = "streams-wordcount-processor-input";
```



```
// sink-topic name created by the user, that is, output topic  
private static final String OUTPUT_TOPIC_NAME = "streams-wordcount-processor-output";
```

- Create the input topic:

```
kafka-topics.sh --create --zookeeper  
192.168.0.11:2181,192.168.0.12:2181,192.168.0.13:2181/kafka --  
replication-factor 1 --partitions 1 --topic streams-wordcount-  
processor-input
```

- Create the output topic:

```
kafka-topics.sh --create --zookeeper  
192.168.0.11:2181,192.168.0.12:2181,192.168.0.13:2181/kafka --  
replication-factor 1 --partitions 1 --topic streams-wordcount-  
processor-output--config cleanup.policy=compact
```

- Run sample code **WordCountProcessorDemo.java**. For details, see [Commissioning an Application in Windows](#) and [Commissioning an Application in Linux](#).

3. Use the Linux client to write messages and view the statistics result.

Run the **kafka-console-producer.sh** command to write messages to the input topic.

```
# kafka-console-producer.sh --broker-list 192.168.0.11:21007,192.168.0.12:21007,192.168.0.13:21007 --  
topic streams-wordcount-processor-input --producer.config /opt/client/Kafka/kafka/config/  
producer.properties  
>This is Kafka Streams test  
>now Kafka Streams is running  
>test end  
>
```

Run the **kafka-console-consumer.sh** command to consume data from the output topic and view the statistics result.

```
# kafka-console-consumer.sh --topic streams-wordcount-processor-output --bootstrap-server  
192.168.0.11:21007,192.168.0.12:21007,192.168.0.13:21007 --consumer.config /opt/client/Kafka/kafka/  
config/consumer.properties --from-beginning --property print.key=true --property print.value=true  
is 1  
kafka 1  
streams 1  
test 1  
this 1  
end 1  
is 2  
kafka 2  
now 1  
running 1  
streams 2  
test 2  
this 1
```

20.4.4 Running Guide of the Kafka Token Authentication Mechanism Sample Code

Procedure

Kafka Server Configuration

- Step 1** Log in to FusionInsight Manager and choose **Cluster** > *Name of the desired cluster* > **Services** > **Kafka**. On the displayed page, click the **Configurations** tab. The Kafka service configuration page is displayed.

Step 2 Enable the token authentication mechanism.

Find the **delegation.token.master.key** parameter, which specifies the master key used to generate and verify tokens. Check whether the parameter has been configured. If it has been configured, and the value is not **null**, the token authentication mechanism has been enabled and does not need to be reconfigured. If the token authentication mechanism is configured again, the original token cannot be used.

Step 3 Specify the SASL authentication mechanism for a specified service.

Find the **sasl.enabled.mechanisms** parameter and set it to **GSSAPI,SCRAM-SHA-256,SCRAM-SHA-512**. Use commas (,) to separate the three items.

Step 4 Log in to a component using Scram

Find the customized parameter **kafka.config.expandor** and set its name to **listener.name.sasl_plaintext.scram-sha-512.sasl.jaas.config**. Set the value to **org.apache.kafka.common.security.scram.ScramLoginModule required**.

Step 5 Log in to FusionInsight Manager and restart all broker instances of the Kafka service.

Kafka Client Configuration

Step 6 Generate a token for the user. For details about how to generate a token, see [Kafka Token Authentication Mechanism Tool Usage](#).

Secondary Development Sample Code Project Configuration

The token authentication mechanism can be used for APIs. Therefore, you can configure the token authentication mechanism in **Producer()** and **Consumer()** of the secondary development sample.

Step 7 Enable the token authentication mechanism.

Set **tokenauth** to **true** on the client.

Step 8 Specify the SASL authentication mechanism for a specified service.

Set **sasl.mechanism** to **SCRAM-SHA-512** on the client.

Step 9 Configure the Java Authentication and Authorization Service (JAAS) file.

Configure **sasl.jaas.config** on the client as follows:

```
org.apache.kafka.common.security.scram.ScramLoginModule required
username="TOKENID"
password="HMAC";
```

Where **TOKENID** and **HMAC** are generated when tokens are generated in [Step 6](#).

 **NOTE**

For details about how to enable the token authentication mechanism, specify the SASL authentication mechanism for a specified service, and configure the JAAS file, see [Sample Code of the Kafka Token Authentication Mechanism](#).

- Step 10** Run the sample code based on the running environment. For details, see section [Commissioning an Application in Windows](#) and [Commissioning an Application in Linux](#).

----End

20.5 More Information

20.5.1 External Interfaces

20.5.1.1 Shell

1. Query the list of topics in current clusters.
bin/kafka-topics.sh --list --zookeeper <ZooKeeper cluster IP address:2181/kafka>
bin/kafka-topics.sh --list --bootstrap-server <Kafka cluster IP address: 21007> --command-config config/client.properties
2. Query the details of a topic.
bin/kafka-topics.sh --describe --zookeeper <ZooKeeper cluster IP address: 2181/kafka> --topic <Topic name>
bin/kafka-topics.sh --describe --bootstrap-server <Kafka cluster IP address:21007> --command-config config/client.properties --topic <Topic name>
3. Delete a topic (this operation can be performed by an administrator).
bin/kafka-topics.sh --delete --zookeeper <ZooKeeperCluster IP address: 2181/kafka> --topic <Topic name>
bin/kafka-topics.sh --delete --bootstrap-server <Kafka cluster IP address: 21007>--command-config config/client.properties --topic <Topic name>
4. Create a topic (this operation can be performed by an administrator).
bin/kafka-topics.sh --create --zookeeper <ZooKeeperCluster IP address: 2181/kafka> --partitions 6 --replication-factor 2 --topic <Topic name>
bin/kafka-topics.sh --create --bootstrap-server <Kafka cluster IP address: 21007> --command-config config/client.properties --partitions 6 --replication-factor 2 --topic <Topic name>
5. Assign permissions to the Consumer (this operation is performed by an administrator).
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=<ZooKeeper cluster IP address:2181/kafka > --add --allow-principal User:<User name> --consumer --topic <Topic name> --group <Consumer group name>
bin/kafka-acls.sh --bootstrap-server <Kafka cluster IP address:21007> --command-config config/client.properties --add --allow-principal User:<User name> --consumer --topic <Topic name> --group <Consumer group name>
6. Assign permissions to the Producer (this operation is performed by an administrator).

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=<ZooKeeper cluster IP address:2181/kafka > --add --allow-principal User:<User name> --producer --topic <Topic name>
```

```
bin/kafka-acls.sh --bootstrap-server <Kafka cluster IP address:21007> --command-config config/client.properties --add --allow-principal User:<User name> --producer --topic <Topic name>
```

7. Produce messages (this operation requires the producer permission of the desired topic).

```
bin/kafka-console-producer.sh --broker-list <KafkaCluster IP address: 21007> --topic <Topic name> --producer.config config/producer.properties
```

8. Consume data (the producer permission of the topic is required).

```
bin/kafka-console-consumer.sh --topic <Topic name> --bootstrap-server <KafkaCluster IP address:21007> --consumer.config config/consumer.properties
```

20.5.1.2 Java API

Versions of interfaces adopted by Kafka are consistent with those in Open Source Community. For details, see <https://kafka.apache.org/24/documentation.html>.

Major Interfaces of a Producer

Table 20-4 Major parameters of a Producer

Parameter	Description	Remarks
bootstrap.servers	Broker address list	The Producer creates connections with the Broker based on this parameter.
security.protocol	Security protocol type	The Producer uses the security protocol of the type specified by this parameter. In security mode, only the SASL protocol is supported, so this parameter needs to be configured as SASL_PLAINTEXT .
sasl.kerberos.service.name	Service name	This parameter specifies the Kerberos user name used by the Kafka cluster for running. This parameter needs to be configured as kafka .
key.serializer	Serialization of key values in messages	This parameter specifies how to serialize the key values in messages.

Parameter	Description	Remarks
value.serializer	Serialization of messages	This parameter specifies how to serialize transmitted messages.

Table 20-5 Major interface functions of a Producer

Return Value	Interface Function	Description
java.util.concurrent.Future<RecordMetadata>	send(ProducerRecord<K, V> record)	Indicates a TX interface without a callback function. Generally, the get() function of Future is used for synchronous transmission.
java.util.concurrent.Future<RecordMetadata>	send(ProducerRecord<K, V> record, Callback callback)	Indicates a TX interface with a callback function. Generally, this interface uses the callback function to process transmission results after asynchronous transmission.
void	onCompletion(RecordMetadata metadata, Exception exception);	Indicates the interface method for a callback function. This method is used to process asynchronous transmission results.

Major Interfaces of a Consumer

Table 20-6 Major parameters of a Consumer

Parameter	Description	Remarks
bootstrap.servers	Broker address list	The Consumer creates connections with the Broker based on this parameter.

Parameter	Description	Remarks
security.protocol	Security protocol type	The Consumer uses the security protocol of the type specified by this parameter. In security mode, only the SASL protocol is supported, so this parameter needs to be configured as SASL_PLAINTEXT .
sasl.kerberos.service.name	Service name	This parameter specifies the Kerberos user name used by the Kafka cluster for running. This parameter needs to be configured as kafka .
key.deserializer	Deserialization of key values in messages	This parameter specifies how to deserialize the key values in messages.
value.deserializer	Deserialization of messages	This parameter specifies how to deserialize received messages.

Table 20-7 Major interface functions of a Consumer

Return Value	Interface Function	Description
void	close()	Indicates the interface method for closing the Consumer.
void	subscribe(java.util.Collection<java.lang.String> topics)	Indicates the interface method for subscribing topics.
ConsumerRecords<K,V>	poll(final Duration timeout)	Indicates the interface method for requesting messages.

20.5.1.3 Security Ports

1. By default, port 21007 is used for access to secure Kafka clusters, and port 9092 is used for access to normal Kafka clusters.
2. API supports access via both ports 9092 and 21007.

20.5.1.4 SSL Encryption Function Used by a Client

Prerequisites

1. Before enabling the SSL function on the client, ensure that the SSL service function on the server has been enabled (**ssl.mode.enable** of the server has been set to **true**).
2. The SSL function requires APIs. For details, see [Safety Instruction on Using Kafka](#).

Description

- **SSL used by a Linux client**
 - a. Change the value of **security.protocol** in the **client installation directory/Kafka/kafka/config/producer.properties** and **client installation directory/Kafka/kafka/config/consumer.properties** directories to **SASL_SSL** or **SSL**.
 - b. When using the Shell commands, enter a port ID corresponding to the protocol set in Step 1. For example, if **security.protocol** is set to **SASL_SSL**, an SASL_SSL protocol port ID is required, which is **21009** by default:

```
bin/kafka-console-producer.sh --broker-list <IP address of a Kafka cluster:21009> --topic <Topic name> --producer.config config/producer.properties
```

```
bin/kafka-console-consumer.sh --topic <Topic name> --bootstrap-server <IP address of a Kafka cluster:21009> --consumer.config config/consumer.properties
```
- **SSL used by a Windows client**
 - a. Download the Kafka client, decompress the client, and find the **ca.crt** file in the root directory.
 - b. Use the **ca.crt** file to generate the TrustStore file of the client.
Run the **keytool -noprompt -import -alias myservercert -file ca.crt -keystore truststore.jks** command in the Java running environment.
 - c. Copy the generated **truststore.jks** file to the **conf** directory of the IntelliJ IDEA project and add the following codes to the client codes (construction methods for **Producer.java** or **Consumer.java**):

```
//truststore file address  
props.put("ssl.truststore.location", System.getProperty("user.dir") + File.separator + "conf" + File.separator + "truststore.jks");  
//truststore file password (password when the TrustStore file is generated)  
props.put("ssl.truststore.password", "XXXXXX");
```
 - d. Change the values of **security.protocol** in **producer.properties** and **consumer.properties** in the **src/main/resources** directory of the client sample project as required, and change the value of **bootstrap.servers** in the **producer.properties** file to ensure that the type of **security.protocol** matches with the port ID of **bootstrap.servers**.

20.5.2 Kafka Access Configuration on Windows Using EIPs

Scenario

This section describes how to bind Elastic IP addresses (EIPs) to a cluster and configure Kafka files so that sample files can be compiled locally.

Procedure

Step 1 Apply for an EIP for each node in the cluster and add public IP addresses and corresponding host domain names of all nodes to the Windows local **hosts** file. (If a host name contains uppercase letters, change them to lowercase letters.)

1. On the VPC console, apply for EIPs (the number of EIPs you buy should be equal to the number of nodes in the cluster), click the name of each node in the MRS cluster, and bind an EIP to each node on the **EIPs** page.

For details, see [Virtual Private Cloud > User Guide > EIP > Assigning an EIP and Binding It to an ECS](#).

2. Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.

```

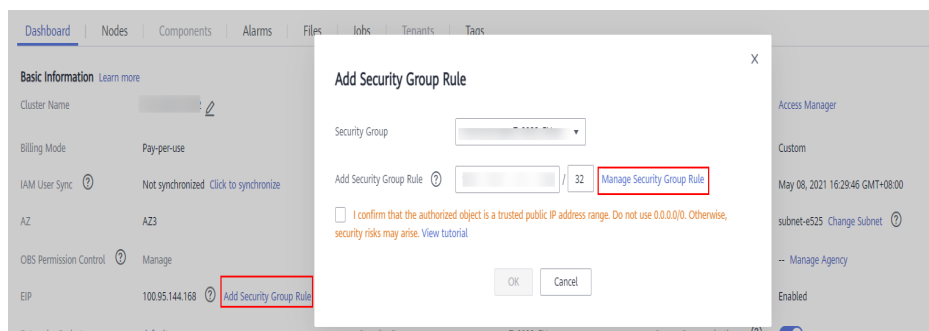
1 Mapping between public IP addresses and private IP addresses
2 100.95.144.168 172.16.0.120
3 100.95.144.168 172.16.0.42
4 100.95.144.168 172.16.0.62
5 100.95.144.168 172.16.0.200
6 100.93.144.168 172.16.0.139
7 100.93.144.168 172.16.0.214
8
9 hosts file in the cluster
10 172.16.0.120 node-group-1XZI00002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI00002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
11 172.16.0.42 node-master3vint.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3vint.ead64699-185a-4290-bbef-1a07e2f0459b.com.
12 172.16.0.62 node-group-1XZI00003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI00003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
13 172.16.0.200 node-master1ceip.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1ceip.ead64699-185a-4290-bbef-1a07e2f0459b.com.
14 172.16.0.139 node-group-1XZI00001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI00001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
15 172.16.0.214 node-master2pvnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pvnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.
16
17 hosts file that should be added to Windows
18 100.95.144.168 node-group-1xzi00002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1xzi00002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
19 100.95.144.168 node-master3vint.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3vint.ead64699-185a-4290-bbef-1a07e2f0459b.com.
20 100.93.144.168 node-group-1xzi00003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1xzi00003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
21 100.95.144.168 node-master1ceip.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1ceip.ead64699-185a-4290-bbef-1a07e2f0459b.com.
22 100.93.144.168 node-group-1xzi00001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1xzi00001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
23 100.93.144.168 node-master2pvnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pvnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.

```

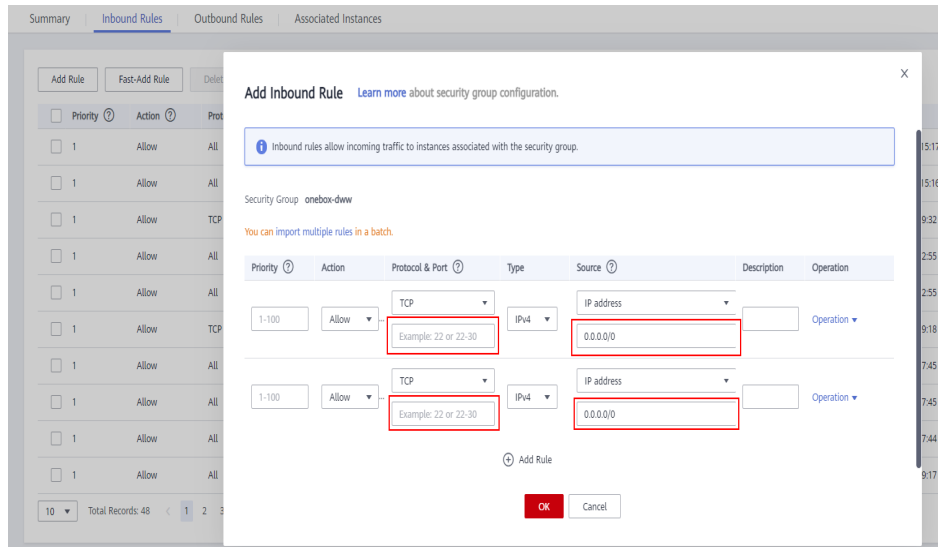
Step 2 Change the IP addresses in the **krb5.conf** file to the corresponding host names.

Step 3 Configure security group rules for the cluster.

1. On the **Dashboard** page, choose **Add Security Group Rule > Manage Security Group Rule**.



2. On the **Inbound Rules** tab page, click **Add Rule**. In the **Add Inbound Rule** dialog box, configure the Windows IP address and port 21007.



Step 4 On Manager, choose **Cluster > Services > Kafka > Configurations > All Configurations**, search for and add the key-value pair **advertised.listeners = SASL_PLAINTEXT://:21007,SASL_SSL://:21009,TRACE://:21013** in the **kafka.config.expandor** parameter, save the configuration, and restart the Kafka cluster.

Step 5 Before running the sample code, change the Kafka connection string in the sample code to **hostname1:21007, hostname2:21007, hostname3:21007**.

----End

20.5.3 FAQ

20.5.3.1 Topic Authentication Fails During Sample Running and "example-metric1=TOPIC_AUTHORIZATION_FAILED" Is Displayed

Troubleshooting Procedure

Step 1 Apply to the administrator for the access permission for the related topic.

Step 2 If the topic cannot be logged in after the access permission is assigned, log in to FusionInsight Manager, go to the Kafka service configuration page, search for **allow.everyone.if.no.acl.found**, and change the value to **true**. Then, run the sample again.

----End

20.5.3.2 Running the Producer.java Sample to Obtain Metadata Fails and "ERROR fetching topic metadata for topics..." Is Displayed, Even the Access Permission for the Related Topic

Troubleshooting Procedure

Step 1 Find **bootstrap.servers** in **producer.properties** under the **conf** directory of the project, and check whether the IP address and port ID are configured correctly.

- If the IP address is inconsistent with the service IP address of the Kafka cluster, change the IP address to the correct one.
- If the port ID is 21007 (security mode port), change it to 9092 (normal mode port).

Step 2 Check whether network connections are correct to ensure that the current device can access the Kafka cluster normally.

----**End**

21 Kafka Development Guide (Normal Mode)

21.1 Overview

21.1.1 Development Environment Preparation

Kafka Introduction

Kafka is a distributed message release and subscription system. With features similar to JMS, Kafka processes active streaming data.

Kafka is applicable to message queuing, behavior tracing, operation & maintenance (O&M) data monitoring, log collection, streaming processing, event tracing, and log persistence.

Kafka features:

- High throughput
- Message persistence to disks
- Scalable distributed system
- Fault-tolerant
- Support for online and offline scenarios

Interface Type Introduction

APIs provided by Kafka can be divided into two types: Producer API and Consumer API. Both the types of APIs contain Java API. For details, see section [Java API](#).

21.1.2 Common Concepts

- **Topic**
A same type of messages maintained by Kafka.
- **Partition**

One topic can be divided into multiple partitions, and each partition corresponds to an appendant and log file whose sequence is fixed.

- **Producer**
Role in a Kafka topic to which messages are sent.
- **Consumer**
Role that obtains messages from Kafka topics.
- **Broker**
A node server in a Kafka cluster.

21.1.3 Development Process

Kafka client roles include Producer and Consumer, which share the same application development process.

[Figure 21-1](#) and [Table 21-1](#) show each phase of the development process.

Figure 21-1 Kafka client application development process

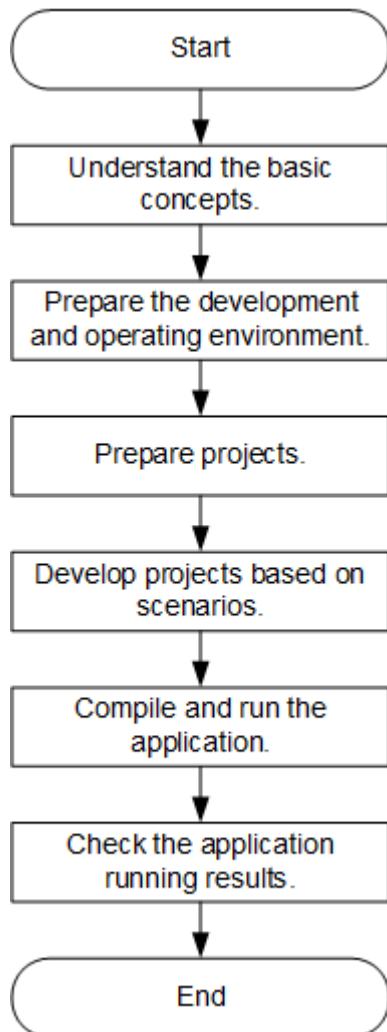


Table 21-1 Kafka client application development process description

Phase	Description	Reference Document
Understand the basic concepts.	Before developing an application, learn basic concepts of Kafka, and determine whether the desired role is Producer or Consumer based on the actual scenario.	Common Concepts
Prepare the development and operating environment.	The Java language is recommended for the development of Java applications. The IntelliJ IDEA tool can be used. The Kafka running environment is the Kafka client. Install and configure the client according to the guide.	Preparing for Development and Operating Environment
Prepare projects	Kafka provides program samples for different scenarios. You can import the program samples for learning.	Configuring and Importing a Sample Project
Develop projects based on scenarios.	Producer and Consumer API usage samples are provided and cover API, and multi-thread usage scenarios, helping you quickly know Kafka interfaces well.	Developing an Application
Compile and run the application.	Compile the developed application and submit it for running.	Application Commissioning
View application running results.	Program running results will be written to and printed on the console. You can use a Linux client to consume topic data to check whether data has been successfully written.	Application Commissioning

21.2 Environment Preparation

21.2.1 Preparing for Development and Operating Environment

Preparing Development Environment

Table 21-2 describes the environment required for secondary development.

Table 21-2 Development environment

Item	Description
OS	<ul style="list-style-type: none"> Development environment: Windows OS. Windows 7 or later is supported. Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.
IntelliJ IDEA installation and configuration	<p>It is the basic configuration for the development environment. JDK 1.8 is used. IntelliJ IDEA 2019.1 or other compatible versions are used.</p> <p>NOTE</p> <ul style="list-style-type: none"> If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK. If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK. If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> Oracle JDK versions: 1.8 IBM JDK versions: 1.8.5.11 For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> OpenJDK: 1.8.0_272 <p>NOTE</p> <p>The server supports only TLS V1.2 or later to ensure security.</p> <p>By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls.</p>

Item	Description
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
7-zip	It is a tool used to decompress .zip and .rar packages. The 7-Zip 16.04 is supported.

Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.

- a. **Log in to the FusionInsight Manager portal** and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is

FusionInsight_Cluster_1_Services_Client.tar, decompress it to obtain **FusionInsight_Cluster_1_Services_ClientConfig.tar** file. Then, decompress **FusionInsight_Cluster_1_Services_ClientConfig.tar** file to the **D:\FusionInsight_Cluster_1_Services_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight_Cluster_1_Services_ClientConfig\Kafka\config**. Obtain the Kafka-related configuration file.

Table 21-3 describes the main configuration files.

Table 21-3 Configuration file

Document Name	Function
server.properties	Configures the Kafka server.
client.properties	Configures the Kafka client.
producer.properties	Configures producer of kafka parameters.
consumer.properties	Configures consumer of kafka parameters.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where

the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client on the node. For example, the client installation directory is **/opt/client**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. [Log in to the FusionInsight Manager portal](#). Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all JAR package in the **FusionInsight_Cluster_1_Services_ClientConfig/Kafka/install_files/kafka/libs** directory to the **lib** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/client/lib**.

For example, if the client software package is **FusionInsight_Cluster_1_Services_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Kafka/install_files/kafka/libs/* root@IP address of the client node:/opt/client/lib
```

- c. Check the network connection of the client node.

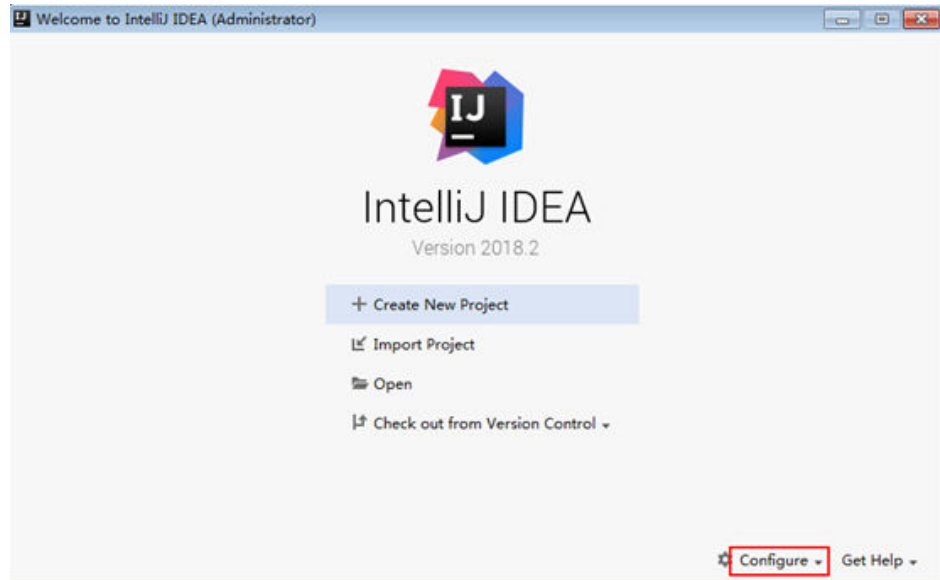
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

21.2.2 Configuring and Importing a Sample Project

- Step 1** Obtain the sample project folder **kafka-examples** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

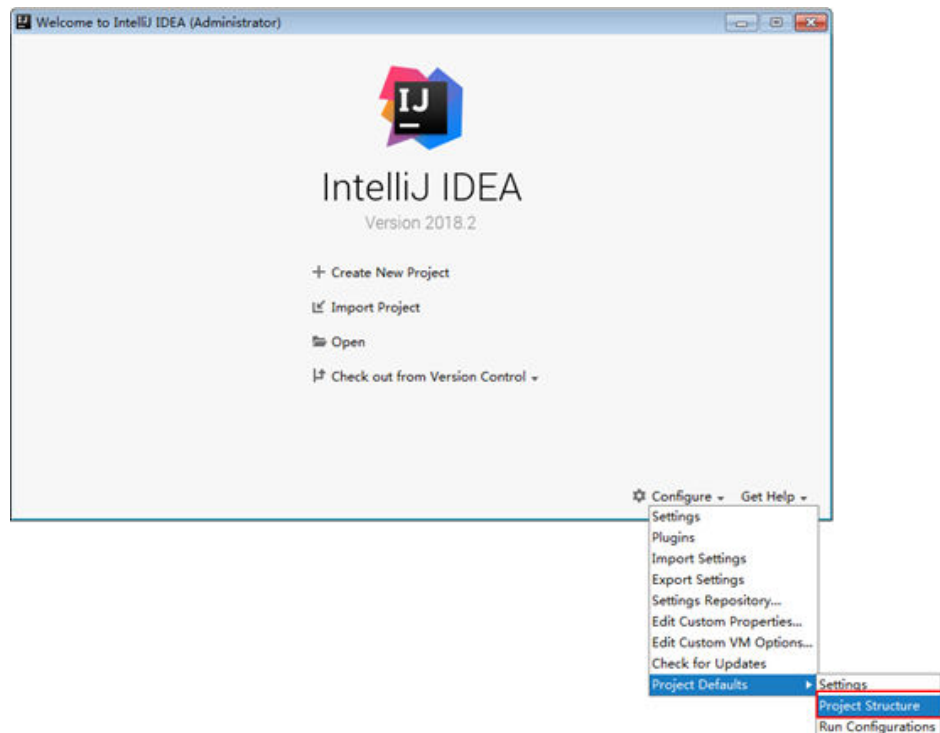
- Step 2** Copy the all the cluster configuration file obtained in section [Preparing an Operating Environment](#) to the `kafka-examples\src\main\resources` directory of the sample project.
- Step 3** After installing the IntelliJ IDEA and JDK tools, configure JDK in IntelliJ IDEA.
 1. Open IntelliJ IDEA and click **Configure**.

Figure 21-2 Quick Start



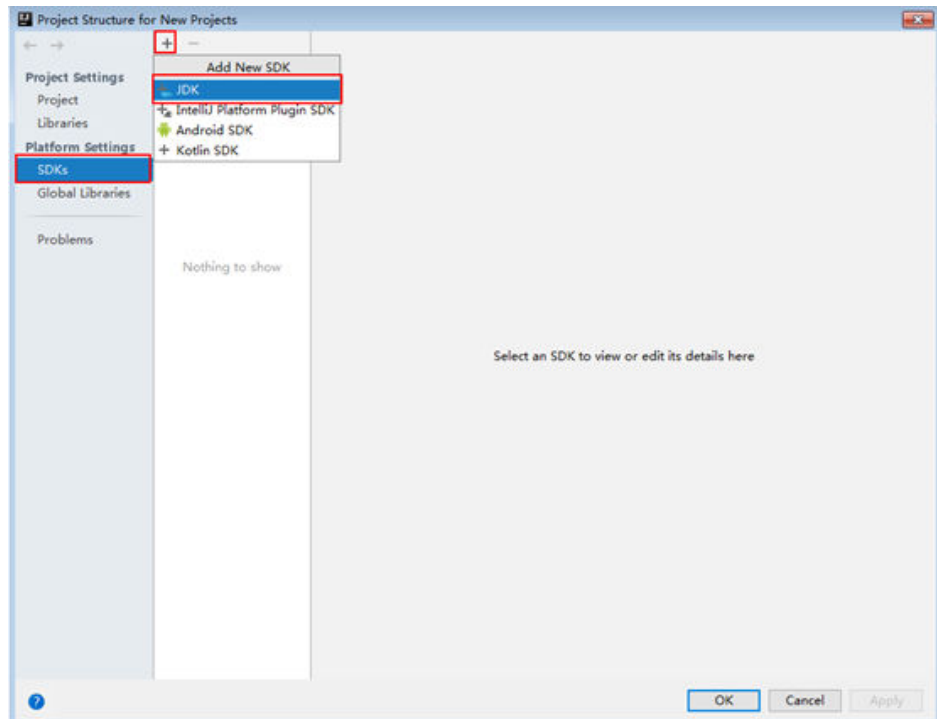
2. Choose **Project Defaults > Project Structure**.

Figure 21-3 Configure



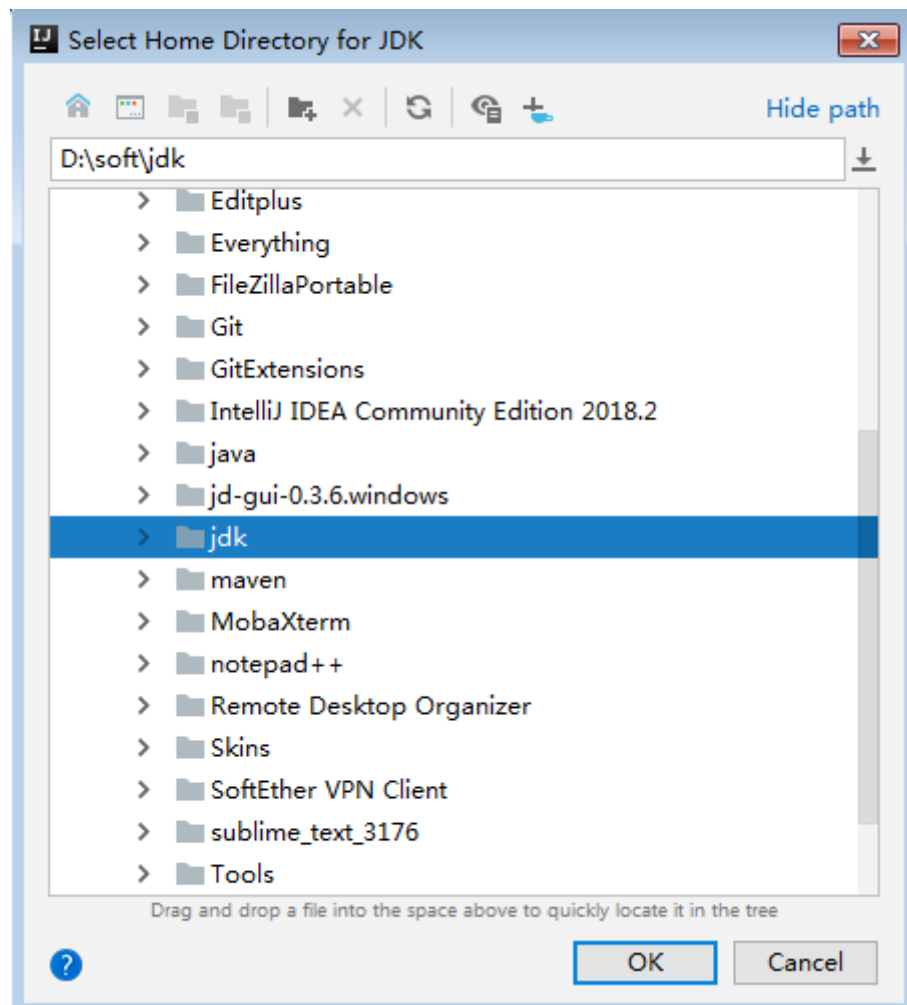
3. In the displayed **Project Structure for New Projects** interface, click **SDKs**. Then click the plus sign (+) and choose **JDK**.

Figure 21-4 Project Structure for New Projects



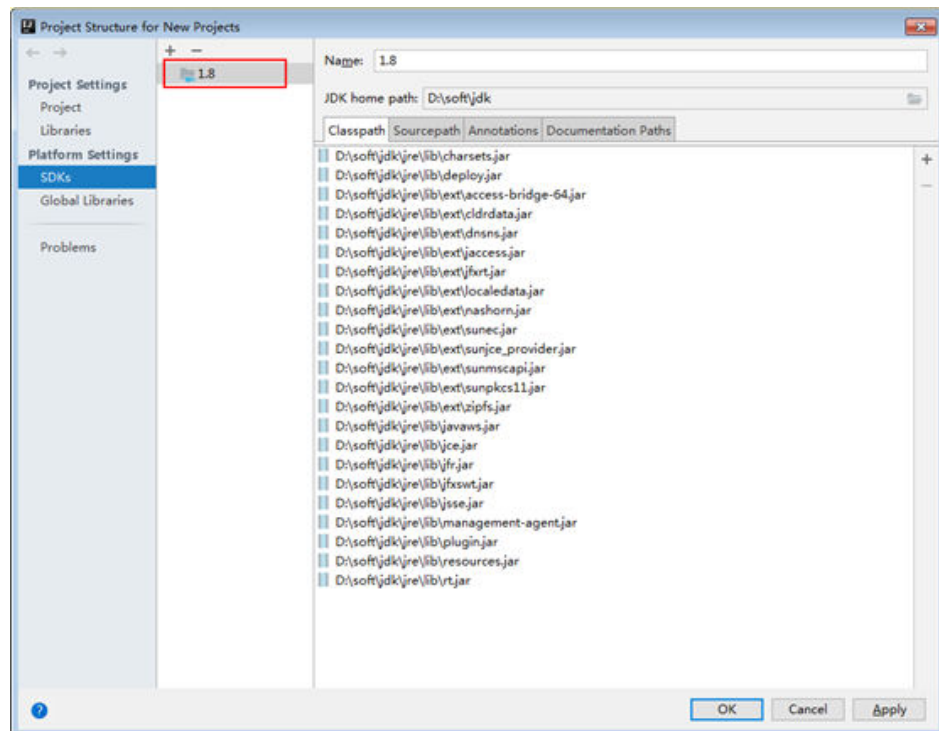
4. In the displayed **Select Home Directory for JDK** interface, select the JDK directory, and click **OK**.

Figure 21-5 Select Home Directory for JDK



5. Click **OK**.

Figure 21-6 Completing the JDK configuration



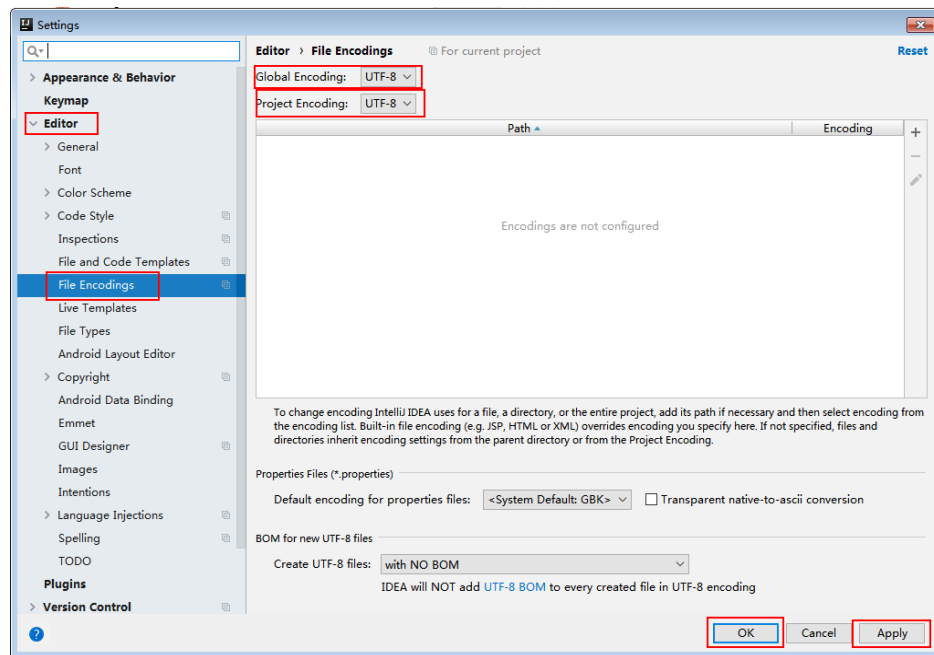
Step 4 Import example projects into the IntelliJ IDEA development environment.

1. Choose **Open**.
The dialog box for browsing directories is displayed.
2. Select the sample project folder and click **OK**.

Step 5 Set the IntelliJ IDEA text file coding format to prevent invalid characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.
The **Settings** window is displayed.
2. Choose **Editor > File Encodings** in the **Project Encoding** area and **Global Encoding** area, set the parameter to **UTF-8**, click **Apply**, and then click **OK**, as shown in [Figure 21-7](#).

Figure 21-7 Setting the IntelliJ IDEA coding format



----End

21.3 Developing an Application

21.3.1 Typical Scenario Description

Scenario

Kafka is a distributed message system, in which messages can be publicized or subscribed. A Producer is to be developed to send a message to a topic of a Kafka cluster every second, and a Consumer is to be implemented to ensure that the topic is subscribed and that messages of the topic are consumed in real time.

Development Idea

1. Use a Linux client to create a topic.
2. Develop a Producer to produce data to the topic.
3. Develop a Consumer to consume the data of the topic.

Suggestions on Performance Tuning

1. Create a topic and plan its partitions based on service requirements. The number of partitions limits the number of concurrent consumers.
2. The key value of a message must be variable to ensure even message distribution.
3. Consumers are advised to proactively submit the offset to avoid repeated consumption.

21.3.2 Example Code Description

21.3.2.1 Producer API Usage Sample

Function Description

The following code snippet belongs to the **run** method of the **com.huawei.bigdata.kafka.example.Producer** class. It is used by the Producer APIs to produce messages for the security topic.

Code Sample

```
/**
 * The Producer thread executes a function to send messages periodically.
 */
public void run() {
    LOG.info("New Producer: start.");
    int messageNo = 1;

    while (messageNo <= MESSAGE_NUM) {
        String messageStr = "Message_" + messageNo;
        long startTime = System.currentTimeMillis();

        // Construct message records.
        ProducerRecord<Integer, String> record = new ProducerRecord<Integer, String>(topic, messageNo,
messageStr);

        if (isAsync) {
            // Sending in asynchronous mode
            producer.send(record, new DemoCallBack(startTime, messageNo, messageStr));
        } else {
            try {
                // Sending in synchronous mode
                producer.send(record).get();
                long elapsedTime = System.currentTimeMillis() - startTime;
                LOG.info("message(" + messageNo + ", " + messageStr + ") sent to topic(" + topic + ") in " +
elapsedTime + " ms.");
            } catch (InterruptedException ie) {
                LOG.info("The InterruptedException occurred : {}", ie);
            } catch (ExecutionException ee) {
                LOG.info("The ExecutionException occurred : {}", ee);
            }
        }
        messageNo++;
    }
}
```

21.3.2.2 Consumer API Usage Sample

Function Description

The following code sample belongs to the **com.huawei.bigdata.kafka.example.Consumer** class. It is used to enable the Consumer API to subscribe a secure topic and consume messages.

Code Sample

```
/**
 * Consumer constructor.
 * @param topic Name of the subscribed topic
 */
```

```
public Consumer(String topic) {
    super("KafkaConsumerExample", false);
    // Initializes the configuration parameters required for starting the consumer. For details, see the code.
    Properties props = initProperties();
    consumer = new KafkaConsumer<Integer, String>(props);
    this.topic = topic;
}

public void doWork() {
    // Subscribe
    consumer.subscribe(Collections.singletonList(this.topic));
    // Message consumption request
    ConsumerRecords<Integer, String> records = consumer.poll(waitTime);
    // Message Processing
    for (ConsumerRecord<Integer, String> record : records) {
        LOG.info("[ConsumerExample], Received message: (" + record.key() + ", " + record.value() + ") at
offset " + record.offset());
    }
}
```

21.3.2.3 Multi-thread Producer Sample

Function Description

The multi-thread producer function is implemented based on the code sample described in section [Producer API Usage Sample](#). Multiple producer threads can be started. Each thread sends messages to the partition whose key is the same as the thread ID.

The following code snippet belongs to the run method in the **com.huawei.bigdata.kafka.example.ProducerMultThread** class, and these code snippets are used to enable multiple threads to produce data.

Code Sample

```
/**
 * Specify the current ThreadID as the key value and send data.
 */
public void run()
{
    LOG.info("Producer: start.");

    // Record the number of messages.
    int messageCount = 1;

    // Specify the number of messages sent by each thread.
    int messagesPerThread = 5;
    while (messageCount <= messagesPerThread)
    {

        // Specify the content of messages to be sent.
        String messageStr = new String("Message_" + sendThreadId + "_" + messageCount);

        // Specify a key value for each thread to enable the thread to send messages to only a specified
        partition.
        String key = String.valueOf(sendThreadId);

        // Send messages.
        producer.send(new KeyedMessage<String, String>(sendTopic, key, messageStr));
        LOG.info("Producer: send " + messageStr + " to " + sendTopic + " with key: " + key);
        messageCount++;
    }
}
```

21.3.2.4 Multi-thread Consumer Sample

Function Description

The multi-thread consumer function is implemented based on the code sample described in section [Consumer API Usage Sample](#). The number of consumer threads that can be started to consume the messages in partitions is the same as the number of partitions in the topic.

The following code snippet belongs to the run method in the **com.huawei.bigdata.kafka.example.ConsumerMultThread** class, and these code snippets are used to implement concurrent consumption of messages in a specified topic.

Code Sample

```
/**
 * Start the concurrent multi-thread consumer.
 */
public void run() {
    LOG.info("Consumer: start.");
    Properties props = Consumer.initProperties();
    // Start a specified number of consumer threads to consume.
    // Note: When this parameter is larger than the number of partitions of the topic to be consumed, the
    extra threads fails to consume data.
    for (int threadNum = 0; threadNum < CONCURRENCY_THREAD_NUM; threadNum++) {
        new ConsumerThread(threadNum, topic, props).start();
        LOG.info("Consumer Thread " + threadNum + " Start.");
    }
}

private class ConsumerThread extends ShutdownableThread {
    private int threadNum = 0;
    private String topic;
    private Properties props;
    private KafkaConsumer<String, String> consumer = null;

    /**
     * Construction method of the consumer thread class
     *
     * @param threadNum Thread number
     * @param topic topic
     */
    public ConsumerThread(int threadNum, String topic, Properties props) {
        super("ConsumerThread" + threadNum, true);
        this.threadNum = threadNum;
        this.topic = topic;
        this.props = props;
        this.consumer = new KafkaConsumer<String, String>(props);
    }

    public void doWork() {
        consumer.subscribe(Collections.singleton(this.topic));
        ConsumerRecords<String, String> records = consumer.poll(waitTime);
        for (ConsumerRecord<String, String> record : records) {
            LOG.info("Consumer Thread-" + this.threadNum + " partitions:" + record.partition() + " record: "
                + record.value() + " offsets: " + record.offset());
        }
    }
}
```


21.3.3 Kafka Streams Sample Code Description

Scenario Description

Kafka Streams is a lightweight stream processing framework provided by Apache Kafka. The input and output of Kafka Streams are stored in the Kafka cluster.

The following describes the most common WordCount samples.

Development Guideline

1. Create two topics on the Linux client to serve as the input and output topics.
2. Develop a Kafka Streams to implement the word count function. The system collects statistics on the number of words in each message by reading the message in the input topic, consumes data from the output topic, and outputs the statistical result in the form of a key-value pair.

21.3.4 Kafka Streams Sample Code Description

21.3.4.1 High level KafkaStreams API Usage Sample

Function Description

The following code snippets are used in the `createWordCountStream` method of the `com.huawei.bigdata.kafka.example.WordCountDemo` class to implement the following function:

Collects statistics on input records. Same words are divided into a group, which is used as a key value. The occurrence times of each word are calculated as a value and are output in the form of a key-value pair.

Code Sample

```
static void createWordCountStream(final StreamsBuilder builder) {
    // Receive the input records from input-topic.
    final KStream<String, String> source = builder.stream(INPUT_TOPIC_NAME);

    // Aggregate the calculation result of the key-value pair.
    final KTable<String, Long> counts = source
        // Process the received records and split according to the regular expression REGEX_STRING.
        .flatMapValues(value ->
Arrays.asList(value.toLowerCase(Locale.getDefault()).split(REGEX_STRING)))
        // Aggregate the calculation result of the key-value pair.
        .groupBy((key, value) -> value)
        // The final calculation result
        .count();

    // Output the key-value pair of the calculation result from the output topic.
    counts.toStream().to(OUTPUT_TOPIC_NAME, Produced.with(Serdes.String(), Serdes.Long()));
}
```

21.3.4.2 Low level KafkaStreams API Usage Sample

Function Description

The following code snippets are used in the **com.huawei.bigdata.kafka.example.WordCountProcessorDemo** class to implement the following function:

Collects statistics on input records. Same words are divided into a group, which is used as a key value. The occurrence times of each word are calculated as a value and are output in the form of a key-value pair.

Code Sample

```
private static class MyProcessorSupplier implements ProcessorSupplier<String, String> {
    @Override
    public Processor<String, String> get() {
        return new Processor<String, String>() {
            // ProcessorContext instance, which provides the access of the metadata of the records being
            // processed
            private ProcessorContext context;
            private KeyValueStore<String, Integer> kvStore;

            @Override
            @SuppressWarnings("unchecked")
            public void init(ProcessorContext context) {
                // Save processor context in the local host, because it will be used for punctuate() and commit().
                this.context = context;
                // Execute punctuate() once every second.
                this.context.schedule(Duration.ofSeconds(1), PunctuationType.STREAM_TIME, timestamp -> {
                    try (final KeyValueIterator<String, Integer> iter = kvStore.all()) {
                        System.out.println("----- " + timestamp + " -----");
                        while (iter.hasNext()) {
                            final KeyValue<String, Integer> entry = iter.next();
                            System.out.println "[" + entry.key + ", " + entry.value + "]";
                            // Send the new records to the downstream processor as key-value pairs.
                            context.forward(entry.key, entry.value.toString());
                        }
                    }
                });
                // Search for the key-value states storage area named KEY_VALUE_STATE_STORE_NAME to
                // memorize the recently received input records.
                this.kvStore = (KeyValueStore<String, Integer>)
                context.getStateStore(KEY_VALUE_STATE_STORE_NAME);
            }

            // Process the receiving records of input topic. Split the records into words, and count the words.
            @Override
            public void process(String dummy, String line) {
                String[] words = line.toLowerCase(Locale.getDefault()).split(REGEX_STRING);

                for (String word : words) {
                    Integer oldValue = this.kvStore.get(word);

                    if (oldValue == null) {
                        this.kvStore.put(word, 1);
                    } else {
                        this.kvStore.put(word, oldValue + 1);
                    }
                }
            }

            @Override
            public void close() {
            }
        };
    }
};
```

```
}  
}
```

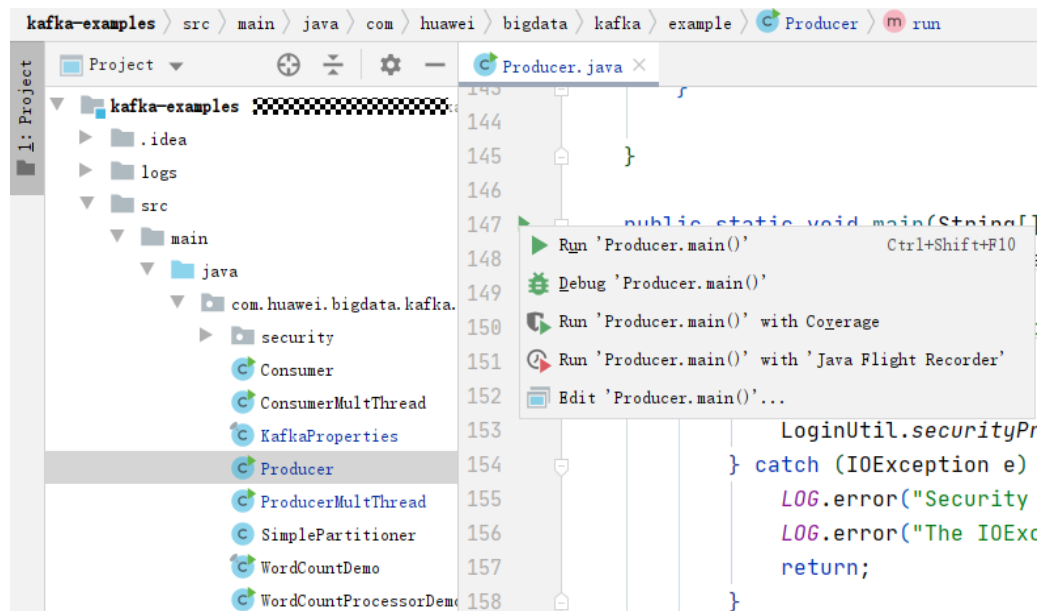
21.4 Application Commissioning

21.4.1 Commissioning an Application in Windows

Starting Producer

- Step 1** Ensure that the mappings between the host names and service IP addresses of all the hosts in the remote cluster are configured in the local **hosts** file.
- Step 2** Run **Producer.java** on IntelliJ IDEA, as shown in **Figure 21-8**.

Figure 21-8 Run **Producer.java** on IntelliJ IDEA



- Step 3** The console window appears. You can find that Producer is sending messages to the default topic (example-metric1). One piece of log is printed when every 10 messages are sent.

Figure 21-9 Proceducer running window

```
[2019-06-12 10:31:37,865] INFO Updated cluster metadata version 2 to Cluster(id = 1cPugJnSQAernMH8RS_-jA, nodes = [187.  
[2019-06-12 10:31:51,729] INFO Updated cluster metadata version 3 to Cluster(id = 1cPugJnSQAernMH8RS_-jA, nodes = [187.  
[2019-06-12 10:31:53,140] INFO The Producer have send 10 messages. (com.huawei.bigdata.kafka.example.NewProducer)  
[2019-06-12 10:31:54,516] INFO The Producer have send 20 messages. (com.huawei.bigdata.kafka.example.NewProducer)  
[2019-06-12 10:31:55,906] INFO The Producer have send 30 messages. (com.huawei.bigdata.kafka.example.NewProducer)  
[2019-06-12 10:31:57,299] INFO The Producer have send 40 messages. (com.huawei.bigdata.kafka.example.NewProducer)  
[2019-06-12 10:31:58,686] INFO The Producer have send 50 messages. (com.huawei.bigdata.kafka.example.NewProducer)  
[2019-06-12 10:32:00,070] INFO The Producer have send 60 messages. (com.huawei.bigdata.kafka.example.NewProducer)
```

----End

Starting Consumer

Step 1 Run the `Consumer.java` file.

Step 2 Click **Run**. In the console window that appears, you can find that Producer starts after Consumer successfully starts, and then you can view messages received in real time.

Figure 21-10 Consumer.java running window

```
[2019-06-23 17:49:48,609] INFO [Consumer clientId=consumer-1, groupId=DemoConsumer] (Re-)joining group (org.apache.kafka.clients.consumer.internals.AbstractCoordinator)
[2019-06-23 17:49:51,865] INFO [Consumer clientId=consumer-1, groupId=DemoConsumer] Successfully joined group with generation 5 (org.apache.kafka.clients.consumer.internals.Coordinator)
[2019-06-23 17:49:51,866] INFO [Consumer clientId=consumer-1, groupId=DemoConsumer] Setting newly assigned partitions [example-metric1-0, example-metric1-1] (org.apache.kafka.clients.consumer.internals.Coordinator)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (1, Message_1) at offset 188 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (5, Message_5) at offset 189 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (8, Message_8) at offset 190 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (9, Message_9) at offset 191 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (15, Message_15) at offset 192 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (18, Message_18) at offset 193 (com.huawei.bigdata.kafka.example.NewConsumer)
```

----End

Starting Other Code Samples

The procedures for starting other code samples are similar to the procedures for starting Producer and Consumer in this section.

21.4.2 Commissioning an Application in Linux

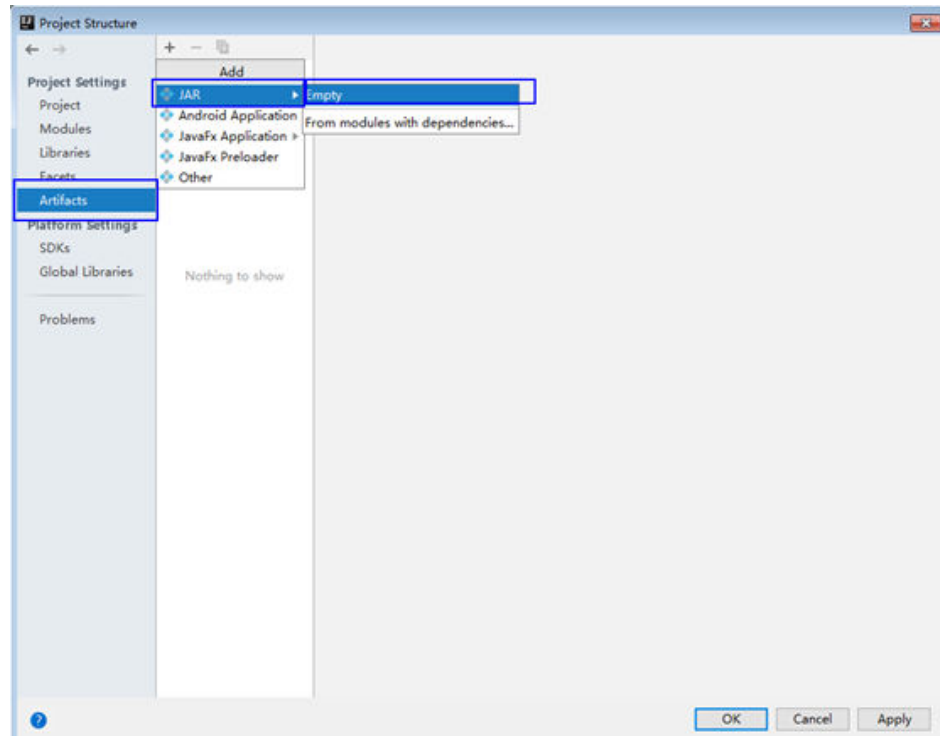
Scenario

Run a sample program in Linux after code development is complete.

Procedure

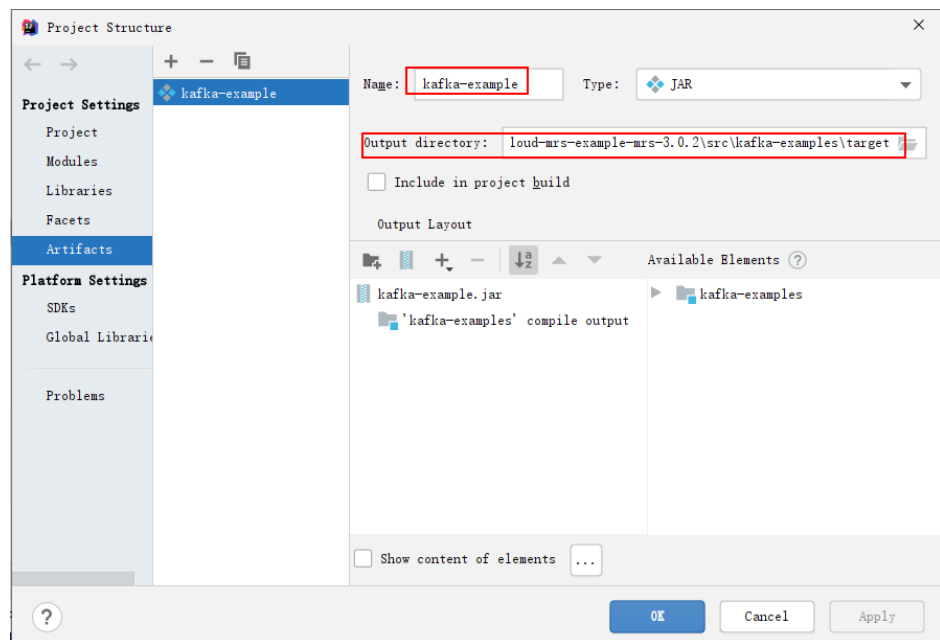
- Step 1** In IntelliJ IDEA, configure the Artifacts information of the project before generating the JAR package.
1. Enter IntelliJ IDEA and choose **File > Project Structure**.
 2. On the **Project Structure** page, select **Artifacts**, click the plus sign (+), and choose **JAR > Empty**.

Figure 21-11 Add Artifacts



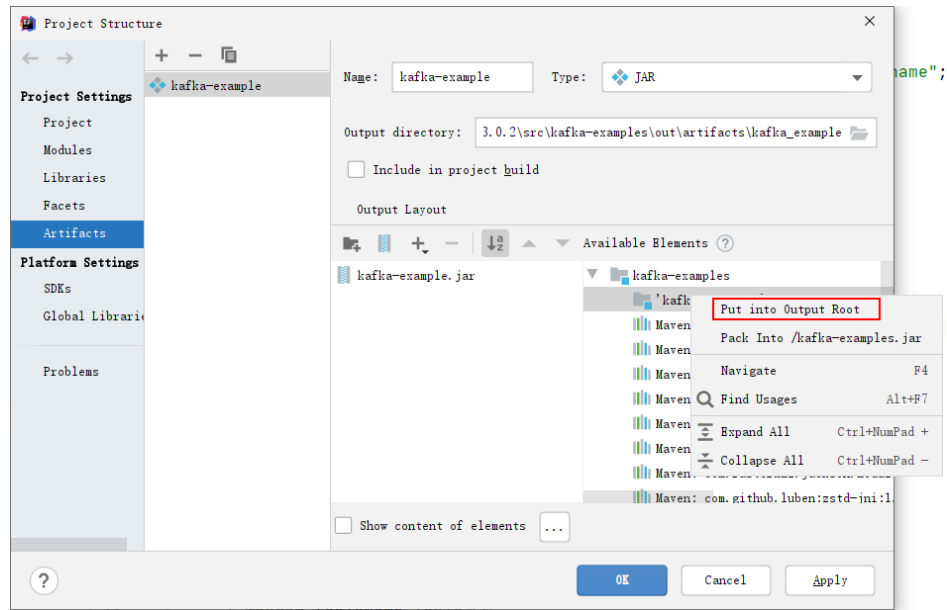
3. Set the name, type, and output path of the JAR package based on the site requirements.

Figure 21-12 Setting basic information



4. Choose 'kafka-examples' compile output, right-click **Put into Output Root**, and click **Apply**.

Figure 21-13 Put into Output Root

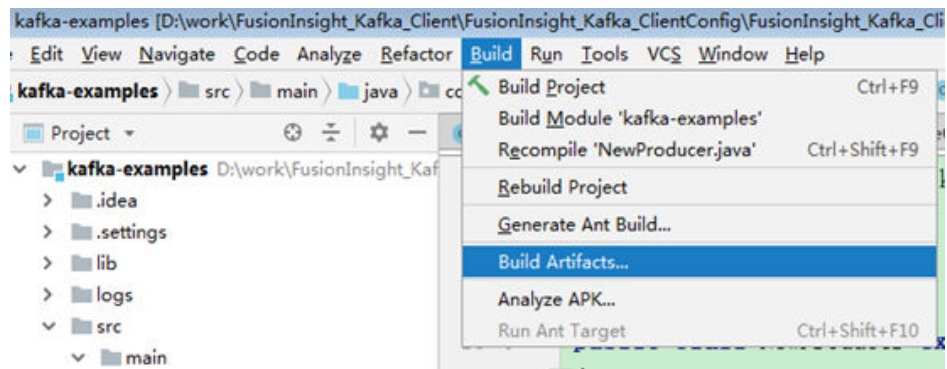


5. Click **OK** to complete the configuration.

Step 2 Generate the JAR package.

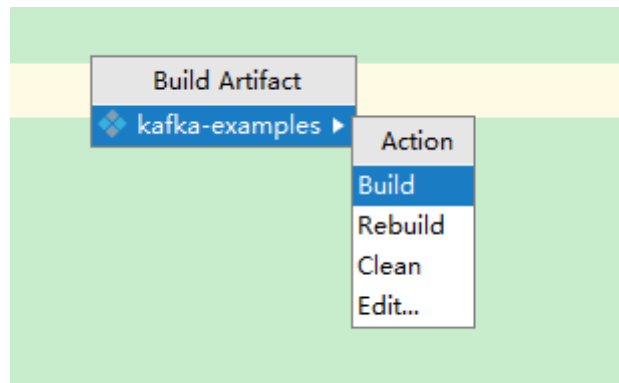
1. Enter IntelliJ IDEA and choose **Build > Build Artifacts**.

Figure 21-14 Build Artifacts



2. In the displayed interface, choose **kafka-example:jar > Build** to generate the JAR package.

Figure 21-15 Build

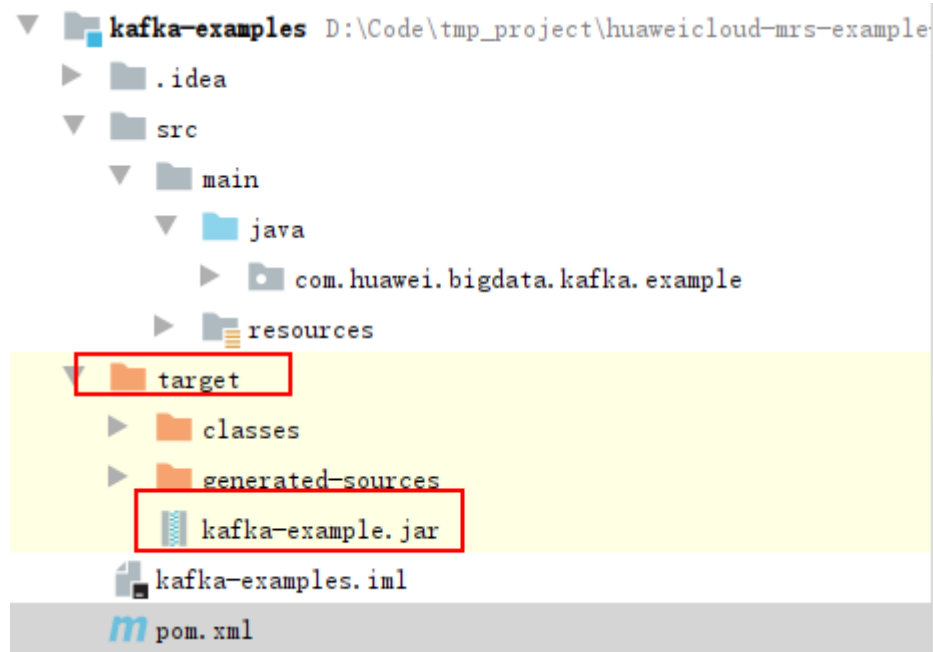


3. If log similar to the following is displayed in event logs, the JAR package is successfully generated. Obtain the JAR package from the **Output directory** configured shown in [Figure 21-12](#)

```
14:37 Compilation completed successfully in 25 s 991 ms
```

Step 3 Copy the generated jar package to **/opt/client/lib**.

Figure 21-16 File Location



Step 4 Copy all files in the **src/main/resources** directory of the IntelliJ IDEA project to the **src/main/resources** directory at the same level as the **lib** folder, that is, **/opt/client/src/main/resources**.

Step 5 Ensure that the current user has read permission of all the files in the **src/main/resources** and **lib** folders in **/opt/client**, jdk has been installed, and java environment variables are set. Then, run the command, for example, **java -cp /opt/client/lib/*:/opt/client/src/main/resources com.huawei.bigdata.kafka.example.Producer** to run the example project.

----End

21.4.3 Kafka Streams Sample Running Guide

21.4.3.1 High Level Kafka Streams API Sample Usage Guide

1. Use the Linux client to create the input and output topics. Ensure that the topic names are the same as those in the sample code, set the policy for clearing the output topic to compact, and run the sample code.

```
// source-topic name created by the user, that is, input topic  
private static final String INPUT_TOPIC_NAME = "streams-wordcount-input";  
// sink-topic name created by the user, that is, output topic  
private static final String OUTPUT_TOPIC_NAME = "streams-wordcount-output";
```

- Create the input topic:

- kafka-topics.sh --create --zookeeper 192.168.0.11:2181,192.168.0.12:2181,192.168.0.13:2181/kafka --replication-factor 1 --partitions 1 --topic streams-wordcount-input**

 - Create the output topic:

kafka-topics.sh --create --zookeeper 192.168.0.11:2181,192.168.0.12:2181,192.168.0.13:2181/kafka --replication-factor 1 --partitions 1 --topic streams-wordcount-output --config cleanup.policy=compact

 - Run sample code **WordCountDemo.java**. For details, see [Commissioning an Application in Windows](#) and [Commissioning an Application in Linux](#).
2. Use the Linux client to write messages and view the statistics result.

Run the **kafka-console-producer.sh** command to write messages to the input topic.

```
# kafka-console-producer.sh --broker-list 192.168.0.11:21005,192.168.0.12:21005,192.168.0.13:21005 --topic streams-wordcount-input --producer.config /opt/client/Kafka/kafka/config/producer.properties
>This is Kafka Streams test
>test starting
>now Kafka Streams is running
>test end
>
```

Run the **kafka-console-consumer.sh** command to consume data from the output topic and view the statistics result.

```
# kafka-console-consumer.sh --topic streams-wordcount-output --bootstrap-server 192.168.0.11:21005,192.168.0.12:21005,192.168.0.13:21005 --consumer.config /opt/client/Kafka/kafka/config/consumer.properties --from-beginning --property print.key=true --property print.value=true --property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer --property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer --formatter kafka.tools.DefaultMessageFormatter
this 1
is 6
kafka 12
streams 8
test 8
test 9
starting 1
now 1
kafka 13
streams 9
is 7
running 1
test 10
end 1
```

21.4.3.2 Low level Kafka Streams API Sample Usage Guide

1. Use the Linux client to create the input and output topics. Ensure that the topic names are the same as those in the sample code, set the policy for clearing the output topic to compact, and run the sample code.

```
// source-topic name created by the user, that is, input topic
private static final String INPUT_TOPIC_NAME = "streams-wordcount-processor-input";

// sink-topic name created by the user, that is, output topic
private static final String OUTPUT_TOPIC_NAME = "streams-wordcount-processor-output";
```

- Create the input topic:

kafka-topics.sh --create --zookeeper 192.168.0.11:2181,192.168.0.12:2181,192.168.0.13:2181/kafka --replication-factor 1 --partitions 1 --topic streams-wordcount-processor-input

- Create the output topic:
kafka-topics.sh --create --zookeeper 192.168.0.11:2181,192.168.0.12:2181,192.168.0.13:2181/kafka --replication-factor 1 --partitions 1 --topic streams-wordcount-processor-output--config cleanup.policy=compact
 - Run sample code **WordCountProcessorDemo.java**. For details, see [Commissioning an Application in Windows](#) and [Commissioning an Application in Linux](#).
2. Use the Linux client to write messages and view the statistics result.

Run the **kafka-console-producer.sh** command to write messages to the input topic.

```
# kafka-console-producer.sh --broker-list 192.168.0.11:9092,192.168.0.12:9092,192.168.0.13:9092 --topic streams-wordcount-processor-input --producer.config /opt/client/Kafka/kafka/config/producer.properties
>This is Kafka Streams test
>now Kafka Streams is running
>test end
>
```

Run the **kafka-console-consumer.sh** command to consume data from the output topic and view the statistics result.

```
# kafka-console-consumer.sh --topic streams-wordcount-processor-output --bootstrap-server 192.168.0.11:9092,192.168.0.12:9092,192.168.0.13:9092 --consumer.config /opt/client/Kafka/kafka/config/consumer.properties --from-beginning --property print.key=true --property print.value=true
is 1
kafka 1
streams 1
test 1
this 1
end 1
is 2
kafka 2
now 1
running 1
streams 2
test 2
this 1
```

21.5 More Information

21.5.1 External Interfaces

21.5.1.1 Shell

1. Query the list of topics in current clusters.
bin/kafka-topics.sh --list --zookeeper <ZooKeeper cluster IP address:2181/kafka>
bin/kafka-topics.sh --list --bootstrap-server <Kafka cluster IP address: 21007> --command-config config/client.properties
2. Query the details of a topic.
bin/kafka-topics.sh --describe --zookeeper <ZooKeeper cluster IP address: 2181/kafka> --topic <Topic name>

- ```
bin/kafka-topics.sh --describe --bootstrap-server <Kafka cluster IP address:21007> --command-config config/client.properties --topic <Topic name>
```
3. Delete a topic (this operation can be performed by an administrator).

```
bin/kafka-topics.sh --delete --zookeeper <ZooKeeperCluster IP address: 2181/kafka> --topic <Topic name>
```

```
bin/kafka-topics.sh --delete --bootstrap-server <Kafka cluster IP address: 21007> --command-config config/client.properties --topic <Topic name>
```
  4. Create a topic (this operation can be performed by an administrator).

```
bin/kafka-topics.sh --create --zookeeper <ZooKeeperCluster IP address: 2181/kafka> --partitions 6 --replication-factor 2 --topic <Topic name>
```

```
bin/kafka-topics.sh --create --bootstrap-server <Kafka cluster IP address: 21007> --command-config config/client.properties --partitions 6 --replication-factor 2 --topic <Topic name>
```
  5. Assign permissions to the Consumer (this operation is performed by an administrator).

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=<ZooKeeper cluster IP address:2181/kafka > --add --allow-principal User:<User name> --consumer --topic <Topic name> --group <Consumer group name>
```

```
bin/kafka-acls.sh --bootstrap-server <Kafka cluster IP address:21007> --command-config config/client.properties --add --allow-principal User:<User name> --consumer --topic <Topic name> --group <Consumer group name>
```
  6. Assign permissions to the Producer (this operation is performed by an administrator).

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=<ZooKeeper cluster IP address:2181/kafka > --add --allow-principal User:<User name> --producer --topic <Topic name>
```

```
bin/kafka-acls.sh --bootstrap-server <Kafka cluster IP address:21007> --command-config config/client.properties --add --allow-principal User:<User name> --producer --topic <Topic name>
```
  7. Produce messages (this operation requires the producer permission of the desired topic).

```
bin/kafka-console-producer.sh --broker-list <KafkaCluster IP address: 21007> --topic <Topic name> --producer.config config/producer.properties
```
  8. Consume data (the producer permission of the topic is required).

```
bin/kafka-console-consumer.sh --topic <Topic name> --bootstrap-server <KafkaCluster IP address:21007> --consumer.config config/consumer.properties
```

### 21.5.1.2 Java API

Versions of interfaces adopted by Kafka are consistent with those in Open Source Community. For details, see <https://kafka.apache.org/24/documentation.html>.

## Major Interfaces of a Producer

**Table 21-4** Major parameters of a Producer

| Parameter                  | Description                             | Remarks                                                                                                                                                                                                     |
|----------------------------|-----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| bootstrap.servers          | Broker address list                     | The Producer creates connections with the Broker based on this parameter.                                                                                                                                   |
| security.protocol          | Security protocol type                  | The Producer uses the security protocol of the type specified by this parameter. In security mode, only the SASL protocol is supported, so this parameter needs to be configured as <b>SASL_PLAINTEXT</b> . |
| sasl.kerberos.service.name | Service name                            | This parameter specifies the Kerberos user name used by the Kafka cluster for running. This parameter needs to be configured as <b>kafka</b> .                                                              |
| key.serializer             | Serialization of key values in messages | This parameter specifies how to serialize the key values in messages.                                                                                                                                       |
| value.serializer           | Serialization of messages               | This parameter specifies how to serialize transmitted messages.                                                                                                                                             |

**Table 21-5** Major interface functions of a Producer

| Return Value                                | Interface Function                | Description                                                                                                                         |
|---------------------------------------------|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| java.util.concurrent.Future<RecordMetadata> | send(ProducerRecord<K, V> record) | Indicates a TX interface without a callback function. Generally, the get() function of Future is used for synchronous transmission. |

| Return Value                                | Interface Function                                          | Description                                                                                                                                                              |
|---------------------------------------------|-------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| java.util.concurrent.Future<RecordMetadata> | send(ProducerRecord<K, V> record, Callback callback)        | Indicates a TX interface with a callback function. Generally, this interface uses the callback function to process transmission results after asynchronous transmission. |
| void                                        | onCompletion(RecordMetadata metadata, Exception exception); | Indicates the interface method for a callback function. This method is used to process asynchronous transmission results.                                                |

## Major Interfaces of a Consumer

Table 21-6 Major parameters of a Consumer

| Parameter                  | Description                               | Remarks                                                                                                                                                                                                     |
|----------------------------|-------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| bootstrap.servers          | Broker address list                       | The Consumer creates connections with the Broker based on this parameter.                                                                                                                                   |
| security.protocol          | Security protocol type                    | The Consumer uses the security protocol of the type specified by this parameter. In security mode, only the SASL protocol is supported, so this parameter needs to be configured as <b>SASL_PLAINTEXT</b> . |
| sasl.kerberos.service.name | Service name                              | This parameter specifies the Kerberos user name used by the Kafka cluster for running. This parameter needs to be configured as <b>kafka</b> .                                                              |
| key.deserializer           | Deserialization of key values in messages | This parameter specifies how to deserialize the key values in messages.                                                                                                                                     |

| Parameter          | Description                 | Remarks                                                        |
|--------------------|-----------------------------|----------------------------------------------------------------|
| value.deserializer | Deserialization of messages | This parameter specifies how to deserialize received messages. |

**Table 21-7** Major interface functions of aConsumer

| Return Value         | Interface Function                                       | Description                                              |
|----------------------|----------------------------------------------------------|----------------------------------------------------------|
| void                 | close()                                                  | Indicates the interface method for closing the Consumer. |
| void                 | subscribe(java.util.Collection<java.lang.String> topics) | Indicates the interface method for subscribing topics.   |
| ConsumerRecords<K,V> | poll(final Duration timeout)                             | Indicates the interface method for requesting messages.  |

## 21.5.2 Kafka Access Configuration on Windows Using EIPs

### Scenario

This section describes how to bind Elastic IP addresses (EIPs) to a cluster and configure Kafka files so that sample files can be compiled locally.

### Procedure

- Step 1** Apply for an EIP for each node in the cluster and add public IP addresses and corresponding host domain names of all nodes to the Windows local **hosts** file. (If a host name contains uppercase letters, change them to lowercase letters.)
- On the VPC console, apply for EIPs (the number of EIPs you buy should be equal to the number of nodes in the cluster), click the name of each node in the MRS cluster, and bind an EIP to each node on the **EIPs** page.  
For details, see **Virtual Private Cloud > User Guide > EIP > Assigning an EIP and Binding It to an ECS**.
  - Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.

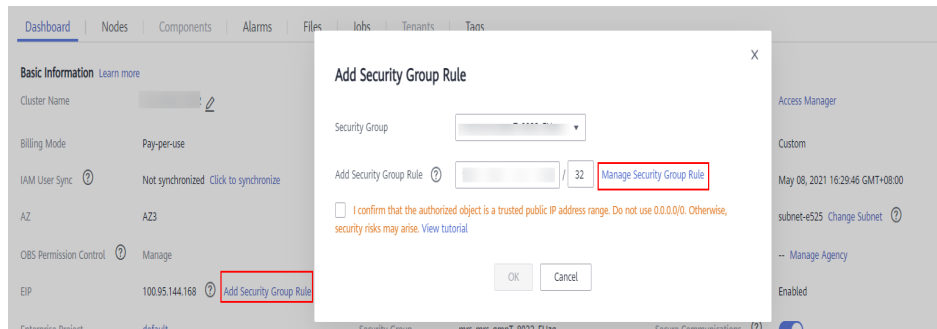
```

0 10 20 30 40 50 60 70 80 90 100 110 120 130 140
1 Mapping between public IP addresses and private IP addresses
2 100.95.111.229 172.16.0.128
3 100.95.111.230 172.16.0.42
4 100.93.10.115 172.16.0.62
5 100.95.111.231 172.16.0.200
6 100.93.10.116 172.16.0.139
7 100.93.10.117 172.16.0.214
8
9 hosts file in the cluster
10 172.16.0.128 node-group-1XZI0002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
11 172.16.0.42 node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com.
12 172.16.0.62 node-group-1XZI0003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
13 172.16.0.200 node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
14 172.16.0.139 node-group-1XZI0001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
15 172.16.0.214 node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.
16
17 hosts file that should be added to Windows
18 100.95.111.229 node-group-1xzi0002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
19 100.95.111.230 node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com.
20 100.93.10.115 node-group-1xzi0003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
21 100.95.111.231 node-master1ceip.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1ceip.ead64699-185a-4290-bbef-1a07e2f0459b.com.
22 100.93.10.116 node-group-1xzi0001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
23 100.93.10.117 node-master2pvnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pvnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.

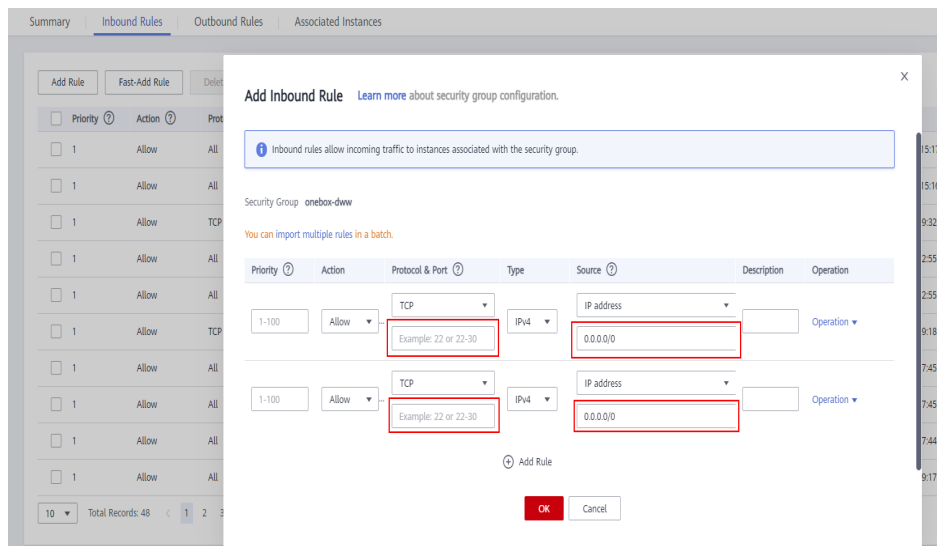
```

**Step 2** Configure security group rules for the cluster.

1. On the **Dashboard** page, choose **Add Security Group Rule > Manage Security Group Rule**.



2. On the **Inbound Rules** tab page, click **Add Rule**. In the **Add Inbound Rule** dialog box, configure the Windows IP address and port 9092.



- Step 3** On Manager, choose **Cluster > Services > Kafka > Configurations > All Configurations**, search for and add the key-value pair **advertised.listeners =PLAINTEXT://:9092,SSL://:9093,TRACE://:21013** in the **kafka.config.expander** parameter, save the configuration, and restart the Kafka cluster.

- Step 4** Before running the sample code, change the Kafka connection string in the sample code to **hostname1:9092, hostname2:9092, hostname3:9092**.

```

security.protocol = PLAINTEXT
kerberos.domain.name = hadoop.hadoop.com
acks = 1
bootstrap.servers = node-group-1XZ100002.ead64699-185a-4298-bbef-1a67e2f0459b.com:9092,node-group-1XZ100003.ead64699-185a-4298-bbef-1a67e2f0459b.com:9092,node-group-1XZ100004.ead64699-185a-4298-bbef-1a67e2f0459b.com:9092
producer.type = sync
serializer.class = kafka.serializer.DefaultEncoder

```

----End

## 21.5.3 FAQ

### 21.5.3.1 Running the Producer.java Sample to Obtain Metadata Fails and "ERROR fetching topic metadata for topics..." Is Displayed, Even the Access Permission for the Related Topic

#### Troubleshooting Procedure

- Step 1** Find **bootstrap.servers** in **producer.properties** under the **conf** directory of the project, and check whether the IP address and port ID are configured correctly.
- If the IP address is inconsistent with the service IP address of the Kafka cluster, change the IP address to the correct one.
  - If the port ID is 21007 (security mode port), change it to 9092 (normal mode port).

- Step 2** Check whether network connections are correct to ensure that the current device can access the Kafka cluster normally.

----End

# 22 Kudu Development Guide (Security Mode)

---

## 22.1 Overview

### 22.1.1 Introduction to Kudu

#### Introduction to Kudu

**Kudu** is a columnar storage manager developed for the Apache Hadoop platform. Kudu shares the common technical properties of Hadoop ecosystem applications: it runs on commodity hardware, is horizontally scalable, and supports highly available operation.

Kudu's design has the following benefits:

- Fast processing of OLAP workloads
- Integration with MapReduce, Spark and other Hadoop ecosystem components
- Tight integration with Apache Impala, making it a good, mutable alternative to using HDFS with Apache Parquet
- Strong but flexible consistency model, allowing you to choose consistency requirements on a per-request basis, including the option for strict-serializable consistency
- Strong performance for running sequential and random workloads simultaneously
- Easy to manage
- High availability. TServers and Masters use the Raft algorithm, which ensures that as long as more than half the total number of replicas is available, the tablet is available for reads and writes. For example, if 2 out of 3 replicas or 3 out of 5 replicas are available, the tablet is available. Reads can be serviced by read-only follower tablets, even in the event of a leader tablet failure.
- Structured data model



By combining all of these properties, Kudu targets support for families of applications that are difficult or impossible to implement on current generation Hadoop storage technologies.

A few examples of applications for which Kudu is a great solution are:

- Reporting applications where newly-arrived data needs to be immediately available for end users
- Time-series applications that must simultaneously support queries across large amounts of historic data and granular queries about an individual entity that must return very quickly
- Applications that use predictive models to make real-time decisions with periodic refreshes of the predictive model based on all historic data

## Kudu APIs

Kudu is developed using C++. However, it supports programming languages including C++, Java, and Python. You are advised to use Java to develop Kudu applications.

Kudu uses the same APIs as Apache Kudu. For details, see <https://kudu.apache.org/apidocs/>.

## 22.1.2 Basic Concepts

### Table

A table has the schema and primary key attributes and can be divided into multiple tablets.

### Tablet

A tablet refers to a data slice. The number of replicas can be specified and stored on multiple tablet servers. One of the replicas is the leader tablet. All replicas can be read, but only the leader can perform the write operation. The write operation uses the Raft algorithm.

### Tablet Server

A tablet server is a data storage node. It stores tablets and responds to client requests. One tablet server stores multiple tablets.

### Master

A master node is the central management node, which manages the association between all tablets, tablet servers, and replicas. Only one acting master (leader master) exists in the cluster at a time. If the leader master is suspended, a new master will be elected using the Raft algorithm. All master data is stored in a tablet, and the tablet is copied to all candidate masters. The tablet server periodically sends heartbeat messages to the master.

## kudu

Kudu management tool, which can be used to check the cluster health status and perform routine O&M

## Keytab File

Key file for storing user information. The application uses this key file for API authentication in the component.

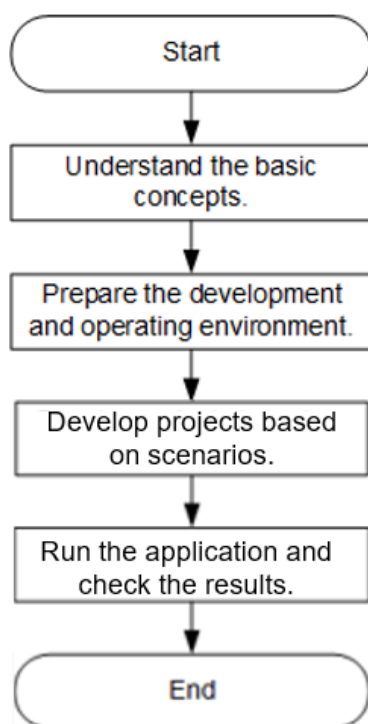
## Schema

Table information, which is used to indicate the column information in the table.

## 22.1.3 Development Process

[Figure 22-1](#) and [Table 22-1](#) describe the phases in the development process.

**Figure 22-1** Kudu application development process



**Table 22-1** Description of the Kudu application development process

| Phase                      | Description                                                                                   | Reference Document             |
|----------------------------|-----------------------------------------------------------------------------------------------|--------------------------------|
| Understand basic concepts. | Before the application development, the basic concepts of Kudu are required to be understood. | <a href="#">Basic Concepts</a> |

| Phase                                               | Description                                                                                                                                                                                    | Reference Document                                                                     |
|-----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| Prepare the development and operating environments. | Kudu applications can be developed in multiple languages. Generally, Java is used. The Eclipse or IntelliJ IDEA tool is recommended. Configure the development environment based on the guide. | <a href="#">Preparing the Development and Running Environment</a>                      |
| Develop projects based on the scenarios.            | Kudu provides sample projects to help you quickly understand APIs of Kudu components.                                                                                                          | <a href="#">Developing an Application</a><br><a href="#">Developing an Application</a> |
| Check the application running results.              | This phase provides guidance for users to submit a developed application for running and view the result.                                                                                      | <a href="#">Commissioning the Application</a>                                          |

## 22.2 Environment Preparation

### 22.2.1 Preparing the Development and Running Environment

#### Preparing the Development Environment

Table 1-2 describes the environment required for application development.

**Table 22-2** Development environment

| Preparation | Description                                                                                                                                                                                                                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OS          | <ul style="list-style-type: none"> <li>Development environment: Windows OS. Windows 7 or later is supported.</li> <li>Operating environment: Linux OSs.<br/>If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li> </ul> |

| Preparation                                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| JDK Installation                             | <p>Basic configurations of the development and operating environments. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_242, and therefore a JDK replacement is not allowed.</p> <p>For customers' applications that need to reference the JAR files of SDK to run in the application processes:</p> <ul style="list-style-type: none"> <li>• x86 client: Oracle JDK 1.8 is supported.</li> <li>• TaiShan client: OpenJDK (1.8.0_242 version is supported).</li> </ul> <p><b>NOTE</b><br/>For security purposes, the server supports only TLS V1.2 and later encryption protocols.</p> |
| IntelliJ IDEA installation and configuration | <p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>• If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK.</li> <li>• If you use Open JDK, ensure that the JDK configured in IntelliJ IDEA is Open JDK.</li> <li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li> </ul>                                                                                                                                            |
| Maven installation                           | Basic configurations of the development environment for project management throughout the lifecycle of software development.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Preparing a development user                 | Prepare a cluster user for application development and grant permissions to the user. For details, see <a href="#">Preparing a Developer Account</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 7-zip                                        | This tool is used to decompress *.zip and *.rar files. 7-zip 16.04 is supported.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

## Preparing the Operation Environment

During application development, you need to prepare the Linux environment for code running and commissioning to verify that the application is running properly.

1. Install the client on the node. For example, the client installation directory is `/opt/client`.  
 Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
 By default, the client has been installed on the Master or Core node in the cluster. You can directly use the client. For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).
2. [Log in to the FusionInsight Manager portal](#). Download the cluster client software package to the active management node and decompress it. Then,

log in to the active management node as user **root**. Go to the directory where the cluster client is decompressed. Copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Kudu/config** directory to the **conf** directory where the JAR file to be compiled is stored, for example, **/opt/client/conf**, on the client node for subsequent commissioning.

The **keytab** file obtained during development user preparation is also stored in this directory. **Table 22-3** describes the main configuration files.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Kudu/config/* root@IP address of the client node:/opt/client/conf
```

**Table 22-3** Configuration files

| File        | Function                                                        |
|-------------|-----------------------------------------------------------------|
| user.keytab | Provides user information for Kerberos security authentication. |
| krb5.conf   | Contains Kerberos Server configuration information.             |

3. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides. Ensure that the local host can communicate with each host in the cluster on the network.

## 22.2.2 Preparing for Security Authentication

### Scenario

Before accessing services in the secure cluster environment, you must be authorized by Kerberos. Codes for security authentication need to be written into the Kudu applications to ensure that the applications can work properly.

Two security authentication methods are described as follows:

- Authentication by running command lines:

Before running the Kudu applications, run the following command on the Kudu client to obtain authentication:

```
kinit component service user
```

 NOTE

This method applies only to the Linux OS that is installed with the Kudu client.

- Code authentication:  
Obtain the principal and keytab files of the client for authentication.

## 22.3 Developing an Application

### 22.3.1 Typical Application Scenario

#### Description

You can quickly learn and master the Kudu development process and know key interface functions in a typical application scenario.

### 22.3.2 Development Idea

#### Development Idea

As a storage engine, Kudu works with the computing engine in the following scenarios:

- Run SQL statements on the computing engine (such as Impala) to create a table object.
- Then, the Kudu driver writes data to the table.
- Data in the table can be directly queried on the computing engine.

In this development program example, Kudu is used as an example to avoid introducing extra computing engines. Java APIs are used to describe how to:

- Establish a connection.
- Create a table.
- Write data.
- Modify a table.
- Delete a table.

### 22.3.3 Sample Code Description

#### 22.3.3.1 Establish Connections

#### Function

Create a `KuduClient` object using the `KuduClient.KuduClientBuilder(KUDU_MASTERS).build()` method. The input parameter `KUDU_MASTERS` indicates the master address list of the Kudu cluster. If there are multiple master nodes, separate them with commas (,).

## Sample Code

The following is a code snippet for establishing a connection:

```
// Create the Kudu connection object.
KuduClient client = new KuduClient.KuduClientBuilder(KUDU_MASTERS).build();
```

### NOTE

In the sample code, **KUDU\_MASTERS** indicates the masters address list of the Kudu cluster, for example, **192.168.0.100:7051, 192.168.0.101:7051, 192.168.0.102:7051**. The format is *address.port number*. Use commas (,) to separate different addresses. The persistent connection object is recommended for production.

## 22.3.3.2 Creating a Table

### Function

Use the **KuduClient.createTable(String name, Schema schema, CreateTableOptions builder)** method to create a table object, in which the schema and partition information of the table must be specified.

## Sample Code

The code snippet for creating a table is as follows:

```
// Set up a table name.
String tableName = "example";

// Set up a simple schema.
List<ColumnSchema> columns = new ArrayList<>(2);
columns.add(new ColumnSchema.ColumnSchemaBuilder("key", Type.INT32)
 .key(true)
 .build());
columns.add(new ColumnSchema.ColumnSchemaBuilder("value",Type.STRING).nullable(true)
 .build());
Schema schema = new Schema(columns);

// Set up the partition schema, which distributes rows to different tablets by hash.
// Kudu also supports partitioning by key range. Hash and range partitioning can be combined.
// For more information, see http://kudu.apache.org/docs/schema_design.html.
CreateTableOptions cto = new CreateTableOptions();
List<String> hashKeys = new ArrayList<>(1);
hashKeys.add("key");
int numBuckets = 8;
cto.addHashPartitions(hashKeys, numBuckets);

// Create the table.
client.createTable(tableName, schema, cto);
```

### NOTE

In the sample code, a table is defined, including the **key** and **value** columns. **key** is a primary key field of the int32 type, and **value** is a non-primary key field of the string type and can be empty. In addition, eight hash partitions are created for the key field, indicating that the data is divided into eight independent tablets.

### 22.3.3.3 Opening a Table

#### Function

Use the **KuduClient.openTable(final String name)** method to open a table object.

#### Sample Code

The code snippet for opening the table is as follows:

```
// Open the Kudu table.
KuduTable table = client.openTable(tableName);
```

#### NOTE

In the sample code, *tableName* is the name of the table to be opened.

### 22.3.3.4 Modifying a Table

#### Function

Modify a table object using **KuduClient.alterTable(String name, AlterTableOptions ato)**.

#### Sample Code

The code snippet for writing data is as follows:

```
// Alter the table, adding a column with a default value.
// Note: after altering the table, the table needs to be re-opened.
AlterTableOptions ato = new AlterTableOptions();
ato.addColumn("added", org.apache.kudu.Type.DOUBLE, DEFAULT_DOUBLE);
client.alterTable(tableName, ato);
```

#### NOTE

In the sample code, **AlterTableOptions** is the set of table attributes to be modified. In the example, a column is added to the table.

### 22.3.3.5 Writing Data

#### Function

Use the **KuduClient.newSession()** method to generate a KuduSession object and execute the record insertion action to the Kudu table.

#### Sample Code

The code snippet for writing data is as follows:

```
// Create a KuduSession.
KuduSession session = client.newSession();
for (int i = 0; i < numRows; i++) {
 Insert insert = table.newInsert();
 PartialRow row = insert.getRow();
 row.addInt("key", i);
 // Make even-keyed row have a null 'value'.
}
```



```
if (i % 2 == 0) {
row.setNull("value");
} else {
row.addString("value", "value " + i);
}
session.apply(insert);
}

// Call session.close() to end the session and ensure the rows are
// flushed and errors are returned.
// You can also call session.flush() to do the same without ending the session.
// When flushing in AUTO_FLUSH_BACKGROUND mode (the default mode recommended
// for most workloads, you must check the pending errors as shown below, since
// write operations are flushed to Kudu in background threads.
session.close();
if (session.countPendingErrors() != 0) {
System.out.println("errors inserting rows");
org.apache.kudu.client.RowErrorsAndOverflowStatus roStatus = session.getPendingErrors();
org.apache.kudu.client.RowError[] errs = roStatus.getRowErrors();
int numErrs = Math.min(errs.length, 5);
System.out.println("there were errors inserting rows to Kudu");
System.out.println("the first few errors follow:");
for (int i = 0; i < numErrs; i++) {
System.out.println(errs[i]);
}
if (roStatus.isOverflowed()) {
System.out.println("error buffer overflowed: some errors were discarded");
}
}
throw new RuntimeException("error inserting rows to Kudu");
}
System.out.println("Inserted " + numRows + " rows");
```

#### NOTE

In the sample code, *numRows* indicates the number of records to be written. Pay special attention to the response result of the request.

### 22.3.3.6 Read Data

#### Function

Generate a `KuduScanner` object by using the **`KuduClient.newScannerBuilder(KuduTable table)`** method, and then read data from the Kudu table by setting the predicate condition.

#### Sample Code

The code snippet for reading data is as follows:

```
KuduTable table = client.openTable(tableName);
Schema schema = table.getSchema();

// Scan with a predicate on the 'key' column, returning the 'value' and "added" columns.
List<String> projectColumns = new ArrayList<>(2);
projectColumns.add("key");
projectColumns.add("value");
projectColumns.add("added");
int lowerBound = 0;
KuduPredicate lowerPred = KuduPredicate.newComparisonPredicate(
 schema.getColumn("key"),
 ComparisonOp.GREATER_EQUAL,
 lowerBound);
int upperBound = numRows / 2;
KuduPredicate upperPred = KuduPredicate.newComparisonPredicate(
 schema.getColumn("key"),
```

```

ComparisonOp.LESS,
upperBound);
KuduScanner scanner = client.newScannerBuilder(table)
 .setProjectedColumnNames(projectColumns)
 .addPredicate(lowerPred)
 .addPredicate(upperPred)
 .build();

// Check the correct number of values and null values are returned, and
// that the default value was set for the new column on each row.
// Note: scanning a hash-partitioned table will not return results in primary key order.
int resultCount = 0;
int nullCount = 0;
while (scanner.hasMoreRows()) {
 RowResultIterator results = scanner.nextRows();
 while (results.hasNext()) {
 RowResult result = results.next();
 if (result.isNull("value")) {
 nullCount++;
 }
 double added = result.getDouble("added");
 if (added != DEFAULT_DOUBLE) {
 throw new RuntimeException("expected added=" + DEFAULT_DOUBLE +
 " but got added= " + added);
 }
 resultCount++;
 }
}

```

#### NOTE

In the sample code:

- **projectColumns** indicates the column information to be returned.
- **lowerPred** and **upperBound** indicate the predicates that take effect on the primary key.

### 22.3.3.7 Deleting a Table

#### Function

Use the **KuduClient.deleteTable(String name)** method to delete a table object.

#### Sample Code

The code snippet for deleting a table is as follows:

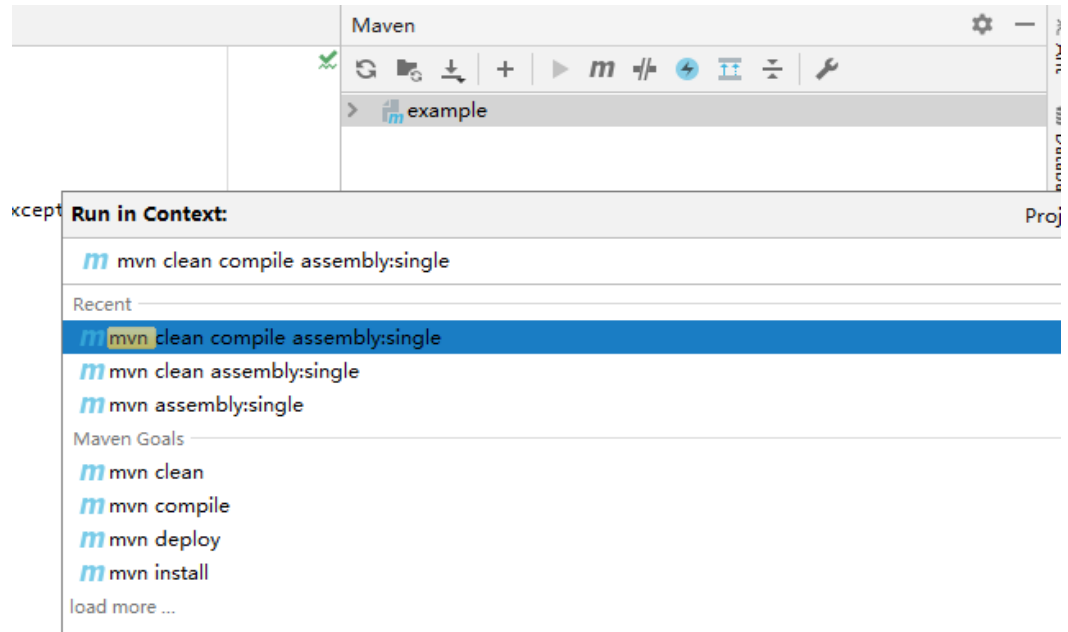
```

// Delete the table.
client.deleteTable(tableName);

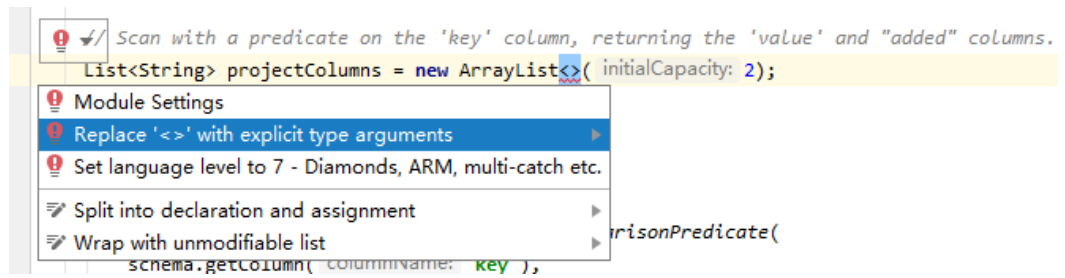
```

## 22.4 Commissioning the Application

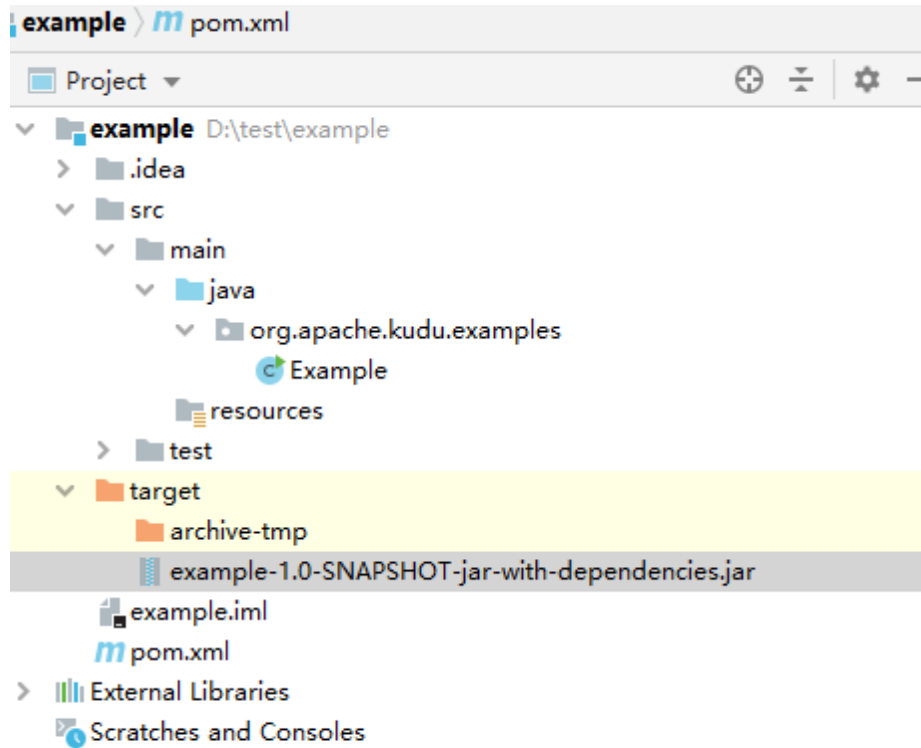
**Step 1** Use Maven for compilation.



**Step 2** Solve the exception "Error:(123,49) java: -source 1.5 does not support diamond operators".



The output package is as follows:



**Step 3** Upload the JAR package to the Linux server for execution.

```
[root@ecs-a4e6 ~]# java -DkuduMasters=localhost:7051 -jar example-1.0-SNAPSHOT-jar-with-dependencies.jar

Will try to connect to Kudu master(s) at localhost:7051
Run with -DkuduMasters=master-0:port,master-1:port,... to override.

log4j:WARN No appenders could be found for logger (org.apache.kudu.shaded.io.netty.util.internal.logging.InternalLoggerFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Created table java_example-1599014880440
Inserted 150 rows
Altered the table
Scanned some rows and checked the results
Deleted the table
[root@ecs-a4e6 ~]#
```

----End

## 22.5 More Information

### 22.5.1 Common APIs

#### 22.5.1.1 Java API

For details about Kudu APIs, visit the following website: <https://kudu.apache.org/apidocs>.

# 23 Kudu Development Guide (Normal Mode)

---

## 23.1 Overview

### 23.1.1 Introduction to Kudu

#### Introduction to Kudu

**Kudu** is a columnar storage manager developed for the Apache Hadoop platform. Kudu shares the common technical properties of Hadoop ecosystem applications: it runs on commodity hardware, is horizontally scalable, and supports highly available operation.

Kudu's design has the following benefits:

- Fast processing of OLAP workloads
- Integration with MapReduce, Spark and other Hadoop ecosystem components
- Tight integration with Apache Impala, making it a good, mutable alternative to using HDFS with Apache Parquet
- Strong but flexible consistency model, allowing you to choose consistency requirements on a per-request basis, including the option for strict-serializable consistency
- Strong performance for running sequential and random workloads simultaneously
- Easy to manage
- High availability. TServers and Masters use the Raft algorithm, which ensures that as long as more than half the total number of replicas is available, the tablet is available for reads and writes. For example, if 2 out of 3 replicas or 3 out of 5 replicas are available, the tablet is available. Reads can be serviced by read-only follower tablets, even in the event of a leader tablet failure.
- Structured data model

By combining all of these properties, Kudu targets support for families of applications that are difficult or impossible to implement on current generation Hadoop storage technologies.

A few examples of applications for which Kudu is a great solution are:

- Reporting applications where newly-arrived data needs to be immediately available for end users
- Time-series applications that must simultaneously support queries across large amounts of historic data and granular queries about an individual entity that must return very quickly
- Applications that use predictive models to make real-time decisions with periodic refreshes of the predictive model based on all historic data

## Kudu APIs

Kudu is developed using C++. However, it supports programming languages including C++, Java, and Python. You are advised to use Java to develop Kudu applications.

Kudu uses the same APIs as Apache Kudu. For details, see <https://kudu.apache.org/apidocs/>.

## 23.1.2 Basic Concepts

### Table

A table has the schema and primary key attributes and can be divided into multiple tablets.

### Tablet

A tablet refers to a data slice. The number of replicas can be specified and stored on multiple tablet servers. One of the replicas is the leader tablet. All replicas can be read, but only the leader can perform the write operation. The write operation uses the Raft algorithm.

### Tablet Server

A tablet server is a data storage node. It stores tablets and responds to client requests. One tablet server stores multiple tablets.

### Master

A master node is the central management node, which manages the association between all tablets, tablet servers, and replicas. Only one acting master (leader master) exists in the cluster at a time. If the leader master is suspended, a new master will be elected using the Raft algorithm. All master data is stored in a tablet, and the tablet is copied to all candidate masters. The tablet server periodically sends heartbeat messages to the master.

## kudu

Kudu management tool, which can be used to check the cluster health status and perform routine O&M

## Keytab File

Key file for storing user information. The application uses this key file for API authentication in the component.

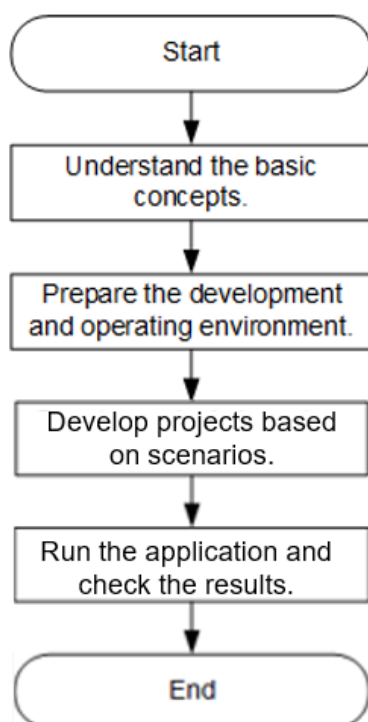
## Schema

Table information, which is used to indicate the column information in the table.

### 23.1.3 Development Process

[Figure 23-1](#) and [Table 23-1](#) describe the phases in the development process.

**Figure 23-1** Kudu application development process



**Table 23-1** Description of the Kudu application development process

| Phase                      | Description                                                                                   | Reference Document             |
|----------------------------|-----------------------------------------------------------------------------------------------|--------------------------------|
| Understand basic concepts. | Before the application development, the basic concepts of Kudu are required to be understood. | <a href="#">Basic Concepts</a> |

| Phase                                               | Description                                                                                                                                                                                    | Reference Document                                                                     |
|-----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| Prepare the development and operating environments. | Kudu applications can be developed in multiple languages. Generally, Java is used. The Eclipse or IntelliJ IDEA tool is recommended. Configure the development environment based on the guide. | <a href="#">Preparing the Development and Running Environment</a>                      |
| Develop projects based on the scenarios.            | Kudu provides sample projects to help you quickly understand APIs of Kudu components.                                                                                                          | <a href="#">Developing an Application</a><br><a href="#">Developing an Application</a> |
| Check the application running results.              | This phase provides guidance for users to submit a developed application for running and view the result.                                                                                      | <a href="#">Commissioning the Application</a>                                          |

## 23.2 Environment Preparation

### 23.2.1 Preparing the Development and Running Environment

#### Preparing the Development Environment

Table 1-2 describes the environment required for application development.

**Table 23-2** Development environment

| Preparation | Description                                                                                                                                                                                                                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OS          | <ul style="list-style-type: none"> <li>Development environment: Windows OS. Windows 7 or later is supported.</li> <li>Operating environment: Linux OSs.<br/>If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li> </ul> |



| Preparation                                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| JDK Installation                             | <p>Basic configurations of the development and operating environments. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_242, and therefore a JDK replacement is not allowed.</p> <p>For customers' applications that need to reference the JAR files of SDK to run in the application processes:</p> <ul style="list-style-type: none"> <li>• x86 client: Oracle JDK 1.8 is supported.</li> <li>• TaiShan client: OpenJDK (1.8.0_242 version is supported).</li> </ul> <p><b>NOTE</b><br/>For security purposes, the server supports only TLS V1.2 and later encryption protocols.</p> |
| IntelliJ IDEA installation and configuration | <p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>• If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK.</li> <li>• If you use Open JDK, ensure that the JDK configured in IntelliJ IDEA is Open JDK.</li> <li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li> </ul>                                                                                                                                            |
| Maven installation                           | Basic configurations of the development environment for project management throughout the lifecycle of software development.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 7-zip                                        | This tool is used to decompress *.zip and *.rar files. 7-zip 16.04 is supported.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

## Preparing the Operation Environment

During application development, you need to prepare the Linux environment for code running and commissioning to verify that the application is running properly.

1. Install the client on the node. For example, the client installation directory is `/opt/client`.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
By default, the client has been installed on the Master or Core node in the cluster. You can directly use the client. For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).
2. [Log in to the FusionInsight Manager portal](#). Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the directory where the cluster client is decompressed. Copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Kudu/config** directory to the **conf** directory where the JAR file to be compiled is stored, for example, **/opt/client/conf**, on the client node for subsequent commissioning.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Kudu/config/* root@IP address of the client node:/opt/client/conf
```

3. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides. Ensure that the local host can communicate with each host in the cluster on the network.

## 23.3 Developing an Application

### 23.3.1 Typical Application Scenario

#### Description

You can quickly learn and master the Kudu development process and know key interface functions in a typical application scenario.

### 23.3.2 Development Idea

#### Development Idea

As a storage engine, Kudu works with the computing engine in the following scenarios:

- Run SQL statements on the computing engine (such as Impala) to create a table object.
- Then, the Kudu driver writes data to the table.
- Data in the table can be directly queried on the computing engine.

In this development program example, Kudu is used as an example to avoid introducing extra computing engines. Java APIs are used to describe how to:

- Establish a connection.
- Create a table.
- Write data.
- Modify a table.
- Delete a table.

## 23.3.3 Sample Code Description

### 23.3.3.1 Establish Connections

#### Function

Create a `KuduClient` object using the `KuduClient.KuduClientBuilder(KUDU_MASTERS).build()` method. The input parameter `KUDU_MASTERS` indicates the master address list of the Kudu cluster. If there are multiple master nodes, separate them with commas (,).

#### Sample Code

The following is a code snippet for establishing a connection:

```
// Create the Kudu connection object.
KuduClient client = new KuduClient.KuduClientBuilder(KUDU_MASTERS).build();
```

#### NOTE

In the sample code, `KUDU_MASTERS` indicates the masters address list of the Kudu cluster, for example, `192.168.0.100:7051, 192.168.0.101:7051, 192.168.0.102:7051`. The format is *address.port number*. Use commas (,) to separate different addresses. The persistent connection object is recommended for production.

### 23.3.3.2 Creating a Table

#### Function

Use the `KuduClient.createTable(String name, Schema schema, CreateTableOptions builder)` method to create a table object, in which the schema and partition information of the table must be specified.

#### Sample Code

The code snippet for creating a table is as follows:

```
// Set up a table name.
String tableName = "example";

// Set up a simple schema.
List<ColumnSchema> columns = new ArrayList<>(2);
columns.add(new ColumnSchema.ColumnSchemaBuilder("key", Type.INT32)
 .key(true)
 .build());
columns.add(new ColumnSchema.ColumnSchemaBuilder("value", Type.STRING).nullable(true)
 .build());
Schema schema = new Schema(columns);

// Set up the partition schema, which distributes rows to different tablets by hash.
// Kudu also supports partitioning by key range. Hash and range partitioning can be combined.
// For more information, see http://kudu.apache.org/docs/schema_design.html.
CreateTableOptions cto = new CreateTableOptions();
List<String> hashKeys = new ArrayList<>(1);
hashKeys.add("key");
int numBuckets = 8;
cto.addHashPartitions(hashKeys, numBuckets);
```

```
// Create the table.
client.createTable(tableName, schema, cto);
```

#### NOTE

In the sample code, a table is defined, including the **key** and **value** columns. **key** is a primary key field of the int32 type, and **value** is a non-primary key field of the string type and can be empty. In addition, eight hash partitions are created for the key field, indicating that the data is divided into eight independent tablets.

### 23.3.3.3 Opening a Table

#### Function

Use the **KuduClient.openTable(final String name)** method to open a table object.

#### Sample Code

The code snippet for opening the table is as follows:

```
// Open the Kudu table.
KuduTable table = client.openTable(tableName);
```

#### NOTE

In the sample code, *tableName* is the name of the table to be opened.

### 23.3.3.4 Modifying a Table

#### Function

Modify a table object using **KuduClient.alterTable(String name, AlterTableOptions ato)**.

#### Sample Code

The code snippet for writing data is as follows:

```
// Alter the table, adding a column with a default value.
// Note: after altering the table, the table needs to be re-opened.
AlterTableOptions ato = new AlterTableOptions();
ato.addColumn("added", org.apache.kudu.Type.DOUBLE, DEFAULT_DOUBLE);
client.alterTable(tableName, ato);
```

#### NOTE

In the sample code, **AlterTableOptions** is the set of table attributes to be modified. In the example, a column is added to the table.

### 23.3.3.5 Writing Data

#### Function

Use the **KuduClient.newSession()** method to generate a KuduSession object and execute the record insertion action to the Kudu table.

## Sample Code

The code snippet for writing data is as follows:

```
// Create a KuduSession.
KuduSession session = client.newSession();
for (int i = 0; i < numRows; i++) {
 Insert insert = table.newInsert();
 PartialRow row = insert.getRow();
 row.addInt("key", i);
 // Make even-keyed row have a null 'value'.
 if (i % 2 == 0) {
 row.setNull("value");
 } else {
 row.addString("value", "value " + i);
 }
 session.apply(insert);
}

// Call session.close() to end the session and ensure the rows are
// flushed and errors are returned.
// You can also call session.flush() to do the same without ending the session.
// When flushing in AUTO_FLUSH_BACKGROUND mode (the default mode recommended
// for most workloads, you must check the pending errors as shown below, since
// write operations are flushed to Kudu in background threads.
session.close();
if (session.countPendingErrors() != 0) {
 System.out.println("errors inserting rows");
 org.apache.kudu.client.RowErrorsAndOverflowStatus roStatus = session.getPendingErrors();
 org.apache.kudu.client.RowError[] errs = roStatus.getRowErrors();
 int numErrs = Math.min(errs.length, 5);
 System.out.println("there were errors inserting rows to Kudu");
 System.out.println("the first few errors follow:");
 for (int i = 0; i < numErrs; i++) {
 System.out.println(errs[i]);
 }
 if (roStatus.isOverflowed()) {
 System.out.println("error buffer overflowed: some errors were discarded");
 }
 throw new RuntimeException("error inserting rows to Kudu");
}
System.out.println("Inserted " + numRows + " rows");
```

### NOTE

In the sample code, *numRows* indicates the number of records to be written. Pay special attention to the response result of the request.

### 23.3.3.6 Read Data

#### Function

Generate a `KuduScanner` object by using the **`KuduClient.newScannerBuilder(KuduTable table)`** method, and then read data from the Kudu table by setting the predicate condition.

#### Sample Code

The code snippet for reading data is as follows:

```
KuduTable table = client.openTable(tableName);
Schema schema = table.getSchema();

// Scan with a predicate on the 'key' column, returning the 'value' and "added" columns.
List<String> projectColumns = new ArrayList<>(2);
```

```
projectColumns.add("key");
projectColumns.add("value");
projectColumns.add("added");
int lowerBound = 0;
KuduPredicate lowerPred = KuduPredicate.newComparisonPredicate(
 schema.getColumn("key"),
 ComparisonOp.GREATER_EQUAL,
 lowerBound);
int upperBound = numRows / 2;
KuduPredicate upperPred = KuduPredicate.newComparisonPredicate(
 schema.getColumn("key"),
 ComparisonOp.LESS,
 upperBound);
KuduScanner scanner = client.newScannerBuilder(table)
 .setProjectedColumnNames(projectColumns)
 .addPredicate(lowerPred)
 .addPredicate(upperPred)
 .build();

// Check the correct number of values and null values are returned, and
// that the default value was set for the new column on each row.
// Note: scanning a hash-partitioned table will not return results in primary key order.
int resultCount = 0;
int nullCount = 0;
while (scanner.hasMoreRows()) {
 RowResultIterator results = scanner.nextRows();
 while (results.hasNext()) {
 RowResult result = results.next();
 if (result.isNull("value")) {
 nullCount++;
 }
 double added = result.getDouble("added");
 if (added != DEFAULT_DOUBLE) {
 throw new RuntimeException("expected added=" + DEFAULT_DOUBLE +
 " but got added=" + added);
 }
 resultCount++;
 }
}
```

#### NOTE

In the sample code:

- **projectColumns** indicates the column information to be returned.
- **lowerPred** and **upperBound** indicate the predicates that take effect on the primary key.

### 23.3.3.7 Deleting a Table

#### Function

Use the **KuduClient.deleteTable(String name)** method to delete a table object.

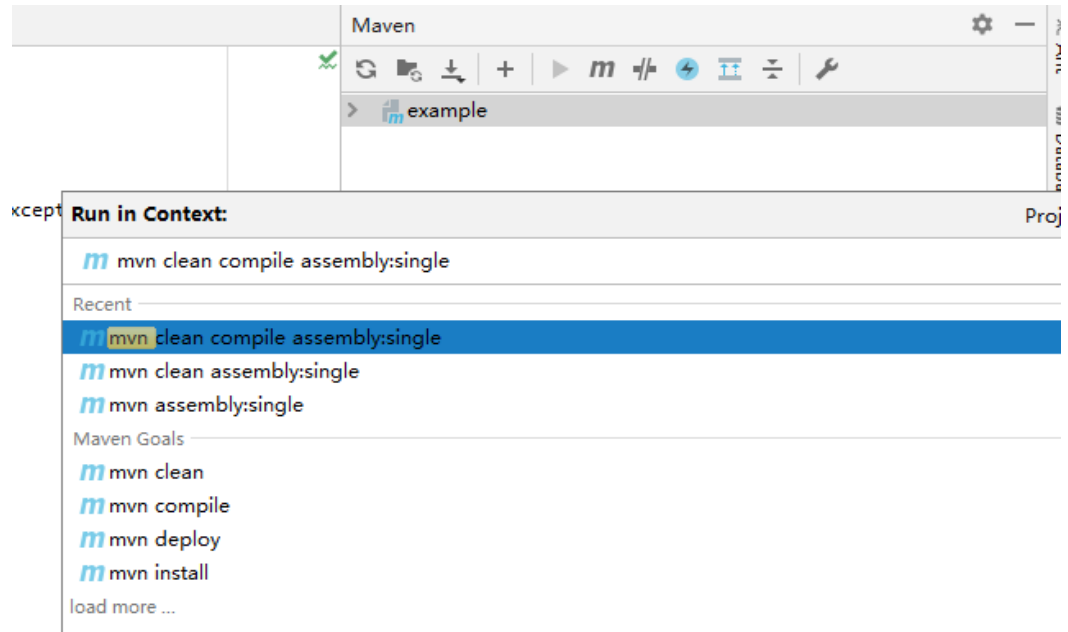
#### Sample Code

The code snippet for deleting a table is as follows:

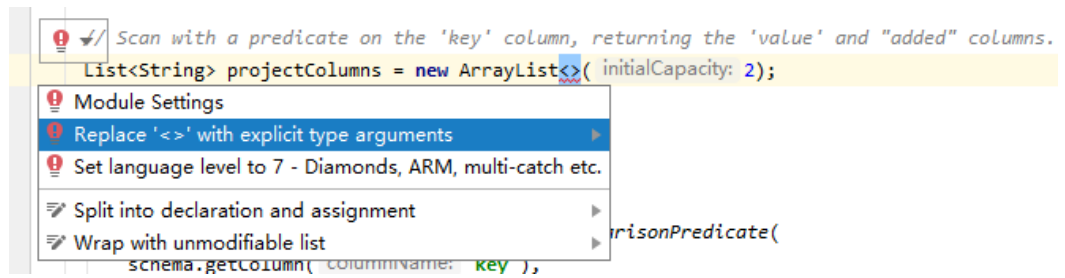
```
// Delete the table.
client.deleteTable(tableName);
```

## 23.4 Commissioning the Application

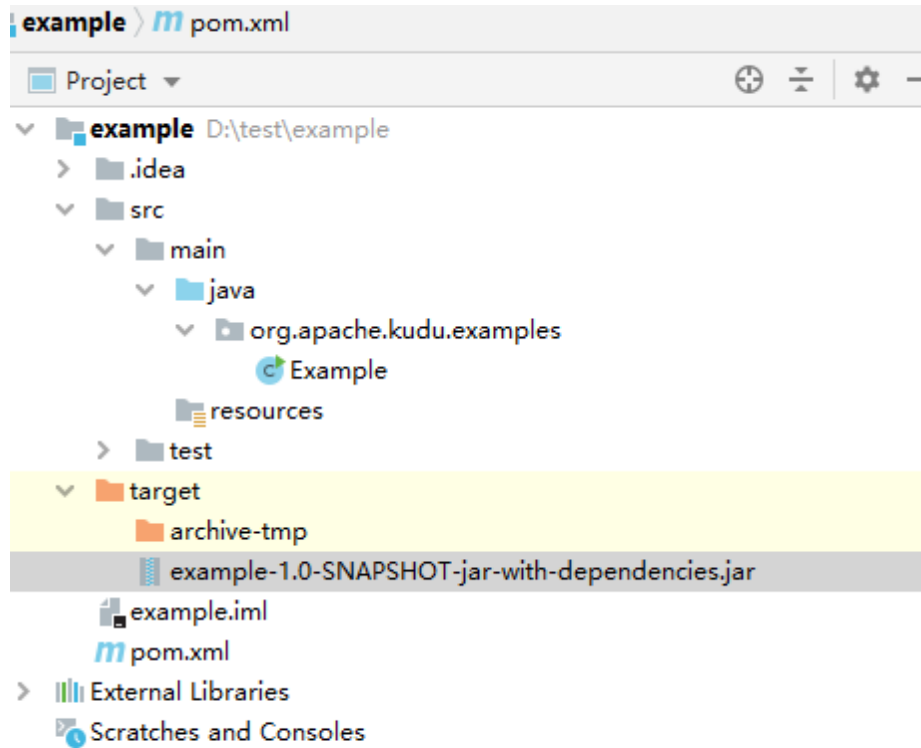
**Step 1** Use Maven for compilation.



**Step 2** Solve the exception "Error:(123,49) java: -source 1.5 does not support diamond operators".



The output package is as follows:



**Step 3** Upload the JAR package to the Linux server for execution.

```
[root@ecs-a4e6 ~]# java -DkuduMasters=localhost:7051 -jar example-1.0-SNAPSHOT-jar-with-dependencies.jar

Will try to connect to Kudu master(s) at localhost:7051
Run with -DkuduMasters=master-0:port,master-1:port,... to override.

log4j:WARN No appenders could be found for logger (org.apache.kudu.shaded.io.netty.util.internal.logging.InternalLoggerFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Created table java_example-1599014880440
Inserted 150 rows
Altered the table
Scanned some rows and checked the results
Deleted the table
[root@ecs-a4e6 ~]#
```

----End

## 23.5 More Information

### 23.5.1 Common APIs

#### 23.5.1.1 Java API

For details about Kudu APIs, visit the following website: <https://kudu.apache.org/apidocs>.



# 24 MapReduce Development Guide (Security Mode)

---

## 24.1 Overview

### 24.1.1 MapReduce Overview

#### MapReduce Introduction

Hadoop MapReduce is an easy-to-use parallel computing software framework. Applications developed based on MapReduce can run on large clusters consisting of thousands of servers and process data sets larger than 1 TB in fault tolerance (FT) mode.

A MapReduce job (application or job) splits an input data set into several data blocks which then are processed by Map tasks in parallel mode. The framework sorts output results of the Map task, sends the results to Reduce tasks, and returns a result to the client. Input and output information is stored in the Hadoop Distributed File System (HDFS). The framework schedules and monitors tasks and re-executes failed tasks.

MapReduce supports the following features:

- Large-scale parallel computing
- Large data set processing
- High FT and reliability
- Reasonable resource scheduling

### 24.1.2 Basic Concepts

#### Hadoop shell command

Basic hadoop shell commands include commands that are used to submit MapReduce jobs, kill MapReduce jobs, and perform operations on the HDFS.

## MapReduce InputFormat and OutputFormat

Based on the specified InputFormat, the MapReduce framework splits data sets, reads data, provides key-value pairs for Map tasks, and determines the number of Map tasks that are started in parallel mode. Based on the OutputFormat, the MapReduce framework outputs the generated key-value pairs to data in a specific format.

Map and Reduce tasks are running based on <key,value> pairs. In other words, the framework regards the input information about a job as a group of key-value pairs and outputs a group of key-value pairs. Two groups of key-value pairs may be of different types. For a single Map or Reduce task, key-value pairs are processed in single-thread serial mode.

The framework needs to perform serialized operations on key and value classes. Therefore, the classes must support the Writable interface. To facilitate sorting operations, key classes must support the WritableComparable interface.

The input and output types of a MapReduce job are as follows:

(input) <k1,v1> -> Map -> <k2,v2> -> Summary data -> <k2, List(v2)> -> Reduce -> <k3,v3> (output)

## Job Core

In normal cases, an application only needs to inherit Mapper and Reducer classes and rewrite map and reduce methods to implement service logic. The map and reduce methods constitute the core of jobs.

## MapReduce WebUI

Allows users to monitor running or historical MapReduce jobs, view logs, and implement fine-grained job development, configuration, and optimization.

## Keytab file

A key file for storing user information. Applications use the key file for application programming interface (API) authentication on product.

## Reduce

A processing model function that merges all intermediate values associated with the same intermediate key.

## Shuffle

A process of outputting data from a Map task to a Reduce task.

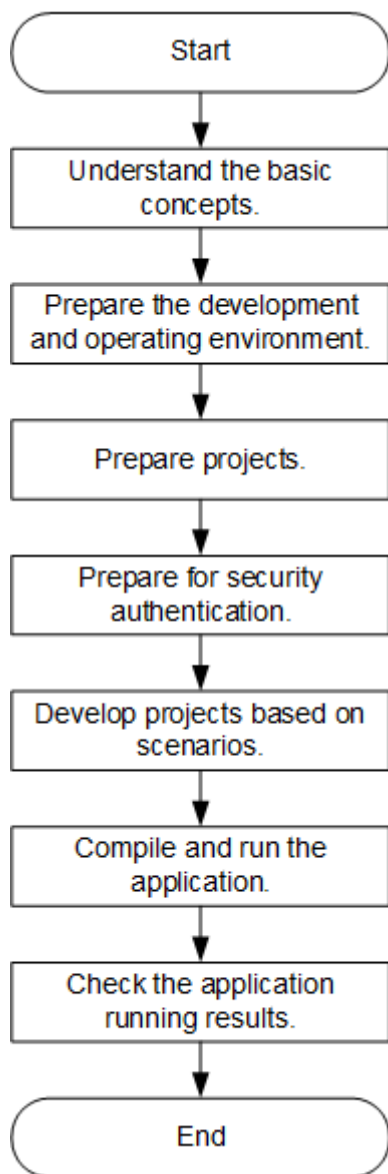
## Map

A method used to map a group of key-value pairs into a new group of key-value pairs.

### 24.1.3 Development Process

All stages of the development process are shown and described in [Figure 24-1](#) and [Table 24-1](#).

**Figure 24-1** MapReduce development process



**Table 24-1** Description of MapReduce development process

| Stage                          | Description                                                                                        | Reference                      |
|--------------------------------|----------------------------------------------------------------------------------------------------|--------------------------------|
| Understand the basic concepts. | Before the application development, the basic concepts of MapReduce are required to be understood. | <a href="#">Basic Concepts</a> |

| Stage                                              | Description                                                                                                                                                                                                   | Reference                                                                                                      |
|----------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| Prepare the development and operating environment. | Use IntelliJ IDEA and configure the development environment based on the reference.<br>The running environment of MapReduce is the MapReduce client. Install and configure the client based on the reference. | <a href="#">Preparing for Development and Operating Environment</a>                                            |
| Prepare projects                                   | MapReduce provides sample projects in various scenarios. Sample projects can be imported for studying. Or you can create a new MapReduce project based on the reference.                                      | <a href="#">Configuring and Importing Sample Projects</a><br><a href="#">Creating a New Project (Optional)</a> |
| Prepare for safety certification.                  | If a safe cluster is used, the safety certification must be performed.                                                                                                                                        | <a href="#">Preparing the Authentication Mechanism</a>                                                         |
| Develop projects based on scenarios.               | Provide the example project.<br>This helps users to learn about the programming interfaces of all Spark components quickly.                                                                                   | <a href="#">Developing the Project</a>                                                                         |
| Compile and run the application.                   | Users compile the developed application and deliver it for running based on the reference.                                                                                                                    | <a href="#">Commissioning the Application</a>                                                                  |
| Check the application running results.             | Application running results are stored in the directory specified by users. Users can also check the running results through the UI.                                                                          | <a href="#">Commissioning the Application</a>                                                                  |

## 24.2 Environment Preparation

### 24.2.1 Preparing for Development and Operating Environment

#### Preparing Development Environment

[Table 24-2](#) describes the environment required for application development.

**Table 24-2** Development environment

| Preparation Item                             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OS                                           | <ul style="list-style-type: none"> <li>Development environment: Windows OS. Windows 7 or later is supported.</li> <li>Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| IntelliJ IDEA installation and configuration | <p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li> <li>If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li> <li>If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li> <li>Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Maven installation                           | <p>Basic configuration of the development environment for project management throughout the lifecycle of software development.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| JDK installation                             | <p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> <li>For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> <li>Oracle JDK versions: 1.8</li> <li>IBM JDK versions: 1.8.5.11</li> </ul> </li> <li>For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> <li>OpenJDK: 1.8.0_272</li> </ul> </li> </ul> <p><b>NOTE</b></p> <p>The server supports only TLS V1.2 or later to ensure security.</p> <p>By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls">https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls</a>.</p> |

| Preparation Item              | Description                                                                            |
|-------------------------------|----------------------------------------------------------------------------------------|
| Developer account preparation | See <a href="#">Preparing a Developer Account</a> for configuration.                   |
| 7-zip                         | Used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-Zip 16.04 is supported. |

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. [Log in to the FusionInsight Manager portal](#) and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
 For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar file**. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar file** to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Yarn\config** and obtain the cluster configuration file. The configuration file will be imported to the configuration file directory (usually the **conf** folder) of the MapReduce sample project.
  - c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.

- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client on the node. For example, the client installation directory is **/opt/client**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. [Log in to the FusionInsight Manager portal](#). Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Yarn\config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/client/conf**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Yarn/config/* root@IP address of the client node:/opt/client/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

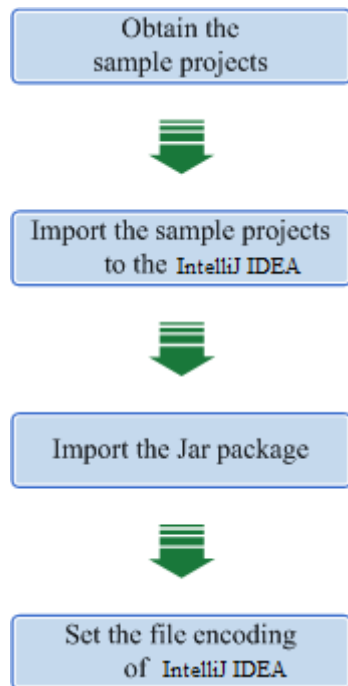
## 24.2.2 Configuring and Importing Sample Projects

### Scenario

MapReduce provides sample projects for multiple scenarios to help you quickly learn MapReduce projects.

The procedure of importing MapReduce example code is described as follows: [Figure 24-2](#) shows the procedure.

**Figure 24-2** Procedure of importing sample projects

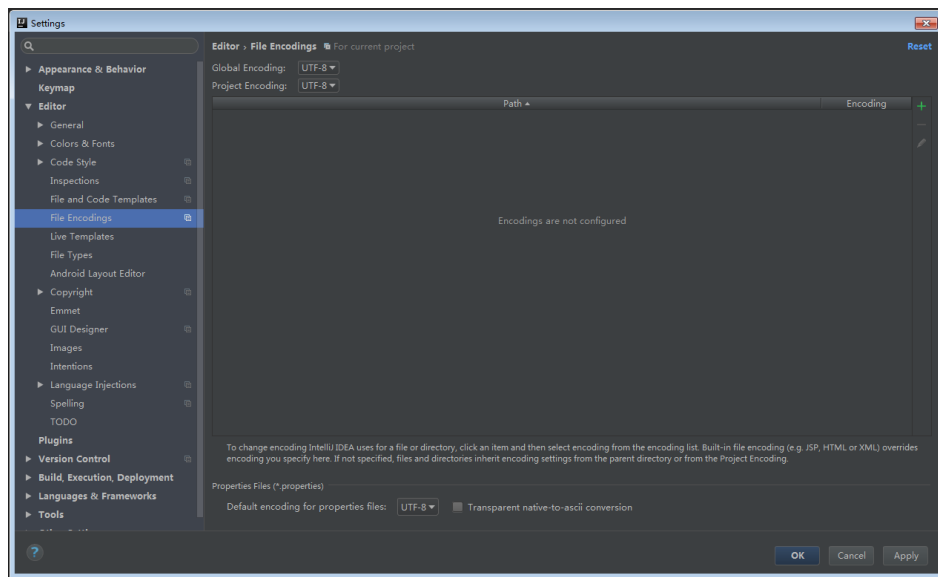


## Procedure

- Step 1** Obtain the sample project folder **mapreduce-example-security** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the **user.keytab** and **krb5.conf** files obtained during [Preparing a Developer Account](#) and the cluster configuration file obtained during running environment preparation in [Preparing an Operating Environment](#) to the **conf** directory of the sample project.
- Step 3** Import the sample project to the IntelliJ IDEA development environment.
  1. Open IntelliJ IDEA and choose **File > Open**.
  2. Choose the directory of the example project **mapreduce-example-security**. Click **OK**.
- Step 4** Set the IntelliJ IDEA text file coding format to prevent garbled characters.
  1. On the IntelliJ IDEA menu bar, choose **File > Settings**. The **Settings** window is displayed.
  2. Choose **Editor > File Encodings** from the navigation tree. In the "Global Encoding" and "Project Encodings" area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 24-3](#)



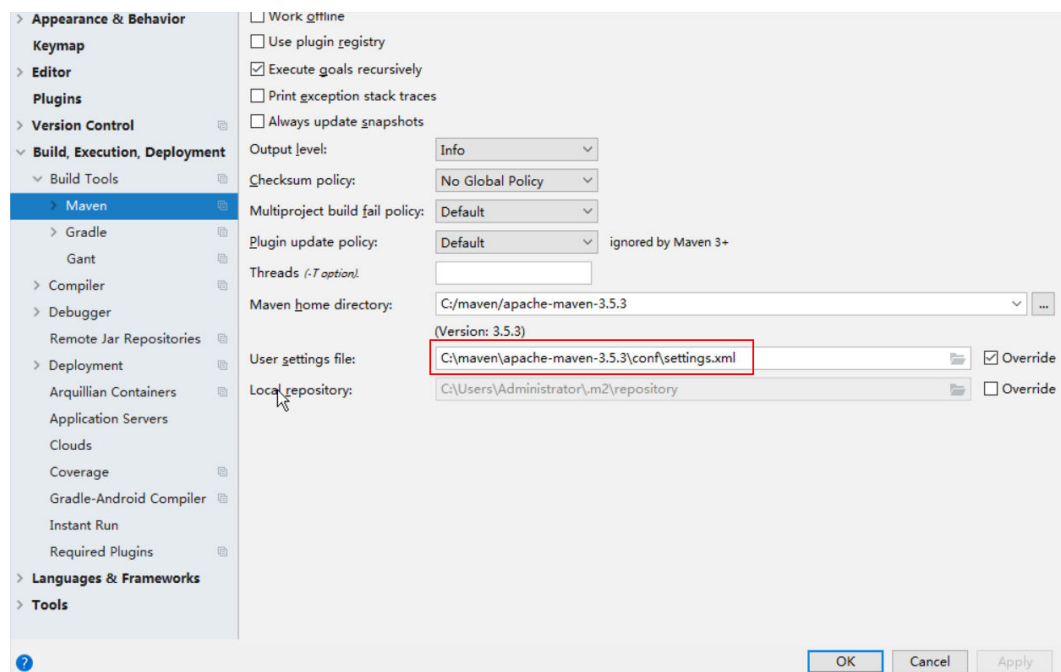
Figure 24-3 Setting the IntelliJ IDEA coding format



**Step 5** Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).

On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is `<Local Maven installation directory>\conf\settings.xml`.

Figure 24-4 Directory for storing the settings.xml file



----End

## References

The dependency packages mapped to the sample projects of MapReduce are as follows:

- MapReduce statistics sample project  
No additional jars.
- MapReduce accessing multi-components sample project

### NOTE

- If you want to use the multi-component accessing sample project after importing a sample project, ensure that the Hive and HBase services have been installed in the cluster.
- If you do not use the multi-components accessing sample project, you can ignore errors about the multi-components accessing sample project as long as the compilation of the statistics sample project is not affected. Otherwise, delete the files about the multi-components accessing sample project after importing the sample projects.

## 24.2.3 Creating a New Project (Optional)

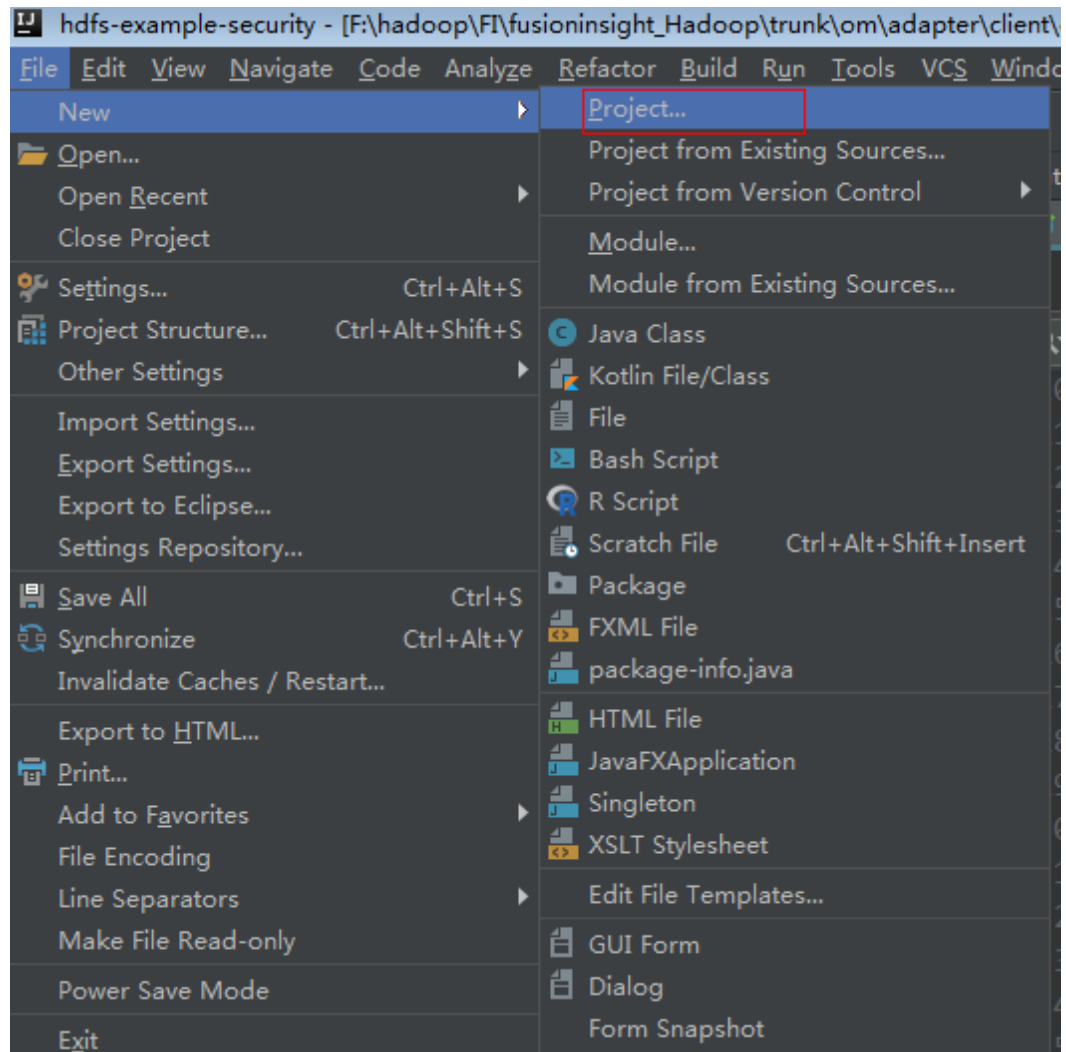
### Scenario

Apart from importing MapReduce sample projects, you can also create a new MapReduce project using IntelliJ IDEA.

### Procedure

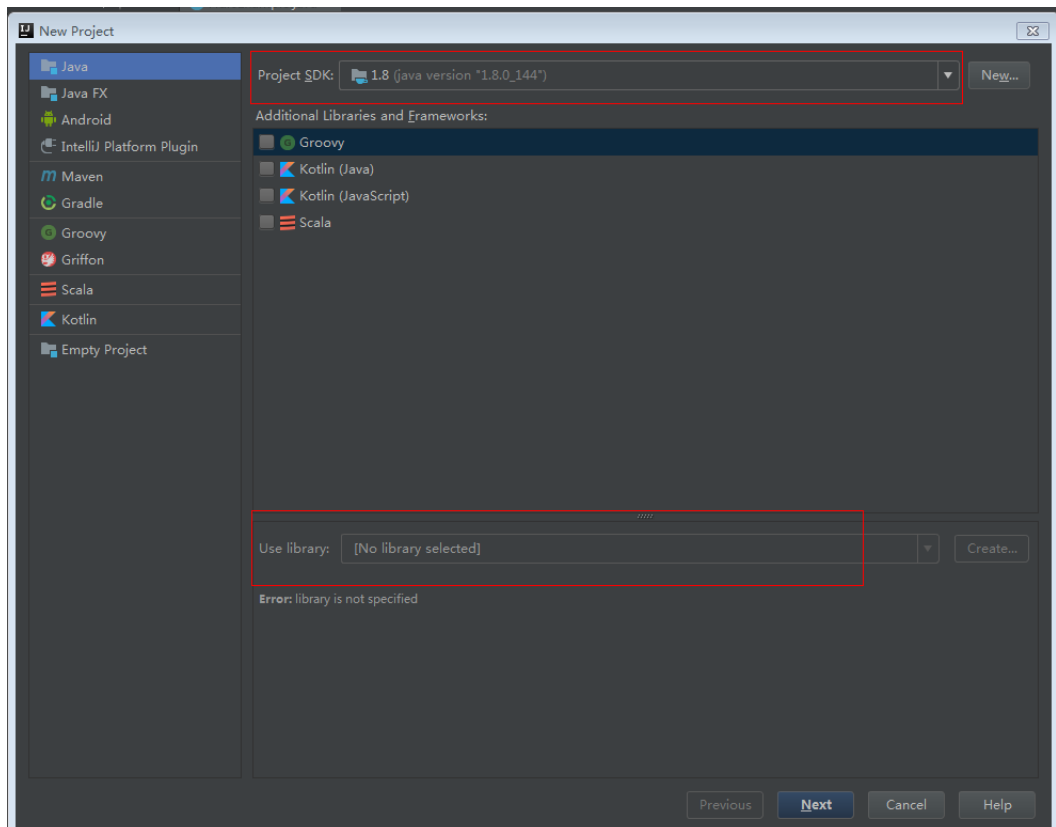
- Step 1** Open IntelliJ IDEA and choose **File > New > Project**, as shown in [Figure 24-5](#).

**Figure 24-5** Creating a project



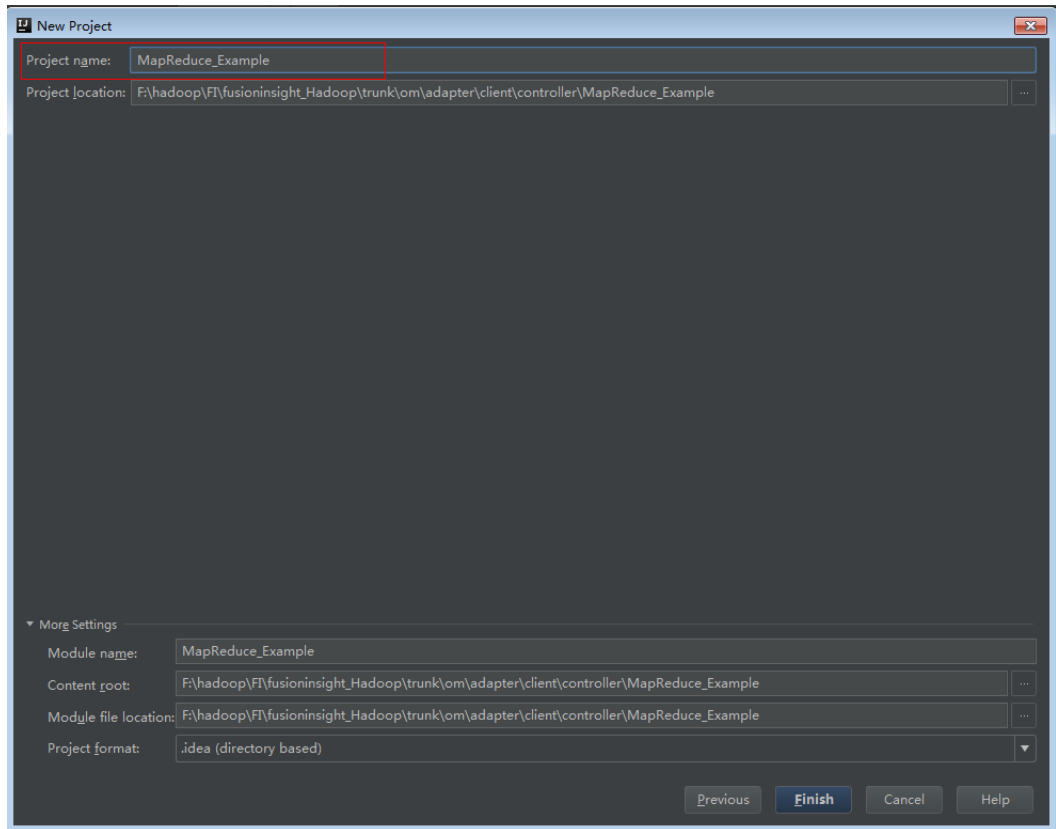
**Step 2** On the **New Project** page, select **Java**, and then configure the JDK and other Java libraries required by the project. As shown in the following figure. After the configuration is complete, click **Next**.

**Figure 24-6** Configuring sdk information required by the project



**Step 3** Enter the name of the new project in the dialog box. Click **Finish**.

Figure 24-7 Enter the project name



----End

## 24.2.4 Preparing the Authentication Mechanism

### Scenario

In a safe cluster environment, the communication among components cannot be a simple communication. Components must be authorized by each other before the communication to ensure the security of the communication.

When users are developing the MapReduce application, the MapReduce is required to interwork with Yarn and HDFS in certain scenarios. Therefore, security authentication code must be written into the MapReduce application to ensure that the MapReduce application can run properly.

Two security authentication methods are described as follows:

- Authentication by running command lines:  
Before submitting the MapReduce application for running, run the following command in the MapReduce client to obtain authentication:  
***kinit component service user***
- Authentication by adding code:  
Authenticate by obtaining principal and keytab files of the client.

## MapReduce Security Authentication Code

The security authentication is completed by calling the LoginUtil class.

In the code of the **FemaleInfoCollector** class in the **com.huawei.bigdata.mapreduce.examples** package of the sample project **MapReduce**, `test@<system domain name>`, `user.keytab`, and `krb5.conf` are examples. In practice, contact the admin to obtain your keytab and `krb5.conf` files and place them in the `conf` directory.):

```
public static final String PRINCIPAL= "test@<system domain name>";
public static final String KEYTAB =
FemaleInfoCollector.class.getClassLoader().getResource("user.keytab").getPath();
public static final String KRB =
FemaleInfoCollector.class.getClassLoader().getResource("krb5.conf").getPath();
...
// Security login
LoginUtil.login(PRINCIPAL, KEYTAB, KRB, conf);
```

## 24.3 Developing the Project

### 24.3.1 MapReduce Statistics Sample Project

#### 24.3.1.1 Typical Scenarios

##### Scenario

Develop a MapReduce application to perform the following operations on logs about dwell durations of netizens for shopping online:

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.
- The first column in the log file records names, the second column records genders, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

**log1.txt:** logs collected on Saturday

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

**log2.txt:** logs collected on Sunday

```
LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
FangBo,female,50
GuoYijun,male,5
CaiXuyu,female,50
```

```
Liyuan,male,20
CaiXuyu,female,50
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
FangBo,female,50
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

## Data Preparation

Save log files in the Hadoop distributed file system (HDFS).

1. Create text files `input_data1.txt` and `input_data2.txt` on the Linux operating system, and copy `log1.txt` to `input_data1.txt` and `log2.txt` to `input_data2.txt`.
2. Create `/tmp/input` on the HDFS, and run the following commands to upload `input_data1.txt` and `input_data2.txt` to `/tmp/input`:
  - a. On the Linux client, run `hdfs dfs -mkdir /tmp/input`.
  - b. On the Linux client, run `hdfs dfs -put local_file_path /tmp/input`.

## Development Idea

Collects the information about female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

- Read the source file data.
- Filter the data information about the time that female netizens spend online.
- Aggregate the total time that each female netizen spends online.
- Filter the information about female netizens who spend more than 2 hours online.

### 24.3.1.2 Example Code

#### Function

Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.

The operation is performed in three steps:

- Filter the online time of female netizens in original files using the `CollectionMapper` class inherited from the `Mapper` abstract class.
- Count the online time of each female netizen, and output information about female netizens who dwell online for more than 2 hours using the `CollectionReducer` class inherited from the `Reducer` abstract class.
- The main method creates a MapReduce job and submits the MapReduce job to the Hadoop cluster.

## Example Code

The following code snippets are used as an example. For complete code, see the `com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector` class.

Example 1: The `CollectionMapper` class defines the `map()` and `setup()` methods of the `Mapper` abstract class.

```
public static class CollectionMapper extends
 Mapper<Object, Text, Text, IntWritable> {

 // Delimiter.
 String delim;
 // Filter sex.
 String sexFilter;

 // Name.
 private Text nameInfo = new Text();

 // Output <key,value> must be serialized.
 private IntWritable timeInfo = new IntWritable(1);

 /**
 * Distributed computing
 *
 * @param key Object: location offset of the source file.
 * @param value Text: a row of characters in the source file.
 * @param context Context: output parameter.
 * @throws IOException , InterruptedException
 */
 public void map(Object key, Text value, Context context)
 throws IOException, InterruptedException
 {
 String line = value.toString();

 if (line.contains(sexFilter))
 {
 // A character string that has been read.
 String name = line.substring(0, line.indexOf(delim));
 nameInfo.set(name);
 // Obtain the dwell duration.
 String time = line.substring(line.lastIndexOf(delim) + 1,
 line.length());
 timeInfo.set(Integer.parseInt(time));

 // The Map task outputs a key-value pair.
 context.write(nameInfo, timeInfo);
 }
 }

 /**
 * map use to init.
 *
 * @param context Context.
 */
 public void setup(Context context) throws IOException,
 InterruptedException
 {
 // Obtain configuration information using Context.
 delim = context.getConfiguration().get("log.delimiter", ",");

 sexFilter = delim
 + context.getConfiguration()
 .get("log.sex.filter", "female") + delim;
 }
}
```



Example 2: The CollectionReducer class defines the reduce() method of the Reducer abstract class.

```
public static class CollectionReducer extends
 Reducer<Text, IntWritable, Text, IntWritable>
{
 // Statistical results.
 private IntWritable result = new IntWritable();

 // Total time threshold.
 private int timeThreshold;

 /**
 * @param key Text : key after Mapper.
 * @param values Iterable : all statistical results with the same key.
 * @param context Context
 * @throws IOException , InterruptedException
 */
 public void reduce(Text key, Iterable<IntWritable> values,
 Context context) throws IOException, InterruptedException
 {
 int sum = 0;
 for (IntWritable val : values) {
 sum += val.get();
 }

 // No results are output if the time is less than the threshold.
 if (sum < timeThreshold)
 {
 return;
 }
 result.set(sum);

 // In the output information, key indicates netizen information, and value indicates the total online
 time of the netizen.
 context.write(key, result);
 }

 /**
 * The setup() method is invoked for only once before the map() method or reduce() method.
 */
 @param context Context
 @throws IOException , InterruptedException
 public void setup(Context context) throws IOException,
 InterruptedException
 {
 // Context obtains configuration information.
 timeThreshold = context.getConfiguration().getInt(
 "log.time.threshold", 120);
 }
}
```

Example 3: Use the main() method to create a job, set parameters, and submit the job to the hadoop cluster.

```
public static void main(String[] args) throws Exception {
 // Initialize environment variables.
 Configuration conf = new Configuration();

 // Security login.
 LoginUtil.login(PRINCIPAL, KEYTAB, KRB, conf);

 // Obtain input parameters.
 String[] otherArgs = new GenericOptionsParser(conf, args)
 .getRemainingArgs();
 if (otherArgs.length != 2) {
```

```
System.err.println("Usage: collect female info <in> <out>");
System.exit(2);
}

// Initialize the job object.
@SuppressWarnings("deprecation")
Job job = new Job(conf, "Collect Female Info");
job.setJarByClass(FemaleInfoCollector.class);

// Set map and reduce classes to be executed, or specify the map and reduce classes using configuration
files.
job.setMapperClass(CollectionMapper.class);
job.setReducerClass(CollectionReducer.class);

// Set the Combiner class. The combiner class is not used by default. Classes same as the reduce class are
used.
// Exercise caution when using the Combiner class. You can specify it using configuration files.
job.setCombinerClass(CollectionReducer.class);

// Set the output type of the job.
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

// Submit the job to a remote environment for execution.
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Example 4: CollectionCombiner class combines the mapped data on the map side to reduce the amount of data transmitted from map to reduce.

```
/**
 * Combiner class
 */
public static class CollectionCombiner extends
Reducer<Text, IntWritable, Text, IntWritable> {

// Intermediate statistical results
private IntWritable intermediateResult = new IntWritable();

/**
 * @param key Text : key after Mapper
 * @param values Iterable : all results with the same key in this map task
 * @param context Context
 * @throws IOException , InterruptedException
 */
public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}

intermediateResult.set(sum);

// In the output information, key indicates netizen information,
// and value indicates the total online time of the netizen in this map task.
context.write(key, intermediateResult);
}
}
```

## 24.3.2 MapReduce Accessing Multi-Component Example Project

## 24.3.2.1 Instance

### Scenario

The sample project illustrates how to compile MapReduce jobs to visit multiple service components in HDFS, HBase, and Hive, helping users to understand key actions such as certificating and configuration loading.

The logic of the sample project is as follows:

The input data is HDFS text file and the input file is **log1.txt**.

```
YuanJing,male,10
GuoYijun,male,5
```

Map:

1. Obtain one row of the input data and extract the user name.
2. Query one piece of data from HBase.
3. Query one piece of data from Hive.
4. Combine the data queried from HBase and that from Hive as the output of Map as the output of Map.

Reduce:

1. Obtain the last piece of data from Map output.
2. Import the data to HBase.
3. Save the data to HDFS.

### Data Planning

1. Create an HDFS data file.
  - a. Create a text file named **data.txt** in the Linux-based HDFS and copy the content of **log1.txt** to **data.txt**.
  - b. Run the following commands to create a directory **/tmp/examples/multi-components/mapreduce/input/** and copy the **data.txt** to the directory:
    - i. ***hdfs dfs -mkdir -p /tmp/examples/multi-components/mapreduce/input/***
    - ii. ***hdfs dfs -put data.txt /tmp/examples/multi-components/mapreduce/input/***
2. Create a HBase table and insert data into it.
  - a. Run the **source bigdata\_env** command on a Linux-based HBase client and run the **hbase shell** command.
  - b. Run the **create 'table1', 'cf'** command in the HBase shell to create table1 with column family **cf**.
  - c. Run the **put 'table1', '1', 'cf:cid', '123'** command to insert data whose rowkey is **1**, column name is **cid**, and data value is **123**.
  - d. Run the **quit** command to exit the table.
3. Create a Hive table and load data to it.

- a. Run the **beeline** command on a Linux-based Hive client.
  - b. In the Hive beeline interaction window, run the **CREATE TABLE person(name STRING, gender STRING, stayTime INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ';' stored as textfile;** command to create data table **person** with three fields.
  - c. In the Hive beeline interaction window, run the **LOAD DATA INPATH '/tmp/examples/multi-components/mapreduce/input/' OVERWRITE INTO TABLE person;** command to load data files to the **person** table.
  - d. Run **!q** to exit the table.
4. The data of HDFS is cleared in the preceding step. Therefore, perform **1** again.

### 24.3.2.2 Example Code

#### Function

The functions of the sample project are as follows:

- Collect the name information from HDFS source files, query and combine data of HBase and Hive using the MultiComponentMapper class inherited from the Mapper abstract class.
- Obtain the last piece of mapped data and output to HBase and HDFS, using the MultiComponentMapper class inherited from the Reducer abstract class.
- The main function creates a MapReduce job and submits the MapReduce job to Hadoop clusters.

#### Example Code

For details about code, see the class `com.huawei.bigdata.mapreduce.examples.MultiComponentExamp`.

Example code of the map function used by MultiComponentMapper class to define the Mapper abstract class.

```
private static class MultiComponentMapper extends Mapper<Object, Text, Text, Text> {
 Configuration conf;

 @Override protected void map(Object key, Text value, Context context) throws IOException,
 InterruptedException {

 conf = context.getConfiguration();

 // for components that depend on Zookeeper, need provide the conf of jaas and krb5
 // Notice, no need to login again here, will use the credentials in main function
 String krb5 = "krb5.conf";
 String jaas = "jaas_mr.conf";
 // These files are uploaded at main function
 File jaasFile = new File(jaas);
 File krb5File = new File(krb5);
 System.setProperty("java.security.auth.login.config", jaasFile.getCanonicalPath());
 System.setProperty("java.security.krb5.conf", krb5File.getCanonicalPath());
 System.setProperty("zookeeper.sasl.client", "true");

 LOG.info("UGI : " + UserGroupInformation.getCurrentUser());

 String name = "";
 String line = value.toString();
 if (line.contains("male")) {
```

```
// A character string that has been read
name = line.substring(0, line.indexOf(","));
}
// 1. read from HBase
String hbaseData = readHBase();

// 2. read from Hive
String hiveData = readHive(name);

// The Map task outputs a key-value pair.
context.write(new Text(name), new Text("hbase:" + hbaseData + ", hive:" + hiveData));
}
```

Example code of the readHBase function.

```
private String readHBase() {
 String tableName = "table1";
 String columnFamily = "cf";
 String hbaseKey = "1";
 String hbaseValue;

 Configuration hbaseConfig = HBaseConfiguration.create(conf);
 org.apache.hadoop.hbase.client.Connection conn = null;
 try {
 // Create a HBase connection
 conn = ConnectionFactory.createConnection(hbaseConfig);
 // get table
 Table table = conn.getTable(TableName.valueOf(tableName));
 // Instantiate a Get object.
 Get get = new Get(hbaseKey.getBytes());
 // Submit a get request.
 Result result = table.get(get);
 hbaseValue = Bytes.toString(result.getValue(columnFamily.getBytes(), "cid".getBytes()));

 return hbaseValue;
 } catch (IOException e) {
 LOG.warn("Exception occur ", e);
 } finally {
 if (conn != null) {
 try {
 conn.close();
 } catch (Exception e1) {
 LOG.error("Failed to close the connection ", e1);
 }
 }
 }

 return "";
}
```

Example of the readHive function.

```
private String readHive(String name) throws IOException {
 //Load the configuration file
 Properties clientInfo = null;
 String userdir = System.getProperty("user.dir") + "/";
 InputStream fileInputStream = null;
 try {
 clientInfo = new Properties();
 String hiveclientProp = userdir + "hiveclient.properties";
 File propertiesFile = new File(hiveclientProp);
 fileInputStream = new FileInputStream(propertiesFile);
 clientInfo.load(fileInputStream);
 } catch (Exception e) {
 throw new IOException(e);
 } finally {
 if (fileInputStream != null) {
 fileInputStream.close();
 }
 }
}
```

```

 }
 }
 String zkQuorum = clientInfo.getProperty("zk.quorum");
 String zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
 String serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
 // Read this carefully:
 // MapReduce can only use Hive through JDBC.
 // Hive will submit another MapReduce Job to execute query.
 // So we run Hive in MapReduce is not recommended.
 final String driver = "org.apache.hive.jdbc.HiveDriver";

 String sql = "select name,sum(stayTime) as " + "stayTime from person where name = '" + name + "'
group by name";

 StringBuilder sBuilder = new StringBuilder("jdbc:hive2://").append(zkQuorum).append("/");
 // in map or reduce, use 'auth=delegationToken'
 sBuilder
 .append(";serviceDiscoveryMode=")

 .append(serviceDiscoveryMode)
 .append(";zooKeeperNamespace=")
 .append(zooKeeperNamespace)
 .append(";auth=delegationToken;");
 String url = sBuilder.toString();
 Connection connection = null;
 PreparedStatement statement = null;
 ResultSet resultSet = null;
 try {
 Class.forName(driver);
 connection = DriverManager.getConnection(url, "", "");
 statement = connection.prepareStatement(sql);
 resultSet = statement.executeQuery();

 if (resultSet.next()) {
 return resultSet.getString(1);
 }
 } catch (ClassNotFoundException e) {
 LOG.warn("Exception occur ", e);
 } catch (SQLException e) {
 LOG.warn("Exception occur ", e);
 } finally {
 if (null != resultSet) {
 try {
 resultSet.close();
 } catch (SQLException e) {
 // handle exception
 }
 }
 if (null != statement) {
 try {
 statement.close();
 } catch (SQLException e) {
 // handle exception
 }
 }
 if (null != connection) {
 try {
 connection.close();
 } catch (SQLException e) {
 // handle exception
 }
 }
 }

 return "";
}

```

Example code of the reduce function used by MultiComponentReducer class to define the Reducer abstract class.

```
private static class MultiComponentReducer extends Reducer<Text, Text, Text, Text> {
 Configuration conf;

 public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
 InterruptedException {
 conf = context.getConfiguration();

 // for components that depend on Zookeeper, need provide the conf of jaas and krb5
 // Notice, no need to login again here, will use the credentials in main function
 String krb5 = "krb5.conf";
 String jaas = "jaas_mr.conf";
 // These files are uploaded at main function
 File jaasFile = new File(jaas);
 File krb5File = new File(krb5);
 System.setProperty("java.security.auth.login.config", jaasFile.getCanonicalPath());
 System.setProperty("java.security.krb5.conf", krb5File.getCanonicalPath());
 System.setProperty("zookeeper.sasl.client", "true");

 Text finalValue = new Text("");
 // just pick the last value as the data to save
 for (Text value : values) {
 finalValue = value;
 }

 // write data to HBase
 writeHBase(key.toString(), finalValue.toString());

 // save result to HDFS
 context.write(key, finalValue);
 }
}
```

Example of the writeHBase function.

```
private void writeHBase(String rowKey, String data) {
 String tableName = "table1";
 String columnFamily = "cf";

 try {
 LOG.info("UGI read :" + UserGroupInformation.getCurrentUser());
 } catch (IOException e1) {
 // handler exception
 }

 Configuration hbaseConfig = HBaseConfiguration.create(conf);
 org.apache.hadoop.hbase.client.Connection conn = null;
 try {
 // Create a HBase connection
 conn = ConnectionFactory.createConnection(hbaseConfig);
 // get table
 Table table = conn.getTable(TableName.valueOf(tableName));

 // create a Put to HBase
 List<Put> list = new ArrayList<Put>();
 byte[] row = Bytes.toBytes("row" + rowKey);
 Put put = new Put(row);
 byte[] family = Bytes.toBytes(columnFamily);
 byte[] qualifier = Bytes.toBytes("value");
 byte[] value = Bytes.toBytes(data);
 put.addColumn(family, qualifier, value);
 list.add(put);
 // execute Put
 table.put(list);
 } catch (IOException e) {
 LOG.warn("Exception occur ", e);
 } finally {
 if (conn != null) {
 try {

```

```
 conn.close();
 } catch (Exception e1) {
 LOG.error("Failed to close the connection ", e1);
 }
}
}
```

Example code: the main() function creates a job, configures the dependency and permission, and submits the job to Hadoop clusters.

```
public static void main(String[] args) throws Exception {
 //Load the hiveclient.properties configuration file
 Properties clientInfo = null;
 try {
 clientInfo = new Properties();

clientInfo.load(MultiComponentExample.class.getClassLoader().getResourceAsStream("hiveclient.properties")
);
 } catch (Exception e) {
 throw new IOException(e);
 } finally {

 }
 String zkQuorum = clientInfo.getProperty("zk.quorum");
 String zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
 String serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
 String principal = clientInfo.getProperty("principal");
 String auth = clientInfo.getProperty("auth");
 String sasl_qop = clientInfo.getProperty("sasL.qop");
 String hbaseKeytab =
MultiComponentExample.class.getClassLoader().getResource("user.keytab").getPath();
 String hbaseJaas =
MultiComponentExample.class.getClassLoader().getResource("jaas_mr.conf").getPath();
 String hiveClientProperties =
MultiComponentExample.class.getClassLoader().getResource("hiveclient.properties").getPath();
 // a list of files, separated by comma
 String files = "file://" + KEYTAB + "," + "file://" + KRB + "," + "file://" + JAAS;
 files = files + "," + "file://" + hbaseKeytab;
 files = files + "," + "file://" + hbaseJaas;
 files = files + "," + "file://" + hiveClientProperties;
 // this setting will ask Job upload these files to HDFS
 config.set("tmpfiles", files);

 // clean control files before job submit
 MultiComponentExample.cleanupBeforeRun();

 // Security login
 LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, PRINCIPAL, hbaseKeytab);
 LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);
 LoginUtil.login(PRINCIPAL, KEYTAB, KRB, config);

 // find dependency jars for hive
 Class hiveDriverClass = Class.forName("org.apache.hive.jdbc.HiveDriver");
 Class thriftClass = Class.forName("org.apache.thrift.TException");
 Class serviceThriftCLIClass = Class.forName("org.apache.hive.service.rpc.thrift.TCLIService");
 Class hiveConfClass = Class.forName("org.apache.hadoop.hive.conf.HiveConf");
 Class hiveTransClass = Class.forName("org.apache.thrift.transport.HiveTSaslServerTransport");
 Class hiveMetaClass = Class.forName("org.apache.hadoop.hive.metastore.api.MetaException");
 Class hiveShimClass =
Class.forName("org.apache.hadoop.hive.metastore.security.HadoopThriftAuthBridge23");
 Class thriftCLIClass = Class.forName("org.apache.hive.service.cli.thrift.ThriftCLIService");

 // add dependency jars to Job
 JarFinderUtil.addDependencyJars(config, hiveDriverClass, serviceThriftCLIClass, thriftCLIClass, thriftClass,
 hiveConfClass, hiveTransClass, hiveMetaClass, hiveShimClass);

 // add hive config file
 config.addResource("hive-site.xml");
}
```



```
// add hbase config file
Configuration conf = HBaseConfiguration.create(conf);

// Initialize the job object.
Job job = Job.getInstance(conf);
job.setJarByClass(MultiComponentExample.class);

// set mapper&reducer class
job.setMapperClass(MultiComponentMapper.class);
job.setReducerClass(MultiComponentReducer.class);

//set job input&output
FileInputFormat.addInputPath(job, new Path(baseDir, INPUT_DIR_NAME + File.separator + "data.txt"));
FileOutputFormat.setOutputPath(job, new Path(baseDir, OUTPUT_DIR_NAME));

// Set the output type of the job.
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);

// HBase use this utility class to add dependency jars of hbase to MR job
TableMapReduceUtil.addDependencyJars(job);

// this is mandatory when access to security HBase cluster
// HBase add security credentials to Job, and will use in map and reduce
TableMapReduceUtil.initCredentials(job);

// create Hive security credentials

StringBuilder sBuilder = new StringBuilder("jdbc:hive2://").append(zkQuorum).append("/");
sBuilder.append(";serviceDiscoveryMode=").append(serviceDiscoveryMode).append(";zooKeeperNamespace=")
.append(zooKeeperNamespace)
.append(";sasL.qop=")
.append(sasl_qop)
.append(";auth=")
.append(auth)
.append(";principal=")
.append(principal)
.append(";");
String url = sBuilder.toString();
Connection connection = DriverManager.getConnection(url, "", "");
String tokenStr = ((HiveConnection) connection)
.getDelegationToken(UserGroupInformation.getCurrentUser().getShortUserName(), PRINCIPAL);
connection.close();
Token<DelegationTokenIdentifier> hive2Token = new Token<DelegationTokenIdentifier>();
hive2Token.decodeFromUrlString(tokenStr);
// add Hive security credentials to Job
job.getCredentials().addToken(new Text("hive.server2.delegation.token"), hive2Token);
job.getCredentials().addToken(new Text(HiveAuthConstants.HS2_CLIENT_TOKEN), hive2Token);

// Submit the job to a remote environment for execution.
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

#### NOTE

Replace all the zkQuorum objects with the actual information about the ZooKeeper cluster nodes.

## 24.4 Commissioning the Application

## 24.4.1 Commissioning the Application in the Windows Environment

### 24.4.1.1 Compiling and Running the Application

#### Scenario

After the code development is complete, you can run an application in the Windows development environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

#### NOTE

- If IBM JDK is used in the Windows environment, the application cannot be run in the Windows environment.
- Do not restart HDFS service while MapReduce application is in running status, otherwise the application will fail.

#### Running MapReduce Statistics Sample Project

**Step 1** Ensure that all JAR packages on which the sample project depends have been obtained.

**Step 2** In the IntelliJ IDEA development environment, select the LocalRunner.java project.

Right-click the project and choose **Run > LocalRunner.main()** from the shortcut menu to run the project. Click to run the related application project.

----End

#### Running MapReduce Accessing Multi-Component Example Project

**Step 1** Save the **user.keytab**, **hive-site.xml**, **hbase-site.xml**, and **hiveclient.properties** files to the **conf** directory of the project.

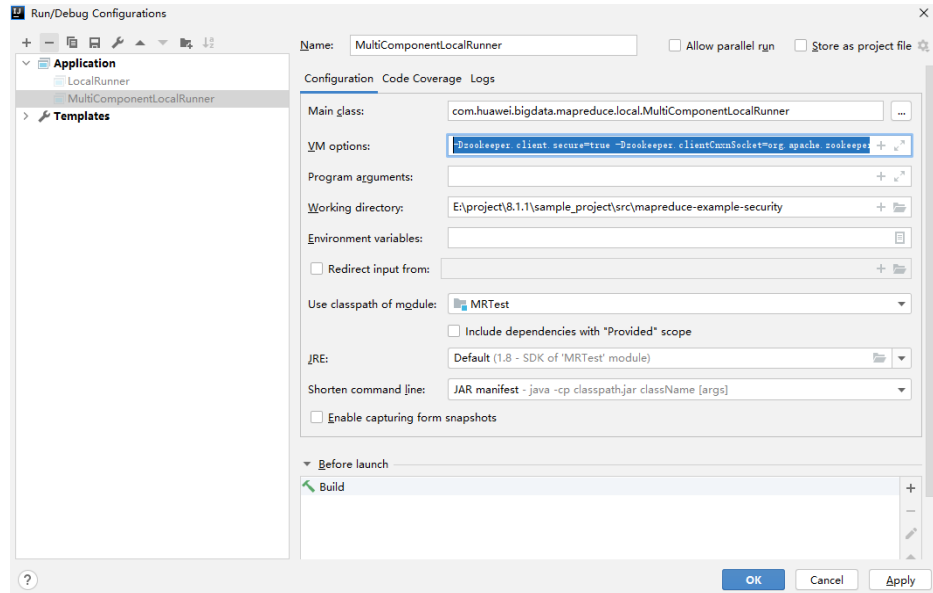
Create the **jaas\_mr.conf** file in the **conf** directory and add the following content (**test** is the user corresponding to **user.keytab**):

```
Client {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 keyTab="user.keytab"
 principal="test@<system domain name>"
 useTicketCache=false
 storeKey=true
 debug=true;
};
```

**Step 2** Ensure that all Hive and HBase JAR packages on which the sample project depends have been obtained.

**Step 3** In the IntelliJ IDEA development environment, click **MultiComponentLocalRunner.java** to run the application project. You can also right-click the project and choose **Run MultiComponentLocalRunner.main()** from the shortcut menu to run the project.

If ZooKeeper SSL is enabled for the cluster, add the -  
**Dzookeeper.client.secure=true** -  
**Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty**  
parameter to the position shown in the following figure before running this  
example.



----End

### 24.4.1.2 Checking the Commissioning Result

#### Scenario

After a MapReduce application is run, you can check the running result through one of the following methods:

- Viewing the program running status in IntelliJ IDEA.
- Viewing MapReduce logs.
- Logging in to the MapReduce WebUI.
- Logging in to the Yarn WebUI.

#### NOTE

You must have permission to access WebUI. If not, contact the admin to obtain an account and password.

#### Procedure

- **Check the running result by obtaining the MapReduce log.**

As shown below, the console outputs the application running result.

```
1848 [main] INFO org.apache.hadoop.security.UserGroupInformation - Login successful for user
admin@<system domain name> using keytab file
Login success!!!!!!!!!!!!!!
7093 [main] INFO org.apache.hadoop.hdfs.PeerCache - SocketCache disabled.
9614 [main] INFO org.apache.hadoop.hdfs.DFSClient - Created HDFS_DELEGATION_TOKEN token 45
for admin on ha-hdfs:hacluster
9709 [main] INFO org.apache.hadoop.mapreduce.security.TokenCache - Got dt for hdfs://hacluster;
```

```
Kind: HDFS_DELEGATION_TOKEN, Service: ha-hdfs:hacluster, Ident:
(HDFS_DELEGATION_TOKEN token 45 for admin)
10914 [main] INFO org.apache.hadoop.yarn.client.ConfiguredRMFailoverProxyProvider - Failing over
to 53
12136 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to
process : 2
12731 [main] INFO org.apache.hadoop.mapreduce.JobSubmitter - number of splits:2
13405 [main] INFO org.apache.hadoop.mapreduce.JobSubmitter - Submitting tokens for job:
job_1456738266914_0006
13405 [main] INFO org.apache.hadoop.mapreduce.JobSubmitter - Kind: HDFS_DELEGATION_TOKEN,
Service: ha-hdfs:hacluster, Ident: (HDFS_DELEGATION_TOKEN token 45 for admin)
16019 [main] INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl - Application submission is
not finished,
submitted application application_1456738266914_0006 is still in NEW
16975 [main] INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl - Submitted application
application_1456738266914_0006
17069 [main] INFO org.apache.hadoop.mapreduce.Job - The url to track the job: https://linux2:8090/
proxy/application_1456738266914_0006/
17086 [main] INFO org.apache.hadoop.mapreduce.Job - Running job: job_1456738266914_0006
29811 [main] INFO org.apache.hadoop.mapreduce.Job - Job job_1456738266914_0006 running in
uber mode : false
29811 [main] INFO org.apache.hadoop.mapreduce.Job - map 0% reduce 0%
41492 [main] INFO org.apache.hadoop.mapreduce.Job - map 100% reduce 0%
53161 [main] INFO org.apache.hadoop.mapreduce.Job - map 100% reduce 100%
53265 [main] INFO org.apache.hadoop.mapreduce.Job - Job job_1456738266914_0006 completed
successfully
53393 [main] INFO org.apache.hadoop.mapreduce.Job - Counters: 50
```

**NOTE**

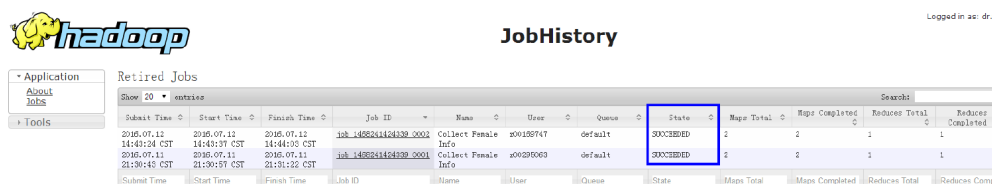
In the Windows environment, the following exception occurs but does not affect services.

java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.

- **Check the running result by using MapReduce WebUI.**

Log in to FusionInsight Manager as a user who has the permission to view tasks and choose **Cluster** > *Name of the desired cluster* > **Services** > **Mapreduce** > **JobHistoryServer**. On the web page that is displayed, view the task execution status.

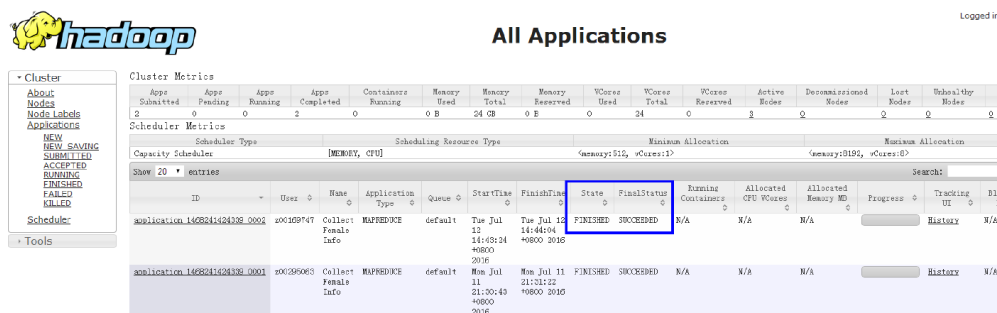
**Figure 24-8** JobHistory Web UI



- **Check the running result by using YARN WebUI.**

Log in to FusionInsight Manager as a user who has the permission to view tasks and choose **Cluster** > *Name of the desired cluster* > **Services** > **Yarn** > **ResourceManager (Active)**. On the web page that is displayed, view the task execution status.

Figure 24-9 ResourceManager Web UI



- **View MapReduce logs to learn application running conditions.**  
MapReduce logs offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 24.4.2 Commissioning an Application in the Linux Environment

### 24.4.2.1 Compiling and Running the Application

#### Scenario

After the code development is complete, you can run an application in the Linux development environment.

#### Prerequisites

You have installed the Yarn client.

#### Procedure

- Step 1** Go to the local root directory of the project and run the following command in Windows cmd to compress the package:

```
mvn -s "{maven_setting_path}" clean package
```

#### NOTE

- In the preceding command, {maven\_setting\_path} is the path of the setting.xml file of the local maven.
- After the package is successfully packed, obtain the JAR package, for example, *MRTTest-XXX.jar*, from the **target** subdirectory in the root directory of the project. The name of the JAR package varies according to the actual package.

- Step 2** Upload **MRTTest-XXX.jar** to the Linux client, such as **/opt/client/conf**, the same directory where the configuration files are located in.

- Step 3** Run the sample project in the Linux environment.

- Run the following command for MapReduce statistics sample project to configure parameters and submit jobs:

**yarn jar MRTest-XXX.jar**  
**com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector**  
*<inputPath> <outputPath>*

*<inputPath>* indicates the input path and *<outputPath>* indicates the output path in HDFS.

 NOTE

- Before running the **yarn jar MRTest-XXX.jar com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector <inputPath> <outputPath>** command, upload the **log1.txt** and **log2.txt** files to the *<inputPath>* directory in HDFS.
  - Before running the **yarn jar MRTest-XXX.jar com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector <inputPath> <outputPath>** command, ensure that the *<outputPath>* directory is deleted. Otherwise, an error will occur.
  - Do not restart the HDFS service during the running of MapReduce tasks. Otherwise, the tasks may fail.
- In MapReduce accessing multi-components sample project, perform the following operations:
    - a. Obtain the **user.keytab**, **krb5.conf**, **hbase-site.xml**, **hiveclient.properties**, and **hive-site.xml** configuration files, create a folder for example **/opt/client/conf**, and save the configurations files.

 NOTE

The account permission files **user.keytab** and **krb5.conf** are acquired from the administrator. The account permission file **hbase-site.xml** is acquired from the HBase client, **hiveclient.properties** and **hive-site.xml** are acquired from the Hive client.

- b. Create the **jaas\_mr.conf** file in the created folder. The content of the **jaas\_mr.conf** file is as follows:

```
Client {
com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true
keyTab="user.keytab"
principal="test@<system domain name>"
useTicketCache=false
storeKey=true
debug=true;
};
```

 NOTE

The **test@<system domain name>** in the preceding file content is an example of user. Modify it as required.

- c. Add the classpath required for sample projects in the Linux environment, Example of classpath:  
**export YARN\_USER\_CLASSPATH=/opt/client/conf:/opt/client/HBase/hbase/lib/\*:/opt/client/HBase/hbase/lib/client-facing-thirdparty/\*:/opt/client/Hive/Beeline/lib/\***
- d. Submit MapReduce jobs. Run the following command to run the sample project:  
**yarn jar MRTest-XXX.jar**  
**com.huawei.bigdata.mapreduce.examples.MultiComponentExample**

----End

## 24.4.2.2 Checking the Commissioning Result

### Scenario

After a MapReduce application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Logging in to the MapReduce WebUI
- Logging in to the Yarn WebUI
- Viewing MapReduce logs

#### NOTE

You must have permission to access WebUI. If not, contact the admin to obtain an account and password.

### Procedure

- **Check the running result by using MapReduce WebUI.**

Log in to FusionInsight Manager as a user who has the permission to view tasks and choose **Cluster > Name of the desired cluster > Services > Mapreduce > JobHistoryServer**. On the web page that is displayed, view the task execution status.

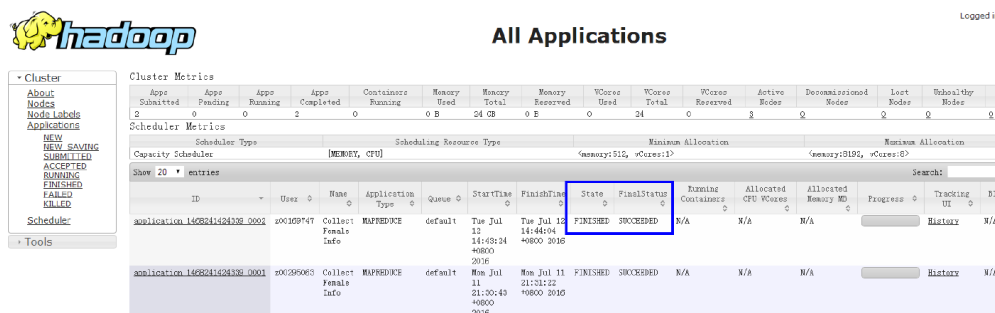
Figure 24-10 JobHistory Web UI



- **Check the running result by using YARN WebUI.**

Log in to FusionInsight Manager as a user who has the permission to view tasks and choose **Cluster > Name of the desired cluster > Services > Yarn > ResourceManager(Active)**. On the web page that is displayed, view the task execution status.

Figure 24-11 ResourceManager Web UI



- **Check the running result of the MapReduce application.**

- After running the **yarn jar MRTest-XXX.jar** command in the Linux environment, you can check the running status of the application by the returned information about the command.

```
linux1:/opt # yarn jar MRTest-XXX.jar /user/mapred/example/input/ /output6
16/02/24 15:45:40 INFO security.UserGroupInformation: Login successful for user
admin@<system domain name> using keytab file user.keytab
Login success!!!!!!!!!!!!!!
16/02/24 15:45:40 INFO hdfs.PeerCache: SocketCache disabled.
16/02/24 15:45:41 INFO hdfs.DFSClient: Created HDFS_DELEGATION_TOKEN token 28 for
admin on ha-hdfs:hacluster
16/02/24 15:45:41 INFO security.TokenCache: Got dt for hdfs://hacluster; Kind:
HDFS_DELEGATION_TOKEN, Service: ha-hdfs:hacluster,
Ident: (HDFS_DELEGATION_TOKEN token 28 for admin)
16/02/24 15:45:41 INFO input.FileInputFormat: Total input files to process : 2
16/02/24 15:45:41 INFO mapreduce.JobSubmitter: number of splits:2
16/02/24 15:45:42 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1455853029114_0027
16/02/24 15:45:42 INFO mapreduce.JobSubmitter: Kind: HDFS_DELEGATION_TOKEN, Service: ha-
hdfs:hacluster,
Ident: (HDFS_DELEGATION_TOKEN token 28 for admin)
16/02/24 15:45:42 INFO impl.YarnClientImpl: Submitted application
application_1455853029114_0027
16/02/24 15:45:42 INFO mapreduce.Job: The url to track the job: https://linux1:8090/proxy/
application_1455853029114_0027/
16/02/24 15:45:42 INFO mapreduce.Job: Running job: job_1455853029114_0027
16/02/24 15:45:50 INFO mapreduce.Job: Job job_1455853029114_0027 running in uber mode :
false
16/02/24 15:45:50 INFO mapreduce.Job: map 0% reduce 0%
16/02/24 15:45:56 INFO mapreduce.Job: map 100% reduce 0%
16/02/24 15:46:03 INFO mapreduce.Job: map 100% reduce 100%
16/02/24 15:46:03 INFO mapreduce.Job: Job job_1455853029114_0027 completed successfully
16/02/24 15:46:03 INFO mapreduce.Job: Counters: 49
```

- In the Linux environment, run the **yarn application -status <ApplicationID>** command to check the running result of the current application. Example:

```
linux1:/opt # yarn application -status application_1455853029114_0027
Application Report :
 Application-Id : application_1455853029114_0027
 Application-Name : Collect Female Info
 Application-Type : MAPREDUCE
 User : admin
 Queue : default
 Start-Time : 1456299942302
 Finish-Time : 1456299962343
 Progress : 100%
 State : FINISHED
 Final-State : SUCCEEDED
 Tracking-URL : https://linux1:26014/jobhistory/job/job_1455853029114_0027
 RPC Port : 27100
 AM Host : SZV1000044726
 Aggregate Resource Allocation : 114106 MB-seconds, 42 vcore-seconds
 Log Aggregation Status : SUCCEEDED
 Diagnostics : Application finished execution.
 Application Node Label Expression : <Not set>
 AM container Node Label Expression : <DEFAULT_PARTITION>
```

- **View MapReduce logs to learn application running conditions.**

MapReduce logs offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 24.5 More Information



## 24.5.1 Common APIs

### 24.5.1.1 Java API

Directly consult official website for detailed API of MapReduce:

<http://hadoop.apache.org/docs/r3.1.1/api/index.html>

#### Common Interfaces

Common classes in MapReduce are as follows:

- org.apache.hadoop.mapreduce.Job: an interface for users to submit MR jobs and used to set job parameters, submit jobs, control job executions, and query job status.
- org.apache.hadoop.mapred.JobConf: configuration class of a MapReduce job and major configuration interface for users to submit jobs to Hadoop.

**Table 24-3** Common interfaces of org.apache.hadoop.mapreduce.Job

| Interface                                                        | Description                                                                                                                                                                                                                                                                                                                                                                             |
|------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Job(Configuration conf, String jobName), Job(Configuration conf) | Creates a MapReduce client for configuring job attributes and submitting the job.                                                                                                                                                                                                                                                                                                       |
| setMapperClass(Class<extends Mapper> cls)                        | A core interface used to specify the Mapper class of a MapReduce job. The Mapper class is empty by default. You can also configure <b>mapreduce.job.map.class</b> in <b>mapred-site.xml</b> .                                                                                                                                                                                           |
| setReducerClass(Class<extends Reducer> cls)                      | A core interface used to specify the Reducer class of a MapReduce job. The Reducer class is empty by default. You can also configure <b>mapreduce.job.reduce.class</b> in <b>mapred-site.xml</b> .                                                                                                                                                                                      |
| setCombinerClass(Class<extends Reducer> cls)                     | Specifies the Combiner class of a MapReduce job. The Combiner class is empty by default. You can also configure <b>mapreduce.job.combine.class</b> in <b>mapred-site.xml</b> . The Combiner class can be used only when the input and output key and value types of the reduce task are the same.                                                                                       |
| setInputFormatClass(Class<extends InputFormat> cls)              | A core interface used to specify the InputFormat class of a MapReduce job. The default InputFormat class is <b>TextInputFormat</b> . You can also configure <b>mapreduce.job.inputformat.class</b> in <b>mapred-site.xml</b> . This interface can be used to specify the InputFormat class for processing data in different formats, reading data, and splitting data into data blocks. |

| Interface                                                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| setJarByClass(Class< > cls)                                | A core interface used to specify the local location of the JAR package of a class. Java locates the JAR package based on the class file and uploads the JAR package to the Hadoop distributed file system (HDFS).                                                                                                                                                                                                                                                                                 |
| setJar(String jar)                                         | Specifies the local location of the JAR package of a class. You can directly set the location of a JAR package and upload the JAR package to the HDFS. Use either setJar(String jar) or setJarByClass(Class< > cls). You can also configure <b>mapreduce.job.jar</b> in <b>mapred-site.xml</b> .                                                                                                                                                                                                  |
| setOutputFormatClass(Class<extends OutputFormat> theClass) | A core interface used to specify the OutputFormat class of a MapReduce job. The default OutputFormat class is <b>TextOutputFormat</b> . You can also configure <b>mapred.output.format.class</b> in <b>mapred-site.xml</b> . In the default <b>TextOutputFormat</b> , each key and value are recorded in text. <b>OutputFormat</b> is not specified usually.                                                                                                                                      |
| setOutputKeyClass(Class< > theClass)                       | A core interface used to specify the output key type of a MapReduce job. You can also configure <b>mapreduce.job.output.key.class</b> in <b>mapred-site.xml</b> .                                                                                                                                                                                                                                                                                                                                 |
| setOutputValueClass(Class< > theClass)                     | A core interface used to specify the output value type of a MapReduce job. You can also configure <b>mapreduce.job.output.value.class</b> in <b>mapred-site.xml</b> .                                                                                                                                                                                                                                                                                                                             |
| setPartitionerClass(Class<extends Partitioner> theClass)   | Specifies the Partitioner class of a MapReduce job. You can also configure <b>mapred.partitioner.class</b> in <b>mapred-site.xml</b> . This method is used to allocate Map output results to reduce classes. <b>HashPartitioner</b> is used by default, which evenly allocates the key-value pairs of a map task. For example, in HBase applications, different key-value pairs belong to different regions. In this case, you must specify the Partitioner class to allocate map output results. |
| setSortComparatorClass(Class<extends RawComparator> cls)   | Specifies the compression class for output results of a map task. Compression is not implemented by default. You can also configure <b>mapreduce.map.output.compress</b> and <b>mapreduce.map.output.compress.codec</b> in <b>mapred-site.xml</b> . You can compress data for transmission when the map task outputs a large amount of data.                                                                                                                                                      |

| Interface                         | Description                                                                                                                                                                                                                                                                         |
|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| setPriority(JobPriority priority) | Specifies the priority of a MapReduce job. Five priorities can be set: <b>VERY_HIGH</b> , <b>HIGH</b> , <b>NORMAL</b> , <b>LOW</b> , and <b>VERY_LOW</b> . The default priority is <b>NORMAL</b> . You can also configure <b>mapreduce.job.priority</b> in <b>mapred-site.xml</b> . |

**Table 24-4** Common interfaces of org.apache.hadoop.mapred.JobConf

| Interface                      | Description                                                                                                                                                                                                                                                                                                                                     |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| setNumMapTasks(int n)          | A core interface used to specify the number of map tasks in a MapReduce job. You can also configure <b>mapreduce.job.maps</b> in <b>mapred-site.xml</b> .<br><b>NOTE</b><br>The InputFormat class controls the number of map tasks. Ensure that the InputFormat class supports setting the number of map tasks on the client.                   |
| setNumReduceTasks(int n)       | A core interface used to specify the number of reduce tasks in a MapReduce job. Only one reduce task is started by default. You can also configure <b>mapreduce.job.reduces</b> in <b>mapred-site.xml</b> . The number of reduce tasks is controlled by users. In most cases, the number of reduce tasks is one-fourth the number of map tasks. |
| setQueueName(String queueName) | Specifies the queue where a MapReduce job is submitted. The default queue is used by default. You can also <b>configure</b> <b>mapreduce.job.queueName</b> in <b>mapred-site.xml</b> .                                                                                                                                                          |

### 24.5.1.2 REST API

#### Function Description

Use the HTTP REST API to view more information about MapReduce tasks. Currently, the REST API of MapResuce can be used to query the status of completed tasks. For details about the API, see the official website:

<http://hadoop.apache.org/docs/r3.1.1/hadoop-mapreduce-client/hadoop-mapreduce-client-hs/HistoryServerRest.html>

## Preparing the Running Environment

1. Install the client, for example, to the `/opt/client` directory on the node. For details, see section "Installing a Client."
2. Go the client installation directory and run the following commands to configure the environment variables:

```
source bigdata_env
```

```
kinit service user
```

### NOTE

The validity duration of kinit authentication is 24 hours. If you run the sample again 24 hours later, you need to run the kinit command again.

3. HTTPS-based access is different from HTTP-based access. When you access MapReduce using HTTPS, you must ensure that the SSL protocol supported by the curl command is supported by the cluster because SSL security encryption is used. If the cluster does not support the SSL protocol, change the SSL protocol in the cluster. For example, if the cURL supports only the TLSv1 protocol, perform the following steps:

Log in to FusionInsight Manager, choose **Cluster** > *Name of the desired cluster* > **Services** > **Yarn** > **Configurations** > **All Configurations**, search for **hadoop.ssl.enabled.protocols** in the search box, and check whether the parameter value contains **TLSv1**. If the parameter value does not contain **TLSv1**, add **TLSv1** in the **hadoop.ssl.enabled.protocols** configuration item. Clear the value of **ssl.server.exclude.cipher.list**. Otherwise, you cannot access Yarn using HTTPS. Click **Save**, and click **More** > **Restart Service** to restart the service.

### NOTE

- The values of MapReduce configuration items **hadoop.ssl.enabled.protocols** and **ssl.server.exclude.cipher.list** directly reference the values of the corresponding configuration items in Yarn. Therefore, you need to change the values of the corresponding configuration items in Yarn and restart the Yarn and MapReduce services.
- TLSv1 has security vulnerabilities. Exercise caution when using it.

## Procedure

Obtain detailed information about tasks that have been completed on MapReduce.

- Commands for the operation:

```
curl -k -i --negotiate -u : "https://10.120.85.2:26014/ws/v1/history/mapreduce/jobs"
```

In the preceding command, **10.120.85.2** indicates the value of **JHS\_FLOAT\_IP** for MapReduce, and **26014** indicates the port ID of the JobHistoryServer node.

### NOTE

In RedHat 6.x and CentOS 6.x, a compatibility problem occurs when the curl command is used to access the JobHistoryServer. As a result, the correct result cannot be returned.

- You can view the status information about historical tasks, such as the task IDs, start time, end time, and task execution status.

- Execution result

```
{
 "jobs":{
 "job":[
 {
 "submitTime":1525693184360,
 "startTime":1525693194840,
 "finishTime":1525693215540,
 "id":"job_1525686535456_0001",
 "name":"QuasiMonteCarlo",
 "queue":"default",
 "user":"mapred",
 "state":"SUCCEEDED",
 "mapsTotal":1,
 "mapsCompleted":1,
 "reducesTotal":1,
 "reducesCompleted":1
 }
]
 }
}
```

- Result analysis:

Using this API, you can query the completed MapReduce tasks in the current cluster and obtain information listed in [Table 1](#).

**Table 24-5** Common information

| Parameter  | Description                    |
|------------|--------------------------------|
| submitTime | Time when a task is submitted  |
| startTime  | Start time                     |
| finishTime | End time                       |
| queue      | Task queue                     |
| user       | User who submits the task      |
| state      | Task state, success or failure |

## 24.5.2 FAQ

### 24.5.2.1 No Response from the Client When Submitting the MapReduce Application

#### Question

After the MapReduce task is submitted to the YARN server, the client is prompted with following information without response for a long time.

```
16/03/03 16:44:56 INFO hdfs.DFSClient: Created HDFS_DELEGATION_TOKEN token 44 for admin on ha-hdfs:hacluster
16/03/03 16:44:56 INFO security.TokenCache: Got dt for hdfs://hacluster; Kind: HDFS_DELEGATION_TOKEN, Service: ha-hdfs:hacluster, Ident: (HDFS_DELEGATION_TOKEN token 44 for admin)
16/03/03 16:44:56 INFO client.ConfiguredRMFailoverProxyProvider: Failing over to 53
16/03/03 16:44:57 INFO input.FileInputFormat: Total input files to process : 200
16/03/03 16:44:57 INFO mapreduce.JobSubmitter: number of splits:200
```

```
16/03/03 16:44:57 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1456738266914_0005
16/03/03 16:44:57 INFO mapreduce.JobSubmitter: Kind: HDFS_DELEGATION_TOKEN, Service: ha-
hdfs:hacluster, Ident: (HDFS_DELEGATION_TOKEN token 44 for admin)
16/03/03 16:44:57 INFO impl.YarnClientImpl: Submitted application application_1456738266914_0005
16/03/03 16:44:57 INFO mapreduce.Job: The url to track the job: https://linux2:8090/proxy/
application_1456738266914_0005/
16/03/03 16:44:57 INFO mapreduce.Job: Running job: job_1456738266914_0005
```

## Answer

For the above problem, ResourceManager provides the key diagnosis information about MapReduce operating status on the WebUI. For the MapReduce application that is submitted to the YARN, users can obtain the current application status and the reason to be in the status with the diagnosis information.

Procedures:

Log in to FusionInsight Manager, and select **Cluster > Name of the desired cluster > Services > Yarn > ResourceManager (Active)**. Enter the WebUI, click the submitted MapReduce application on the WebUI of ResourceManager (Active), check the diagnosis information on the WebUI, and take measures according to the diagnosis information.

MapReduce logs offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

### 24.5.2.2 When an Application Is Run, An Abnormality Occurs Due to Network Faults

#### Question

When an application is run in the Windows environment, the application fails to connect the cluster. While the application running in the Linux environment (the same network with the host that is installed with the MRS Cluster) works properly.

#### Answer

Kerberos authentication requires the UDP protocol. But the firewall disables the required UDP ports with special operations, resulting in that the network of the application running in the Windows environment fails to connect with the MRS Cluster. Reset the firewall, and open the UDP ports that are required, to ensure that the host running the application in the Windows environment can connect with the MRS.

### 24.5.2.3 How to Perform Remote Debugging During MapReduce Secondary Development?

#### Question

During the secondary development of MapReduce, how to perform remote debugging?

## Answer

MapReduce adopts Java remote debugging mechanism. Run Java remote debugging commands when starting the Map or Reduce tasks.

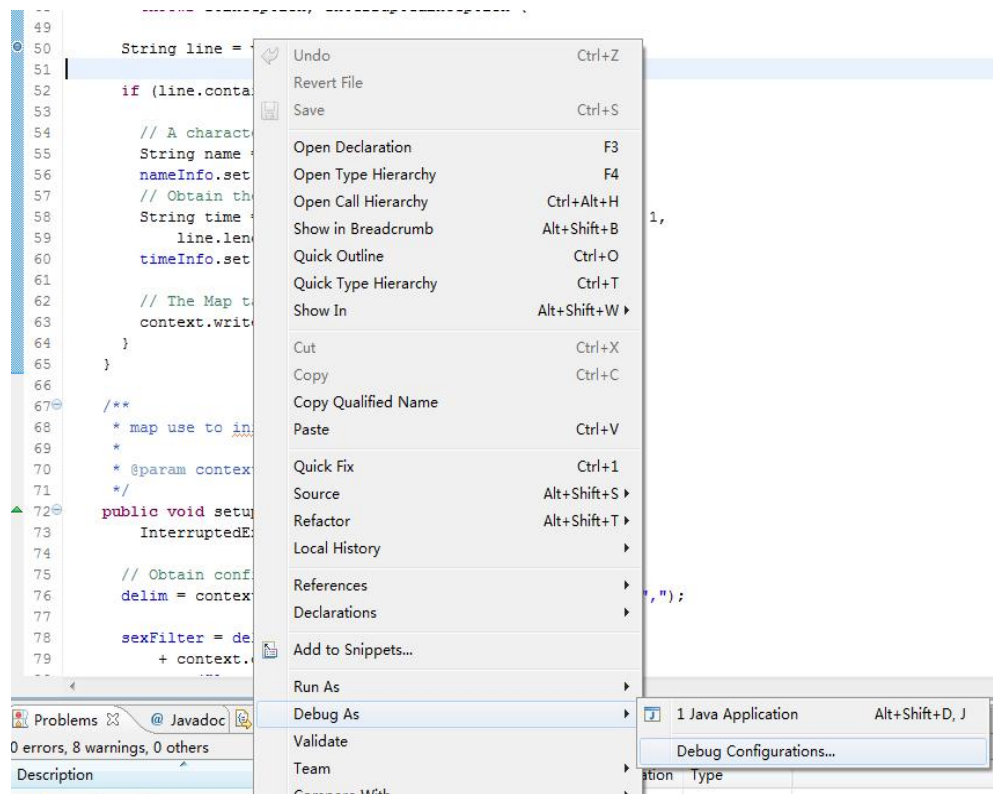
- Step 1** The `mapreduce.map.java.opts` and `mapreduce.reduce.java.opts` parameters specify the JVM startup parameters of Map and Reduce tasks respectively. In the `mapred-site.xml` configuration file on the client, add the command - `agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=8000` to the `mapreduce.map.java.opts` and `mapreduce.reduce.java.opts` parameters.
- Step 2** MapReduce is a distributed computing framework, and the node where Map or Reduce tasks are running are not fixed. It is advisable to keep only one NodeManager running in the cluster to ensure that the task is executed in the running NodeManager.
- Step 3** Submit MapReduce tasks on the client, then the Map or Reduce tasks will be suspended and listen to the port 8000 to wait for remote debugging.
- Step 4** In IDE, select the implementation class of MapReduce tasks and configure the remote debugging information to perform debugging.
1. Double-click the area in the blue box to configure or cancel the break point.

```

39 /**
40 * Distributed computing
41 *
42 * @param key Object : location offset of the source file
43 * @param value Text : a row of characters in the source file
44 * @param context Context : output parameter
45 * @throws IOException , InterruptedException
46 */
47 public void map(Object key, Text value, Context context)
48 throws IOException, InterruptedException {
49
50 String line = value.toString();
51
52 if (line.contains(sexFilter)) {
53
54 // A character string that has been read
55 String name = line.substring(0, line.indexOf(delim));
56 nameInfo.set(name);
57 // Obtain the dwell duration.
58 String time = line.substring(line.lastIndexOf(delim) + 1,
59 line.length());
60 timeInfo.set(Integer.parseInt(time));
61
62 // The Map task outputs a key-value pair.
63 context.write(nameInfo, timeInfo);
64 }
65 }
66

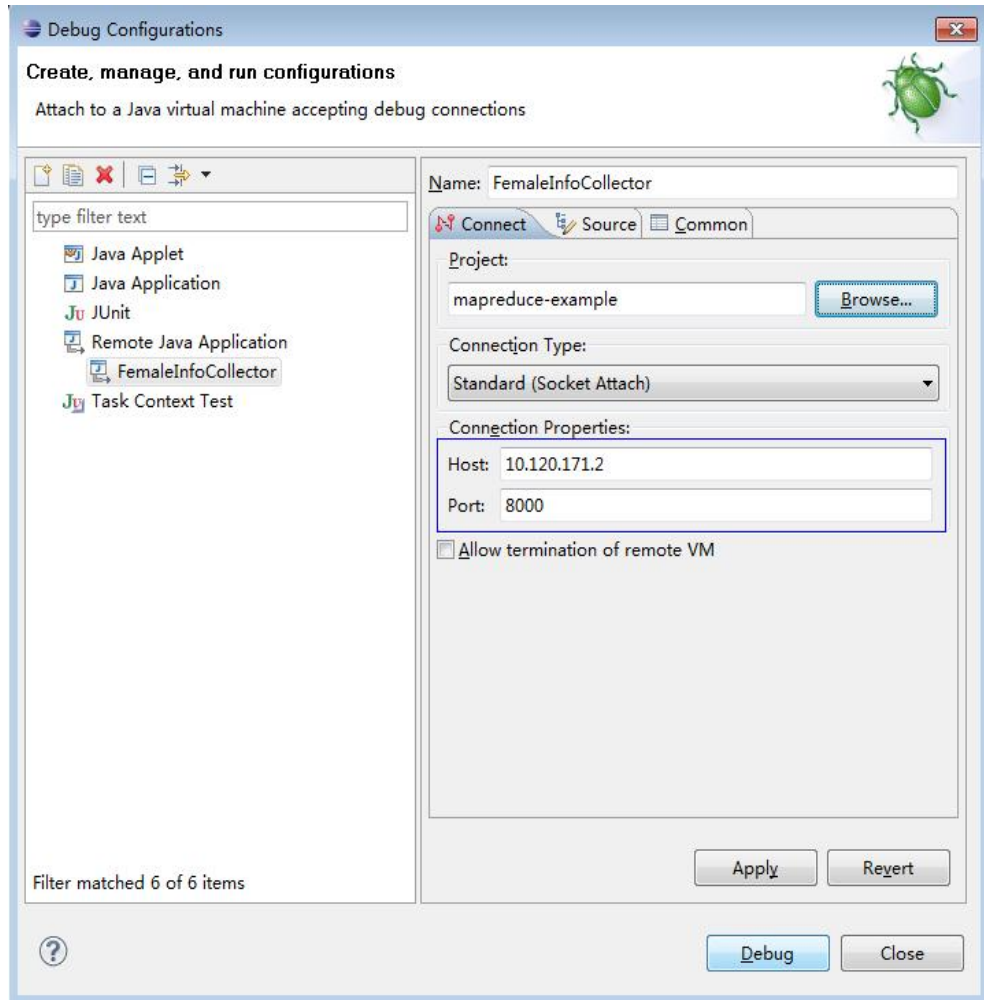
```

2. Right-click the break point and choose **Debug As->Debug Configurations...** from the shortcut menu.



3. On the displayed page, double-click **Remote Java Application** and configure the **Connection Properties** area. Set **Host** to the IP address of the NodeManager and **Port** to **8000**, and then click **Debug**.





----End

 **NOTE**

If you use IDE to submit MapReduce tasks, the IDE is the client. Modify the **mapred-site.xml** file of the secondary development project by referring to [Step 1](#).

# 25 MapReduce Development Guide (Normal Mode)

---

## 25.1 Overview

### 25.1.1 MapReduce Overview

#### MapReduce Introduction

Hadoop MapReduce is an easy-to-use parallel computing software framework. Applications developed based on MapReduce can run on large clusters consisting of thousands of servers and process data sets larger than 1 TB in fault tolerance (FT) mode.

A MapReduce job (application or job) splits an input data set into several data blocks which then are processed by Map tasks in parallel mode. The framework sorts output results of the Map task, sends the results to Reduce tasks, and returns a result to the client. Input and output information is stored in the Hadoop Distributed File System (HDFS). The framework schedules and monitors tasks and re-executes failed tasks.

MapReduce supports the following features:

- Large-scale parallel computing
- Large data set processing
- High FT and reliability
- Reasonable resource scheduling

### 25.1.2 Basic Concepts

#### Hadoop shell command

Basic hadoop shell commands include commands that are used to submit MapReduce jobs, kill MapReduce jobs, and perform operations on the HDFS.

## MapReduce InputFormat and OutputFormat

Based on the specified InputFormat, the MapReduce framework splits data sets, reads data, provides key-value pairs for Map tasks, and determines the number of Map tasks that are started in parallel mode. Based on the OutputFormat, the MapReduce framework outputs the generated key-value pairs to data in a specific format.

Map and Reduce tasks are running based on <key,value> pairs. In other words, the framework regards the input information of a job as a group of key-value pairs and outputs a group of key-value pairs. Two groups of key-value pairs may be of different types. For a single Map or Reduce task, key-value pairs are processed in single-thread serial mode.

The framework needs to perform serialized operations on key and value classes. Therefore, the classes must support the Writable interface. To facilitate sorting operations, key classes must support the WritableComparable interface.

The input and output types of a MapReduce job are as follows:

(input) <k1,v1> -> Map -> <k2,v2> -> Summary data -> <k2, List(v2)> -> Reduce -> <k3,v3> (output)

## Job Core

In normal cases, an application only needs to inherit Mapper and Reducer classes and rewrite map and reduce methods to implement service logic. The map and reduce methods constitute the core of jobs.

## MapReduce WebUI

Allows users to monitor running or historical MapReduce jobs, view logs, and implement fine-grained job development, configuration, and optimization.

## Reduce

A processing model function that merges all intermediate values associated with the same intermediate key.

## Shuffle

A process of outputting data from a Map task to a Reduce task.

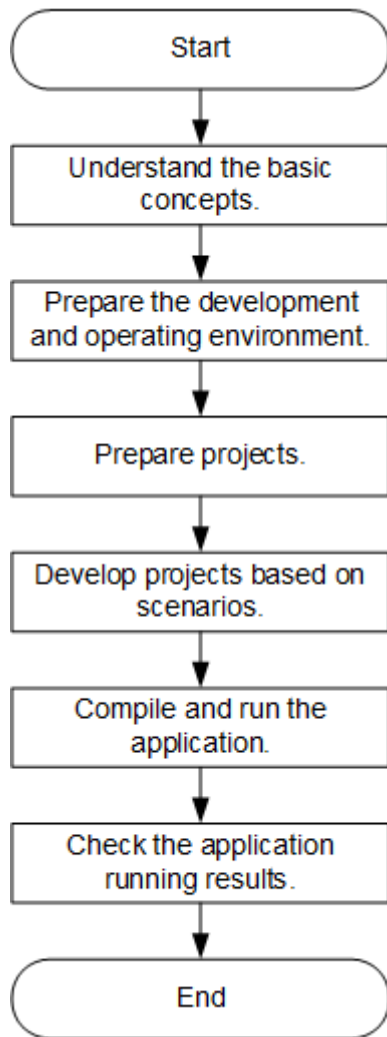
## Map

A method used to map a group of key-value pairs into a new group of key-value pairs.

## 25.1.3 Development Process

All stages of the development process are shown and described in [Figure 25-1](#) and [Table 25-1](#).

**Figure 25-1** MapReduce development process



**Table 25-1** Description of MapReduce development process

| Stage                                              | Description                                                                                                                                                                                                   | Reference                                                       |
|----------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|
| Understand the basic concepts.                     | Before the application development, the basic concepts of MapReduce are required to be understood.                                                                                                            | <a href="#">Basic Concepts</a>                                  |
| Prepare the development and operating environment. | Use IntelliJ IDEA and configure the development environment based on the reference.<br>The running environment of MapReduce is the MapReduce client. Install and configure the client based on the reference. | <a href="#">Preparing Development and Operating Environment</a> |

| Stage                                  | Description                                                                                                                                                              | Reference                                                                                                      |
|----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| Prepare projects                       | MapReduce provides sample projects in various scenarios. Sample projects can be imported for studying. Or you can create a new MapReduce project based on the reference. | <a href="#">Configuring and Importing Sample Projects</a><br><a href="#">Creating a New Project (Optional)</a> |
| Develop projects based on scenarios.   | Provide the example project. This helps users to learn about the programming interfaces of all Spark components quickly.                                                 | <a href="#">Developing the Project</a>                                                                         |
| Compile and run the application.       | Users compile the developed application and deliver it for running based on the reference.                                                                               | <a href="#">Commissioning the Application</a>                                                                  |
| Check the application running results. | Application running results are stored in the directory specified by users. Users can also check the running results through the UI.                                     | <a href="#">Commissioning the Application</a>                                                                  |

## 25.2 Environment Preparation

### 25.2.1 Preparing Development and Operating Environment

#### Preparing Development Environment

[Table 25-2](#) describes the environment required for application development.

**Table 25-2** Development environment

| Preparation Item | Description                                                                                                                                                                                                                                                                                                                     |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OS               | <ul style="list-style-type: none"> <li>Development environment: Windows OS. Windows 7 or later is supported.</li> <li>Operating environment: Windows OS or Linux OS<br/>If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li> </ul> |

| Preparation Item                             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IntelliJ IDEA installation and configuration | <p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li> <li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li> <li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li> <li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Maven installation                           | Basic configuration of the development environment for project management throughout the lifecycle of software development.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| JDK installation                             | <p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> <li>• For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> <li>- Oracle JDK versions: 1.8</li> <li>- Supported IBM JDK versions: 1.8.5.11</li> </ul> </li> <li>• For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> <li>- OpenJDK: 1.8.0_272</li> </ul> </li> </ul> <p><b>NOTE</b></p> <p>The server supports only TLS V1.2 or later to ensure security.</p> <p>By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls">https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls</a>.</p> |
| 7-zip                                        | <p>Used to decompress <b>.zip</b> and <b>.rar</b> packages.</p> <p>7-Zip 16.04 is supported.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host;

obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.

- a. [Log in to the FusionInsight Manager portal](#) and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is

**FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar file**. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar file** to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Yarn\config** and obtain the cluster configuration file. The configuration file will be imported to the configuration file directory (usually the **conf** folder) of the MapReduce sample project.
- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
  - The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client on the node. For example, the client installation directory is **/opt/client**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. [Log in to the FusionInsight Manager portal](#). Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Yarn\config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/client/conf**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Yarn/config/* root@IP address of the client node:/opt/client/conf
```

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

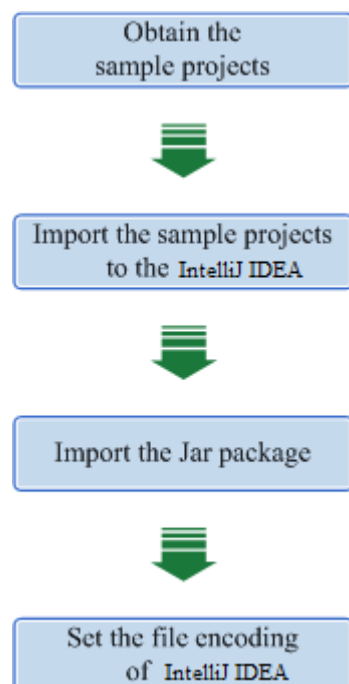
## 25.2.2 Configuring and Importing Sample Projects

### Scenario

MapReduce provides sample projects for multiple scenarios to help you quickly learn MapReduce projects.

The procedure of importing MapReduce example codes is described as follows: [Figure 25-2](#) shows the procedure.

**Figure 25-2** Procedure of importing sample projects

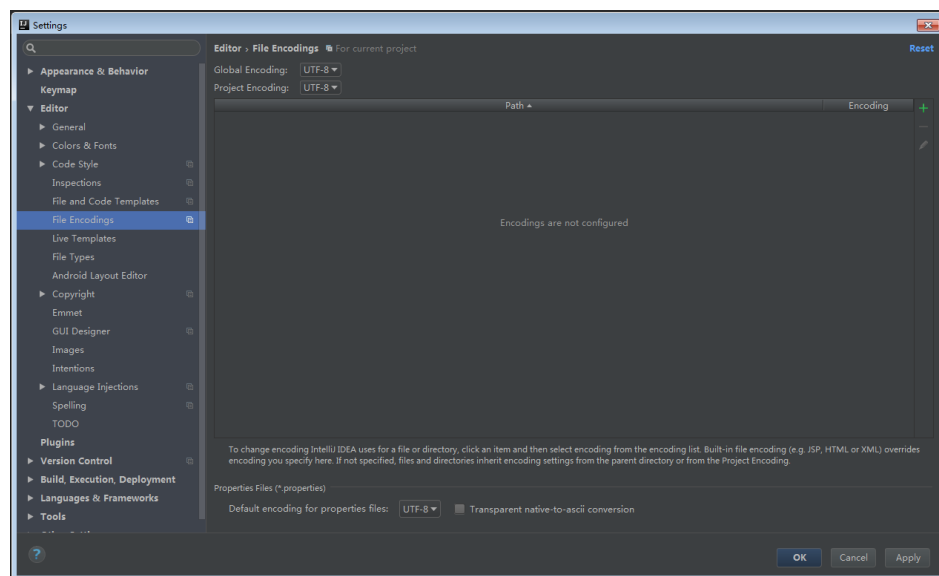




## Procedure

- Step 1** Obtain the sample project **mapreduce-example-normal** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Import the example project to the IntelliJ IDEA development environment.
1. Open IntelliJ IDEA and choose **File > Open**.
  2. Choose the directory of the example project **mapreduce-example-normal**. Click **OK**.
- Step 3** Set the IntelliJ IDEA text file coding format to prevent garbled characters.
1. On the IntelliJ IDEA menu bar, choose **File > Settings**.  
The **Settings** window is displayed.
  2. Choose **Editor > File Encodings** from the navigation tree. In the "Global Encoding" and "Project Encodings" area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 25-3](#)

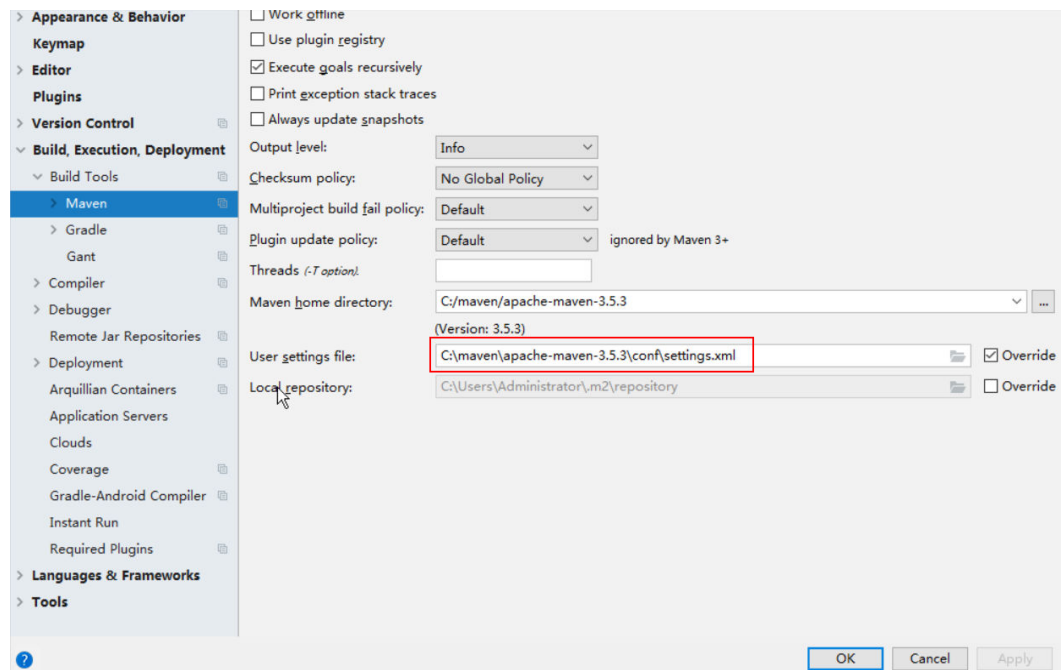
**Figure 25-3** Setting the IntelliJ IDEA coding format



- Step 4** Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).

On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is `<Local Maven installation directory>\conf \settings.xml`.

Figure 25-4 Directory for storing the settings.xml file



----End

## Reference

The dependency packages mapped to the sample projects of MapReduce are as follows:

- MapReduce statistics sample project  
No additional jars.
- MapReduce accessing multi-components sample project

### NOTE

- If you want to use the multi-component accessing sample project after importing a sample project, ensure that the HBase service have been installed in the cluster.
- If you do not use the multi-components accessing sample project, you can ignore errors about the multi-components accessing sample project as long as the compilation of the statistics sample project is not affected. Otherwise, delete the files about the multi-components accessing sample project after importing the sample projects.

## 25.2.3 Creating a New Project (Optional)

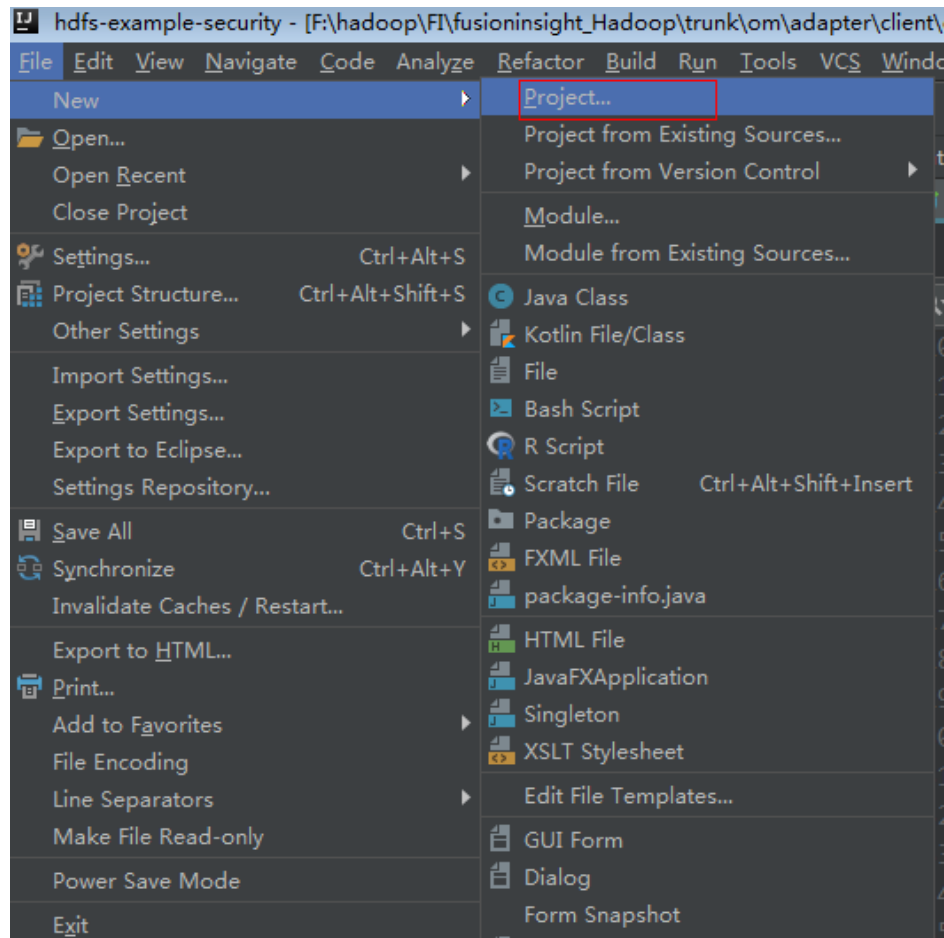
### Scenario

Apart from importing MapReduce sample projects, you can also create a new MapReduce project using IntelliJ IDEA.

### Procedure

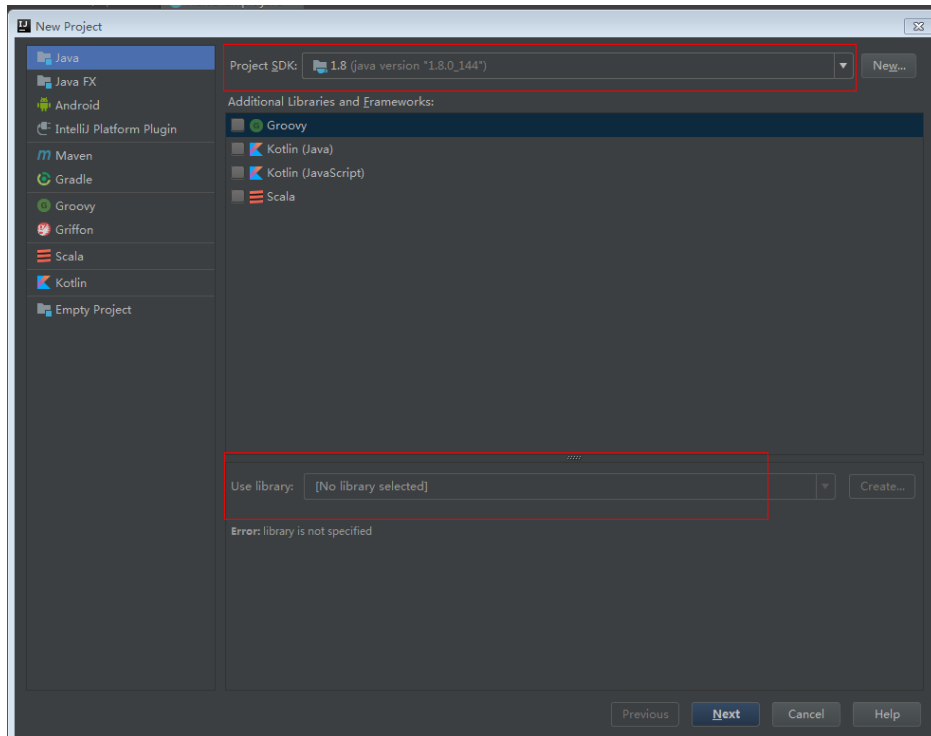
**Step 1** Open IntelliJ IDEA and choose **File > New > Project**, as shown in [Figure 25-5](#).

**Figure 25-5** Creating a project



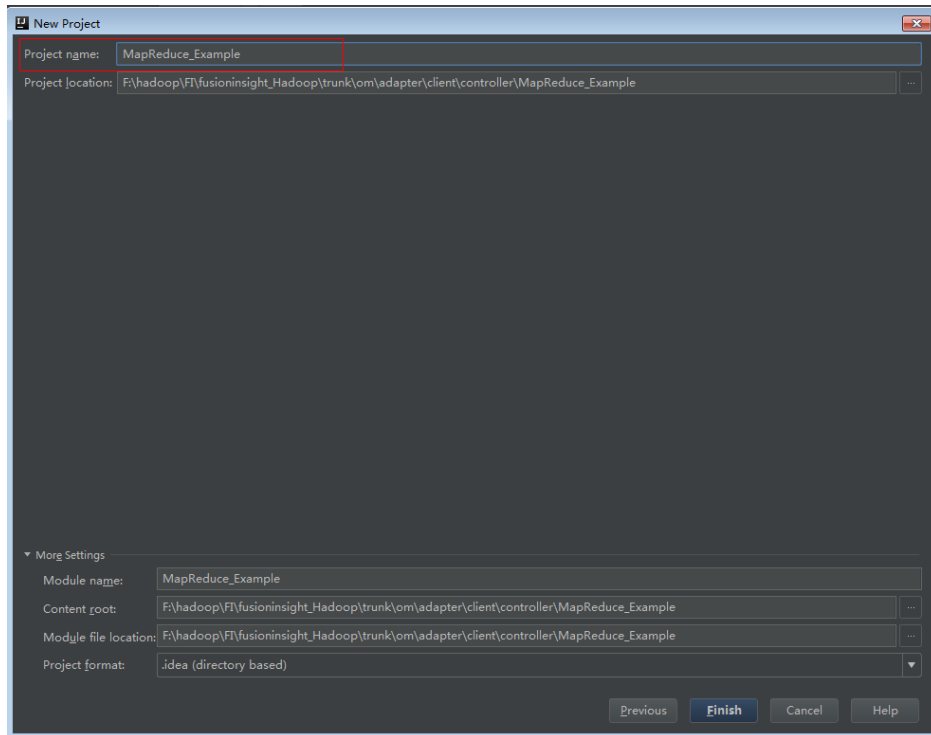
**Step 2** On the **New Project** page, select **Java**, and then configure the JDK and other Java libraries required by the project. [Figure 25-6](#) shows the SDK information required for configuring a project. After the configuration is complete, click **Next**.

**Figure 25-6** Configuring SDK Information Required by the Project



**Step 3** Enter the name of the new project in the dialog box. Click **Finish**.

**Figure 25-7** Enter the project name



----End

## 25.3 Developing the Project

### 25.3.1 MapReduce Statistics Sample Project

#### 25.3.1.1 Typical Scenarios

##### Scenario

Develop a MapReduce application to perform the following operations on logs about dwell durations of netizens for shopping online:

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.
- The first column in the log file records names, the second column records gender, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

**log1.txt:** logs collected on Saturday

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

**log2.txt:** logs collected on Sunday

```
LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
FangBo,female,50
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
CaiXuyu,female,50
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
FangBo,female,50
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

##### Data Preparation

Save log files in the Hadoop distributed file system (HDFS).

1. Create text files `input_data1.txt` and `input_data2.txt` on the Linux operating system, and copy `log1.txt` to `input_data1.txt` and `log2.txt` to `input_data2.txt`.
2. Create `/tmp/input` on the HDFS, and run the following commands to upload `input_data1.txt` and `input_data2.txt` to `/tmp/input`:

- a. On the Linux client, run ***hdfs dfs -mkdir /tmp/input.***
- b. On the Linux client, run ***hdfs dfs -put local\_file\_path /tmp/input.***

## Development Idea

Collects the information about female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

- Read the source file data.
- Filter the data information about the time that female netizens spend online.
- Aggregate the total time that each female netizen spends online.
- Filter the information about female netizens who spend more than 2 hours online.

### 25.3.1.2 Example Codes

#### Function

Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.

The operation is performed in three steps:

- Filter the online time of female netizens in original files using the `CollectionMapper` class inherited from the `Mapper` abstract class.
- Count the online time of each female netizen, and output information about female netizens who dwell online for more than 2 hours using the `CollectionReducer` class inherited from the `Reducer` abstract class.
- The main method creates a MapReduce job and submits the MapReduce job to the Hadoop cluster.

#### Example Codes

The following code snippets are used as an example. For complete codes, see the `com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector` class.

Example 1: The `CollectionMapper` class defines the `map()` and `setup()` methods of the `Mapper` abstract class.

```
public static class CollectionMapper extends
 Mapper<Object, Text, Text, IntWritable> {

 // Delimiter.
 String delim;
 // Filter sex.
 String sexFilter;

 // Name.
 private Text nameInfo = new Text();

 // Output <key,value> must be serialized.
 private IntWritable timeInfo = new IntWritable(1);

 /**
 * Distributed computing
```

```

*
* @param key Object: location offset of the source file.
* @param value Text: a row of characters in the source file.
* @param context Context: output parameter.
* @throws IOException , InterruptedException
*/
public void map(Object key, Text value, Context context)
 throws IOException, InterruptedException
{
 String line = value.toString();

 if (line.contains(sexFilter))
 {

 // A character string that has been read.
 String name = line.substring(0, line.indexOf(delim));
 nameInfo.set(name);
 // Obtain the dwell duration.
 String time = line.substring(line.lastIndexOf(delim) + 1,
 line.length());
 timeInfo.set(Integer.parseInt(time));

 // The Map task outputs a key-value pair.
 context.write(nameInfo, timeInfo);
 }
}
/**
 * map use to init.
 *
 * @param context Context.
 */
public void setup(Context context) throws IOException,
 InterruptedException
{
 // Obtain configuration information using Context.
 delim = context.getConfiguration().get("log.delimiter", ",");

sexFilter = delim
 + context.getConfiguration()
 .get("log.sex.filter", "female") + delim;
}
}

```

Example 2: The CollectionReducer class defines the reduce() method of the Reducer abstract class.

```

public static class CollectionReducer extends
 Reducer<Text, IntWritable, Text, IntWritable>
{
 // Statistical results.
 private IntWritable result = new IntWritable();

 // Total time threshold.
 private int timeThreshold;

 /**
 * @param key Text : key after Mapper.
 * @param values Iterable : all statistical results with the same key.
 * @param context Context
 * @throws IOException , InterruptedException
 */
 public void reduce(Text key, Iterable<IntWritable> values,
 Context context) throws IOException, InterruptedException
 {
 int sum = 0;
 for (IntWritable val : values) {

```

```
 sum += val.get();
 }

 // No results are output if the time is less than the threshold.
 if (sum < timeThreshold)
 {
 return;
 }
 result.set(sum);

 // In the output information, key indicates netizen information, and value indicates the total online
time of the netizen.
 context.write(key, result);
}

/**
 * The setup() method is invoked for only once before the map() method or reduce() method.
 *
 * @param context Context
 * @throws IOException , InterruptedException
 */
public void setup(Context context) throws IOException,
 InterruptedException
{
 // Context obtains configuration information.
 timeThreshold = context.getConfiguration().getInt(
 "log.time.threshold", 120);
}
}
```

Example 3: Use the main() method to create a job, set parameters, and submit the job to the hadoop cluster.

```
public static void main(String[] args) throws Exception {
 // Initialize environment variables.
 Configuration conf = new Configuration();

 // Obtain input parameters.
 String[] otherArgs = new GenericOptionsParser(conf, args)
 .getRemainingArgs();
 if (otherArgs.length != 2) {
 System.err.println("Usage: collect female info <in> <out>");
 System.exit(2);
 }

 // Initialize the job object.
 @SuppressWarnings("deprecation")
 Job job = Job.getInstance(conf, "Collect Female Info");
 job.setJarByClass(FemaleInfoCollector.class);

 // Set map and reduce classes to be executed, or specify the map and reduce classes using configuration
files.
 job.setMapperClass(CollectionMapper.class);
 job.setReducerClass(CollectionReducer.class);

 // Set the Combiner class. The combiner class is not used by default. Classes same as the reduce class are
used.
 // Exercise caution when using the Combiner class. You can specify it using configuration files.
 job.setCombinerClass(CollectionCombiner.class);

 // Set the output type of the job.
 job.setOutputKeyClass(Text.class);
 job.setOutputValueClass(IntWritable.class);
 FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
 FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

 // Submit the job to a remote environment for execution.
 System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```



Example 4: CollectionCombiner class combines the mapped data on the map side to reduce the amount of data transmitted from map to reduce.

```
/**
 * Combiner class
 */
public static class CollectionCombiner extends
Reducer<Text, IntWritable, Text, IntWritable> {

// Intermediate statistical results
private IntWritable intermediateResult = new IntWritable();

/**
 * @param key Text : key after Mapper
 * @param values Iterable : all results with the same key in this map task
 * @param context Context
 * @throws IOException , InterruptedException
 */
public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}

intermediateResult.set(sum);

// In the output information, key indicates netizen information,
// and value indicates the total online time of the netizen in this map task.
context.write(key, intermediateResult);
}
}
```

## 25.3.2 MapReduce Accessing Multi-Component Example Project

### 25.3.2.1 Instance

#### Scenario

The sample project illustrates how to compile MapReduce jobs to visit multiple service components in HDFS, HBase, and Hive, helping users to understand key actions such as certificating and configuration loading.

The logic of the sample project is as follows:

The input data is HDFS text file and the input file is **log1.txt**.

```
YuanJing,male,10
GuoYijun,male,5
```

Map:

1. Obtain one row of the input data and extract the user name.
2. Query one piece of data from HBase.
3. Query one piece of data from Hive.
4. Combine the data queried from HBase and that from Hive as the output of Map as the output of Map.

Reduce:

1. Obtain the last piece of data from Map output.
2. Import the data to HBase.
3. Save the data to HDFS.

## Data Planning

1. Create an HDFS data file.
  - a. Create a text file named **data.txt** in the Linux-based HDFS and copy the content of **log1.txt** to **data.txt**.
  - b. Run the following commands to create a directory **/tmp/examples/multi-components/mapreduce/input/** and copy the **data.txt** to the directory:
    - i. ***hdfs dfs -mkdir -p /tmp/examples/multi-components/mapreduce/input/***
    - ii. ***hdfs dfs -put data.txt /tmp/examples/multi-components/mapreduce/input/***
2. Create a HBase table and insert data into it.
  - a. Run the **source bigdata\_env** command on a Linux-based HBase client and run the **hbase shell** command.
  - b. Run the **create 'table1', 'cf'** command in the HBase shell to create table1 with column family **cf**.
  - c. Run the **put 'table1', '1', 'cf:cid', '123'** command to insert data whose rowkey is **1**, column name is **cid**, and data value is **123**.
  - d. Run the **quit** command to exit the table.
3. Create a Hive table and load data to it.
  - a. Run the **beeline** command on a Linux-based Hive client.
  - b. In the Hive beeline interaction window, run the **CREATE TABLE person(name STRING, gender STRING, stayTime INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ';' stored as textfile;** command to create data table **person** with three fields.
  - c. In the Hive beeline interaction window, run the **LOAD DATA INPATH '/tmp/examples/multi-components/mapreduce/input/' OVERWRITE INTO TABLE person;** command to load data files to the **person** table.
  - d. Run **!q** to exit the table.
4. The data of HDFS is cleared in the preceding step. Therefore, perform **1** again.

### 25.3.2.2 Example Code

#### Function

The functions of the sample project are as follows:

- Collect the name information from HDFS source files, query and combine data of HBase and Hive using the MultiComponentMapper class inherited from the Mapper abstract class.
- Obtain the last piece of mapped data and output to HBase and HDFS, using the MultiComponentMapper class inherited from the Reducer abstract class.

- The main function creates a MapReduce job and submits the MapReduce job to Hadoop clusters.

## Example Code

For details about code, see the class `com.huawei.bigdata.mapreduce.examples.MultiComponentExample`.

Example code of the map function used by `MultiComponentMapper` class to define the Mapper abstract class.

```
private static class MultiComponentMapper extends Mapper<Object, Text, Text, Text> {

 Configuration conf;

 @Override protected void map(Object key, Text value, Context context) throws IOException,
 InterruptedException {

 conf = context.getConfiguration();

 String name = "";
 String line = value.toString();
 if (line.contains("male")) {
 // A character string that has been read
 name = line.substring(0, line.indexOf(","));
 }
 // 1. read from HBase
 String hbaseData = readHBase();

 // 2. read from Hive
 String hiveData = readHive(name);

 // The Map task outputs a key-value pair.
 context.write(new Text(name), new Text("hbase:" + hbaseData + ", hive:" + hiveData));
 }
}
```

Example code of the `readHBase` function.

```
private String readHBase() {
 String tableName = "table1";
 String columnFamily = "cf";
 String hbaseKey = "1";
 String hbaseValue;

 Configuration hbaseConfig = HBaseConfiguration.create(conf);
 org.apache.hadoop.hbase.client.Connection conn = null;
 try {
 // Create a HBase connection
 conn = ConnectionFactory.createConnection(hbaseConfig);
 // get table
 Table table = conn.getTable(TableName.valueOf(tableName));
 // Instantiate a Get object.
 Get get = new Get(hbaseKey.getBytes());
 // Submit a get request.
 Result result = table.get(get);
 hbaseValue = Bytes.toString(result.getValue(columnFamily.getBytes(), "cid".getBytes()));

 return hbaseValue;
 } catch (IOException e) {
 LOG.warn("Exception occur ", e);
 } finally {
 if (conn != null) {
 try {
 conn.close();
 } catch (Exception e1) {
 LOG.error("Failed to close the connection ", e1);
 }
 }
 }
}
```

```

 }
 }
}
return "";
}

```

Example of the readHive function.

```

private String readHive(String name) throws IOException {
 //Load the configuration file.
 Properties clientInfo = null;
 String userdir = System.getProperty("user.dir") + "/";
 InputStream fileInputStream = null;
 try {
 clientInfo = new Properties();
 String hiveclientProp = userdir + "hiveclient.properties";
 File propertiesFile = new File(hiveclientProp);
 fileInputStream = new FileInputStream(propertiesFile);
 clientInfo.load(fileInputStream);
 } catch (Exception e) {
 throw new IOException(e);
 } finally {
 if (fileInputStream != null) {
 fileInputStream.close();
 }
 }
 String zkQuorum = clientInfo.getProperty("zk.quorum");
 String zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
 String serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
 // Read this carefully:
 // MapReduce can only use Hive through JDBC.
 // Hive will submit another MapReduce Job to execute query.
 // So we run Hive in MapReduce is not recommended.
 final String driver = "org.apache.hive.jdbc.HiveDriver";

 String sql = "select name,sum(stayTime) as " + "stayTime from person where name = " + name + "
group by name";

 StringBuilder sBuilder = new StringBuilder("jdbc:hive2://").append(zkQuorum).append("/");
 sBuilder
 .append(";serviceDiscoveryMode=")
 .append(serviceDiscoveryMode)
 .append(";zooKeeperNamespace=")
 .append(zooKeeperNamespace)
 .append(";");
 String url = sBuilder.toString();
 Connection connection = null;
 PreparedStatement statement = null;
 ResultSet resultSet = null;
 try {
 Class.forName(driver);
 connection = DriverManager.getConnection(url, "", "");
 statement = connection.prepareStatement(sql);
 resultSet = statement.executeQuery();

 if (resultSet.next()) {
 return resultSet.getString(1);
 }
 } catch (ClassNotFoundException e) {
 LOG.warn("Exception occur ", e);
 } catch (SQLException e) {
 LOG.warn("Exception occur ", e);
 } finally {
 if (null != resultSet) {
 try {
 resultSet.close();
 } catch (SQLException e) {
 // handle exception
 }
 }
 }
}

```

```

 }
 if (null != statement) {
 try {
 statement.close();
 } catch (SQLException e) {
 // handle exception
 }
 }
 if (null != connection) {
 try {
 connection.close();
 } catch (SQLException e) {
 // handle exception
 }
 }
}

return "";
}

```

 **NOTE**

Replace all the zkQuorum objects with the actual information about the ZooKeeper cluster nodes.

Example code of the reduce function used by MultiComponentReducer class to define the Reducer abstract class.

```

private static class MultiComponentReducer extends Reducer<Text, Text, Text, Text> {
 Configuration conf;

 public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
 InterruptedException {
 conf = context.getConfiguration();

 Text finalValue = new Text("");
 // just pick the last value as the data to save
 for (Text value : values) {
 finalValue = value;
 }

 // write data to HBase
 writeHBase(key.toString(), finalValue.toString());

 // save result to HDFS
 context.write(key, finalValue);
 }
}

```

Example of the writeHBase function.

```

private void writeHBase(String rowKey, String data) {
 String tableName = "table1";
 String columnFamily = "cf";

 Configuration hbaseConfig = HBaseConfiguration.create(conf);
 org.apache.hadoop.hbase.client.Connection conn = null;
 try {
 // Create a HBase connection
 conn = ConnectionFactory.createConnection(hbaseConfig);
 // get table
 Table table = conn.getTable(TableName.valueOf(tableName));

 // create a Put to HBase
 List<Put> list = new ArrayList<Put>();
 byte[] row = Bytes.toBytes("row" + rowKey);
 Put put = new Put(row);
 byte[] family = Bytes.toBytes(columnFamily);
 byte[] qualifier = Bytes.toBytes("value");
 }
}

```



## 25.4 Commissioning the Application

### 25.4.1 Commissioning the Application in the Windows Environment

#### 25.4.1.1 Compiling and Running the Application

##### Scenario

After the code development is complete, you can run an application in the Windows development environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

##### NOTE

Do not restart HDFS service while MapReduce application is in running status, otherwise the application will fail.

#### Running MapReduce Statistics Sample Project

- Step 1** Ensure that all JAR packages on which the sample project depends have been obtained.
- Step 2** In the IntelliJ IDEA development environment, select the LocalRunner.java project. Right-click the project and choose **Run MultiComponentLocalRunner.main()** from the shortcut menu to run the project. Click to run the related application project.

----End

#### Running Sample Applications About Multi-Components

- Step 1** Save the **hive-site.xml**, **hbase-site.xml**, and **hiveclient.properties** files to the **conf** directory of the project.
- Step 2** Ensure that all Hive and HBase JAR packages on which the sample project depends have been obtained.
- Step 3** In the MultiComponentLocalRunner.java code, `System.setProperty("HADOOP_USER_NAME", "root");` specifies that user **root** is used. Ensure that user **root** is used for uploading data, or change user **root** in the code to the username used for uploading data.
- Step 4** In the IntelliJ IDEA development environment, click **MultiComponentLocalRunner.java** to run the application project. You can also right-click the project and choose **Run MultiComponentLocalRunner.main()** from the shortcut menu to run the project.

----End

## 25.4.1.2 Checking the Commissioning Result

### Scenario

After a MapReduce application is run, you can check the running result through one of the following methods:

- Viewing the program running status in IntelliJ IDEA.
- Viewing MapReduce logs.
- Logging in to the MapReduce WebUI.
- Logging in to the Yarn WebUI.

#### NOTE

You must have permission to access WebUI. If not, contact the admin to obtain an account and password.

### Procedure

- **Check the running result by obtaining the MapReduce log.**

As shown below, the console outputs the application running result.

```
3614 [main] INFO org.apache.hadoop.hdfs.PeerCache - SocketCache disabled.
10159 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to
process : 2
11378 [main] INFO org.apache.hadoop.mapreduce.JobSubmitter - number of splits:2
12707 [main] INFO org.apache.hadoop.mapreduce.JobSubmitter - Submitting tokens for job:
job_1468241424339_0002
16434 [main] INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl - Submitted application
application_1468241424339_0002
16656 [main] INFO org.apache.hadoop.mapreduce.Job - The url to track the job: http://
10-120-180-170:8088/proxy/application_1468241424339_0002/
16657 [main] INFO org.apache.hadoop.mapreduce.Job - Running job: job_1468241424339_0002
31177 [main] INFO org.apache.hadoop.mapreduce.Job - Job job_1468241424339_0002 running in
uber mode : false
31200 [main] INFO org.apache.hadoop.mapreduce.Job - map 0% reduce 0%
45893 [main] INFO org.apache.hadoop.mapreduce.Job - map 100% reduce 0%
57172 [main] INFO org.apache.hadoop.mapreduce.Job - map 100% reduce 100%
58554 [main] INFO org.apache.hadoop.mapreduce.Job - Job job_1468241424339_0002 completed
successfully
58908 [main] INFO org.apache.hadoop.mapreduce.Job - Counters: 49
File System Counters
FILE: Number of bytes read=75
FILE: Number of bytes written=436979
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=674
HDFS: Number of bytes written=23
HDFS: Number of read operations=9
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=2
Launched reduce tasks=1
Data-local map tasks=2
Total time spent by all maps in occupied slots (ms)=206088
Total time spent by all reduces in occupied slots (ms)=73824
Total time spent by all map tasks (ms)=25761
Total time spent by all reduce tasks (ms)=9228
Total vcore-seconds taken by all map tasks=25761
Total vcore-seconds taken by all reduce tasks=9228
Total megabyte-seconds taken by all map tasks=105517056
Total megabyte-seconds taken by all reduce tasks=37797888
```



```

Map-Reduce Framework
Map input records=26
Map output records=16
Map output bytes=186
Map output materialized bytes=114
Input split bytes=230
Combine input records=16
Combine output records=6
Reduce input groups=3
Reduce shuffle bytes=114
Reduce input records=6
Reduce output records=2
Spilled Records=12
Shuffled Maps =2
Failed Shuffles=0
Merged Map outputs=2
GC time elapsed (ms)=356
CPU time spent (ms)=2860
Physical memory (bytes) snapshot=1601576960
Virtual memory (bytes) snapshot=12999819264
Total committed heap usage (bytes)=2403336192
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=444
File Output Format Counters
Bytes Written=23

```

**NOTE**

In the Windows environment, the following exception occurs but does not affect services.

java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.

- **Check the running result by using MapReduce WebUI.**

LLog in to FusionInsight Manager as a user who has the permission to view task and choose **Cluster > Name of the desired cluster > Services > Mapreduce > JobHistoryServer**. On the web page that is displayed, view the task execution status.

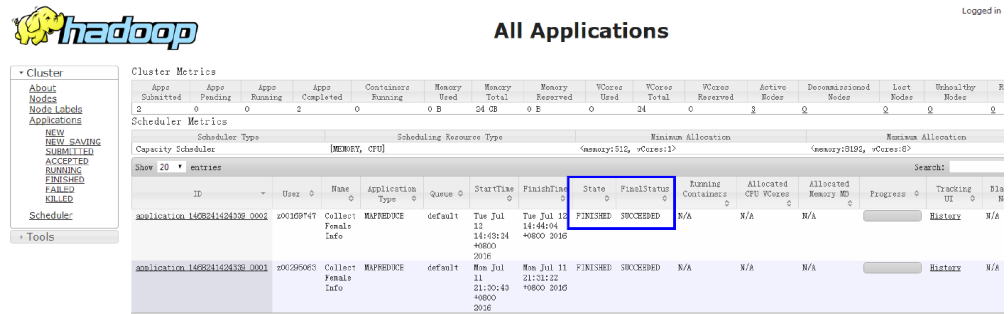
**Figure 25-8** JobHistory Web UI



- **Check the running result by using YARN WebUI.**

Log in to FusionInsight Manager as a user who has the permission to view task and choose **Cluster > Name of the desired cluster > Services > Yarn > ResourceManager(Active)**. On the web page that is displayed, view the task execution status.

Figure 25-9 ResourceManager Web UI



- **View MapReduce logs to learn application running conditions.**  
MapReduce logs offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 25.4.2 Commissioning the Application in the Linux Environment

### 25.4.2.1 Compiling and Running the Application

#### Scenario

After the code development is complete, you can run an application in the Linux development environment.

#### Procedure

- Step 1** Go to the local root directory of the project and run the following command in Windows cmd to compress the package:

```
mvn -s "{maven_setting_path}" clean package
```

**NOTE**

- In the preceding command, {maven\_setting\_path} is the path of the setting.xml file of the local maven.
- After the package is successfully packed, obtain the JAR package, for example, *MRTest-XXX.jar*, from the **target** subdirectory in the root directory of the project. The name of the JAR package varies according to the actual package.

- Step 2** Upload **MRTest-XXX.jar** to the Linux client, such as **/opt/client/conf**, the same directory where the configuration files are located in.

- Step 3** Submit the MapReduce job in the Linux environment and execute the sample project.

- Run the following command for MapReduce statistics sample project to configure parameters and submit jobs:

```
yarn jar MRTest-XXX.jar
com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector
<inputPath> <outputPath>
```

<inputPath> indicates the input path and <outputPath> indicates the output path in HDFS.

 NOTE

- Before running the `yarn jar MRTest-XXX.jar com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector <inputPath> <outputPath>` command, upload the `log1.txt` and `log2.txt` files to the `<inputPath>` directory in HDFS. See [Typical Scenarios](#).
  - Before running the `yarn jar MRTest-XXX.jar com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector <inputPath> <outputPath>` command, ensure that the `<outputPath>` directory is deleted. Otherwise, an error will occur.
  - Do not restart the HDFS service during the running of MapReduce tasks. Otherwise, the tasks may fail.
- In MapReduce accessing multi-components sample project, perform the following operations:
    - a. Obtain the `hbase-site.xml`, `hiveclient.properties` and `hive-site.xml` configuration files, create a folder for example `/opt/client/conf`, and save the configurations files.

 NOTE

The file `hbase-site.xml` is acquired from the HBase client, `hiveclient.properties` and `hive-site.xml` are acquired from the Hive client.

- b. Add the classpath required for sample projects in the Linux environment, Example of classpath:

```
export YARN_USER_CLASSPATH=/opt/client/conf:/opt/client/HBase/hbase/lib/*:/opt/client/HBase/hbase/lib/client-facing-thirdparty/*:/opt/client/Hive/Beeline/lib/*
```

- c. Submit MapReduce jobs. Run the following command to run the sample project:

```
yarn jar MRTest-XXX.jar com.huawei.bigdata.mapreduce.examples.MultiComponentExample
```

----End

## 25.4.2.2 Checking the Commissioning Result

### Scenario

After a MapReduce application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Logging in to the MapReduce WebUI.
- Logging in to the Yarn WebUI.
- Viewing MapReduce logs.

 NOTE

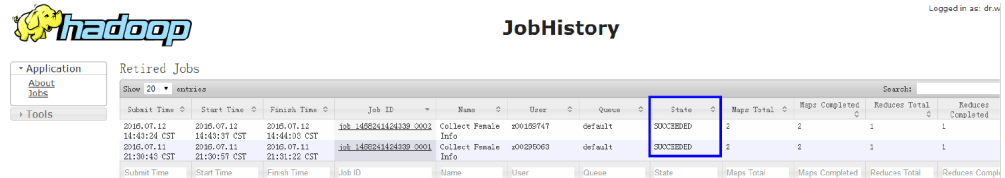
You must have permission to access WebUI. If not, contact the admin to obtain an account and password.

## Procedure

- **Check the running result by using MapReduce WebUI.**

Log in to FusionInsight Manager as a user who has the permission to view task and choose **Cluster > Name of the desired cluster > Services > Mapreduce > JobHistoryServer**. On the web page that is displayed, view the task execution status.

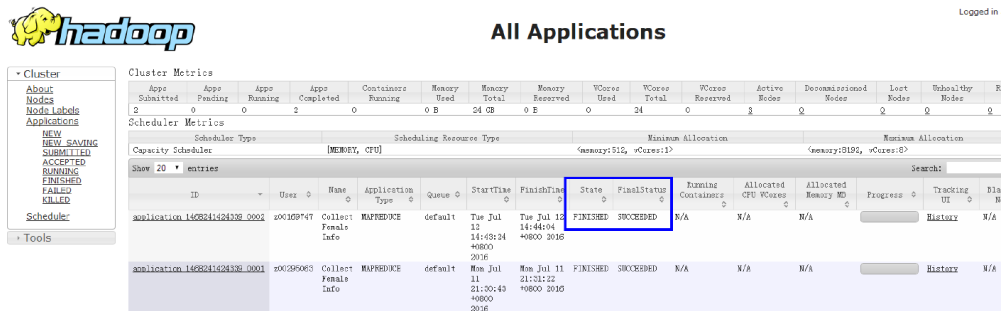
Figure 25-10 JobHistory Web UI



- **Check the running result by using YARN WebUI.**

Log in to FusionInsight Manager as a user who has the permission to view task and choose **Cluster > Name of the desired cluster > Services > Yarn > ResourceManager(Active)**. On the web page that is displayed, view the task execution status.

Figure 25-11 ResourceManager Web UI



- **Check the running result of the MapReduce application.**

- After running the **yarn jar MRTest-XXX.jar** command in the Linux environment, you can check the running status of the application by the returned information about the command.

```
yarn jar MRTest-XXX.jar /tmp/mapred/example/input/ /tmp/root/output/1
16/07/12 17:07:16 INFO hdfs.PeerCache: SocketCache disabled.
16/07/12 17:07:17 INFO input.FileInputFormat: Total input files to process : 2
16/07/12 17:07:18 INFO mapreduce.JobSubmitter: number of splits:2
16/07/12 17:07:18 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1468241424339_0006
16/07/12 17:07:18 INFO impl.YarnClientImpl: Submitted application
application_1468241424339_0006
16/07/12 17:07:18 INFO mapreduce.Job: The url to track the job: http://10-120-180-170:8088/
proxy/application_1468241424339_0006/
16/07/12 17:07:18 INFO mapreduce.Job: Running job: job_1468241424339_0006
16/07/12 17:07:31 INFO mapreduce.Job: Job job_1468241424339_0006 running in uber mode :
false
16/07/12 17:07:31 INFO mapreduce.Job: map 0% reduce 0%
16/07/12 17:07:41 INFO mapreduce.Job: map 50% reduce 0%
16/07/12 17:07:43 INFO mapreduce.Job: map 100% reduce 0%
16/07/12 17:07:51 INFO mapreduce.Job: map 100% reduce 100%
16/07/12 17:07:51 INFO mapreduce.Job: Job job_1468241424339_0006 completed successfully
16/07/12 17:07:51 INFO mapreduce.Job: Counters: 49
File System Counters
FILE: Number of bytes read=75
```

```
FILE: Number of bytes written=435659
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=674
HDFS: Number of bytes written=23
HDFS: Number of read operations=9
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
 Launched map tasks=2
 Launched reduce tasks=1
 Data-local map tasks=2
 Total time spent by all maps in occupied slots (ms)=144984
 Total time spent by all reduces in occupied slots (ms)=56280
 Total time spent by all map tasks (ms)=18123
 Total time spent by all reduce tasks (ms)=7035
 Total vcore-milliseconds taken by all map tasks=18123
 Total vcore-milliseconds taken by all reduce tasks=7035
 Total megabyte-milliseconds taken by all map tasks=74231808
 Total megabyte-milliseconds taken by all reduce tasks=28815360
Map-Reduce Framework
 Map input records=26
 Map output records=16
 Map output bytes=186
 Map output materialized bytes=114
 Input split bytes=230
 Combine input records=16
 Combine output records=6
 Reduce input groups=3
 Reduce shuffle bytes=114
 Reduce input records=6
 Reduce output records=2
 Spilled Records=12
 Shuffled Maps =2
 Failed Shuffles=0
 Merged Map outputs=2
 GC time elapsed (ms)=202
 CPU time spent (ms)=2720
 Physical memory (bytes) snapshot=1595645952
 Virtual memory (bytes) snapshot=12967759872
 Total committed heap usage (bytes)=2403860480
Shuffle Errors
 BAD_ID=0
 CONNECTION=0
 IO_ERROR=0
 WRONG_LENGTH=0
 WRONG_MAP=0
 WRONG_REDUCE=0
File Input Format Counters
 Bytes Read=444
File Output Format Counters
 Bytes Written=23
```

- In the Linux environment, run the ***yarn application -status <ApplicationID>*** command to check the running result of the current application. Example:

```
yarn application -status application_1468241424339_0006
Application Report :
 Application-Id : application_1468241424339_0006
 Application-Name : Collect Female Info
 Application-Type : MAPREDUCE
 User : root
 Queue : default
 Start-Time : 1468314438442
 Finish-Time : 1468314470080
 Progress : 100%
 State : FINISHED
 Final-State : SUCCEEDED
```

```
Tracking-URL : http://10-120-180-170:19888/jobhistory/job/job_1468241424339_0006
RPC Port : 27100
AM Host : 10-120-169-46
Aggregate Resource Allocation : 172153 MB-seconds, 64 vcore-seconds
Log Aggregation Status : SUCCEEDED
Diagnostics : Application finished execution.
Application Node Label Expression : <Not set>
AM container Node Label Expression : <DEFAULT_PARTITION>
```

- **View MapReduce logs to learn application running conditions.**  
MapReduce logs offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 25.5 More Information

### 25.5.1 Common APIs

#### 25.5.1.1 Java API

Directly consult official website for detailed API of MapReduce: **Common Interfaces**

<http://hadoop.apache.org/docs/r3.1.1/api/index.html>

Common classes in MapReduce are as follows:

- org.apache.hadoop.mapreduce.Job: an interface for users to submit MR jobs and used to set job parameters, submit jobs, control job executions, and query job status.
- org.apache.hadoop.mapred.JobConf: configuration class of a MapReduce job and major configuration interface for users to submit jobs to Hadoop.

**Table 25-3** Common interfaces of org.apache.hadoop.mapreduce.Job

| Interface                                                        | Description                                                                                                                                                                                        |
|------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Job(Configuration conf, String jobName), Job(Configuration conf) | Creates a MapReduce client for configuring job attributes and submitting the job.                                                                                                                  |
| setMapperClass(Class<extends Mapper> cls)                        | A core interface used to specify the Mapper class of a MapReduce job. The Mapper class is empty by default. You can also configure <b>mapreduce.job.map.class</b> in <b>mapred-site.xml</b> .      |
| setReducerClass(Class<extends Reducer> cls)                      | A core interface used to specify the Reducer class of a MapReduce job. The Reducer class is empty by default. You can also configure <b>mapreduce.job.reduce.class</b> in <b>mapred-site.xml</b> . |

| Interface                                                  | Description                                                                                                                                                                                                                                                                                                                                                                             |
|------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| setCombinerClass(Class<extends Reducer> cls)               | Specifies the Combiner class of a MapReduce job. The Combiner class is empty by default. You can also configure <b>mapreduce.job.combine.class</b> in <b>mapred-site.xml</b> . The Combiner class can be used only when the input and output key and value types of the reduce task are the same.                                                                                       |
| setInputFormatClass(Class<extends InputFormat> cls)        | A core interface used to specify the InputFormat class of a MapReduce job. The default InputFormat class is <b>TextInputFormat</b> . You can also configure <b>mapreduce.job.inputformat.class</b> in <b>mapred-site.xml</b> . This interface can be used to specify the InputFormat class for processing data in different formats, reading data, and splitting data into data blocks. |
| setJarByClass(Class< > cls)                                | A core interface used to specify the local location of the JAR package of a class. Java locates the JAR package based on the class file and uploads the JAR package to the Hadoop distributed file system (HDFS).                                                                                                                                                                       |
| setJar(String jar)                                         | Specifies the local location of the JAR package of a class. You can directly set the location of a JAR package and upload the JAR package to the HDFS. Use either <code>setJar(String jar)</code> or <code>setJarByClass(Class&lt; &gt; cls)</code> . You can also configure <b>mapreduce.job.jar</b> in <b>mapred-site.xml</b> .                                                       |
| setOutputFormatClass(Class<extends OutputFormat> theClass) | A core interface used to specify the OutputFormat class of a MapReduce job. The default OutputFormat class is <b>TextOutputFormat</b> . You can also configure <code>mapred.output.format.class</code> in <b>mapred-site.xml</b> . In the default <b>TextOutputFormat</b> , each key and value are recorded in text. <b>OutputFormat</b> is not specified usually.                      |
| setOutputKeyClass(Class< > theClass)                       | A core interface used to specify the output key type of a MapReduce job. You can also configure <b>mapreduce.job.output.key.class</b> in <b>mapred-site.xml</b> .                                                                                                                                                                                                                       |
| setOutputValueClass(Class< > theClass)                     | A core interface used to specify the output value type of a MapReduce job. You can also configure <b>mapreduce.job.output.value.class</b> in <b>mapred-site.xml</b> .                                                                                                                                                                                                                   |

| Interface                                                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| setPartitionerClass(Class<extends Partitioner> theClass) | Specifies the Partitioner class of a MapReduce job. You can also configure <b>mapred.partitioner.class</b> in <b>mapred-site.xml</b> . This method is used to allocate Map output results to reduce classes. <b>HashPartitioner</b> is used by default, which evenly allocates the key-value pairs of a map task. For example, in HBase applications, different key-value pairs belong to different regions. In this case, you must specify the Partitioner class to allocate map output results. |
| setSortComparatorClass(Class<extends RawComparator> cls) | Specifies the compression class for output results of a map task. Compression is not implemented by default. You can also configure <b>mapreduce.map.output.compress</b> and <b>mapreduce.map.output.compress.codec</b> in <b>mapred-site.xml</b> . You can compress data for transmission when the map task outputs a large amount of data.                                                                                                                                                      |
| setPriority(JobPriority priority)                        | Specifies the priority of a MapReduce job. Five priorities can be set: <b>VERY_HIGH</b> , <b>HIGH</b> , <b>NORMAL</b> , <b>LOW</b> , and <b>VERY_LOW</b> . The default priority is <b>NORMAL</b> . You can also configure <b>mapreduce.job.priority</b> in <b>mapred-site.xml</b> .                                                                                                                                                                                                               |

**Table 25-4** Common interfaces of org.apache.hadoop.mapred.JobConf

| Interface             | Description                                                                                                                                                                                                                                                                                                                   |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| setNumMapTasks(int n) | A core interface used to specify the number of map tasks in a MapReduce job. You can also configure <b>mapreduce.job.maps</b> in <b>mapred-site.xml</b> .<br><b>NOTE</b><br>The InputFormat class controls the number of map tasks. Ensure that the InputFormat class supports setting the number of map tasks on the client. |



| Interface                      | Description                                                                                                                                                                                                                                                                                                                                     |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| setNumReduceTasks(int n)       | A core interface used to specify the number of reduce tasks in a MapReduce job. Only one reduce task is started by default. You can also configure <b>mapreduce.job.reduces</b> in <b>mapred-site.xml</b> . The number of reduce tasks is controlled by users. In most cases, the number of reduce tasks is one-fourth the number of map tasks. |
| setQueueName(String queueName) | Specifies the queue where a MapReduce job is submitted. The default queue is used by default. You can also <b>configure mapreduce.job.queueName</b> in <b>mapred-site.xml</b> .                                                                                                                                                                 |

### 25.5.1.2 REST API

#### Function Description

Use the HTTP REST API to view more information about MapReduce tasks. Currently, the REST API of MapResuce can be used to query the status of completed tasks. For details about the API, see the official website:

<http://hadoop.apache.org/docs/r3.1.1/hadoop-mapreduce-client/hadoop-mapreduce-client-hs/HistoryServerRest.html>

#### Preparing the Running Environment

1. Install the client, for example, to the **/opt/client** directory on the node. For details, see section "Installing a Client."
2. Go the client installation directory and run the following commands to configure the environment variables:

```
source bigdata_env
```

#### Procedure

Obtain detailed information about tasks that have been completed on MapReduce.

- Commands for the operation:

```
curl -k -i --negotiate -u : "http://10.120.85.2:19888/ws/v1/history/mapreduce/jobs"
```

In the preceding command, **10.120.85.2** indicates the value of **JHS\_FLOAT\_IP** for MapReduce, and **19888** indicates the port ID of the JobHistoryServer node.

 **NOTE**

In RedHat 6.x and CentOS 6.x, a compatibility problem occurs when the curl command is used to access the JobHistoryServer. As a result, the correct result cannot be returned.

- You can view the status information about historical tasks, such as the task IDs, start time, end time, and task execution status.
- Execution result

```
{
 "jobs":{
 "job":[
 {
 "submitTime":1525693184360,
 "startTime":1525693194840,
 "finishTime":1525693215540,
 "id":"job_1525686535456_0001",
 "name":"QuasiMonteCarlo",
 "queue":"default",
 "user":"mapred",
 "state":"SUCCEEDED",
 "mapsTotal":1,
 "mapsCompleted":1,
 "reducesTotal":1,
 "reducesCompleted":1
 }
]
 }
}
```

- Result analysis:  
Using this API, you can query the completed MapReduce tasks in the current cluster and obtain information listed in [Table 1](#).

**Table 25-5** Common information

| Parameter  | Description                    |
|------------|--------------------------------|
| submitTime | Time when a task is submitted  |
| startTime  | Start time                     |
| finishTime | End time                       |
| queue      | Task queue                     |
| user       | User who submits the task      |
| state      | Task state, success or failure |

## 25.5.2 FAQ

### 25.5.2.1 No Response from the Client When Submitting the MapReduce Application

#### Question

After the MapReduce task is submitted to the YARN server, the client is prompted without response for a long time.

#### Answer

For the above problem, ResourceManager provides the key diagnosis information of MapReduce operating status on the WebUI. For the MapReduce application that is submitted to the YARN, users can obtain the current application status and the reason to be in the status with the diagnosis information.

Procedures:

Log in to FusionInsight Manager, and select **Cluster > Name of the desired cluster > Services > Yarn > ResourceManager(Active)**. Enter the WebUI, click the submitted MapReduce application on the WebUI of ResourceManager (Active), check the diagnosis information on the WebUI, and take measures according to the diagnosis information.

MapReduce logs offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

### 25.5.2.2 How to Perform Remote Debugging During MapReduce Secondary Development?

#### Question

During the secondary development of MapReduce, how to perform remote debugging?

#### Answer

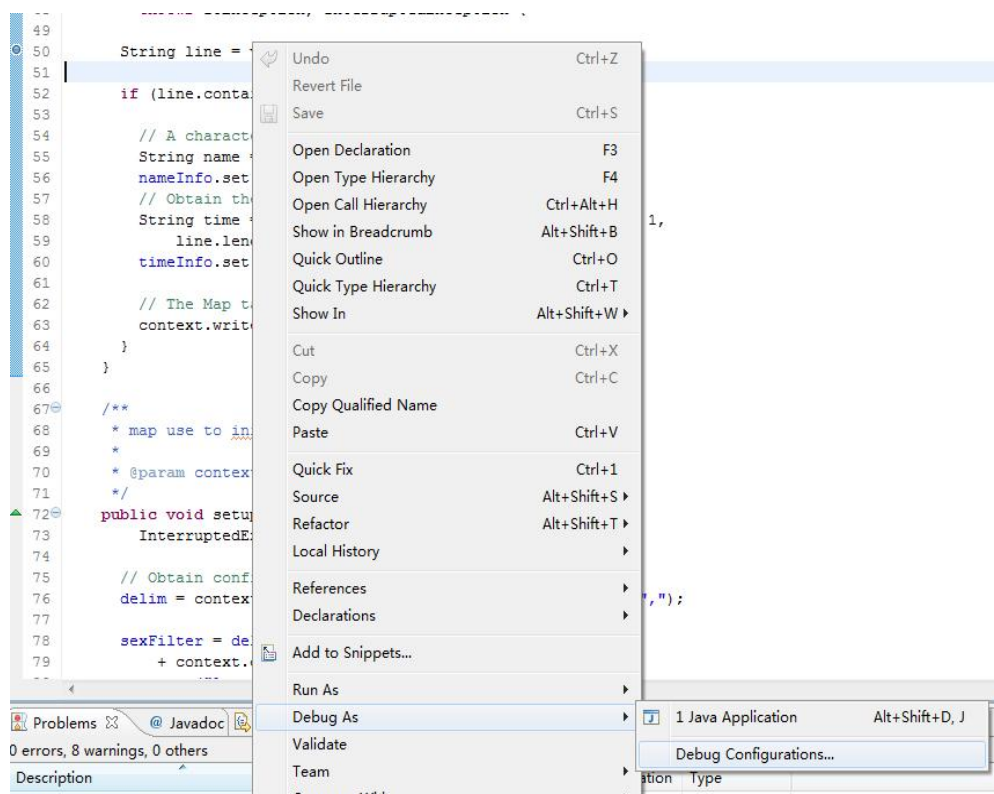
MapReduce adopts Java remote debugging mechanism. Run Java remote debugging commands when starting the Map or Reduce tasks.

- Step 1** The **mapreduce.map.java.opts** and **mapreduce.reduce.java.opts** parameters specify the JVM startup parameters of Map and Reduce tasks respectively. In the **mapred-site.xml** configuration file on the client, add the command - **agentlib:jdwp=transport=dt\_socket,server=y,suspend=y,address=8000** to the **mapreduce.map.java.opts** and **mapreduce.reduce.java.opts** parameters.
- Step 2** MapReduce is a distributed computing framework, and the node where Map or Reduce tasks are running are not fixed. It is advisable to keep only one NodeManager running in the cluster to ensure that the task is executed in the running NodeManager.
- Step 3** Submit MapReduce tasks on the client, then the Map or Reduce tasks will be suspended and listen to the port 8000 to wait for remote debugging.
- Step 4** In IDE, select the implementation class of MapReduce tasks and configure the remote debugging information to perform debugging.

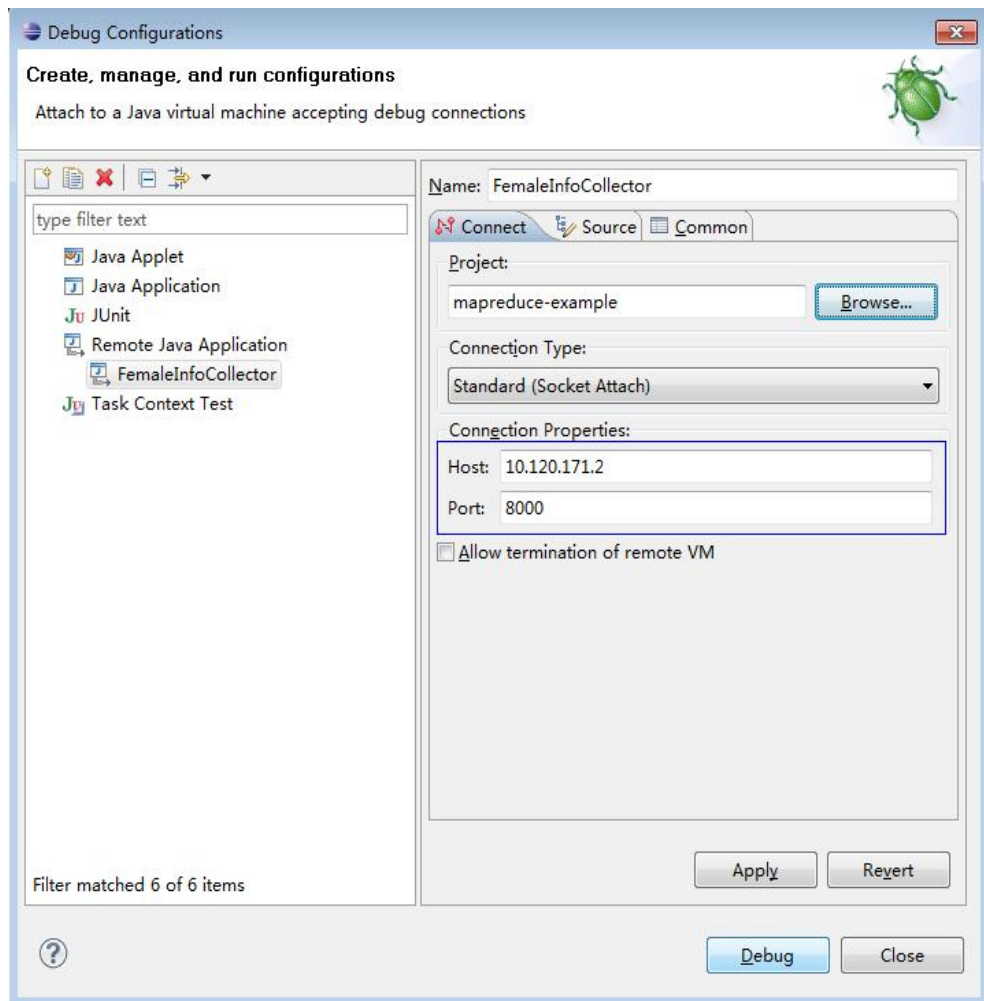
1. Double-click the area in the blue box to configure or cancel the break point.

```
39 /**
40 * Distributed computing
41 *
42 * @param key Object : location offset of the source file
43 * @param value Text : a row of characters in the source file
44 * @param context Context : output parameter
45 * @throws IOException , InterruptedException
46 */
47 public void map(Object key, Text value, Context context)
48 throws IOException, InterruptedException {
49
50 String line = value.toString();
51
52 if (line.contains(sexFilter)) {
53
54 // A character string that has been read
55 String name = line.substring(0, line.indexOf(delim));
56 nameInfo.set(name);
57 // Obtain the dwell duration.
58 String time = line.substring(line.lastIndexOf(delim) + 1,
59 line.length());
60 timeInfo.set(Integer.parseInt(time));
61
62 // The Map task outputs a key-value pair.
63 context.write(nameInfo, timeInfo);
64 }
65 }
66
```

2. Right-click the break point and choose **Debug As->Debug Configurations...** from the shortcut menu.



3. On the displayed page, double-click **Remote Java Application** and configure the **Connection Properties** area. Set **Host** to the IP address of the NodeManager and **Port** to **8000**, and then click **Debug**.



----End

 **NOTE**

If you use IDE to submit MapReduce tasks, the IDE is the client. Modify the **mapred-site.xml** file of the secondary development project by referring to [Step 1](#).

# 26 Oozie Development Guide (Security Mode)

---

## 26.1 Overview

### 26.1.1 Application Development Overview

#### Oozie Introduction

Oozie is a workflow engine used to manage Hadoop jobs. Oozie workflows are defined and described based on the directed acyclic graph (DAG). Oozie supports multiple types of workflow modes and workflow scheduled triggering mechanisms. Oozie features easy extensibility, convenient maintenance, and high reliability and works closely with each component in the Hadoop ecosystem.

Oozie workflows are classified into three types:

- Workflow  
Provides a complete basic service workflow.
- Coordinator  
Built on workflows, triggers workflows in a scheduled manner or based on conditions.
- Bundle  
Built on coordinators, centrally schedules, controls, and manages coordinators.

Oozie provides the following features:

- Supports distribution, merging, and choice workflow modes.
- Works closely with each component in the Hadoop ecosystem.
- Supports parameterized workflow variables.
- Supports scheduled workflow triggering.
- Provides a web console that allows users to view and monitor workflows and view logs.

## 26.1.2 Common Concepts

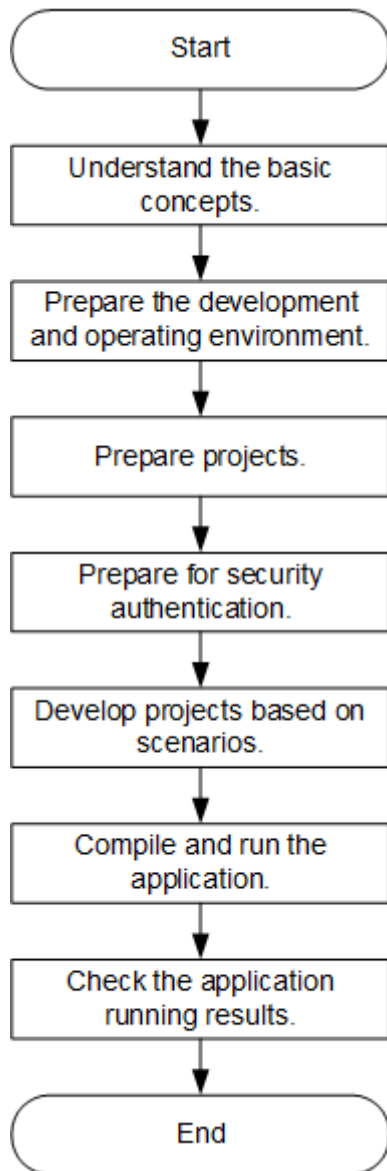
- **Workflow definition file**  
Workflow definition files refer to XML files that describe service logic, including **workflow.xml**, **coordinator.xml**, and **bundle.xml**. Workflow definition files are parsed and executed by the Oozie engine.
- **Workflow property file**  
The workflow property file refers to the parameter configuration file named **job.properties** for workflow execution. Each workflow has only one property file.
- **keytab file**  
The keytab file is a key file that stores user information. In security mode, applications use the key file for API authentication.
- **Client**  
Users can access the server from the client through the Java API, Shell API, REST API, or WebUI to access the Oozie server. The client in this document includes the example code used for accessing Oozie through the Java API.
- **Oozie WebUI**  
Log in to the Oozie WebUI via **https://Oozie server IP address:21003/oozie**.

## 26.1.3 Development Process

This document describes Oozie application development based on the Java API.

For information about each phase in the development process, see [Figure 26-1](#) and [Table 26-1](#).

**Figure 26-1** Oozie application development process



**Table 26-1** Description of Oozie development process

| Phase                          | Description                                                                                                                 | Reference Document              |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| Understand the basic concepts. | Before developing an application, learn basic concepts of Oozie and understand the scenario requirements and design tables. | <a href="#">Common Concepts</a> |



| Phase                                              | Description                                                                                                                | Reference Document                                              |
|----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|
| Prepare the development and operating environment. | The Java language is recommended for the development of Oozie applications. The IDEA tool can be used.                     | <a href="#">Preparing Development and Operating Environment</a> |
| Prepare projects.                                  | Oozie provides example projects for different scenarios. You can import an example project to learn the application.       | <a href="#">Downloading and Importing Sample Projects</a>       |
| Prepare for security authentication.               | If you use a security cluster, you need to perform security authentication.                                                | <a href="#">Preparing Authentication Mechanism Code</a>         |
| Develop projects based on scenarios.               | An example project based on the Java language is provided.                                                                 | <a href="#">Developing the Project</a>                          |
| Compile and run the application.                   | Compile the developed application and submit it for running.                                                               | <a href="#">Commissioning the Application</a>                   |
| View application running results.                  | Application running results are written into a specified path. You can also view the application running status on the UI. | <a href="#">Commissioning the Application</a>                   |

## 26.2 Environment Preparation

### 26.2.1 Preparing Development and Operating Environment

[Table 26-2](#) describes the environment required for secondary development.

**Table 26-2** Development environment

| Item | Description                                                                                                                                                                                                  |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OS   | Windows OS. Windows 7 or later is supported.<br>If the program needs to be commissioned locally, the development and running environment must be able to communicate with the cluster service plane network. |

| Item                                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| JDK installation                    | <p>Basic configuration for the development and running environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> <li>• For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> <li>- Oracle JDK versions: 1.8</li> <li>- IBM JDK versions: 1.8.5.11</li> </ul> </li> <li>• For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> <li>- OpenJDK: 1.8.0_272</li> </ul> </li> </ul> <p><b>NOTE</b><br/>The server supports only TLS V1.2 or later to ensure security.</p> |
| IDEA installation and configuration | <p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>• If the IBM JDK is used, ensure that the JDK configured in IDEA is the IBM JDK.</li> <li>• If the Oracle JDK is used, ensure that the JDK configured in IDEA is the Oracle JDK.</li> <li>• If the Open JDK is used, ensure that the JDK configured in IDEA is the Open JDK.</li> <li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li> </ul>                                                                                                                                                                                                                                                                                               |
| Maven installation                  | <p>Basic configuration of the development environment for project management throughout the lifecycle of software development.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| User development preparation        | <p>See <a href="#">Preparing a Developer Account</a> for configuration.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 7-zip                               | <p>Used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-zip 16.04 is supported.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

## 26.2.2 Downloading and Importing Sample Projects

### Scenario

Download sample projects and import them to the IDEA on Windows for learning.

## Prerequisites

- A developer account, for example, **developuser**, has been prepared by following instructions in [Preparing Development and Operating Environment](#), and the user authentication credential file has been downloaded to the local host.

The user has the common user permission of Oozie, HDFS access permission, Hive table read and write permission, HBase read and write permission, and Yarn queue submission permission.

- A complete cluster client has been installed in the Linux environment.
- The URL of a running Oozie server (any node) has been obtained. The URL is the target address to which the client submits a workflow job.

The URL format is `https://Oozie node service IP address:21003/oozie`. The port number is the value of **OOZIE\_HTTPS\_PORT**, which is **21003** by default.

for example, `https://10.10.10.176:21003/oozie`.

## Procedure

- Step 1** Obtain the **OozieMapReduceExample**, **OozieSparkHBaseExample**, and **OozieSparkHiveExample** sample projects from the sample project folder **ooziesecurity-examples** in the **src\oozie-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the keytab file **user.keytab** and user authentication credential file **krb5.conf** obtained in [Preparing a Developer Account](#) to the **\src\main\resources** directory of the **OozieMapReduceExample**, **OozieSparkHBaseExample**, and **OozieSparkHiveExample** sample projects.
- Step 3** In an application development environment, import the sample projects to the IDEA development environment.
1. Choose **File > Open**.  
The **Browse** dialog box is displayed.
  2. Select a sample project folder, and click **OK**.
- Step 4** Modify the parameters in the sample project. For details, see [Table 26-3](#).

**Table 26-3** Parameters to be modified

| File Name                                          | Parameter         | Value                                                     | Example Value                                 |
|----------------------------------------------------|-------------------|-----------------------------------------------------------|-----------------------------------------------|
| \src\main<br>\resources<br>\job.properties         | userName          | User who submits a job.                                   | developuser                                   |
| \src\main<br>\resources<br>\application.properties | submit_user       | User who submits a job.                                   | developuser                                   |
|                                                    | oozie_url_default | <code>https://Oozie service IP address:21003/oozie</code> | <code>https://10.10.10.176:21003/oozie</code> |

**Step 5** Select the sample project to be run.

- For the **OozieMapReduceExample** sample project, go to [Step 6](#).
- For the **OozieSparkHBaseExample** and **OozieSparkHiveExample** sample projects, see [Scheduling Spark2x to Access HBase and Hive Using Oozie](#).

**Step 6** Use the client to upload the **examples** folder of Oozie to HDFS.

1. Log in to the node where the client is installed, and switch to the directory of the client, for example **/opt/client**.

```
cd /opt/client
```

2. Run the following command to configure environment variables:

```
source bigdata_env
```

3. Run the following command to perform user authentication. Change the password upon the first login.

```
kinit developuser
```

4. Run the following commands to create a directory in HDFS and upload the sample project to the directory:

```
hdfs dfs -mkdir /user/developuser
```

```
hdfs dfs -put -f /opt/client/Oozie/oozie-client-*/examples /user/
developuser
```

 **NOTE**

In the preceding command, **oozie-client-\*** indicates the version number. Replace it with the actual version number.

----End

## 26.2.3 Preparing Authentication Mechanism Code

### Scenario

In a secure cluster environment, components must perform mutual authentication before communicating with each other to ensure communication security.

In some cases, Oozie needs to communicate with Hadoop and Hive when users develop Oozie applications. Codes for security authentication need to be written into the Oozie applications to ensure that the applications can work properly.

Two security authentication modes are available:

- Command line authentication:

Before running the Oozie applications, run the following command on the Oozie client to obtain authentication:

```
kinit Component service user
```

- Code authentication (Kerberos security authentication):

You can contact the administrator to create and obtain keytab and krb5.conf files used for Kerberos security authentication. For details, see the sample codes.

The sample codes invoke the LoginUtil class for security authentication and support the Oracle JAVA platform and the IBM JAVA platform.

Set **userName** to the actual user name based on the actual situation, for example, **developuser**.

```
private static void login(String keytabFilePath, String krb5FilePath, String user) throws IOException {
 Configuration conf = new Configuration();
 conf.set(KERBEROS_PRINCIPAL, user);
 conf.set(KEYTAB_FILE, keytabFilePath);
 conf.set(HADOOP_SECURITY_AUTHENTICATION, "kerberos");
 conf.set(HADOOP_SECURITY_AUTHORIZATION, "true");

 /*
 * if need to connect zk, please provide jaas info about zk. of course,
 * you can do it as below:
 * System.setProperty("java.security.auth.login.config", confDirPath +
 * "jaas.conf"); but the demo can help you more : Note: if this process
 * will connect more than one zk cluster, the demo may be not proper. you
 * can contact us for more help
 */
 LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, user, keytabFilePath);
 LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);
 LoginUtil.login(user, keytabFilePath, krb5FilePath, conf);
}
```

## 26.3 Developing the Project

### 26.3.1 Development of Configuration Files

#### 26.3.1.1 Description

##### Development Process

1. Configure the **workflow.xml** workflow configuration file (**coordinator.xml** schedules the workflow, and **bundle.xml** manages a pair of Coordinators) and **job.properties**.
2. If you want to implement codes, develop relevant jar packages, for example, Java Action. If you want to use Hive, develop SQL files.
3. Upload the configuration file and jar packages (including dependent jar packages) to the HDFS. The upload path is **oozie.wf.application.path** in **workflow.xml**.
4. The workflow can be implemented by using the following three methods. For details, see [More Information](#).
  - Shell command
  - Java API
  - Hue
5. The Oozie client provides examples for your reference, involving various Actions and how to use Coordinator and Bundle. For example, if the installation directory of the Oozie client is `/opt/client`, the examples directory is `/opt/client/Oozie/oozie-client-*/examples`.

The following example shows you how to configure a configuration file by using the Mapreduce workflow and invoke the configuration file by running the Shell command.

## Description

Provides that a user needs to analyze website logs offline every day, and collect statistics on the access frequency of each module of the website. Log files are stored in the HDFS.

Jobs are submitted through templates and configuration files in the client.

### 26.3.1.2 Development Procedure

#### Step 1 Analyze the service.

1. Analyze and process logs using Mapreduce in the client example directory.
2. Move Mapreduce analysis results to the data analysis result directory, and set the data file access permission to **660**.
3. To analyze data every day, perform [Step 1.1](#) and [Step 1.2](#) every day.

#### Step 2 Implement the service.

1. Log in to the node where the client is located, and create the **dataLoad** directory, for example, **/opt/client/Oozie/oozie-client-\*/examples/apps/dataLoad/**. This directory is used as a program running directory to store files that are edited subsequently.

#### NOTE

You can directly copy the content in the **map-reduce** directory of the example directory to the **dataLoad** directory and edit the content.

Replace **oozie-client-\*** in the directory with the actual version number.

2. Compile a workflow job property file **job.properties**.  
For details, see [job.properties](#).
3. Compile a workflow job using **workflow.xml**.

**Table 26-4** Actions in a Workflow

| No. | Procedure                    | Description                                       |
|-----|------------------------------|---------------------------------------------------|
| 1   | Define the start action.     | For details, see <a href="#">Start Action</a>     |
| 2   | Define the MapReduce action. | For details, see <a href="#">MapReduce Action</a> |
| 3   | Define the FS action.        | For details, see <a href="#">FS Action</a>        |
| 4   | Define the end action.       | For details, see <a href="#">End Action</a>       |
| 5   | Define the kill action.      | For details, see <a href="#">Kill Action</a>      |

#### NOTE

Dependent or newly developed JAR packages must be saved in **dataLoad/lib**.  
The following provides an example workflow file:

```

<workflow-app xmlns="uri:oozie:workflow:1.0" name="data_load">
 <start to="mr-dataLoad"/>
 <action name="mr-dataLoad">
 <map-reduce> <resource-manager>${resourceManager}</resource-manager>
 <name-node>${nameNode}</name-node>
 <prepare>
 <delete path="${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-
reduce"/>
 </prepare>
 <configuration>
 <property>
 <name>mapred.job.queue.name</name>
 <value>${queueName}</value>
 </property>
 <property>
 <name>mapred.mapper.class</name>
 <value>org.apache.oozie.example.SampleMapper</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.apache.oozie.example.SampleReducer</value>
 </property>
 <property>
 <name>mapred.map.tasks</name>
 <value>1</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
 <value>/user/oozie/${dataLoadRoot}/input-data/text</value>
 </property>
 <property>
 <name>mapred.output.dir</name>
 <value>/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce</value>
 </property>
 </configuration>
 </map-reduce>
 <ok to="copyData"/>
 <error to="fail"/>
 </action>

 <action name="copyData">
 <fs>
 <delete path='${nameNode}/user/oozie/${dataLoadRoot}/result'/>
 <move source='${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce'
target='${nameNode}/user/oozie/${dataLoadRoot}/result'/>
 <chmod path='${nameNode}/user/oozie/${dataLoadRoot}/result' permissions='-rwxrw-rw-'
dir-files='true'/></chmod>
 </fs>
 <ok to="end"/>
 <error to="fail"/>
 </action>

 <kill name="fail">
 <message>This workflow failed, error message[${wf.errorMessage(wf.lastErrorNode())}]</
message>
 </kill>
 <end name="end"/>
</workflow-app>

```

#### 4. Compile a Coordinator job using **coordinator.xml**.

The Coordinator job is used to analyze data every day. For details, see [coordinator.xml](#).

### Step 3 Upload the workflow file.

1. Use or switch to the user account that is granted with rights to upload files to the HDFS. For details about developer account preparation, see [Preparing Development and Operating Environment](#).

2. Implement Kerberos authentication for the user account.
3. Run the HDFS upload command to upload the **dataLoad** folder to a specified directory on the HDFS (user **developuser** must have the read/write permission for the directory).

 **NOTE**

The specified directory must be the same as **oozie.coord.application.path** and **workflowAppUri** defined in **job.properties**.

**Step 4** Execute the workflow file.

1. Log in to the client node, implement Kerberos authentication for user **developuser**.

```
cd /opt/client
source bigdata_env
kinit developuser
```

2. Run the following command to start the workflow:

Command:

```
oozie job -oozie https://oozie server hostname:port/oozie -config
job.propertiesfile path -run
```

Parameter list:

**Table 26-5** Parameters

| Parameter | Description                                        |
|-----------|----------------------------------------------------|
| job       | Indicates that a job is to be executed.            |
| -oozie    | Indicates the (any instance) Oozie server address. |
| -config   | Indicates the path of <b>job.properties</b> .      |
| -run      | Indicates the starts workflow.                     |

For example:

```
oozie job -oozie https://10-1-130-10:21003/oozie -config job.properties -run
----End
```

## 26.3.2 Example Codes

### 26.3.2.1 job.properties

#### Function

**job.properties** is a workflow property file that defines external parameters used for workflow execution.



## Parameter Description

**Table 26-6** describes parameters in `job.properties`.

**Table 26-6** Parameters

| Parameter                                 | Meaning                                                                       |
|-------------------------------------------|-------------------------------------------------------------------------------|
| <code>nameNode</code>                     | Indicates the Hadoop distributed file system (HDFS) NameNode cluster address. |
| <code>resourceManager</code>              | Indicates the Yarn ResourceManager address.                                   |
| <code>queueName</code>                    | Identifies the MapReduce queue where a workflow job is executed.              |
| <code>dataLoadRoot</code>                 | Identifies the folder where the workflow job resides.                         |
| <code>oozie.coord.application.path</code> | Indicates the storage path of a Coordinator job in the HDFS.                  |
| <code>start</code>                        | Indicates the time when a scheduled workflow job is started.                  |
| <code>end</code>                          | Indicates the time when a scheduled workflow job is stopped.                  |
| <code>workflowAppUri</code>               | Indicates the storage path of a workflow job in the HDFS.                     |

### NOTE

You can define parameters in the key=value format based on service requirements.

## Example Codes

```
nameNode=hdfs://hacluster
resourceManager=10.1.130.10:26004
queueName=QueueA
dataLoadRoot=examples

oozie.coord.application.path=${nameNode}/user/oozie_cli/${dataLoadRoot}/apps/dataLoad
start=2013-04-02T00:00Z
end=2014-04-02T00:00Z
workflowAppUri=${nameNode}/user/oozie_cli/${dataLoadRoot}/apps/dataLoad
```

### 26.3.2.2 workflow.xml

#### Function

**workflow.xml** describes a complete service workflow. A workflow consists of a start node, an end node, and multiple action nodes.

## Parameter Description

[Table 26-7](#) describes parameters in `workflow.xml`.

**Table 26-7** Parameters

| Parameter | Meaning                                                                 |
|-----------|-------------------------------------------------------------------------|
| name      | Identifies a workflow file.                                             |
| start     | Indicates the workflow start node.                                      |
| end       | Indicates the workflow end node.                                        |
| action    | Indicates nodes (one or multiple) that are used to implement a service. |

## Example Codes

```
<workflow-app xmlns="uri:oozie:workflow:1.0" name="data_load">
 <start to="copyData"/>
 <action name="copyData">
 </action>

 <end name="end"/>
</workflow-app>
```

### 26.3.2.3 Start Action

## Function

The Start Action node indicates the start point of a workflow job. Each workflow job has only one Start Action node.

## Parameter Description

[Table 26-8](#) describes the parameter used on the Start Action node.

**Table 26-8** Parameters

Parameter	Meaning
to	Identifies a subsequent action node.

## Example Codes

```
<start to="mr-dataLoad"/>
```

### 26.3.2.4 End Action

#### Function

The End Action node indicates the end point of a workflow job. Each workflow job has only one End Action node.

#### Parameter Description

[Table 26-9](#) describes the parameter used on the End Action node.

**Table 26-9** Parameters

Parameter	Meaning
name	Identifies an end action.

#### Example Codes

```
<end name="end"/>
```

### 26.3.2.5 Kill Action

#### Function

The Kill Action node indicates the end point of a workflow job when an error occurs.

#### Parameter Description

[Table 26-10](#) describes parameters used on the Kill Action node.

**Table 26-10** Parameters

Parameter	Meaning
name	Identifies a kill action.
message	Provides error messages.
\$ {wf:errorMessage(wf:lastErrorNode())}	Indicates the internal error message function in the Oozie system.

#### Example Codes

```
<kill name="fail">
 <message>
 This workflow failed, error message[{$wf:errorMessage(wf:lastErrorNode())}]
 </message>
</kill>
```

### 26.3.2.6 FS Action

#### Function

The FS Action node is a Hadoop distributed file system (HDFS) operation node. You can create and delete HDFS files and folders and grant permissions for HDFS files and folders using this node.

#### Parameter Description

[Table 26-11](#) describes parameters used on the FS Action node.

**Table 26-11** Parameters

Parameter	Meaning
name	Identifies an FS action.
delete	Deletes a specified file or folder.
move	Moves a file from the source directory to the target directory.
chmod	Modifies file or folder access rights.
path	Indicates the current file path.
source	Indicates the source file path.
target	Indicates the target file path.
permissions	Indicates permissions.

#### NOTE

**`${variable name}`** indicates the value defined in **job.properties**.

For example, **`/${nameNode}`** indicates **hdfs://hacluster**. (See [job.properties](#).)

#### Example Codes

```
<action name="copyData">
 <fs>
 <delete path=${nameNode}/user/oozie_cli/${dataLoadRoot}/result'/>
 <move source=${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce' target=${nameNode}/user/oozie_cli/${dataLoadRoot}/result'/>
 <chmod path=${nameNode}/user/oozie_cli/${dataLoadRoot}/reuslt' permissions='-rwxrw-rw-' dir-files='true'></chmod>
 </fs>
 <ok to="end"/>
 <error to="fail"/>
</action>
```

## 26.3.2.7 MapReduce Action

### Function

The MapReduce Action node is used to execute a map-reduce job.

### Parameter Description

[Table 26-12](#) describes parameters used on the MapReduce Action node.

**Table 26-12** Parameters

Parameter	Meaning
name	Identifies a map-reduce action.
resourceManager	Indicates the MapReduce ResourceManager address.
name-node	Indicates the Hadoop distributed file system (HDFS) NameNode address.
queueName	Identifies the MapReduce queue where a job is executed.
mapred.mapper.class	Identifies the Mapper class.
mapred.reducer.class	Identifies the Reducer class.
mapred.input.dir	Indicates the input directory of MapReduce processed data.
mapred.output.dir	Indicates the output directory of MapReduce processing results.
mapred.map.tasks	Indicates the number of map tasks.

#### NOTE

**`${variable name}`** indicates the value defined in **job.properties**.

For example, **`${nameNode}`** indicates **hdfs://hacluster**. (See [job.properties](#).)

### Example Codes

```
<action name="mr-dataLoad">
 <map-reduce>
 <resource-manager>${resourceManager}</resource-manager>
 <name-node>${nameNode}</name-node>
 <prepare>
 <delete path="${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce"/>
 </prepare>
 <configuration>
 <property>
 <name>mapred.job.queue.name</name>
 <value>${queueName}</value>
 </property>
 </configuration>
 </map-reduce>
</action>
```

```

 <name>mapred.mapper.class</name>
 <value>org.apache.oozie.example.SampleMapper</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.apache.oozie.example.SampleReducer</value>
 </property>
 <property>
 <name>mapred.map.tasks</name>
 <value>1</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
 <value>/user/oozie/${dataLoadRoot}/input-data/text</value>
 </property>
 <property>
 <name>mapred.output.dir</name>
 <value>/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce</value>
 </property>
</configuration>
</map-reduce>
<ok to="copyData"/>
<error to="fail"/>
</action>

```

### 26.3.2.8 coordinator.xml

#### Function

**coordinator.xml** is used to periodically execute a workflow job.

#### Parameter Description

[Table 26-13](#) describes parameters in **coordinator.xml**.

**Table 26-13** Parameters

Parameter	Meaning
frequency	Indicates the workflow execution interval.
start	Indicates the time when a scheduled workflow job is started.
end	Indicates the time when a scheduled workflow job is stopped.
workflowAppUri	Indicates the storage path of a workflow job in the HDFS.
resourceManager	Indicates the MapReduce ResourceManager address.
queueName	Identifies the MapReduce queue where a job is executed.
nameNode	Indicates the Hadoop distributed file system (HDFS) NameNode address.

 NOTE

**`${variable name}`** indicates the value defined in **job.properties**.

For example, **`${nameNode}`** indicates **hdfs://hacluster**. (See [job.properties](#).)

## Example Codes

```
<coordinator-app name="cron-coord" frequency="${coord:days(1)}" start="${start}" end="${end}"
timezone="UTC" xmlns="uri:oozie:coordinator:0.2">
 <action>
 <workflow>
 <app-path>${workflowAppUri}</app-path>
 <configuration>
 <property>
 <name>resourceManager</name>
 <value>${resourceManager}</value>
 </property>
 <property>
 <name>nameNode</name>
 <value>${nameNode}</value>
 </property>
 <property>
 <name>queueName</name>
 <value>${queueName}</value>
 </property>
 </configuration>
 </workflow>
 </action>
</coordinator-app>
```

## 26.3.3 Development of Java

### 26.3.3.1 Description

These typical scenarios help you quickly understand the development procedure of Oozie and learn key API functions.

This example shows you how to submit a MapReduce job and query the job status by using the Java API. The sample code relates to the MapReduce job only, but the API invocation codes of other jobs are the same. The difference is that the configuration of **job.properties** in a job and that of **workflow.xml** in a workflow is different.

 NOTE

After the operations in [Downloading and Importing Sample Projects](#) are complete, you can submit MapReduce jobs and query job status through Java APIs.

### 26.3.3.2 Sample Code

#### Function

Oozie submits a job from the run method of **org.apache.oozie.client.OozieClient** and obtains job information from **getJobInfo**.

#### Sample Code

Change **OOZIE\_URL\_DEFALUT** in the code example to the actual host name of any Oozie node, for example, <https://10-1-131-131:21003/oozie/>.

```

public void test(String jobFilePath) {
 try {
 UserGroupInformation.getLoginUser()
 .doAs(
 new PrivilegedExceptionAction<Void>() {
 /**
 * run job
 *
 * @return null
 * @throws Exception exception
 */
 public Void run() throws Exception {
 runJob(jobFilePath);
 return null;
 }
 }
);
 } catch (Exception e) {
 e.printStackTrace();
 }
}

private void runJob(String jobFilePath) throws OozieClientException, InterruptedException {

 Properties conf = getJobProperties(jobFilePath);

 // submit and start the workflow job
 String jobId = wc.run(conf);

 logger.info("Workflow job submitted : {}" , jobId);

 // wait until the workflow job finishes printing the status every 10 seconds
 while (wc.getJobInfo(jobId).getStatus() == WorkflowJob.Status.RUNNING) {
 logger.info("Workflow job running ... {}" , jobId);
 Thread.sleep(10 * 1000);
 }

 // print the final status of the workflow job
 logger.info("Workflow job completed ... {}" , jobId);
 logger.info(String.valueOf(wc.getJobInfo(jobId)));
}

/**
 * Get job.properties File in filePath
 *
 * @param filePath file path
 * @return job.properties
 * @since 2020-09-30
 */
public Properties getJobProperties(String filePath) {
 File configFile = new File(filePath);
 if (!configFile.exists()) {
 logger.info(filePath , "{} is not exist.");
 }

 InputStream inputStream = null;

 // create a workflow job configuration
 Properties properties = wc.createConfiguration();
 try {
 inputStream = new FileInputStream(filePath);
 properties.load(inputStream);
 } catch (Exception e) {
 e.printStackTrace();
 } finally {
 if (inputStream != null) {
 try {
 inputStream.close();
 } catch (IOException e) {

```



```

 e.printStackTrace();
 }
}
return properties;
}

```

## Precautions

Implement the security authentication when you use the Java API to access the Oozie. For details, see section "Preparing for the Development Environment". Upload the dependent configuration file (For details about how to develop the **Workflow.xml** configuration file, see [workflow.xml](#)) and the jar package to the HDFS, and ensure that users who have passed the security authentication are granted the rights to access the relevant directory on the HDFS. (The owner of the directory is the authenticated users, or is in the same user group with the users).

## 26.3.4 Scheduling Spark2x to Access HBase and Hive Using Oozie

### Prerequisites

Prerequisites in [Downloading and Importing Sample Projects](#) have been met.

### Preparing a Development Environment

- Step 1** Obtain the **OozieSparkHBaseExample** and **OozieSparkHiveExample** sample projects from the sample project folder **ooziesecurity-examples** in the **src\oozie-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the keytab file **user.keytab** and user authentication credential file **krb5.conf** obtained in [Preparing a Developer Account](#) to the **\src\main\resources** directory of the **OozieSparkHBaseExample** and **OozieSparkHiveExample** sample projects.
- Step 3** Modify the parameters in each sample project. For details, see [Table 26-14](#).

**Table 26-14** Parameters to be modified

File Name	Parameter	Value	Example Value
src\main \resources \application.properties	submit_user	User who submits a job.	developuser
	oozie_url_default	<i>https://Oozie service IP address.21003/oozie/</i>	https://10.10.10.176:21003/oozie/
src\main \resources \job.properties	userName	User who submits a job.	developuser

File Name	Parameter	Value	Example Value
	examplesRoot	Use the default value or change the value based on the site requirements.	myjobs
	oozie.wf.application.path	Use the default value or change the value based on the site requirements.	\${nameNode}/user/\${userName}/\${examplesRoot}/apps/spark2x <b>NOTICE</b> Ensure that the path is the same as the path with the <code>&lt;jar&gt;</code> and <code>&lt;spark-opts&gt;</code> tags in the <code>src/main/resources/workflow.xml</code> file.
src/main/resources/workflow.xml	<jar> </jar>	Change <b>OoizeSparkHBase-1.0.jar</b> to the actual JAR package name.	<jar>\${nameNode}/user/\${userName}/\${examplesRoot}/apps/spark2x/lib/OoizeSparkHBase-1.0.jar</jar>

 **NOTE**

Go to the root directory of the project, for example, `D:\sample_project\src\oozie-examples\ooziesecurity-examples\OoizeSparkHBaseExample`, and run the `mvn clean package -DskipTests` command. After the operation is successful, the package is in the target directory.

**Step 4** Create the following folders on the HDFS client in the configured path:

```
hdfs dfs -mkdir -p /user/developuser/myjobs/apps/spark2x/lib
```

```
hdfs dfs -mkdir -p /user/developuser/myjobs/apps/spark2x/hbase
```

```
hdfs dfs -mkdir -p /user/developuser/myjobs/apps/spark2x/hive
```

**Step 5** Upload the files listed in [Table 26-15](#) to the corresponding path.

**Table 26-15** Files to be uploaded

Initial File Path	File	Destination Path
Spark client directory (for example, <code>/opt/client/Spark2x/spark/conf</code> )	hive-site.xml	<code>/user/developuser/myjobs/apps/spark2x</code> directory in the HDFS.
	hbase-site.xml	

Initial File Path	File	Destination Path
Keytab file obtained in <a href="#">Preparing a Developer Account</a>	user.keytab	
	krb5.conf	
Spark client directory (for example, <code>/opt/client/Spark2x/spark/jars</code> )	JAR package	Share HDFS <code>/user/oozie/share/lib/spark2x/</code> directory of Oozie.
JAR package of the sample projects to be used, for example, <b>OoizeSparkHBase-1.0.jar</b>	JAR package	<code>/user/developuser/myjobs/apps/spark2x/lib/</code> directory in the HDFS.
OozieSparkHiveExample sample project directory <code>src\main\resources</code>	workflow.xml	<code>/user/developuser/myjobs/apps/spark2x/hive/</code> directory in the HDFS. <b>NOTE</b> Change the path of <code>spark-archive-2x.zip</code> in <code>&lt;spark-opts&gt;</code> based on the actual HDFS file path.
OozieSparkHBaseExample sample project directory <code>src\main\resources</code>	workflow.xml	<code>/user/developuser/myjobs/apps/spark2x/hbase/</code> directory in the HDFS. <b>NOTE</b> Change the path of <code>spark-archive-2x.zip</code> in <code>&lt;spark-opts&gt;</code> based on the actual HDFS file path.

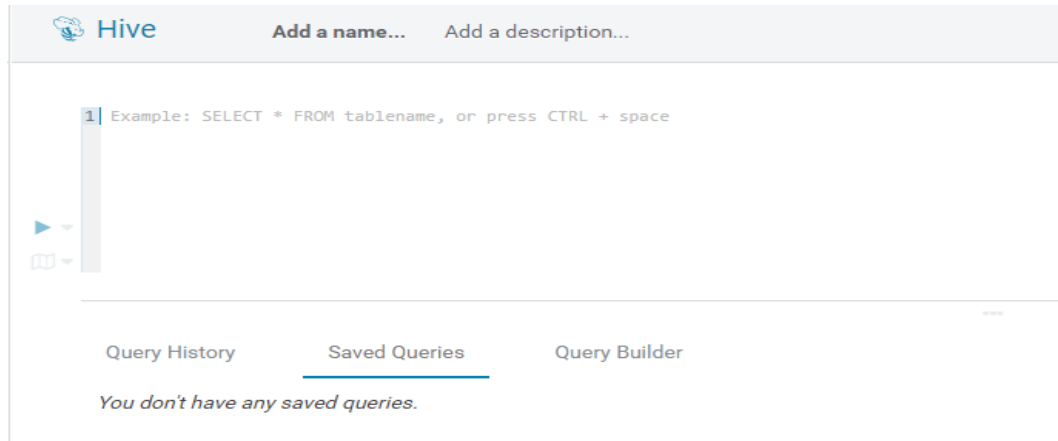
**Step 6** Change the value of `hive.security.authenticator.manager` in the `hive-site.xml` file in the `/user/developuser/myjobs/apps/spark2x` directory of HDFS from `org.apache.hadoop.hive.ql.security.SessionStateUserMSGroupAuthenticator` to `org.apache.hadoop.hive.ql.security.SessionStateUserGroupAuthenticator`.

**Step 7** If ZooKeeper SSL is enabled, add the following content to the `hbase-site.xml` file in the `/user/developuser/myjobs/apps/spark2x` directory of the HDFS:

```
<property>
<name>HBASE_ZK_SSL_ENABLED</name>
<value>true</value>
</property>
```

**Step 8** Run the following commands to create a Hive table:

You can enter the following SQL statements in the Hive panel on the Hue UI:



```
CREATE DATABASE test;
```

```
CREATE TABLE IF NOT EXISTS `test`.`usr` (user_id int comment
'userID',user_name string comment 'userName',age int comment
'age')PARTITIONED BY (country string)STORED AS PARQUET;
```

```
CREATE TABLE IF NOT EXISTS `test`.`usr2` (user_id int comment
'userID',user_name string comment 'userName',age int comment
'age')PARTITIONED BY (country string)STORED AS PARQUET;
```

```
INSERT INTO TABLE test.usr partition(country='CN') VALUES(1,'maxwell',45),
(2,'minwell',30),(3,'mike',22);
```

```
INSERT INTO TABLE test.usr partition(country='USA') VALUES(4,'minbin',35);
```

**Step 9** Use HBase Shell to run the following commands to create an HBase table:

```
create 'SparkHBase',{NAME=>'cf1'}
put 'SparkHBase','01','cf1:name','Max'
put 'SparkHBase','01','cf1:age','23'
----End
```

## 26.4 Commissioning the Application

### 26.4.1 Commissioning an Application in the Windows Environment

#### 26.4.1.1 Compiling and Running Applications

##### Scenario

After the code development is complete using Java APIs, you can run applications in the Windows development environment. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host.

## Procedure

- The HTTPS SSL certificate is required for running applications in Windows.
  - a. Log in to any node in the cluster and go to the following directory to download the **ca.crt** file.

```
cd ${BIGDATA_HOME}/om-agent_8.1.0.1/nodeagent/security/cert/
subcert/certFile/
```
  - b. Download the **ca.crt** file to a local directory and open the CLI as the administrator.  
Run the following command:

```
keytool -import -v -trustcacerts -alias ca -file "D:\xx\ca.crt" -storepass
changeit -keystore "%JAVA_HOME%\jre\lib\security\cacerts"
```

In the preceding command, **D:\xx\ca.crt** is the path for storing the **ca.crt** file. **%JAVA\_HOME %** indicates the JDK installation path.
  - c. In the development environment (such as IDEA), right-click **OozieRestApiMain.java**, and choose **Run 'OozieRestApiMain.main()'** to run the application project.
- Run the following command on the Oozie client:

```
oozie job -oozie https://Oozie service IP:21003/oozie -config job.properties
-run
```

You need to copy the **job.properties** file in the **src\main\resources** directory of the sample project to the directory where the Oozie client is located in advance.

### 26.4.1.2 Checking the Commissioning Result

#### Scenario

The results can be viewed on the console after the Oozie sample project is completed.

#### Procedure

The following information is displayed if the sample project is successful:

```
log4j:WARN No appenders could be found for logger (com.huawei.hadoop.security.LoginUtil).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/D:/temp/newClientSec/oozie-example/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/
impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/D:/temp/newClientSec/oozie-example/lib/slf4j-simple-1.7.1.jar!/org/slf4j/
impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
current user is developuser@<system domain name> (auth:KERBEROS)
login user is developuser@<system domain name> (auth:KERBEROS)
cluset status is true
Warning: Could not get charToByteConverterClass!
Workflow job submitted: 0000071-160729120057089-oozie-omm-W
Workflow job running ...0000071-160729120057089-oozie-omm-W
Workflow job running ...0000071-160729120057089-oozie-omm-W
Workflow job running ...0000071-160729120057089-oozie-omm-W
Workflow job running ...0000071-160729120057089-oozie-omm-W
Workflow job running ...0000071-160729120057089-oozie-omm-W
```

```
Workflow job running ...0000071-160729120057089-oozie-omm-W
Workflow job running ...0000071-160729120057089-oozie-omm-W
Workflow job running ...0000071-160729120057089-oozie-omm-W
Workflow job running ...0000071-160729120057089-oozie-omm-W
Workflow job completed ...0000071-160729120057089-oozie-omm-W
Workflow id[0000071-160729120057089-oozie-omm-W] status[SUCCEEDED]
-----finish Oozie -----
```

Directory **/user/developuser/examples/output-data/map-reduce** is generated on the HDFS. The directory contains the following two files:

- `_SUCCESS`
- `part-00000`

You can view the files by using the file browser of the Hue or running the following commands on the HDFS:

```
hdfs dfs -ls /user/developuser/examples/output-data/map-reduce
```

 **NOTE**

In the Windows environment, the following exception may occur but does not affect services.

```
java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.
```

## 26.5 More Information

### 26.5.1 Common API Introduce

#### 26.5.1.1 Shell

**Table 26-16** Interface parameters

Command	Parameter	Meaning
oozie version	-	The version information
oozie job	-config <arg>	Indicates the path to the job configuration file <b>job.properties</b> .
	-oozie <arg>	Indicates the Oozie server address.
	-haconfig <arg>	Indicates the path to the Oozie service configuration file <b>oozie-site.xml</b> .
	-run	Runs a job.
	-start <arg>	Starts a specified job.
	-submit	Submits a job.

Command	Parameter	Meaning
	-kill <arg>	Deletes a specified job.
	-suspend <arg>	Suspends a specified job.
	-resume <arg>	Resumes a specified job.
	-D <property=value>	Sets a property.
oozie admin	-oozie <arg>	Indicates the Oozie server address.
	-status	Indicates the service status of the Oozie service.

The Oozie command and other parameters can be found in the following address:  
[https://oozie.apache.org/docs/5.1.0/DG\\_CommandLineTool.html](https://oozie.apache.org/docs/5.1.0/DG_CommandLineTool.html).

### 26.5.1.2 Java

Java APIs are provided by **org.apache.oozie.client.OozieClient**.

**Table 26-17** API

Item	Description
public String run(Properties conf)	Runs a job.
public void start(String jobId)	Starts the specified job.
public String submit(Properties conf)	Submits a job.
public void kill(String jobId)	Delete the specified job.
public void suspend(String jobId)	Suspends the specified job.
public void resume(String jobId)	Resumes the specified job.
public WorkflowJob getJobInfo(String jobId)	Obtains job information.

### 26.5.1.3 REST

The common APIs of REST are the same as the APIs of Java. For details, see  
<http://oozie.apache.org/docs/5.1.0/WebServicesAPI.html>.

# 27 Oozie Development Guide (Normal Mode)

---

## 27.1 Overview

### 27.1.1 Application Development Overview

#### Oozie Introduction

Oozie is a workflow engine used to manage Hadoop jobs. Oozie workflows are defined and described based on the directed acyclic graph (DAG). Oozie supports multiple types of workflow modes and workflow scheduled triggering mechanisms. Oozie features easy extensibility, convenient maintenance, and high reliability and works closely with each component in the Hadoop ecosystem.

Oozie workflows are classified into three types:

- Workflow  
Provides a complete basic service workflow.
- Coordinator  
Built on workflows, triggers workflows in a scheduled manner or based on conditions.
- Bundle  
Built on coordinators, centrally schedules, controls, and manages coordinators.

Oozie provides the following features:

- Supports distribution, merging, and choice workflow modes.
- Works closely with each component in the Hadoop ecosystem.
- Supports parameterized workflow variables.
- Supports scheduled workflow triggering.
- Provides a web console that allows users to view and monitor workflows and view logs.



## 27.1.2 Common Concepts

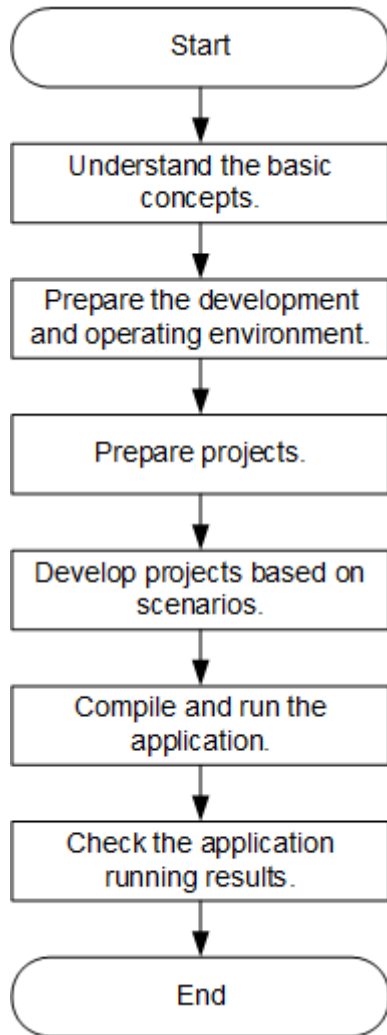
- **Workflow definition file**  
Workflow definition files refer to XML files that describe service logic, including **workflow.xml**, **coordinator.xml**, and **bundle.xml**. Workflow definition files are parsed and executed by the Oozie engine.
- **Workflow property file**  
The workflow property file refers to the parameter configuration file named **job.properties** for workflow execution. Each workflow has only one property file.
- **Client**  
Users can access the server from the client through the Java API, Shell API, REST API, or WebUI to access the Oozie server. The client in this document includes the example code used for accessing Oozie through the Java API.
- **Oozie WebUI**  
Log in to the Oozie WebUI via **https://Oozie server IP address:21003/oozie**.

## 27.1.3 Development Process

This document describes Oozie application development based on the Java API.

For information about each phase in the development process, see [Figure 27-1](#) and [Table 27-1](#).

**Figure 27-1** Oozie application development process



**Table 27-1** Description of Oozie development process

Phase	Description	Reference Document
Understand the basic concepts.	Before developing an application, learn basic concepts of Oozie and understand the scenario requirements and design tables.	<a href="#">Common Concepts</a>

Phase	Description	Reference Document
Prepare the development and operating environment.	The Java language is recommended for the development of Oozie applications. The IDEA tool can be used.	<a href="#">Preparing for Development and Operating Environment</a>
Prepare projects.	Oozie provides example projects for different scenarios. You can import an example project to learn the application.	<a href="#">Downloading and Importing Sample Projects</a>
Develop projects based on scenarios.	An example project based on the Java language is provided.	<a href="#">Developing the Project</a>
Compile and run the application.	Compile the developed application and submit it for running.	<a href="#">Commissioning the Application</a>
View application running results.	Application running results are written into a specified path. You can also view the application running status on the UI.	<a href="#">Commissioning the Application</a>

## 27.2 Environment Preparation

### 27.2.1 Preparing for Development and Operating Environment

[Table 27-2](#) describes the environment required for secondary development.

**Table 27-2** Development environment

Item	Description
OS	<p>Windows OS. Windows 7 or later is supported.</p> <p>If the program needs to be commissioned locally, the development and running environment must be able to communicate with the cluster service plane network.</p>
JDK installation	<p>Basic configuration for the development and running environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> <li>• For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> <li>- Oracle JDK versions: 1.8</li> <li>- Supported IBM JDK versions: 1.8.5.11</li> </ul> </li> <li>• For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> <li>- OpenJDK: 1.8.0_272</li> </ul> </li> </ul> <p><b>NOTE</b> The server supports only TLS V1.2 or later to ensure security.</p>
IDEA installation and configuration	<p>configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>• If the IBM JDK is used, ensure that the JDK configured in IDEA is the IBM JDK.</li> <li>• If the Oracle JDK is used, ensure that the JDK configured in IDEA is the Oracle JDK.</li> <li>• If the Open JDK is used, ensure that the JDK configured in IDEA is the Open JDK.</li> <li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li> </ul>
Maven installation	<p>Basic configuration of the development environment for project management throughout the lifecycle of software development.</p>
7-zip	<p>Used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-zip 16.04 is supported.</p>

## 27.2.2 Downloading and Importing Sample Projects

### Scenario

Download sample projects and import them to the IDEA on Windows for learning.

## Prerequisites

- A complete client has been installed in the Linux environment.
- The URL of a running Oozie server (any node) has been obtained. The URL is the target address to which the client submits a workflow job.

The URL format is `https://Oozie service IP address:21003/oozie`, for example, `https://10.10.10.176:21003/oozie`.

## Procedure

- Step 1** Obtain the **OozieMapReduceExample**, **OozieSparkHBaseExample**, and **OozieSparkHiveExample** sample projects from the sample project folder **oozinormal-examples** in the **src\oozie-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In an application development environment, import the sample projects to the IDEA development environment.
1. Choose **File > Open**.  
The **Browse** dialog box is displayed.
  2. Select a sample project folder, and click **OK**.
- Step 3** Modify the parameters in the sample project. For details, see [Table 27-3](#).

**Table 27-3** Parameters to be modified

File Name	Parameter	Value	Example Value
\src\main \resources \job.properties	userName	User who submits a job.	developuser
\src\main \resources \application.properties	submit_user	User who submits a job.	developuser
	oozie_url_default	https://Oozie service IP address:21003/oozie	https://10.10.10.176:21003/oozie

- Step 4** Select the sample project to be run.
- For the **OozieMapReduceExample** sample project, go to [Step 5](#).
  - For the **OozieSparkHBaseExample** and **OozieSparkHiveExample** sample projects, see [Scheduling Spark2x to Access HBase and Hive Using Oozie](#).
- Step 5** Use the client to upload the example files of Oozie to HDFS.
1. Log in to the node where the client is installed, and switch to the directory of the client, for example **/opt/client**.  
**cd /opt/client**
  2. Run the following command to configure environment variables:  
**source bigdata\_env**

3. Run the following commands to create a directory in HDFS and upload the sample project to the directory:

```
hdfs dfs -mkdir /user/developuser
```

```
hdfs dfs -put -f /opt/client/Oozie/oozie-client-*/examples /user/
developuser
```

 NOTE

In the preceding command, **oozie-client-\*** indicates the version number. Replace it with the actual version number.

----End

## 27.3 Developing the Project

### 27.3.1 Development of Configuration Files

#### 27.3.1.1 Description

##### Development Process

1. Configure the **workflow.xml** workflow configuration file (**coordinator.xml** schedules the workflow, and **bundle.xml** manages a pair of Coordinators) and **job.properties**.
2. If you want to implement codes, develop relevant jar packages, for example, Java Action. If you want to use Hive, develop SQL files.
3. Upload the configuration file and jar packages (including dependent jar packages) to the HDFS. The upload path is **oozie.wf.application.path** in **workflow.xml**.
4. The workflow can be implemented by using the following three methods. For details, see [More Information](#).
  - Shell command
  - Java API
  - Hue
5. The Oozie client provides examples for your reference, involving various Actions and how to use Coordinator and Bundle. For example, if the installation directory of the Oozie client is **/opt/client**, the examples directory is **/opt/client/Oozie/oozie-client-\*/examples**.

The following example shows you how to configure a configuration file by using the Mapreduce workflow and invoke the configuration file by running the Shell command.

##### Description

Provides that a user needs to analyze website logs offline every day, and collect statistics on the access frequency of each module of the website. Log files are stored in the HDFS.

Jobs are submitted through templates and configuration files in the client.

### 27.3.1.2 Development Procedure

**Step 1** Analyze the service.

1. Analyze and process logs using Mapreduce in the client example directory.
2. Move Mapreduce analysis results to the data analysis result directory, and set the data file access permission to **660**.
3. To analyze data every day, perform [Step 1.1](#) and [Step 1.2](#) every day.

**Step 2** Implement the service.

1. Log in to the node where the client is located, and create the **dataLoad** directory, for example, **/opt/client/Oozie/oozie-client-\*/examples/apps/dataLoad/**. This directory is used as a program running directory to store files that are edited subsequently.

 **NOTE**

You can directly copy the content in the **map-reduce** directory of the example directory to the **dataLoad** directory and edit the content.

Replace **oozie-client-\*** in the directory with the actual version number.

2. Compile a workflow job property file **job.properties**.  
For details, see [job.properties](#).
3. Compile a workflow job using **workflow.xml**.

**Table 27-4** Actions in a Workflow

No.	Procedure	Description
1	Define the startaction.	For details, see <a href="#">Start Action</a>
2	Define the MapReduceaction.	For details, see <a href="#">MapReduce Action</a>
3	Define the FS action.	For details, see <a href="#">FS Action</a>
4	Define the end action.	For details, see <a href="#">End Action</a>
5	Define the killaction.	For details, see <a href="#">Kill Action</a>

 **NOTE**

Dependent or newly developed JAR packages must be saved in **dataLoad/lib**.

The following provides an example workflow file:

```
<workflow-app xmlns="uri:oozie:workflow:1.0" name="data_load">
 <start to="mr-dataLoad"/>
 <action name="mr-dataLoad">
 <map-reduce><resource-manager>${resourceManager}</resource-manager>
 <name-node>${nameNode}</name-node>
 <prepare>
```

```

 <delete path="${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-
reduce"/>
 </prepare>
 <configuration>
 <property>
 <name>mapred.job.queue.name</name>
 <value>${queueName}</value>
 </property>
 <property>
 <name>mapred.mapper.class</name>
 <value>org.apache.oozie.example.SampleMapper</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.apache.oozie.example.SampleReducer</value>
 </property>
 <property>
 <name>mapred.map.tasks</name>
 <value>1</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
 <value>/user/oozie/${dataLoadRoot}/input-data/text</value>
 </property>
 <property>
 <name>mapred.output.dir</name>
 <value>/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce</value>
 </property>
 </configuration>
</map-reduce>
<ok to="copyData"/>
<error to="fail"/>
</action>

<action name="copyData">
 <fs>
 <delete path="${nameNode}/user/oozie/${dataLoadRoot}/result"/>
 <move source="${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce'
target="${nameNode}/user/oozie/${dataLoadRoot}/result"/>
 <chmod path="${nameNode}/user/oozie/${dataLoadRoot}/result' permissions='-rwxrw-rw-'
dir-files='true'></chmod>
 </fs>
 <ok to="end"/>
 <error to="fail"/>
</action>

<kill name="fail">
 <message>This workflow failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</
message>
</kill>
<end name="end"/>
</workflow-app>

```

4. Compile a Coordinator job using **coordinator.xml**.

The Coordinator job is used to analyze data every day. For details, see [coordinator.xml](#).

**Step 3** Upload the workflow file.

1. Use or switch to the user account that is granted with rights to upload files to the HDFS.
2. Run the HDFS upload command to upload the **dataLoad** folder to a specified directory on the HDFS (user **oozie\_cli** must have the read/write permission for the directory).



 NOTE

The specified directory must be the same as **oozie.coord.application.path** and **workflowAppUri** defined in **job.properties**.

**Step 4** Execute the workflow file.

Command:

```
oozie job -oozie https://oozie server hostname:port/oozie -config
job.propertiesfile path -run
```

Parameter list:

**Table 27-5** Parameters

Parameter	Description
job	Indicates that a job is to be executed.
-oozie	Indicates the (any instance) Oozie server address.
-config	Indicates the path of <b>job.properties</b> .
-run	Indicates the starts workflow.

For example:

```
oozie job -oozie https://10-1-130-10:21003/oozie -config job.properties -run
----End
```

## 27.3.2 Example Codes

### 27.3.2.1 job.properties

#### Function

**job.properties** is a workflow property file that defines external parameters used for workflow execution.

#### Parameter Description

[Table 27-6](#) describes parameters in **job.properties**.

**Table 27-6** Parameters

Parameter	Meaning
nameNode	Indicates the Hadoop distributed file system (HDFS) NameNode cluster address.

Parameter	Meaning
resourceManager	Indicates the Yarn ResourceManager address.
queueName	Identifies the MapReduce queue where a workflow job is executed.
dataLoadRoot	Identifies the folder where the workflow job resides.
oozie.coord.application.path	Indicates the storage path of a Coordinator job in the HDFS.
start	Indicates the time when a scheduled workflow job is started.
end	Indicates the time when a scheduled workflow job is stopped.
workflowAppUri	Indicates the storage path of a workflow job in the HDFS.

 **NOTE**

You can define parameters in the key=value format based on service requirements.

## Example Codes

```
nameNode=hdfs://hacluster
resourceManager=10.1.130.10:26004
queueName=QueueA
dataLoadRoot=examples

oozie.coord.application.path=${nameNode}/user/oozie_cli/${dataLoadRoot}/apps/dataLoad
start=2013-04-02T00:00Z
end=2014-04-02T00:00Z
workflowAppUri=${nameNode}/user/oozie_cli/${dataLoadRoot}/apps/dataLoad
```

### 27.3.2.2 workflow.xml

#### Function

**workflow.xml** describes a complete service workflow. A workflow consists of a start node, an end node, and multiple action nodes.

#### Parameter Description

[Table 27-7](#) describes parameters in **workflow.xml**.

**Table 27-7** Parameters

Parameter	Meaning
name	Identifies a workflow file.
start	Indicates the workflow start node.
end	Indicates the workflow end node.
action	Indicates nodes (one or multiple) that are used to implement a service.

## Example Codes

```
<workflow-app xmlns="uri:oozie:workflow:1.0" name="data_load">
 <start to="copyData"/>
 <action name="copyData">
 </action>
 ii
 <end name="end"/>
</workflow-app>
```

### 27.3.2.3 Start Action

#### Function

The Start Action node indicates the start point of a workflow job. Each workflow job has only one Start Action node.

#### Parameter Description

[Table 27-8](#) describes the parameter used on the Start Action node.

**Table 27-8** Parameters

Parameter	Meaning
to	Identifies a subsequent action node.

## Example Codes

```
<start to="mr-dataLoad"/>
```

### 27.3.2.4 End Action

#### Function

The End Action node indicates the end point of a workflow job. Each workflow job has only one End Action node.

## Parameter Description

[Table 27-9](#) describes the parameter used on the End Action node.

**Table 27-9** Parameters

Parameter	Meaning
name	Identifies an end action.

## Example Codes

```
<end name="end"/>
```

### 27.3.2.5 Kill Action

#### Function

The Kill Action node indicates the end point of a workflow job when an error occurs.

## Parameter Description

[Table 27-10](#) describes parameters used on the Kill Action node.

**Table 27-10** Parameters

Parameter	Meaning
name	Identifies a kill action.
message	Provides error messages.
`\${wf:errorMessage(wf:lastErrorNode())}`	Indicates the internal error message function in the Oozie system.

## Example Codes

```
<kill name="fail">
 <message>
 This workflow failed, error message[${wf:errorMessage(wf:lastErrorNode())}]
 </message>
</kill>
```

### 27.3.2.6 FS Action

#### Function

The FS Action node is a Hadoop distributed file system (HDFS) operation node. You can create and delete HDFS files and folders and grant permissions for HDFS files and folders using this node.

## Parameter Description

**Table 27-11** describes parameters used on the FS Action node.

**Table 27-11** Parameters

Parameter	Meaning
name	Identifies an FS action.
delete	Deletes a specified file or folder.
move	Moves a file from the source directory to the target directory.
chmod	Modifies file or folder access rights.
path	Indicates the current file path.
source	Indicates the source file path.
target	Indicates the target file path.
permissions	Indicates permissions.

### NOTE

**`${variable name}`** indicates the value defined in **job.properties**.

For example, **`${nameNode}`** indicates **hdfs://hacluster**. (See [job.properties](#).)

## Example Codes

```
<action name="copyData">
 <fs>
 <delete path='${nameNode}/user/oozie_cli/${dataLoadRoot}/result' />
 <move source='${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce' target='${nameNode}/user/oozie_cli/${dataLoadRoot}/result' />
 <chmod path='${nameNode}/user/oozie_cli/${dataLoadRoot}/reuslt' permissions='-rwxrw-rw-' dir-files='true' /></chmod>
 </fs>
 <ok to="end" />
 <error to="fail" />
</action>
```

### 27.3.2.7 MapReduce Action

## Function

The MapReduce Action node is used to execute a map-reduce job.

## Parameter Description

**Table 27-12** describes parameters used on the MapReduce Action node.

**Table 27-12** Parameters

Parameter	Meaning
name	Identifies a map-reduce action.
resourceManager	Indicates the MapReduce ResourceManager address.
name-node	Indicates the Hadoop distributed file system (HDFS) NameNode address.
queueName	Identifies the MapReduce queue where a job is executed.
mapred.mapper.class	Identifies the Mapper class.
mapred.reducer.class	Identifies the Reducer class.
mapred.input.dir	Indicates the input directory of MapReduce processed data.
mapred.output.dir	Indicates the output directory of MapReduce processing results.
mapred.map.tasks	Indicates the number of map tasks.

 **NOTE**

**`${variable name}`** indicates the value defined in **job.properties**.

For example, **`${nameNode}`** indicates **`hdfs://hacluster`**. (See [job.properties](#).)

## Example Codes

Change **OOZIE\_URL\_DEFALUT** in the code example to the actual host name of any Oozie node, for example, *`https://10-1-131-131:21003/oozie/`*.

```
<action name="mr-dataLoad">
 <map-reduce>
 <resource-manager>${resourceManager}</resource-manager>
 <name-node>${nameNode}</name-node>
 <prepare>
 <delete path="${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce"/>
 </prepare>
 <configuration>
 <property>
 <name>mapred.job.queue.name</name>
 <value>${queueName}</value>
 </property>
 <property>
 <name>mapred.mapper.class</name>
 <value>org.apache.oozie.example.SampleMapper</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.apache.oozie.example.SampleReducer</value>
 </property>
 <property>
 <name>mapred.map.tasks</name>
 <value>1</value>
 </property>
 </configuration>
 </map-reduce>
</action>
```

```

</property>
<property>
 <name>mapred.input.dir</name>
 <value>/user/oozie/${dataLoadRoot}/input-data/text</value>
</property>
<property>
 <name>mapred.output.dir</name>
 <value>/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce</value>
</property>
</configuration>
</map-reduce>
<ok to="copyData"/>
<error to="fail"/>
</action>

```

### 27.3.2.8 coordinator.xml

#### Function

**coordinator.xml** is used to periodically execute a workflow job.

#### Parameter Description

**Table 27-13** describes parameters in **coordinator.xml**.

**Table 27-13** Parameters

Parameter	Meaning
frequency	Indicates the workflow execution interval.
start	Indicates the time when a scheduled workflow job is started.
end	Indicates the time when a scheduled workflow job is stopped.
resourceManager	Indicates the MapReduce ResourceManager address.
queueName	Identifies the MapReduce queue where a job is executed.
nameNode	Indicates the Hadoop distributed file system (HDFS) NameNode address.

#### NOTE

**\${variable name}** indicates the value defined in **job.properties**.

For example, **\${nameNode}** indicates **hdfs://hacluster**. (See [job.properties](#).)

#### Example Codes

```

<coordinator-app name="cron-coord" frequency="${coord:days(1)}" start="${start}" end="${end}"
 timezone="UTC" xmlns="uri:oozie:coordinator:0.2">
 <action>

```

```
<workflow>
 <app-path>${workflowAppUri}</app-path>
 <configuration>
 <property>
 <name>resourceManager</name>
 <value>${resourceManager}</value>
 </property>
 <property>
 <name>nameNode</name>
 <value>${nameNode}</value>
 </property>
 <property>
 <name>queueName</name>
 <value>${queueName}</value>
 </property>
 </configuration>
</workflow>
</action>
</coordinator-app>
```

## 27.3.3 Development of Java

### 27.3.3.1 Description

These typical scenarios help you quickly understand the development procedure of Oozie and learn key API functions.

This example shows you how to submit a MapReduce job and query the job status by using the Java API. The sample code relates to the MapReduce job only, but the API invocation codes of other jobs are the same. The difference is that the configuration of **job.properties** in a job and that of **workflow.xml** in a workflow is different.

#### NOTE

After the operations in [Downloading and Importing Sample Projects](#) are complete, you can submit MapReduce jobs and query job status through Java APIs.

### 27.3.3.2 Sample Code

#### Function

Oozie submits a job from the run method of **org.apache.oozie.client.OozieClient** and obtains job information from **getJobInfo**.

#### Sample Code

```
public void test(String jobFilePath) {
 try {
 runJob(jobFilePath);
 } catch (Exception exception) {
 exception.printStackTrace();
 }
}

private void runJob(String jobFilePath) throws OozieClientException, InterruptedException {
 Properties conf = getJobProperties(jobFilePath);
 String user = PropertiesCache.getInstance().getProperty("submit_user");
 conf.setProperty("user.name", user);
}
```



```
// submit and start the workflow job
String jobId = oozieClient.run(conf);

logger.info("Workflow job submitted: {}", jobId);

// wait until the workflow job finishes printing the status every 10 seconds
while (oozieClient.getJobInfo(jobId).getStatus() == WorkflowJob.Status.RUNNING) {
 logger.info("Workflow job running ... {}", jobId);
 Thread.sleep(10 * 1000);
}

// print the final status of the workflow job
logger.info("Workflow job completed ... {}", jobId);
logger.info(String.valueOf(oozieClient.getJobInfo(jobId)));
}

/**
 * Get job.properties File in filePath
 *
 * @param filePath file path
 * @return job.properties
 * @since 2020-09-30
 */
public Properties getJobProperties(String filePath) {
 File configFile = new File(filePath);
 if (!configFile.exists()) {
 logger.info(filePath, "{} is not exist.");
 }

 InputStream inputStream = null;

 // create a workflow job configuration
 Properties properties = oozieClient.createConfiguration();
 try {
 inputStream = new FileInputStream(filePath);
 properties.load(inputStream);
 } catch (Exception e) {
 e.printStackTrace();
 } finally {
 if (inputStream != null) {
 try {
 inputStream.close();
 } catch (IOException ex) {
 ex.printStackTrace();
 }
 }
 }

 return properties;
}
```

## 27.3.4 Scheduling Spark2x to Access HBase and Hive Using Oozie

### Prerequisites

Prerequisites in [Downloading and Importing Sample Projects](#) have been met.

### Preparing a Development Environment

**Step 1** Obtain the **OozieMapReduceExample**, **OozieSparkHBaseExample**, and **OozieSparkHiveExample** sample projects from the sample project folder **oozienormal-examples** in the **src\oozie-examples** directory where the sample

code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** Modify the parameters in each sample project. For details, see [Table 27-14](#).

**Table 27-14** Parameters to be modified

File Name	Parameter	Value	Example Value
src\main \resources \application.properties	submit_user	User who submits a job.	developuser
	oozie_url_default	https:// <i>Oozie service IP address</i> :21003/oozie/	https://10.10.10.233:21003/oozie/
src\main \resources \job.properties	userName	User who submits a job.	developuser
	examplesRoot	Use the default value or change the value based on the site requirements.	myjobs
	oozie.wf.application.path	Use the default value or change the value based on the site requirements.	\${nameNode}/user/\${userName}/\${examplesRoot}/apps/spark2x <b>NOTICE</b> Ensure that the path is the same as the path with the <code>&lt;jar&gt;</code> and <code>&lt;spark-opts&gt;</code> tags in the <code>src\main\resources\workflow.xml</code> file.
src\main \resources \workflow.xml	<code>&lt;jar&gt; &lt;/jar&gt;</code>	Change <b>OozieSparkHBase-1.0.jar</b> to the actual JAR package name.	<code>&lt;jar&gt;\${nameNode}/user/\${userName}/\${examplesRoot}/apps/spark2x/lib/OozieSparkHBase-1.0.jar&lt;/jar&gt;</code>

 **NOTE**

Go to the root directory of the project, for example, `D:\sample_project\src\oozie-examples\oozienormal-examples\OozieSparkHBaseExample`, and run the `mvn clean package -DskipTests` command. After the operation is successful, the package is in the target directory.

**Step 3** Create the following folders on the HDFS client in the configured path:

`/user/developuser/myjobs/apps/spark2x/lib`

/user/developuser/myjobs/apps/spark2x/hbase

/user/developuser/myjobs/apps/spark2x/hive

**Step 4** Upload the files listed in [Table 27-15](#) to the corresponding path.

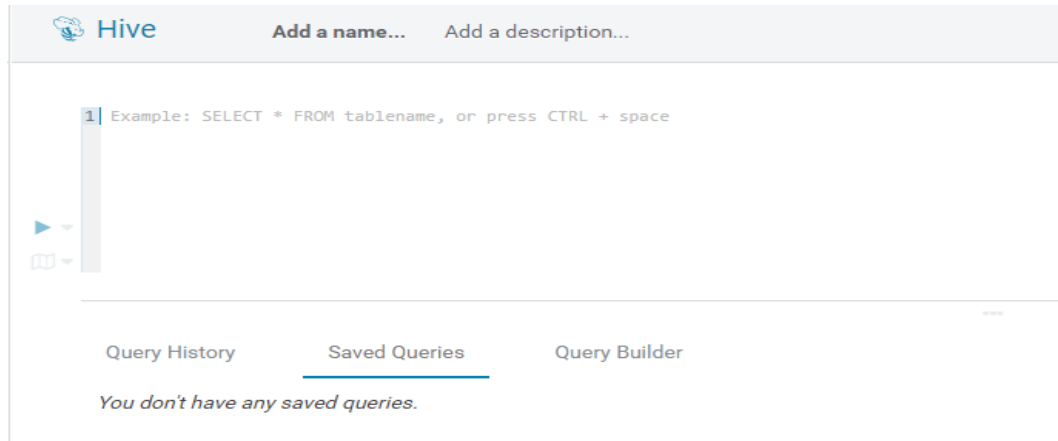
**Table 27-15** Files to be uploaded

Initial File Path	File	Destination Path
Spark client directory (for example, /opt/client/Spark2x/spark/conf)	hive-site.xml	/user/developuser/myjobs/apps/spark2x directory in the HDFS.
	hbase-site.xml	
Spark client directory (for example, /opt/client/Spark2x/spark/jars)	JAR package	Share HDFS /user/oozie/share/lib/spark2x directory of Oozie. <b>NOTE</b> This file must be uploaded as user <b>oozie</b> . Run the <b>su - oozie</b> command to switch to user <b>oozie</b> . After the file is uploaded, restart the Oozie service.
JAR package of the sample projects to be used	JAR package	/user/developuser/myjobs/apps/spark2x/lib/ directory in the HDFS.
OozieSparkHiveExample sample project directory src \main\resources	workflow.xml	/user/developuser/myjobs/apps/spark2x/hive directory in the HDFS. <b>NOTE</b> Change the path of <b>spark-archive-2x.zip</b> in <spark-opts> based on the actual HDFS file path.
OozieSparkHBaseExample sample project directory src \main\resources	workflow.xml	/user/developuser/myjobs/apps/spark2x/hbase directory in the HDFS. <b>NOTE</b> Change the path of <b>spark-archive-2x.zip</b> in <spark-opts> based on the actual HDFS file path.

**Step 5** Change the value of **hive.security.authenticator.manager** in the **hive-site.xml** file in the /user/developuser/myjobs/apps/spark2x directory of HDFS from **org.apache.hadoop.hive.ql.security.SessionStateUserMSGGroupAuthenticator** to **org.apache.hadoop.hive.ql.security.SessionStateUserGroupAuthenticator**.

**Step 6** Run the following commands to create a Hive table:

Enter the following SQL statements in the Hive panel on the Hue UI:



```
CREATE DATABASE test;
```

```
CREATE TABLE IF NOT EXISTS `test`.`usr` (user_id int comment
'userID',user_name string comment 'userName',age int comment
'age')PARTITIONED BY (country string)STORED AS PARQUET;
```

```
CREATE TABLE IF NOT EXISTS `test`.`usr2` (user_id int comment
'userID',user_name string comment 'userName',age int comment
'age')PARTITIONED BY (country string)STORED AS PARQUET;
```

```
INSERT INTO TABLE test.usr partition(country='CN') VALUES(1,'maxwell',45),
(2,'minwell',30),(3,'mike',22);
```

```
INSERT INTO TABLE test.usr partition(country='USA') VALUES(4,'minbin',35);
```

**Step 7** Use HBase Shell to run the following commands to create an HBase table:

```
create 'SparkHBase',{NAME=>'cf1'}
put 'SparkHBase','01','cf1:name','Max'
put 'SparkHBase','01','cf1:age','23'
----End
```

## 27.4 Commissioning the Application

### 27.4.1 Commissioning an Application in the Windows Environment

#### 27.4.1.1 Compiling and Running Applications

##### Scenario

After the code development is complete using Java APIs, you can run applications in the Windows development environment. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host.

## Procedure

- The HTTPS SSL certificate is required for running applications in Windows.
  - a. Log in to any node in the cluster and go to the following directory to download the **ca.crt** file.  
**cd \${BIGDATA\_HOME}/om-agent\_8.1.0.1/nodeagent/security/cert/subcert/certFile/**
  - b. Download the **ca.crt** file to a local directory and open the CLI as the administrator.  
Run the following command:  
**keytool -import -v -trustcacerts -alias ca -file "D:\xx\ca.crt" -storepass changeit -keystore "%JAVA\_HOME%\jre\lib\security\cacerts"**  
In the preceding command, **D:\xx\ca.crt** is the path for storing the **ca.crt** file. **%JAVA\_HOME %** indicates the JDK installation path.
  - c. In the development environment (such as IDEA), right-click **OozieRestApiMain.java**, and choose **Run 'OozieRestApiMain.main()'** to run the application project.
- Run the following command on the Oozie client:  
**oozie job -oozie https://Oozie service IP:21003/oozie -config job.properties -run**  
You need to copy the **job.properties** file in the **src\main\resources** directory of the sample project to the directory where the Oozie client is located in advance.

### 27.4.1.2 Checking the Commissioning Result

#### Scenario

The results can be viewed on the console after the Oozie sample project is completed.

#### Procedure

The following information is displayed if the sample project is successful:

```
cluset status is false
Warning: Could not get charToByteConverterClass!
Workflow job submitted: 0000067-160729120057089-oozie-omm-W
Workflow job running ...0000067-160729120057089-oozie-omm-W
Workflow job running ...0000067-160729120057089-oozie-omm-W
Workflow job running ...0000067-160729120057089-oozie-omm-W
Workflow job running ...0000067-160729120057089-oozie-omm-W
Workflow job running ...0000067-160729120057089-oozie-omm-W
Workflow job running ...0000067-160729120057089-oozie-omm-W
Workflow job running ...0000067-160729120057089-oozie-omm-W
Workflow job completed ...0000067-160729120057089-oozie-omm-W
Workflow id[0000067-160729120057089-oozie-omm-W] status[SUCCEEDED]
-----finish Oozie -----
```

Directory **/user/developuser/examples/output-data/map-reduce** is generated on the HDFS. The directory contains the following two files:

- **\_SUCCESS**

- part-00000

You can view the files by using the file browser of the Hue or running the following commands on the HDFS:

```
hdfs dfs -ls /user/developuser/examples/output-data/map-reduce
```

 NOTE

In the Windows environment, the following exception may occur but does not affect services.

```
java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.
```

## 27.5 More Information

### 27.5.1 Common API Introduce

#### 27.5.1.1 Shell

**Table 27-16** Interface parameters

Command	Parameter	Meaning
oozie version	-	The version information
oozie job	-config <arg>	Indicates the path to the job configuration file <b>job.properties</b> .
	-oozie <arg>	Indicates the Oozie server address.
	-haconfig <arg>	Indicates the path to the Oozie service configuration file <b>oozie-site.xml</b> .
	-run	Runs a job.
	-start <arg>	Starts a specified job.
	-submit	Submits a job.
	-kill <arg>	Deletes a specified job.
	-suspend <arg>	Suspends a specified job.
	-resume <arg>	Resumes a specified job.
	-D <property=value>	Sets a property.
oozie admin	-oozie <arg>	Indicates the Oozie server address.

Command	Parameter	Meaning
	-status	Indicates the service status of the Oozie service.

The Oozie command and other parameters can be found in the following address:[https://oozie.apache.org/docs/5.1.0/DG\\_CommandLineTool.html](https://oozie.apache.org/docs/5.1.0/DG_CommandLineTool.html).

### 27.5.1.2 Java

Java APIs are provided by **org.apache.oozie.client.OozieClient**.

**Table 27-17** API

Item	Description
public String run(Properties conf)	Runs a job.
public void start(String jobId)	Starts the specified job.
public String submit(Properties conf)	Submits a job.
public void kill(String jobId)	Delete the specified job.
public void suspend(String jobId)	Suspends the specified job.
public void resume(String jobId)	Resumes the specified job.
public WorkflowJob getJobInfo(String jobId)	Obtains job information.

### 27.5.1.3 REST

The common APIs of REST are the same as the APIs of Java. For details, see <http://oozie.apache.org/docs/5.1.0/WebServicesAPI.html>.

# 28 Presto Development Guide (Security Mode)

---

## 28.1 Overview

### 28.1.1 Application Development Overview

#### Introduction to Presto

Presto is an open source distributed SQL query engine for running interactive analytic queries against data sources of all sizes ranging from gigabytes to petabytes.

Presto has the following characteristics:

- **Multiple data sources:** Presto supports multiple types of connectors, such as MySQL, Hive, and JMX.
- **Support for SQL:** Presto supports the ANSI SQL. You can directly run the SQL shell command for query.
- **Hybrid computing:** You can query multiple catalogs using JOIN.

#### 28.1.2 Basic Concepts

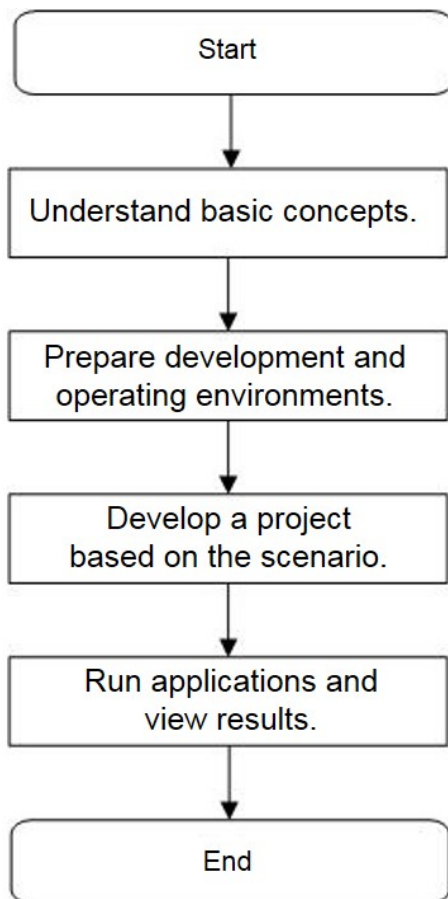
- **Connector**  
A connector adapts Presto to a data source such as Hive or a relational database.
- **Catalog**  
A Presto catalog contains schemas and references a data source via a connector.
- **Schema**  
Schemas are a way to organize tables.



## 28.1.3 Application Development Process

[Figure 28-1](#) and [Table 28-1](#) describe the phases in the development process.

**Figure 28-1** Presto application development process



**Table 28-1** Description of the Presto application development process

Phase	Description	Reference
Understand basic concepts.	Before application development, learn basic concepts of Presto.	<a href="#">Basic Concepts</a>
Prepare development and operating environments.	Presto applications can be developed in Java. Use the Eclipse tool to configure the development environment according to the guide.	<a href="#">Preparing Development and Operating Environment</a>
Develop a project based on the scenario.	Presto provides a Java sample project and a sample project of data query.	<a href="#">Typical Application Scenario</a>

Phase	Description	Reference
Run applications and view results.	This phase provides guidance for users to submit a developed application for running and view the result.	<a href="#">Running the JDBC Client and Viewing Results</a>

## 28.2 Environment Preparation

### 28.2.1 Preparing Development and Operating Environment

#### Preparing Development Environment

**Table 28-2** describes the local environment required for application development. You also need to prepare a Linux environment for verifying whether the application is running properly.

**Table 28-2** Development environment

Item	Description
OS	<ul style="list-style-type: none"> <li>Development environment: Windows 7 or later version is recommended.</li> <li>Operating environment: Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li> </ul>

Item	Description
<p>JDK installation</p>	<p>Basic configurations of the development and operating environments. The version requirements are as follows:</p> <ul style="list-style-type: none"> <li>• The server and client of an MRS cluster support only built-in Oracle JDK 1.8, which cannot be replaced.</li> <li>• If users' applications need to reference the JAR files of the SDK class in the user application processes, Oracle JDK and IBM JDK are supported. <ul style="list-style-type: none"> <li>– Oracle JDK versions: 1.7 and 1.8</li> <li>– IBM JDK versions: 1.7.8.10, 1.7.9.40, and 1.8.3.0</li> </ul> </li> </ul> <p><b>NOTE</b> For security purpose, MRS cluster servers support only TLS 1.1 and TLS 1.2 encryption protocols in the Presto development environment. IBM JDK supports only TLS 1.0 by default. If you use IBM JDK, set <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the parameter setting, TLS1.0/1.1/1.2 can be supported at the same time. For details, visit <a href="https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls">https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls</a>.</p>
<p>IntelliJ IDEA installation and configuration</p>	<p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>• If you use IBM JDK, ensure that the JDK configured in IntelliJ IDEA is IBM JDK.</li> <li>• If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK.</li> <li>• If you use ODBC for secondary development, ensure that JDK 1.8 or later is used.</li> <li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li> </ul>

Item	Description
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
User development preparation	See <a href="#">Preparing a Developer Account</a> for configuration.
7-zip	Used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-zip 16.04 is supported.

## Preparing an Operating Environment

During application development, you need to prepare a Linux environment for verifying whether the application is running properly.

1. Install the client on the node. For example, the client installation directory is `/opt/client`.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to use the client on a Master or Core node in the cluster, see [Using the Client](#). For details about how to install the client outside the MRS cluster, see [Installing the Client](#).  
2. [Download the cluster client software package](#) to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Presto/config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, `/opt/client/conf`.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is `/tmp/FusionInsight-Client` on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Presto/config/* root@IP address of the client node:/opt/client/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 28-3](#) describes the main configuration files.

**Table 28-3** Configuration files

File	Function
user.keytab	Provides user information for Kerberos security authentication.

File	Function
krb5.conf	Contains Kerberos server configuration information.

3. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

## 28.2.2 Configuring and Importing the JDBC API Sample Project

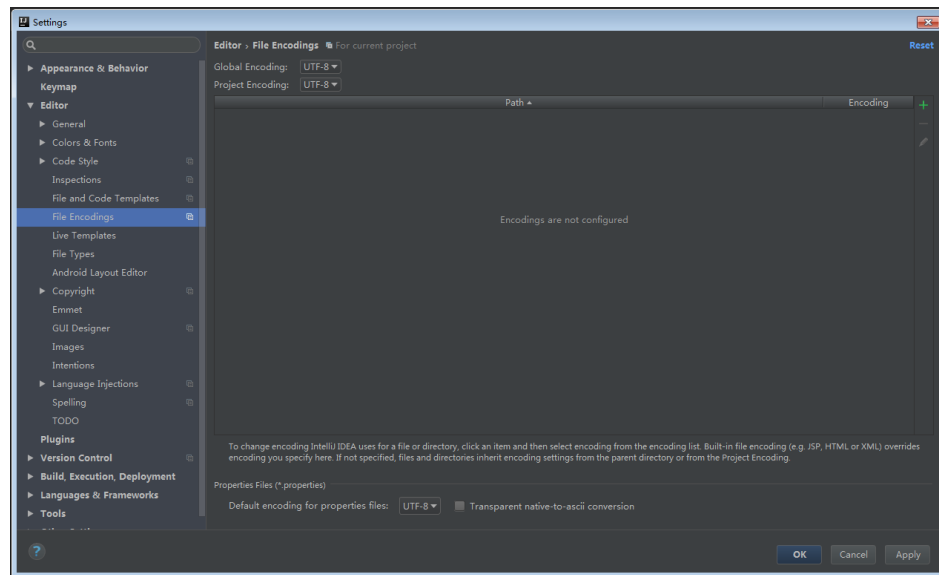
### Scenario

To run the JDBC API sample code of Presto, you need to perform the following operations. The following example develops an application that uses JDBC to connect to Presto in the Windows environment.

### Procedure

- Step 1** Obtain the Presto sample project. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the root directory of the Presto sample project, run the **mvn install** command to perform compilation.
- Step 3** In the root directory of the Presto sample project, run the **mvn idea:idea** command to create an IntelliJ IDEA project.
- Step 4** Copy the **user.keytab** and **krb5.conf** files obtained in [Preparing a Developer Account](#) section to the **conf** directory of the sample project.
- Step 5** Import the example project to the IntelliJ IDEA development environment.
  1. Open IntelliJ IDEA and choose **File > Open**.
  2. In the **Open File or Project** dialog box that is displayed, select the Presto sample project folder and click **OK**. After the folder is imported, the **PrestoJDBCExample** class is the JDBC API sample code.
- Step 6** Set the IntelliJ IDEA text file coding format to prevent garbled characters.
  1. On the IntelliJ IDEA menu bar, choose **File > Settings**.
  2. Choose **Editor > File Encodings** from the navigation tree of the **Settings** window. In the "Global Encoding" and "Project Encodings" area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 28-2](#)

Figure 28-2 Setting the IntelliJ IDEA coding format



**Step 7** Modify the sample. You can skip this step for a cluster with Kerberos authentication disabled.

After obtaining the **krb5.conf** and **user.keytab** files of the new development user in [Preparing a Developer Account](#), modify **KerberosPrincipal** in **presto.properties** to the principal of the new user, **KerberosConfigPath** to the path of the **krb5.conf** file, and **KerberosKeytabPath** to the path of the **keytab** file.

----End

## 28.2.3 Configuring and Importing the HCatalog API Sample Project

### Scenario

To run the HCatalog API sample code of Presto, you need to perform the following operations. The following example develops an application that uses HCatalog to connect to Presto in the Windows environment.

### Procedure

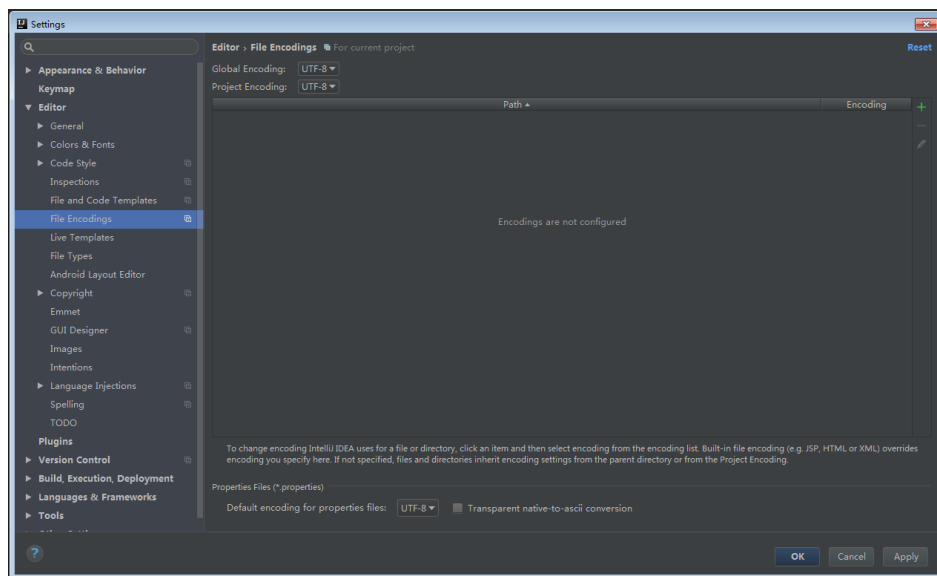
- Step 1** Obtain the Presto sample project. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the root directory of the Presto sample project, run the **mvn install** command to perform compilation.
- Step 3** In the root directory of the Presto sample project, run the **mvn idea:idea** command to create an IntelliJ IDEA project.
- Step 4** Import the example project to the IntelliJ IDEA development environment.
  1. Open IntelliJ IDEA and choose **File > Open**.

2. In the **Open File or Project** dialog box that is displayed, select the Presto sample project folder and click **OK**. After the folder is imported, the **HCatalogExample** class in the **com.huawei.bigdata.presto.example** folder is the HCatalog API sample code.

**Step 5** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.
2. Choose **Editor > File Encodings** from the navigation tree of the **Settings** window. In the "Global Encoding" and "Project Encodings" area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 28-3](#)

**Figure 28-3** Setting the IntelliJ IDEA coding format



----End

## 28.3 Developing the Project

### 28.3.1 Typical Application Scenario

#### Scenario Description

Assume that you need to develop a Presto data analysis application to obtain the **call\_center** table of TPCDS Catalog provided by Presto.

#### Development Guidelines

**Step 1** Prepare data.

1. Create three tables: employee information table **employees\_info**, employee contact table **employees\_contact**, and extended employee information table **employees\_info\_extended**.
  - Employee information table **employees\_info** contains fields such as employee ID, name, salary currency, salary, tax category, work place, and hire date. In salary currency, R indicates RMB and D indicates USD.

- Fields in the **employees\_contact** table include the employee ID, phone number, and email address.
- Fields in the **employees\_info\_extended** table include the employee ID, name, mobile phone number, e-mail address, salary currency, salary, tax category, and work place. The partition field is the hire date.

For details about table creation codes, see [Creating a Table](#).

2. Load employee information to **employees\_info**.

For details about data loading codes, see [Loading Data](#).

[Table 28-4](#) provides employee information.

**Table 28-4** Employee information

Employee ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hire Date
1	Wang	R	8000.01	personal income tax&0.05	China:Shenzhen	2014
3	Tom	D	12000.02	personal income tax&0.09	America:NewYork	2014
4	Jack	D	24000.03	personal income tax&0.09	America:Manhattan	2014
6	Linda	D	36000.04	personal income tax&0.09	America:NewYork	2014
8	Zhang	R	9000.05	personal income tax&0.05	China:Shanghai	2014

3. Load employee contact information to **employees\_contact**.

[Table 28-5](#) provides employee contact information.

**Table 28-5** Employee contact information

Employee ID	Phone Number	e-mail
1	135 XXXX XXXX	xxxx@xx.com
3	159 XXXX XXXX	xxxxx@xx.com.cn
4	186 XXXX XXXX	xxxx@xx.org
6	189 XXXX XXXX	xxxx@xxx.cn
8	134 XXXX XXXX	xxxx@xxxx.cn



**Step 2** Analyze data.

For details about data analysis codes, see [Querying Data](#).

- Query contact information of employees whose salaries are paid in USD.
- Query the IDs and names of employees who were hired in 2014, and load the query results to the partition with the hire date of 2014 in the **employees\_info\_extended** table.
- Collect the number of records in the **employees\_info** table.
- Query information about employees whose email addresses end with "cn".

**Step 3** Submit a data analysis task to collect the number of records in the **employees\_info** table. For details, see [Example Program Guide](#).

----End

## 28.3.2 Sample Code Description

### Presto JDBC Usage Example

The following code snippet is in the **PrestoJDBCExample** class and is used to connect JDBC to Presto TPCDS Catalog.

```
private static Connection connection;
private static Statement statement;
/**
 * Only when Kerberos authentication enabled, configurations in presto-examples/conf/presto.properties
 * should be set. More details please refer to https://prestodb.io/docs/0.215/installation/jdbc.html.
 */
private static void initConnection(String url, boolean krbsEnabled) throws SQLException {
 if (krbsEnabled) {
 String filePath = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
 File proFile = new File(filePath + "presto.properties"); if (proFile.exists()) {
 Properties props = new Properties();
 try {
 props.load(new FileInputStream(proFile));
 } catch (IOException e) {
 e.printStackTrace();
 }
 connection = DriverManager.getConnection(url, props);
 }
 } else {
 connection = DriverManager.getConnection(url, "presto", null);
 }
 statement = connection.createStatement();
}

private static void releaseConnection() throws SQLException {
 statement.close();
 connection.close();
}

public static void main(String[] args) throws SQLException {
 try {
 /**
 * Replace example_ip with your cluster presto server ip.
 * By default, Kerberos authentication disabled cluster presto service port is 7520, Kerberos
 * authentication enabled cluster presto service port is 7521
 * The postfix /tpcds/sf1 means to use tpcds catalog and sf1 schema, you can use hive catalog as well
 * If Kerberos authentication enabled, set the second param to true.
 * see PrestoJDBCExample#initConnection(java.lang.String, boolean).
 */
 initConnection("jdbc:presto://example_ip:7520/tpcds/sf1", false);
 }
}
```

```
//initConnection("jdbc:presto://example_ip:7521/tpcds/sf1", true);
ResultSet resultSet = statement.executeQuery("select * from call_center");
while (resultSet.next()) {
 System.out.println(resultSet.getString("cc_name") + " : " + resultSet.getString("cc_employees"));
}
} catch (SQLException e) {
 e.printStackTrace();
} finally {
 releaseConnection();
}
}
```

## 28.4 Commissioning the Application

### Running the JDBC Client and Viewing Results

- Step 1** Run the `mvn clean compile assembly:single` command to generate a JAR file and obtain it from the `target` directory in the project directory, for example, `presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar`.
- Step 2** Create a directory as the running directory in the running and commissioning environment, for example, `/opt/presto_examples` (Linux), and create the `conf` subdirectory in the directory.  
  
Copy `presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar` exported in [Step 1](#) to `/opt/presto_examples`.
- Step 3** To enable Kerberos authentication for a cluster, you need to copy the obtained `user.keytab` and `krb5.conf` files to the `/opt/presto_examples/conf` directory and modify the `presto.properties` file in the `conf` directory of the sample code. Skip this step for a cluster with Kerberos authentication disabled.

**Table 28-6** presto.properties parameters

Parameter	Description
user	Username used for Kerberos authentication, that is, the username of the development user created in <a href="#">Preparing a Developer Account</a> .
KerberosPrincipal	Username used for authentication. That is, the username of the development user created in <a href="#">Preparing a Developer Account</a> needs to be certificated.
KerberosConfigPath	Path where <code>krb5.conf</code> is stored
KerberosKeytabPath	Path where <code>user.keytab</code> is stored

#### Example

```
user = prestouser
SSL = true
KerberosRemoteServiceName = HTTP
```

```
KerberosPrincipal = prestouser
KerberosConfigPath = /opt/presto_examples/conf/krb5.conf
KerberosKeytabPath = /opt/presto_examples/conf/user.keytab
```

**Step 4** In Linux, run the sample program.

```
chmod +x /opt/presto_examples -R
cd /opt/presto_examples
java -jar presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
```

**Step 5** In the CLI, view the query results of the example code.

If the following information is displayed, the sample project execution is successful in Linux.

```
NY Metro : 2
Mid Atlantic : 6
Mid Atlantic : 6
North Midwest : 1
North Midwest : 3
North Midwest : 7
```

----End

## 28.5 Presto APIs

Presto JDBC APIs comply with the Java JDBC driver standard. For details, see JDK 1.7 API. For details about how to use the Presto JDBC, see <https://prestodb.io/docs/current/installation/jdbc.html>.

## 28.6 FAQs

### 28.6.1 No Certificate Is Available When PrestoJDBCExample Run on a Node Outside the Cluster

#### Question

The **presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar** file is running properly on nodes in the cluster. However, no certificate is available when PrestoJDBCExample runs on a node outside the cluster to connect to the cluster with Kerberos authentication enabled, the following error message is displayed:

```
java.sql.SQLException: Error executing query
 at
com.facebook.presto.jdbc.PrestoStatement.internalExecute(PrestoStatement.java:274)
 at com.facebook.presto.jdbc.PrestoStatement.execute(PrestoStatement.java:227)
 at
com.facebook.presto.jdbc.PrestoStatement.executeQuery(PrestoStatement.java:76)
 at
PrestoJDBCExample.main(PrestoJDBCExample.java:65)
Caused by: java.io.UncheckedIOException:
javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException:
PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to find
valid certification path to requested target
 at
com.facebook.presto.jdbc.internal.client.JsonResponse.execute(JsonResponse.java:154)
 at
com.facebook.presto.jdbc.internal.client.StatementClientV1.<init>(StatementClientV1.java:129)
 at
```

```
com.facebook.presto.jdbc.internal.client.StatementClientFactory.newStatementClient(StatementClientFactory.java:24)
 at
com.facebook.presto.jdbc.QueryExecutor.startQuery(QueryExecutor.java:46)
 at
com.facebook.presto.jdbc.PrestoConnection.startQuery(PrestoConnection.java:683)
 at
com.facebook.presto.jdbc.PrestoStatement.internalExecute(PrestoStatement.java:239)
 ... 3 more
Caused by: javax.net.ssl.SSLHandshakeException:
sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to find
valid certification path to requested target
 at
sun.security.ssl.Alerts.getSSLException(Alerts.java:192)
 at
sun.security.ssl.SSLSocketImpl.fatal(SSLSocketImpl.java:1959)
 at
sun.security.ssl.Handshaker.fatalSE(Handshaker.java:302)
 at
sun.security.ssl.Handshaker.fatalSE(Handshaker.java:296)
 at
sun.security.ssl.ClientHandshaker.serverCertificate(ClientHandshaker.java:1514)
 at
sun.security.ssl.ClientHandshaker.processMessage(ClientHandshaker.java:216)
 at
sun.security.ssl.Handshaker.processLoop(Handshaker.java:1026)
 at
sun.security.ssl.Handshaker.process_record(Handshaker.java:961)
 at
sun.security.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:1072)
 at
sun.security.ssl.SSLSocketImpl.performInitialHandshake(SSLSocketImpl.java:1385)
 at
sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:1413)
 at
sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:1397)
 at
com.facebook.presto.jdbc.internal.okhttp3.internal.connection.RealConnection.connectTls(RealConnection.java:318)
 at
com.facebook.presto.jdbc.internal.okhttp3.internal.connection.RealConnection.establishProtocol(RealConnection.java:282)
 at
com.facebook.presto.jdbc.internal.okhttp3.internal.connection.RealConnection.connect(RealConnection.java:167)
 at
com.facebook.presto.jdbc.internal.okhttp3.internal.connection.StreamAllocation.findConnection(StreamAllocation.java:257)
 at
com.facebook.presto.jdbc.internal.okhttp3.internal.connection.StreamAllocation.findHealthyConnection(StreamAllocation.java:135)
 at
com.facebook.presto.jdbc.internal.okhttp3.internal.connection.StreamAllocation.newStream(StreamAllocation.java:114)
 at
com.facebook.presto.jdbc.internal.okhttp3.internal.connection.ConnectInterceptor.intercept(ConnectInterceptor.java:42)
 at
com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
 at
com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
 at
com.facebook.presto.jdbc.internal.okhttp3.internal.cache.CacheInterceptor.intercept(CacheInterceptor.java:93)
 at
com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
```

```
at
com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
at
com.facebook.presto.jdbc.internal.okhttp3.internal.http.BridgeInterceptor.intercept(BridgeInterceptor.java:93)
at
com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
at
com.facebook.presto.jdbc.internal.okhttp3.internal.http.RetryAndFollowUpInterceptor.intercept(RetryAndFollowUpInterceptor.java:126)
at
com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
at
com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
at
com.facebook.presto.jdbc.internal.client.SpnegoHandler.intercept(SpnegoHandler.java:109)
at
com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
at
com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
at
com.facebook.presto.jdbc.internal.client.OkHttpUtil.lambda$userAgent$0(OkHttpUtil.java:77)
at
com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
at
com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
at
com.facebook.presto.jdbc.internal.okhttp3.RealCall.getResponseWithInterceptorChain(RealCall.java:200)
at
com.facebook.presto.jdbc.internal.okhttp3.RealCall.execute(RealCall.java:77)
at
com.facebook.presto.jdbc.internal.client.JsonResponse.execute(JsonResponse.java:131)
... 8 more
Caused by: sun.security.validator.ValidatorException: PKIX
path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to find
valid certification path to requested target
at
sun.security.validator.PKIXValidator.doBuild(PKIXValidator.java:397)
at
sun.security.validator.PKIXValidator.engineValidate(PKIXValidator.java:302)
at
sun.security.validator.Validator.validate(Validator.java:260)
at
sun.security.ssl.X509TrustManagerImpl.validate(X509TrustManagerImpl.java:324)
at
sun.security.ssl.X509TrustManagerImpl.checkTrusted(X509TrustManagerImpl.java:229)
at
sun.security.ssl.X509TrustManagerImpl.checkServerTrusted(X509TrustManagerImpl.java:124)
at
sun.security.ssl.ClientHandshaker.serverCertificate(ClientHandshaker.java:1496)
... 41 more
Caused by: sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target
at
sun.security.provider.certpath.SunCertPathBuilder.build(SunCertPathBuilder.java:141)
at
sun.security.provider.certpath.SunCertPathBuilder.engineBuild(SunCertPathBuilder.java:126)
at
java.security.cert.CertPathBuilder.build(CertPathBuilder.java:280)
at
```

```
sun.security.validator.PKIXValidator.doBuild(PKIXValidator.java:392)
... 47 more
```

## Answer

When the HTTPS protocol is used to connect to the security cluster, the server certificate is not authenticated. As a result, the connection fails.

You can replace the cacerts file in the **java jdk** directory on the current node with the cacerts file (for example, **/opt/Bigdata/jdk1.8.0\_232/jre/lib/security/cacerts**) in the **java jdk** directory on a node in the cluster.

## 28.6.2 When a Node Outside a Cluster Is Connected to a Cluster with Kerberos Authentication Enabled, HTTP Cannot Find the Corresponding Record in the Kerberos Database

### Question

The **presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar** file is running properly on nodes in the cluster. However, when PrestoJDBCExample running on a node outside the cluster connect to the cluster with Kerberos authentication enabled, the following error messages is displayed:

Error 1:

```
java.sql.SQLException:
Kerberos error for [HTTP@10.33.11.138]: No valid credentials provided
(Mechanism level: No valid credentials provided (Mechanism level: Server not
found in Kerberos database (7) - UNKNOWN_SERVER))
at
io.prestosql.jdbc.PrestoStatement.internalExecute(PrestoStatement.java:281)
at
io.prestosql.jdbc.PrestoStatement.execute(PrestoStatement.java:229)
at
io.prestosql.jdbc.PrestoStatement.executeQuery(PrestoStatement.java:78)
at PrestoJDBCExample.main(PrestoJDBCExample.java:68)
Caused by:
io.prestosql.jdbc.$internal.client.ClientException: Kerberos error for
[HTTP@10.33.11.138]: No valid credentials provided (Mechanism level: No valid
credentials provided (Mechanism level: Server not found in Kerberos database
(7) - UNKNOWN_SERVER))
at
io.prestosql.jdbc.$internal.client.SpnegoHandler.generateToken(SpnegoHandler.java:174)
at
io.prestosql.jdbc.$internal.client.SpnegoHandler.authenticate(SpnegoHandler.java:140)
at
io.prestosql.jdbc.$internal.client.SpnegoHandler.authenticate(SpnegoHandler.java:128)
at
io.prestosql.jdbc.
$internal.okhttp3.internal.http.RetryAndFollowUpInterceptor.followUpRequest(RetryAndFollowUpInterceptor.j
ava:289)
at
io.prestosql.jdbc.
$internal.okhttp3.internal.http.RetryAndFollowUpInterceptor.intercept(RetryAndFollowUpInterceptor.java:
157)
at
io.prestosql.jdbc.$internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
at
io.prestosql.jdbc.$internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
at
io.prestosql.jdbc.$internal.client.SpnegoHandler.intercept(SpnegoHandler.java:115)
at
```

```
io.prestosql.jdbc.$internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
at io.prestosql.jdbc.$internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:
121)
at
io.prestosql.jdbc.$internal.client.OkHttpUtil.lambda$userAgent$0(OkHttpUtil.java:64)
at
io.prestosql.jdbc.$internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
at
io.prestosql.jdbc.$internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
at
io.prestosql.jdbc.$internal.okhttp3.RealCall.getResponseWithInterceptorChain(RealCall.java:200)
at
io.prestosql.jdbc.$internal.okhttp3.RealCall.execute(RealCall.java:77)
at
io.prestosql.jdbc.$internal.client.JsonResponse.execute(JsonResponse.java:131)
at
io.prestosql.jdbc.$internal.client.StatementClientV1.<init>(StatementClientV1.java:132)
at
io.prestosql.jdbc.$internal.client.StatementClientFactory.newStatementClient(StatementClientFactory.java:24)
at
io.prestosql.jdbc.QueryExecutor.startQuery(QueryExecutor.java:46)
at io.prestosql.jdbc.PrestoConnection.startQuery(PrestoConnection.java:714)
at
io.prestosql.jdbc.PrestoStatement.internalExecute(PrestoStatement.java:241)
... 3 more
Caused by: GSSException:
No valid credentials provided (Mechanism level: No valid credentials provided
(Mechanism level: Server not found in Kerberos database (7) - UNKNOWN_SERVER))
at
sun.security.jgss.spnego.SpNegoContext.initSecContext(SpNegoContext.java:454)
at
sun.security.jgss.GSSContextImpl.initSecContext(GSSContextImpl.java:248)
at sun.security.jgss.GSSContextImpl.initSecContext(GSSContextImpl.java:179)
at
io.prestosql.jdbc.$internal.client.SpnegoHandler.generateToken(SpnegoHandler.java:167)
... 23 more
Caused by: GSSException:
No valid credentials provided (Mechanism level: Server not found in Kerberos database
(7) - UNKNOWN_SERVER)
at
sun.security.jgss.krb5.Krb5Context.initSecContext(Krb5Context.java:772)
at
sun.security.jgss.GSSContextImpl.initSecContext(GSSContextImpl.java:248)
at
sun.security.jgss.GSSContextImpl.initSecContext(GSSContextImpl.java:179)
at
sun.security.jgss.spnego.SpNegoContext.GSS_initSecContext(SpNegoContext.java:882)
at
sun.security.jgss.spnego.SpNegoContext.initSecContext(SpNegoContext.java:317)
... 26 more
Caused by: KrbException:
Server not found in Kerberos database (7) - UNKNOWN_SERVER
at
sun.security.krb5.KrbTgsRep.<init>(KrbTgsRep.java:73)
at
sun.security.krb5.KrbTgsReq.getReply(KrbTgsReq.java:251)
at
sun.security.krb5.KrbTgsReq.sendAndGetCreds(KrbTgsReq.java:262)
at
sun.security.krb5.internal.CredentialsUtil.serviceCreds(CredentialsUtil.java:308)
at
sun.security.krb5.internal.CredentialsUtil.acquireServiceCreds(CredentialsUtil.java:126)
at
sun.security.krb5.Credentials.acquireServiceCreds(Credentials.java:466)
at sun.security.jgss.krb5.Krb5Context.initSecContext(Krb5Context.java:695)
... 30 more
Caused by: KrbException:
Identifier doesn't match expected value (906)
at
```

```
sun.security.krb5.internal.KDCRep.init(KDCRep.java:140)
at
sun.security.krb5.internal.TGSRep.init(TGSRep.java:65)
at
sun.security.krb5.internal.TGSRep.<init>(TGSRep.java:60)
at
sun.security.krb5.KrbTgsRep.<init>(KrbTgsRep.java:55)
... 36 more
```

## Error 2:

```
java.sql.SQLException:
Authentication failed: Authentication failed for token:
YIIC1wYGKwYBBQUCoIIcYzCCAsegDTALBgkqhkiG9xIABgKhBAMCAaiggKuBIIcQmCCAqYgG0MjBfNzI0Q180Q
UFCX0FBOEJfNzIwN0E3OEuW0EU1LkNPTalfMB2gAwIBAKEWMBQbBEhUVFABDDE5Mi4xNjguMC45MaOCAS
YwggEioAMCARKhAwIBAAKCARQEggEQpYLaTplwpMc0EjXgU/
+bAn9Evk1UHysyhTPajzFpHtxzTAZCPm85ROufj
+cLlloGEcp2JBH9Le8bSL2Y1cE0mdK2Mlw2+S7J9G0mZKQYugMUXolqS14XbnA8tvgPEXFa6e15InqEUoRNb4Y
Ht7Rr/zRvsPbWKC4HQnKbtI8HSAid2K2JpTuVvXkOQpa+kgfMwgfCgAwIBEqKB6ASB5U3I4qvcqbfPhy
+0o97agfW9xBfeuZUjfKUUQC165Z8nbY7RJmM+3v6mrqsYRcGG2Agepd9F+neeL7Ljcf7ASzSb4otvPDTWsv07/
OCASyWggEioAMTrRgdWQQLMggfUPbendaKESx1QtuEpJuoGtyPJ9QzKI4rbk1UHysyhTPajzFpHtxzTAZCPm85R
+qUBQvbd3xhUlkPkng6pOCRv9gr6E=
at
com.facebook.presto.jdbc.PrestoStatement.internalExecute(PrestoStatement.java:271)
at
com.facebook.presto.jdbc.PrestoStatement.execute(PrestoStatement.java:227)
at
com.facebook.presto.jdbc.PrestoStatement.executeQuery(PrestoStatement.java:76)
at
PrestoJDBCExample.main(PrestoJDBCExample.java:65)
Caused by:
com.facebook.presto.jdbc.internal.client.ClientException: Authentication failed:
Authentication failed for token:
YIIC1wYGKwYBBQUCoIIcYzCCAsoGcDBQAgAAAo4IBhGGCAYAwggF8oAMCAQWhKhsoNDI1Nzg0MjBfNzI0Q
180QUFCX0FBOEJfNzIwN0E3OEuW0EU1LkNPTalfMB2gAwIBAKEWMBQbBEhUVFABDDE5Mi4xNjguMC45Ma
OCASyWggEioAMTrRgdWQQLMggfUPbendaKESx1QtuEpJuoGtyPJ9QzKI4rbk1UHysyhTPajzFpHtxzTAZCPm85R
Oufj
+cLlloGEcp2JBH9Le8bSL2Y1cE0mdK2Mlw2+S7J9G0mZKQYugMUXolqS14XbnA8tvgPEXFa6eEICV2sE02w1fN
dOMilfeqZffUMfNBSanRkHQFW1xlyb3EK2JpTuVvXkOQpa+kgfMwgfCgAwIBEqKB6ASB5U3I4qvcqbfPhy
+0o97agfW9xBfeuZUjfKUUQC165Z8nbY7RJmM
+3v6mrqsYRcGG2AgepV2NKDSySvpgzPEfGspJpkVeBbi5YJSVL3G/
fOFNkpLDjDpjDP4EDfbwXaYwASU9zXZZ75FJoO+ly3DLsyX68rgwSJiolIjK4jDRqVVoXTNRbeWipkiOoLsXHf9q
+qUBQvbd3xhUlkPkng6pOCRv9gr6
at
com.facebook.presto.jdbc.internal.client.StatementClientV1.requestFailedException(StatementClientV1.java:
432)
at
com.facebook.presto.jdbc.internal.client.StatementClientV1.<init>(StatementClientV1.java:132)
at
com.facebook.presto.jdbc.internal.client.StatementClientFactory.newStatementClient(StatementClientFactory.
java:24)
at
com.facebook.presto.jdbc.QueryExecutor.startQuery(QueryExecutor.java:46)
at
com.facebook.presto.jdbc.PrestoConnection.startQuery(PrestoConnection.java:683)
at
com.facebook.presto.jdbc.PrestoStatement.internalExecute(PrestoStatement.java:239)
... 3 more
```

## Answer

The principal of HTTP concatenated by the client is inconsistent with that in the Kerberos database (Error 1) or the obtained token cannot be connected to Presto.

Run the **cat /etc/hosts** command in the cluster to add the IP address and host name of the Presto coordinator to the **/etc/hosts** file of the current node.



```
192.168.0.91 node-masterlbico.42578420-724c-4aab-aa8b-7207a78e08e5.com
192.168.0.185 node-ana-coredYqj.42578420-724c-4aab-aa8b-7207a78e08e5.com
```

# 29 Presto Development Guide (Normal Mode)

---

## 29.1 Overview

### 29.1.1 Application Development Overview

#### Introduction to Presto

Presto is an open source distributed SQL query engine for running interactive analytic queries against data sources of all sizes ranging from gigabytes to petabytes.

Presto has the following characteristics:

- **Multiple data sources:** Presto supports multiple types of connectors, such as MySQL, Hive, and JMX.
- **Support for SQL:** Presto supports the ANSI SQL. You can directly run the SQL shell command for query.
- **Hybrid computing:** You can query multiple catalogs using JOIN.

#### 29.1.2 Basic Concepts

- **Connector**  
A connector adapts Presto to a data source such as Hive or a relational database.
- **Catalog**  
A Presto catalog contains schemas and references a data source via a connector.
- **Schema**  
Schemas are a way to organize tables.

### 29.1.3 Application Development Process

Figure 29-1 and Table 29-1 describe the phases in the development process.

Figure 29-1 Presto application development process

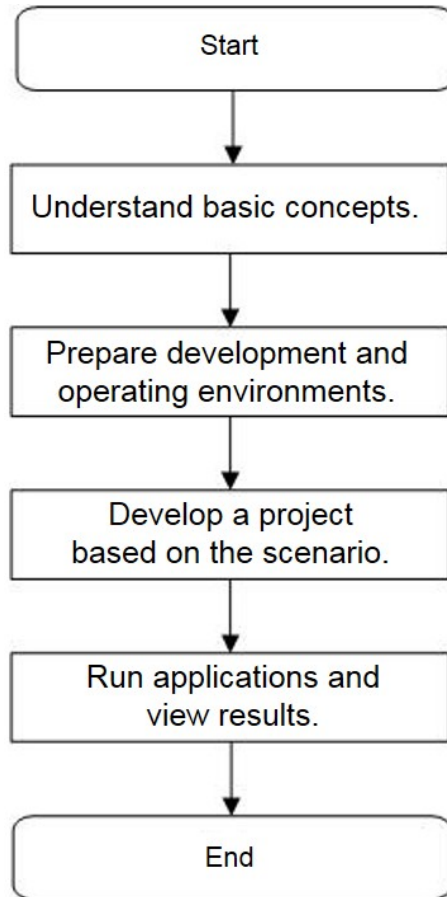


Table 29-1 Description of the Presto application development process

Phase	Description	Reference
Understand basic concepts.	Before application development, learn basic concepts of Presto.	<a href="#">Basic Concepts</a>
Prepare development and operating environments.	Presto applications can be developed in Java. Use the Eclipse tool to configure the development environment according to the guide.	<a href="#">Preparing Development and Operating Environment</a>
Develop a project based on the scenario.	Presto provides a Java sample project and a sample project of data query.	<a href="#">Typical Application Scenario</a>

Phase	Description	Reference
Run applications and view results.	This phase provides guidance for users to submit a developed application for running and view the result.	<a href="#">Running the JDBC Client and Viewing Results</a>

## 29.2 Environment Preparation

### 29.2.1 Preparing Development and Operating Environment

#### Preparing Development Environment

**Table 29-2** describes the local environment required for application development. You also need to prepare a Linux environment for verifying whether the application is running properly.

**Table 29-2** Development environment

Item	Description
OS	<ul style="list-style-type: none"> <li>Development environment: Windows 7 or later version is recommended.</li> <li>Operating environment: Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li> </ul>

Item	Description
JDK installation	<p>Basic configurations of the development and operating environments. The version requirements are as follows:</p> <ul style="list-style-type: none"> <li>• The server and client of an MRS cluster support only built-in Oracle JDK 1.8, which cannot be replaced.</li> <li>• If users' applications need to reference the JAR files of the SDK class in the user application processes, Oracle JDK and IBM JDK are supported. <ul style="list-style-type: none"> <li>– Oracle JDK versions: 1.7 and 1.8</li> <li>– IBM JDK versions: 1.7.8.10, 1.7.9.40, and 1.8.3.0</li> </ul> </li> </ul> <p><b>NOTE</b> For security purpose, MRS cluster servers support only TLS 1.1 and TLS 1.2 encryption protocols in the Presto development environment. IBM JDK supports only TLS 1.0 by default. If you use IBM JDK, set <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the parameter setting, TLS1.0/1.1/1.2 can be supported at the same time. For details, visit <a href="https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls">https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls</a>.</p>
IntelliJ IDEA installation and configuration	<p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>• If you use IBM JDK, ensure that the JDK configured in IntelliJ IDEA is IBM JDK.</li> <li>• If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK.</li> <li>• If you use ODBC for secondary development, ensure that JDK 1.8 or later is used.</li> <li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li> </ul>

Item	Description
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
7-zip	Used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-zip 16.04 is supported.

## Preparing an Operating Environment

During application development, you need to prepare a Linux environment for verifying whether the application is running properly.

1. Install the client on the node. For example, the client installation directory is `/opt/client`.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [. For details about how to install the client outside the MRS cluster, see \[.\]\(#\)](#)
2. [. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user \*\*root\*\*. Go to the decompression path of the cluster client and copy all configuration files in the \*\*FusionInsight\\_Cluster\\_1\\_Services\\_ClientConfig/Presto/config\*\* directory to the \*\*conf\*\* directory where the compiled JAR package is stored for subsequent commissioning, for example, \*\*/opt/client/conf\*\*.](#)

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Presto/config/* root@IP address of the client node:/opt/client/conf
```
3. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

## 29.2.2 Configuring and Importing the JDBC API Sample Project

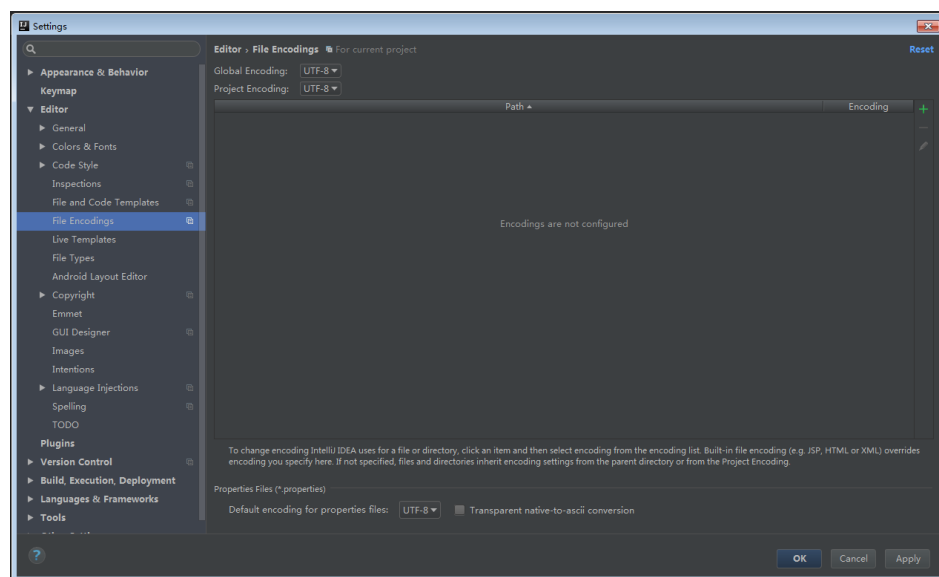
### Scenario

To run the JDBC API sample code of Presto, you need to perform the following operations. The following example develops an application that uses JDBC to connect to Presto in the Windows environment.

### Procedure

- Step 1** Obtain the Presto sample project. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the root directory of the Presto sample project, run the `mvn install` command to perform compilation.
- Step 3** In the root directory of the Presto sample project, run the `mvn idea:idea` command to create an IntelliJ IDEA project.
- Step 4** Import the example project to the IntelliJ IDEA development environment.
  1. Open IntelliJ IDEA and choose **File > Open**.
  2. In the **Open File or Project** dialog box that is displayed, select the Presto sample project folder and click **OK**. After the folder is imported, the **PrestoJDBCExample** class is the JDBC API sample code.
- Step 5** Set the IntelliJ IDEA text file coding format to prevent garbled characters.
  1. On the IntelliJ IDEA menu bar, choose **File > Settings**.
  2. Choose **Editor > File Encodings** from the navigation tree of the **Settings** window. In the "Global Encoding" and "Project Encodings" area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 29-2](#)

Figure 29-2 Setting the IntelliJ IDEA coding format



----End

## 29.2.3 Configuring and Importing the HCatalog API Sample Project

### Scenario

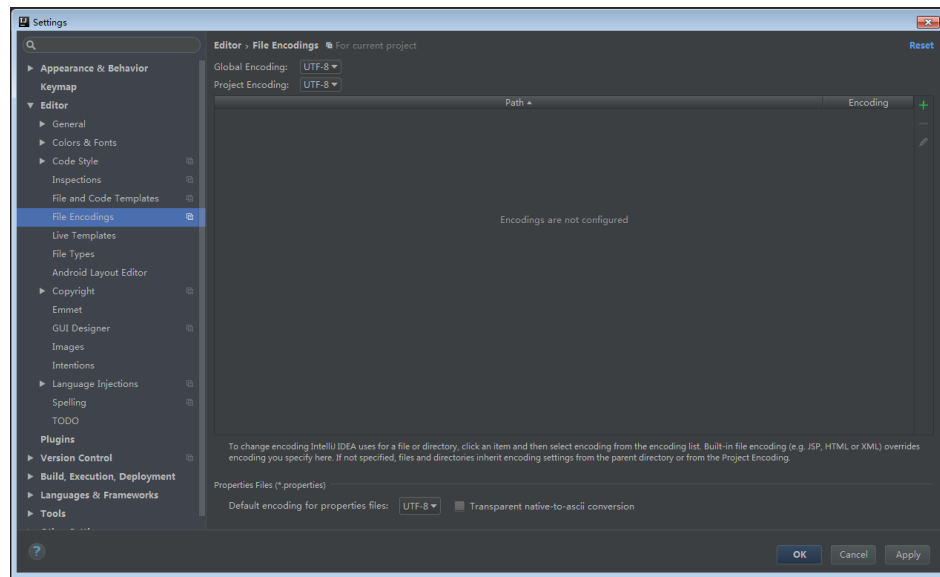
To run the HCatalog API sample code of Presto, you need to perform the following operations. The following example develops an application that uses HCatalog to connect to Presto in the Windows environment.

### Procedure

- Step 1** Obtain the Presto sample project. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the root directory of the Presto sample project, run the `mvn install` command to perform compilation.
- Step 3** In the root directory of the Presto sample project, run the `mvn idea:idea` command to create an IntelliJ IDEA project.
- Step 4** Import the example project to the IntelliJ IDEA development environment.
  1. Open IntelliJ IDEA and choose **File > Open**.
  2. In the **Open File or Project** dialog box that is displayed, select the Presto sample project folder and click **OK**. After the folder is imported, the **HCatalogExample** class in the `com.huawei.bigdata.presto.example` folder is the HCatalog API sample code.
- Step 5** Set the IntelliJ IDEA text file coding format to prevent garbled characters.
  1. On the IntelliJ IDEA menu bar, choose **File > Settings**.
  2. Choose **Editor > File Encodings** from the navigation tree of the **Settings** window. In the "Global Encoding" and "Project Encodings" area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 29-3](#)



Figure 29-3 Setting the IntelliJ IDEA coding format



----End

## 29.3 Developing the Project

### 29.3.1 Typical Application Scenario

#### Scenario Description

Assume that you need to develop a Presto data analysis application to obtain the `call_center` table of TPCDS Catalog provided by Presto.

#### Development Guidelines

##### Step 1 Prepare data.

1. Create three tables: employee information table **employees\_info**, employee contact table **employees\_contact**, and extended employee information table **employees\_info\_extended**.
  - Employee information table **employees\_info** contains fields such as employee ID, name, salary currency, salary, tax category, work place, and hire date. In salary currency, R indicates RMB and D indicates USD.
  - Fields in the **employees\_contact** table include the employee ID, phone number, and email address.
  - Fields in the **employees\_info\_extended** table include the employee ID, name, mobile phone number, e-mail address, salary currency, salary, tax category, and work place. The partition field is the hire date.For details about table creation codes, see [Creating a Table](#).
2. Load employee information to **employees\_info**.  
For details about data loading codes, see [Loading Data](#).  
[Table 29-3](#) provides employee information.

**Table 29-3** Employee information

Employee ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hire Date
1	Wang	R	8000.01	personal income tax&0.05	China:Shenzhen	2014
3	Tom	D	12000.02	personal income tax&0.09	America:NewYork	2014
4	Jack	D	24000.03	personal income tax&0.09	America:Manhattan	2014
6	Linda	D	36000.04	personal income tax&0.09	America:NewYork	2014
8	Zhang	R	9000.05	personal income tax&0.05	China:Shanghai	2014

3. Load employee contact information to **employees\_contact**.

[Table 29-4](#) provides employee contact information.

**Table 29-4** Employee contact information

Employee ID	Phone Number	e-mail
1	135 XXXX XXXX	xxxx@xx.com
3	159 XXXX XXXX	xxxxx@xx.com.cn
4	186 XXXX XXXX	xxxx@xx.org
6	189 XXXX XXXX	xxxx@xxx.cn
8	134 XXXX XXXX	xxxx@xxxx.cn

**Step 2** Analyze data.

For details about data analysis codes, see [Querying Data](#).

- Query contact information of employees whose salaries are paid in USD.
- Query the IDs and names of employees who were hired in 2014, and load the query results to the partition with the hire date of 2014 in the **employees\_info\_extended** table.
- Collect the number of records in the **employees\_info** table.
- Query information about employees whose email addresses end with "cn".

**Step 3** Submit a data analysis task to collect the number of records in the `employees_info` table. For details, see [Example Program Guide](#).

----End

## 29.3.2 Sample Code Description

### Presto JDBC Usage Example

The following code snippet is in the `PrestoJDBCExample` class and is used to connect JDBC to Presto TPCDS Catalog.

```
private static Connection connection;
private static Statement statement;
/**
 * Only when Kerberos authentication enabled, configurations in presto-examples/conf/presto.properties
 * should be set. More details please refer to https://prestodb.io/docs/0.215/installation/jdbc.html.
 */
private static void initConnection(String url, boolean krbsEnabled) throws SQLException {
 if (krbsEnabled) {
 String filePath = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
 File proFile = new File(filePath + "presto.properties"); if (proFile.exists()) {
 Properties props = new Properties();
 try {
 props.load(new FileInputStream(proFile));
 } catch (IOException e) {
 e.printStackTrace();
 }
 connection = DriverManager.getConnection(url, props);
 }
 } else {
 connection = DriverManager.getConnection(url, "presto", null);
 }
 statement = connection.createStatement();
}

private static void releaseConnection() throws SQLException {
 statement.close();
 connection.close();
}

public static void main(String[] args) throws SQLException {
 try {
 /**
 * Replace example_ip with your cluster presto server ip.
 * By default, Kerberos authentication disabled cluster presto service port is 7520, Kerberos
 * authentication enabled cluster presto service port is 7521
 * The postfix /tpcds/sf1 means to use tpcds catalog and sf1 schema, you can use hive catalog as well
 * If Kerberos authentication enabled, set the second param to true.
 * see PrestoJDBCExample#initConnection(java.lang.String, boolean).
 */
 initConnection("jdbc:presto://example_ip:7520/tpcds/sf1", false);
 //initConnection("jdbc:presto://example_ip:7521/tpcds/sf1", true);
 ResultSet resultSet = statement.executeQuery("select * from call_center");
 while (resultSet.next()) {
 System.out.println(resultSet.getString("cc_name") + " : " + resultSet.getString("cc_employees"));
 }
 } catch (SQLException e) {
 e.printStackTrace();
 } finally {
 releaseConnection();
 }
}
```

## 29.4 Commissioning the Application

### Running the JDBC Client and Viewing Results

**Step 1** Run the `mvn clean compile assembly:single` command to generate a JAR file and obtain it from the `target` directory in the project directory, for example, `presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar`.

**Step 2** Create a directory as the running directory in the running and commissioning environment, for example, `/opt/presto_examples` (Linux), and create the `conf` subdirectory in the directory.

Copy `presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar` exported in [Step 1](#) to `/opt/presto_examples`.

**Step 3** In Linux, run the sample program.

```
chmod +x /opt/presto_examples -R
cd /opt/presto_examples
java -jar presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
```

**Step 4** In the CLI, view the query results of the example code.

If the following information is displayed, the sample project execution is successful in Linux.

```
NY Metro : 2
Mid Atlantic : 6
Mid Atlantic : 6
North Midwest : 1
North Midwest : 3
North Midwest : 7
```

----End

## 29.5 Presto APIs

Presto JDBC APIs comply with the Java JDBC driver standard. For details, see JDK 1.7 API. For details about how to use the Presto JDBC, see <https://prestodb.io/docs/current/installation/jdbc.html>.

# 30 Spark2x Development Guide (Security Mode)

---

## 30.1 Overview

### 30.1.1 Application Development Overview

#### Spark Introduction

Spark is a distributed batch processing system as well as an analysis and mining engine. It provides an iterative memory computation framework and supports the development in multiple programming languages, including Scala, Java, and Python. The application scenarios of Spark include:

- Data processing: Spark can process data quickly and has fault tolerance and scalability.
- Iterative computation: Spark supports iterative computation to meet the requirements of multi-step data processing logic.
- Data mining: Based on massive data, Spark can handle complex data mining and analysis and support multiple data mining and machine learning algorithms.
- Streaming processing: Spark supports stream processing at a seconds-level delay and supports multiple external data sources.
- Query analysis: Sparks supports standard SQL query analysis, provides the DSL (DataFrame), and supports multiple external inputs.

This section focuses on the application development guides of Spark, Spark SQL and Spark Streaming.

#### Spark Development API Introduction

Spark supports the development in multiple programming languages, including Scala, Java, and Python. Since Spark is developed in Scala and Scala is easy to read, users are advised to develop Spark application in Scala.

Divided by different languages, the APIs of Spark are listed in [Table 30-1](#).

**Table 30-1** Spark APIs

Function	Description
Scala API	Indicates the API in Scala. For common APIs of Spark Core, Spark SQL and Spark Streaming, see <a href="#">Scala</a> . Since Scala is easy to read, users are advised to use Scala APIs in the program development.
Java API	Indicates the API in Java. For common APIs of Spark Core, Spark SQL and Spark Streaming, see <a href="#">Java</a> .
Python API	Indicates the API in Python. For common APIs of Spark Core, Spark SQL and Spark Streaming, see <a href="#">Python</a> .

Divided by different modes, APIs listed in the preceding table are used in the development of Spark Core and Spark Streaming. Spark SQL supports CLI and JDBCServer for accessing. There are two ways to access the JDBCServer: Beeline and the JDBC client code. For details, see [JDBCServer Interface](#).

 **NOTE**

For spark-sql, spark-shell and spark-submit (which application contains SQL operations), do not use the **proxy user** parameter to submit a task. This is partly because the spark-sql script with the **proxy user** parameter does not support task submission and partly because the sample program mentioned in this document already contains security authentication.

## 30.1.2 Basic Concepts

### Basic Concepts

- **RDD**

Resilient Distributed Dataset (RDD) is a core concept of Spark. It indicates a read-only and partitioned distributed dataset. Partial or all data of this dataset can be cached in the memory and reused between computations.

**RDD Generation**

- An RDD can be generated from the Hadoop file system or other storage systems that are compatible with Hadoop, such as Hadoop Distributed File System (HDFS).
- A new RDD can be transferred from a parent RDD.
- An RDD can be converted from a collection.

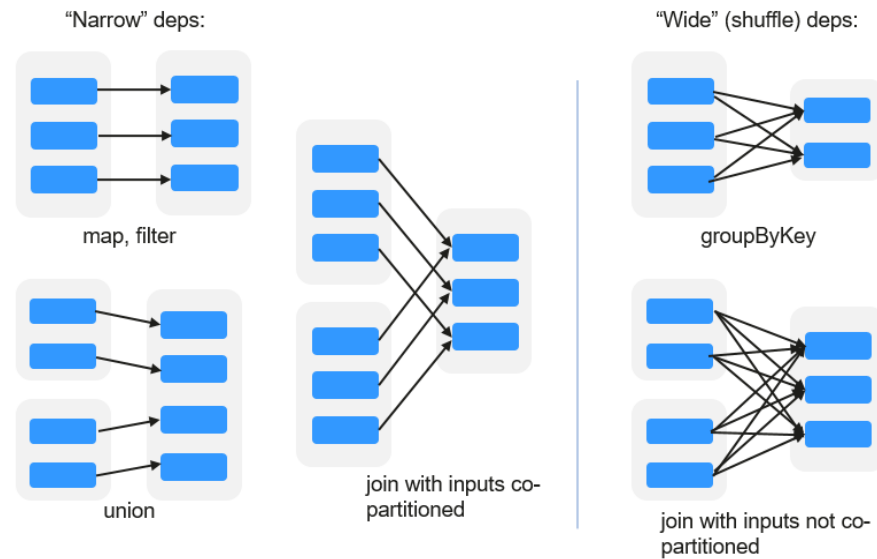
**RDD Storage**

- Users can select different storage levels to store an RDD for reuse. (There are 11 storage levels to store an RDD.)
- The current RDD is stored in the memory by default. When the memory is insufficient, the RDD overflows to the disk.

- **RDD Dependency**

The RDD dependency includes the narrow dependency and wide dependency.

**Figure 30-1** RDD dependency



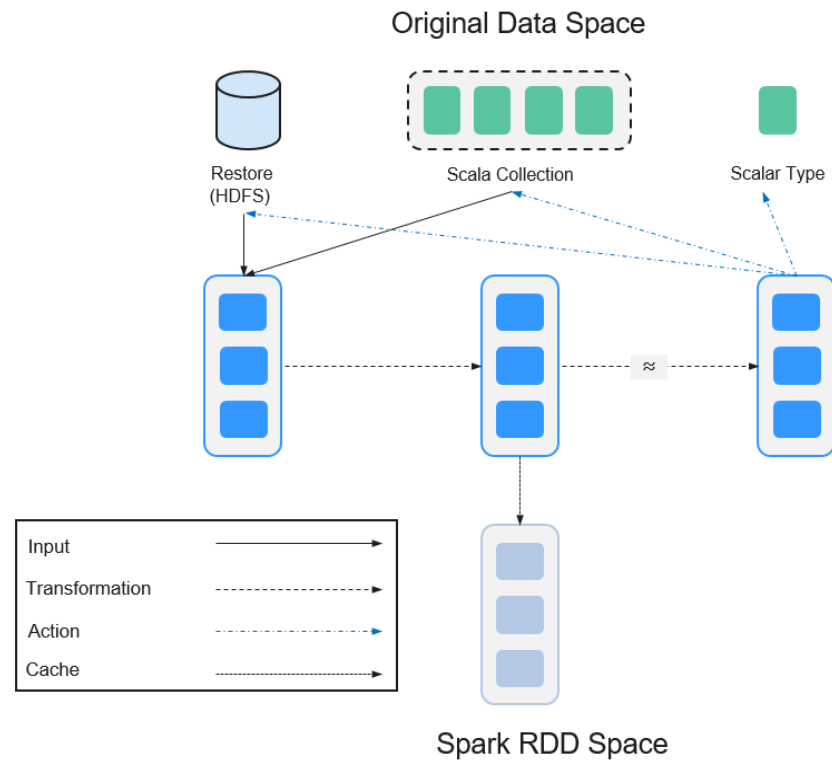
- **Narrow dependency:** Each partition of the parent RDD is used by at most one partition of the child RDD partition.
- **Wide dependency:** Partitions of the child RDD depend on all partitions of the parent RDD due to shuffle operations.

The narrow dependency facilitates the optimization. Logically, each RDD operator is a fork/join process. Fork the RDD to each partition, and then perform the computation. After the computation, join the results, and then perform the fork/join operation on next operator. It takes a long period of time to directly translate the RDD to physical implementation. There are two reasons: Each RDD (even the intermediate results) must be physicalized to the memory or storage, which takes time and space; the partitions can be joined only when the computation of all partitions is complete (if the computation of a partition is slow, the entire join process is slowed down). If the partitions of the child RDD narrowly depend on the partitions of the parent RDD, the two fork/join processes can be combined to optimize the entire process. If the relationship in the continuous operator sequence is narrow dependency, multiple fork/join processes can be combined to reduce the time for waiting and improve the performance. This is called pipeline optimization in Spark.

- **Transformation and Action (RDD Operations)**

Operations on RDD include transformation (the returned value is an RDD) and action (the returned value is not an RDD). [Figure 30-2](#) shows the process. The transformation is lazy, which indicates that the transformation from one RDD to another RDD is not immediately executed. Spark only records the transformation but does not execute it immediately. The real computation is started only when the action is started. The action returns results or writes the RDD data into the storage system. The action is the driving force for Spark to start the computation.

**Figure 30-2** RDD operation



The data and operation model of RDD are quite different from those of Scala.

```
val file = sc.textFile("hdfs://...")
val errors = file.filter(_.contains("ERROR"))
errors.cache()
errors.count()
```

- The **textFile** operator reads log files from HDFS and returns **file** (as an RDD).
- The filter operator filters rows with ERROR and assigns them to errors (a new RDD). The filter operator is a transformation.
- The cache operator caches errors for future use.
- The count operator returns the number of rows of errors. The count operator is an action.

**Transformation includes the following types:**

- The RDD elements are regarded as simple elements:  
The input and output have the one-to-one relationship, and the partition structure of the result RDD remains unchanged, for example, map.  
The input and output have the one-to-many relationship, and the partition structure of the result RDD remains unchanged, for example, flatMap (one element becomes a sequence containing multiple elements and then flattens to multiple elements).
- The input and output have the one-to-one relationship, but the partition structure of the result RDD changes, for example, union (two RDDs integrates to one RDD, and the number of partitions becomes the sum of the number of partitions of two RDDs) and coalesce (partitions are reduced).



Operators of some elements are selected from the input, such as filter, distinct (duplicate elements are deleted), subtract (elements only exist in this RDD are retained), and sample (samples are taken).

- The RDD elements are regarded as Key-Value pairs.

Perform the one-to-one calculation on the single RDD, such as mapValues (the partition mode of the source RDD is retained, which is different from map).

Sort the single RDD, such as sort and partitionBy (partitioning with consistency, which is important to the local optimization).

Restructure and reduce the single RDD based on key, such as groupByKey and reduceByKey.

Join and restructure two RDDs based on key, such as join and cogroup.

 NOTE

The later three operations involve sorting and are called shuffle operations.

**Action is classified into the following types:**

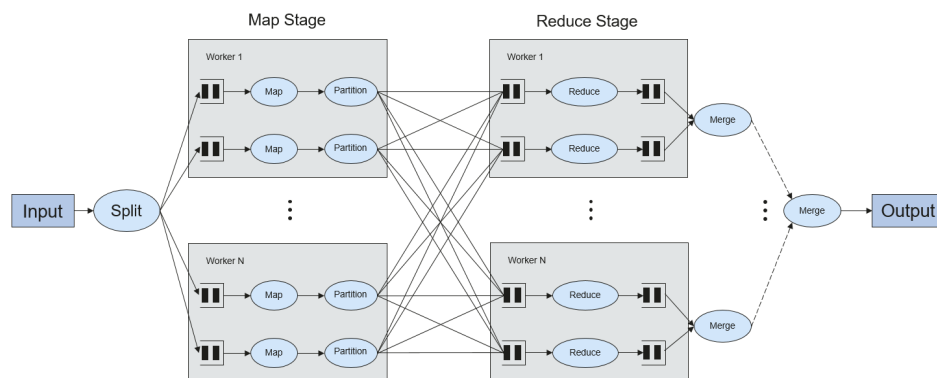
- Generate scalar configuration items, such as count (the number of elements in the returned RDD), reduce, fold/aggregate (the number of scalar configuration items that are returned), and take (the number of elements before the return).
- Generate the Scala collection, such as collect (import all elements in the RDD to the Scala collection) and lookup (look up all values corresponds to the key).
- Write data to the storage, such as saveAsTextFile (which corresponds to the preceding textFile).
- Check points, such as checkpoint. When Lineage is long (which occurs frequently in graphics computation), it takes a long period of time to execute the whole sequence again when a fault occurs. In this case, checkpoint is used as the check point to write the current data to stable storage.

- **Shuffle**

Shuffle is a specific phase in the MapReduce framework, which is located between the Map phase and the Reduce phase. If the output results of Map are to be used by Reduce, each output result must be hashed based on the key and distributed to the corresponding Reducer. This process is called Shuffle. Shuffle involves the read and write of the disk and the transmission of the network, so that the performance of Shuffle directly affects the operation efficiency of the entire program.

The following figure describes the entire process of the MapReduce algorithm.

**Figure 30-3** Algorithm process



Shuffle is a bridge connecting data. The following describes the implementation of shuffle in Spark.

Shuffle divides the Job of a Spark into multiple stages. The former stages contain one or multiple ShuffleMapTasks, and the last stage contains one or multiple ResultTasks.

- **Spark Application Structure**

The Spark application structure includes the initial SparkContext and the main program.

- Initial SparkContext: constructs the operation environment of the Spark application.

Constructs the SparkContext object. For example:

```
new SparkContext(master, appName, [SparkHome], [jars])
```

Parameter description

**master:** indicates the link string. The link modes include local, YARN-cluster, and YARN-client.

**appName:** indicates the application name.

**SparkHome:** indicates the directory where Spark is installed in the cluster.

**jars:** indicates the code and dependency package of the application.

- Main program: processes data.

For submitting applications details, see <https://spark.apache.org/docs/3.1.1/submitting-applications.html>

- **Spark Shell Command**

The basic Spark shell command supports the submitting of the Spark application. The Spark shell command is

```
./bin/spark-submit \
--class <main-class> \
--master <master-url> \
... # other options \
<application-jar> \
[application-arguments]
```

Parameter description:

--class: indicates the name of the class of the Spark application.

--master: indicates the master that the Spark application links, such as YARN-client and YARN-cluster.

application-jar: indicates the path of the jar package of the Spark application.

application-arguments: indicates the parameter required to submit the Spark application. (This parameter can be empty.)

- **Spark JobHistory Server**

The Spark web UI is used to monitor the details in each phase of the Spark framework of a running or historical Spark job and provide the log display, which helps users to develop, configure, and optimize the job in more fine-grained units.

## Spark SQL Basic Concepts

### DataSet

A DataSet is a strongly typed collection of domain-specific objects that can be transformed in parallel using functional or relational operations. Each Dataset also has an untyped view called a DataFrame, which is a Dataset of Row.

DataFrame is a structured distributed data set composed of several columns, which is similar to a table in the relationship database or the data frame in R/Python. DataFrame is a basic concept in Spark SQL, and can be created by using multiple methods, such as structured data set, Hive table, external database, or RDD.

## Spark Streaming Basic Concepts

### DStream

DStream (Discretized Stream) is an abstract concept provided by the Spark Streaming.

DStream is a continuous data stream which is obtained from the data source or transformed and generated by the inflow. In essence, a DStream is a series of continuous RDDs. The RDD is a distributed dataset which can be read only and divided into partitions.

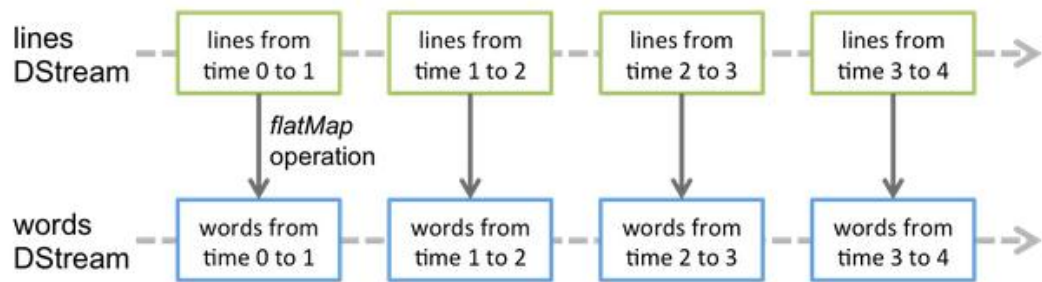
Each RDD in DStream contains the data of a partition as shown in [Figure 30-4](#).

**Figure 30-4** Relationship between DStream and RDD



All operators applied in the DStream are translated to the operations in the lower RDD, as shown in [Figure 30-5](#). The transformation of the lower RDDs is calculated by using the Spark engine. Most operation details are concealed in DStream operators and High-level APIs are provided for developers.

**Figure 30-5** DStream operator transfer



## Structured Streaming Basic Concepts

- **Input Source**

Input data sources. Data sources must support data replay based on the offset. Different data sources have different fault tolerance capabilities.

- **Sink**

Data output. Sink must support idempotence write operations. Different Sinks have different fault tolerance capabilities.

- **outputMode**

Result output mode, which can be:

- Complete Mode: The entire updated result table will be written to the external storage. It is up to the storage connector to decide how to handle writing of the entire table.
- Append Mode: If an interval is triggered, only the new rows appended in the Result Table will be written into an external system. This mode is applicable only to a result set that has already existed and will not be updated.
- Update Mode: If an interval is triggered, only updated data in the result table will be written into an external system, which is the difference between the Complete Mode and Update Mode.

- **Trigger**

Output trigger. Currently, the following trigger types are supported:

- Default: Micro-batch mode. After a batch is complete, the next batch is automatically executed.
- Specific interval: Processing is performed at a specific interval.
- One-time execution: Query is performed only once.
- Continuous mode: This is an experimental feature. In this mode, the stream processing delay can be decreased to 1 ms.

Structured Streaming supports the micro-batch mode and continuous mode. The micro-batch mode cannot ensure low-delay but has a larger throughput within the same time. The continuous mode is suitable for data processing requiring millisecond-level delay, which is an experimental feature.

### NOTE

In the current version, if the stream-to-batch joins function is required, **outputMode** must be set to **Append Mode**.

Figure 30-6 Running process in micro-batch mode

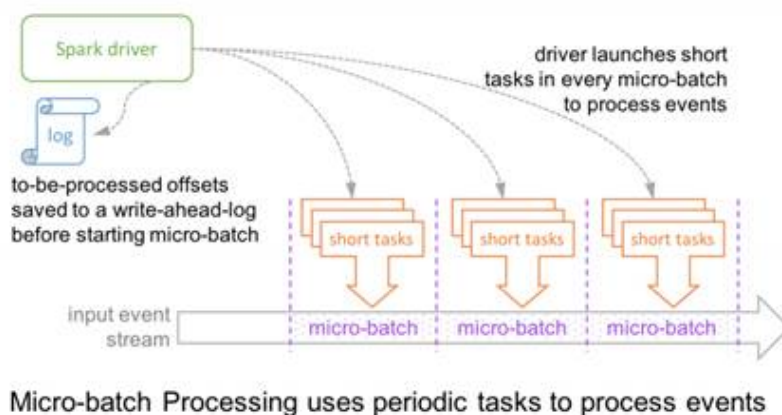
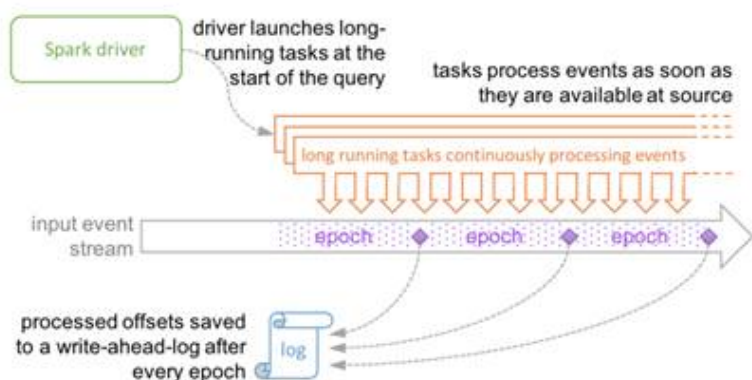


Figure 30-7 Running process in continuous mode



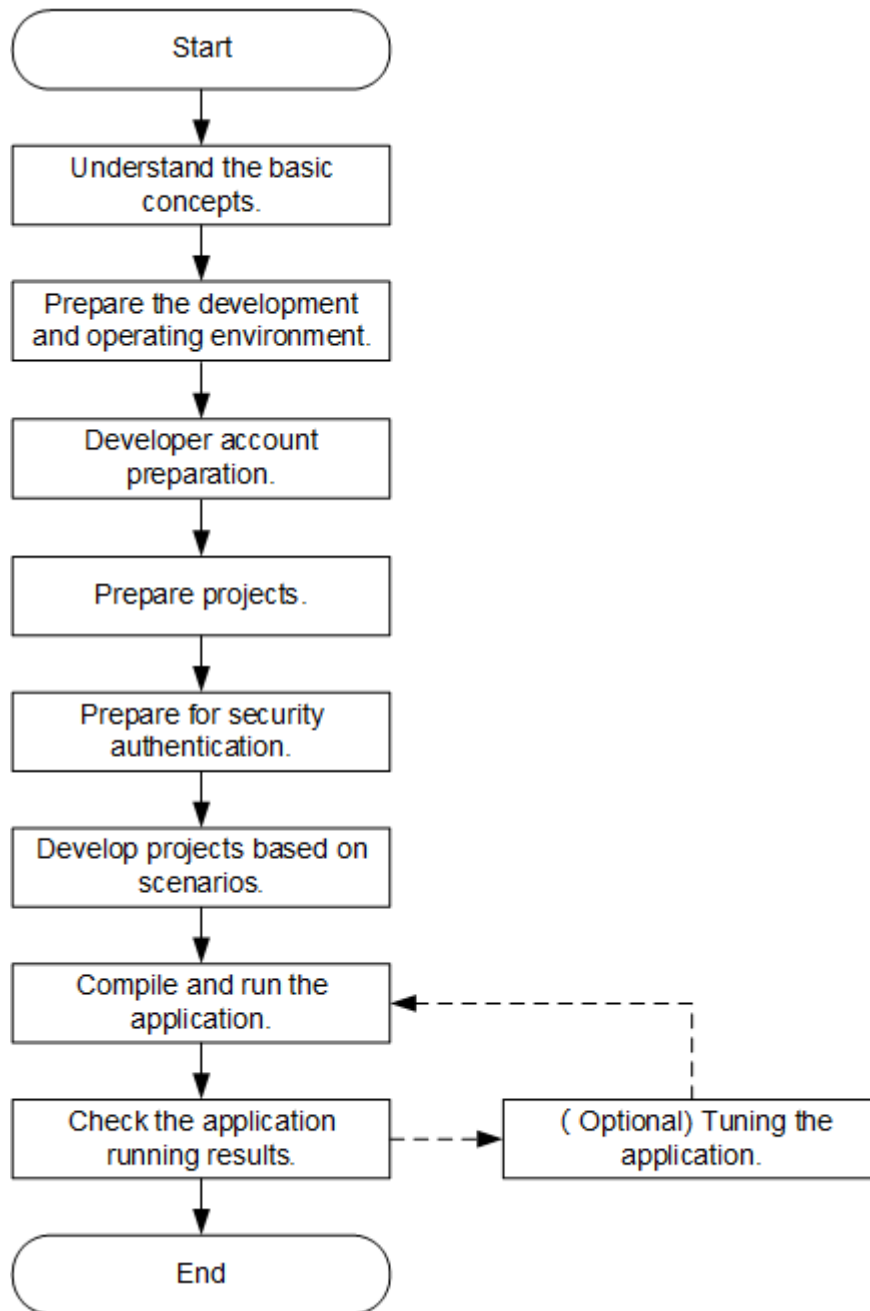
### 30.1.3 Development Process

#### Development Process of a Spark Application

Spark includes Spark Core, Spark SQL and Spark Streaming, whose development processes are the same.

Figure 30-8 and Table 30-2 describe the stages in the development process.

**Figure 30-8** Spark development process



**Table 30-2** Description of Spark development process

Stage	Description	Reference
Understand the basic concepts.	Before the application development, the basic concepts of Spark are required to be understood. Choose the concepts required to be understood based on the actual scenario. The basic concepts include the basic concept of Spark Core, basic concept of Spark SQL and basic concept of Spark Streaming.	<a href="#">Basic Concepts</a>

Stage	Description	Reference
Prepare the development and operating environment.	The Spark application is developed in Scala, Java, and Python. The IDEA tool is recommended to prepare development environments in different languages based on the reference. The running environment of Spark is the Spark client. Install and configure the client based on the reference.	<a href="#">Preparing for Development and Operating Environment</a>
Prepare a developer account.	The development account is used to run the sample project. Only the developer user with permission to access HDFS, YARN, Kafka, and Hive is allowed to run Spark sample projects.	<a href="#">Preparing a Developer Account</a>
Prepare projects.	Spark provides sample projects in various scenarios. Sample projects can be imported for studying. Or you can create a new Spark project based on the reference.	<a href="#">Configuring and Importing Sample Projects</a> <a href="#">Creating a New Project (Optional)</a>
Prepare for safety authentication.	If a safe cluster is used, the safety authentication must be performed.	<a href="#">Preparing for Security Authentication</a>
Develop projects based on scenarios	Sample projects in different languages including Scala, Java, and Python are provided. Sample projects in different scenarios including Streaming, SQL, JDBC client program, and Spark on HBase are also provided. This helps users to better understand the programming interfaces of all Spark components quickly.	<a href="#">Developing the Project</a>
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	<a href="#">Compiling and Running the Application</a>
Check the application running results.	Application running results are stored in the directory specified by users. Users can also check the running results through the UI.	<a href="#">Checking the Commissioning Result</a>
Tune the application.	Based on the application running results, tune the application to meet the requirements of the service scenario. After application tuning, compile and run the application again.	<a href="#">Spark2x Performance Optimization</a>

## 30.2 Preparing for the Environment

### 30.2.1 Preparing for Development and Operating Environment

- Spark2x applications can be developed in Scala, Java, and Python. [Table 30-3](#) describes the the development and operating environment to be prepared.

**Table 30-3** Development environment

Preparation Item	Description
OS	<ul style="list-style-type: none"> <li>• Development environment: Windows OS. Windows 7 or later is supported.</li> <li>• Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the runtime environment must be able to communicate with the cluster service plane network.</li> </ul>
JDK installation	<p>Basic configuration for the Java/Scala development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> <li>• For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> <li>- Oracle JDK versions: 1.8</li> <li>- IBM JDK versions: 1.8.5.11</li> </ul> </li> <li>• For TaiShan nodes that run clients, the following JDK can be used: <ul style="list-style-type: none"> <li>- OpenJDK: 1.8.0_272</li> </ul> </li> </ul> <p><b>NOTE</b></p> <p>The server supports only TLS V1.2 or later to ensure security.</p> <p>By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls">https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls</a>.</p>



Preparation Item	Description
IntelliJ IDEA installation and configuration	It is a tool used to develop Spark applications. Version 2019.1 or other compatible versions are recommended. <b>NOTE</b> <ul style="list-style-type: none"> <li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li> <li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li> <li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li> <li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li> </ul>
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.
Scala installation	It is the basic configuration for the Scala development environment. The required version is 2.12.10.
Scala plug-in installation	It is the basic configuration for the Scala development environment. The required version is 2018.2.11 or other compatible versions.
Editra installation	Editra is an editor in the Python development environment and is used to compile Python programs. You can also use other IDEs for Python programming.
Developer account preparation	See <a href="#">Preparing a Developer Account</a> for configuration.
7-zip	Used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-Zip 16.04 is supported.
Python installation	Its version must be 2.6.6 or later.

## Preparing a Runtime Environment

During application development, you need to prepare the environment for code running and commissioning to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. [Log in to the FusionInsight Manager portal](#) and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Configuration Files Only**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86

architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is

**FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain the **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file, and then decompress it to the **D:**

**\FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles\Spark2x\config** and manually import the configuration file to the configuration file directory (usually the **resources** folder) of the Hive sample project. The keytab file obtained during the operation of [Preparing a Developer Account](#) is also stored in this directory. [Table 30-4](#) describes the main configuration files.

**Table 30-4** Configuration files

File	Function
carbon.properties	CarbonData configuration file
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.
hbase-site.xml	Configures HBase parameters.
hive-site.xml	Configures Hive parameters.
jaas-zk.conf	Java authentication configuration file
log4j-executor.properties	executor log configuration file
mapred-site.xml	Hadoop mapreduce configuration file
ranger-spark-audit.xml	Ranger audit log configuration file
ranger-spark-security.xml	Ranger permission management configuration file
yarn-site.xml	Configures Yarn parameters.
spark-defaults.conf	Configures Spark2x parameters.
spark-env.sh	Spark2x environment variable configuration file
user.keytab	Provides HDFS user information for Kerberos security authentication.
krb5.conf	Provides Kerberos server configuration information.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client in a directory, for example, **/opt/client**, on the node. Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. [Log in to the FusionInsight Manager portal](#). Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Spark2x/config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/client/conf**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Spark2x/config/* root@IP address of the client node:/opt/client/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 30-5](#) describes the main configuration files.

**Table 30-5** Configuration files

File	Function
carbon.properties	CarbonData configuration file

File	Function
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.
hbase-site.xml	Configures HBase parameters.
hive-site.xml	Configures Hive parameters.
jaas-zk.conf	Java authentication configuration file
log4j-executor.properties	executor log configuration file
mapred-site.xml	Hadoop mapreduce configuration file
ranger-spark-audit.xml	Ranger audit log configuration file
ranger-spark-security.xml	Ranger permission management configuration file
yarn-site.xml	Configures Yarn parameters.
spark-defaults.conf	Configures Spark2x parameters.
spark-env.sh	Spark2x environment variable configuration file
user.keytab	Provides HDFS user information for Kerberos security authentication.
krb5.conf	Provides Kerberos server configuration information.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

## 30.2.2 Configuring and Importing Sample Projects

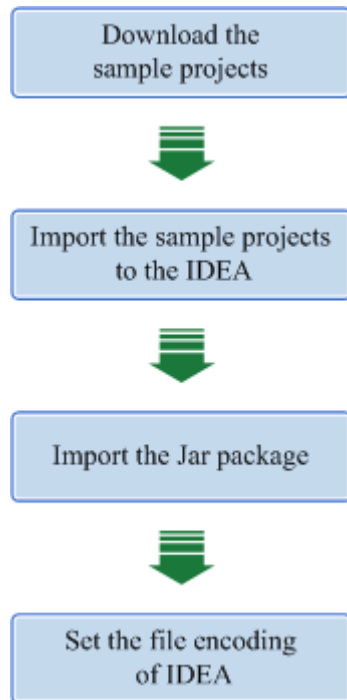
### Scenario

Spark provides sample projects for multiple scenarios, including Java projects and Scala projects. This helps users to learn Spark projects quickly.

Import methods of Java and Scala projects are the same. Sample projects developed by using the Python do not need to be imported, and you only need to open the Python file (\*.py).

The import of Java sample codes is used as a sample in the following procedure. [Figure 30-9](#) shows the procedure if importing sample projects.

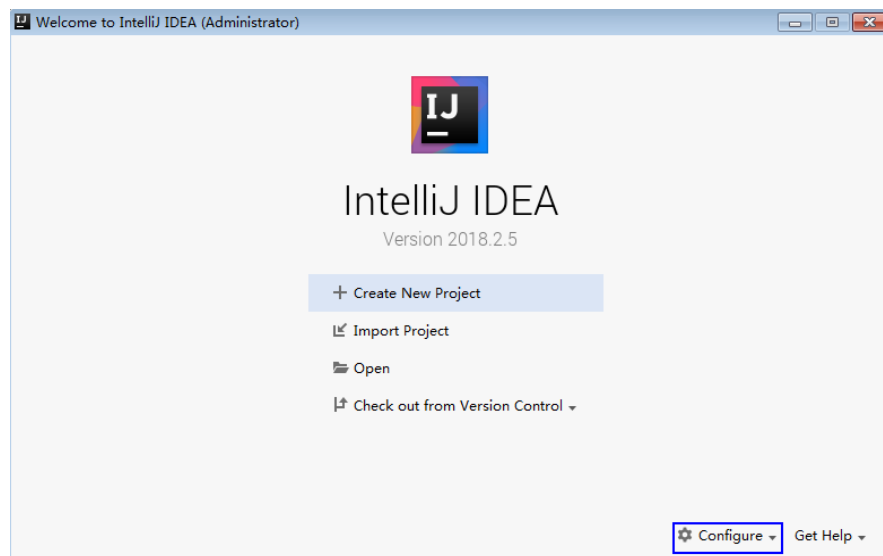
**Figure 30-9** Procedure of importing sample projects



## Procedure

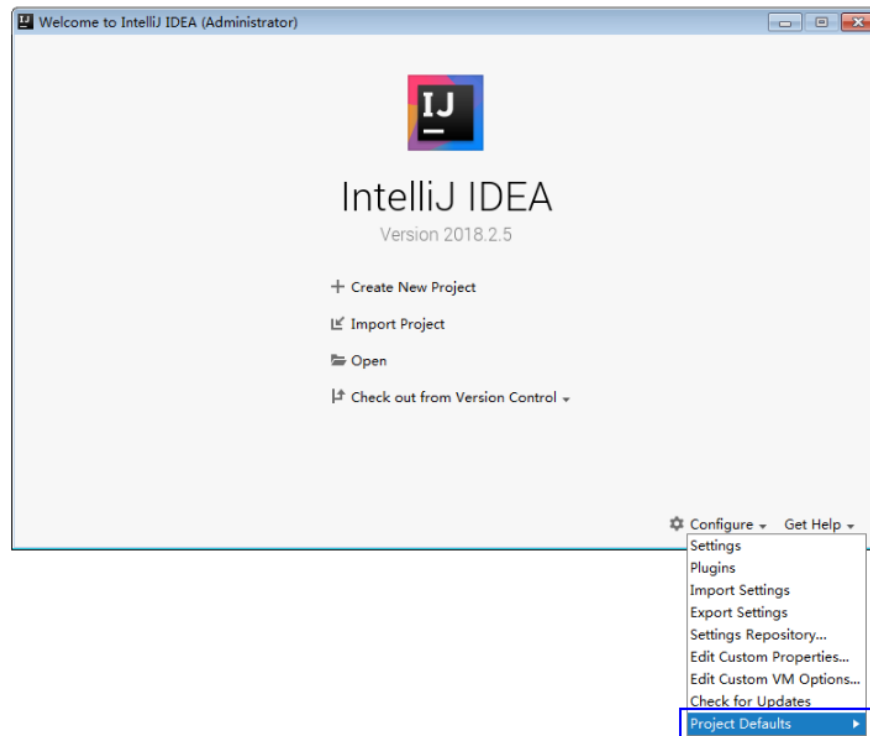
- Step 1** Obtain multiple sample projects such as Scala and Spark Streaming in the **sparksecurity-examples** folder in the **spark-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** After the IntelliJ IDEA and JDK are installed, configure the JDK in IntelliJ IDEA.
  1. Start the IntelliJ IDEA and select **Configure**.

**Figure 30-10** Quick Start



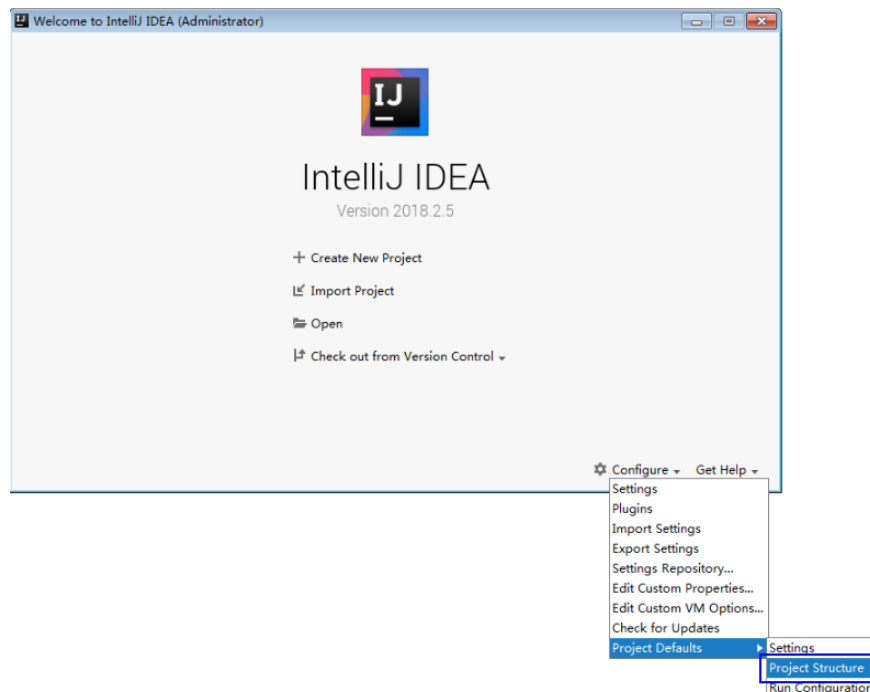
2. Select **Project Defaults** from the **Configure** drop-down list.

Figure 30-11 Configure



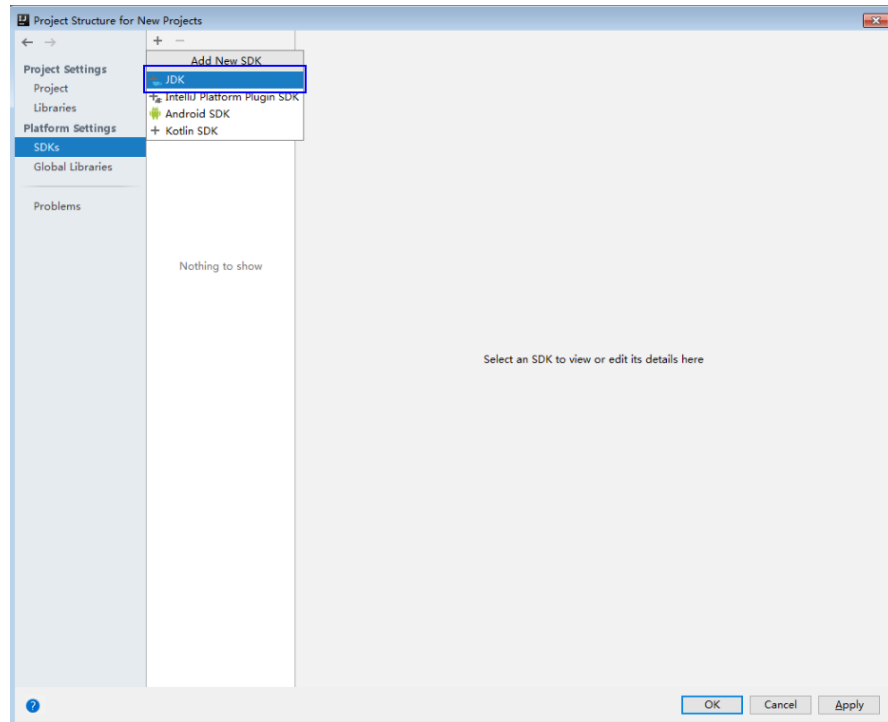
3. Select **Project Structure** from **Project Defaults**.

Figure 30-12 Project Defaults



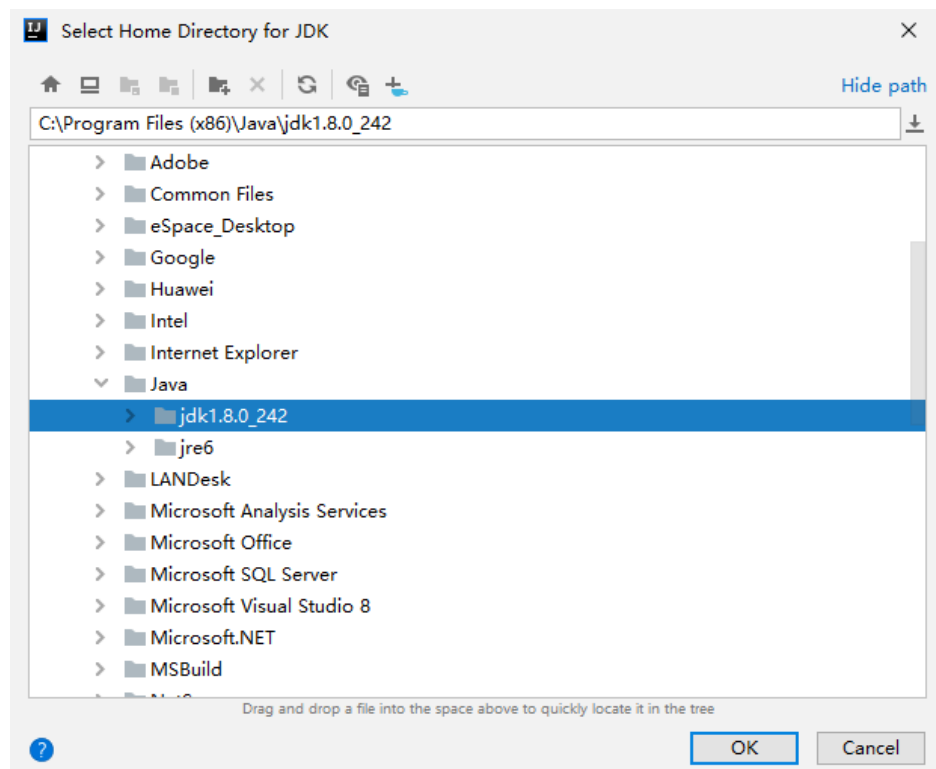
4. On the displayed **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 30-13 Adding the JDK



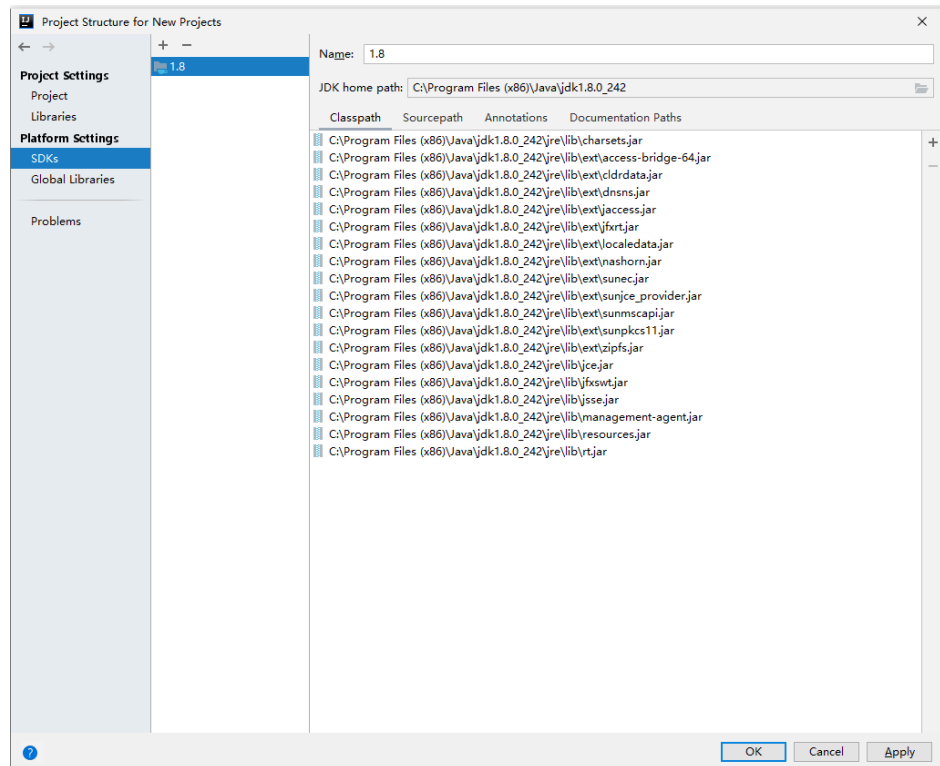
5. In the displayed **Select Home Directory for JDK** window, select a home directory for JDK and click **OK**.

Figure 30-14 Selecting a home directory for the JDK



6. After selecting the JDK, click **OK** to complete the configuration.

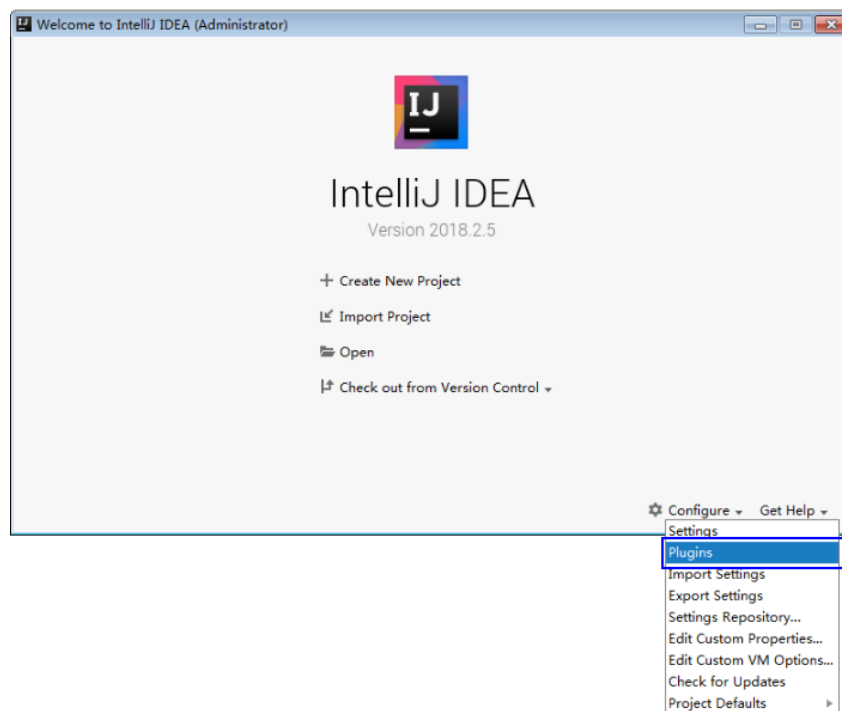
Figure 30-15 Completing the configuration



**Step 3** (Optional) If the Scala development environment is used, install the Scala plug-in in IntelliJ IDEA.

1. Select **Plugins** from the **Configure** drop-down list.

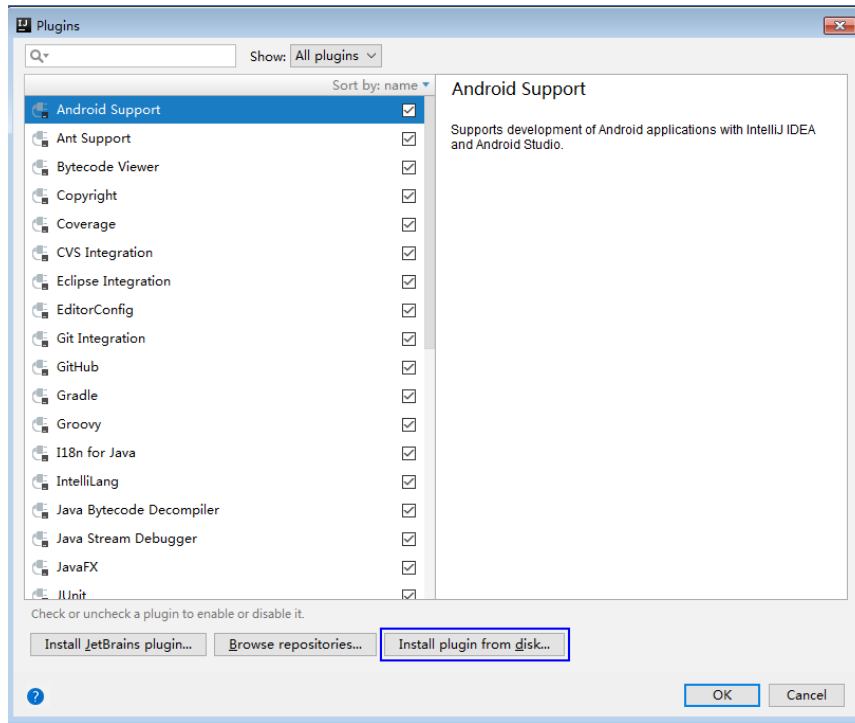
Figure 30-16 Plugins



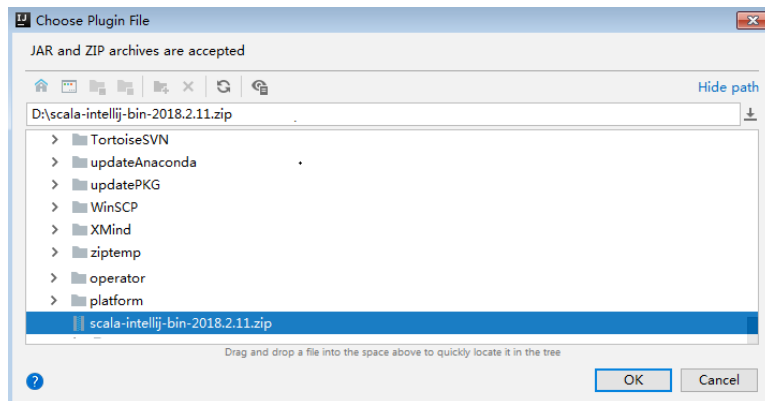


2. On the **Plugins** page, select **Install plugin from disk**.

**Figure 30-17** Install plugin from disk

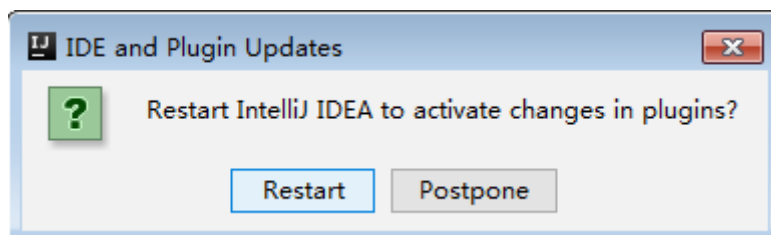


3. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.



4. On the **Plugins** page, click **Apply** to install the Scala plugin.
5. On the displayed **Plugins Changed** page, click **Restart** for the configuration to take effect.

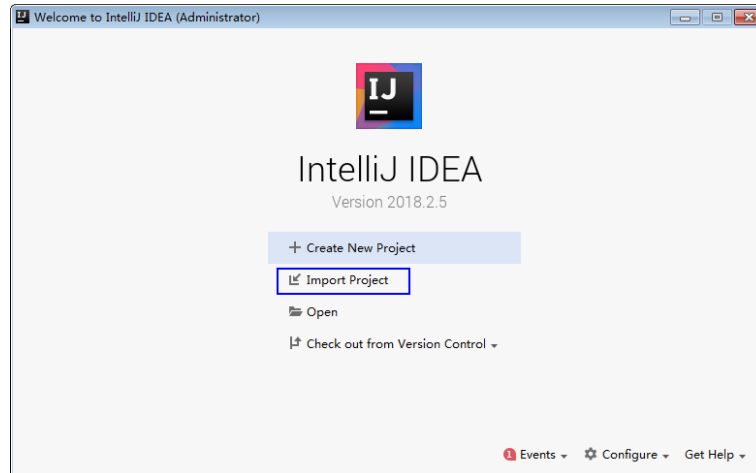
**Figure 30-18** Plugins Changed



**Step 4** Import the Java sample projects to the IDEA.

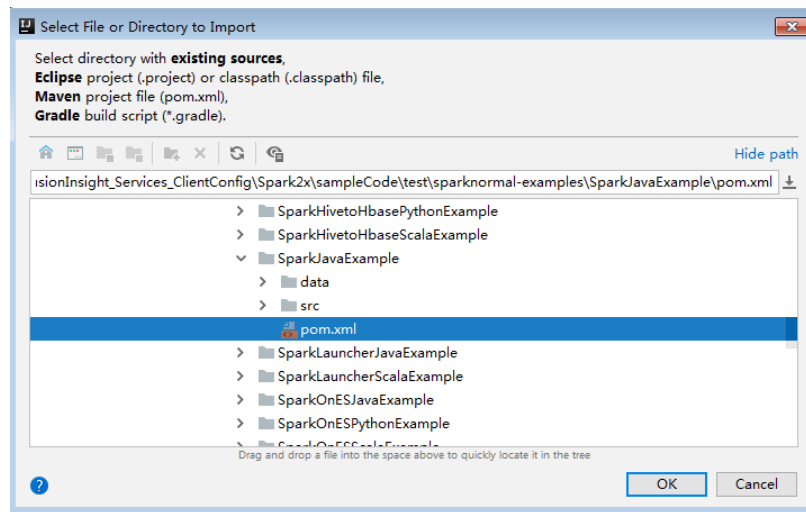
1. Start the IntelliJ IDEA. On the **Quick Start** page, select **Import Project**.  
Or, for the used IDEA tool, add projects directly from the IDEA homepage. Select **File > Import project...** to import projects.

**Figure 30-19** Import Project (on the Quick Start page)



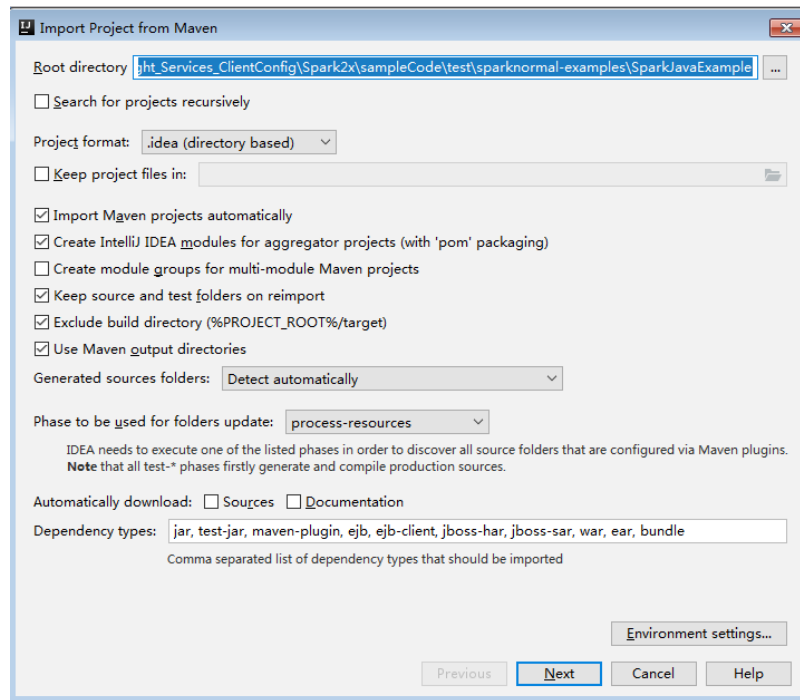
2. Select the directory to store the imported projects and the pom file, and click **OK**.

**Figure 30-20** Select File or Directory to Import



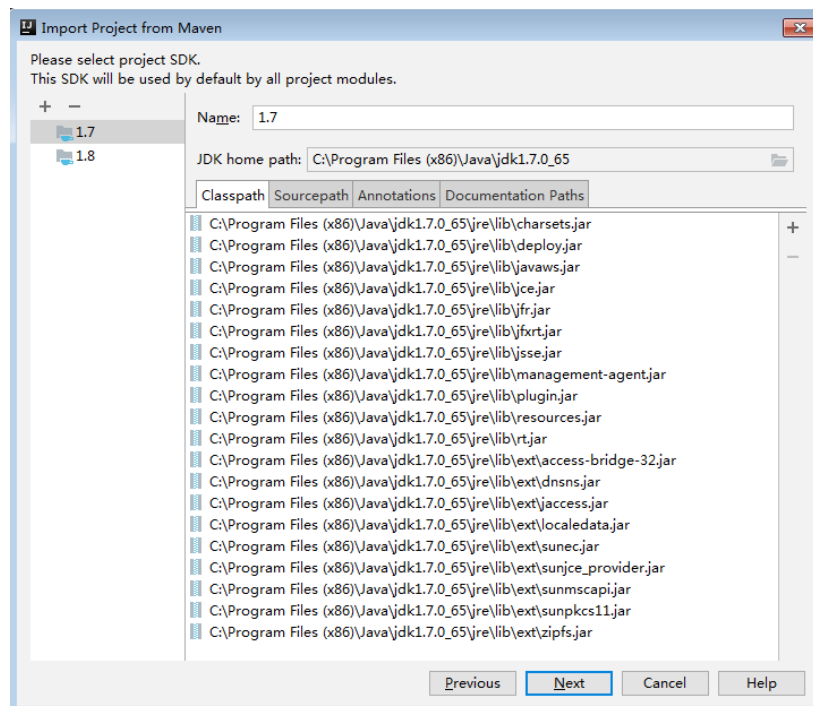
3. Confirm the import directory and project name, and click **Next**.

Figure 30-21 Import Project from Maven



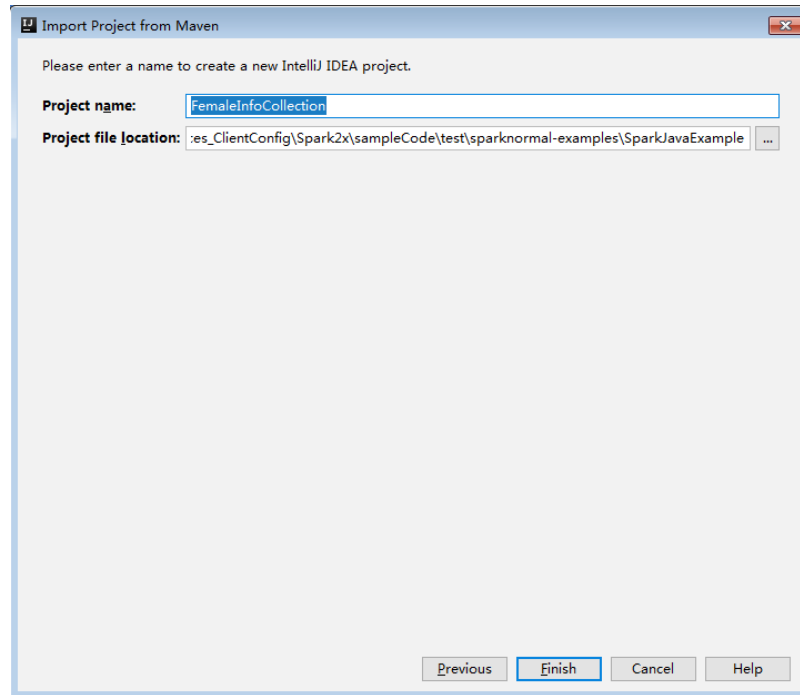
4. Select the projects to import and click **Next**.
5. Confirm the project JDK and click **Next**.

Figure 30-22 Select project SDK



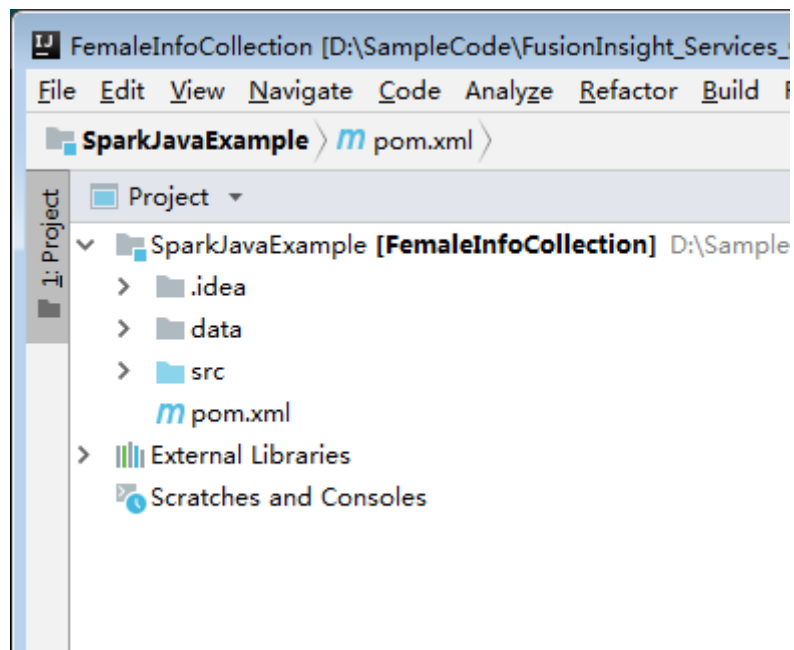
6. Confirm the project name and project file location, and click **Finish** to complete the import.

**Figure 30-23** Confirm the project name and file location



7. After the import, the imported projects are displayed on the IDEA homepage.

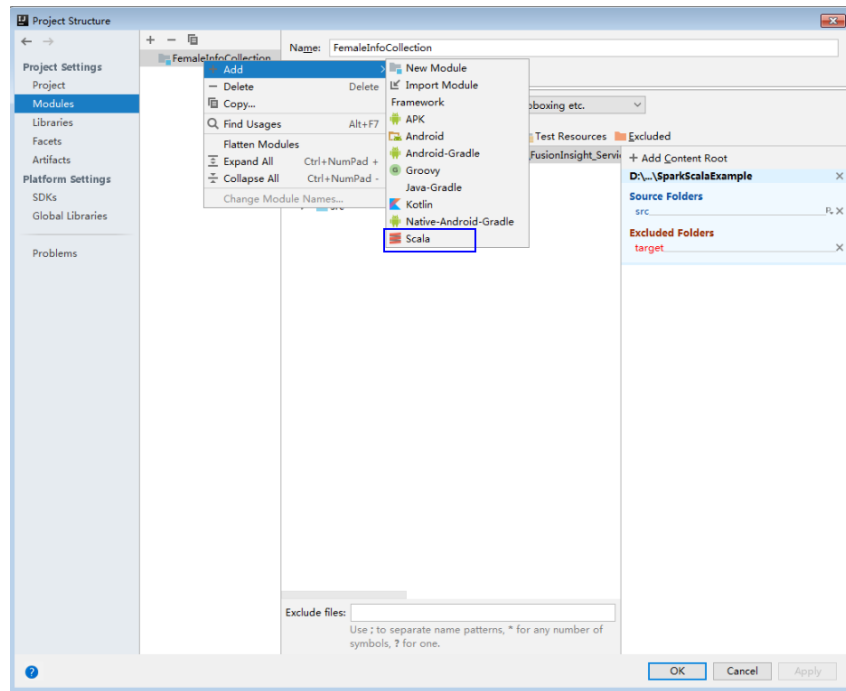
**Figure 30-24** Imported projects



**Step 5** (Optional) If a sample application developed in Scala is imported, configure the language for the project.

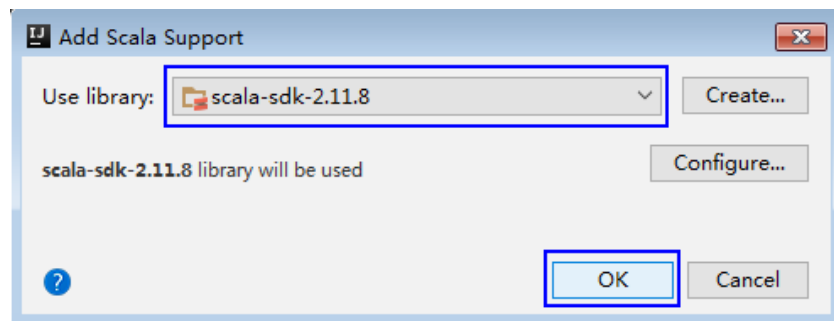
1. On the main page of the IDEA, choose **File > Project Structures...** to access the **Project Structure** page.
2. Choose **Modules**, right-click the project name, and choose **Add > Scala**.

Figure 30-25 Selecting the Scala language



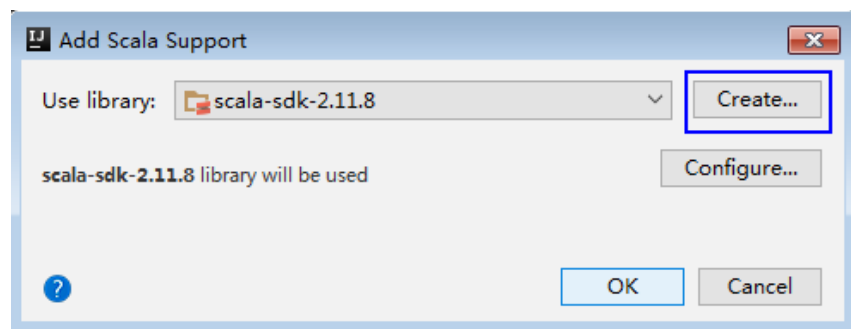
3. Wait until IDEA identifies Scala SDK, select the dependency JAR packages in the **Add Scala Support** dialog box, and then click **OK**

Figure 30-26 Add Scala Support



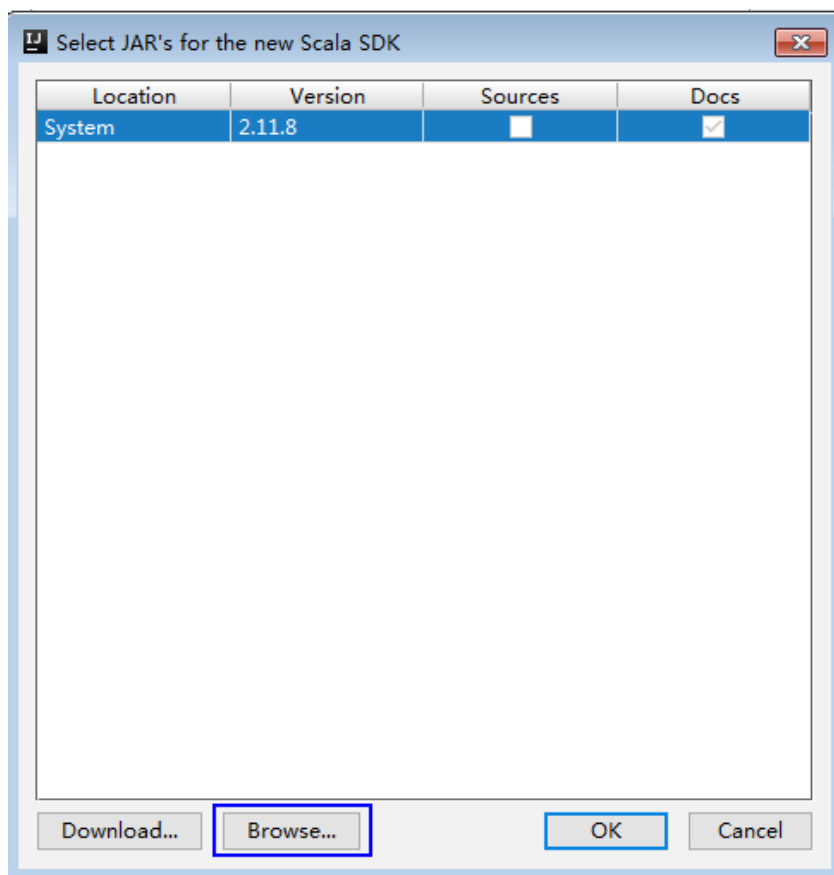
4. If IDEA fails to identify Scala SDK, you are required to create a Scala SDK.
  - a. Click **Create...**

Figure 30-27 Create...



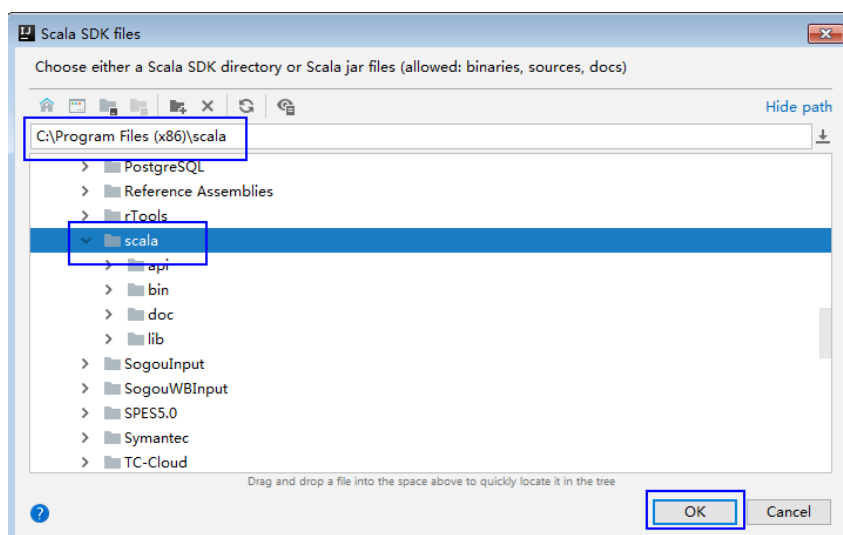
- b. On the **Select JAR's for the new Scala SDK** page, click **Browse...**

**Figure 30-28** Select JAR's for the new Scala SDK



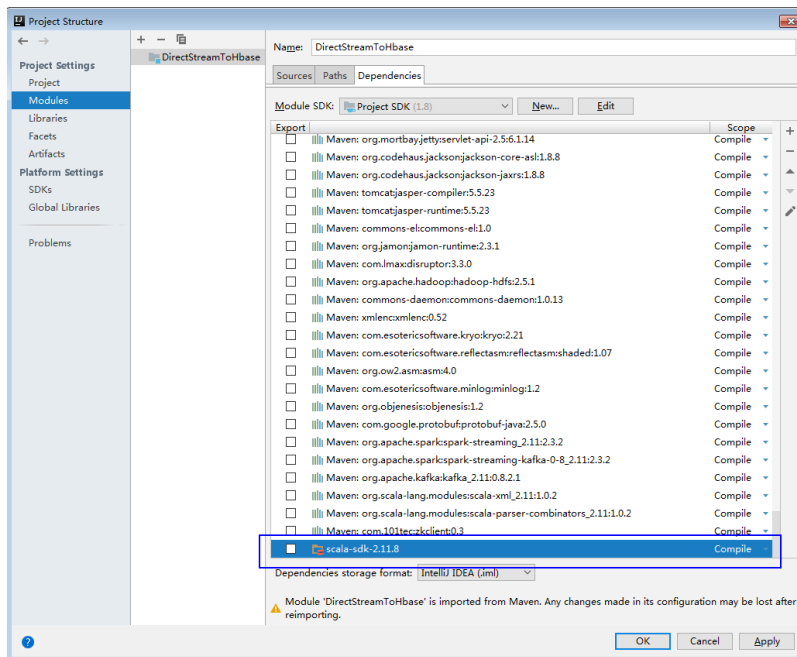
- c. On the **Scala SDK files** page, select the **scala sdk** directory, and then click **OK**.

**Figure 30-29** Scala SDK files



- 5. Click **OK**.

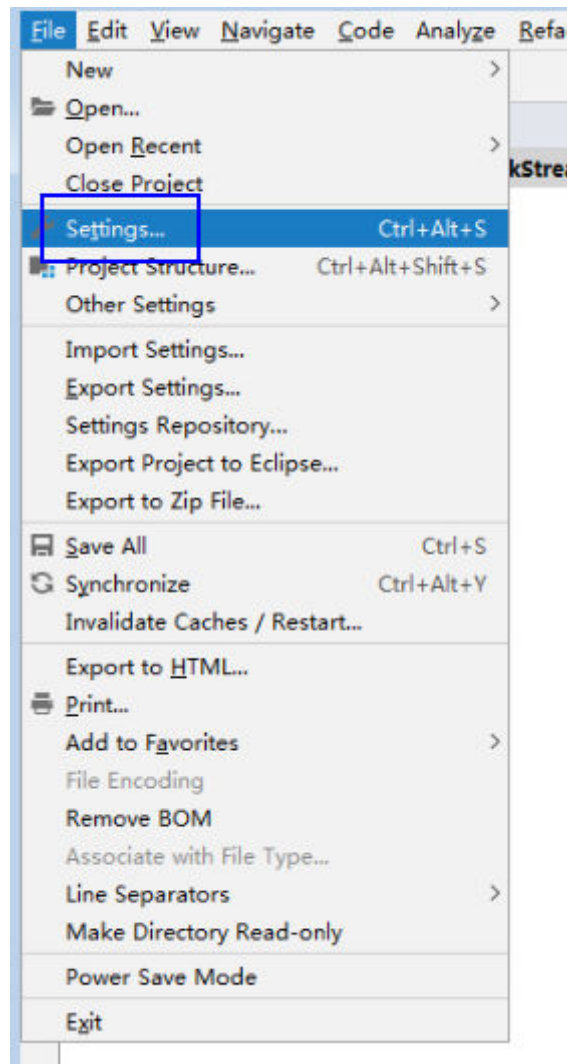
Figure 30-30 Successful configuration



**Step 6** Set the file encoding of IDEA and solve the display of garble characters.

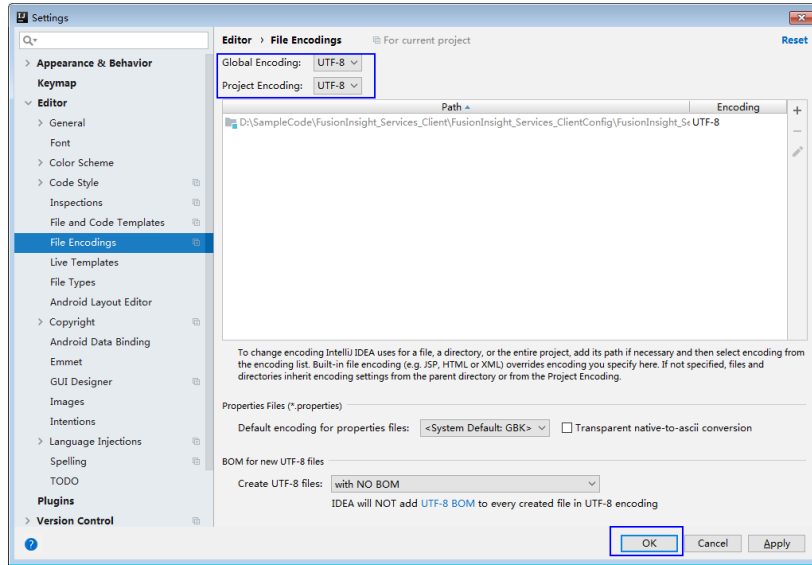
1. On the IDEA homepage, choose **File > Settings...**

Figure 30-31 Choosing Settings



2. Configure the encoding.
  - a. On the **Settings** page, choose **Editor > File Encodings**.
  - b. In the **Global Encoding** and **Project Encoding** drop-down lists, select **UTF-8**, respectively.
  - c. Click **Apply**.
  - d. Click **OK** to complete the encoding configuration.





----End

## Sample Code Path Description

Table 30-6 Sample code path description

Sample Code Project	Sample Name	Sample Development Language
SparkJavaExample	Spark Core Project	Java
SparkScalaExample	Spark Core Project	Scala
SparkPythonExample	Spark Core Project	Python
SparkSQLJavaExample	Spark SQL Project	Java
SparkSQLScalaExample	Spark SQL Project	Scala
SparkSQLPythonExample	Spark SQL Project	Python
SparkThriftServerJavaExample	Accessing the Spark SQL Through JDBC	Java
SparkThriftServerScalaExample	Accessing the Spark SQL Through JDBC	Scala
SparkOnHbaseJavaExample-AvroSource	Spark on HBase-Performing Operations on Data in Avro Format	Java
SparkOnHbaseScalaExample-AvroSource	Spark on HBase-Performing Operations on Data in Avro Format	Scala

Sample Code Project	Sample Name	Sample Development Language
SparkOnHbasePythonExample-AvroSource	Spark on HBase-Performing Operations on Data in Avro Format	Python
SparkOnHbaseJavaExample-HbaseSource	Spark on HBase-Performing Operations on the HBase Data Source	Java
SparkOnHbaseScalaExample-HbaseSource	Spark on HBase-Performing Operations on the HBase Data Source	Scala
SparkOnHbasePythonExample-HbaseSource	Spark on HBase-Performing Operations on the HBase Data Source	Python
SparkOnHbaseJavaExample-JavaHBaseBulkPutExample	Spark on HBase-Using the BulkPut Interface	Java
SparkOnHbaseScalaExample-HBaseBulkPutExample	Spark on HBase-Using the BulkPut Interface	Scala
SparkOnHbasePythonExample-HBaseBulkPutExample	Spark on HBase-Using the BulkPut Interface	Python
SparkOnHbaseJavaExample-JavaHBaseBulkGetExample	Spark on HBase-Using the BulkGet Interface	Java
SparkOnHbaseScalaExample-HBaseBulkGetExample	Spark on HBase-Using the BulkGet Interface	Scala
SparkOnHbasePythonExample-HBaseBulkGetExample	Spark on HBase-Using the BulkGet Interface	Python
SparkOnHbaseJavaExample-JavaHBaseBulkDeleteExample	Spark on HBase-Using the BulkDelete Interface	Java
SparkOnHbaseScalaExample-HBaseBulkDeleteExample	Spark on HBase-Using the BulkDelete Interface	Scala
SparkOnHbasePythonExample-HBaseBulkDeleteExample	Spark on HBase-Using the BulkDelete Interface	Python
SparkOnHbaseJavaExample-JavaHBaseBulkLoadExample	Spark on HBase-Using the BulkLoad Interface	Java
SparkOnHbaseScalaExample-HBaseBulkLoadExample	Spark on HBase-Using the BulkLoad Interface	Scala
SparkOnHbasePythonExample-HBaseBulkLoadExample	Spark on HBase-Using the BulkLoad Interface	Python

Sample Code Project	Sample Name	Sample Development Language
SparkOnHbaseJavaExample-JavaHBaseForEachPartitionExample	Spark on HBase-Using the foreachPartition Interface	Java
SparkOnHbaseScalaExample-HBaseForEachPartitionExample	Spark on HBase-Using the foreachPartition Interface	Scala
SparkOnHbasePythonExample-HBaseForEachPartitionExample	Spark on HBase-Using the foreachPartition Interface	Python
SparkOnHbaseJavaExample-JavaHBaseDistributedScanExample	Spark on HBase-Distributedly Scanning HBase Tables	Java
SparkOnHbaseScalaExample-HBaseDistributedScanExample	Spark on HBase-Distributedly Scanning HBase Tables	Scala
SparkOnHbasePythonExample-HBaseDistributedScanExample	Spark on HBase-Distributedly Scanning HBase Tables	Python
SparkOnHbaseJavaExample-JavaHBaseMapPartitionExample	Spark on HBase-Using the mapPartition Interface	Java
SparkOnHbaseScalaExample-HBaseMapPartitionExample	Spark on HBase-Using the mapPartition Interface	Scala
SparkOnHbasePythonExample-HBaseMapPartitionExample	Spark on HBase-Using the mapPartition Interface	Python
SparkOnHbaseJavaExample-JavaHBaseStreamingBulkPutExample	Spark on HBase-Writing Data to HBase Tables In Batches Using SparkStreaming	Java
SparkOnHbaseScalaExample-HBaseStreamingBulkPutExample	Spark on HBase-Writing Data to HBase Tables In Batches Using SparkStreaming	Scala
SparkOnHbasePythonExample-HBaseStreamingBulkPutExample	Spark on HBase-Writing Data to HBase Tables In Batches Using SparkStreaming	Python
SparkHbasetoHbaseJavaExample	Reading Data from HBase and Write It Back to HBase	Java
SparkHbasetoHbaseScalaExample	Reading Data from HBase and Write It Back to HBase	Scala

Sample Code Project	Sample Name	Sample Development Language
SparkHbaseToHbasePythonExample	Reading Data from HBase and Write It Back to HBase	Python
SparkHiveToHbaseJavaExample	Reading Data from Hive and Write It to HBase	Java
SparkHiveToHbaseScalaExample	Reading Data from Hive and Write It to HBase	Scala
SparkHiveToHbasePythonExample	Reading Data from Hive and Write It to HBase	Python
SparkStreamingKafka010JavaExample	Streaming Connecting to Kafka0-10	Java
SparkStreamingKafka010ScalaExample	Streaming Connecting to Kafka0-10	Scala
SparkStructuredStreamingJavaExample	Structured Streaming Project	Java
SparkStructuredStreamingScalaExample	Structured Streaming Project	Scala
SparkStructuredStreamingPythonExample	Structured Streaming Project	Python
StructuredStreamingADScalaExample	Structured Streaming Stream-Stream Join	Scala
StructuredStreamingStateScalaExample	Structured Streaming Status Operation	Scala
SparkOnMultiHbaseScalaExample	Concurrent Access from Spark to HBase in Two Clusters	Scala
SparkOnHudiJavaExample	Using Spark to Perform Basic Hudi Operations	Java
SparkOnHudiPythonExample	Using Spark to Perform Basic Hudi Operations	Python
SparkOnHudiScalaExample	Using Spark to Perform Basic Hudi Operations	Scala

## 30.2.3 Creating a New Project (Optional)

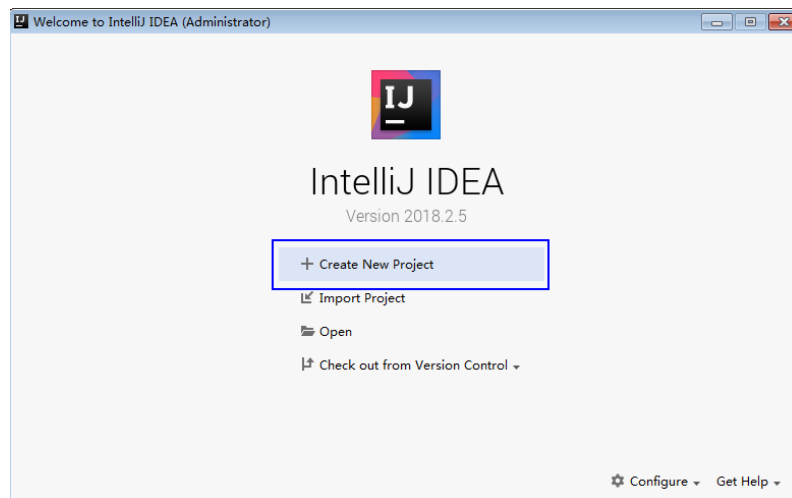
### Scenario

Besides importing Spark sample projects, the IDEA can be used to create a new Spark project. A Scala project is used as an example in the following steps.

### Procedure

**Step 1** Start the IDEA and select **Create New Project**.

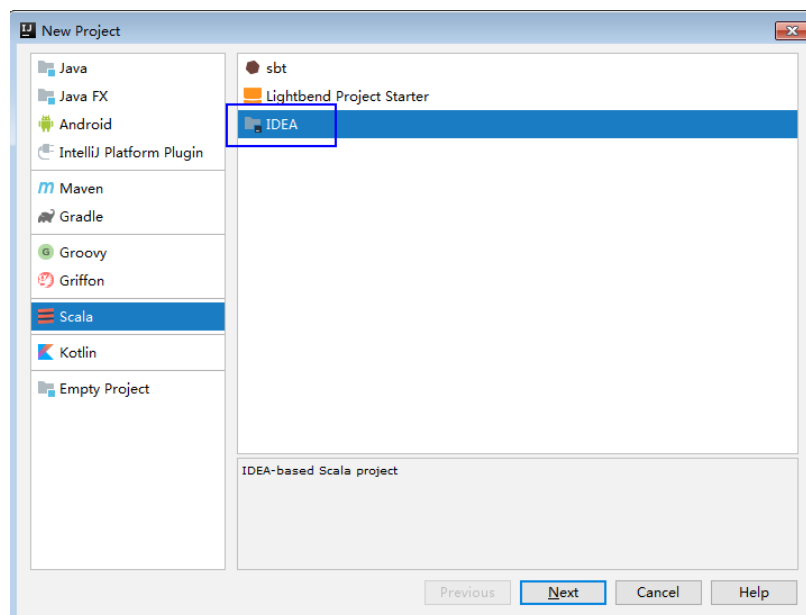
**Figure 30-32** Create a project



**Step 2** On the **New Project** page, select **Scala** as the development environment, select **IDEA**, and click **Next**.

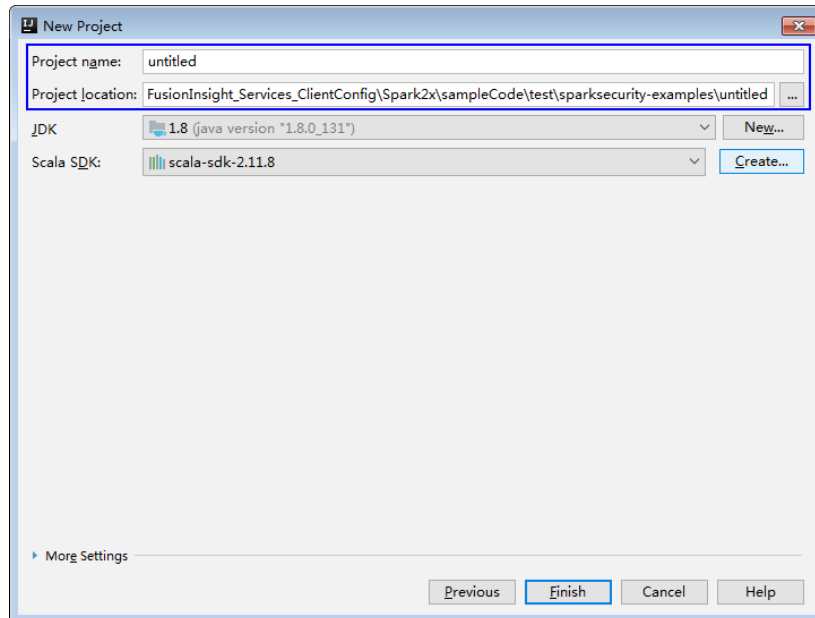
If a new project in Java is required, choose corresponding parameters.

**Figure 30-33** Select the development language



- Step 3** On the project information page, specify **Project name**, **Project location**, **Project JDK**, and **Scala SDK**, and click **Finish** to complete the project creation.

**Figure 30-34** Add the project information



----End

## 30.2.4 Preparing for Security Authentication

### Scenario

In a safe cluster environment, the communication among components cannot be a simple communication. Components must be authorized by each other before the communication to ensure the security of the communication.

When users are developing the Spark application, the Spark is required to interwork with Hadoop and HBase in certain scenarios. Therefore, security authentication codes must be written into the Spark application to ensure that the Spark application can run properly.

Three security authentication modes are available:

- Authentication by running command lines:  
Before submitting the Spark application for running or using the CLI to log in to the Spark SQL, run the following command in the Spark client to obtain authentication:  
**kinit component service user**
- Authentication by configuring parameters  
You can use any of the following methods to specify the security authentication information.
  - Configure the **spark.kerberos.keytab** and **spark.kerberos.principal** parameters in the spark-defaults.conf file on the client.
  - Add the following parameters to the **bin/spark-submit** command:

**--conf spark.kerberos.keytab=<keytab file path> --conf spark.kerberos.principal=<Principal account>**

- Add the following parameter to the **bin/spark-submit** command:  
**--keytab <keytab file path> --principal <Principal account>**
- Authentication by adding codes:  
Authenticate in the application by obtaining principal and keytab files of the client.

**Table 30-7** lists the authentication method for the sample code in security cluster environment.

**Table 30-7** Authentication methods

Sample Code	Mode	Authentication Method
sparknormal-examples	yarn-client	Command authentication, configuration authentication, or code authentication
	yarn-cluster	By running command lines or configuring parameters.
sparksecurity-examples (containing authentication code)	yarn-client	By adding code.
	yarn-cluster	Not supported.

 **NOTE**

- In the yarn cluster mode, the security authentication is not supported in the Spark projects. The security authentication needs to be completed before the application is started.
- For the safety authentication codes of the Python sample project that are not provided, configure the safety authentication parameter in the command that runs the application.

## Safety Security Code (Java)

The safety authentication of the sample codes is completed by invoking the LoginUtil class. For the secure login process, see the chapter Security Authentication Interfaces.

In the Spark sample project code, different sample projects use different authentication codes which are basic safety authentication and the basic safety authentication with the ZooKeeper authentication. The example authentication parameters used in the sample project are displayed as **Table 30-8**. Modify the parameter value as required.

**Table 30-8** Parameter Description

Parameter	Example Parameter Value	Description
userPrincipal	sparkuser	Users use the authenticated account Principal, Use the developer account prepared in <a href="#">Preparing a Developer Account</a> .
userKeytabPath	/opt/FIclient/ user.keytab	Users use the authenticated Keytab file, Copy the <b>user.keytab</b> file of the prepared developer account to the directory indicated by the example parameter value.
ZKServerPrincipal	zookeeper/ hadoop.<system domain name>	The principal of the server in ZooKeeper. Contact the administrator to obtain the account.

The following code snippet belongs to the main method of the **FemaleInfoCollection** class in the **com.huawei.bigdata.spark.examples** package.

- Basic safety authentication:

Spark Core and Spark SQL programs needs only the basic safety authentication codes because both of them do not need to access the HBase or ZooKeeper. Add the following codes in the program, and configure the safety authentication parameter as required.

```
String userPrincipal = "sparkuser";
String userKeytabPath = "/opt/FIclient/user.keytab";
String krb5ConfPath = "/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf";
Configuration hadoopConf = new Configuration();
LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf);
```

- Basic safety authentication with ZooKeeper Authentication:

Because the sample programs "Spark Streaming", "access Spark SQL with JDBC", and "Spark on HBase" require not only the basic safety authentication, but also the Principal of the ZooKeeper server to complete the safety authentication. Add the following codes in the program, and configure the safety authentication parameter as required.

```
String userPrincipal = "sparkuser";
String userKeytabPath = "/opt/FIclient/user.keytab";
String krb5ConfPath = "/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf";
String ZKServerPrincipal = "zookeeper/hadoop.< System domain name >";

String ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME = "Client";
String ZOOKEEPER_SERVER_PRINCIPAL_KEY = "zookeeper.server.principal";

Configuration hadoopConf = new Configuration();
LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, userPrincipal, userKeytabPath);
LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY, ZKServerPrincipal);
LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf);
```



## Safety Authentication Codes for the Communication of Spark and Zookeeper (Scala)

Currently the safety authentication of the sample codes is completed by invoking the LoginUtil class. For the secure login process, see the chapter about the unified authentication.

In the Spark sample project code, different sample projects use different authentication codes, which are basic safety authentication and basic safety authentication with ZooKeeper authentication. The example authentication parameters used in the sample project are displayed as [Table 30-9](#). Modify the parameter value as required.

**Table 30-9** Parameter Description

Parameter	Example Parameter Value	Description
userPrincipal	sparkuser	Users use the authenticated account Principal, Use the developer account prepared in <a href="#">Preparing a Developer Account</a> .
userKeytabPath	/opt/FIclient/ user.keytab	Users use the authenticated Keytab file, Copy the <b>user.keytab</b> file of the prepared developer account to the directory indicated by the example parameter value.
ZKServerPrincipal	zookeeper/ hadoop.<System domain name>	The principal of the server in ZooKeeper. Contact the administrator to obtain the account.

- Basic safety authentication:

Spark Core and Spark SQL programs needs only the basic safety authentication codes because both of them do not need to access the HBase or ZooKeeper. Add the following codes in the program, and configure the safety authentication parameter as required.

```
val userPrincipal = "sparkuser"
val userKeytabPath = "/opt/FIclient/user.keytab"
val krb5ConfPath = "/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf"
val hadoopConf: Configuration = new Configuration()
LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf);
```
- Basic safety authentication with ZooKeeper Authentication:

Because the sample programs "Spark Streaming", "access Spark SQL with JDBC", and "Spark on HBase" require not only the basic safety authentication, but also the Principal of the ZooKeeper server to complete the safety authentication. Add the following codes in the program, and configure the safety authentication parameter as required.

```
val userPrincipal = "sparkuser"
val userKeytabPath = "/opt/FIclient/user.keytab"
val krb5ConfPath = "/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf"
val ZKServerPrincipal = "zookeeper/hadoop.<system domain name>"
```

```
val ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME: String = "Client"
val ZOOKEEPER_SERVER_PRINCIPAL_KEY: String = "zookeeper.server.principal"
val hadoopConf: Configuration = new Configuration();
LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, userPrincipal, userKeytabPath)
LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY, ZKServerPrincipal)
LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf);
```

## 30.2.5 Configuring the Python3 Sample Project

### Scenario

To run the Python3 interface sample code of the Spark2x component of MRS, perform the following operations.

### Procedure

**Step 1** Install Python3 of 3.6 or a higher version on the client.

The Python version can be viewed by running the **python3** command on the CLI of the client. The following information indicates that the Python version is 3.8.2.

```
Python 3.8.2 (default, Jun 23 2020, 10:26:03)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

**Step 2** Setuptools 47.3.1 must be installed on the client.

To obtain the software, visit the official websites.

<https://pypi.org/project/setuptools/#files>

Copy the downloaded setuptools package to the client, decompress the package, go to the decompressed directory, and run the **python3 setup.py install** command in the CLI of the client.

The following information indicates that setuptools 47.3.1 is installed successfully.

```
Finished processing dependencies for setuptools==47.3.1
```

**Step 3** Install Python on the client.

1. Obtain the sample project folder **python3-examples** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python3-examples** folder.
3. Go to the **dependency\_python3.6**, **dependency\_python3.7**, or **dependency\_python3.8** folder based on the Python3 version.
4. Run the **whereis easy\_install** command to find the path of the **easy\_install** program. If there are multiple paths, run the **easy\_install --version** command to select the **easy\_install** path corresponding to the **setuptools** version, for example, **/usr/local/bin/easy\_install**.
5. Run the **easy\_install** command to install the EGG files in the **dependency\_python3.x** folder in sequence. Example:

```
/usr/local/bin/easy_install future-0.18.2-py3.8.egg
```

If the following information is displayed, the EGG file is successfully installed.

```
Finished processing dependencies for future==0.18.2
```

----End

## 30.3 Developing the Project

### 30.3.1 Spark Core Project

#### 30.3.1.1 Overview

##### Scenarios

Develop a Spark application to perform the following operations on logs about dwell durations of netizens for shopping online:

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.
- The first column in the log file records names, the second column records gender, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

log1.txt: logs collected on Saturday

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

log2.txt: logs collected on Sunday

```
LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
FangBo,female,50
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
CaiXuyu,female,50
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
FangBo,female,50
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

##### Data Preparation

Save log files in the Hadoop distributed file system (HDFS).

1. Create text files **input\_data1.txt** and **input\_data2.txt** on a local computer, and copy **log1.txt** to **input\_data1.txt** and **log2.txt** to **input\_data2.txt**.
2. Create **/tmp/input** on HDFS client path, and run the following commands to upload **input\_data1.txt** and **input\_data2.txt** to **/tmp/input**:

- a. On the HDFS client, run the following commands to obtain the safety authentication:  

```
cd /opt/hadoopclient
source bigdata_env
kinit <Service user for authentication>
```
- b. On the Linux HDFS client, run the **hadoop fs -mkdir /tmp/input** command (a `hdfs dfs` command provides the same function.) to create a directory.
- c. Go to the **/tmp/input** directory on the HDFS client, on the Linux HDFS client, run the **hadoop fs -put input\_data1.txt /tmp/input** and **hadoop fs -put input\_data2.txt /tmp/input** commands to upload data files.

## Development Idea

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

- Read the source file data.
- Filter the data information of the time that female netizens spend online.
- Aggregate the total time that each female netizen spends online.
- Filter the information of female netizens who spend more than 2 hours online.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

1. Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.
2. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

### NOTE

- Before compilation and packaging, change the paths of the **user.keytab** and **krb5.conf** files in the sample code to the actual paths on the client server where the files are located. Example: **/opt/female/user.keytab** and **/opt/female/krb5.conf**.
  - To run the Python sample code, you do not need to use Maven for packaging. You only need to upload the **user.keytab** and **krb5.conf** files to the server where the client is located.
3. Upload the JAR file to any directory (for example, **/opt/female/**) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the **bin/spark-submit** script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

- Run the Scala and Java sample programs.

```
bin/spark-submit --class
com.huawei.bigdata.spark.examples.FemaleInfoCollection --master yarn --
deploy-mode client /opt/female/FemaleInfoCollection-1.0.jar <inputPath>
<inputPath> indicates the input path in HDFS.
```

- Run the Python sample program.

### NOTE

- The Python sample code does not provide authentication information. Configure **--keytab** and **--principal** to specify authentication information.

```
bin/spark-submit --master yarn --deploy-mode client --keytab /opt/
Flclient/user.keytab --principal sparkuser /opt/female/
SparkPythonExample/collectFemaleInfo.py <inputPath>
<inputPath> indicates the input path in HDFS.
```

### 30.3.1.2 Java Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

#### Sample Code

The following code segment is only an example. For details, see the `com.huawei.bigdata.spark.examples.FemaleInfoCollection` class.

```
// Create a configuration class SparkConf, and then create a SparkContext.
SparkSession spark = SparkSession
.builder()
.appName("CollectFemaleInfo")
.config("spark.some.config.option", "some-value")
.getOrCreate();

// Read the source file data, and transfer each row of records to an element of the RDD.
JavaRDD<String> data = spark.read()
.textFile(args[0])
.javaRDD();

// Split each column of each record, and generate a Tuple.
JavaRDD<Tuple3<String,String,Integer>> person = data.map(new
Function<String,Tuple3<String,String,Integer>>()
{
 private static final long serialVersionUID = -2381522520231963249L;
 public Tuple3<String, String, Integer> call(String s) throws Exception
 {
 // Split a row of data by commas (,).
 String[] tokens = s.split(",");

 // Integrate the three split elements to a ternary Tuple.
 Tuple3<String, String, Integer> person = new Tuple3<String, String, Integer>(tokens[0], tokens[1],
Integer.parseInt(tokens[2]));
```

```
 return person;
 }
 });

 // Use the filter function to filter the data information about the time that female netizens spend online.
 JavaRDD<Tuple3<String,String,Integer>> female = person.filter(new
Function<Tuple3<String,String,Integer>, Boolean>()
 {
 private static final long serialVersionUID = -4210609503909770492L;

 public Boolean call(Tuple3<String, String, Integer> person) throws Exception
 {
 // Filter the records of which the gender in the second column is female.
 Boolean isFemale = person._2().equals("female");
 return isFemale;
 }
 });

 // Aggregate the total time that each female netizen spends online.
 JavaPairRDD<String, Integer> females = female.mapToPair(new PairFunction<Tuple3<String, String,
Integer>, String, Integer>()
 {
 private static final long serialVersionUID = 8313245377656164868L;

 public Tuple2<String, Integer> call(Tuple3<String, String, Integer> female) throws Exception
 {
 // Extract the two columns representing the name and online time for the sum of online time by
name during further operations.
 Tuple2<String, Integer> femaleAndTime = new Tuple2<String, Integer>(female._1(), female._3());
 return femaleAndTime;
 }
 });
 JavaPairRDD<String, Integer> femaleTime = females.reduceByKey(new Function2<Integer, Integer,
Integer>()
 {
 private static final long serialVersionUID = -3271456048413349559L;

 public Integer call(Integer integer, Integer integer2) throws Exception
 {
 // Sum two online time durations of the same female netizen.
 return (integer + integer2);
 }
 });

 // Filter the information about female netizens who spend more than 2 hours online.
 JavaPairRDD<String, Integer> rightFemales = females.filter(new Function<Tuple2<String, Integer>,
Boolean>()
 {
 private static final long serialVersionUID = -3178168214712105171L;

 public Boolean call(Tuple2<String, Integer> s) throws Exception
 {
 // Extract the total time that female netizens spend online, and determine whether the time is
more than 2 hours.
 if(s._2() > (2 * 60))
 {
 return true;
 }
 return false;
 }
 });

 // Print the information about female netizens who meet the requirements.
 for(Tuple2<String, Integer> d: rightFemales.collect())
 {
 System.out.println(d._1() + "," + d._2());
 }
}
```

### 30.3.1.3 Scala Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

#### Sample Code

The following code segment is only an example. For details, see the `com.huawei.bigdata.spark.examples.FemaleInfoCollection` class.

Example: `CollectMapper` class

```
val spark = SparkSession
 .builder()
 .appName("CollectFemaleInfo")
 .config("spark.some.config.option", "some-value")
 .getOrCreate()

// Read data. This code indicates the data path that the input parameter args(0) specifies.
val text = spark.sparkContext.textFile(args(0))
// Filter the data information about the time that female netizens spend online.
val data = text.filter(_.contains("female"))
// Aggregate the time that each female netizen spends online.
val femaleData:RDD[(String,Int)] = data.map{line =>
 val t= line.split(',')
 (t(0),t(2).toInt)
}.reduceByKey(_ + _)
// Filter the information about female netizens who spend more than 2 hours online, and export the results.
val result = femaleData.filter(line => line._2 > 120)
result.collect().map(x => x._1 + ',' + x._2).foreach(println)
spark.stop()
```

### 30.3.1.4 Python Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

#### Sample Code

The following code segment is only an example. For details, see `collectFemaleInfo.py`.

```
def contains(str, substr):
 if substr in str:
 return True
 return False

if __name__ == "__main__":
 if len(sys.argv) < 2:
 print "Usage: CollectFemaleInfo <file>"
 exit(-1)

 spark = SparkSession \
 .builder \
 .appName("CollectFemaleInfo") \
 .getOrCreate()


```

The following programs are used to implement the following functions:

1. Read data. This code indicates the data path that the input parameter argv[1] specifies. - text
2. Filter data about the time that female netizens spend online. - filter
3. Aggregate the total time that each female netizen spends online. - map/map/reduceByKey
4. Filter information about female netizens who spend more than 2 hours online. - filter

```

inputPath = sys.argv[1]
result = spark.read.text(inputPath).rdd.map(lambda r: r[0])\
 .filter(lambda line: contains(line, "female")) \
 .map(lambda line: line.split(',')) \
 .map(lambda dataArr: (dataArr[0], int(dataArr[2]))) \
 .reduceByKey(lambda v1, v2: v1 + v2) \
 .filter(lambda tupleVal: tupleVal[1] > 120) \
 .collect()
for (k, v) in result:
 print k + "," + str(v)

Stop SparkContext.
spark.stop()

```

## 30.3.2 Spark SQL Project

### 30.3.2.1 Overview

#### Scenario

Develop a Spark application to perform the following operations on logs about dwell durations of netizens for shopping online:

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.
- The first column in the log file records names, the second column records gender, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

log1.txt: logs collected on Saturday

```

LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60

```

log2.txt: logs collected on Sunday

```

LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
FangBo,female,50
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
CaiXuyu,female,50
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
FangBo,female,50
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60

```



## Data Preparation

Save log files in the Hadoop distributed file system (HDFS).

1. Create text files **input\_data1.txt** and **input\_data2.txt** on a local computer, and copy **log1.txt** to **input\_data1.txt** and **log2.txt** to **input\_data2.txt**.
2. Create **/tmp/input** on the HDFS client path, and run the following commands to upload **input\_data1.txt** and **input\_data2.txt** to **/tmp/input**:
  - a. On the HDFS client, run the following commands to obtain the safety authentication:

```
cd /opt/hadoopclient
source bigdata_env
kinit <component service user>
```
  - b. On the Linux HDFS client, run the **hadoop fs -mkdir /tmp/input** command (a **hdfs dfs** command provides the same function.) to create a directory.
  - c. Go to the **/tmp/input** directory on the HDFS client, on the Linux HDFS client, run the **hadoop fs -put input\_data1.txt /tmp/input** and **hadoop fs -put input\_data2.txt /tmp/input** commands to upload data files.

## Development Idea

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

- Create a table and import the log files into the table.
- Filter the data information of the time that female netizens spend online.
- Aggregate the total time that each female netizen spends online.
- Filter the information of female netizens who spend more than 2 hours online.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

1. Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.
2. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

 NOTE

- Before compilation and packaging, change the paths of the `user.keytab` and `krb5.conf` files in the sample code to the actual paths on the client server where the files are located. Example: `/opt/female/user.keytab` and `/opt/female/krb5.conf`.
  - To run the Python sample code, you do not need to use Maven for packaging. You only need to upload the `user.keytab` and `krb5.conf` files to the server where the client is located.
3. Upload the JAR file to any directory (for example, `/opt/female/`) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the `bin/spark-submit` script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

- Run the Scala and Java sample programs.  
**bin/spark-submit --class**  
*com.huawei.bigdata.spark.examples.FemaleInfoCollection* **--master yarn --**  
**deploy-mode client** */opt/female/SparkSqlScalaExample-1.0.jar* *<inputPath>*  
*<inputPath>* indicates the input path in HDFS.
- Run the Python sample program

 NOTE

- The Python sample code does not provide authentication information. Configure `--keytab` and `--principal` to specify authentication information.  
**bin/spark-submit --master yarn --deploy-mode client --keytab /opt/FIClient/user.keytab --principal sparkuser /opt/female/SparkPythonExample/SparkSQLPythonExample.py** *<inputPath>*  
*<inputPath>* indicates the input path in HDFS.

### 30.3.2.2 Java Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

#### Sample Code

The following code segment is only an example. For details, see `com.huawei.bigdata.spark.examples.FemaleInfoCollection`.

```
public static void main(String[] args) throws Exception {
 SparkSession spark = SparkSession
 .builder()
 .appName("CollectFemaleInfo")
 .config("spark.some.config.option", "some-value")
 .getOrCreate();

 // Convert RDD to DataFrame through the implicit conversion.
 JavaRDD<FemaleInfo> femaleInfoJavaRDD = spark.read().textFile(args[0]).javaRDD().map(
```

```
new Function<String, FemaleInfo>() {
 @Override
 public FemaleInfo call(String line) throws Exception {
 String[] parts = line.split(",");
 FemaleInfo femaleInfo = new FemaleInfo();
 femaleInfo.setName(parts[0]);
 femaleInfo.setGender(parts[1]);
 femaleInfo.setStayTime(Integer.parseInt(parts[2].trim()));
 return femaleInfo;
 }
};

// Register table.
Dataset<ROW> schemaFemaleInfo = spark.createDataFrame(femaleInfoJavaRDD, FemaleInfo.class);
schemaFemaleInfo.registerTempTable("FemaleInfoTable");

// Run SQL query
Dataset<ROW> femaleTimeInfo = spark.sql("select * from " +
 "(select name,sum(stayTime) as totalStayTime from FemaleInfoTable " +
 "where gender = 'female' group by name)" +
 " tmp where totalStayTime >120");

// Collect the columns of a row in the result.
List<String> result = femaleTimeInfo.javaRDD().map(new Function<Row, String>() {
 public String call(Row row) {
 return row.getString(0) + "," + row.getLong(1);
 }
}).collect();
System.out.println(result);
spark.stop();
}
```

In the preceding code example, data processing logic is implemented by SQL statements. It can also be implemented by invoking the SparkSQL interface in Scala/Java/Python code. For details, see <http://spark.apache.org/docs/3.1.1/sql-programming-guide.html#running-sql-queries-programmatically>

### 30.3.2.3 Scala Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

#### Sample Code

The following code segment is only an example. For details, see `com.huawei.bigdata.spark.examples.FemaleInfoCollection`.

```
object FemaleInfoCollection
{
 //Table structure, used for mapping the text data to df
 case class FemaleInfo(name: String, gender: String, stayTime: Int)
 def main(args: Array[String]) {
 //Configure Spark application name
 val spark = SparkSession
 .builder()
 .appName("FemaleInfo")
 .config("spark.some.config.option", "some-value")
 .getOrCreate()
 import spark.implicits._
 //Convert RDD to DataFrame through the implicit conversion, then register table.
 spark.sparkContext.textFile(args(0)).map(_.split(","))
 .map(p => FemaleInfo(p(0), p(1), p(2).trim.toInt))
 .toDF.registerTempTable("FemaleInfoTable")
 }
}
```

```
//Via SQL statements to screen out the time information of female stay on the Internet , and aggregated the same names.
val femaleTimeInfo = spark.sql("select name,sum(stayTime) as stayTime from FemaleInfoTable where gender = 'female' group by name")
//Filter information about female netizens who spend more than 2 hours online.
val c = femaleTimeInfo.filter("stayTime >= 120").collect().foreach(println)
spark.stop()
}
}
```

In the preceding code example, data processing logic is implemented by SQL statements. It can also be implemented by invoking the SparkSQL interface in Scala/Java/Python code. For details, see <http://spark.apache.org/docs/3.1.1/sql-programming-guide.html#running-sql-queries-programmatically>

### 30.3.2.4 Python Sample Code

#### Function

Collect information about female netizens who have spent more than 2 hours in online shopping on the weekend.

#### Sample Code

The following code segment is only an example. For details, see `SparkSQLPythonExample`.

```
-*- coding:utf-8 -*-

import sys
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext

def contains(str1, substr1):
 if substr1 in str1:
 return True
 return False

if __name__ == "__main__":
 if len(sys.argv) < 2:
 print "Usage: SparkSQLPythonExample.py <file>"
 exit(-1)

 # Initialize the SparkSession and SQLContext.
 sc = SparkSession.builder.appName("CollectFemaleInfo").getOrCreate()
 sqlCtx = SQLContext(sc)

 #Convert RDD to DataFrame.
 inputPath = sys.argv[1]
 inputRDD = sc.read.text(inputPath).rdd.map(lambda r: r[0])\
 .map(lambda line: line.split(",")\
 .map(lambda dataArr: (dataArr[0], dataArr[1], int(dataArr[2]))))\
 .collect()
 df = sqlCtx.createDataFrame(inputRDD)

 # Register a table.
 df.registerTempTable("FemaleInfoTable")

 # Run SQL query statements and display the result.
 FemaleTimeInfo = sqlCtx.sql("SELECT * FROM " +
 "(SELECT _1 AS Name,SUM(_3) AS totalStayTime FROM FemaleInfoTable " +
 "WHERE _2 = 'female' GROUP BY _1)" +
 " WHERE totalStayTime >120").show()
```

```
sc.stop()
```

## 30.3.3 Accessing the Spark SQL Through JDBC

### 30.3.3.1 Overview

#### Scenarios

Users customize JDBCServer clients and use JDBC connections to create, load data to, query, and delete data tables.

#### Data Preparation

**Step 1** Ensure that the JDBCServer service has been started in multi-active instance HA mode and at least one instance provides connections for client. Create the **/home/data** file on every available instance nodes of the JDBCServer.

```
Miranda,32
Karlie,23
Candice,27
```

**Step 2** Ensure that the user whose starts the JDBCServer has the read and write permission on the file.

**Step 3** Ensure that the **hive-site.xml** file exists in **classpath**, and set parameters required for the client connection. For details about parameters required for the JDBCServer, see [JDBCServer Interface](#).

----End

#### Development Idea

1. Create a child table in the default database.
2. Add data in **/home/data** to the child table.
3. Query data in the child table.
4. Delete the child table.

#### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on FusionInsight Manager. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

#### Packaging the Project

- Upload the **krb5.conf** and **user.keytab** files to the server where the client is located.
- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

 NOTE

Before compilation and packaging, change the paths of the user.keytab and krb5.conf files in the sample code to the actual paths on the client server where the files are located. Example: `/opt/female/user.keytab` and `/opt/female/krb5.conf`.

- Upload the JAR file to any directory (for example, `/opt/female/`) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the `java -cp` command to run the code (the class name and file name must be the same as those in the actual code. The following is only an example).

- Run the Java sample code:

```
java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/hive/*:$SPARK_HOME/conf:/opt/female/SparkThriftServerJavaExample-1.0.jar com.huawei.bigdata.spark.examples.ThriftServerQueriesTest $SPARK_HOME/conf/hive-site.xml $SPARK_HOME/conf/spark-defaults.conf
```

- Run the Scala sample code:

```
java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/hive/*:$SPARK_HOME/conf:/opt/female/SparkThriftServerExample-1.0.jar com.huawei.bigdata.spark.examples.ThriftServerQueriesTest $SPARK_HOME/conf/hive-site.xml $SPARK_HOME/conf/spark-defaults.conf
```

 NOTE

After the SSL feature of ZooKeeper is enabled for the cluster (check the `ssl.enabled` parameter of the ZooKeeper service), add the `-Dzookeeper.client.secure=true -Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty` parameter to the command:

```
java -Dzookeeper.client.secure=true -Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/hive/*:$SPARK_HOME/conf:/opt/female/SparkThriftServerJavaExample-1.0.jar com.huawei.bigdata.spark.examples.ThriftServerQueriesTest $SPARK_HOME/conf/hive-site.xml $SPARK_HOME/conf/spark-defaults.conf
```

### 30.3.3.2 Java Sample Code

#### Function

The JDBC interface of the user-defined client is used to submit the data analysis task and return the results.

#### Sample Code

- Step 1** Define an SQL statement. The SQL statement must be a single statement that cannot contain the semicolon (;). For example,

```
ArrayList<String> sqlList = new ArrayList<String>();
sqlList.add("CREATE TABLE CHILD (NAME STRING, AGE INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','");
sqlList.add("LOAD DATA LOCAL INPATH '/home/data' INTO TABLE CHILD");
sqlList.add("SELECT * FROM child");
sqlList.add("DROP TABLE child");
executeSql(url, sqlList);
```

 NOTE

The data file in the sample project must be placed into the Home directory of the host where the JDBCServer is located.

**Step 2** Assemble the JDBC URL.

```
String securityConfig = ";sasLQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.<system domain name>@<system domain name>;user.principal=sparkuser;user.keytab=/opt/Flclient/user.keytab;";
Configuration config = new Configuration();
config.addResource(new Path(args[0]));
String zkUrl = config.get("spark.deploy.zookeeper.url");

String zkNamespace = null;
zkNamespace = fileInfo.getProperty("spark.thriftserver.zookeeper.namespace");
if (zkNamespace != null) {
 //Remove redundant characters from configuration items
 zkNamespace = zkNamespace.substring(1);
}

StringBuilder sb = new StringBuilder("jdbc:hive2://"
 + zkUrl
 + ";serviceDiscoveryMode=zooKeeper;zooKeeperNamespace="
 + zkNamespace
 + securityConfig);
String url = sb.toString();
```

 NOTE

The default validity period of KERBEROS authentication is one day. After the validity period, the authentication needs to be performed again if you want to connect the client and JDBCServer. You can add the user.principal and user.keytab authentication information to **url** to ensure that the authentication is performed each time the connection is established. For example, add **user.principal=sparkuser;user.keytab=/opt/client/user.keytab** to **url**.

**Step 3** Load the Hive JDBC driver.

```
Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance();
```

**Step 4** Obtain the JDBC connection, execute the HQL, export the obtained column name and results to the console, and close the JDBC connection.

The **zk.quorum** in the connection string can be replaced by **spark.deploy.zookeeper.url** in the configuration file.

In network congestion, configure the timeout of the connection between the client and JDBCServer to avoid the suspending of the client due to timeless wait of the return from the server. The method is as follows:

Before executing the DriverManager.getConnection script to obtain the JDBC connection, add the DriverManager.setLoginTimeout(n) script to configure the timeout. n indicates the timeout length of waiting for the return from the server. The unit is second, the type is Int, and the default value is 0 (indicating never timing out).

```
static void executeSql(String url, ArrayList<String> sqls) throws ClassNotFoundException, SQLException {
 try {
 Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance();
 } catch (Exception e) {
 e.printStackTrace();
 }
 Connection connection = null;
 PreparedStatement statement = null;

 try {
 connection = DriverManager.getConnection(url);
```

```

for (int i =0 ; i < sqls.size(); i++) {
 String sql = sqls.get(i);
 System.out.println("---- Begin executing sql: " + sql + " ----");
 statement = connection.prepareStatement(sql);
 ResultSet result = statement.executeQuery();
 ResultSetMetaData resultMetaData = result.getMetaData();
 Integer colNum = resultMetaData.getColumnCount();
 for (int j =1; j <= colNum; j++) {
 System.out.println(resultMetaData.getColumnLabel(j) + "\t");
 }
 System.out.println();

 while (result.next()) {
 for (int j =1; j <= colNum; j++){
 System.out.println(result.getString(j) + "\t");
 }
 System.out.println();
 }
 System.out.println("---- Done executing sql: " + sql + " ----");
}

} catch (Exception e) {
 e.printStackTrace();
} finally {
 if (null != statement) {
 statement.close();
 }
 if (null != connection) {
 connection.close();
 }
}
}
}

```

----End

### 30.3.3.3 Scala Sample Code

#### Function

The JDBC interface of the user-defined client is used to submit the data analysis task and return the results.

#### Sample Code

- Step 1** Define an SQL statement. The SQL statement must be a single statement that cannot contain the semicolon (;). For example,

```

val sqlList = new ArrayBuffer[String]
sqlList += "CREATE TABLE CHILD (NAME STRING, AGE INT) " +
"ROW FORMAT DELIMITED FIELDS TERMINATED BY ','"
sqlList += "LOAD DATA LOCAL INPATH '/home/data' INTO TABLE CHILD"
sqlList += "SELECT * FROM child"
sqlList += "DROP TABLE child"

```

#### NOTE

The data file in the sample project must be placed into the Home directory of the host where the JDBCServer is located.

- Step 2** Assemble the JDBC URL.

```

val securityConfig = ";sasqop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.<system domain name>@<system domain name>" + ";"
val config: Configuration = new Configuration()
config.addResource(new Path(args(0)))

```



```
val zkUrl = config.get("spark.deploy.zookeeper.url")

var zkNamespace: String = null
zkNamespace = fileInfo.getProperty("spark.thriftserver.zookeeper.namespace")
//Remove redundant characters from configuration items
if (zkNamespace != null) zkNamespace = zkNamespace.substring(1)
val sb = new StringBuilder("jdbc:hive2://"
+ zkUrl
+ ";serviceDiscoveryMode=zooKeeper;zooKeeperNamespace="
+ zkNamespace
+ securityConfig)
val url = sb.toString()
```

#### NOTE

The default validity period of KERBEROS authentication is one day. After the validity period, the authentication needs to be performed again if you want to connect the client and JDBCServer. You can add the `user.principal` and `user.keytab` authentication information to `url` to ensure that the authentication is performed each time the connection is established. For example, add `user.principal=sparkuser;user.keytab=/opt/client/user.keytab` to `url`.

- Step 3** Load the Hive JDBC driver. Obtain the JDBC connection, execute the HQL, export the obtained column name and results to the console, and close the JDBC connection.

The `zk.quorum` in the connection string can be replaced by `spark.deploy.zookeeper.url` in the configuration file.

In network congestion, configure the timeout of the connection between the client and JDBCServer to avoid the suspending of the client due to timeless wait of the return from the server. The method is as follows:

Before executing the `DriverManager.getConnection` script to obtain the JDBC connection, add the `DriverManager.setLoginTimeout(n)` script to configure the timeout. `n` indicates the timeout length of waiting for the return from the server. The unit is second, the type is `Int`, and the default value is 0 (indicating never timing out).

```
def executeSql(url: String, sqls: Array[String]): Unit = {
//Load the Hive JDBC driver.
Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance()

var connection: Connection = null
var statement: PreparedStatement = null
try {
connection = DriverManager.getConnection(url)
for (sql <- sqls) {
println(s"---- Begin executing sql: $sql ----")
statement = connection.prepareStatement(sql)

val result = statement.executeQuery()

val resultMetaData = result.getMetaData
val colNum = resultMetaData.getColumnCount
for (i <- 1 to colNum) {
print(resultMetaData.getColumnLabel(i) + "\t")
}
println()

while (result.next()) {
for (i <- 1 to colNum) {
print(result.getString(i) + "\t")
}
println()
}
println(s"---- Done executing sql: $sql ----")
}
```

```
}
} finally {
 if (null != statement) {
 statement.close()
 }

 if (null != connection) {
 connection.close()
 }
}
}
```

----End

## 30.3.4 Spark on HBase

### 30.3.4.1 Performing Operations on Data in Avro Format

#### Scenario

Users can use HBase as data sources in Spark applications. In this example, data is stored in HBase in Avro format. Data is read from the HBase, and the read data is filtered.

#### Data Planning

On the client, run the **hbase shell** command to go to the HBase command line and run the following commands to create HBase tables to be used in the sample code:

```
create 'ExampleAvrotable','rowkey','cf1'
```

```
create 'ExampleAvrotableInsert','rowkey','cf1'
```

#### Development Guideline

1. Create an RDD.
2. Perform operations on HBase to treat it as the data source and write the generated RDD into HBase tables.
3. Read data from HBase tables and performs simple operations on the data.

#### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **super**, change the value to the prepared development user name.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.

- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

 NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name of the case code is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. Run the following commands in the **\$SPARK\_HOME** directory.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.AvroSource SparkOnHbaseJavaExample.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample.jar,/opt/female/protobuf-java-2.5.0.jar AvroSource.py
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.AvroSource --files /opt/user.keytab,/opt/krb5.conf SparkOnHbaseJavaExample.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --files /opt/user.keytab,/opt/krb5.conf --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample.jar,/opt/female/protobuf-java-2.5.0.jar AvroSource.py
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the AvroSource file in SparkOnHbaseJavaExample.

```
public static void main(JavaSparkContext jsc) throws IOException {
 LoginUtil.loginWithUserKeytab();
 SQLContext sqlContext = new SQLContext(jsc);
 Configuration hbaseconf = new HBaseConfiguration().create();
 JavaHBaseContext hBaseContext = new JavaHBaseContext(jsc, hbaseconf);
 List list = new ArrayList<AvroHBaseRecord>();
 for(int i=0; i<=255; ++i){
 list.add(AvroHBaseRecord.apply(i));
 }
 try{
 Map<String, String> map = new HashMap<String, String>();
 map.put(HBaseTableCatalog.tableCatalog(), catalog);
 map.put(HBaseTableCatalog.newTable(), "5");
 sqlContext.createDataFrame(list,
AvroHBaseRecord.class).write().options(map).format("org.apache.hadoop.hbase.spark").save();
 Dataset<Row> ds = withCatalog(sqlContext,catalog);
 ds.show();
 ds.printSchema();
 ds.registerTempTable("ExampleAvrotable");
 Dataset<Row> c= sqlContext.sql("select count(1) from ExampleAvrotable");
 c.show();
 Dataset<Row> filtered = ds.select("col0", "col1.favorite_array").where("col0 = 'name1'");
 filtered.show();
 java.util.List<Row> collected = filtered.collectAsList();
 if (collected.get(0).get(1).toString().equals("number1")) {
 throw new UserCustomizedSampleException("value invalid", new Throwable());
 }
 if (collected.get(0).get(1).toString().equals("number2")) {
 throw new UserCustomizedSampleException("value invalid", new Throwable());
 }
 Map avroCatalogInsertMap = new HashMap<String,String>();
 avroCatalogInsertMap.put("avroSchema" , AvroHBaseRecord.schemaString);
 avroCatalogInsertMap.put(HBaseTableCatalog.tableCatalog(), avroCatalogInsert);
 ds.write().options(avroCatalogInsertMap).format("org.apache.hadoop.hbase.spark").save();
 Dataset<Row> newDS = withCatalog(sqlContext,avroCatalogInsert);
 newDS.show();
 newDS.printSchema();
 if (newDS.count() != 256) {
 throw new UserCustomizedSampleException("value invalid", new Throwable());
 }
 ds.filter("col1.name = 'name5' || col1.name <= 'name5'").select("col0","col1.favorite_color",
"col1.favorite_number").show();
 } finally{
 jsc.stop();
 }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the AvroSource file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
 LoginUtil.loginWithUserKeytab()
 val sparkConf = new SparkConf().setAppName("AvroSourceExample")
 val sc = new SparkContext(sparkConf)
 val sqlContext = new SQLContext(sc)
 val hbaseConf = HBaseConfiguration.create()
 val hBaseContext = new HBaseContext(sc, hbaseConf)
 import sqlContext.implicits._
 def withCatalog(cat: String): DataFrame = {
 sqlContext
 .read
```

```
.options(Map("avroSchema" -> AvroHBaseRecord.schemaString, HBaseTableCatalog.tableCatalog ->
avroCatalog))
 .format("org.apache.hadoop.hbase.spark")
 .load()
}
val data = (0 to 255).map { i =>
 AvroHBaseRecord(i)
}
try {
 sc.parallelize(data).toDF.write.options(
 Map(HBaseTableCatalog.tableCatalog -> catalog, HBaseTableCatalog.newTable -> "5"))
 .format("org.apache.hadoop.hbase.spark")
 .save()

 val df = withCatalog(catalog)
 df.show()
 df.printSchema()
 df.registerTempTable("ExampleAvrotable")
 val c = sqlContext.sql("select count(1) from ExampleAvrotable")
 c.show()

 val filtered = df.select($"col0", $"col1.favorite_array").where($"col0" === "name001")
 filtered.show()
 val collected = filtered.collect()
 if (collected(0).getSeq[String](1)(0) != "number1") {
 throw new UserCustomizedSampleException("value invalid")
 }
 if (collected(0).getSeq[String](1)(1) != "number2") {
 throw new UserCustomizedSampleException("value invalid")
 }

 df.write.options(
 Map("avroSchema" -> AvroHBaseRecord.schemaString, HBaseTableCatalog.tableCatalog ->
avroCatalogInsert,
 HBaseTableCatalog.newTable -> "5"))
 .format("org.apache.hadoop.hbase.spark")
 .save()
 val newDF = withCatalog(avroCatalogInsert)
 newDF.show()
 newDF.printSchema()
 if (newDF.count() != 256) {
 throw new UserCustomizedSampleException("value invalid")
 }
 df.filter($"col1.name" === "name005" || $"col1.name" <= "name005")
 .select("col0", "col1.favorite_color", "col1.favorite_number")
 .show()
} finally {
 sc.stop()
}
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the AvroSource file in SparkOnHbasePythonExample.

```
-*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide Hbase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
(2). If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark2x/spark/conf/ is set to true on the Spark2x client.
Set spark.yarn.security.credentials.hbase.enabled to true.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
```

```
.builder\
 .appName("AvroSourceExample")\
 .getOrCreate()
Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.datasources.AvroSource')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.AvroSource().execute(spark._jsc)
Stop the SparkSession instance.
spark.stop()
```

### 30.3.4.2 Performing Operations on the HBase Data Source

#### Scenario

Users can use HBase as data sources in Spark applications, write DataFrame to HBase, read data from HBase, and filter the read data.

#### Data Planning

On the client, run the **hbase shell** command to go to the HBase command line and run the following commands to create HBase tables to be used in the sample code:

```
create 'HBaseSourceExampleTable','rowkey','cf1','cf2','cf3','cf4','cf5','cf6','cf7','cf8'
```

#### Development Guideline

1. Create an RDD.
2. Perform operations on HBase to treat it as the data source and write the generated RDD into HBase tables.
3. Read data from HBase tables and perform simple operations on the data.

#### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **super**, change the value to the prepared development user name.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

 NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name of the case code is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$\$SPARK\_HOME** directory on the client. Run the following commands in the **\$\$SPARK\_HOME** directory.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.HBaseSource SparkOnHbaseJavaExample.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample.jar,/opt/female/protobuf-java-2.5.0.jar HBaseSource.py
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.HBaseSource --files /opt/user.keytab,/opt/krb5.conf SparkOnHbaseJavaExample.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --files /opt/user.keytab,/opt/krb5.conf --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample.jar,/opt/female/protobuf-java-2.5.0.jar HBaseSource.py
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the **HBaseSource** file in **SparkOnHbaseJavaExample**.

```
public static void main(String args[]) throws IOException{
 LoginUtil.loginWithUserKeytab();
 SparkConf sparkConf = new SparkConf().setAppName("HBaseSourceExample");
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 SQLContext sqlContext = new SQLContext(jsc);

 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc,conf);
 try{
 List<HBaseRecord> list = new ArrayList<HBaseRecord>();
 for(int i=0 ; i<256; i++){
 list.add(new HBaseRecord(i));
 }
 Map map = new HashMap<String, String>();
 map.put(HBaseTableCatalog.tableCatalog(), cat);
 map.put(HBaseTableCatalog.newTable(), "5");
 System.out.println("Before insert data into hbase table");
 sqlContext.createDataFrame(list,
HBaseRecord.class).write().options(map).format("org.apache.hadoop.hbase.spark").save();
 Dataset<Row> ds = withCatalog(sqlContext, cat);
 System.out.println("After insert data into hbase table");
 ds.printSchema();
 ds.show();
 ds.filter("key <= 'row5']").select("key","col1").show();
 ds.registerTempTable("table1");
 Dataset<Row> tempDS = sqlContext.sql("select count(col1) from table1 where key < 'row5'");
 tempDS.show();
 } finally {
 jsc.stop();
 }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseSource file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
 LoginUtil.loginWithUserKeytab()
 val sparkConf = new SparkConf().setAppName("HBaseSourceExample")
 val sc = new SparkContext(sparkConf)
 val sqlContext = new SQLContext(sc)
 val conf = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc,conf)
 import sqlContext.implicits._
 def withCatalog(cat: String): DataFrame = {
 sqlContext
 .read
 .options(Map(HBaseTableCatalog.tableCatalog->cat))
 .format("org.apache.hadoop.hbase.spark")
 .load()
 }
 val data = (0 to 255).map { i =>
 HBaseRecord(i)
 }
 try{
 sc.parallelize(data).toDF.write.options(
 Map(HBaseTableCatalog.tableCatalog -> cat, HBaseTableCatalog.newTable -> "5"))
 .format("org.apache.hadoop.hbase.spark")
 .save()
 val df = withCatalog(cat)
 df.show()
 df.filter($"col0" <= "row005")
 .select($"col0", $"col1").show
 df.registerTempTable("table1")
 val c = sqlContext.sql("select count(col1) from table1 where col0 < 'row050'")
 c.show()
 } finally {
```



```
sc.stop()
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseSource file in SparkOnHbasePythonExample.

```
-*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement the required operations.
(2). If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark2x/spark/conf/ is set to true on the Spark2x client.
 Set spark.yarn.security.credentials.hbase.enabled to true.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("HBaseSourceExample")\
 .getOrCreate()
Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.datasources.HBaseSource')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.HBaseSource().execute(spark._jsc)
Stop the SparkSession instance.
spark.stop()
```

### 30.3.4.3 Using the BulkPut Interface

#### Scenario

Users can use the HBaseContext method to use HBase in Spark applications and write the constructed RDD into HBase.

#### Data Planning

On the client, run the **hbase shell** command to go to the HBase command line and run the following commands to create HBase tables to be used in the sample code:

```
create 'bulktable','cf1'
```

#### Development Guideline

1. Create an RDD.
2. Perform operations on HBase in HBaseContext mode and write the generated RDD into the HBase table.

#### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on

FusionInsight Manager. The user in the sample code is **super**, change the value to the prepared development user name.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory. Java is displayed before the class name of the Java API. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkPutExa
mple SparkOnHbaseJavaExample.jar bulktable cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars
SparkOnHbaseJavaExample.jar HBaseBulkPutExample.py bulktable cf1
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkPutExa
mple --files /opt/user.keytab,/opt/krb5.conf
SparkOnHbaseJavaExample.jar bulktable cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar
HBaseBulkPutExample.py bulktable cf1
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the JavaHBaseBulkPutExample file in SparkOnHbaseJavaExample.

```
public static void main(String[] args) throws Exception{
 if (args.length < 2) {
 System.out.println("JavaHBaseBulkPutExample " +
 "{tableName} {columnFamily}");
 return;
 }
 LoginUtil.loginWithUserKeytab();
 String tableName = args[0];
 String columnFamily = args[1];
 SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkPutExample " + tableName);
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 try {
 List<String> list = new ArrayList<String>(5);
 list.add("1," + columnFamily + ",1,1");
 list.add("2," + columnFamily + ",1,2");
 list.add("3," + columnFamily + ",1,3");
 list.add("4," + columnFamily + ",1,4");
 list.add("5," + columnFamily + ",1,5");
 list.add("6," + columnFamily + ",1,6");
 list.add("7," + columnFamily + ",1,7");
 list.add("8," + columnFamily + ",1,8");
 list.add("9," + columnFamily + ",1,9");
 list.add("10," + columnFamily + ",1,10");
 JavaRDD<String> rdd = jsc.parallelize(list);
 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
 hbaseContext.bulkPut(rdd,
 TableName.valueOf(tableName),
 new PutFunction());
 System.out.println("Bulk put into Hbase successfully!");
 } finally {
 jsc.stop();
 }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkPutExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
 if (args.length < 2) {
 System.out.println("HBaseBulkPutTimestampExample {tableName} {columnFamily} are missing an
argument")
 return
 }
 LoginUtil.loginWithUserKeytab()
 val tableName = args(0)
 val columnFamily = args(1)
 val sparkConf = new SparkConf().setAppName("HBaseBulkPutTimestampExample " +
 tableName + " " + columnFamily)
 val sc = new SparkContext(sparkConf)
 try {
 val rdd = sc.parallelize(Array(
 Bytes.toBytes("1"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("1"))),
 Bytes.toBytes("2"),
))
 }
}
```

```
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("2"))),
 (Bytes.toBytes("3"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("3"))),
 (Bytes.toBytes("4"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("4"))),
 (Bytes.toBytes("5"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("5"))),
 (Bytes.toBytes("6"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("6"))),
 (Bytes.toBytes("7"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("7"))),
 (Bytes.toBytes("8"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("8"))),
 (Bytes.toBytes("9"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("9"))),
 (Bytes.toBytes("10"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("10")))))
val conf = HBaseConfiguration.create()
val timeStamp = System.currentTimeMillis()
val hbaseContext = new HBaseContext(sc, conf)
hbaseContext.bulkPut[(Array[Byte], Array[(Array[Byte], Array[Byte], Array[Byte])])](rdd,
 TableName.valueOf(tableName),
 (putRecord) => {
 val put = new Put(putRecord._1)
 putRecord._2.foreach((putValue) => put.addColumn(putValue._1, putValue._2,
 timeStamp, putValue._3))
 put
 })
} finally {
 sc.stop()
}
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkPutExample` file in `SparkOnHbasePythonExample`.

```
-*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark2x/spark/conf/ is set to true on the Spark2x client.
 Set spark.yarn.security.credentials.hbase.enabled to true.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseBulkPutExample")\
 .getOrCreate()
Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkPutExample')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseBulkPutExample().execute(spark._jsc, sys.argv)
Stop the SparkSession instance.
spark.stop()
```

### 30.3.4.4 Using the BulkGet Interface

#### Scenario

Users can use the `HBaseContext` method to use HBase in Spark applications, construct the rowkey of the data to be obtained into RDDs, and obtain the data

corresponding to the rowkey in the HBase tables through the BulkGet interface of HBaseContext.

## Data Planning

Perform operations based on the HBase tables and data in the tables that are created in [Using the BulkPut Interface](#).

## Development Guideline

1. Creates RDDs containing the rowkey to be obtained.
2. Perform operations on HBase in HBaseContext mode and obtain data corresponding to rowkey in HBase tables through the BulkGet interface of HBaseContext.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on FusionInsight Manager. The user in the sample code is **super**, change the value to the prepared development user name.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkGetExa
mple SparkOnHbaseJavaExample.jar bulktable
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars
SparkOnHbaseJavaExample.jar HBaseBulkGetExample.py bulktable
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkGetExa
mple --files /opt/user.keytab,/opt/krb5.conf
SparkOnHbaseJavaExample.jar bulktable
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar
HBaseBulkGetExample.py bulktable
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkGetExample` file in `SparkOnHbaseJavaExample`.

```
public static void main(String[] args) throws IOException{
 if (args.length < 1) {
 System.out.println("JavaHBaseBulkGetExample {tableName}");
 return;
 }
 LoginUtil.loginWithUserKeytab();
 String tableName = args[0];
 SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkGetExample " + tableName);
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 try {
 List<byte[]> list = new ArrayList<byte[]>(5);
 list.add(Bytes.toBytes("1"));
 list.add(Bytes.toBytes("2"));
 list.add(Bytes.toBytes("3"));
 list.add(Bytes.toBytes("4"));
 list.add(Bytes.toBytes("5"));
 JavaRDD<byte[]> rdd = jsc.parallelize(list);
 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
 List resultList = hbaseContext.bulkGet(TableName.valueOf(tableName), 2, rdd, new GetFunction(),
 new ResultFunction()).collect();
 for(int i =0 ;i<resultList.size();i++){
 System.out.println(resultList.get(i));
 }
 } finally {
 jsc.stop();
 }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkGetExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
 if (args.length < 1) {
 println("HBaseBulkGetExample {tableName} missing an argument")
 return
 }
 LoginUtil.loginWithUserKeytab()
 val tableName = args(0)
 val sparkConf = new SparkConf().setAppName("HBaseBulkGetExample " + tableName)
 val sc = new SparkContext(sparkConf)
 try {
 //[(Array[Byte])]
 val rdd = sc.parallelize(Array(
 Bytes.toBytes("1"),
 Bytes.toBytes("2"),
 Bytes.toBytes("3"),
 Bytes.toBytes("4"),
 Bytes.toBytes("5"),
 Bytes.toBytes("6"),
 Bytes.toBytes("7")))
 val conf = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc, conf)
 val getRdd = hbaseContext.bulkGet[Array[Byte], String](
 TableName.valueOf(tableName),
 2,
 rdd,
 record => {
 System.out.println("making Get")
 new Get(record)
 },
 (result: Result) => {
 val it = result.listCells().iterator()
 val b = new StringBuilder
 b.append(Bytes.toString(result.getRow) + ":")
 while (it.hasNext) {
 val cell = it.next()
 val q = Bytes.toString(CellUtil.cloneQualifier(cell))
 if (q.equals("counter")) {
 b.append("(" + q + "," + Bytes.toLong(CellUtil.cloneValue(cell)) + ")")
 } else {
 b.append("(" + q + "," + Bytes.toString(CellUtil.cloneValue(cell)) + ")")
 }
 }
 b.toString()
 })
 getRdd.collect().foreach(v => println(v))
 } finally {
 sc.stop()
 }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkGetExample file in SparkOnHbasePythonExample.

```
-*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark2x/spark/conf/ is set to true on the Spark2x client.
Set spark.yarn.security.credentials.hbase.enabled to true.
```

```
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseBulkGetExample")\
 .getOrCreate()
Import required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkGetExample')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseBulkGetExample().execute(spark._jsc, sys.argv)
Stop the SparkSession instance.
spark.stop()
```

### 30.3.4.5 Using the BulkDelete Interface

#### Scenario

Users can use the HBaseContext method to use HBase in Spark applications, construct rowkey of the data to be deleted into RDDs, and delete the data corresponding to the rowkey in HBase tables through the BulkDelete interface of HBaseContext.

#### Data Planning

Perform operations based on the HBase tables and data in the tables that are created in [Using the BulkPut Interface](#).

#### Development Guideline

1. Create RDDs containing the rowkey to be deleted.
2. Perform operations on the HBase in HBaseContext mode and delete the data corresponding to the rowkey in HBase tables through the BulkDelete interface of HBaseContext.

#### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on FusionInsight Manager. The user in the sample code is **super**, change the value to the prepared development user name.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).



 NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkDeleteE
xample SparkOnHbaseJavaExample.jar bulktable
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars
SparkOnHbaseJavaExample.jar HBaseBulkDeleteExample.py bulktable
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkDeleteE
xample --files /opt/user.keytab,/opt/krb5.conf
SparkOnHbaseJavaExample.jar bulktable
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar
HBaseBulkDeleteExample.py bulktable
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the **HBaseBulkDeleteExample** file in **SparkOnHbaseJavaExample**.

```
public static void main(String[] args) throws IOException {
 if (args.length < 1) {
 System.out.println("JavaHBaseBulkDeleteExample {tableName}");
 return;
 }
}
```

```
LoginUtil.loginWithUserKeytab();
String tableName = args[0];
SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkDeleteExample " + tableName);
JavaSparkContext jsc = new JavaSparkContext(sparkConf);
try {
 List<byte[]> list = new ArrayList<byte[]>(5);
 list.add(Bytes.toBytes("1"));
 list.add(Bytes.toBytes("2"));
 list.add(Bytes.toBytes("3"));
 list.add(Bytes.toBytes("4"));
 list.add(Bytes.toBytes("5"));
 JavaRDD<byte[]> rdd = jsc.parallelize(list);
 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
 hbaseContext.bulkDelete(rdd,
 TableName.valueOf(tableName), new DeleteFunction(), 4);
 System.out.println("Bulk Delete successfully!");
} finally {
 jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkDeleteExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
 if (args.length < 1) {
 println("HBaseBulkDeleteExample {tableName} missing an argument")
 return
 }
 LoginUtil.loginWithUserKeytab()
 val tableName = args(0)
 val sparkConf = new SparkConf().setAppName("HBaseBulkDeleteExample " + tableName)
 val sc = new SparkContext(sparkConf)
 try {
 //Array[Byte]
 val rdd = sc.parallelize(Array(
 Bytes.toBytes("1"),
 Bytes.toBytes("2"),
 Bytes.toBytes("3"),
 Bytes.toBytes("4"),
 Bytes.toBytes("5")
))
 val conf = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc, conf)
 hbaseContext.bulkDelete[Array[Byte]](rdd,
 TableName.valueOf(tableName),
 putRecord => new Delete(putRecord),
 4)
 } finally {
 sc.stop()
 }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkDeleteExample file in SparkOnHbasePythonExample.

```
def main(args: Array[String]) {
-*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. This example uses Python to invoke Java code to
implement required operations.
```

```
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the spark-defaults.conf file under Spark2x/spark/conf/ is set to true on the Spark2x client.
Set spark.yarn.security.credentials.hbase.enabled to true.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseBulkDeleteExample")\
 .getOrCreate()
Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkDeleteExample')
Create a class instance and invoke the method, Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseBulkDeleteExample().execute(spark._jsc, sys.argv)
Stop the SparkSession instance.
spark.stop()
```

### 30.3.4.6 Using the BulkLoad Interface

#### Scenario

Users can use HBaseContext to use HBase in Spark applications, construct rowkey of the data to be inserted into RDDs, write RDDs into HFiles through the BulkLoad interface of HBaseContext. The following command is used to import the generated HFiles to the HBase table and will not be described in this section.

```
hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles
{hfilePath} {tableName}
```

#### Data Planning

1. Run the **hbase shell** command on the client to go to the HBase command line.
2. Run the following command to create HBase tables:  
**create 'bulkload-table-test','f1','f2'**

#### Development Guideline

1. Construct the data to be imported into RDDs
2. Perform operations on HBase in HBaseContext mode and write RDDs into HFiles through the BulkLoad interface of HBaseContext.

#### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on FusionInsight Manager. The user in the sample code is **super**, change the value to the prepared development user name.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.

- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkLoadExample SparkOnHbaseJavaExample.jar /tmp/hfile bulkload-table-test
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars
SparkOnHbaseJavaExample.jar HBaseBulkLoadExample.py /tmp/hfile
bulkload-table-test
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkLoadExample --files /opt/user.keytab,/opt/krb5.conf
SparkOnHbaseJavaExample.jar /tmp/hfile bulkload-table-test
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar
HBaseBulkLoadExample.py /tmp/hfile bulkload-table-test
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `JavaHBaseBulkLoadExample` file in `SparkOnHbaseJavaExample`.

```
public static void main(String[] args) throws IOException{
 if (args.length < 2) {
 System.out.println("JavaHBaseBulkLoadExample {outputPath} {tableName}");
 return;
 }
 LoginUtil.loginWithUserKeytab();
 String outputPath = args[0];
 String tableName = args[1];
 String columnFamily1 = "f1";
```

```
String columnFamily2 = "f2";
SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkLoadExample " + tableName);
JavaSparkContext jsc = new JavaSparkContext(sparkConf);
try {
 List<String> list= new ArrayList<String>();
 // row1
 list.add("1," + columnFamily1 + ",b,1");
 // row3
 list.add("3," + columnFamily1 + ",a,2");
 list.add("3," + columnFamily1 + ",b,1");
 list.add("3," + columnFamily2 + ",a,1");
 /* row2 */
 list.add("2," + columnFamily2 + ",a,3");
 list.add("2," + columnFamily2 + ",b,3");
 JavaRDD<String> rdd = jsc.parallelize(list);
 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
 hbaseContext.bulkLoad(rdd, TableName.valueOf(tableName),new BulkLoadFunction(), outputPath,
 new HashMap<byte[], FamilyHFileWriteOptions>(), false, HConstants.DEFAULT_MAX_FILE_SIZE);
} finally {
 jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkLoadExample` file in `SparkOnHbaseScalaExample`.

```
def main(args: Array[String]) {
 if(args.length < 2) {
 println("HBaseBulkLoadExample {outputPath} {tableName}")
 return
 }
 LoginUtil.loginWithUserKeytab()
 val Array(outputPath, tableName) = args
 val columnFamily1 = "f1"
 val columnFamily2 = "f2"
 val sparkConf = new SparkConf().setAppName("JavaHBaseBulkLoadExample " + tableName)
 val sc = new SparkContext(sparkConf)
 try {
 val arr = Array("1," + columnFamily1 + ",b,1",
 "2," + columnFamily1 + ",a,2",
 "3," + columnFamily1 + ",b,1",
 "3," + columnFamily2 + ",a,1",
 "4," + columnFamily2 + ",a,3",
 "5," + columnFamily2 + ",b,3")

 val rdd = sc.parallelize(arr)
 val config = HBaseConfiguration.create
 val hbaseContext = new HBaseContext(sc, config)
 hbaseContext.bulkLoad[String](rdd,
 TableName.valueOf(tableName),
 (putRecord) => {
 if(putRecord.length > 0) {
 val strArray = putRecord.split(",")
 val kfq = new KeyFamilyQualifier(Bytes.toBytes(strArray(0)), Bytes.toBytes(strArray(1)),
Bytes.toBytes(strArray(2)))
 val ite = (kfq, Bytes.toBytes(strArray(3)))
 val itea = List(ite).iterator
 itea
 } else {
 null
 }
 },
 outputPath)
 } finally {
 sc.stop()
 }
}
```

```
}
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkLoadPythonExample` file in `SparkOnHbasePythonExample`.

```
-*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark2x/spark/conf/ is set to true on the Spark2x client.
Set spark.yarn.security.credentials.hbase.enabled to true.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseBulkLoadExample")\
 .getOrCreate()
Import required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.HBaseBulkLoadPythonExample')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.HBaseBulkLoadPythonExample().hbaseBulkLoad(spark._jsc, sys.argv[1], sys.argv[2])
Stop the SparkSession instance.
spark.stop()
```

### 30.3.4.7 Using the `foreachPartition` Interface

#### Scenario

Users can use `HBaseContext` to perform operations on HBase in the Spark application, construct rowkey of the data to be inserted into RDDs, and write RDDs to HBase tables through the `mapPartition` interface of `HBaseContext`.

#### Data Planning

1. Run the `hbase shell` command on the client to go to the HBase command line.
2. Run the following command to create an HBase table:

```
create 'table2','cf1'
```

#### Development Guideline

1. Construct the data to be imported into RDDs
2. Perform operations on HBase in `HBaseContext` mode and concurrently write data to HBase through the `foreachPartition` interface of `HBaseContext`.

#### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (`user.keytab` and `krb5.conf`). The `user.keytab` and `krb5.conf` files are authentication files in the security mode. Download the authentication credentials of the user `principal` on

the FusionInsight Manager page. The user in the sample code is **super**, change the value to the prepared development user name.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode client --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseForEachPartitionExample SparkOnHbaseJavaExample.jar table2 cf1**  
Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.  
**bin/spark-submit --master yarn --deploy-mode client --jars SparkOnHbaseJavaExample.jar HBaseForEachPartitionExample.py table2 cf1**
- yarn-cluster mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode cluster --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseForEachPartitionExample --files /opt/user.keytab,/opt/krb5.conf SparkOnHbaseJavaExample.jar table2 cf1**  
Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar
HBaseForEachPartitionExample.py table2 cf1
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `JavaHBaseForEachPartitionExample` file in `SparkOnHbaseJavaExample`.

```
public static void main(String[] args) throws IOException {
 if (args.length < 1) {
 System.out.println("JavaHBaseForEachPartitionExample {tableName} {columnFamily}");
 return;
 }
 LoginUtil.loginWithUserKeytab();
 final String tableName = args[0];
 final String columnFamily = args[1];
 SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkGetExample " + tableName);
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 try {
 List<byte[]> list = new ArrayList<byte[]>(5);
 list.add(Bytes.toBytes("1"));
 list.add(Bytes.toBytes("2"));
 list.add(Bytes.toBytes("3"));
 list.add(Bytes.toBytes("4"));
 list.add(Bytes.toBytes("5"));
 JavaRDD<byte[]> rdd = jsc.parallelize(list);
 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
 hbaseContext.foreachPartition(rdd,
 new VoidFunction<Tuple2<Iterator<byte[]>, Connection>>() {
 public void call(Tuple2<Iterator<byte[]>, Connection> t)
 throws Exception {
 Connection con = t._2();
 Iterator<byte[]> it = t._1();
 BufferedMutator buf = con.getBufferedMutator(TableName.valueOf(tableName));
 while (it.hasNext()) {
 byte[] b = it.next();
 Put put = new Put(b);
 put.add(Bytes.toBytes(columnFamily), Bytes.toBytes("cid"), b);
 buf.mutate(put);
 }
 buf.flush();
 buf.close();
 }
 }
);
 } finally {
 jsc.stop();
 }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseForEachPartitionExample` file in `SparkOnHbaseScalaExample`.

```
def main(args: Array[String]) {
 if (args.length < 2) {
 println("HBaseForEachPartitionExample {tableName} {columnFamily} are missing an arguments")
 return
 }
 LoginUtil.loginWithUserKeytab()
 val tableName = args(0)
 val columnFamily = args(1)
 val sparkConf = new SparkConf().setAppName("HBaseForEachPartitionExample " +
```



```
tableName + " " + columnFamily)
val sc = new SparkContext(sparkConf)
try {
 //[(Array[Byte], Array[(Array[Byte], Array[Byte], Array[Byte])])]
 val rdd = sc.parallelize(Array(
 (Bytes.toBytes("1"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("1"))),
 (Bytes.toBytes("2"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("2"))),
 (Bytes.toBytes("3"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("3"))),
 (Bytes.toBytes("4"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("4"))),
 (Bytes.toBytes("5"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("5"))))
))
 val conf = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc, conf)
 rdd.hbaseForeachPartition(hbaseContext,
 (it, connection) => {
 val m = connection.getBufferedMutator(tableName.valueOf(tableName))
 it.foreach(r => {
 val put = new Put(r._1)
 r._2.foreach((putValue) =>
 put.addColumn(putValue._1, putValue._2, putValue._3))
 m.mutate(put)
 })
 m.flush()
 m.close()
 })
 } finally {
 sc.stop()
 }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseForEachPartitionExample` file in `SparkOnHbasePythonExample`.

```
-*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark2x/spark/conf/ is set to true
Set spark.yarn.security.credentials.hbase.enabled to true.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseForEachPartitionExample")\
 .getOrCreate()
Import the required class to sc._jvm.
java_import(spark._jvm,
'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseForEachPartitionExample')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseForEachPartitionExample().execute(spark._jsc, sys.argv)
Stop the SparkSession instance.
spark.stop()
```

### 30.3.4.8 Distributedly Scanning HBase Tables

#### Scenario

Users can use HBaseContext to perform operations on HBase in Spark applications and use HBase RDDs to scan HBase tables based on specific rules.

#### Data Planning

Use HBase tables created in [Performing Operations on Data in Avro Format](#)

#### Development Guideline

1. Set the scanning rule. For example: setCaching.
2. Use specific rules to scan the HBase table.

#### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **super**, change the value to the prepared development user name.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

#### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set configuration item **spark.inputFormat.cache.enabled** to **false**.

#### Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseDistributedS
canExample SparkOnHbaseJavaExample.jar ExampleAvrotable
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars
SparkOnHbaseJavaExample.jar HBaseDistributedScanExample.py
ExampleAvrotable
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseDistributedS
canExample --files /opt/user.keytab,/opt/krb5.conf
SparkOnHbaseJavaExample.jar ExampleAvrotable
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar
HBaseDistributedScanExample.py ExampleAvrotable
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `JavaHBaseDistributedScanExample` file in `SparkOnHbaseJavaExample`.

```
public static void main(String[] args) throws IOException{
 if (args.length < 1) {
 System.out.println("JavaHBaseDistributedScan {tableName}");
 return;
 }
 LoginUtil.loginWithUserKeytab();
 String tableName = args[0];
 SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseDistributedScan " + tableName);
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 try {
 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
 Scan scan = new Scan();
 scan.setCaching(100);
 JavaRDD<Tuple2<ImmutableBytesWritable, Result>> javaRdd =
 hbaseContext.hbaseRDD(tableName.valueOf(tableName), scan);
 List<String> results = javaRdd.map(new ScanConvertFunction()).collect();
 System.out.println("Result Size: " + results.size());
 } finally {
 jsc.stop();
 }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseDistributedScanExample` file in `SparkOnHbaseScalaExample`.

```
def main(args: Array[String]) {
 if (args.length < 1) {
 println("HBaseDistributedScanExample {tableName} missing an argument")
 return
 }
 LoginUtil.loginWithUserKeytab()
 val tableName = args(0)
 val sparkConf = new SparkConf().setAppName("HBaseDistributedScanExample " + tableName)
 val sc = new SparkContext(sparkConf)
 try {
 val conf = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc, conf)
 val scan = new Scan()
 scan.setCaching(100)
 val getRdd = hbaseContext.hbaseRDD(tableName.valueOf(tableName), scan)
 getRdd.foreach(v => println(Bytes.toString(v._1.get())))
 println("Length: " + getRdd.map(r => r._1.copyBytes()).collect().length);
 } finally {
 sc.stop()
 }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseDistributedScanExample file in SparkOnHbasePythonExample.

```
-*- coding:utf-8 -*-
-*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark2x/spark/conf/ is set to true on the Spark2x client.
Set spark.yarn.security.credentials.hbase.enabled to true.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseDistributedScan")\
 .getOrCreate()
Import the required class into sc._jvm.
java_import(spark._jvm,
'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseDistributedScanExample')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseDistributedScan().execute(spark._jsc, sys.argv)
Stop the SparkSession instance.
spark.stop()
```

### 30.3.4.9 Using the mapPartition Interface

#### Scenario

Users can use the HBaseContext method to perform operations on HBase in Spark applications and use the mapPartition interface to traverse HBase tables in parallel.

#### Data Planning

Use HBase tables created in [Using the foreachPartition Interface](#).

## Development Guideline

1. Construct RDDs corresponding to rowkey in HBase tables to be traversed.
2. Use the mapPartition interface to traverse the data corresponding to rowkey and perform simple operations.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **super**, change the value to the prepared development user name.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode client --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseMapPartitionExample SparkOnHbaseJavaExample.jar table2**  
Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.  
**bin/spark-submit --master yarn --deploy-mode client --jars SparkOnHbaseJavaExample.jar HBaseMapPartitionExample.py table2**
- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseMapPartitio
nExample --files /opt/user.keytab,/opt/krb5.conf
SparkOnHbaseJavaExample.jar table2
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar
HBaseMapPartitionExample.py table2
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `JavaHBaseMapPartitionExample` file in `SparkOnHbaseJavaExample`.

```
public static void main(String args[]) throws IOException {
 if(args.length <1){
 System.out.println("JavaHBaseMapPartitionExample {tableName} is missing an argument");
 return;
 }
 LoginUtil.loginWithUserKeytab();
 final String tableName = args[0];
 SparkConf sparkConf = new SparkConf().setAppName("HBaseMapPartitionExample " + tableName);
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 try{
 List<byte []> list = new ArrayList();
 list.add(Bytes.toBytes("1"));
 list.add(Bytes.toBytes("2"));
 list.add(Bytes.toBytes("3"));
 list.add(Bytes.toBytes("4"));
 list.add(Bytes.toBytes("5"));

 JavaRDD<byte []> rdd = jsc.parallelize(list);
 Configuration hbaseconf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, hbaseconf);
 JavaRDD getRDD = hbaseContext.mapPartitions(rdd, new
FlatMapFunction<Tuple2<Iterator<byte[]>,Connection>,Object>() {
 public Iterator call(Tuple2<Iterator<byte[]>, Connection> t)
 throws Exception {
 Table table = t._2.getTable(TableName.valueOf(tableName));
 //go through rdd
 List<String> list = new ArrayList<String>();
 while(t._1.hasNext()){
 byte[] bytes = t._1.next();
 Result result = table.get(new Get(bytes));
 Iterator<Cell> it = result.listCells().iterator();
 StringBuilder sb = new StringBuilder();
 sb.append(Bytes.toString(result.getRow()) + " ");
 while(it.hasNext()){
 Cell cell = it.next();
 String column = Bytes.toString(cell.getQualifierArray());
 if(column.equals("counter")){
 sb.append("(" + column + "," + Bytes.toLong(cell.getValueArray()) + ")");
 } else {
 sb.append("(" + column + "," + Bytes.toString(cell.getValueArray()) + ")");
 }
 }
 }
 list.add(sb.toString());
 }
 }
 }
}
```

```
 }
 return list.iterator();
 }
});
List<byte[]> resultList = getrdd.collect();
if(null == resultList || 0 == resultList.size()){
 System.out.println("Nothing matches!");
}else{
 for(int i =0; i< resultList.size(); i++){
 System.out.println(resultList.get(i));
 }
}
} finally {
 jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseMapPartitionExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
 if (args.length < 1) {
 println("HBaseMapPartitionExample {tableName} is missing an argument")
 return
 }
 LoginUtil.loginWithUserKeytab()
 val tableName = args(0)
 val sparkConf = new SparkConf().setAppName("HBaseMapPartitionExample " + tableName)
 val sc = new SparkContext(sparkConf)
 try {
 //[(Array[Byte])]
 val rdd = sc.parallelize(Array(
 Bytes.toBytes("1"),
 Bytes.toBytes("2"),
 Bytes.toBytes("3"),
 Bytes.toBytes("4"),
 Bytes.toBytes("5")))
 val conf = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc, conf)
 val b = new StringBuilder
 val getRdd = rdd.hbaseMapPartitions[String](hbaseContext, (it, connection) => {
 val table = connection.getTable(TableName.valueOf(tableName))
 it.map{r =>
 //batching would be faster. This is just an example
 val result = table.get(new Get(r))
 val it = result.listCells().iterator()
 b.append(Bytes.toString(result.getRow) + ":")
 while (it.hasNext) {
 val cell = it.next()
 val q = Bytes.toString(cell.getQualifierArray)
 if (q.equals("counter")) {
 b.append("(" + q + "," + Bytes.toLong(cell.getValueArray) + ")")
 } else {
 b.append("(" + q + "," + Bytes.toString(cell.getValueArray) + ")")
 }
 }
 }
 b.toString()
 })
 getRdd.collect().foreach(v => println(v))
 } finally {
 sc.stop()
 }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseMapPartitionExample file in SparkOnHbasePythonExample.

```
-*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark2x/spark/conf/ is set to true on the Spark2x client.
 Set spark.yarn.security.credentials.hbase.enabled to true.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseMapPartitionExample")\
 .getOrCreate()
Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseMapPartitionExample')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseMapPartitionExample().execute(spark._jsc, sys.argv)
Stop the SparkSession instance.
spark.stop()
```

### 30.3.4.10 Writing Data to HBase Tables In Batches Using SparkStreaming

#### Scenario

Users can use HBaseContext to perform operations on HBase in Spark applications and write streaming data to HBase tables using the streamBulkPut interface.

#### Data Planning

1. Create a session connected to the client and run the **hbase shell** command in the session to go to the HBase command line.
2. Run the following command in the HBase command line to create an HBase table:

```
create 'streamingTable','cf1'
```

3. In another session of the client, run the Linux command to construct a port for receiving data. (The command may be different for servers running different operating systems. For the SUSE operating system, the following command is used: **netcat -lk 9999**.)

```
nc -lk 9999
```

After the command for submitting a task is executed, enter the data to be submitted in this command and receive the data through the HBase table.

#### NOTE

To construct a port for receiving data, you need to install netcat on the server where the client is located.

#### Development Guideline

1. Use SparkStreaming to continuously read data from a specific port.



2. Write the read Dstream to HBase tables through the streamBulkPut interface.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **super**, change the value to the prepared development user name.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:  
Java/Scala version. (The class name must be the same as the actual code. The following is only an example.) **#{ip}** must be the IP address of the host where the **nc -lk 9999** command is executed.  
**bin/spark-submit --master yarn --deploy-mode client --class com.huawei.bigdata.spark.examples.streaming.JavaHBaseStreamingBulkPutExample SparkOnHbaseJavaExample.jar #{ip} 9999 streamingTable cf1**  
Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.  
**bin/spark-submit --master yarn --deploy-mode client --jars SparkOnHbaseJavaExample.jar HBaseStreamingBulkPutExample.py #{ip} 9999 streamingTable cf1**
- yarn-cluster mode:

Java/Scala version. (The class name must be the same as the actual code. The following is only an example.) `#{ip}` must be the IP address of the host where the `nc -lk 9999` command is executed.

```
bin/spark-submit --master yarn --deploy-mode cluster --class
com.huawei.bigdata.spark.examples.streaming.JavaHBaseStreamingBulkP
utExample --files /opt/user.keytab,/opt/krb5.conf
SparkOnHbaseJavaExample.jar #{ip} 9999 streamingTable cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is `SparkOnHbaseJavaExample.jar` and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar
HBaseStreamingBulkPutExample.py #{ip} 9999 streamingTable cf1
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `JavaHBaseStreamingBulkPutExample` file in `SparkOnHbaseJavaExample`.

### NOTE

The `awaitTerminationOrTimeout()` method is used to set the task timeout interval (in milliseconds). You are advised to set this parameter based on the expected task execution time.

```
public static void main(String[] args) throws IOException {
 if (args.length < 4) {
 System.out.println("JavaHBaseBulkPutExample " +
 "{host} {port} {tableName}");
 return;
 }
 LoginUtil.loginWithUserKeytab();
 String host = args[0];
 String port = args[1];
 String tableName = args[2];
 String columnFamily = args[3];
 SparkConf sparkConf =
 new SparkConf().setAppName("JavaHBaseStreamingBulkPutExample " +
 tableName + ":" + port + ":" + tableName);
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 try {
 JavaStreamingContext jssc =
 new JavaStreamingContext(jsc, new Duration(1000));
 JavaReceiverInputDStream<String> javaDstream =
 jssc.socketTextStream(host, Integer.parseInt(port));
 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
 hbaseContext.streamBulkPut(javaDstream,
 TableName.valueOf(tableName),
 new PutFunction(columnFamily));
 jssc.start();
 jssc.awaitTerminationOrTimeout(60000);
 jssc.stop(false,true);
 } catch (InterruptedException e){
 e.printStackTrace();
 } finally {
 jsc.stop();
 }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseStreamingBulkPutExample` file in `SparkOnHbaseScalaExample`.

### NOTE

The `awaitTerminationOrTimeout()` method is used to set the task timeout interval (in milliseconds). You are advised to set this parameter based on the expected task execution time.

```
def main(args: Array[String]): Unit = {
 loginUtil.loginWithUserKeytab()
 val host = args(0)
 val port = args(1)
 val tableName = args(2)
 val columnFamily = args(3)
 val conf = new SparkConf()
 conf.setAppName("HBase Streaming Bulk Put Example")
 val sc = new SparkContext(conf)
 try {
 val config = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc, config)
 val ssc = new StreamingContext(sc, Seconds(1))
 val lines = ssc.socketTextStream(host, port.toInt)
 hbaseContext.streamBulkPut[String](lines,
 TableName.valueOf(tableName),
 (putRecord) => {
 if (putRecord.length() > 0) {
 val put = new Put(Bytes.toBytes(putRecord))
 put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes("foo"), Bytes.toBytes("bar"))
 put
 } else {
 null
 }
 })
 ssc.start()
 ssc.awaitTerminationOrTimeout(60000)
 ssc.stop(stopSparkContext = false)
 } finally {
 sc.stop()
 }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseStreamingBulkPutExample` file in `SparkOnHbasePythonExample`.

```
-*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark2x/spark/conf/ is set to true on the Spark2x client.
Set spark.yarn.security.credentials.hbase.enabled to true.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseStreamingBulkPutExample")\
 .getOrCreate()
Import required class to sc._jvm.
java_import(spark._jvm,
```

```
'com.huawei.bigdata.spark.examples.streaming.JavaHBaseStreamingBulkPutExample')
Create class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseStreamingBulkPutExample().execute(spark._jsc, sys.argv)
Stop the SparkSession instance.
spark.stop()
```

## 30.3.5 Reading Data from HBase and Write It Back to HBase

### 30.3.5.1 Overview

#### Scenarios

Assume table1 of HBase stores the user consumption amount of the current day and table2 stores the history consumption data.

In table1, key=1 and cf:cid=100 indicate that the consumption amount of user 1 in the current day is 100 CNY.

In table2, key=1 and cf:cid=1000 indicate that the history consumption amount of user 1 is 1000 CNY.

The Spark application shall achieve the following function:

Add the current consumption amount (100) to the history consumption amount (1000).

The running result is that the total consumption amount of user 1 (key=1) in table2 is 1100 CNY (cf:cid=1100).

#### Data Preparation

Use the Spark-Beeline tool to create table1 and table2 (Spark table and HBase table, respectively), and insert data by HBase.

**Step 1** Ensure that JDBCServer is started. On the Spark2x client, perform the following operations using the Spark-Beeline command tool:

**Step 2** Use the Spark-Beeline tool to create Spark table1:

```
create table table1
(
key string,
cid string
)
using org.apache.spark.sql.hbase.HBaseSource
options(
hbaseTableName "table1",
keyCols "key",
colsMapping "cid=cf.cid");
```

**Step 3** Run the following command on HBase to insert data to table1:

```
put 'table1', '1', 'cf:cid', '100'
```

**Step 4** Use the Spark-Beeline tool to create Spark table2:

```
create table table2
(
 key string,
 cid string
)
using org.apache.spark.sql.hbase.HBaseSource
options(
 hbaseTableName "table2",
 keyCols "key",
 colsMapping "cid=cf.cid");
```

**Step 5** Run the following command on HBase to insert data to table 2:

```
put 'table2', '1', 'cf:cid', '1000'
----End
```

## Development Idea

1. Query the data in table1.
2. Query the data in table2 using the key value of table1.
3. Add up the queried data.
4. Write the results of the preceding step to table2.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

1. Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.
2. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

 NOTE

- Before compilation and packaging, change the paths of the `user.keytab` and `krb5.conf` files in the sample code to the actual paths on the client server where the files are located. Example: `/opt/female/user.keytab` and `/opt/female/krb5.conf`.
  - Before running the sample program, set the `spark.yarn.security.credentials.hbase.enabled` configuration item to `true` in the `spark-defaults.conf` configuration file of Spark client. (The default value is `false`. Changing the value to `true` does not affect existing services.) If you want to uninstall the HBase service, change the value back to `false` first.
3. Upload the JAR file to any directory (for example, `/opt/female/`) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the `bin/spark-submit` script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

- Run Java or Scala sample code.  
**`bin/spark-submit --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.SparkHbaseToHbase --master yarn --deploy-mode client /opt/female/SparkHbaseToHbase-1.0.jar`**
- Run the Python sample program.

 NOTE

- PySpark does not provide HBase-related APIs. Therefore, Python is used to invoke Java code in this sample. Use Maven to pack the provided Java code into a JAR package and place it in the same driver class directory. When running the Python program, configure `--jars` to load the JAR package to the directory where the Python file resides.
- The Python sample code does not provide authentication information. Configure `--keytab` and `--principal` to specify authentication information

```
bin/spark-submit --master yarn --deploy-mode client --keytab /opt/FIClient/user.keytab --principal sparkuser --conf spark.yarn.user.classpath.first=true --jars /opt/female/SparkHbaseToHbasePythonExample/SparkHbaseToHbase-1.0.jar,/opt/female/protobuf-java-2.5.0.jar /opt/female/SparkHbaseToHbasePythonExample/SparkHbaseToHbasePythonExample.py
```

### 30.3.5.2 Java Example Code

#### Function

Call HBase API using Spark to operate HBase table1, analyze the data, and then write the analyzed data to HBase table2.

#### Example Code

For details about the code, see the class `com.huawei.bigdata.spark.examples.SparkHbaseToHbase`.

## Example code:

```
/**
 * calculate data from hbase1/hbase2,then update to hbase2
 */
public class SparkHbaseToHbase {

 public static void main(String[] args) throws Exception {

 SparkConf conf = new SparkConf().setAppName("SparkHbaseToHbase");
 conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer");
 conf.set("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrator");
 JavaSparkContext jsc = new JavaSparkContext(conf);
 // Create the configuration parameter to connect the HBase. The hbase-site.xml must be included in the
 classpath.
 Configuration hbConf = HBaseConfiguration.create(jsc.hadoopConfiguration());

 // Declare the information of the table to be queried.
 Scan scan = new org.apache.hadoop.hbase.client.Scan();
 scan.addFamily(Bytes.toBytes("cf")); //column family
 org.apache.hadoop.hbase.protobuf.generated.ClientProtos.Scan proto = ProtobufUtil.toScan(scan);
 String scanToString = Base64.encodeBytes(proto.toByteArray());
 hbConf.set(TableInputFormat.INPUT_TABLE, "table1"); //table name
 hbConf.set(TableInputFormat.SCAN, scanToString);

 // Obtain the data in the table through the Spark interface.
 JavaPairRDD rdd = jsc.newAPIHadoopRDD(hbConf, TableInputFormat.class,
 ImmutableBytesWritable.class, Result.class);

 // Traverse every Partition in the HBase table1 and update the HBase table2
 //If less data, you can use rdd.foreach()

 rdd.foreachPartition(
 new VoidFunction<Iterator<Tuple2<ImmutableBytesWritable, Result>>>() {
 public void call(Iterator<Tuple2<ImmutableBytesWritable, Result>> iterator) throws Exception {
 hBaseWriter(iterator);
 }
 }
);

 jsc.stop();
 }

 /**
 * write to table2 in exetutor
 *
 * @param iterator partition data from table1
 */
 private static void hBaseWriter(Iterator<Tuple2<ImmutableBytesWritable, Result>> iterator) throws
 IOException {
 //read hbase
 String tableName = "table2";
 String columnFamily = "cf";
 String qualifier = "cid";
 Configuration conf = HBaseConfiguration.create();

 Connection connection = null;
 Table table = null;

 try {
 connection = ConnectionFactory.createConnection(conf);
 table = connection.getTable(TableName.valueOf(tableName));

 List<Get> rowList = new ArrayList<Get>();
 List<Tuple2<ImmutableBytesWritable, Result>> table1List = new
 ArrayList<Tuple2<ImmutableBytesWritable, Result>>();
 while (iterator.hasNext()) {
 Tuple2<ImmutableBytesWritable, Result> item = iterator.next();
 Get get = new Get(item._2().getRow());
 table1List.add(item);
 }
 }
 }
}
```





### 30.3.5.3 Scala Example Code

#### Function

Call HBase API using Spark to operate HBase table1, analyze the data, and then write the analyzed data to HBase table2.

#### Example Code

For details about code, see `com.huawei.bigdata.spark.examples.SparkHbasetoHbase`.

Example code:

```
/**
 * calculate data from hbase1/hbase2,then update to hbase2
 */
object SparkHbasetoHbase {

 case class FemaleInfo(name: String, gender: String, stayTime: Int)

 def main(args: Array[String]) {

 val conf = new SparkConf().setAppName("SparkHbasetoHbase")
 conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
 conf.set("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrator")
 val sc = new SparkContext(conf)
 // Create the configuration parameter to connect the HBase. The hbase-site.xml must be included in the
 classpath.
 val hbConf = HBaseConfiguration.create(sc.hadoopConfiguration)

 // Declare the information of the table to be queried.
 val scan = new Scan()
 scan.addFamily(Bytes.toBytes("cf")) //column family
 val proto = ProtobufUtil.toScan(scan)
 val scanToString = Base64.encodeBytes(proto.toByteArray)
 hbConf.set(TableInputFormat.INPUT_TABLE, "table1") //table name
 hbConf.set(TableInputFormat.SCAN, scanToString)

 // Obtain the data in the table through the Spark interface.
 val rdd = sc.newAPIHadoopRDD(hbConf, classOf[TableInputFormat], classOf[ImmutableBytesWritable],
 classOf[Result])

 // Traverse every Partition in the HBase table1 and update the HBase table2
 //If less data, you can use rdd.foreach()
 rdd.foreachPartition(x => hBaseWriter(x))

 sc.stop()
 }

 /**
 * write to table2 in exetutor
 *
 * @param iterator partition data from table1
 */
 def hBaseWriter(iterator: Iterator[(ImmutableBytesWritable, Result)]): Unit = {
 //read hbase
 val tableName = "table2"
 val columnFamily = "cf"
 val qualifier = "cid"
 val conf = HBaseConfiguration.create()

 var table: Table = null
 var connection: Connection = null

 try {
```

```

connection = ConnectionFactory.createConnection(conf)
table = connection.getTable(TableName.valueOf(tableName))

val iteratorArray = iterator.toArray
val rowList = new util.ArrayList[Get]()
for (row <- iteratorArray) {
 val get = new Get(row._2.getRow)
 rowList.add(get)
}

//get data from hbase table2
val resultDataBuffer = table.get(rowList)

//set data for hbase
val putList = new util.ArrayList[Put]()
for (i <- 0 until iteratorArray.size) {
 val resultData = resultDataBuffer(i) //hbase2 row
 if (!resultData.isEmpty) {
 //query hbase1Value
 var hbase1Value = ""
 val it = iteratorArray(i)._2.listCells().iterator()
 while (it.hasNext) {
 val c = it.next()
 // query table1 value by column family and column qualifier
 if (columnFamily.equals(Bytes.toString(CellUtil.cloneFamily(c)))
 && qualifier.equals(Bytes.toString(CellUtil.cloneQualifier(c)))) {
 hbase1Value = Bytes.toString(CellUtil.cloneValue(c))
 }
 }

 val hbase2Value = Bytes.toString(resultData.getValue(columnFamily.getBytes, qualifier.getBytes))
 val put = new Put(iteratorArray(i)._2.getRow)

 //calculate result value
 val resultValue = hbase1Value.toInt + hbase2Value.toInt
 //set data to put
 put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes(qualifier),
 Bytes.toBytes(resultValue.toString))
 putList.add(put)
 }
}

if (putList.size() > 0) {
 table.put(putList)
}
} catch {
 case e: IOException =>
 e.printStackTrace();
} finally {
 if (table != null) {
 try {
 table.close()
 } catch {
 case e: IOException =>
 e.printStackTrace();
 }
 }
}
if (connection != null) {
 try {
 //Close the HBase connection.
 connection.close()
 } catch {
 case e: IOException =>
 e.printStackTrace()
 }
}
}
}
}

```

```
}
/**
 * Define serializer class.
 */
class MyRegistrar extends KryoRegistrar {
 override def registerClasses(kryo: Kryo) {
 kryo.register(classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable])
 kryo.register(classOf[org.apache.hadoop.hbase.client.Result])
 kryo.register(classOf[Array[(Any, Any)])
 kryo.register(classOf[Array[org.apache.hadoop.hbase.Cell]])
 kryo.register(classOf[org.apache.hadoop.hbase.NoTagsKeyValue])
 kryo.register(classOf[org.apache.hadoop.hbase.protobuf.generated.ClientProtos.RegionLoadStats])
 }
}
```

### 30.3.5.4 Python Example Code

#### Function

Use Spark to call an HBase API to operate HBase table1 and write the data analysis result of table1 to HBase table2.

#### Example Code

PySpark does not provide HBase related APIs. In this example, Python with the Java programming language invoked is used.

The following code snippets are used as an example. For complete codes, see [SparkHbaseToHbasePythonExample](#).

```
-*- coding:utf-8 -*-

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession

Create the SparkContext and set kryo serialization.
spark = SparkSession\
 .builder\
 .appName("SparkHbaseToHbase") \
 .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
 .config("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrar") \
 .getOrCreate()

Import the class that will run into sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.SparkHbaseToHbase')

Create a class instance and invoke the method.
spark._jvm.SparkHbaseToHbase().hbaseToHbase(spark._jsc)

Stop the SparkSession
spark.stop()
```

## 30.3.6 Reading Data from Hive and Write It to HBase

### 30.3.6.1 Overview

#### Scenarios

Assume that table **person** of Hive stores the user consumption amount of the current day and table2 of HBase stores the history consumption data.

In table person, name=1 and account=100 indicates that the consumption amount of user 1 in the current day is 100 CNY.

In table2, key=1 and cf:cid=1000 indicate that the history consumption amount of user 1 is 1000 CNY.

The Spark application shall achieve the following function:

Add the current consumption amount (100) to the history consumption amount (1000).

The running result is that the total consumption amount of user 1 (key=1) in table2 is 1100 CNY (cf:cid=1100).

## Data Preparation

Before develop the application, create the Hive table **person** and insert data to it. Create HBase table2.

**Step 1** Place the source log file to HDFS.

1. Create a blank file **log1.txt** in the local and write the following content to the file:  
1,100
2. Create a directory **/tmp/input** in HDFS and copy the **log1.txt** file to the directory.
  - a. On the HDFS client, run the following commands to obtain the security authentication:  
**cd /opt/hadoopclient**  
**kinit <Service user for authentication>**
  - b. On the Linux HDFS client, run the **hadoop fs -mkdir /tmp/input** command (a hdfs dfs command provides the same function.) to create a directory.
  - c. On the Linux HDFS client, run the **hadoop fs -put log1.txt /tmp/input** command to upload the data file.

**Step 2** Store the imported data to the Hive table.

Ensure that JDBCServer is started. Use the Beeline tool to create a Hive table and insert data to it.

1. Run the following commands to create the Hive table person:  
**create table person**  
**(**  
**name STRING,**  
**account INT**  
**)**  
**ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ESCAPED BY '\\'**  
**STORED AS TEXTFILE;**
2. Run the following command to insert data to the table:  
**load data inpath '/tmp/input/log1.txt' into table person;**

### Step 3 Create a HBase table:

Ensure that JDBCServer is started. Use the Spark-Beeline tool to create a HBase table and insert data to it.

1. Run the following commands to create the HBase table table2:

```
create table table2
(
 key string,
 cid string
)
using org.apache.spark.sql.hbase.HBaseSource
options(
 hbaseTableName "table2",
 keyCols "key",
 colsMapping "cid=cf.cid"
);
```

2. Run the following command on HBase to insert data to the table:

```
put 'table2', '1', 'cf:cid', '1000'
```

----End

## Development Idea

1. Query the data in Hive table person.
2. Query the data in table2 using the key value of table person.
3. Add the queried data.
4. Write the results of the preceding step to table2.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

1. Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.
2. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

 NOTE

- Before compilation and packaging, change the paths of the `user.keytab` and `krb5.conf` files in the sample code to the actual paths on the client server where the files are located. Example: `/opt/female/user.keytab` and `/opt/female/krb5.conf`.
  - Before running the sample program, set the `spark.yarn.security.credentials.hbase.enabled` configuration item to `true` in the `spark-defaults.conf` configuration file of Spark client. (The default value is `false`. Changing the value to `true` does not affect existing services.) If you want to uninstall the HBase service, change the value back to `false` first.
3. Upload the JAR file to any directory (for example, `/opt/female/`) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the `bin/spark-submit` script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

- Run Java or Scala sample code.  
**bin/spark-submit --class**  
*com.huawei.bigdata.spark.examples.SparkHivetoHbase* **--master yarn --**  
**deploy-mode client** */opt/female/SparkHivetoHbase-1.0.jar*
- Run the Python sample program.

 NOTE

- PySpark does not provide HBase-related APIs. Therefore, Python is used to invoke Java code in this sample. Use Maven to pack the provided Java code into a JAR file and place it in the same directory. When running the Python program, use `--jars` to load the JAR file to `classpath`.
- The Python sample code does not provide authentication information. Configure `--keytab` and `--principal` to specify authentication information.

```
bin/spark-submit --master yarn --deploy-mode client --keytab /opt/
FIclient/user.keytab --principal sparkuser --jars /opt/female/
SparkHivetoHbasePythonExample/SparkHivetoHbase-1.0.jar /opt/female/
SparkHivetoHbasePythonExample/SparkHivetoHbasePythonExample.py
```

### 30.3.6.2 Java Example Code

#### Function

In Spark application, call Hive API using Spark to operate Hive table, analyze the data, and then write the analyzed data to the HBase table.

#### Example Code

For details about code, see `com.huawei.bigdata.spark.examples.SparkHbasetoHbase`.

Example code

```
/**
 * calculate data from hive/hbase,then update to hbase
```

```
*/
public class SparkHivetoHbase {

 public static void main(String[] args) throws Exception {

 String userPrincipal = "sparkuser";
 String userKeytabPath = "/opt/FIclient/user.keytab";
 String krb5ConfPath = "/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf";
 Configuration hadoopConf = new Configuration();
 LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf);
 // Obtain the data in the table through the Spark interface.
 SparkConf conf = new SparkConf().setAppName("SparkHivetoHbase");
 JavaSparkContext jsc = new JavaSparkContext(conf);
 HiveContext sqlContext = new org.apache.spark.sql.hive.HiveContext(jsc);

 SparkSession spark = SparkSession
 .builder()
 .appName("SparkHivetoHbase")
 .config("spark.sql.warehouse.dir", "spark-warehouse")
 .enableHiveSupport()
 .getOrCreate();

 Dataset<Row>dataFrame = spark.sql("select name, account from person");

 // Traverse every Partition in the hive table and update the hbase table
 // If less data, you can use rdd.foreach()

 dataFrame.toJavaRDD().foreachPartition(
 new VoidFunction<Iterator<Row>>() {
 public void call(Iterator<Row> iterator) throws Exception {
 hBaseWriter(iterator);
 }
 }
);

 spark.stop();
 }

 /**
 * write to hbase table in exetutor
 *
 * @param iterator partition data from hive table
 */
 private static void hBaseWriter(Iterator<Row> iterator) throws IOException {
 // read hbase
 String tableName = "table2";
 String columnFamily = "cf";
 Configuration conf = HBaseConfiguration.create();

 Connection connection = null;
 Table table = null;
 try {
 connection = ConnectionFactory.createConnection(conf);
 table = connection.getTable(tableName);

 List<Row> table1List = new ArrayList<Row>();
 List<Get> rowList = new ArrayList<Get>();
 while (iterator.hasNext()) {
 Row item = iterator.next();
 Get get = new Get(item.getString(0).getBytes());
 table1List.add(item);
 rowList.add(get);
 }

 // get data from hbase table
 Result[] resultDataBuffer = table.get(rowList);

 // set data for hbase
 List<Put> putList = new ArrayList<Put>();
```

```
for (int i = 0; i < resultDataBuffer.length; i++) {
 // hbase row
 Result resultData = resultDataBuffer[i];
 if (!resultData.isEmpty()) {
 // get hiveValue
 int hiveValue = table1List.get(i).getInt(1);

 // get hbaseValue by column Family and colomn qualifier
 String hbaseValue = Bytes.toString(resultData.getValue(columnFamily.getBytes(), "cid".getBytes()));
 Put put = new Put(table1List.get(i).getString(0).getBytes());

 // calculate result value
 int resultValue = hiveValue + Integer.valueOf(hbaseValue);

 // set data to put
 put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes("cid"),
Bytes.toBytes(String.valueOf(resultValue)));
 putList.add(put);
 }
}

if (putList.size() > 0) {
 table.put(putList);
}
} catch (IOException e) {
 e.printStackTrace();
} finally {
 if (table != null) {
 try {
 table.close();
 } catch (IOException e) {
 e.printStackTrace();
 }
 }
}
if (connection != null) {
 try {
 // Close the HBase connection.
 connection.close();
 } catch (IOException e) {
 e.printStackTrace();
 }
}
}
}
```

### 30.3.6.3 Scala Example Code

#### Function

In Spark application, call Hive API using Spark to operate Hive table, analyze the data, and then write the analyzed data to the HBase table.

#### Example Code

For details about code, see [com.huawei.bigdata.spark.examples.SparkHbasetoHbase](#).

#### Example code

```
/**
 * calculate data from hive/hbase,then update to hbase
 */
object SparkHivetoHbase {

 case class FemaleInfo(name: String, gender: String, stayTime: Int)
```



```
def main(args: Array[String]) {

 String userPrincipal = "sparkuser";
 String userKeytabPath = "/opt/FIclient/user.keytab";
 String krb5ConfPath = "/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf";
 Configuration hadoopConf = new Configuration();
 LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf);
 // Obtain the data in the table through the Spark interface.

 val spark = SparkSession
 .builder()
 .appName("SparkHiveHbase")
 .config("spark.sql.warehouse.dir", "spaek-warehouse")
 .enableHiveSupport()
 .getOrCreate()

 import spark.implicits._
 val dataframe = spark.sql("select name, account from person")

 // Traverse every Partition in the hive table and update the hbase table
 // If less data, you can use rdd.foreach()
 dataframe.rdd.foreachPartition(x => hBaseWriter(x))

 spark.stop()
}

/**
 * write to hbase table in exetutor
 *
 * @param iterator partition data from hive table
 */
def hBaseWriter(iterator: Iterator[Row]): Unit = {
 // read hbase
 val tableName = "table2"
 val columnFamily = "cf"
 val conf = HBaseConfiguration.create()

 var table: Table = null
 var connection: Connection = null

 try {
 connection = ConnectionFactory.createConnection(conf)
 table = connection.getTable(tableName)

 val iteratorArray = iterator.toArray
 val rowList = new util.ArrayList[Get]()
 for (row <- iteratorArray) {
 val get = new Get(row.getString(0).getBytes)
 rowList.add(get)
 }

 // get data from hbase table
 val resultDataBuffer = table.get(rowList)

 // set data for hbase
 val putList = new util.ArrayList[Put]()
 for (i <- 0 until iteratorArray.size) {
 // hbase row
 val resultData = resultDataBuffer(i)
 if (!resultData.isEmpty) {
 // get hiveValue
 var hiveValue = iteratorArray(i).getInt(1)

 // get hbaseValue by column Family and colomn qualifier
 val hbaseValue = Bytes.toString(resultData.getValue(columnFamily.getBytes, "cid".getBytes))
 val put = new Put(iteratorArray(i).getString(0).getBytes)

 // calculate result value
```

```
 val resultValue = hiveValue + hbaseValue.toInt

 // set data to put
 put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes("cid"),
Bytes.toBytes(resultValue.toString))
 putList.add(put)
 }
}

if (putList.size() > 0) {
 table.put(putList)
}
} catch {
 case e: IOException =>
 e.printStackTrace();
} finally {
 if (table != null) {
 try {
 table.close()
 } catch {
 case e: IOException =>
 e.printStackTrace();
 }
 }
 if (connection != null) {
 try {
 //Close the HBase connection.
 connection.close()
 } catch {
 case e: IOException =>
 e.printStackTrace()
 }
 }
}
}
```

### 30.3.6.4 Python Example Code

#### Function

In a Spark application, use Spark to call a Hive API to operate a Hive table, and write the data analysis result of the Hive table to an HBase table.

#### Example Code

PySpark does not provide HBase related APIs. In this example, Python with the Java programming language invoked is used.

The following code snippets are used as an example. For complete codes, see [SparkHivetoHbasePythonExample](#).

```
-*- coding:utf-8 -*-

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession

Create the SparkSession
spark = SparkSession\
 .builder\
 .appName("SparkHivetoHbase") \
 .getOrCreate()

Import the class that will run into sc._jvm.
```

```
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.SparkHivetoHbase')

Create a class instance and invoke the method.
spark._jvm.SparkHivetoHbase().hivetohbase(spark._jsc)

Stop the SparkSession
spark.stop()
```

## 30.3.7 Streaming Connecting to Kafka0-10

### 30.3.7.1 Overview

#### Scenarios

Assume that the Kafka component receives one word record every 1 second.

The developed Spark application needs to achieve the following function:

Calculate the sum of records for each word in real time.

**log1.txt** example file:

```
LiuYang
YuanJing
GuoYijun
CaiXuyu
Liyuan
FangBo
LiuYang
YuanJing
GuoYijun
CaiXuyu
FangBo
```

#### Data Planning

Spark Streaming sample project data is stored in the Kafka component. A user with Kafka permission sends data to Kafka component.

1. Ensure that the cluster, including HDFS, Yarn, Spark, and Kafka is successfully installed.

2. Create the **input\_data1.txt** file in the local and copy the content of the **log1.txt** file to the **input\_data1.txt** file.

On the client installation node, create the **/home/data** directory and upload the **input\_data1.txt** file to the **/home/data** directory.

3. Change the value of **allow.everyone.if.no.acl.found**, the Broker configuration value of Kafka, to **true**.

4. Create a topic.

{zkQuorum} indicates ZooKeeper cluster information. The format is IP:port.

```
$KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper {zkQuorum}/
kafka --replication-factor 1 --partitions 3 --topic {Topic}
```

5. Start the Producer of Kafka and send data to Kafka.

```
java -cp {ClassPath}
com.huawei.bigdata.spark.examples.StreamingExampleProducer
{BrokerList} {Topic}
```

In this command, **ClassPath** must contain the absolute path of the Kafka JAR package on the Spark client in addition to the path of the sample JAR package, for example: `/opt/client/Spark2x/spark/jars/*:/opt/client/Spark2x/spark/jars/streamingClient010/*:{ClassPath}`.

## Development Approach

1. Receive data from Kafka and generate DStream.
2. Collect the statistics of word records by category.
3. Calculate and print the result.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.
- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

### NOTE

Before compilation and packaging, change the paths of the **user.keytab** and **krb5.conf** files in the sample code to the actual paths on the client server where the files are located. Example: `/opt/female/user.keytab` and `/opt/female/krb5.conf`.

- Upload the JAR package to any directory (for example, **/opt**) on the server where the Spark client is located.
- Prepare dependency packages and upload the following JAR packages to the **\$SPARK\_HOME/jars/streamingClient010** directory on the server where the Spark client is located.
  - `spark-streaming-kafkaWriter-0-10_2.12-3.1.1-hw-ei-311001.jar`
  - `kafka-clients-xxx.jar`
  - `spark-sql-kafka-0-10_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar`
  - `spark-streaming-kafka-0-10_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar`
  - `spark-token-provider-kafka-0-10_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar`

### NOTE

- For the dependency package whose version number contains "hw-ei", download from the Huawei open-source image site. For details, see [Configuring Huawei Open-Source Mirrors](#).
- For the dependency package whose version number does not contain "hw-ei", obtain them from the Maven central repository. For details, see [Obtaining Sample Projects](#).

## Running Tasks

When running the sample program, you need to specify **<checkpointDir>**, **<brokers>**, **<topic>**, and **<batchTime>**. **<checkpointDir>** indicates the path for storing the program result backup in HDFS. **<brokers>** indicates the Kafka address for obtaining metadata. **<topic>** indicates the topic name read from Kafka. **<batchTime>** indicates the interval for Streaming processing in batches.

### NOTE

The location of Spark Streaming Kafka dependency package on the client is different from the location of other dependency packages. For example, the path to the Spark Streaming Kafka dependency package is **\$SPARK\_HOME/jars/streamingClient010**, whereas the path to other dependency packages is **\$SPARK\_HOME/jars**. Therefore, when running an application, you need to add a configuration item to the **spark-submit** command to specify the path of the dependency package of Spark Streaming Kafka, for example, **--jars \$(files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar); IFS=,; echo "\${files[\*]}")**

Because the cluster is security mode, you need to add configuration items and modify the command parameters.

1. Add configuration items to **\$SPARK\_HOME/conf/jaas.conf**:

```
KafkaClient {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=false
 useTicketCache=true
 debug=false;
};
```

2. Add configuration items to **\$SPARK\_HOME/conf/jaas-zk.conf**:

```
KafkaClient {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 keyTab="/user.keytab"
 principal="sparkuser@<system domain name>"
 useTicketCache=false
 storeKey=true
 debug=true;
};
```

3. Use **--files** and relative path to submit the **keytab** file to ensure that the **keytab** file is loaded to the container of the executor.
4. For the port number in **<brokers>**, use **SASL\_PLAINTEXT** for the **Kafka 0-10 Write To Print** example, use **PLAINTEXT** for the **Write To Kafka 0-10** example.

Go to the Spark client directory and run the following commands to invoke the **bin/spark-submit** script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

- Sample code (**Spark Streaming read Kafka 0-10 Write To Print**)

```
bin/spark-submit --master yarn --deploy-mode client --files ./jaas.conf,./
user.keytab --jars $(files=($SPARK_HOME/jars/streamingClient010/*.jar);
IFS=,; echo "${files[*]}") --class
com.huawei.bigdata.spark.examples.SecurityKafkaWordCount /opt/
SparkStreamingKafka010JavaExample-1.0.jar <checkpointDir> <brokers>
<topic> <batchTime>
```

The configuration example is as follows:

```
--files ./jaas.conf,./user.keytab //Use --files to specify the jaas.conf and keytab files.
```

- Sample code (**Spark Streaming Write To Kafka 0-10**)

```
bin/spark-submit --master yarn --deploy-mode client --jars $
(files=($SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "$
{files[*]}") --class
```

```
com.huawei.bigdata.spark.examples.JavaDstreamKafkaWriter /opt/
SparkStreamingKafka010JavaExample-1.0.jar <groupId> <brokers> <topics>
```

### 30.3.7.2 Java Example Code

#### Function

In Spark applications, use Streaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word, and write the data to Kafka 0-10.

#### Example Code (Streaming Read Data from Kafka 0-10)

The following code is an example. For details, see `com.huawei.bigdata.spark.examples.SecurityKafkaWordCount`.

```
**
 * Consumes messages from one or more topics in Kafka.
 * <checkPointDir> is the Spark Streaming checkpoint directory.
 * <brokers> is for bootstrapping and the producer will only use it for getting metadata
 * <topics> is a list of one or more kafka topics to consume from
 * <batchTime> is the Spark Streaming batch duration in seconds.
 */
public class SecurityKafkaWordCount
{
 public static void main(String[] args) throws Exception {
 JavaStreamingContext ssc = createContext(args);

 //The Streaming system starts.
 ssc.start();
 try {
 ssc.awaitTermination();
 } catch (InterruptedException e) {
 }
 }

 private static JavaStreamingContext createContext(String[] args) throws Exception {
 String checkPointDir = args[0];
 String brokers = args[1];
 String topics = args[2];
 String batchTime = args[3];

 // Create a Streaming startup environment.
 SparkConf sparkConf = new SparkConf().setAppName("KafkaWordCount");
 JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, new
Duration(Long.parseLong(batchTime) * 1000));

 //Configure the CheckPoint directory for the Streaming.
 //This parameter is mandatory because of existence of the window concept.
 ssc.checkpoint(checkPointDir);

 // Get the list of topic used by kafka
 String[] topicArr = topics.split(",");
 Set<String> topicSet = new HashSet<String>(Arrays.asList(topicArr));
 Map<String, Object> kafkaParams = new HashMap();
 kafkaParams.put("bootstrap.servers", brokers);
 kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
 kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
 kafkaParams.put("group.id", "DemoConsumer");
 kafkaParams.put("security.protocol", "SASL_PLAINTEXT");
 kafkaParams.put("sas.l.kerberos.service.name", "kafka");
 kafkaParams.put("kerberos.domain.name", "hadoop.<system domain name>");

 LocationStrategy locationStrategy = LocationStrategies.PreferConsistent();
 ConsumerStrategy consumerStrategy = ConsumerStrategies.Subscribe(topicSet, kafkaParams);
```

```
// Create direct kafka stream with brokers and topics
// Receive data from the Kafka and generate the corresponding DStream
JavaInputDStream<ConsumerRecord<String, String>> messages = KafkaUtils.createDirectStream(ssc,
locationStrategy, consumerStrategy);

// Obtain field properties in each row.
JavaDStream<String> lines = messages.map(new Function<ConsumerRecord<String, String>, String>() {
 @Override
 public String call(ConsumerRecord<String, String> tuple2) throws Exception {
 return tuple2.value();
 }
});

// Aggregate the total time that calculate word count
JavaPairDStream<String, Integer> wordCounts = lines.mapToPair(
 new PairFunction<String, String, Integer>() {
 @Override
 public Tuple2<String, Integer> call(String s) {
 return new Tuple2<String, Integer>(s, 1);
 }
 }).reduceByKey(new Function2<Integer, Integer, Integer>() {
 @Override
 public Integer call(Integer i1, Integer i2) {
 return i1 + i2;
 }
}).updateStateByKey(
 new Function2<List<Integer>, Optional<Integer>, Optional<Integer>>() {
 @Override
 public Optional<Integer> call(List<Integer> values, Optional<Integer> state) {
 int out = 0;
 if (state.isPresent()) {
 out += state.get();
 }
 for (Integer v : values) {
 out += v;
 }
 return Optional.of(out);
 }
 });

// print the results
wordCounts.print();
return ssc;
}
```

## Example Code (Streaming Write To Kafka 0-10)

The following code segment is only an example. For details, see [com.huawei.bigdata.spark.examples.DstreamKafkaWriter](#).

### NOTE

It is advisable to use the new API `createDirectStream` instead of the old API `createStream` for application development. The old API continues to exist, but its performance and stability are worse than the new API.

```
/**
 * Parameter description:
 * <groupId> is the group ID for the consumer.
 * <brokers> is for bootstrapping and the producer will only use
 * <topic> is a kafka topic to consume from.
 */
public class JavaDstreamKafkaWriter {

 public static void main(String[] args) throws InterruptedException {
 if (args.length != 3) {
```

```
System.err.println("Usage: JavaDstreamKafkaWriter <groupId> <brokers> <topic>");
System.exit(1);
}

final String groupId = args[0];
final String brokers = args[1];
final String topic = args[2];

SparkConf sparkConf = new SparkConf().setAppName("KafkaWriter");

// Populate Kafka properties
Map<String, Object> kafkaParams = new HashMap<String, Object>();
kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
kafkaParams.put("value.serializer", "org.apache.kafka.common.serialization.ByteArraySerializer");
kafkaParams.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
kafkaParams.put("bootstrap.servers", brokers);
kafkaParams.put("group.id", groupId);
kafkaParams.put("auto.offset.reset", "smallest");

// Create Spark Java streaming context
JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.milliseconds(500));

// Populate data to write to kafka
List<String> sendData = new ArrayList();
sendData.add("kafka_writer_test_msg_01");
sendData.add("kafka_writer_test_msg_02");
sendData.add("kafka_writer_test_msg_03");

// Create Java RDD queue
Queue<JavaRDD<String>> sent = new LinkedList();
sent.add(ssc.sparkContext().parallelize(sendData));

// Create java Dstream with the data to be written
JavaDStream wStream = ssc.queueStream(sent);

// Write to kafka
JavaDStreamKafkaWriterFactory.fromJavaDStream(wStream).writeToKafka(
 JavaConverters.mapAsScalaMapConverter(kafkaParams).asScala(),
 new Function<String, ProducerRecord<String, byte[]>>() {
 public ProducerRecord<String, byte[]> call(String s) throws Exception {
 return new ProducerRecord(topic, s.toString().getBytes());
 }
 });

ssc.start();
ssc.awaitTermination();
}
```

### 30.3.7.3 Scala Example Code

#### Function

In Spark applications, use Streaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word, and write the data to Kafka 0-10.

#### Example Code (Streaming Read Data from Kafka 0-10)

The following code is an example. For details, see [com.huawei.bigdata.spark.examples.SecurityKafkaWordCount](#).

```
/**
 * Consumes messages from one or more topics in Kafka.
```



```
* <checkPointDir> is the Spark Streaming checkpoint directory.
* <brokers> is for bootstrapping and the producer will only use it for getting metadata
* <topics> is a list of one or more kafka topics to consume from
* <batchTime> is the Spark Streaming batch duration in seconds.
*/
object SecurityKafkaWordCount {

 def main(args: Array[String]) {
 val ssc = createContext(args)

 //The Streaming system starts.
 ssc.start()
 ssc.awaitTermination()
 }

 def createContext(args : Array[String]) : StreamingContext = {

 val Array(checkPointDir, brokers, topics, batchSize) = args

 // Create a Streaming startup environment.
 val sparkConf = new SparkConf().setAppName("KafkaWordCount")
 val ssc = new StreamingContext(sparkConf, Seconds(batchSize.toLong))

 //Configure the CheckPoint directory for the Streaming.
 //This parameter is mandatory because of existence of the window concept.
 ssc.checkpoint(checkPointDir)

 // Get the list of topic used by kafka
 val topicArr = topics.split(",")
 val topicSet = topicArr.toSet
 val kafkaParams = Map[String, String](
 "bootstrap.servers" -> brokers,
 "value.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
 "key.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
 "group.id" -> "DemoConsumer",
 "security.protocol" -> "SASL_PLAINTEXT",
 "sasl.kerberos.service.name" -> "kafka",
 "kerberos.domain.name" -> "hadoop.<system domain name>"
);

 val locationStrategy = LocationStrategies.PreferConsistent
 val consumerStrategy = ConsumerStrategies.Subscribe[String, String](topicSet, kafkaParams)

 // Create direct kafka stream with brokers and topics
 // Receive data from the Kafka and generate the corresponding DStream
 val stream = KafkaUtils.createDirectStream[String, String](ssc, locationStrategy, consumerStrategy)

 // Obtain field properties in each row.
 val tf = stream.transform (rdd =>
 rdd.map(r => (r.value, 1L))
)

 // Aggregate the total time that calculate word count
 val wordCounts = tf.reduceByKey(_ + _)
 val totalCounts = wordCounts.updateStateByKey(updataFunc)
 totalCounts.print()

 ssc
 }

 def updataFunc(values : Seq[Long], state : Option[Long]) : Option[Long] =
 Some(values.sum + state.getOrElse(0L))
}
```

## Example Code (Streaming Write To Kafka 0-10)

The following code segment is only an example. For details, see [com.huawei.bigdata.spark.examples.DstreamKafkaWriter](http://com.huawei.bigdata.spark.examples.DstreamKafkaWriter).

 NOTE

After updates to Spark, it is advisable to use the new API `createDirectStream` instead of the old API `createStream` for application development. The old API continues to exist, but its performance and stability are worse than the new API.

```
package com.huawei.bigdata.spark.examples

import scala.collection.mutable
import scala.language.postfixOps

import com.huawei.spark.streaming.kafka010.KafkaWriter_
import org.apache.kafka.clients.producer.ProducerRecord
import org.apache.spark.SparkConf
import org.apache.spark.rdd.RDD
import org.apache.spark.streaming.{Milliseconds, StreamingContext}

/**
 * Exaple code to demonstrate the usage of dstream.writeToKafka API
 *
 * Parameter description:
 * <groupId> is the group ID for the consumer.
 * <brokers> is for bootstrapping and the producer will only use
 * <topic> is a kafka topic to consume from.
 */
object DstreamKafkaWriter {
 def main(args: Array[String]) {

 if (args.length != 3) {
 System.err.println("Usage: DstreamKafkaWriter <groupId> <brokers> <topic>")
 System.exit(1)
 }

 val Array(groupId, brokers, topic) = args
 val sparkConf = new SparkConf().setAppName("KafkaWriter")

 // Populate Kafka properties
 val kafkaParams = Map[String, String](
 "bootstrap.servers" -> brokers,
 "value.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
 "key.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
 "value.serializer" -> "org.apache.kafka.common.serialization.ByteArraySerializer",
 "key.serializer" -> "org.apache.kafka.common.serialization.StringSerializer",
 "group.id" -> groupId,
 "auto.offset.reset" -> "latest"
)

 // Create Spark streaming context
 val ssc = new StreamingContext(sparkConf, Milliseconds(500));

 // Populate data to write to kafka
 val sendData = Seq("kafka_writer_test_msg_01", "kafka_writer_test_msg_02",
 "kafka_writer_test_msg_03")

 // Create RDD queue
 val sent = new mutable.Queue[RDD[String]]()
 sent.enqueue(ssc.sparkContext.makeRDD(sendData))

 // Create Dstream with the data to be written
 val wStream = ssc.queueStream(sent)

 // Write to kafka
 wStream.writeToKafka(kafkaParams,
 (x: String) => new ProducerRecord[String, Array[Byte]](topic, x.getBytes))

 ssc.start()
 ssc.awaitTermination()
 }
}
```

## 30.3.8 Structured Streaming Project

### 30.3.8.1 Overview

#### Scenarios

In Spark applications, use StructuredStreaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word.

#### Data Planning

Sample project data of StructuredStreaming is stored in Kafka. A user with Kafka permission sends data to Kafka.

1. Ensure that the cluster, including HDFS, Yarn, Spark, and Kafka is successfully installed.
2. Change the value of **allow.everyone.if.no.acl.found**, the Broker configuration value of Kafka, to **true**.
3. Create a topic.

zkQuorum} indicates ZooKeeper cluster information. The format is IP:port.

```
$KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper {zkQuorum}/
kafka --replication-factor 1 --partitions 1 --topic {Topic}
```

4. Start the Producer of Kafka and send data to Kafka.

{ClassPath} indicates the storage path of engineer JAR packages and is specified by the user. For details, see [Compiling and Running the Application](#).

```
java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/streamingClient010/*:
{ClassPath}
com.huawei.bigdata.spark.examples.KafkaWordCountProducer
{BrokerList} {Topic} {messagesPerSec} {wordsPerMessage}
```

#### Development Approach

1. Receive data from Kafka and generate DataStreamReader.
2. Collect the statistics of word records.
3. Calculate and print the result.

#### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

#### Packaging the Project

- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

 NOTE

Before compilation and packaging, change the paths of the `user.keytab` and `krb5.conf` files in the sample code to the actual paths on the client server where the files are located. Example: `/opt/female/user.keytab` and `/opt/female/krb5.conf`.

- Upload the JAR package to any directory (for example, `/opt`) on the server where the Spark client is located.
- Upload `commons-pool2-xxx.jar` to the `$SPARK_HOME/jars/streamingClient010/` directory. The JAR package can be obtained from the `$SPARK_HOME/tool/carbonPrequery` directory.

## Running Tasks

- When running the sample application, you need to specify `<brokers>`, `<subscribe-type>`, `<topic>`, `<protocol>`, `<service>`, `<domain>` and `<checkpointDir>`. `<brokers>` indicates the Kafka address (port **21007** is required) for obtaining metadata, `<subscribe-type>` indicates the Kafka subscription type (for example, `subscribe`), and `<topic>` indicates the name of the topic read from Kafka. `<protocol>` indicates the secure access protocol (for example, `SASL_PLAINTEXT`). `<service>` indicates the Kerberos service name (for example, `kafka`). `<domain>` indicates the Kerberos domain name (for example, `hadoop.<system domain name>`), `<checkpointDir>` indicates the path for storing checkpoint files.

 NOTE

The path of the Spark Structured Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is `$SPARK_HOME/jars`. Whereas the path of the Spark Structured Streaming Kafka dependency package is `$SPARK_HOME/jars/streamingClient010`. Therefore, when running an application, you need to add a configuration item to the `spark-submit` command to specify the path of the dependency package of Spark Streaming Kafka, for example, `--jars $ (files=($SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "${files[*]}")`

Because the cluster is security mode, you need to add configuration items and modify the command parameters.

1. Add configuration items to `$SPARK_HOME/conf/jaas.conf`:

```
KafkaClient {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=false
 useTicketCache=true
 debug=false;
};
```

2. Add configuration items to `$SPARK_HOME/conf/jaas-zk.conf`:

```
KafkaClient {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 keyTab="/user.keytab"
 principal="sparkuser@<system domain name>"
 useTicketCache=false
 storeKey=true
 debug=true;
};
```

3. Use `--files` and relative path to submit the `keytab` file to ensure that the `keytab` file is loaded to the container of the executor.

 CAUTION

When submitting a structured stream task, you need to run the `--jars` command to specify the path of the Kafka-related JAR file. For the current version, you need to copy the `kafka-clients` jar file from the `$SPARK_HOME/jars/streamingClient010` directory to the `$SPARK_HOME/jars` directory. Otherwise, the "class not found" error is reported.

Go to the Spark client directory and run the following commands to invoke the `bin/spark-submit` script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

- Run Java or Scala sample code:

```
bin/spark-submit --master yarn --deploy-mode client --files <local Path>/jaas.conf,<local path>/user.keytab --jars $(files=($SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "${files[*]}") --class com.huawei.bigdata.spark.examples.SecurityKafkaWordCount /opt/SparkStructuredStreamingScalaExample-1.0.jar <brokers> <subscribe-type> <topic> <protocol> <service> <domain> <checkpointDir>
```

The configuration example is as follows:

```
-files <local Path>/jaas.conf,<local Path>/user.keytab //Use --files to specify the jaas.conf and keytab files.
```

- Run the Python sample code:

 NOTE

When running the Python sample code, you need to add the JAR package of the Java project to the `streamingClient010/` directory.

```
bin/spark-submit --master yarn --deploy-mode client --files /opt/FIClient/user.keytab --jars $(files=($SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "${files[*]}") /opt/female/SparkStructuredStreamingPythonExample/SecurityKafkaWordCount.py <brokers> <subscribe-type> <topic> <protocol> <service> <domain> <checkpointDir>
```

### 30.3.8.2 Java Sample Code

#### Function

In Spark applications, use `StructuredStreaming` to invoke Kafka interface to obtain word records. Collect the statistics of records for each word.

#### Code Sample

The following code is an example. For details, see `com.huawei.bigdata.spark.examples.SecurityKafkaWordCount`.

 NOTE

When new data is available in `Streaming DataFrame/Dataset`, `outputMode` is used for configuring data written to the `Streaming receiver`.

```
public class SecurityKafkaWordCount
{
 public static void main(String[] args) throws Exception {
```

```
if (args.length < 6) {
 System.err.println("Usage: SecurityKafkaWordCount <bootstrap-servers> " +
 "<subscribe-type> <topics> <protocol> <service> <domain>");
 System.exit(1);
}

String bootstrapServers = args[0];
String subscribeType = args[1];
String topics = args[2];
String protocol = args[3];
String service = args[4];
String domain = args[5];

SparkSession spark = SparkSession
 .builder()
 .appName("SecurityKafkaWordCount")
 .getOrCreate();

// Create DataSet representing the stream of input lines from kafka
Dataset<String> lines = spark
 .readStream()
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option(subscribeType, topics)
 .option("kafka.security.protocol", protocol)
 .option("kafka.sasl.kerberos.service.name", service)
 .option("kafka.kerberos.domain.name", domain)
 .load()
 .selectExpr("CAST(value AS STRING)")
 .as(Encoders.STRING());

// Generate running word count
Dataset<Row> wordCounts = lines.flatMap(new FlatMapFunction<String, String>() {
 @Override
 public Iterator<String> call(String x) {
 return Arrays.asList(x.split(" ")).iterator();
 }
}, Encoders.STRING()).groupBy("value").count();

// Start running the query that prints the running counts to the console
StreamingQuery query = wordCounts.writeStream()
 .outputMode("complete")
 .format("console")
 .start();

query.awaitTermination();
}
```

### 30.3.8.3 Scala Sample Code

#### Function

In Spark applications, use StructuredStreaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word.

#### Code Sample

The following code is an example. For details, see `com.huawei.bigdata.spark.examples.SecurityKafkaWordCount`.

 NOTE

When new data is available in Streaming DataFrame/Dataset, **outputMode** is used for configuring data written to the Streaming receiver.

```
object SecurityKafkaWordCount {
 def main(args: Array[String]): Unit = {
 if (args.length < 6) {
 System.err.println("Usage: SecurityKafkaWordCount <bootstrap-servers> " +
 "<subscribe-type> <topics> <protocol> <service> <domain>")
 System.exit(1)
 }

 val Array(bootstrapServers, subscribeType, topics, protocol, service, domain) = args

 val spark = SparkSession
 .builder
 .appName("SecurityKafkaWordCount")
 .getOrCreate()

 import spark.implicits._

 // Create DataSet representing the stream of input lines from kafka
 val lines = spark
 .readStream
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option(subscribeType, topics)
 .option("kafka.security.protocol", protocol)
 .option("kafka.sasl.kerberos.service.name", service)
 .option("kafka.kerberos.domain.name", domain)
 .load()
 .selectExpr("CAST(value AS STRING)")
 .as[String]

 // Generate running word count
 val wordCounts = lines.flatMap(_._2.split(" ")).groupBy("value").count()

 // Start running the query that prints the running counts to the console
 val query = wordCounts.writeStream
 .outputMode("complete")
 .format("console")
 .start()

 query.awaitTermination()
 }
}
```

### 30.3.8.4 Python Sample Code

#### Function

In Spark applications, use StructuredStreaming to invoke Kafka APIs to obtain word records. Classify word records to obtain the number of records of each word.

#### Sample Code

The following code segment is only an example. For details, see SecurityKafkaWordCount

 NOTE

When there is new available data in Streaming DataFrame/Dataset, **outputMode** indicates data written to the Streaming receiver.

```
#!/usr/bin/python
-*- coding: utf-8 -*-

import sys
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, split

if __name__ == "__main__":
 if len(sys.argv) < 6:
 print("Usage: <bootstrapServers> <subscribeType> <topics> <protocol> <service> <domain>")
 exit(-1)

 bootstrapServers = sys.argv[1]
 subscribeType = sys.argv[2]
 topics = sys.argv[3]
 protocol = sys.argv[4]
 service = sys.argv[5]
 domain = sys.argv[6]

 # Initialize the SparkSession.
 spark = SparkSession.builder.appName("SecurityKafkaWordCount").getOrCreate()

 # Create the DataFrame of input lines stream from Kafka.
 # In security mode, set spark/conf/jaas.conf and jaas-zk.conf to KafkaClient.
 lines = spark.readStream.format("kafka")\
 .option("kafka.bootstrap.servers", bootstrapServers)\
 .option(subscribeType, topics)\
 .option("kafka.security.protocol", protocol)\
 .option("kafka.sasl.kerberos.service.name", service)\
 .option("kafka.kerberos.domain.name", domain)\
 .load()\
 .selectExpr("CAST(value AS STRING)")

 # Split lines into words.
 words = lines.select(explode(split(lines.value, " ")).alias("word"))
 # Generate the running word count.
 wordCounts = words.groupBy("word").count()

 # Start to query whether the running counts are printed to the console.
 query = wordCounts.writeStream\
 .outputMode("complete")\
 .format("console")\
 .start()

 query.awaitTermination()
```

## 30.3.9 Structured Streaming Stream-Stream Join

### 30.3.9.1 Overview

#### Scenarios

An advertisement service involves advertisement request events, advertisement display events, and advertisement click events. The advertiser needs to collect statistics on valid advertisement displays and advertisement click data.

Known conditions are as follows:

1. Each time a user requests an advertisement, an advertisement request event is generated and saved to the adRequest topic of Kafka.
2. After an advertisement is requested, the advertisement may be displayed for multiple times. Each time the advertisement is displayed, an advertisement display event is generated and saved to the adShow topic of Kafka.



3. Each advertisement may be clicked for multiple times. Each time it is clicked, an advertisement click event is generated and saved to the adClick topic of Kafka.
4. For advertisement display:
  - a. If the duration from the time when a request is generated to the time when the advertisement is displayed exceeds A minutes, the display is invalid.
  - b. Advertisement display events generated during A minutes are valid events.
5. For advertisement click:
  - a. If the duration from the display event to the click event exceeds B minutes, the click is invalid.
  - b. If there are multiple click events within B minutes, only the first click event is valid.

The simple data structure in this scenario is as follows:

- Advertisement request event  
Data structure: adID^reqTime
- Advertisement display event  
Data structure: adID^showID^showTime
- Advertisement click event  
Data structure: adID^showID^clickTime

Data association relationships are as follows:

- The advertisement request event is associated with the advertisement display event through the adID.
- The advertisement display event is associated with the advertisement click event through the adID and showID.

Data requirements:

- The delay from the time when data is generated to the time when the data reaches the stream processing engine does not exceed two hours.
- The time for advertisement request events, advertisement display events, and advertisement click events reach the stream processing engine may not be in sequence or be aligned with each other.

## Data Planning

1. Enable users with the Kafka permission to generate simulation data in Kafka.  

```
java -cp $SPARK_HOME/conf:$SPARK_HOME/jars/*:$SPARK_HOME/jars/
streamingClient010/*:{ClassPath}
com.huawei.bigdata.spark.examples.KafkaADEventProducer {BrokerList}
{timeOfProduceReqEvent} {eventTimeBeforeCurrentTime} {reqTopic}
{reqEventCount} {showTopic} {showEventMaxDelay} {clickTopic}
{clickEventMaxDelay}
```

 NOTE

- Ensure that the clusters are installed, including HDFS, Yarn, Spark2x, and Kafka.
- Set the **allow.everyone.if.no.acl.found** parameter of Kafka Broker to **true**.
- Start Kafka Producer and enable it to send data to Kafka.
- **{ClassPath}** indicates the path for storing the JAR package of the project. The path is specified by users. For details, see the step of exporting JAR packages in [Compiling and Running the Application](#).

Command example:

```
java -cp /opt/client/Spark2x/spark/conf:/opt/StructuredStreamingADScalaExample-1.0.jar:/opt/client/Spark2x/spark/jars/*:/opt/client/Spark2x/spark/jars/streamingClient010/* com.huawei.bigdata.spark.examples.KafkaADEventProducer 10.132.190.170:21005,10.132.190.165:21005 2h 1h req 10000000 show 5m click 5m
```

This command creates three topics on Kafka: req, show, and click. Ten million data records of request events are generated in two hours. The time range of the request events is from one hour ahead of the current time to the current time, and up to five display events are randomly generated for each request event. The time range of the display events is from the request event time to five minutes after the request event time. Up to five click events are randomly generated for each display event. The time range of the click events is from the display event time to five minutes after the display event time.

## Development Guideline

1. Use structured streaming to receive data from Kafka and generate request flows, display flows, and click flows.
2. Perform join query of data in request flows, display flows, and click flows.
3. Write the statistics result to Kafka.
4. Monitor the flow processing task status in the application.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.
- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

 NOTE

Before compilation and packaging, change the paths of the **user.keytab** and **krb5.conf** files in the sample code to the actual paths on the client server where the files are located. Example: **/opt/female/user.keytab** or **/opt/female/krb5.conf**.

- Upload the JAR package to any directory (for example, **/opt**) on the server where the Spark client is located.

## Running Tasks

When running the sample program, you need to specify **<kafkaBootstrapServers>**, **<maxEventDelay>**, **<reqTopic>**, **<showTopic>**, **<maxShowDelay>**, **<clickTopic>**, **<maxClickDelay>**, **<triggerInterver>**, **<checkpointDir>**, **<kafkaProtocol>**, **<kafkaService>**, and **<kafkaDomain>**. **<kafkaBootstrapServers>** indicates the Kafka address for obtaining metadata (port 21007 is required), and **<maxEventDelay>** indicates the maximum delay from data generation to stream processing. **<reqTopic>** indicates the topic name of the request event. **<showTopic>** indicates the topic name of the display event. **<maxShowDelay>** indicates the maximum delay for displaying the event, and **<clickTopic>** indicates the topic name of the click event. **<maxClickDelay>** indicates the maximum delay time of a valid click event. **<triggerInterver>** indicates the interval for triggering a stream processing task. **<checkpointDir>** indicates the path for storing the checkpoint file, and **<kafkaProtocol>** indicates the secure access protocol (for example, **SASL\_PLAINTEXT**). **<kafkaService>** indicates the Kerberos service name (for example, **kafka**), and **<kafkaDomain>** indicates the Kerberos domain name (for example, **hadoop.<system domain name>**).

### NOTE

The path of the Spark Structured Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is **\$SPARK\_HOME/jars**. Whereas the path of the Spark Structured Streaming Kafka dependency package is **\$SPARK\_HOME/jars/streamingClient010**. Therefore, when running an application, you need to add a configuration item to the **spark-submit** command to specify the path of the dependency package of Spark Streaming Kafka, for example, **--jars \$(files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar); IFS=,; echo "\${files[\*]}")**

Because the cluster is security mode, you need to add configuration items and modify the command parameters.

1. Add configuration items to **\$SPARK\_HOME/conf/jaas.conf**:

```
KafkaClient {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=false
 useTicketCache=true
 debug=false;
};
```

2. Add configuration items to **\$SPARK\_HOME/conf/jaas-zk.conf**

```
KafkaClient {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 keyTab="/user.keytab"
 principal="sparkuser@<System domain name>"
 useTicketCache=false
 storeKey=true
 debug=true;
};
```

3. Use **--files** and relative path to submit the **keytab** file to ensure that the **keytab** file is loaded to the container of the executor.

**CAUTION**

When submitting a structured stream task, you need to run the `--jars` command to specify the path of the Kafka-related JAR file. For the current version, you need to copy the `kafka-clients.jar` file from the `$SPARK_HOME/jars/streamingClient010` directory to the `$SPARK_HOME/jars` directory. Otherwise, the "class not found" error is reported.

Go to the Spark client directory and run the following command to invoke the `bin/spark-submit` script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

```
bin/spark-submit --master yarn --deploy-mode client --files <local Path>/jaas.conf,<local path>/user.keytab --jars $(files=($SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "${files[*]}") --conf "spark.sql.streaming.statefulOperator.checkCorrectness.enabled=false" --class com.huawei.bigdata.spark.examples.KafkaADCount /opt/StructuredStreamingADScalaExample-1.0.jar <kafkaBootstrapServers> <maxEventDelay> <reqTopic> <showTopic> <maxShowDelay> <clickTopic> <maxClickDelay> <triggerInterver> <checkpointDir> <kafkaProtocol> <kafkaService> <kafkaDomain>
```

Go to the Spark client directory and run the following command to invoke the `bin/spark-submit` script to run the code:

```
--files ./jaas.conf,./user.keytab //Use --files to specify the jaas.conf and keytab files.
```

### 30.3.9.2 Scala Example Code

#### Function

Structured Streaming is used to read advertisement request data, display data, and click data from Kafka, obtain effective display statistics and click statistics in real time, and write the statistics to Kafka.

#### Example code

The following code snippets are used as an example. For complete codes, see `com.huawei.bigdata.spark.examples.KafkaADCount`.

```
/**
 * Run the Structured Streaming task to collect statistics on valid advertisement display and click data. The
 * result is written into Kafka.
 */
object KafkaADCount {
 def main(args: Array[String]): Unit = {
 if (args.length < 12) {
 System.err.println("Usage: KafkaWordCount <bootstrap-servers> " +
 "<maxEventDelay> <reqTopic> <showTopic> <maxShowDelay> " +
 "<clickTopic> <maxClickDelay> <triggerInterver> " +
 "<checkpointLocation> <protocol> <service> <domain>")
 System.exit(1)
 }
 }

 val Array(bootstrapServers, maxEventDelay, reqTopic, showTopic,
 maxShowDelay, clickTopic, maxClickDelay, triggerInterver, checkpointLocation,
 protocol, service, domain) = args
}
```

```

val maxEventDelayMills = JavaUtils.timeStringAs(maxEventDelay, TimeUnit.MILLISECONDS)
val maxShowDelayMills = JavaUtils.timeStringAs(maxShowDelay, TimeUnit.MILLISECONDS)
val maxClickDelayMills = JavaUtils.timeStringAs(maxClickDelay, TimeUnit.MILLISECONDS)
val triggerMills = JavaUtils.timeStringAs(triggerInterval, TimeUnit.MILLISECONDS)

val spark = SparkSession
 .builder
 .appName("KafkaADCount")
 .getOrCreate()

spark.conf.set("spark.sql.streaming.checkpointLocation", checkpointLocation)

import spark.implicits._

// Create DataSet representing the stream of input lines from kafka
val reqDf = spark
 .readStream
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option("kafka.security.protocol", protocol)
 .option("kafka.sasl.kerberos.service.name", service)
 .option("kafka.kerberos.domain.name", domain)
 .option("subscribe", reqTopic)
 .load()
 .selectExpr("CAST(value AS STRING)")
 .as[String]
 .map{
 _._split('^') match {
 case Array(reqAdID, reqTime) => ReqEvent(reqAdID,
 Timestamp.valueOf(reqTime))
 }
 }
 .as[ReqEvent]
 .withWatermark("reqTime", maxEventDelayMills +
 maxShowDelayMills + " millisecond")

val showDf = spark
 .readStream
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option("kafka.security.protocol", protocol)
 .option("kafka.sasl.kerberos.service.name", service)
 .option("kafka.kerberos.domain.name", domain)
 .option("subscribe", showTopic)
 .load()
 .selectExpr("CAST(value AS STRING)")
 .as[String]
 .map{
 _._split('^') match {
 case Array(showAdID, showID, showTime) => ShowEvent(showAdID,
 showID, Timestamp.valueOf(showTime))
 }
 }
 .as[ShowEvent]
 .withWatermark("showTime", maxEventDelayMills +
 maxShowDelayMills + maxClickDelayMills + " millisecond")

val clickDf = spark
 .readStream
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option("kafka.security.protocol", protocol)
 .option("kafka.sasl.kerberos.service.name", service)
 .option("kafka.kerberos.domain.name", domain)
 .option("subscribe", clickTopic)
 .load()
 .selectExpr("CAST(value AS STRING)")
 .as[String]

```

```
.map{
 _split('^') match {
 case Array(clickAdID, clickShowID, clickTime) => ClickEvent(clickAdID,
 clickShowID, Timestamp.valueOf(clickTime))
 }
}
.as[ClickEvent]
.withWatermark("clickTime", maxEventDelayMills + " millisecond")

val showStaticsQuery = reqDf.join(showDf,
 expr(s"""
 reqAdID = showAdID
 AND showTime >= reqTime + interval ${maxShowDelayMills} millisecond
 """))
.selectExpr("concat_ws('^', showAdID, showID, showTime) as value")
.writeStream
.queryName("showEventStatics")
.outputMode("append")
.trigger(Trigger.ProcessingTime(triggerMills.millis))
.format("kafka")
.option("kafka.bootstrap.servers", bootstrapServers)
.option("kafka.security.protocol", protocol)
.option("kafka.sasl.kerberos.service.name", service)
.option("kafka.kerberos.domain.name", domain)
.option("topic", "showEventStatics")
.start()

val clickStaticsQuery = showDf.join(clickDf,
 expr(s"""
 showAdID = clickAdID AND
 showID = clickShowID AND
 clickTime >= showTime + interval ${maxClickDelayMills} millisecond
 """), joinType = "rightouter")
.dropDuplicates("showAdID")
.selectExpr("concat_ws('^', clickAdID, clickShowID, clickTime) as value")
.writeStream
.queryName("clickEventStatics")
.outputMode("append")
.trigger(Trigger.ProcessingTime(triggerMills.millis))
.format("kafka")
.option("kafka.bootstrap.servers", bootstrapServers)
.option("kafka.security.protocol", protocol)
.option("kafka.sasl.kerberos.service.name", service)
.option("kafka.kerberos.domain.name", domain)
.option("topic", "clickEventStatics")
.start()

new Thread(new Runnable {
 override def run(): Unit = {
 while(true) {
 println("-----get showStatic progress-----")
 //println(showStaticsQuery.lastProgress)
 println(showStaticsQuery.status)
 println("-----get clickStatic progress-----")
 //println(clickStaticsQuery.lastProgress)
 println(clickStaticsQuery.status)
 Thread.sleep(10000)
 }
 }
}).start

spark.streams.awaitAnyTermination()
}
```

### 30.3.10 Structured Streaming Status Operation

### 30.3.10.1 Overview

#### Scenarios

Assume that you need to collect statistics on the number of events in each session and the start and end timestamp of the sessions.

You need to export the sessions that are in the updated state in this batch.

#### Data Planning

1. Generate simulated data in Kafka (the Kafka permission is required).
2. Ensure that the cluster has been installed, including the HDFS, Yarn, Spark2x, and Kafka services.
3. Set the **allow.everyone.if.no.acl.found** parameter of Kafka Broker to **true**.
4. Create a topic.

**{zkQuorum}** indicates ZooKeeper cluster information in the *IP address.Port number* format.

```
$KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper {zkQuorum}/
kafka --replication-factor 1 --partitions 1 --topic {Topic}
```

5. Start the Producer of Kafka and send data to Kafka.

*{ClassPath}* indicates the storage path of the project JAR package that is specified by the user. For details, see [Compiling and Running the Application](#).

```
java -cp $SPARK_HOME/conf:$SPARK_HOME/jars/*:$SPARK_HOME/jars/
streamingClient010/*:{ClassPath}
com.huawei.bigdata.spark.examples.KafkaProducer {brokerlist} {topic}
{number of events produce every 0.02s}
```

Example:

```
java -cp /opt/client/Spark2x/spark/conf:/opt/
StructuredStreamingState-1.0.jar:/opt/client/Spark2x/spark/jars/*:/opt/
client/Spark2x/spark/jars/streamingClient010/*
com.huawei.bigdata.spark.examples.KafkaProducer xxx.xxx.xxx.xxx:
21005,xxx.xxx.xxx.xxx:21005,xxx.xxx.xxx.xxx:21005 mytopic 10
```

#### Development Guideline

1. Receive data from Kafka and generate the corresponding `DataStreamReader`.
2. Collect statistics by category.
3. Calculate the result and print it.

#### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, `/opt`) on the server where the Spark client is located.
- Upload the `user.keytab` and `krb5.conf` files to the server where the client is installed. (The file upload path must be the same as the path of the generated JAR file).

## Running Tasks

When running the example program, you need to specify `<brokers>`, `<subscribe-type>`, `<kafkaProtocol>`, `<kafkaService>`, `<kafkaDomain>`, `<topic>`, and `<checkpointLocation>`, where `<brokers>` indicates the Kafka address for obtaining metadata (port 21007 is required). `<subscribe-type>` indicates the Kafka consumption mode, and `<kafkaProtocol>` indicates the secure access protocol (such as `SASL_PLAINTEXT`). `<kafkaService>` indicates the Kerberos service name (for example, `kafka`). `<kafkaDomain>` indicates the Kerberos domain name (for example, `hadoop.<system domain name>`). `<topic>` indicates the Kafka topic to be consumed, and `<checkpointLocation>` indicates the path for storing the checkpoint of the Spark task.

### NOTE

The path of the Spark Structured Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is `$SPARK_HOME/jars`. Whereas the path of the Spark Streaming Structured Kafka dependency package is `$SPARK_HOME/jars/streamingClient010`. Therefore, when running an application, you need to add a configuration item to the `spark-submit` command to specify the path of the dependency package of Spark Streaming Kafka, for example, `--jars $(files=($SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "${files[*]}")`

Because the cluster is security mode, you need to add configuration items and modify the command parameters.

1. Add configuration items to `$SPARK_HOME/conf/jaas.conf`:

```
KafkaClient {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=false
 useTicketCache=true
 debug=false;
};
```

2. Add configuration items to `$SPARK_HOME/conf/jaas-zk.conf`:

```
KafkaClient {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 keyTab="/user.keytab"
 principal="sparkuser@<system domain name>"
 useTicketCache=false
 storeKey=true
 debug=true;
};
```

3. Use `--files` and relative path to submit the `keytab` file to ensure that the `keytab` file is loaded to the container of the executor.

Go to the Spark client directory and run the following command to invoke the `bin/spark-submit` script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):



- **bin/spark-submit --master yarn --deploy-mode client --files <local Path>/jaas.conf,<local path>/user.keytab --jars \$(files=(\$SPARK\_HOME/jars/streamingClient010/\*jar); IFS=,; echo "\${files[\*]}") --class com.huawei.bigdata.spark.examples.kafkaSessionization /opt/StructuredStreamingState-1.0.jar <brokers> <subscribe-type> <kafkaProtocol> <kafkaService> <kafkaDomain> <topic> <checkpointLocation>**

The configuration example is as follows:

```
--files ./jaas.conf,./user.keytab //Use --files to specify the jaas.conf and keytab files.
```

### CAUTION

When submitting a structured stream task, you need to run the **--jars** command to specify the path of the Kafka-related JAR file. For the current version, you need to copy the `kafka-clients.jar` file from the `$SPARK_HOME/jars/streamingClient010` directory to the `$SPARK_HOME/jars` directory. Otherwise, the "class not found" error is reported.

## 30.3.10.2 Scala Sample Code

### Function

In the Spark structure flow application, the number of events in each session and the start and end timestamp of the sessions are collected in different batches. At the same time, the system exports the sessions that are in the updated state in this batch.

### Code Example

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.kafkaSessionization**.

#### NOTE

When new data is available in Streaming DataFrame/Dataset, **outputMode** is used for configuring data written to the Streaming receiver. .

```
object kafkaSessionization {
 def main(args: Array[String]): Unit = {
 if (args.length < 7) {
 System.err.println("Usage: kafkaSessionization <bootstrap-servers> " +
 "<subscribe-type> <protocol> <service> <domain> <topics> <checkpointLocation>")
 System.exit(1)
 }

 val Array(bootstrapServers, subscribeType, protocol, service, domain, topics, checkpointLocation) = args

 val spark = SparkSession
 .builder
 .appName("kafkaSessionization")
 .getOrCreate()

 spark.conf.set("spark.sql.streaming.checkpointLocation", checkpointLocation)

 spark.streams.addListener(new StreamingQueryListener {

 @volatile private var startTime: Long = 0L
 @volatile private var endTime: Long = 0L
 })
 }
}
```

```
@volatile private var numRecs: Long = 0L

override def onQueryStarted(event: StreamingQueryListener.QueryStartedEvent): Unit = {
 println("Query started: " + event.id)
 startTime = System.currentTimeMillis
}

override def onQueryProgress(event: StreamingQueryListener.QueryProgressEvent): Unit = {
 println("Query made progress: " + event.progress)
 numRecs += event.progress.numInputRows
}

override def onQueryTerminated(event: StreamingQueryListener.QueryTerminatedEvent): Unit = {
 println("Query terminated: " + event.id)
 endTime = System.currentTimeMillis
}
})

import spark.implicits._

val df = spark
 .readStream
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option("kafka.security.protocol", protocol)
 .option("kafka.sasl.kerberos.service.name", service)
 .option("kafka.kerberos.domain.name", domain)
 .option(subscribeType, topics)
 .load()
 .selectExpr("CAST(value AS STRING)")
 .as[String]
 .map { x =>
 val splitStr = x.split(",")
 (splitStr(0), Timestamp.valueOf(splitStr(1)))
 }.as[(String, Timestamp)].flatMap { case (line, timestamp) =>
 line.split(" ").map(word => Event(sessionId = word, timestamp))
 }

// Sessionize the events. Track number of events, start and end timestamps of session, and
// and report session updates.
val sessionUpdates = df
 .groupByKey(event => event.sessionId)
 .mapGroupsWithState[SessionInfo, SessionUpdate](GroupStateTimeout.ProcessingTimeTimeout) {

 case (sessionId: String, events: Iterator[Event], state: GroupState[SessionInfo]) =>

 // If timed out, then remove session and send final update
 if (state.hasTimedOut) {
 val finalUpdate =
 SessionUpdate(sessionId, state.get.durationMs, state.get.numEvents, expired = true)
 state.remove()
 finalUpdate
 } else {
 // Update start and end timestamps in session
 val timestamps = events.map(_._2.getTime).toSeq
 val updatedSession = if (state.exists) {
 val oldSession = state.get
 SessionInfo(
 oldSession.numEvents + timestamps.size,
 oldSession.startTimestampMs,
 math.max(oldSession.endTimestampMs, timestamps.max))
 } else {
 SessionInfo(timestamps.size, timestamps.min, timestamps.max)
 }
 state.update(updatedSession)

 // Set timeout such that the session will be expired if no data received for 10 seconds
 state.setTimeoutDuration("10 seconds")
 SessionUpdate(sessionId, state.get.durationMs, state.get.numEvents, expired = false)
 }
 }
}
```

```
 }
 }
 // Start running the query that prints the session updates to the console
 val query = sessionUpdates
 .writeStream
 .outputMode("update")
 .format("console")
 .start()

 query.awaitTermination()
}
```

## 30.3.11 Concurrent Access from Spark to HBase in Two Clusters

### 30.3.11.1 Overview

#### Scenarios

Spark can access HBase in two clusters concurrently on condition that mutual trust has been configured between these two clusters.

#### Data Planning

1. Configure the IP addresses and host names of all ZooKeeper and HBase nodes in **cluster2** to the `/etc/hosts` file on the client node of **cluster1**.
2. In **cluster1** and **cluster2**, find the `hbase-site.xml` file in the `conf` directory of the Spark2x client, and save it to the `/opt/example/A` and `/opt/example/B` directories.
3. Run the following `spark-submit` command:

#### NOTE

Before running the sample program, set the `spark.yarn.security.credentials.hbase.enabled` configuration item to `true` in the `spark-defaults.conf` configuration file of Spark client. (The default value is `false`. Changing the value to `true` does not affect existing services.) If you want to uninstall the HBase service, change the value back to `false` first.

```
spark-submit --master yarn --deploy-mode client --files /opt/example/B/hbase-site.xml --keytab /opt/
FIClient/user.keytab --principal sparkuser --class com.huawei.spark.examples.SparkOnMultiHbase /opt/
example/SparkOnMultiHbase-1.0.jar
```

#### Development Approach

1. When accessing HBase, you need to use the configuration file of the corresponding cluster to create a **Configuration** object for creating a **Connection** object.
2. Use the **Connection** object you create to perform operations on the HBase table, such as creating a table and inserting, viewing, and printing data.

### 30.3.11.2 Scala Sample Code

The following code snippets are used as an example. For complete codes, see `com.huawei.spark.examples.SparkOnMultiHbase`

```
def main(args: Array[String]): Unit = {
 val conf = new SparkConf().setAppName("SparkOnMultiHbaseExample")
 conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
 conf.set("spark.kryo.registrator", "com.huawei.spark.examples.MyRegistrator")
 val sc = new SparkContext(conf)val tableName = "SparkOnMultiHbase"
 val clusterFlagList=List("B", "A")
 clusterFlagList.foreach{ item =>
 val hbaseConf = getConf(item)
 println(hbaseConf.get("hbase.zookeeper.quorum"))
 val hbaseUtil = new HbaseUtil(sc,hbaseConf)
 hbaseUtil.writeToHbase(tableName)
 hbaseUtil.readFromHbase(tableName)
 }
 sc.stop()
}
private def getConf(item:String):Configuration={
 val conf: Configuration = HBaseConfiguration.create()
 val url = "/opt" + File.separator + "example" + File.separator + item + File.separator + "hbase-site.xml"
 conf.addResource(new File(url).toURI.toURL)
 conf
}
```

## 30.3.12 Synchronizing HBase Data from Spark to CarbonData

### 30.3.12.1 Overview

#### Scenarios

Data is written to HBase in real time for point query services and is synchronized to CarbonData tables in batches at a specified interval for analytical query services.

#### Configuration Operations Before Running

In security mode, the sample code needs to read the **user.keytab** and **krb5.conf** files. The **user.keytab** and **krb5.conf** files are the authentication files in security mode. You need to download the authentication credential of the principal user on FusionInsight Manager. The user used in the sample code is **sparkuser**, which needs to be changed to the prepared development user.

#### Packaging the Project

1. Upload the **user.keytab** and **krb5.conf** files to the server where the client is located.
2. Use the Maven tool provided by IDEA to package the project and generate the JAR file. For details, see [Compiling and Running the Application](#).

#### NOTE

- Before compilation and packaging, change the paths of the **user.keytab** and **krb5.conf** files in the sample code to the actual paths on the client server, for example, **/opt/user.keytab** and **/opt/krb5.conf**.
- Before running the sample program, set the **spark.yarn.security.credentials.hbase.enabled** configuration item to **true** in the **spark-defaults.conf** configuration file of Spark client. (The default value is **false**. Changing the value to **true** does not affect existing services.) If you want to uninstall the HBase service, change the value back to **false** first.

3. Upload the generated JAR package to any directory (for example, **/opt/**) on the server where the Spark client is located.

## Data Preparation

1. Create an HBase table and construct data with **key**, **modify\_time**, and **valid** columns. **key** of each data record is unique in the table. **modify\_time** indicates the modification time, and **valid** indicates whether the data is valid. In this example, **1** indicates that the data is valid, and **0** indicates that the data is invalid.

For example, go to HBase Shell and run the following commands:

```
create 'hbase_table','key','info'
put 'hbase_table','1','info:modify_time','2019-11-22 23:28:39'
put 'hbase_table','1','info:valid','1'
put 'hbase_table','2','info:modify_time','2019-11-22 23:28:39'
put 'hbase_table','2','info:valid','1'
put 'hbase_table','3','info:modify_time','2019-11-22 23:28:39'
put 'hbase_table','3','info:valid','0'
put 'hbase_table','4','info:modify_time','2019-11-22 23:28:39'
put 'hbase_table','4','info:valid','1'
```

### NOTE

The values of **modify\_time** in the preceding information can be set to the time earlier than the current time.

```
put 'hbase_table','5','info:modify_time','2021-03-03 15:20:39'
put 'hbase_table','5','info:valid','1'
put 'hbase_table','6','info:modify_time','2021-03-03 15:20:39'
put 'hbase_table','6','info:valid','1'
put 'hbase_table','7','info:modify_time','2021-03-03 15:20:39'
put 'hbase_table','7','info:valid','0'
put 'hbase_table','8','info:modify_time','2021-03-03 15:20:39'
put 'hbase_table','8','info:valid','1'
put 'hbase_table','4','info:valid','0'
put 'hbase_table','4','info:modify_time','2021-03-03 15:20:39'
```

### NOTE

The values of **modify\_time** in the preceding information can be set to the time within 30 minutes after the sample program is started. (30 minutes is the default synchronization interval of the sample program and can be modified.)

```
put 'hbase_table','9','info:modify_time','2021-03-03 15:32:39'
put 'hbase_table','9','info:valid','1'
put 'hbase_table','10','info:modify_time','2021-03-03 15:32:39'
put 'hbase_table','10','info:valid','1'
put 'hbase_table','11','info:modify_time','2021-03-03 15:32:39'
put 'hbase_table','11','info:valid','0'
```

```
put 'hbase_table','12','info:modify_time','2021-03-03 15:32:39'
put 'hbase_table','12','info:valid','1'
```

 NOTE

The values of **modify\_time** in the preceding information can be set to the time from 30 minutes to 60 minutes after the sample program is started, that is, the second synchronization period.

2. Run the following commands to create a Hive foreign table for HBase in SparkSQL:

```
create table external_hbase_table(key string ,modify_time STRING, valid STRING)
```

```
using org.apache.spark.sql.hbase.HBaseSource
```

```
options(hbaseTableName "hbase_table", keyCols "key", colsMapping "modify_time=info.modify_time,valid=info.valid");
```

3. Run the following command to create a CarbonData table in SparkSQL:

```
create table carbon01(key string,modify_time STRING, valid STRING) stored as carbondata;
```

4. Initialize and load all data in the current HBase table to the CarbonData table.

```
insert into table carbon01 select * from external_hbase_table where valid='1';
```

5. Run the following **spark-submit** command:

```
spark-submit --master yarn --deploy-mode client --keytab /opt/FIclient/user.keytab --principal sparkuser --class com.huawei.bigdata.spark.examples.HBaseExternalHivetoCarbon /opt/example/HBaseExternalHivetoCarbon-1.0.jar
```

### 30.3.12.2 Java Example Code

The following code snippets are used as an example. For complete code, see **com.huawei.spark.examples.HBaseExternalHivetoCarbon**.

```
public static void main(String[] args) throws Exception {
 spark = SparkSession.builder().appName("HBaseExternalHiveToCarbon").getOrCreate();

 Timer timer = new Timer();
 timer.schedule(new TimerTask() {
 public void run() {
 timeEnd = timeStart + TIMEWINDOW;

 queryTimeStart = transferDateToStr(timeStart);
 queryTimeEnd = transferDateToStr(timeEnd);

 //run delete logic
 cmdsb = new StringBuilder();
 cmdsb.append("delete from ")
 .append(carbonTableName)
 .append(" where key in (select key from ")
 .append(externalHiveTableName)
 .append(" where modify_time>")
 .append(queryTimeStart)
 .append(" and modify_time<")
 .append(queryTimeEnd)
 .append(" and valid=0)");
 spark.sql(cmdsb.toString());

 //run insert logic
 cmdsb = new StringBuilder();
 cmdsb.append("insert into ")
 .append(carbonTableName)
```

```
.append(" select * from ")
.append(externalHiveTableName)
.append(" where modify_time>")
.append(queryTimeStart)
.append("'" and modify_time<")
.append(queryTimeEnd)
.append("'" and valid='1'");
spark.sql(cmdsb.toString());

timeStart = timeEnd;
}
}, TIMEWINDOW, TIMEWINDOW);
}
```

## 30.3.13 Using Spark to Perform Basic Hudi Operations

### 30.3.13.1 Overview

#### Scenarios

This section describes how to use Spark to perform operations such as data insertion, query, update, incremental query, query at a specific time point, and data deletion on Hudi.

For details, see the sample code.

#### Packaging the Project

1. Upload the **user.keytab** and **krb5.conf** files to the server where the client is located.
2. Use the Maven tool provided by IDEA to package the project and generate the JAR file. For details, see [Compiling and Running the Application](#).

#### NOTE

- Before compilation and packaging, change the paths of the **user.keytab** and **krb5.conf** files in the sample code to the actual paths on the client server.
  - The Python sample code does not need to be packaged using Maven.
3. Upload the generated JAR file to any directory (for example, **/opt/example/**) on the server where the Spark client is located.

#### Running tasks

After compiling and building the sample code, you can use the **spark-submit** command to perform the write, update, query, and delete operations in sequence.

- Run the Java sample program.  
**spark-submit --keytab <user\_keytab\_path> --principal=<principal\_name> --class com.huawei.bigdata.hudi.examples.HoodieWriteClientExample /opt/example/hudi-java-security-examples-1.0.jar hdfs://hacluster/tmp/example/hoodie\_java hoodie\_java <user\_keytab\_path>**  
**<user\_keytab\_path>** indicates the authentication file path,  
**<principal\_name>** indicates the authentication user name, **/opt/example/hudi-java-examples-1.0.jar** indicates the JAR file path, **hdfs://**

**hacluster/tmp/example/hoodie\_java** indicates the storage path of the Hudi table, and **hoodie\_java** indicates the name of the Hudi table.

- Run the Scala sample program.

```
spark-submit --keytab <user_keytab_path> --principal=<principal_name> --class
```

```
com.huawei.bigdata.hudi.examples.HoodieDataSourceExample /opt/example/hudi-scala-security-examples-1.0.jar hdfs://hacluster/tmp/example/hoodie_scala hoodie_scala
```

**/opt/example/hudi-scala-examples-1.0.jar** indicates the JAR file path, **<user\_keytab\_path>** indicates the authentication file path, **<principal\_name>** indicates the authentication user name, **hdfs://hacluster/tmp/example/hoodie\_scala** indicates the storage path of the Hudi table, and **hoodie\_Scala** indicates the name of the Hudi table.

- Run the Python sample program.

```
spark-submit /opt/example/HudiPythonExample.py hdfs://hacluster/tmp/huditest/example/python hudi_trips_cow
```

**hdfs://hacluster/tmp/huditest/example/python** indicates the storage path of the Hudi table, and **hudi\_trips\_cow** indicates the name of the Hudi table.

### 30.3.13.2 Java Example Code

The following code snippets are used as an example. For complete code, see **com.huawei.bigdata.hudi.examples.HoodieWriteClientExample**.

Create a client object to operate Hudi:

```
String tablePath = args[0];
String tableName = args[1];
SparkConf sparkConf = HoodieExampleSparkUtils.defaultSparkConf("hoodie-client-example");
JavaSparkContext jsc = new JavaSparkContext(sparkConf);
// Generator of some records to be loaded in.
HoodieExampleDataGenerator<HoodieAvroPayload> dataGen = new HoodieExampleDataGenerator<>();
// initialize the table, if not done already
Path path = new Path(tablePath);
FileSystem fs = FSUtils.getFs(tablePath, jsc.hadoopConfiguration());
if (!fs.exists(path)) {
 HoodieTableMetaClient.initTableType(jsc.hadoopConfiguration(), tablePath,
 HoodieTableType.valueOf(tableType),
 tableName, HoodieAvroPayload.class.getName());
}

// Create the write client to write some records in
HoodieWriteConfig cfg = HoodieWriteConfig.newBuilder().withPath(tablePath)
 .withSchema(HoodieExampleDataGenerator.TRIP_EXAMPLE_SCHEMA).withParallelism(2, 2)
 .withDeleteParallelism(2).forTable(tableName)
 .withIndexConfig(HoodieIndexConfig.newBuilder().withIndexType(HoodieIndex.IndexType.BLOOM).build()
)
 .withCompactionConfig(HoodieCompactionConfig.newBuilder().archiveCommitsWith(20,
 30).build()).build();
SparkRDDWriteClient<HoodieAvroPayload> client = new SparkRDDWriteClient<>(new
HoodieSparkEngineContext(jsc), cfg);
```

Insert data:

```
String newCommitTime = client.startCommit();
LOG.info("Starting commit " + newCommitTime);
List<HoodieRecord<HoodieAvroPayload>> records = dataGen.generateInserts(newCommitTime, 10);
List<HoodieRecord<HoodieAvroPayload>> recordsSoFar = new ArrayList<>(records);
JavaRDD<HoodieRecord<HoodieAvroPayload>> writeRecords = jsc.parallelize(records, 1);
client.upsert(writeRecords, newCommitTime);
```



## Update data:

```
newCommitTime = client.startCommit();
LOG.info("Starting commit " + newCommitTime);
List<HoodieRecord<HoodieAvroPayload>> toBeUpdated = dataGen.generateUpdates(newCommitTime, 2);
records.addAll(toBeUpdated);
recordsSoFar.addAll(toBeUpdated);
writeRecords = jsc.parallelize(records, 1);
client.upsert(writeRecords, newCommitTime);
```

## Delete data:

```
newCommitTime = client.startCommit();
LOG.info("Starting commit " + newCommitTime);
// just delete half of the records
int numToDelete = recordsSoFar.size() / 2;
List<HoodieKey> toBeDeleted =
recordsSoFar.stream().map(HoodieRecord::getKey).limit(numToDelete).collect(Collectors.toList());
JavaRDD<HoodieKey> deleteRecords = jsc.parallelize(toBeDeleted, 1);
client.delete(deleteRecords, newCommitTime);
```

## Compress data.

```
if (HoodieTableType.valueOf(tableType) == HoodieTableType.MERGE_ON_READ) {
 Option<String> instant = client.scheduleCompaction(Option.empty());
 JavaRDD<WriteStatus> writeStatuses = client.compact(instant.get());
 client.commitCompaction(instant.get(), writeStatuses, Option.empty());
}
```

### 30.3.13.3 Scala Example Code

The following code snippets are used as an example. For complete code, see [com.huawei.bigdata.hudi.examples.HoodieDataSourceExample](#).

## Insert data:

```
def insertData(spark: SparkSession, tablePath: String, tableName: String, dataGen:
HoodieExampleDataGenerator[HoodieAvroPayload]): Unit = {
 val commitTime: String = System.currentTimeMillis().toString
 val inserts = dataGen.convertToStringList(dataGen.generateInserts(commitTime, 20))
 spark.sparkContext.parallelize(inserts, 2)
 val df = spark.read.json(spark.sparkContext.parallelize(inserts, 1)).df.write.format("org.apache.hudi").
 options(getQuickstartWriteConfigs).
 option(PRECOMBINE_FIELD_OPT_KEY, "ts").
 option(RECORDKEY_FIELD_OPT_KEY, "uuid").
 option(PARTITIONPATH_FIELD_OPT_KEY, "partitionpath").
 option(TABLE_NAME, tableName).
 mode(Overwrite).
 save(tablePath)}
```

## Query data.

```
def queryData(spark: SparkSession, tablePath: String, tableName: String, dataGen:
HoodieExampleDataGenerator[HoodieAvroPayload]): Unit = {
 val roViewDF = spark.
 read.
 format("org.apache.hudi").
 load(tablePath + "/*/*/*")
 roViewDF.createOrReplaceTempView("hudi_ro_table")
 spark.sql("select fare, begin_lon, begin_lat, ts from hudi_ro_table where fare > 20.0").show()
 // +-----+-----+-----+-----+
 // | fare| begin_lon| begin_lat| ts|
 // +-----+-----+-----+-----+
 // |98.88075495133515|0.39556048623031603|0.17851135255091155|0.0|
 // ...
 spark.sql("select _hoodie_commit_time, _hoodie_record_key, _hoodie_partition_path, rider, driver, fare from
hudi_ro_table").show()
 // +-----+-----+-----+-----+-----+
 // +-----+
```

```
// [_hoodie_commit_time|_hoodie_record_key|hoodie_partition_path| rider|
driver| fare|
// +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
// | 20191231181501|31cafb9f-0196-4b1...| 2020/01/02|rider-1577787297889|
driver-1577787297889| 98.88075495133515|
// ...
}
```

#### Update data:

```
def updateData(spark: SparkSession, tablePath: String, tableName: String, dataGen:
HoodieExampleDataGenerator[HoodieAvroPayload]): Unit = {
val commitTime: String = System.currentTimeMillis().toString
val updates = dataGen.convertToStringList(dataGen.generateUpdates(commitTime, 10))
val df = spark.read.json(spark.sparkContext.parallelize(updates, 1))
df.write.format("org.apache.hudi").
options(getQuickstartWriteConfigs).
option(PRECOMBINE_FIELD_OPT_KEY, "ts").
option(RECORDKEY_FIELD_OPT_KEY, "uuid").
option(PARTITIONPATH_FIELD_OPT_KEY, "partitionpath").
option(TABLE_NAME, tableName).
mode(Append).
save(tablePath)}
```

#### Incremental query:

```
def incrementalQuery(spark: SparkSession, tablePath: String, tableName: String) {
import spark.implicits._
val commits = spark.sql("select distinct(_hoodie_commit_time) as commitTime from hudi_ro_table order by
commitTime").map(k => k.getString(0)).take(50)
val beginTime = commits(commits.length - 2)

val incViewDF = spark.
read.
format("org.apache.hudi").
option(QUERY_TYPE_OPT_KEY, QUERY_TYPE_INCREMENTAL_OPT_VAL).
option(BEGIN_INSTANTTIME_OPT_KEY, beginTime).
load(tablePath)
incViewDF.createOrReplaceTempView("hudi_incr_table")
spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_incr_table where fare >
20.0").show()}
```

#### Query at a specific time point:

```
def pointInTimeQuery(spark: SparkSession, tablePath: String, tableName: String) {
import spark.implicits._
val commits = spark.sql("select distinct(_hoodie_commit_time) as commitTime from hudi_ro_table order by
commitTime").map(k => k.getString(0)).take(50)
val beginTime = "000"
// Represents all commits > this time.
val endTime = commits(commits.length - 2)
// commit time we are interested in
//incrementally query data
val incViewDF = spark.read.format("org.apache.hudi").
option(QUERY_TYPE_OPT_KEY, QUERY_TYPE_INCREMENTAL_OPT_VAL).
option(BEGIN_INSTANTTIME_OPT_KEY, beginTime).
option(END_INSTANTTIME_OPT_KEY, endTime).
load(tablePath)
incViewDF.createOrReplaceTempView("hudi_incr_table")
spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_incr_table where fare >
20.0").show()}
```

### 30.3.13.4 Python Example Code

The following code snippets are used as an example. For complete code, see [HudiPythonExample.py](#).

### Insert data:

```
#insert
inserts = sc._jvm.org.apache.hudi.QuickstartUtils.convertToStringList(dataGen.generateInserts(10))
df = spark.read.json(spark.sparkContext.parallelize(inserts, 2))
hoodie_options = {
'hoodie.table.name': tableName,
'hoodie.datasource.write.recordkey.field': 'uuid',
'hoodie.datasource.write.partitionpath.field': 'partitionpath',
'hoodie.datasource.write.table.name': tableName,
'hoodie.datasource.write.operation': 'insert',
'hoodie.datasource.write.precombine.field': 'ts',
'hoodie.upsert.shuffle.parallelism': 2,
hoodie.insert.shuffle.parallelism': 2
}
df.write.format("hudi"). \
 options(**hoodie_options). \
 mode("overwrite"). \
 save(basePath)
```

### Query data.

```
tripsSnapshotDF = spark. \
 read. \
 format("hudi"). \
 load(basePath + "/*/*/*/*")
tripsSnapshotDF.createOrReplaceTempView("hudi_trips_snapshot")
spark.sql("select fare, begin_lon, begin_lat, ts from hudi_trips_snapshot where fare > 20.0").show()
spark.sql("select _hoodie_commit_time, _hoodie_record_key, _hoodie_partition_path, rider, driver, fare from hudi_trips_snapshot").show()
```

### Update data:

```
updates = sc._jvm.org.apache.hudi.QuickstartUtils.convertToStringList(dataGen.generateUpdates(10))
df = spark.read.json(spark.sparkContext.parallelize(updates, 2))
df.write.format("hudi"). \
 options(**hoodie_options). \
 mode("append"). \
 save(basePath)
```

### Incremental query:

```
spark. \
 read. \
 format("hudi"). \
 load(basePath + "/*/*/*/*"). \
 createOrReplaceTempView("hudi_trips_snapshot")
incremental_read_options = {
'hoodie.datasource.query.type': 'incremental',
'hoodie.datasource.read.begin.instanttime': beginTime,
}
tripsIncrementalDF = spark.read.format("hudi"). \
 options(**incremental_read_options). \
 load(basePath)
tripsIncrementalDF.createOrReplaceTempView("hudi_trips_incremental")
spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_trips_incremental where fare > 20.0").show()
```

### Query at a specific time point:

```
Represents all commits > this time.
beginTime = "000"
endTime = commits[len(commits) - 2]
point_in_time_read_options = {
'hoodie.datasource.query.type': 'incremental',
'hoodie.datasource.read.end.instanttime': endTime,
'hoodie.datasource.read.begin.instanttime': beginTime
}

tripsPointInTimeDF = spark.read.format("hudi"). \
 options(**point_in_time_read_options). \
```

```
load(basePath)

tripsPointInTimeDF.createOrReplaceTempView("hudi_trips_point_in_time")
spark.sql("select `hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_trips_point_in_time where fare > 20.0").show()
```

Delete data:

```
Obtain the total number of records.
spark.sql("select uuid, partitionpath from hudi_trips_snapshot").count()
Obtain two records to be deleted.
ds = spark.sql("select uuid, partitionpath from hudi_trips_snapshot").limit(2)
Delete the records.
hudi_delete_options = {
 'hoodie.table.name': tableName,
 'hoodie.datasource.write.recordkey.field': 'uuid',
 'hoodie.datasource.write.partitionpath.field': 'partitionpath',
 'hoodie.datasource.write.table.name': tableName,
 'hoodie.datasource.write.operation': 'delete',
 'hoodie.datasource.write.precombine.field': 'ts',
 'hoodie.upsert.shuffle.parallelism': 2,
 'hoodie.insert.shuffle.parallelism': 2
}
from pyspark.sql.functions import lit
deletes = list(map(lambda row: (row[0], row[1]), ds.collect()))
df = spark.sparkContext.parallelize(deletes).toDF(['uuid', 'partitionpath']).withColumn('ts', lit(0.0))
df.write.format("hudi"). \
 options(**hudi_delete_options). \
 mode("append"). \
 save(basePath)
Perform the query in the same way.
roAfterDeleteViewDF = spark. \
 read. \
 format("hudi"). \
 load(basePath + "/*/*/*/*")
roAfterDeleteViewDF.registerTempTable("hudi_trips_snapshot")
Return (total - 2) records.
spark.sql("select uuid, partitionpath from hudi_trips_snapshot").count()
spark.sql("select uuid, partitionpath from hudi_trips_snapshot").show()
```

## 30.3.14 Compiling User-defined Configuration Items for Hudi

### 30.3.14.1 HoodieDeltaStreamer

Compile a user-defined conversion class for **Transformer**.

Compile a user-defined schema for **SchemaProvider**.

Add the following parameters when running **HoodieDeltaStreamer**:

```
--schemaprovider-class Defined schema class --transformer-class Defined transform class
```

Examples of **Transformer** and **SchemaProvider**:

```
public class TransformerExample implements Transformer, Serializable {
 @Override
 public Dataset<Row> apply(JavaSparkContext jsc, SparkSession sparkSession, Dataset<Row> rowDataset,
 TypedProperties properties) {
 JavaRDD<Row> rowJavaRdd = rowDataset.toJavaRDD();
 List<Row> rowList = new ArrayList<>();
 for(Row row: rowJavaRdd.collect()){
 rowList.add(buildRow(row));
 }
 JavaRDD<Row> stringJavaRdd = jsc.parallelize(rowList);
 List<StructField> fields = new ArrayList<>();
 buildFields(fields);
 }
}
```

```
StructType schema = DataTypes.createStructType(fields);
Dataset<Row> dataFrame = sparkSession.createDataFrame(stringJavaRdd, schema);
return dataFrame;
}

private void builFields(List<StructField> fields) {
 fields.add(DataTypes.createStructField("age", DataTypes.StringType, true));
 fields.add(DataTypes.createStructField("id", DataTypes.StringType, true));
 fields.add(DataTypes.createStructField("name", DataTypes.StringType, true));
 fields.add(DataTypes.createStructField("job", DataTypes.StringType, true));
}

private Row buildRow(Row row){
 String age = row.getString(0);
 String id = row.getString(1);
 String job = row.getString(2);
 String name = row.getString(3);
 Row returnRow = RowFactory.create(age, id, job, name);
 return returnRow;
}
}
public class DataSchemaProviderExample extends SchemaProvider {

 public DataSchemaProviderExample(TypedProperties props, JavaSparkContext jssc) {
 super(props, jssc);
 }

 @Override
 public Schema getSourceSchema() {
 Schema avroSchema = new Schema.Parser().parse(
 "{\"type\":\"record\",\"name\":\"hoodie_source\",\"fields\": [{\"name\":\"age\",\"type\":\"string\"},
 {\"name\":\"id\",\"type\":\"string\"},{\"name\":\"job\",\"type\":\"string\"},{\"name\":\"name\",\"type\":
 \"string\"}]}");
 return avroSchema;
 }

 @Override
 public Schema getTargetSchema() {
 Schema avroSchema = new Schema.Parser().parse(
 "{\"type\":\"record\",\"name\":\"mytest_record\",\"namespace\":\"hoodie.mytest\",\"fields\":
 [{\"name\":\"age\",\"type\":\"string\"},{\"name\":\"id\",\"type\":\"string\"},{\"name\":\"job\",\"type\":\"string
 \"},{\"name\":\"name\",\"type\":\"string\"}]}");
 return avroSchema;
 }
}
}
```

### 30.3.14.2 User-defined Partitioner

Compile a user-defined partitioner class that inherits **BulkInsertPartitioner** and add the following configuration when writing data to Hudi:

```
.option(BULKINSERT_USER_DEFINED_PARTITIONER_CLASS, <User-defined partitioner class package name + class name>)
```

Example of the user-defined partitioner:

```
public class HoodieSortExample<T> extends HoodieRecordPayload<T>
 implements BulkInsertPartitioner<JavaRDD<HoodieRecord<T>>> {
 @Override
 public JavaRDD<HoodieRecord<T>> repartitionRecords(JavaRDD<HoodieRecord<T>> records, int
 outputSparkPartitions) {
 JavaPairRDD<String,
 HoodieRecord<T>> stringHoodieRecordJavaPairRDD = records.coalesce(outputSparkPartitions)
 .mapToPair(record -> new Tuple2<>(new StringBuilder().append(record.getPartitionPath())
 .append("+")
 .append(record.getRecordKey())
 .toString(), record));
 }
}
```

```
JavaRDD<HoodieRecord<T>> hoodieRecordJavaRDD =
stringHoodieRecordJavaPairRDD.mapPartitions(partition -> {
 List<Tuple2<String, HoodieRecord<T>>> recordList = new ArrayList<>();
 for (; partition.hasNext()); {
 recordList.add(partition.next());
 }
 Collections.sort(recordList, (o1, o2) -> {
 if (o1._1().split("[+]").[0] == o2._1().split("[+]").[0]) {
 return Integer.parseInt(o1._1().split("[+]").[1]) - Integer.parseInt(o2._1().split("[+]").[1]);
 } else {
 return o1._1().split("[+]").[0].compareTo(o2._1().split("[+]").[0]);
 }
 });
 return recordList.stream().map(e -> e._2).iterator();
});
return hoodieRecordJavaRDD;
}

@Override
public boolean arePartitionRecordsSorted() {
 return true;
}
}
```

## 30.4 Commissioning the Application

### 30.4.1 Commissioning Applications on Windows

#### 30.4.1.1 Spark Access Configuration on Windows Using EIPs

##### Scenario

This section describes how to bind Elastic IP addresses (EIPs) to a cluster and configure Spark files so that sample files can be compiled locally.

This section uses SparkScalaExample as an example.

##### Procedure

**Step 1** Apply for an EIP for each node in the cluster and add public IP addresses and corresponding host domain names of all nodes to the Windows local **hosts** file. (If a host name contains uppercase letters, change them to lowercase letters.)

1. On the VPC console, apply for EIPs (the number of EIPs you buy should be equal to the number of nodes in the cluster), click the name of each node in the MRS cluster, and bind an EIP to each node on the **EIPs** page.

For details, see **Virtual Private Cloud > User Guide > EIP > Assigning an EIP and Binding It to an ECS**.

2. Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.

```

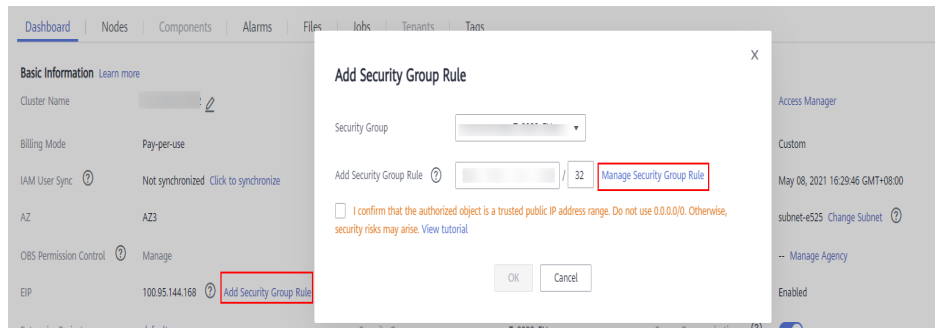
1 Mapping between public IP addresses and private IP addresses
2 100.95.144.120 172.16.0.120
3 100.95.144.139 172.16.0.42
4 100.93.144.168 172.16.0.62
5 100.95.144.168 172.16.0.200
6 100.93.144.168 172.16.0.139
7 100.93.144.168 172.16.0.214
8
9 hosts file in the cluster
10 172.16.0.120 node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
11 172.16.0.42 node-master3VinT.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VinT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
12 172.16.0.62 node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VinT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
13 172.16.0.200 node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
14 172.16.0.139 node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
15 172.16.0.214 node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.
16
17 hosts file that should be added to Windows
18 100.95.144.120 node-group-1xzi00002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
19 100.95.144.139 node-master3VinT.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VinT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
20 100.93.144.168 node-group-1xzi00003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
21 100.95.144.168 node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
22 100.93.144.168 node-group-1xzi00001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
23 100.93.144.168 node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.

```

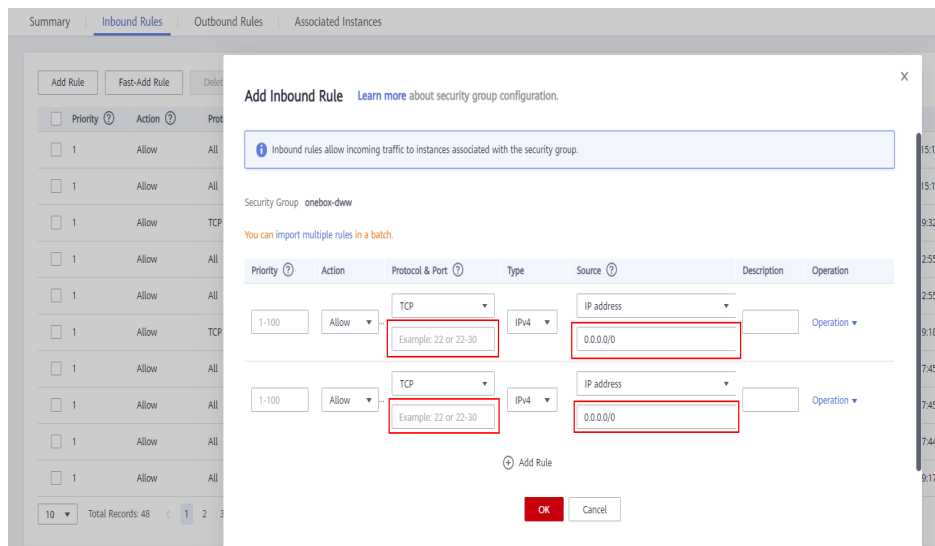
**Step 2** Change the IP addresses in the `krb5.conf` file to the corresponding host names.

**Step 3** Configure security group rules for the cluster.

1. On the **Dashboard** page, choose **Add Security Group Rule > Manage Security Group Rule**.



2. On the **Inbound Rules** tab page, click **Add Rule**. In the **Add Inbound Rule** dialog box, configure Windows IP addresses and ports 21730TCP, 21731TCP/UDP, and 21732TCP/UDP.



**Step 4** On Manager, choose **Cluster > Services > HDFS > More > Download Client**, and copy the `core-site.xml` and `hdfs-site.xml` files on the client to the `conf` directory of the sample project.

Add the following content to the `hdfs-site.xml` file:

```

<property>
 <name>dfs.client.use.datanode.hostname</name>

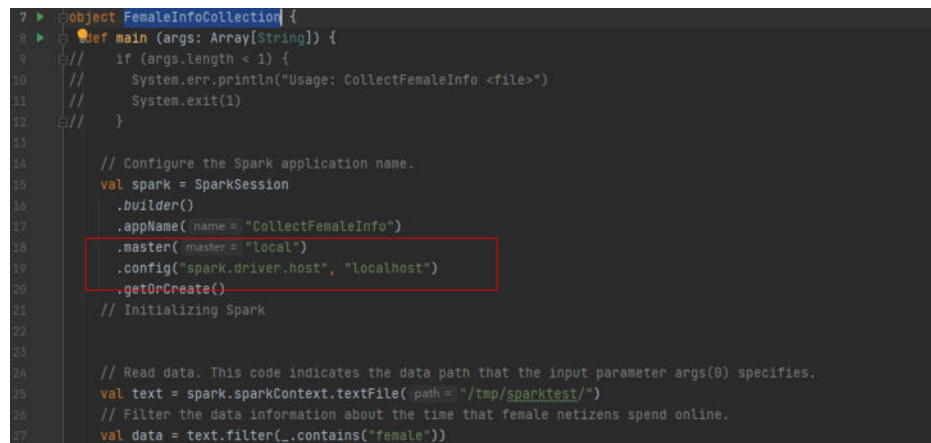
```

```
<value>true</value>
</property>
```

Add the following content to the **pom.xml** file:

```
<dependency>
 <groupId>com.huawei.mrs</groupId>
 <artifactId>hadoop-plugins</artifactId>
 <version>Component package version-302002</version>
</dependency>
```

**Step 5** Before running the sample code, add **.master("local").config("spark.driver.host", "localhost")** to **SparkSession** to set the local running mode for Spart. Change **PRINCIPAL\_NAME** in the sample code to the username for security authentication.

A screenshot of a code editor showing Scala code for a Spark application. The code defines a class `FemaleInfoCollection` with a `main` method. The `main` method checks for command-line arguments and prints usage if there are none. It then configures a `SparkSession` with the name "CollectFemaleInfo", sets the master to "local", and configures the driver host to "localhost". The code then reads a text file from the local file system and filters it for lines containing the word "female".

```
7 ▶ object FemaleInfoCollection {
8 ▶ def main (args: Array[String]) {
9 // if (args.length < 1) {
10 // System.err.println("Usage: CollectFemaleInfo <file>")
11 // System.exit(1)
12 // }
13
14 // Configure the Spark application name.
15 val spark = SparkSession
16 .builder()
17 .appName(name = "CollectFemaleInfo")
18 .master(master = "local")
19 .config("spark.driver.host", "localhost")
20 .getOrCreate()
21 // Initializing Spark
22
23
24 // Read data. This code indicates the data path that the input parameter args(0) specifies.
25 val text = spark.sparkContext.textFile(path = "/tmp/sparktest/")
26 // Filter the data information about the time that female netizens spend online.
27 val data = text.filter(_.contains("female"))
```

----End

## 30.4.1.2 Compiling and Running Applications

### Scenario

You can run applications in the Windows environment after application code development is complete. The procedures for running applications developed using Scala or Java are the same on IDEA.

#### NOTE

- In the Windows environment, only the sample code for accessing Spark SQL using JDBC is provided.
- Ensure that the Maven image repository of the SDK in the Huawei image site has been configured for Maven. For details, see [Configuring Huawei Open-Source Mirrors](#).

### Procedure

**Step 1** Obtain the sample code.

Download the Maven project source code and configuration file of the sample project. For details, see [Obtaining Sample Projects](#).

Import the sample code to IDEA.

**Step 2** Obtain configuration files.



1. Obtain the files from the cluster client. Download the **hive-site.xml** and **spark-defaults.conf** files from **\$SPARK\_HOME/conf** to a local directory.
2. On the FusionInsight Manager page of the cluster, download the user authentication file to a local directory.

### Step 3 Upload data to HDFS.

1. Create a data text file on Linux and save the following data to the data file:  
Miranda,32  
Karlie,23  
Candice,27
2. On the HDFS client, run the following commands for authentication:  
**cd** {Client installation directory}  
**kinit** <Service user for authentication>
3. On the HDFS client running the Linux OS, run the **hadoop fs -mkdir /data** command (or the **hdfs dfs** command) to create a directory.
4. On the HDFS client running the Linux OS, run the **hadoop fs -put data /data** command to upload the data file.

### Step 4 Configure related parameters in the sample code.

1. Configure the authentication information.

Set **userPrincipal** to the username.

Set **userKeytabPath** to the path of the downloaded **keytab** file.

Set **Krb5ConfPath** to the path of the downloaded **krb5.conf** file.

```
public class ThriftServerQueriesTest {
 public static void main(String[] args) throws SQLException, ClassNotFoundException, IOException {
 String userPrincipal = "adminTest";
 String userKeytabPath = "D:\\conf\\keytab\\user.keytab";
 String krb5ConfPath = "D:\\conf\\keytab\\krb5.conf";
 String principalName = KerberosUtil.DEFAULT_REALM;
 String ZKServerPrincipal = "zookeeper/hadoop." + principalName;

 String ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME = "Client";
 String ZOOKEEPER_SERVER_PRINCIPAL_KEY = "zookeeper.server.principal";
 }
}
```

Set the domain name to **DEFAULT\_REALM**. In the **KerberosUtil** class, change **DEFAULT\_REALM** to the domain name of the cluster.

```
public class KerberosUtil {
 private static Logger logger = Logger.getLogger(KerberosUtil.class);

 public static final String JAVA_VENDOR = "java.vendor";
 public static final String IBM_FLAG = "IBM";
 public static final String CONFIG_CLASS_FOR_IBM = "com.ibm.security.krb5.internal.Config";
 public static final String CONFIG_CLASS_FOR_SUN = "sun.security.krb5.Config";
 public static final String METHOD_GET_INSTANCE = "getInstance";
 public static final String METHOD_GET_DEFAULT_REALM = "getDefaultRealm";
 public static final String DEFAULT_REALM = "HADOOP.COM";
}
```

2. Change **user.principal** and **user.keytab** in the string concatenated by **securityConfig** to the corresponding username and path. Note that the path of the **keytab** file must use slashes (/).

```
String securityConfig = ";sasLQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop." + principalName + "@"
 + principalName + ";user.principal=adminTest;user.keytab=D:/conf/keytab/user.keytab";
```

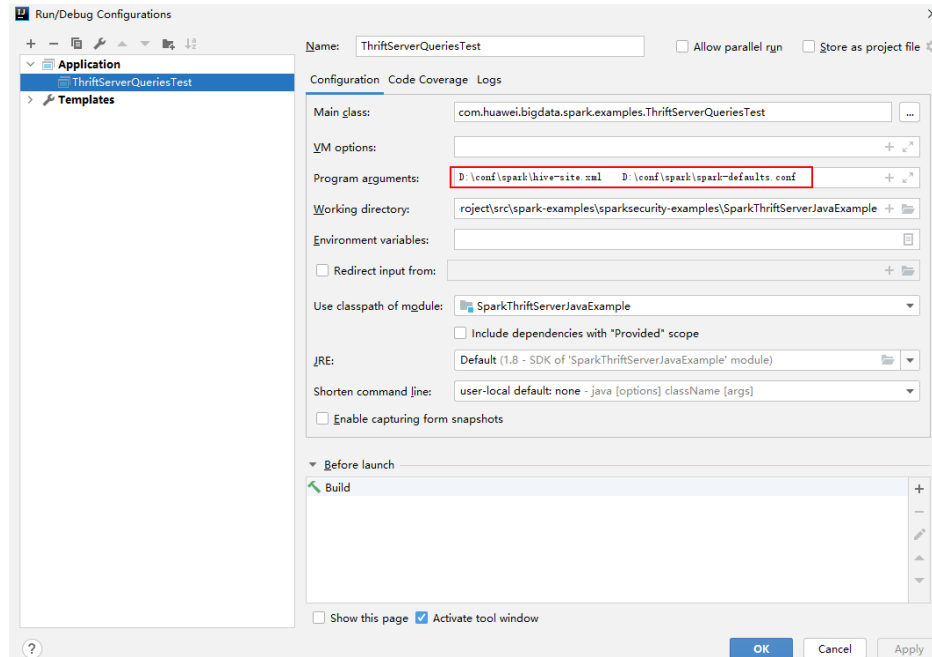
3. Change the SQL statement for loading data to **LOAD DATA INPATH 'hdfs:/data/data' INTO TABLE CHILD**.

```

ArrayList<String> sqlList = new ArrayList<>();
sqlList.add("CREATE TABLE IF NOT EXISTS CHILD (NAME STRING, AGE INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY"
+ " ','");
sqlList.add("LOAD DATA INPATH 'hdfs://data/data' INTO TABLE CHILD");
sqlList.add("SELECT * FROM child");
sqlList.add("DROP TABLE child");
executeSql(url, sqlList);

```

**Step 5** Add running parameters to the `hive-site.xml` and `spark-defaults.conf` files when the application is running.



**Step 6** Run the application.

----End

### 30.4.1.3 View Debugging Results

```

SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/D:/mavenlocal/org/apache/logging/log4j/log4j-slf4j-impl/2.6.2/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/D:/mavenlocal/org/slf4j/slf4j-log4j12/1.7.30/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to the console.
---- Begin executing sql: CREATE TABLE IF NOT EXISTS CHILD (NAME STRING, AGE INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ----
Result
---- Done executing sql: CREATE TABLE IF NOT EXISTS CHILD (NAME STRING, AGE INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ----
---- Begin executing sql: LOAD DATA INPATH 'hdfs://data/data' INTO TABLE CHILD ----
Result
---- Done executing sql: LOAD DATA INPATH 'hdfs://data/data' INTO TABLE CHILD ----
---- Begin executing sql: SELECT * FROM child ----
NAME AGE
Miranda 32
Karlie 23
Candice 27
---- Done executing sql: SELECT * FROM child ----
---- Begin executing sql: DROP TABLE child ----
Result

```

```
---- Done executing sql: DROP TABLE child ----
Process finished with exit code 0
```

## 30.4.2 Commissioning an Application in Linux

### 30.4.2.1 Compiling and Running the Application

#### Scenario

After the program codes are developed, you can upload the codes to the Linux client for running. The running procedures of applications developed in Scala or Java are the same.

#### NOTE

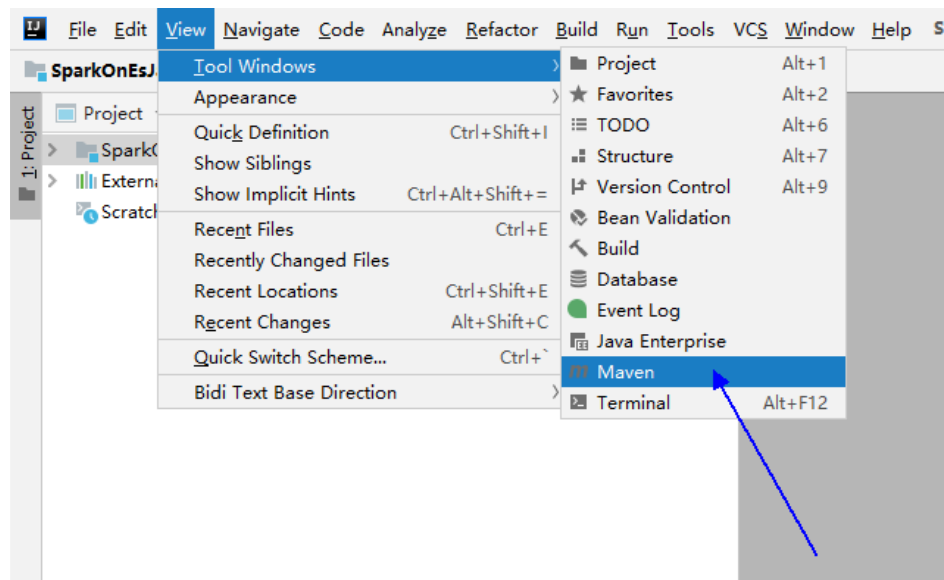
- The Spark application developed in Python does not need to build Artifacts as a jar. You just need to copy the sample projects to the compiler.
- It is needed to ensure that the version of Python installed on the worker and driver is consistent, otherwise the following error will be reported: "Python in worker has different version %s than that in driver %s."
- Ensure that Maven image repository of the SDK in the Huawei image site has been configured in Maven. For details, see [Configuring Huawei Open-Source Mirrors](#).

#### Procedure

**Step 1** In the IntelliJ IDEA, open the Maven tool window.

On the main page of the IDEA, choose **View > Tool Windows > Maven** to open the Maven tool window.

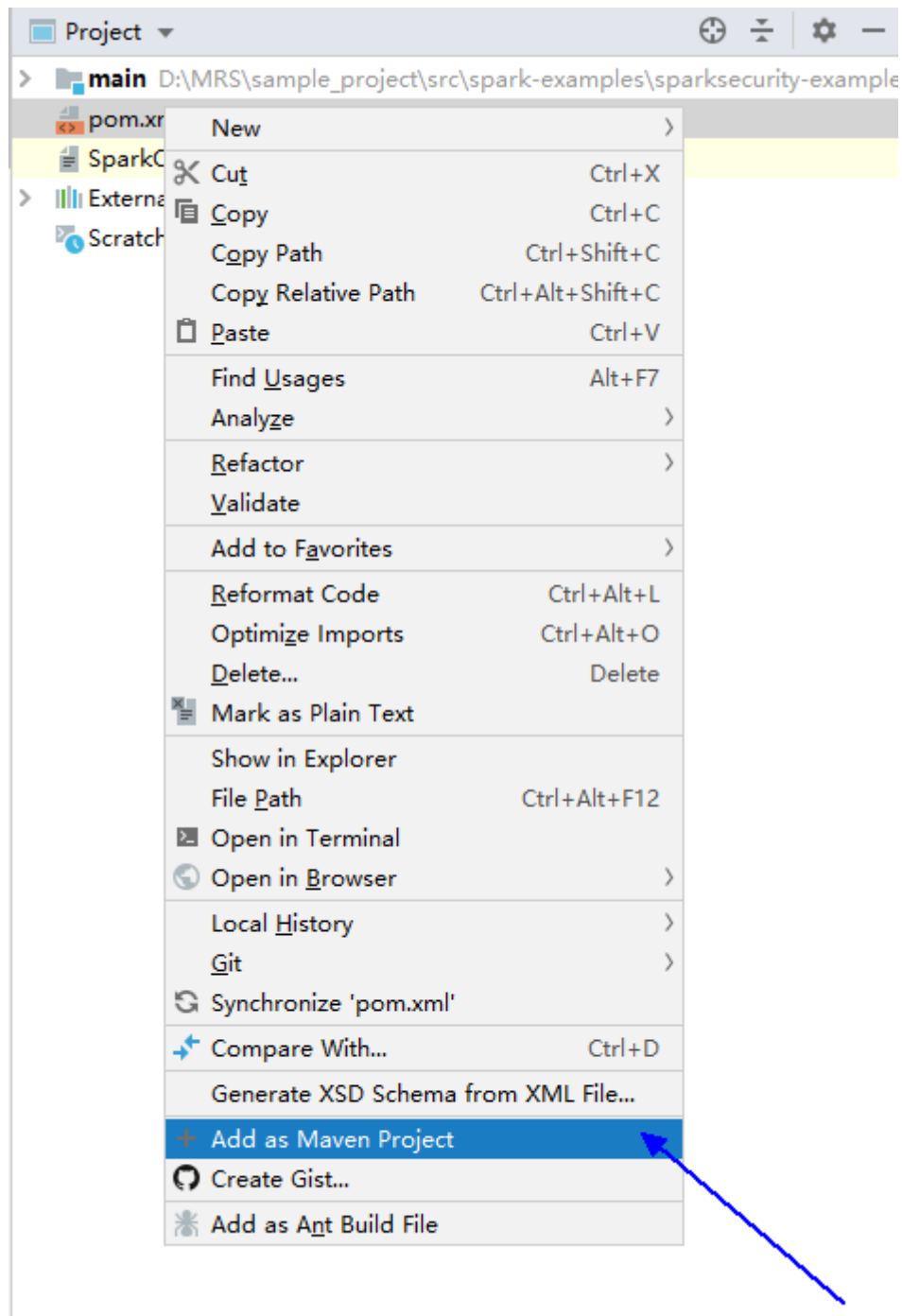
**Figure 30-35** Opening the Maven tool window



If the project is not imported using Maven, perform the following operations:

Right-click the **pom** file in the sample code project and choose **Add as Maven Project** from the shortcut menu to add a Maven project.

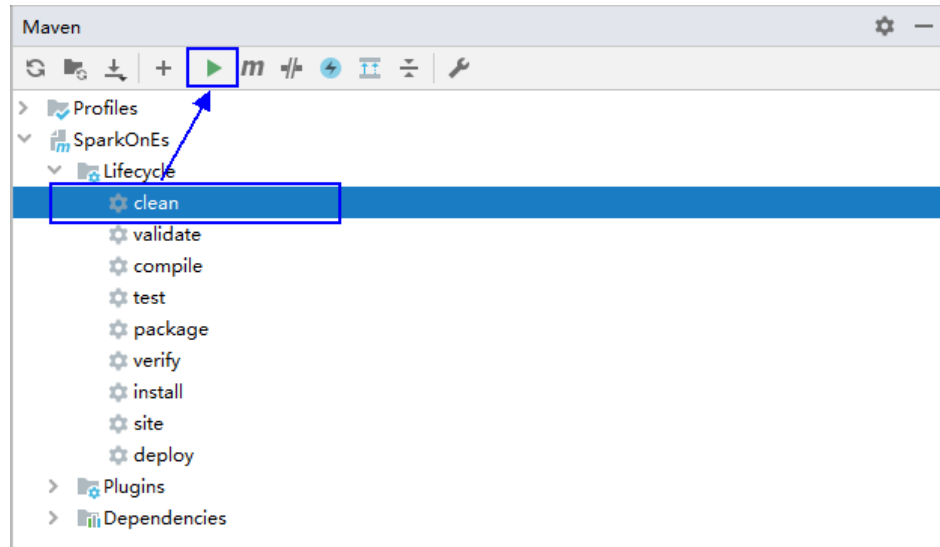
**Figure 30-36** Adding a Maven project



**Step 2** Use Maven to generate a JAR file.

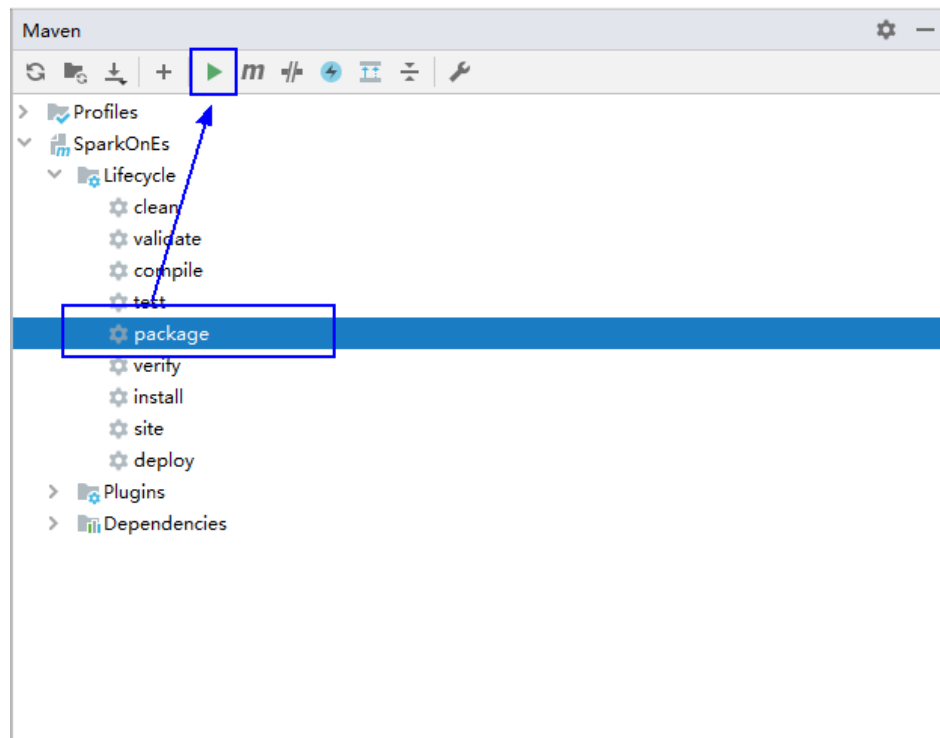
1. In the Maven tool window, select **clean** from **Lifecycle** to execute the Maven building process.

**Figure 30-37** Selecting **clean** from **Lifecycle** and execute the Maven building process



2. In the Maven tool window, select **package** from **Lifecycle** and execute the Maven building process.

**Figure 30-38** Selecting **package** from **Lifecycle** and execute the Maven build process.



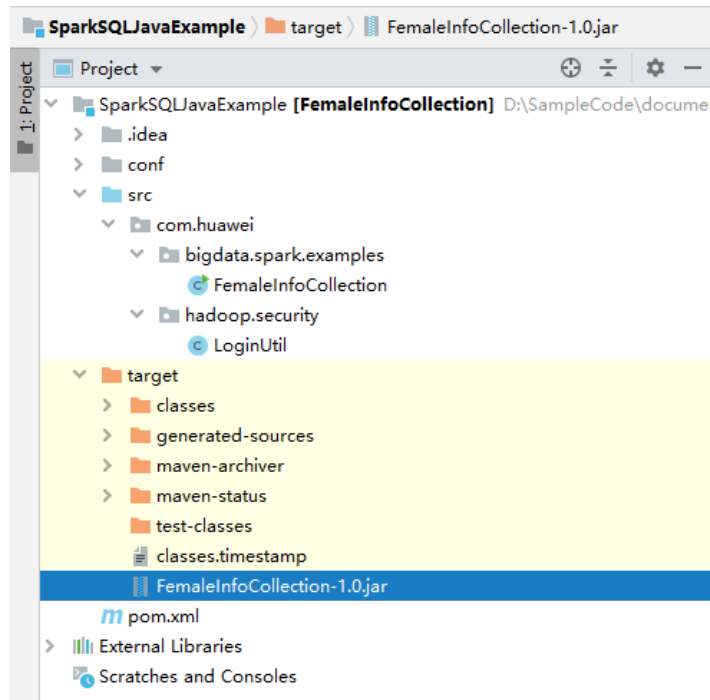
If the following information is displayed in **Run:**, the packaging is successful.

**Figure 30-39** Packaging success message

```
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ FemaleInfoCollection ---
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ FemaleInfoCollection ---
[INFO] Building jar: D:\SampleCode\document\VT\code\sparksecurity-examples\SparkSQLJavaExample\target\FemaleInfoCollection-1.0.jar
[INFO] BUILD SUCCESS
[INFO] Total time: 19.427 s
[INFO] Finished at: 2020-09-21T11:17:31+08:00
```

3. You can obtain the JAR package from the target folder in the project directory.

**Figure 30-40** Obtaining the JAR Package



- Step 3** Copy the JAR file generated in [Step 2](#) (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. Run the Spark application. For details about the example application, see [Developing the Project](#).

**CAUTION**

Do not restart the HDFS service or all DataNode instances during Spark job running. Otherwise, the job may fail and some JobHistory data may be lost.

----End

### 30.4.2.2 Checking the Commissioning Result

#### Scenario

After a Spark application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Logging in to the Spark web UI.
- Viewing Spark logs.

## Procedure

- **Check the operating result data of the Spark application.**  
The data storage directory and format are specified by users in the Spark application. You can obtain the data in the specified file.
- **Check the status of the Spark application.**  
The Spark contains the following two web UIs:
  - The Spark UI displays the status of applications being executed.  
The Spark UI contains the **Spark Jobs**, **Spark Stages**, **Storage Environment**, and **Executors** parts. Besides these parts, **Streaming** is displayed for the Streaming application.  
Access to the interface: On the web UI of the YARN, find the corresponding Spark application, and click **ApplicationMaster** in the last column of the application information. The Spark UI page is displayed.
  - The History Server UI displays the status of all Spark applications.  
The History Server UI displays information such as the application ID, application name, start time, end time, execution time, and user to whom the application belongs. After the application ID is clicked, the Spark UI of the application is displayed.
- **View Spark logs to learn application running conditions.**  
The logs of Spark offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 30.5 More Information

### 30.5.1 Common APIs

#### 30.5.1.1 Java

To avoid API compatibility or reliability issues after updates to the open-source Spark, it is advisable to use APIs of the version you are currently using.

### Spark Core Common Interfaces

Spark mainly uses the following classes:

- **JavaSparkContext**: external interface of Spark, which is used to provide the functions of Spark for Java applications that invoke this class, for example, connecting Spark clusters and generating RDDs, accumulations, and broadcasts. It functions as a container.
- **SparkConf**: Spark application configuration class, which is used to configure the application name, execution model, and executor memory.

- **JavaRDD**: class used to define the JavaRDD in the Java application, which functions like the RDD(Resilient Distributed Dataset) class of Scala.
- **JavaPairRDD**: indicates the JavaRDD in the key-value format. This class provides methods such as `groupByKey` and `reduceByKey`.
- **Broadcast**: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- **StorageLevel**: data storage levels, including memory (`MEMORY_ONLY`), disk (`DISK_ONLY`), and memory+disk (`MEMORY_AND_DISK`).

The JavaRDD supports two types of operations, transformation and action. [Table 30-10](#) and [Table 30-11](#) show the common methods.

**Table 30-10** Transformation

Method	Description
<code>&lt;R&gt; JavaRDD&lt;R&gt; map(Function&lt;T,R&gt; f)</code>	Returns a new RDD by applying a function to all elements of this RDD.
<code>JavaRDD&lt;T&gt; filter(Function&lt;T,Boolean&gt; f)</code>	Invokes Function on all elements of the RDD and returns the element that is <b>true</b> .
<code>&lt;U&gt; JavaRDD&lt;U&gt; flatMap(FlatMapFunction&lt;T,U&gt; f)</code>	Returns a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
<code>JavaRDD&lt;T&gt; sample(boolean withReplacement, double fraction, long seed)</code>	Returns a sampled subset of this RDD.
<code>JavaRDD&lt;T&gt; distinct(int numPartitions)</code>	Returns a new RDD containing the distinct elements in this RDD.
<code>JavaPairRDD&lt;K,Iterable&lt;V&gt;&gt; groupByKey(int numPartitions)</code>	Returns <code>(K,Seq[V])</code> and combines the values of the same key to a set.
<code>JavaPairRDD&lt;K,V&gt; reduceByKey(Function2&lt;V,V,V&gt; func, int numPartitions)</code>	Invokes Function on the values of the same key.
<code>JavaPairRDD&lt;K,V&gt; sortByKey(boolean ascending, int numPartitions)</code>	Sorts by key in ascending or descending order. Ascending is of the boolean type.
<code>JavaPairRDD&lt;K,scala.Tuple2&lt;V,W&gt;&gt; join(JavaPairRDD&lt;K,W&gt; other)</code>	Returns the dataset of <code>(K,(V,W))</code> when the <code>(K,V)</code> and <code>(K,W)</code> datasets exist. <b>numTasks</b> indicates the number of concurrent tasks.



Method	Description
JavaPairRDD<K,scala.Tuple2<Iterable<V>,Iterable<W>>> cogroup(JavaPairRDD<K,W> other, int numPartitions)	Returns the dataset of <K,scala.Tuple2<Iterable<V>,Iterable<W>>> when the (K,V) and (K,W) datasets exist. <b>numTasks</b> indicates the number of concurrent tasks.
JavaPairRDD<T,U> cartesian(JavaRDDLike<U,?> other)	Returns the Cartesian product of the RDD and other RDDs.

**Table 30-11** Action

Method	Description
T reduce(Function2<T,T,T> f)	Invokes Function2 on elements of the RDD.
java.util.List<T> collect()	Returns an array that contains all of the elements in this RDD.
long count()	Returns the number of elements in the dataset.
T first()	Returns the first element in the dataset.
java.util.List<T> take(int num)	Returns the first <i>n</i> elements in this RDD.
java.util.List<T> takeSample(boolean withReplacement, int num, long seed)	Samples the dataset randomly and returns a dataset of num elements. <b>withReplacement</b> indicates whether replacement is used.
void saveAsTextFile(String path, Class<? extends org.apache.hadoop.io.compress.CompressionCodec> codec)	Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file.
java.util.Map<K,Object> > countByKey()	Counts the appearance times of each key.
void foreach(VoidFunction<T> f)	Runs a function <i>func</i> on each element of the RDD.
java.util.Map<T,Long> countByValue()	Counts the times that each element of the RDD occurs.

**Table 30-12** New APIs of Spark core

API	Description
public java.util.concurrent.atomic.AtomicBoolean isSparkContextDown()	<p>Determines whether sparkContext is shut down completely. The initial value is <b>false</b>.</p> <p>The value <b>true</b> indicates that sparkContext is shut down completely.</p> <p>The value <b>false</b> indicates that sparkContext is not shut down.</p> <p>For example, <b>jsc.sc().isSparkContextDown().get() == true</b> indicates that sparkContext is shut down completely.</p>

## Spark Streaming Common Interfaces

Spark Streaming mainly uses the following classes:

- **JavaStreamingContext**: main entrance of the Spark Streaming, which is used to provide methods for creating DStream. Intervals by batch need to be set in the input parameter.
- **JavaDStream**: a type of data which indicates the RDDs continuous sequence. It indicates the continuous data flow.
- **JavaPairDStream**: interface of KV DStream, which is used to provide the reduceByKey and join operations.
- **JavaReceiverInputDStream<T>**: specifies any inflow accepting data from the network.

Common methods of Spark Streaming are the same as those of Spark Core. The following table describes some special Spark Streaming methods.

**Table 30-13** Spark Streaming methods

Method	Description
JavaReceiverInputDStream<java.lang.String> socketStream(java.lang.String hostname,int port)	Creates an inflow to receive data from the corresponding hostname and port through the TCP socket. Received data is resolved to the UTF8 format. The default storage level is the Memory+Disk.
JavaDStream<java.lang.String> textFileStream(java.lang.String directory)	Creates an inflow to detect new files compatible with the Hadoop file system, and read it as a text file. The directory of the input parameter is an HDFS directory.
void start()	Starts the Spark Streaming computing.

Method	Description
void awaitTermination()	Terminates the await of the process, which is similar to pressing <b>Ctrl+C</b> .
void stop()	Stops the Spark Streaming computing.
<T> JavaDStream<T> transform(java.util.List<JavaDStream<?>> dstreams,Function2<java.util.List<JavaRDD<?>>,Time,JavaRDD<T>>> transformFunc)	Performs the Function operation on each RDD to obtain a new DStream. In this function, the sequence of the JavaRDDs must be the same as the corresponding DStreams.
<T> JavaDStream<T> union(JavaDStream<T> first,java.util.List<JavaDStream<T>> rest)	Creates a unified Dstream from multiple DStreams with the same type and sliding window.

**Table 30-14** Spark Streaming enhancement interface

Method	Description
JAVADStreamKafkaWriter.writeToKafka()	Writes data from DStream into Kafka in batch.
JAVADStreamKafkaWriter.writeToKafkaBySingle()	Writes data from DStream into Kafka one by one.

## Spark SQL Common Interfaces

Spark SQL mainly uses the following classes:

- SQLContext: main entrance of the Spark SQL function and DataFrame.
- DataFrame: a distributed dataset organized by naming columns.
- DataFrameReader: interface for loading the DataFrame from external storage systems.
- DataFrameStatFunctions: implementation the statistic function of the DataFrame.
- UserDefinedFunction: function defined by users.

Common Actions methods are described in the following table.

**Table 30-15** Spark SQL methods

Method	Description
Row[] collect()	Returns an array containing all DataFrame columns.

Method	Description
long count()	Returns the number of DataFrame rows.
DataFrame describe(java.lang.String... cols)	Counts the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
Row first()	Returns the first row.
Row[] head(int n)	Returns the first <i>n</i> rows.
void show()	Displays the first 20 rows in table.
Row[] take(int n)	Returns the first <i>n</i> rows in the DataFrame.

**Table 30-16** Basic DataFrame Functions

Method	Description
void explain(boolean extended)	Prints the logical plan and physical plan of the SQL.
void printSchema()	Prints the schema information to the console.
registerTempTable	Registers the DataFrame as a temporary table, whose period is bound to the SQLContext.
DataFrame toDF(java.lang.String... colNames)	Returns a DataFrame whose columns are renamed.
DataFrame sort(java.lang.String sortCol,java.lang.String... sortCols)	Sorts columns in ascending or descending orders based on different columns.
GroupedData rollup(Column... cols)	Performs multiple-dimension crankback on the specified columns in the DataFrame.

### 30.5.1.2 Scala

To avoid API compatibility or reliability issues after updates to the open-source Spark, it is advisable to use APIs of the version you are currently using.

## Spark Core Common Interfaces

Spark mainly uses the following classes:

- **SparkContext**: external interface of Spark, which is used to provide the functions of Spark for Scala applications that invoke this class, for example, connecting Spark clusters and generating RDDs.

- SparkConf: Spark application configuration class, which is used to configure the application name, execution model, and executor memory.
- Resilient Distributed Dataset (RDD): defines the RDD class in the Spark application. The class provides the data collection operation methods, such as map and filter.
- PairRDDFunctions: provides computation operations for the RDD data of the key-value pair, such as groupByKey.
- Broadcast: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- StorageLevel: data storage levels, including memory (MEMORY\_ONLY), disk (DISK\_ONLY), and memory+disk (MEMORY\_AND\_DISK).

RDD supports two types of operations, transformation and action. [Table 30-17](#) and [Table 30-18](#) describe the common methods.

**Table 30-17** Transformation

Method	Description
map[U](f: (T) => U): RDD[U]	Returns a new RDD by applying a function to all elements of this RDD.
filter(f: (T) => Boolean): RDD[T]	Invokes the f method for all RDD elements to generate a satisfied data set that is returned in the form of RDD.
flatMap[U](f: (T) => TraversableOnce[U]) (implicit arg0: ClassTag[U]): RDD[U]	Returns a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
sample(withReplacement: Boolean, fraction: Double, seed: Long = Utils.random.nextLong): RDD[T]	Returns a sampled subset of this RDD.
union(other: RDD[T]): RDD[T]	Returns a new RDD, contains source RDD and the group of RDD's elements.
distinct([numPartitions: Int]): RDD[T]	Returns a new RDD containing the distinct elements in this RDD.
groupByKey(): RDD[(K, Iterable[V])]	Returns (K,Iterable[V]) and combines the values of the same key to a set.
reduceByKey(func: (V, V) => V[, numPartitions: Int]): RDD[(K, V)]	Invokes func on the values of the same key.

Method	Description
sortByKey(ascending: Boolean = true, numPartitions: Int = self.partitions.length): RDD[(K, V)]	Sorts by key in ascending or descending order. Ascending is of the boolean type.
join[W](other: RDD[(K, W)][, numPartitions: Int]): RDD[(K, (V, W))]	Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. <b>numPartitions</b> indicates the number of concurrent tasks.
cogroup[W](other: RDD[(K, W)], numPartitions: Int): RDD[(K, (Iterable[V], Iterable[W]))]	Returns the dataset of (K, (Iterable[V], Iterable[W])) when the (K,V) and (K,W) datasets of two key-value pairs exist. <b>numPartitions</b> indicates the number of concurrent tasks.
cartesian[U](other: RDD[U])(implicit arg0: ClassTag[U]): RDD[(T, U)]	Returns the Cartesian product of the RDD and other RDDs.

**Table 30-18** Action

API	Description
reduce(f: (T, T) => T):	Invokes f on elements of the RDD.
collect(): Array[T]	Returns an array that contains all of the elements in this RDD.
count(): Long	Returns the number of elements in the dataset.
first(): T	Returns the first element in the dataset.
take(num: Int): Array[T]	Returns the first <i>n</i> elements in this RDD.
takeSample(withReplacement: Boolean, num: Int, seed: Long = Utils.random.nextLong): Array[T]	Samples the dataset randomly and returns a dataset of num elements. <b>withReplacement</b> indicates whether replacement is used.
saveAsTextFile(path: String): Unit	Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file.

API	Description
saveAsSequenceFile(path: String, codec: Option[Class[_ <: CompressionCodec]] = None): Unit	This API can be used only on the key-value pair, and then it generates SequenceFile and writes the file to the local or Hadoop file system.
countByKey(): Map[K, Long]	Counts the appearance times of each key.
foreach(func: (T) => Unit): Unit	Applies a function f to all elements of this RDD.
countByValue()(implicit ord: Ordering[T] = null): Map[T, Long]	Counts the times that each element of the RDD occurs.

**Table 30-19** New APIs of Spark core

API	Description
isSparkContextDown: AtomicBoolean	<p>Determines whether sparkContext is shut down completely. The initial value is <b>false</b>.</p> <p>The value <b>true</b> indicates that sparkContext is shut down completely.</p> <p>The value <b>false</b> indicates that sparkContext is not shut down.</p> <p>For example, <b>sc.isSparkContextDown.get() == true</b> indicates that sparkContext is shut down completely.</p>

## Spark Streaming Common Interfaces

Spark Streaming mainly uses the following classes:

- StreamingContext: main entrance of the Spark Streaming, which is used to provide methods for creating DStream. Intervals by batch need to be set in the input parameter.
- dstream.DStream: a type of data which indicates the RDDs continuous sequence. It indicates the continuous data flow.
- dstream.PariDStreamFunctions: Dstream of key-value, common operations are groupByKey and reduceByKey.

The cooperated Java API of Spark Streaming are JavaStreamingContext, JavaDStream, JavaPairDStream.

Common methods of Spark Streaming are the same as those of Spark Core. The following table describes some special Spark Streaming methods.

**Table 30-20** Spark Streaming methods

Method	Description
socketTextStream(hostname: String, port: Int, storageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK_SER_2): ReceiverInputDStream[String]	Creates an input stream from the TCP source host:port.
start():Unit	Starts the Spark Streaming computing.
awaitTermination(timeout: long):Unit	Terminates the await of the process, which is similar to pressing <b>Ctrl+C</b> .
stop(stopSparkContext: Boolean, stopGracefully: Boolean): Unit	Stops the Spark Streaming computing.
transform[T](dstreams: Seq[DStream[_]], transformFunc: (Seq[RDD[_], Time) ? RDD[T])(implicit arg0: ClassTag[T]): DStream[T]	Performs the function operation on each RDD to obtain a new DStream.
UpdateStateByKey(func)	Updates the status of DStream. To use this method, you need to define the state and state update functions.
window(windowLength, slideInterval)	Generates a new DStream by batch calculating according to the window of the source DStream.
countByWindow(windowLength, slideInterval)	Returns the number of sliding window elements in the stream.
reduceByWindow(func, windowLength, slideInterval)	When the key-value pair of DStream is invoked, a new key-value pair of DStream is returned. The value of each key is obtained by aggregating the reduce function in batches in the sliding window.
join(otherStream, [numTasks])	Performs a join operation between different Spark Streamings.
DStreamKafkaWriter.writeToKafka()	Writes data from DStream into Kafka in batch.



Method	Description
DStreamKafkaWriter.writeToKafkaBySingle()	Writes data from DStream into Kafka one by one.

**Table 30-21** Spark Streaming enhancement interface

Method	Description
DStreamKafkaWriter.writeToKafka()	Writes data from DStream into Kafka in batch.
DStreamKafkaWriter.writeToKafkaBySingle()	Writes data from DStream into Kafka one by one.

## SparkSQL Common Interfaces

Spark SQL mainly uses the following classes:

- SQLContext: main entrance of the Spark SQL function and DataFrame.
- DataFrame: a distributed dataset organized by naming columns.
- HiveContext: An instance of the Spark SQL execution engine that integrates with data stored in Hive.

**Table 30-22** Common Actions methods

Method	Description
collect(): Array[Row]	Returns an array containing all DataFrame columns.
count(): Long	Returns the number of DataFrame rows.
describe(cols: String*): DataFrame	Counts the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
first(): Row	Returns the first row.
Head(n:Int): Row	Returns the first <i>n</i> rows.
show(numRows: Int, truncate: Boolean): Unit	Displays DataFrame in a table.
take(n:Int): Array[Row]	Returns the first <i>n</i> rows in the DataFrame.

**Table 30-23** Basic DataFrame Functions

Method	Description
explain(): Unit	Prints the logical plan and physical plan of the SQL.
printSchema(): Unit	Prints the schema information to the console.
registerTempTable(tableName: String): Unit	Registers the DataFrame as a temporary table, whose period is bound to the SQLContext.
toDF(colNames: String*): DataFrame	Returns a DataFrame whose columns are renamed.

### 30.5.1.3 Python

To avoid API compatibility or reliability issues after updates to the open-source Spark, it is advisable to use APIs of the version you are currently using.

## Spark Core Common Interfaces

Spark mainly uses the following classes:

- `pyspark.SparkContext`: external API of Spark. It provides the functions of Spark for Python applications that invoke this class, for example, connecting Spark clusters, creating RDDs, and broadcasting variables.
- `pyspark.SparkConf`: Spark application configuration class. It is used to set an application name, execution mode, and executor memory.
- `pyspark.RDD`: class used to define the RDD in the Spark application. The class provides the data collection operation methods, such as map and filter.
- `pyspark.Broadcast`: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- `pyspark.StorageLevel`: data storage levels, including memory (`MEMORY_ONLY`), disk (`DISK_ONLY`), and memory+disk (`MEMORY_AND_DISK`).
- `pyspark.sql.SQLContext`: Main entry point for SparkSQL functionality. It can be used to create DataFrame, register DataFrame as a table, and execute SQL on a table.
- `pyspark.sql.DataFrame`: A distributed collection of data grouped into named columns. A DataFrame is equivalent to a relational table in Spark SQL, and can be created using various functions in SQLContext.
- `pyspark.sql.DataFrameNaFunctions`: Functionality for working with missing data in DataFrame.
- `pyspark.sql.DataFrameStatFunctions`: Functionality for statistic functions with DataFrame.

The RDD supports two types of operations, transformation and action. [Table 30-24](#) and [Table 30-25](#) show the common methods.

**Table 30-24** Transformation

Method	Description
map(f, preservesPartitioning=False)	Returns a new RDD by applying a function to all elements of this RDD.
filter(f)	Invokes the Func method for all RDD elements to generate a satisfied data set that is returned in the form of RDD.
flatMap(f, preservesPartitioning=False)	Returns a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
sample(withReplacement, fraction, seed=None)	Returns a sampled subset of this RDD.
union(rdds)	Returns a new RDD, contains source RDD and the group of RDD's elements.
distinct([numPartitions: Int]): RDD[T]	Returns a new RDD containing the distinct elements in this RDD.
groupByKey(): RDD[(K, Iterable[V])]	Returns (K,Iterable[V]) and combines the values of the same key to a set.
reduceByKey(func, numPartitions=None)	Invokes Func on the values of the same key.
sortByKey(ascending=True, numPartitions=None, keyfunc=function <lambda>)	Sorts by key in ascending or descending order. Ascending is of the boolean type.
join(other, numPartitions)	Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. <b>numPartitions</b> indicates the number of concurrent tasks.
cogroup(other, numPartitions)	Returns the dataset of (K, (Iterable[V], Iterable[W])) when the (K,V) and (K,W) datasets of two key-value pairs exist. <b>numPartitions</b> indicates the number of concurrent tasks.
cartesian(other)	Returns the Cartesian product of the RDD and other RDDs.

**Table 30-25** Action

API	Description
reduce(f)	Invokes Func on elements of the RDD.

API	Description
collect()	Returns an array that contains all of the elements in this RDD.
count()	Returns the number of elements in the dataset.
first()	Returns the first element in the dataset.
take(num)	Returns the first num elements of the RDD.
takeSample(withReplacement, num, seed)	Samples the dataset randomly and returns a dataset of num elements. <b>withReplacement</b> indicates whether replacement is used.
saveAsTextFile(path, compressionCodecClass)	Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file.
saveAsSequenceFile(path, compressionCodecClass=None)	This API can be used only on the key-value pair, and then it generates SequenceFile and writes the file to the local or Hadoop file system.
countByKey()	Counts the appearance times of each key.
foreach(func)	Applies a function f to all elements of this RDD.
countByValue()	Counts the times that each value of the RDD occurs.

## Spark Streaming Common Interfaces

Spark Streaming mainly uses the following classes:

- `pyspark.streaming.StreamingContext`: main entrance of Spark Streaming. It provides methods for creating the DStream. A batch interval needs to be set in the input parameter.
- `pyspark.streaming.DStream`: A Discretized Stream (DStream), the basic abstraction in Spark Streaming, is a continuous sequence of RDDs (of the same type) representing a continuous stream of data.
- `dstream.PariDStreamFunctions`: Dstream of key-value, common operations are `groupByKey` and `reduceByKey`.

The cooperated Java API of Spark Streaming are `JavaStreamingContext`, `JavaDStream`, `JavaPairDStream`.

Common methods of Spark Streaming are the same as those of Spark Core. The following table describes some special Spark Streaming methods.

**Table 30-26** Spark Streaming common interfaces

Method	Description
socketTextStream(hostname, port, storageLevel)	Creates an input stream from the TCP source host:port.
start()	Starts the Spark Streaming computing.
awaitTermination(timeout)	Terminates the await of the process, which is similar to pressing <b>Ctrl+C</b> .
stop(stopSparkContext, stopGracefully)	Stops Spark Streaming computing. <b>stopSparkContext</b> is used to determine whether SparkContext needs to be terminated. <b>StopGracefully</b> is used to determine whether to wait for all the received data to be processed.
updateStateByKey(func)	Updates the status of DStream. To use this method, you need to define the state and state update functions.
window(windowLength, slideInterval)	Generates a new DStream by batch calculating according to the window of the source DStream.
countByWindow(windowLength, slideInterval)	Returns the number of sliding window elements in the stream.
reduceByWindow(func, windowLength, slideInterval)	When the key-value pair of DStream is invoked, a new key-value pair of DStream is returned. The value of each key is obtained by aggregating the reduce function in batches in the sliding window.
join(other, numPartitions)	Performs a join operation between different Spark Streamings.

## SparkSQL Common Interfaces

Spark SQL mainly uses the following classes:

- pyspark.sql.SQLContext: main entrance of the Spark SQL function and DataFrame.
- pyspark.sql.DataFrame: a distributed dataset organized by naming columns.
- pyspark.sql.HiveContext: main entrance for obtaining data stored in Hive.
- pyspark.sql.DataFrameStatFunctions: Functionality for statistic functions with DataFrame.
- pyspark.sql.functions: A collection of builtin functions.
- pyspark.sql.Window: window function provided by SQL

**Table 30-27** Spark SQL common Actions

Method	Description
collect()	Returns an array containing all DataFrame columns.
count()	Returns the number of DataFrame rows.
describe()	Counts the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
first()	Returns the first row.
head(n)	Returns the first <i>n</i> rows.
show()	Displays DataFrame in a table.
take(num)	Returns the first num rows in the DataFrame.

**Table 30-28** Basic DataFrame Functions

Method	Description
explain()	Prints the logical plan and physical plan of the SQL.
printSchema()	Prints the schema information to the console.
registerTempTable(name)	Registers the DataFrame as a temporary table, whose period is bound to the SQLContext.
toDF()	Returns a DataFrame whose columns are renamed.

### 30.5.1.4 REST API

#### Function Description

The Spark REST API presents some web UI metrics in the JSON format, providing users with a simpler method to create new visualization and monitoring tools. The REST API can be used to query information about running and historical applications. The open-source Spark REST API allows users to query information about Jobs, Stages, Storage, Environment, and Executors. In the FusionInsight version, the REST API used to query SQL, JDBC server, and Streaming information is added. For more information about the open-source REST API, see <https://spark.apache.org/docs/3.1.1/monitoring.html#rest-api>.

#### Preparing Running Environment

Install the FusionInsight client. Install a FusionInsight client on the node. For example, install the client in the **/opt/client** directory.

## REST API

You can use the following commands to dodge the REST API filter and directly obtain the application information:

---

### NOTICE

- In security mode, the JobHistory supports only the HTTPS protocol. Therefore, use the HTTPS protocol in the URL of the following command.
- In security mode, you need to set **spark.ui.customErrorPage** to **false** and restart Spark2x. Change the value of this parameter for the JobHistory2x, JDBCServer2x, and SparkResource2x instances.

---

### NOTE

Perform the following steps to upgrade the curl version on the node:

1. Download the curl installation package from the following website: <http://curl.haxx.se/download/>
2. Run the following command to decompress the installation package:

```
tar -xzvf curl-x.x.x.tar.gz
```

3. Run the following commands to overwrite the old curl version with the new one:

```
cd curl-x.x.x
```

```
./configure
```

```
make
```

```
make install
```

4. Run the following command to update the dynamic link library of curl:

```
ldconfig
```

5. After the installation is successful, log in to the node again and run the following command to check whether the curl version is successfully updated:

```
curl --version
```

- Obtaining information about all applications on the JobHistory node:

- Command:

```
curl -k -i --negotiate -u: "https://192.168.227.16:18080/api/v1/applications"
```

**192.168.227.16** indicates the service IP address of the JobHistory node and **18080** indicates the port number of the JobHistory node.

- Command output:

```
[{
 "id" : "application_1517290848707_0008",
 "name" : "Spark Pi",
 "attempts" : [{
 "startTime" : "2018-01-30T15:05:37.433CST",
 "endTime" : "2018-01-30T15:06:04.625CST",
 "lastUpdated" : "2018-01-30T15:06:04.848CST",
 "duration" : 27192,
 "sparkUser" : "sparkuser",
 "completed" : true,
 "startTimeEpoch" : 1517295937433,
 "endTimeEpoch" : 1517295964625,
 "lastUpdatedEpoch" : 1517295964848
 }]
}, {
 "id" : "application_1517290848707_0145",
```

```
"name" : "Spark shell",
"attempts" : [{
 "startTime" : "2018-01-31T15:20:31.286CST",
 "endTime" : "1970-01-01T07:59:59.999CST",
 "lastUpdated" : "2018-01-31T15:20:47.086CST",
 "duration" : 0,
 "sparkUser" : "admintest",
 "completed" : false,
 "startTimeEpoch" : 1517383231286,
 "endTimeEpoch" : -1,
 "lastUpdatedEpoch" : 1517383247086
}]
}]
```

- Analysis:

With this command, you can query information about all Spark applications in the current cluster, including running applications and the completed applications. [Table 30-29](#) provides information about each application.

**Table 30-29** Parameter description

Parameter	Description
id	Application ID.
name	Application name.
attempts	Attempts executed by the application, including the attempt start time, attempt end time, user who initiates the attempts, and status indicating whether the attempts are completed.

- Obtaining information about a specific application on the JobHistory node:

- Command:

```
curl -k -i --negotiate -u: "https://192.168.227.16:18080/api/v1/applications/
application_1517290848707_0008"
```

**192.168.227.16** indicates the service IP address of the JobHistory node, **18080** indicates the port number of the JobHistory node, and **application\_1517290848707\_0008** indicates the application ID.

- Command output:

```
{
 "id" : "application_1517290848707_0008",
 "name" : "Spark Pi",
 "attempts" : [{
 "startTime" : "2018-01-30T15:05:37.433CST",
 "endTime" : "2018-01-30T15:06:04.625CST",
 "lastUpdated" : "2018-01-30T15:06:04.848CST",
 "duration" : 27192,
 "sparkUser" : "sparkuser",
 "completed" : true,
 "startTimeEpoch" : 1517295937433,
 "endTimeEpoch" : 1517295964625,
 "lastUpdatedEpoch" : 1517295964848
 }]
}
```

- Analysis:



With this command, you can query information about a Spark application. [Table 30-29](#) provides information about the application.

- Obtaining information about the executor of a running application:

- Command of alive executors list:

```
curl -k -i --negotiate -u: "https://192.168.169.84:8090/proxy/
application_1478570725074_0046/api/v1/applications/application_1478570725074_0046/
executors"
```

- Command of all executors (alive and dead) list:

```
curl -k -i --negotiate -u: "https://192.168.169.84:8090/proxy/
application_1478570725074_0046/api/v1/applications/application_1478570725074_0046/
allexecutors"
```

**192.168.169.84** indicates the service IP address of the master node of the ResourceManager, **8090** indicates the port number of the ResourceManager, and **application\_1478570725074\_0046** indicates the application ID in YARN.

- Command output:

```
[{
 "id" : "driver",
 "hostPort" : "192.168.169.84:23886",
 "isActive" : true,
 "rddBlocks" : 0,
 "memoryUsed" : 0,
 "diskUsed" : 0,
 "activeTasks" : 0,
 "failedTasks" : 0,
 "completedTasks" : 0,
 "totalTasks" : 0,
 "totalDuration" : 0,
 "totalInputBytes" : 0,
 "totalShuffleRead" : 0,
 "totalShuffleWrite" : 0,
 "maxMemory" : 278019440,
 "executorLogs" : { }
}, {
 "id" : "1",
 "hostPort" : "192.168.169.84:23902",
 "isActive" : true,
 "rddBlocks" : 0,
 "memoryUsed" : 0,
 "diskUsed" : 0,
 "totalCores" : 1,
 "maxTasks" : 1,
 "activeTasks" : 0,
 "failedTasks" : 0,
 "completedTasks" : 0,
 "totalTasks" : 0,
 "totalDuration" : 0,
 "totalGCTime" : 139,
 "totalInputBytes" : 0,
 "totalShuffleRead" : 0,
 "totalShuffleWrite" : 0,
 "maxMemory" : 555755765,
 "executorLogs" : {
 "stdout" : "https://XTJ-224:8044/node/containerlogs/
container_1478570725074_0049_01_000002/admin/stdout?start=-4096",
 "stderr" : "https://XTJ-224:8044/node/containerlogs/
container_1478570725074_0049_01_000002/admin/stderr?start=-4096"
 }
}]
```

- Analysis:

With this command, you can query information about all executors including drivers of the current application. [Table 30-30](#) provides basic information about each executor.

**Table 30-30** Parameter description

Parameter	Description
id	Executor ID.
hostPort	IP address and port number of the node where the Executor resides. Format: <i>IP address:port number</i> .
executorLogs	Path to Executor logs.

## Enhanced REST API

- SQL related: obtaining all the SQL statements and those with the longest execution time.
  - SparkUI command:  

```
curl -k -i --negotiate -u: "https://192.168.195.232:8090/proxy/application_1476947670799_0053/api/v1/applications/application_1476947670799_0053/SQL"
```

**192.168.195.232** indicates the service IP address of the master node of the ResourceManager, **8090** indicates the port number of the ResourceManager, and **application\_1476947670799\_0053** indicates the application ID in YARN.

### NOTE

You can add parameters to the URL after the command to search for the corresponding SQL statements.

For example, run the following command to view 100 SQL statements:

```
curl -k -i --negotiate -u: "https://192.168.195.232:8090/proxy/application_1476947670799_0053/api/v1/applications/application_1476947670799_0053/SQL?limit=100"
```

Run the following command to view running parameters:

```
curl -k -i --negotiate -u: "https://192.168.195.232:8090/proxy/application_1476947670799_0053/api/v1/applications/application_1476947670799_0053/SQL?completed=false"
```

- JobHistory command:  

```
curl -k -i --negotiate -u: "https://192.168.227.16:18080/api/v1/applications/application_1478570725074_0004/SQL"
```

**192.168.227.16** indicates the service IP address of the JobHistory node, **18080** indicates the port number of the JobHistory node, and **application\_1478570725074\_0004** indicates the application ID.
- Command output:

The command output of the SparkUI and JobHistory commands is as follows:

```
{
 "longestDurationOfCompletedSQL" : [{
 "id" : 0,
 "status" : "COMPLETED",
 "description" : "getCallSite at SQLExecution.scala:48",
 "submissionTime" : "2016/11/08 15:39:00",
 "duration" : "2 s",
 "runningJobs" : [],
 "succeededJobs" : [0],
 "failedJobs" : []
 }],
}
```

```
"sqls" : [{
 "id" : 0,
 "status" : "COMPLETED",
 "description" : "getCallSite at SQLExecution.scala:48",
 "submissionTime" : "2016/11/08 15:39:00",
 "duration" : "2 s",
 "runningJobs" : [],
 "succeededJobs" : [0],
 "failedJobs" : []
}]
}
```

– Analysis:

After running this command, you can obtain all the SQL statements executed by the current application (the **sqls** part of the command output) and the SQL statements with the longest execution time (the **longestDurationOfCompletedSQL** part of the command output). The information about each SQL statement is listed in [Table 30-31](#).

**Table 30-31** Parameter description

Parameter	Description
id	SQL statement ID.
status	Execution status of the SQL statement, which can be: running, completed, and failed.
runningJobs	Jobs that are being executed generated by the SQL statement.
succeededJobs	Jobs that are successfully executed generated by the SQL statement.
failedJobs	Job that fails to be executed generated by the SQL statement.

- JDBC server related: obtaining the number of sessions, number of being-executed SQL statements, information about all sessions, and information about SQL statements.

– Command:

```
curl -k -i --negotiate -u: "https://192.168.195.232:8090/proxy/
application_1476947670799_0053/api/v1/applications/application_1476947670799_0053/
sqlserver"
```

**192.168.195.232** indicates the service IP address of the master node of the ResourceManager, **8090** indicates the port number of the ResourceManager, and **application\_1476947670799\_0053** indicates the application ID in YARN.

– Command output:

```
{
 "sessionNum" : 1,
 "runningSqlNum" : 0,
 "sessions" : [{
 "user" : "spark",
 "ip" : "192.168.169.84",
 "sessionId" : "9dfec575-48b4-4187-876a-71711d3d7a97",
 "startTime" : "2016/10/29 15:21:10",
 "finishTime" : "",
```

```

"duration" : "1 minute 50 seconds",
"totalExecute" : 1
}],
"sqs" : [{
 "user" : "spark",
 "jobId" : [],
 "groupId" : "e49ff81a-230f-4892-a209-a48abea2d969",
 "startTime" : "2016/10/29 15:21:13",
 "finishTime" : "2016/10/29 15:21:14",
 "duration" : "555 ms",
 "statement" : "show tables",
 "state" : "FINISHED",
 "detail" : "== Parsed Logical Plan ==\nShowTablesCommand None\n\n== Analyzed Logical
Plan ==\ntableName: string, isTemporary: boolean\nShowTablesCommand None\n\n== Cached
Logical Plan ==\nShowTablesCommand None\n\n== Optimized Logical Plan ==
\nShowTablesCommand None\n\n== Physical Plan ==\nExecutedCommand
ShowTablesCommand None\n\nCode Generation: true"
}]
}

```

– Analysis:

After running this command, you can query the number of sessions in the current JDBC application, number of being-executed SQL statements, and information about all sessions and SQL statements. The information about each session is listed in [Table 30-32](#), and the information about each SQL statement is listed in [Table 30-33](#):

**Table 30-32** Session parameter description

Parameter	Description
user	User to whom the session connects.
ip	IP address of the node where the session resides.
sessionId	Session ID.
startTime	Time when the session starts the connection.
finishTime	Time when the session ends the connection.
duration	Connection duration of the session.
totalExecute	Number of SQL statements executed by the session.

**Table 30-33** SQL parameter description

Parameter	Description
user	User who executes the SQL statement.

Parameter	Description
jobId	IDs of jobs contained in the SQL statement.
groupId	ID of the group where the SQL statement resides.
startTime	Start time.
finishTime	End time.
duration	SQL statement execution duration.
statement	SQL statement.
detail	Logical/Physical plan.

- JDBC API enhancement cancels the SQL statement that is being executed by using the execution ID obtained from the beeline.
  - Commands for the operation:  

```
curl -k -i --negotiate -X PUT -u: "https://192.168.195.232:8090/proxy/application_1477722033672_0008/api/v1/applications/application_1477722033672_0008/cancel/execution?executionId=8"
```
  - Command output:  
Cancel the job whose execution ID is 8.
  - Remarks:  
Run the SQL statement in spark-beeline. If the SQL statement generates a Spark task, the execution ID of the SQL statement will be printed in beeline. To cancel the execution of the SQL statement, run the preceding command.
- Streaming related: obtaining the average input frequency, average scheduling delay, average execution duration, and average value of the overall delay.
  - Command:  

```
curl -k -i --negotiate -u: "https://192.168.195.232:8090/proxy/application_1477722033672_0008/api/v1/applications/application_1477722033672_0008/streaming/statistics"
```

**192.168.195.232** indicates the service IP address of the master node of the ResourceManager, **8090** indicates the port number of the ResourceManager, and **application\_1477722033672\_0008** indicates the application ID in YARN.
  - Command:  

```
{
 "startTime": "2018-12-25T08:58:10.836GMT",
 "batchDuration": 1000,
 "numReceivers": 1,
 "numActiveReceivers": 1,
 "numInactiveReceivers": 0,
 "numTotalCompletedBatches": 373,
 "numRetainedCompletedBatches": 373,
 "numActiveBatches": 0,
 "numProcessedRecords": 1,
 "numReceivedRecords": 1,
 "avgInputRate": 0.002680965147453083,
 "avgSchedulingDelay": 14,
 "avgProcessingTime": 47,
```

```
"avgTotalDelay" : 62
}
```

– Analysis:

After running this command, you can query the average input frequency (unit: events/sec), average scheduling delay (unit: ms), average execution time (unit: ms), and average value of the total delay (unit: ms) of the current Streaming application.

## 30.5.2 Common CLIs

For details about how to use the Spark CLIs, visit the official website <http://spark.apache.org/docs/3.1.1/quick-start.html>.

### Common CLI

Common Spark CLIs are described as follows:

- **spark-shell**

It provides an easy way to learn APIs, which is similar to the tool for interactive data analysis. It supports two languages including Scala and Python. In the Spark directory, run the `./bin/spark-shell` command to access the interactive interface of Scala, obtain data from HDFS, and then perform the RDD.

For example: a row of codes can count all words in a file.

```
scala> sc.textFile("hdfs://10.96.1.57:9000//wordcount_data.txt").flatMap(l
=> l.split(" ")).map(w => (w,1)).reduceByKey(_+_).collect()
```

You can directly specify the Keytab and Principal in the command line to obtain authentication, and regularly update the keytab and authorized tokens to avoid the authentication expiry. The following command is used as an example.

```
spark-shell --principal spark2x/hadoop.<System domain name>@<System
domain name> --keytab ${BIGDATA_HOME}/
FusionInsight_Spark2x_8.1.0.1/install/FusionInsight-Spark2x-3.1.1/keytab/
spark2x/SparkResource/spark2x.keytab --master yarn
```

- **spark-submit**

It is used to submit the Spark application to the Spark cluster for running and return the running results. The class, master, jar and input parameter need to be specified.

For example: Run the GroupByTest example in the jar. There are four input parameters and the specified running mode of the cluster is local single platform.

```
./bin/spark-submit --class org.apache.spark.examples.GroupByTest --
master local[1] examples/jars/spark-examples_2.12-3.1.1-hw-ei-311001-
SNAPSHOT.jar 6 10 10 3
```

You can directly specify the Keytab and Principal in the command line to obtain authentication, and regularly update the keytab and authorized tokens to avoid the authentication expiry. The following command is used as an example.

```
spark-submit --class org.apache.spark.examples.GroupByTest --master
yarn --principal spark2x/hadoop.<System domain name>@<System domain
```

```
name> --keytab ${BIGDATA_HOME}/FusionInsight_Spark2x_8.1.0.1/install/
FusionInsight-Spark2x-3.1.1/keytab/spark2x/SparkResource/
spark2x.keytab examples/jars/spark-examples_2.12-3.1.1-hw-ei-311001-
SNAPSHOT.jar 6 10 10 3
```

- **spark-sql**

It is used to perform the Hive metadata service and query command lines in the local mode. If its logical plan needs to be queried, add the explain extended before the SQL statement.

For example:

**Select key from src group by key**

You can directly specify the Keytab and Principal in the command line to obtain authentication, and regularly update the keytab and authorized tokens to avoid the authentication expiry. The following command is used as an example.

```
spark-sql --principal spark2x/hadoop.<System domain name>@<System
domain name> --keytab ${BIGDATA_HOME}/
FusionInsight_Spark2x_8.1.0.1/install/FusionInsight-Spark2x-3.1.1/keytab/
spark2x/SparkResource/spark2x.keytab --master yarn
```

- **run-example**

It is used to run or debug the default example in the Spark open-source community.

For example: Run the SparkPi.

```
./run-example SparkPi 100
```

## 30.5.3 JDBCServer Interface

### Overview

The JDBCServer is another implement of HiveServer2 in the Hive. The Spark SQL is used to process the SQL statement at its bottom. Therefore, the JDBCServer has better performance than the Hive.

The JDBCServer is a JDBC interface. Users can log in to the JDBCServer and access the Spark SQL data through the JDBC. When the JDBCServer is started, a Spark SQL application is started, and the clients connected through the JDBC share the resources in this application. That is, various users can share data. When the JDBCServer is started, a listener is also started to wait for the connection of the JDBC client and submit the query after the connection. Therefore, during the configuration of the JDBCServer, at least the host name and port of the JDBCServer must be configured. If Hive data is required, the uris of the hive metastore needs to be provided.

By default, the JDBCServer starts a JDBC service in port 10000 of the installation node. It can be connected by using the Beeline or JDBC client code to run the SQL command.

For other information about the JDBCServer, visit the Spark official website: <http://spark.apache.org/docs/3.1.1/sql-programming-guide.html#distributed-sql-engine>.

## Beeline

For connection methods of the Beeline provided by the open-source community, visit <https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients>.

To solve the connection problem in two scenarios of the Beeline, the authentication information is added in the Beeline connection. The **user.keytab** and **user.principal** parameters are added in the URL of the JDBC. When the key tab expires, the login information of the client can be read automatically and the connection succeeds again.

Users do not want to perform the key tab authentication by running the **init** command because the key tab expires every 24 hours. The Keytab file and principal information can be obtained from the administrator. The following command is used as a connection example of Beeline.

```
sh CLIENT_HOME/spark/bin/beeline -u "jdbc:hive2://
<zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<
zkNode3_Port>;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=spark
thriftserver2x;user.principal=spark2x/hadoop.<system domain name>@<system
domain name>;sasLQop=auth-conf;auth=KERBEROS;principal=spark2x/
hadoop.<system domain name>@<system domain name>";"
```

### NOTE

- **<zkNode1\_IP>:<zkNode1\_Port>,<zkNode2\_IP>:<zkNode2\_Port>,<zkNode3\_IP>:<zkNode3\_Port>** indicates the URL of ZooKeeper. Multiple URLs is separated by comma. For example: **192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181**.
- **sparkthriftserver2x** indicates the directory in Zookeeper where a random JDBCServer instance is selected for the connection to the client.

## JDBC Client Codes

Log in to the JDBCServer by using the JDBC client codes and access the Spark SQL data. For details, see [Accessing the Spark SQL Through JDBC](#).

## Enhanced Features

Compared with the open-source community, Huawei provides two enhanced features: the JDBCServer HA solution and timeout of configuring the JDBCServer.

- The JDBCServer HA solution is described as follows:  
When multiple active nodes of JDBCServer provides services at the same time, a new client will be connected to another active node if a fault occurs on one node, ensuring continuous services for clusters. The operations by using the Beeline and JDBC client codes are the same. The operations by using the Beeline and JDBC client codes are the same.
- Configure the timeout of the connection between the client and JDBCServer.
  - Beeline  
In network congestion, this feature can avoid the suspending of Beeline due to timeless wait of the return from the server. The method is described as follows:  
When the Beeline is started, add **--socketTimeOut=n**. The n indicates the timeout waiting for the service return. The unit is second and the default



value is 0 (indicating never timing out). Set the maximum timeout waiting time as required.

– JDBC Client Codes

In the scenario of network congestion, this feature can avoid the suspending of the client due to limitless wait of the return of server. The method to use is shown as follows:

Before the obtaining of the JDBC by using the `DriverManager.getConnection` method, add the `DriverManager.setLoginTimeout(n)` method to configure the timeout length. `n` indicates the timeout length of waiting for the service return. The unit is second and the type is `Int`. The default value is `0` (indicating never timing out). Set the maximum timeout waiting time as required.

### 30.5.4 Structured Streaming Functions and Reliability

#### Functions Supported by Structured Streaming

1. ETL operations on streaming data are supported.
2. Schema inference and partitioning of streaming DataFrames or Datasets are supported.
3. Operations on the streaming DataFrames or Datasets are supported, including SQL-like operations without types (such as `select`, `where`, and `groupBy`) and RDD operations with types (such as `map`, `filter`, and `flatMap`).
4. Aggregation calculation based on Event Time and processing of delay data are supported.
5. Deduplication of streaming data is supported.
6. Status computing is supported.
7. Stream processing tasks can be monitored.
8. Batch-stream join and stream join are supported.

The following table lists the supported join operations.

Left Table	Right Table	Supported Join Type	Description
Static	Static	All types	In stream processing, join operations with no streaming data involved are supported.
Stream	Static	Inner	Supported, but stateless.
		Left Outer	Supported, but stateless.
		Right Outer	Not supported.
		Full Outer	Not supported.
Stream	Stream	Inner	Supported. The watermark or time range can be used to clear status of the left and right tables.

Left Table	Right Table	Supported Join Type	Description
		Left Outer	Conditionally supported. The watermark can be used to clear status of the left table, and watermark and time range must be used for the right table.
		Right Outer	Conditionally supported. The watermark can be used to clear status of the right table, and watermark and time range must be used for the left table.
		Full Outer	Not supported.

## Functions Not Supported by Structured Streaming

1. Multi-stream aggregation is not supported.
2. The following operations of obtaining multiple rows are not supported: limit, first, and take.
3. Distinct is not supported.
4. Sorting is supported only when output mode is complete.
5. The external connection between streams and static data sets is conditionally supported.
6. Immediate query and result return on some data sets are not supported.
  - count(): Instead of returning a single count from streaming data sets, it uses ds.groupBy().count() to return a streaming data set containing the running count.
  - foreach(): The ds.writeStream.foreach(...) is used to replace it.
  - show(): The output console sink is used to replace it.

## Structured Streaming Reliability

Based on the checkpoint and WAL mechanisms, Structured Streaming provides end-to-end exactly-once error tolerance semantics for the sources that can be replayed and the idempotent sinks that support repeated processing.

1. You can enable the checkpoint function by setting option ("checkpointLocation", "checkpoint path") in the program.

When data is restored from checkpoint, the application or configuration may change. Some changes may result in the failure in restoring data from the checkpoint. The restrictions are as follows:

- a. The number or type of sources cannot be changed.
- b. The source type and query statement determine whether the source parameter can change. For example:
  - The parameters related to rate control can be added, deleted, or modified. For example:

- `spark.readStream.format("kafka").option("subscribe", "topic")` is changed to `spark.readStream.format("kafka").option("subscribe", "topic").option("maxOffsetsPerTrigger", ...)`.
- Unexpected problems may occur when the topic/file of the consumption is modified. For example:  
`spark.readStream.format("kafka").option("subscribe", "topic")` is changed to `spark.readStream.format("kafka").option("subscribe", "newTopic")`.
  - c. The type of sink changes. Specific sinks can be combined. The specific scenarios need to be verified. For example:
    - File sink can be changed to kafka sink. Kafka processes only new data.
    - Kafka sink cannot be changed to file sink.
    - Kafka sink can be changed to foreach sink, and vice versa.
  - d. The sink type and query statement determine whether the sink parameter can change. For example:
    - The output path of file sink cannot be changed.
    - The output topic of Kafka sink can be changed.
    - The custom operator code in foreach sink can be changed, but the change result depends on the user code.
  - e. The projection, filter, and map-like operations can be changed in some scenarios. For example:
    - Filters can be added and deleted. For example: `sdf.selectExpr("a")` is changed to `sdf.where(...).selectExpr("a").filter(...)`.
    - When Output schema is the same, projections can be changed. For example: `sdf.selectExpr("stringColumn AS json").writeStream` is changed to `sdf.select(to_json(...).as("json")).writeStream`.
    - If Output schema is different, projections can be changed in some conditions. For example: when `sdf.selectExpr("a").writeStream` is changed to `sdf.selectExpr("b").writeStream`, no error occurs only when the sink supports schema conversion from a to b.
  - f. In some scenarios, the status restoration will fail after status is changed.
    - Streaming aggregation: For example, in the `sdf.groupBy("a").agg(...)` operation, the type or quantity of the grouping key or the aggregation key cannot be changed.
    - Streaming deduplication: For example, in the `sdf.dropDuplicates("a")` operation, the type or quantity of the grouping key or the aggregation key cannot be changed.
    - Stream-stream join: For example, in the `sdf1.join(sdf2, ...)` operation, the schema of the association key cannot be changed, and the join type cannot be changed. Change of other join conditions may lead to uncertainty results.

- Any status computing: For example, in the `sdf.groupByKey(...).mapGroupsWithState(...)` or `sdf.groupByKey(...).flatMapGroupsWithState(...)` operation, the schema or timeout type of the user-defined status cannot be changed. Users can customize the state-mapping function change, but the change result depends on the user code. If schema changes are required, users can encode or decode the status data into binary data to support schema migration.

2. Fault tolerance list of Source

Sources	Supported Options	Fault Tolerance Supported	Description
File source	<p><b>path</b>: file path, which is mandatory.</p> <p><b>maxFilesPerTrigger</b>: maximum number of files in each trigger. The default value is infinity.</p> <p><b>latestFirst</b>: whether to process limited number of new files. The default value is <b>false</b>.</p> <p><b>fileNameOnly</b>: whether the file name is used as the new file for verification instead of using the complete path. (Default value: <b>false</b>)</p>	Supported	<p>Paths with wildcard are supported, but multiple paths separated by commas (,) are not supported.</p> <p>Files must be placed in a given directory in atomic mode, which can be implemented through file movement in most file systems.</p>
Socket Source	<p><b>host</b>: IP address of the connected node, which is mandatory.</p> <p><b>port</b>: connected port, which is mandatory.</p>	Not supported.	-
Rate Source	<p><b>rowsPerSecond</b>: number of rows generated per second. The default value is <b>1</b>.</p> <p><b>rampUpTime</b>: rising time before the speed specified by <b>rowsPerSecond</b> is reached.</p> <p><b>numPartitions</b>: concurrency degree for generating data rows.</p>	Supported	-
Kafka Source	For details, see <a href="https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html">https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html</a> .	Supported	-

3. Fault tolerance list of Sink

Sinks	Supported Output Mode	Supported Options	Fault Tolerance	Description
File Sink	Append	<b>Path:</b> The specified file format must be specified. For details, see APIs in DataFrameWriter .	exactly-once	Data can be written to partition tables. Time-based partition is better.
Kafka Sink	Append, Update, Complete	For details, see <a href="https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html">https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html</a> .	at-least-once	For details, see <a href="https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html">https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html</a> .
Foreach Sink	Append, Update, Complete	None	Depends on Foreach Writer.	For details, see <a href="https://spark.apache.org/docs/3.1.1/structured-streaming-programming-guide.html#using-foreach">https://spark.apache.org/docs/3.1.1/structured-streaming-programming-guide.html#using-foreach</a> .
ForeachBatch Sink	Append, Update, Complete	None	Depends on operator.	For details, see <a href="https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#using-foreach-and-foreachbatch">https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#using-foreach-and-foreachbatch</a> .
Console Sink	Append, Update, Complete	<b>numRows:</b> number of rows printed in each round. The default value is <b>20</b> . <b>truncate:</b> whether to clear the output when the output is too long. The default value is <b>true</b> .	Not supported	-

Sinks	Supported Output Mode	Supported Options	Fault Tolerance	Description
Memory Sink	Append, Complete	None	Not supported. In complete mode, the entire table is rebuilt after the query is restarted.	-

## 30.5.5 FAQ

### 30.5.5.1 How to Add a User-Defined Library

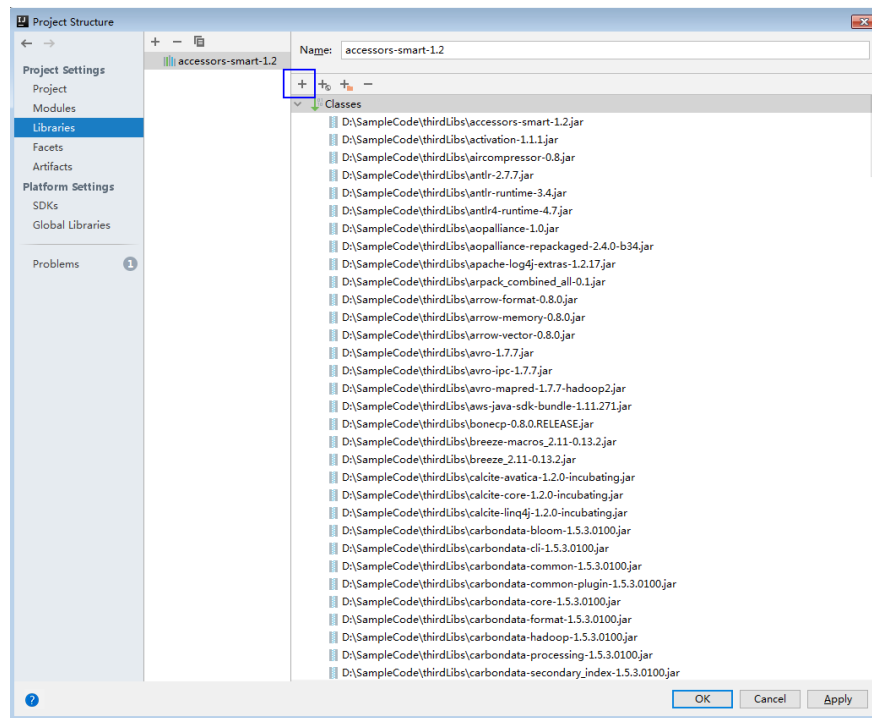
#### Question

In the development of the Spark application, users may add a user-defined dependent library which is different from the sample project. This section describes how to add a dependent library with user-defined codes into the project.

#### Answer

- Step 1** On the main page of the IDEA, choose **File > Project Structures...** to access the **Project Structure** page.
- Step 2** Select the **Libraries** tab, click the icon "+" on the following page, and add the local dependent library.

Figure 30-41 Add the Dependent Library

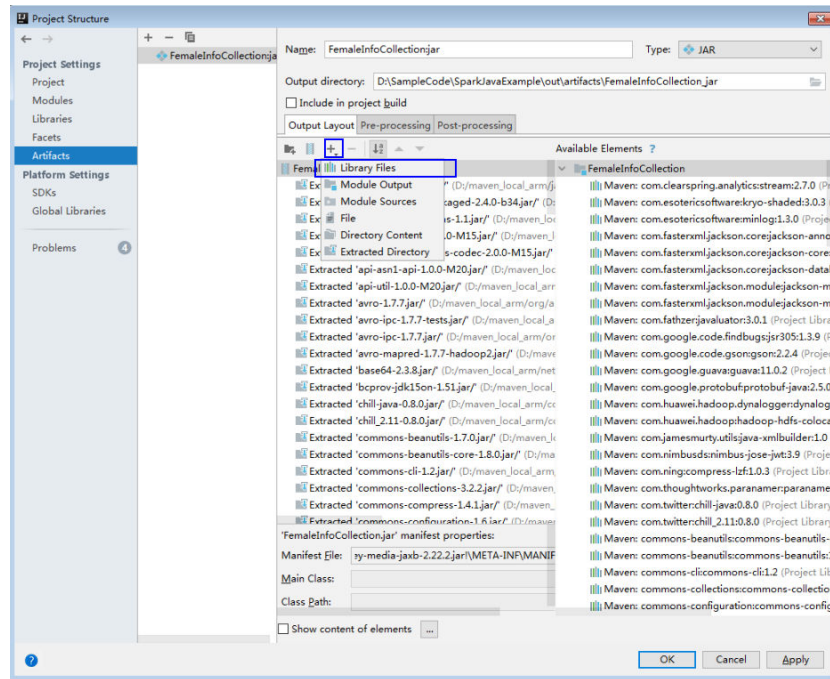


**Step 3** Click **Apply** to load the dependent library and click **OK** to complete the configuration.

**Step 4** Add the dependent library when building Artifacts. This is because the user-defined dependent library does not exist in the operating environment. By adding the library, the created jar contains the user-defined dependent library, which ensures that the Spark application runs properly.

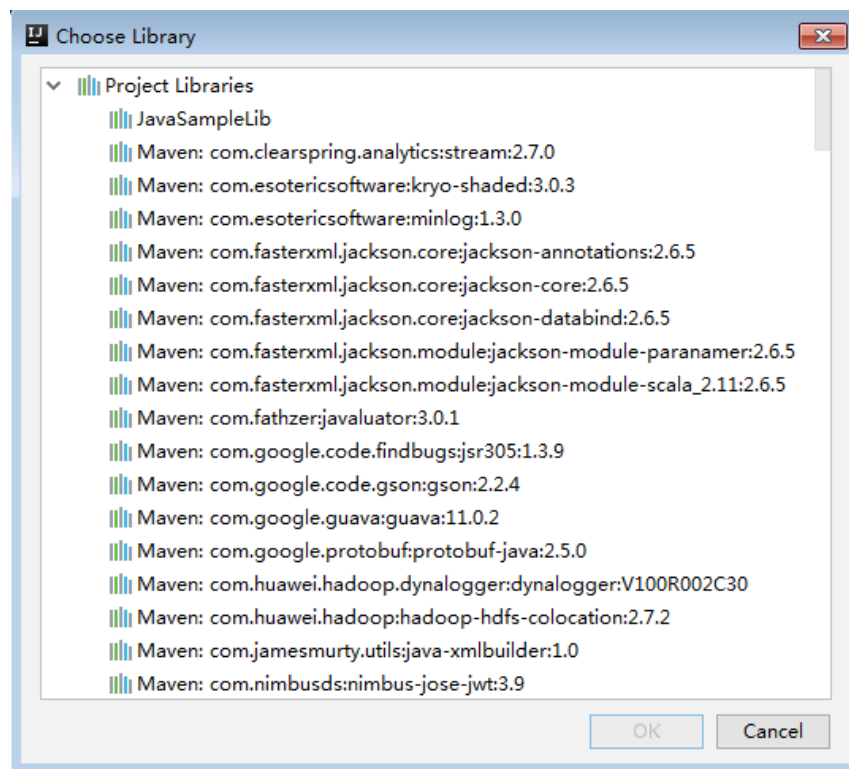
1. On the **Project Structure** page, select the **Artifacts** tab.
2. Click the icon "+" and select **Library Files** in the right window to add the dependent library.

Figure 30-42 Add Library Files



3. Choose the dependent library to be added and click **OK**.

Figure 30-43 Choose Libraries



4. Click **Apply** to load the dependent library and click **OK** to complete the configuration.

----End



### 30.5.5.2 How to Automatically Load Jars Packages?

#### Question

Before importing a project by using the IDEA tool, if Maven has been configured in the IDEA tool, the IDEA tool will automatically load the Jars packages in the Maven configuration. When the automatically loaded Jars packages do not match with the application, the project fails to be built. How to Automatically Load Jars Packages?

#### Answer

After a project is imported, to manually delete the automatically loaded Jars packages, perform the following steps:

1. On the IDEA tool, choose **File > Project Structures...**
2. Select **Libraries** and select the Jars packages that are automatically imported. Right-click the mouse and select **Delete**.

### 30.5.5.3 Why the "Class Does not Exist" Error Is Reported While the SparkStreamingKafka Project Is Running?

#### Question

While the KafkaWordCount task (org.apache.spark.examples.streaming.KafkaWordCount) is being submitted by running the spark-submit script, the log file shows that the Kafka-related class does not exist. The KafkaWordCount sample is provided by the Spark open-source community.

#### Answer

When Spark is deployed, the following JAR files are saved in the `SPARK_HOME/jars/streamingClient010` directory on the client and the `BIGDATA_HOME/FusionInsight_Spark2x_8.1.0.1/install/FusionInsight-Spark2x-3.1.1/spark/jars/streamingClient010` directory on the server.

- kafka-clients-xxx.jar
- kafka\_2.12-xxx.jar
- spark-streaming-kafka-0-10\_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar
- spark-token-provider-kafka-0-10\_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar

Because `SPARK_HOME/jars/streamingClient010/*` is not added in to classpath by default, you need to configure manually.

When the application is submitted and run, add following parameters in the command. See [Compiling and Running the Application](#).

```
--jars $SPARK_CLIENT_HOME/jars/streamingClient010/kafka-client-2.4.0.jar,
$SPARK_CLIENT_HOME/jars/streamingClient010/kafka_2.12-*.jar,
$SPARK_CLIENT_HOME/jars/streamingClient010/spark-streaming-
kafka-0-10_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar
```

You can run the preceding command to submit the self-developed applications and sample projects.

To submit the sample projects such as KafkaWordCount provided by Spark open source community, you need to add other parameters in addition to `--jars`. Otherwise, the `ClassNotFoundException` error will occur. The configurations in `yarn-client` and `yarn-cluster` modes are as follows:

- `yarn-client` mode  
In the configuration file `spark-defaults.conf` on the client, add the path of the client dependency package, for example ``${SPARK_HOME}/jars/streamingClient010/*`, (in addition to `--jars`) to the `spark.driver.extraClassPath` parameter.
- `yarn-cluster` mode  
Perform any one of the following configurations in addition to `--jars`:
  - In the configuration file `spark-defaults.conf` on the client, add the path of the server dependency package, for example ``${BIGDATA_HOME}/FusionInsight_Spark2x_8.1.0.1/install/FusionInsight-Spark2x-3.1.1/spark/jars/streamingClient010/*`, to the `spark.yarn.cluster.driver.extraClassPath` parameter.
  - Delete the `original-spark-examples_2.12-3.1.1-xxx.jar` packages from all the server nodes.
  - In the `spark-defaults.conf` configuration file on the client, modify (or add and modify) the parameter `spark.driver.userClassPathFirst` to `true`.

#### 30.5.5.4 Privilege Control Mechanism of SparkSQL UDF Feature

##### Question

What are the privilege control mechanism for UDF in SparkSQL?

##### Answer

When the SQL statement unable to meet user scenarios, the user can use the UDF feature to do the operations on HDFS or HBase data.

To ensure data security, SparkSQL UDF can only be used by admin users, meanwhile the users must make sure those self-defined functions do not contain malicious codes.

#### 30.5.5.5 Why Does Kafka Fail to Receive the Data Written Back by SLog in to the node where the client is installed as the client installation user.park Streaming?

##### Question

While a running Spark Streaming task is writing data back to Kafka, Kafka cannot receive the written data and Kafka logs contain the following error information:

```
2016-03-02 17:46:19,017 | INFO | [kafka-network-thread-21005-1] | Closing socket connection to /
10.91.8.208 due to invalid request: Request of length
122371301 is not valid, it is larger than the maximum size of 104857600 bytes. | kafka.network.Processor
```

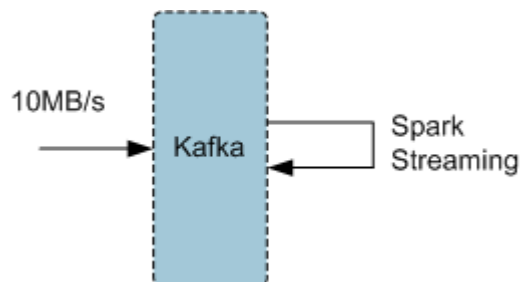
```
(Logging.scala:68)
2016-03-02 17:46:19,155 | INFO | [kafka-network-thread-21005-2] | Closing socket connection to /
10.91.8.208. | kafka.network.Processor (Logging.scala:68)
2016-03-02 17:46:19,270 | INFO | [kafka-network-thread-21005-0] | Closing socket connection to /
10.91.8.208 due to invalid request:
Request of length 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. |
kafka.network.Processor (Logging.scala:68)
2016-03-02 17:46:19,513 | INFO | [kafka-network-thread-21005-1] | Closing socket connection to /
10.91.8.208 due to invalid request:
Request of length 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. |
kafka.network.Processor (Logging.scala:68)
2016-03-02 17:46:19,763 | INFO | [kafka-network-thread-21005-2] | Closing socket connection to /
10.91.8.208 due to invalid request:
Request of length 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. |
kafka.network.Processor (Logging.scala:68)
53393 [main] INFO org.apache.hadoop.mapreduce.Job - Counters: 50
```

## Answer

As shown in the figure below, the logic defined in Spark Streaming applications is as follows: reading data from Kafka -> executing processing -> writing result data back to Kafka.

Imagine that data is written into Kafka at a data rate of 10 MB/s, the interval (defined in Spark Streaming) between write-back operations is 60s, and a total of 600-MB data needs to be written back into Kafka. If Kafka defines that a maximum of 500-MB data can be received at a time, then the size of written-back data exceeds the threshold, triggering the error information.

Figure 30-44 Application scenario



Solution:

- Method 1: On Spark Streaming, reduce the interval between write-back operations to avoid the size of written-back data exceeding the threshold defined by Kafka. The recommended interval is 5-10 seconds.
- Method 2: Increase the threshold defined in Kafka. It is advisable to increase the threshold by adjusting the **socket.request.max.bytes** parameter of Kafka service on FusionInsight Manager.

### 30.5.5.6 Why a Spark Core Application Is Suspended Instead of Being Exited When Driver Memory Is Insufficient to Store Collected Intensive Data?

#### Question

A Spark Core application is attempting to collect intensive data and store it into the Driver. When the Driver runs out of memory, the Spark Core application is suspended. Why does the Spark Core application not exit?

The following is the log information displayed at the time of out-of-memory (OOM) error.

```
16/04/19 15:56:22 ERROR Utils: Uncaught exception in thread task-result-getter-2
java.lang.OutOfMemoryError: Java heap space
at java.lang.reflect.Array.newInstance(Native Method)
at java.lang.reflect.Array.newInstance(Array.java:75)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1671)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1707)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
at org.apache.spark.serializer.JavaDeserializationStream.readObject(JavaSerializer.scala:71)
at org.apache.spark.serializer.JavaSerializerInstance.deserialize(JavaSerializer.scala:91)
at org.apache.spark.scheduler.DirectTaskResult.value(TaskResult.scala:94)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3$$$anonfunrun1.applymcVsp(TaskResultGetter.scala:66)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3$$$anonfunrun1.apply(TaskResultGetter.scala:57)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3$$$anonfunrun1.apply(TaskResultGetter.scala:57)
at org.apache.spark.util.Utils$.logUncaughtExceptions(Utils.scala:1716)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3.run(TaskResultGetter.scala:56)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)
Exception in thread "task-result-getter-2" java.lang.OutOfMemoryError: Java heap space
at java.lang.reflect.Array.newInstance(Native Method)
at java.lang.reflect.Array.newInstance(Array.java:75)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1671)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1707)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
at org.apache.spark.serializer.JavaDeserializationStream.readObject(JavaSerializer.scala:71)
at org.apache.spark.serializer.JavaSerializerInstance.deserialize(JavaSerializer.scala:91)
at org.apache.spark.scheduler.DirectTaskResult.value(TaskResult.scala:94)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3$$$anonfunrun1.applymcVsp(TaskResultGetter.scala:66)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3$$$anonfunrun1.apply(TaskResultGetter.scala:57)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3$$$anonfunrun1.apply(TaskResultGetter.scala:57)
at org.apache.spark.util.Utils$.logUncaughtExceptions(Utils.scala:1716)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3.run(TaskResultGetter.scala:56)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)
```

## Answer

If memory of the Driver is insufficient to store the intensive data that has been collected, the OOM error is reported and the Driver performs garbage collection repeatedly to reclaim the memory occupied by garbage. The Spark Core application remains suspended while garbage collection is under way.

If you expect the Spark Core application to exit forcibly in the event of OOM error, add the following information to the configuration option **spark.driver.extraJavaOptions** in the Spark client configuration file **\$SPARK\_HOME/conf/spark-defaults.conf** when you start the Spark Core application for the first time:

```
-XX:OnOutOfMemoryError='kill -9 %p'
```

### 30.5.5.7 Why the Name of the Spark Application Submitted in Yarn-Cluster Mode Does not Take Effect?

#### Question

The name of the Spark application submitted in yarn-cluster mode does not take effect, whereas the Spark application name submitted in yarn-client mode takes effect. In the following figure, the first application is submitted in yarn-client mode and the application name Spark Pi takes effect. However, the setAppName execution sequence of a task submitted in yarn-client mode is different from that submitted in yarn-cluster mode.

application_1463550673865_0007	zwm	Spark Pi	SPARK	tenant_zwm	Sat May 28 11:58:27 +0800 2016	Sat May 28 11:58:51 +0800 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A	History	N/A
application_1463550673865_0006	zwm	org.apache.spark.examples.SparkPi	SPARK	tenant_zwm	Sat May 28 11:58:29 +0800 2016	Sat May 28 11:58:00 +0800 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A	History	N/A

#### Answer

The reason is that the setAppName execution sequence of a task submitted in yarn-client mode is different from that submitted in yarn-cluster mode. In yarn-client mode, the setAppName is read before the application is registered in yarn. However, in yarn-cluster mode, the setAppName is read after the application registers with yarn, so the name of the second application does not take effect.

Solution:

When submitting tasks using the spark-submit script, set **--name** the same as the application name in sparkconf.setAppName(appname).

For example, if the application name is Spark Pi, in sparkconf.setAppName(appname) in yarn-cluster mode, run the following command:

```
./spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode cluster --name SparkPi jars/original-spark-examples*.jar 10
```

### 30.5.5.8 How to Perform Remote Debugging Using IDEA?

#### Question

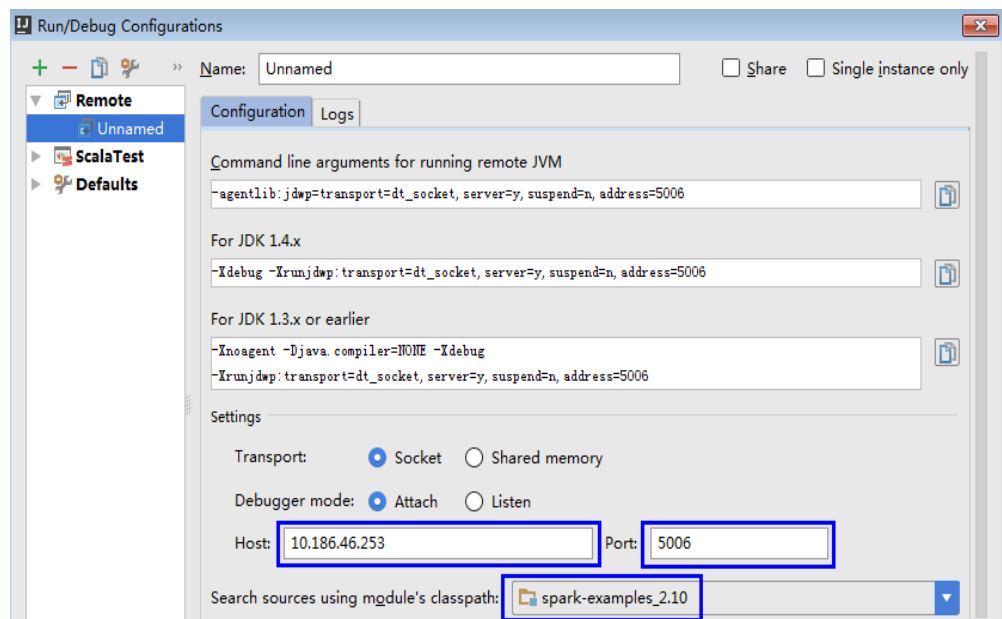
How to perform remote debugging using IDEA during Spark secondary development?

## Answer

The SparkPi application is used as an example here to illustrate how to perform remote debugging using IDEA.

1. Open the project and choose **Run > Edit Configurations**.
2. In the displayed window, click **+** at the upper left corner, and then choose **Remote** on the drop-down menu, as shown in **Figure 30-45**.

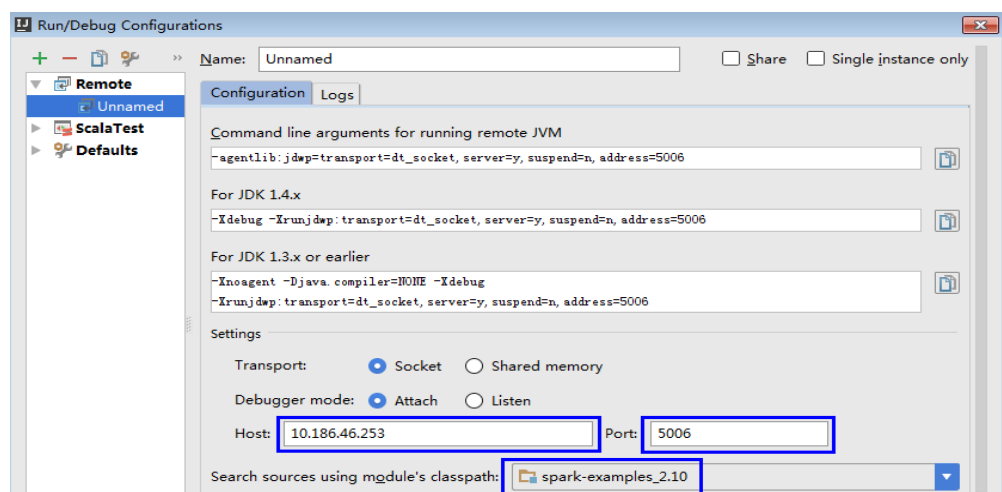
**Figure 30-45** Choosing Remote



3. Configure the **Host**, **Port**, and **Search source using module's classpath**, as shown in **Figure 30-46**.

**Host** indicates the IP address of the Spark client and **Port** indicates the debugging port. Ensure that the port is available on the VM.

**Figure 30-46** Configuring parameters



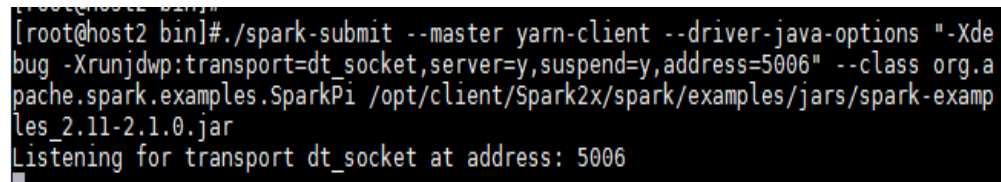
 NOTE

If the value of **Port** is changed, the debugging command of **For JDK1.4.x** must be changed accordingly. For example, if the value of **Port** is changed to **5006**, the debugging command must be changed to **-Xdebug -Xrunjdw:transport=dt\_socket,server=y,suspend=y,address=5006**, which will be used during the startup of Spark.

4. Run the following command to remotely start SparkPi on the Spark client:  
`./spark-submit --master yarn-client --driver-java-options "-Xdebug -Xrunjdw:transport=dt_socket,server=y,suspend=y,address=5006" --class org.apache.spark.examples.SparkPi /opt/Fl-Client/Spark2x/spark/examples/jars/spark-examples_2.12-3.1.1-xxx.jar`

Change the **--class** and JAR package in the preceding command to the **--class** and JAR package of the actual application. Change the **-Xdebug -Xrunjdw:transport=dt\_socket,server=y,suspend=y,address=5006** to the **For JDK1.4.x** debugging command obtained in [3](#).

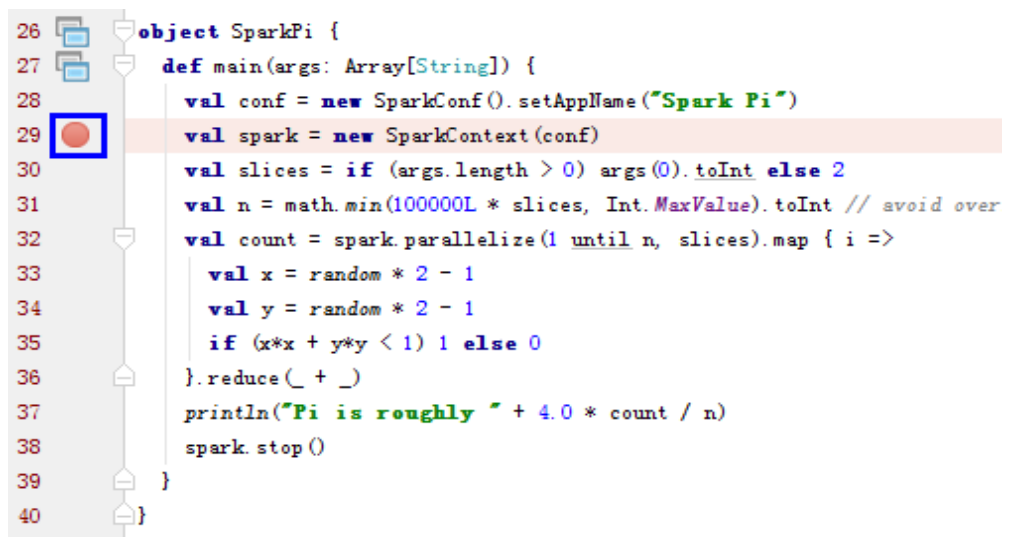
Figure 30-47 Command for running Spark



```
[root@host2 bin]# ./spark-submit --master yarn-client --driver-java-options "-Xdebug -Xrunjdw:transport=dt_socket,server=y,suspend=y,address=5006" --class org.apache.spark.examples.SparkPi /opt/Fl-Client/Spark2x/spark/examples/jars/spark-examples_2.11-2.1.0.jar
Listening for transport dt_socket at address: 5006
```

5. Set the debugging breakpoint.  
Click the blank area on the left of the code editing window to select the breakpoint of code. [Figure 30-48](#) illustrates how to select the breakpoint of the code in row 29 of **SparkPi.scala**.

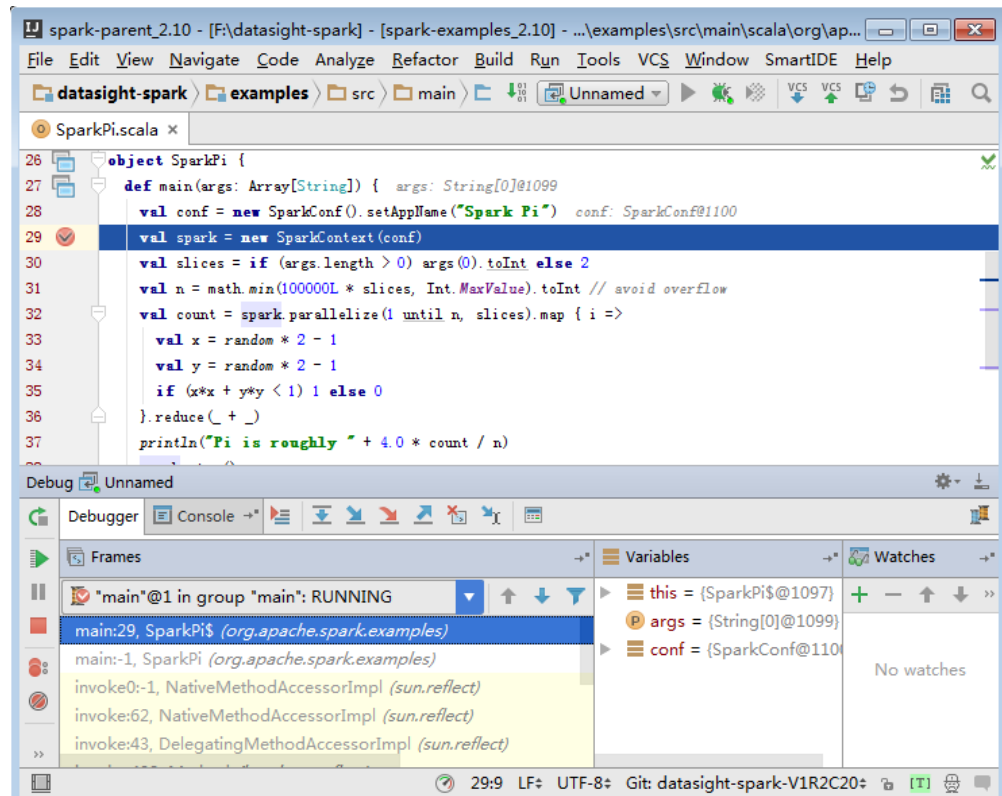
Figure 30-48 Setting the breakpoint



```
26 object SparkPi {
27 def main(args: Array[String]) {
28 val conf = new SparkConf().setAppName("Spark Pi")
29 val spark = new SparkContext(conf)
30 val slices = if (args.length > 0) args(0).toInt else 2
31 val n = math.min(100000L * slices, Int.MaxValue).toInt // avoid over
32 val count = spark.parallelize(1 until n, slices).map { i =>
33 val x = random * 2 - 1
34 val y = random * 2 - 1
35 if (x*x + y*y < 1) 1 else 0
36 }.reduce(_ + _)
37 println("Pi is roughly " + 4.0 * count / n)
38 spark.stop()
39 }
40 }
```

6. Start the debugging.  
On the menu bar of IDEA, choose **Run > Debug 'Unnamed'** to open a debugging window. Start the debugging of **SparkPi**, for example, performing step-by-step debugging, checking call stack information, and tracking variable values, as shown in [Figure 30-49](#).

Figure 30-49 Debugging



### 30.5.5.9 How to Submit the Spark Application Using Java Commands?

#### Question

How to submit the Spark application using Java commands in addition to spark-submit commands?

#### Answer

Use the org.apache.spark.launcher.SparkLauncher class and run Java command to submit the Spark application. The procedure is as follows:

**Step 1** Define the org.apache.spark.launcher.SparkLauncher class. The SparkLauncherJavaExample and SparkLauncherScalaExample are provided by default as sample codes. You can modify the input parameters of sample codes as required.

- If you use Java as the development language, you can compile the SparkLauncher class by referring to the following code:

```

public static void main(String[] args) throws Exception {
 System.out.println("com.huawei.bigdata.spark.examples.SparkLauncherExample <mode>
<jarParh> <app_main_class> <appArgs>");
 SparkLauncher launcher = new SparkLauncher();
 launcher.setMaster(args[0])
 .setAppResource(args[1]) // Specify user app jar path
 .setMainClass(args[2]);
 if (args.length > 3) {
 String[] list = new String[args.length - 3];
 for (int i = 3; i < args.length; i++) {
 list[i-3] = args[i];
 }
 }
}

```



```

 }
 // Set app args
 launcher.addAppArgs(list);
 }

 // Launch the app
 Process process = launcher.launch();
 // Get Spark driver log
 new Thread(new ISRRunnable(process.getErrorStream())).start();
 int exitCode = process.waitFor();
 System.out.println("Finished! Exit code is " + exitCode);
}

```

- If you use Scala as the development language, you can compile the SparkLauncher class by referring to the following code:

```

def main(args: Array[String]) {
 println(s"com.huawei.bigdata.spark.examples.SparkLauncherExample <mode> <jarParh>
<app_main_class> <appArgs>")
 val launcher = new SparkLauncher()
 launcher.setMaster(args(0))
 .setAppResource(args(1)) // Specify user app jar path
 .setMainClass(args(2))
 if (args.drop(3).length > 0) {
 // Set app args
 launcher.addAppArgs(args.drop(3):_*)
 }

 // Launch the app
 val process = launcher.launch()
 // Get Spark driver log
 new Thread(new ISRRunnable(process.getErrorStream)).start()
 val exitCode = process.waitFor()
 println(s"Finished! Exit code is $exitCode")
}

```

**Step 2** Develop the Spark application based on the service logic and configure constant values such as the main class of the user-compiled Spark application. For details about different scenarios, see [Developing the Project](#).

- If you use the security mode, you are advised to prepare the security authentication code, service application code, and related configurations according to the security requirements.

 **NOTE**

In yarn-cluster mode, security authentication cannot be added to the Spark project. Therefore, users need to add security authentication code or run commands to perform security authentication. There is security authentication code in the sample code. In yarn-cluster mode, modify the corresponding security code before running the operation.

- In normal mode, prepare the service application code and related configurations.

**Step 3** Call the `org.apache.spark.launcher.SparkLauncher.launch()` function to submit user applications.

1. Generate jar packages from the SparkLauncher application and user applications, and upload the jar packages to the Spark node of the application. For details about how to generate jar packages, see [Compiling and Running the Application](#).
  - The compilation dependency package of SparkLauncher is **spark-launcher\_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar**. Please obtain it from the **jars** directory of **FusionInsight\_Spark2x\_8.1.0.1.tar.gz** in **Software**.

- The compilation dependency packages of user applications vary with the code. You need to load the dependency package based on the compiled code.
- 2. Upload the dependency jar package of the application to a directory, for example, `$SPARK_HOME/jars` (the node where the application will run).

Upload the dependency packages of the SparkLauncher class and the application to the `jars` directory on the client. The dependency package of the sample code has existed in the `jars` directory on the client.

 **NOTE**

If you want to use the Spark Launcher class, the node where the application runs must have the Spark client installed. The running of the Spark Launcher class is dependent on the configured environment variables, running dependency package, and configuration files.

- 3. In the node where the Spark application is running, run the following command to submit the application. Then you can check the running situation through Spark web UI and check the result by obtaining specified files. See [Checking the Commissioning Result](#) for details.

```
java -cp $SPARK_HOME/conf:$SPARK_HOME/jars/
*:SparkLauncherExample.jar
com.huawei.bigdata.spark.examples.SparkLauncherExample yarn-
client /opt/female/FemaleInfoCollection.jar
com.huawei.bigdata.spark.examples.FemaleInfoCollection <inputPath>
```

----End

### 30.5.5.10 A Message Stating "Problem performing GSS wrap" Is Displayed When IBM JDK Is Used

#### Question

A Message Stating "Problem performing GSS wrap" Is Displayed When IBM JDK Is Used.

#### Answer

##### Possible Causes

The authentication fails because the duration for creating a JDBC connection on IBM JDK exceeds the timeout duration for user authentication (one day by default).

 **NOTE**

The authentication mechanism of IBM JDK differs from that of Oracle JDK. IBM JDK checks time but does not detect external time update. Therefore, time is not updated even though `relogin` is called.

##### Solution

When one JDBC connection fails, disable this connection, and create a new connection to continue performing previous operations.

### 30.5.5.11 Application Fails When ApplicationManager Is Terminated During Data Processing in the Cluster Mode of Structured Streaming

#### Question

If ApplicationManager is terminated during data processing in the cluster mode of Structured Streaming, the following information is displayed when the application is executed, indicating an error:

```
2017-05-09 20:46:02,393 | INFO | main |
client token: Token { kind: YARN_CLIENT_TOKEN, service: }
diagnostics: User class threw exception: org.apache.spark.sql.AnalysisException: This query does not
support recovering from checkpoint location. Delete hdfs://hacluster/structuredtest/checkpoint/offsets to
start over.;
ApplicationMaster host: 10.96.101.170
ApplicationMaster RPC port: 0
queue: default
start time: 1494333891969
final status: FAILED
tracking URL: https://9-96-101-191:8090/proxy/application_1493689105146_0052/
user: spark2x | org.apache.spark.internal.Logging$class.logInfo(Logging.scala:54)
Exception in thread "main" org.apache.spark.SparkException: Application application_1493689105146_0052
finished with failed status
```

#### Answer

**Possible causes:** The error occurs because **recoverFromCheckpointLocation** is determined as **false** but the checkpoint directory is configured.

The value of the **recoverFromCheckpointLocation** parameter is the result of the **outputMode == OutputMode.Complete()** statement in the code. (The default **outputMode** is **append**.)

**Solution:** When compiling an application, you can change the data output mode based on the actual conditions.

When the output mode is changed to **complete**, the value of **recoverFromCheckpointLocation** is determined as **true**. No error would be indicated if the checkpoint directory is configured at the time.

### 30.5.5.12 Restrictions on Restoring the Spark Application from the checkpoint

#### Question

The Spark application can be restored from the checkpoint and continues to execute the task from the breakpoint of the last task, ensuring that data is not lost. However, in some cases, the Spark application fails to be restored from the checkpoint.

#### Answer

The checkpoint contains the object serialization information, task execution status information, and configuration information of the Spark application. Therefore, the Spark application cannot be restored from the checkpoint if the following problems exist:

1. The service code is changed and the SerialVersionUID is not specified in the changed class.
2. The internal Spark class is changed and the SerialVersionUID is not specified in the changed class.

Besides, some configuration items are stored in the checkpoint. Therefore, if some configuration items of the service are modified, the configuration items may remain unchanged when the service is restored from the checkpoint. Currently, only the following configurations are reloaded when the service is restored from the checkpoint.

```
"spark.yarn.app.id",
"spark.yarn.app.attemptId",
"spark.driver.host",
"spark.driver.bindAddress",
"spark.driver.port",
"spark.master",
"spark.yarn.jars",
"spark.yarn.keytab",
"spark.yarn.principal",
"spark.yarn.credentials.file",
"spark.yarn.credentials.renewalTime",
"spark.yarn.credentials.updateTime",
"spark.ui.filters",
"spark.mesos.driver.frameworkId",
"spark.yarn.jars"
```

## Solution

Manually delete the checkpoint directory and restart the service program.

### NOTE

Deleting a file is a high-risk operation. Ensure that the files are no longer needed before performing this operation.

## 30.5.5.13 Support for Third-party JAR Packages on x86 and TaiShan Platforms

### Question

How to enable Spark2x to support the third-party JAR packages (for example, custom UDF packages) if these packages have two versions (x86 and TaiShan)?

### Answer

Use the hybrid solution.

- Step 1** Go to the installation directory of the Spark2x SparkResource on the server. The cluster may be installed on multiple nodes. You can go to any SparkResource node to go to the installation directory of the SparkResource.
- Step 2** Prepare the .jar package, for example, xx.jar packages for the x86 and TaiShan platforms. Copy the xx.jar packages of x86 and TaiShan to the x86 and TaiShan folders respectively.
- Step 3** Run the following commands in the current directory to compress the JAR packages:

```
zip -qDj spark-archive-2x-x86.zip x86/*
```

```
zip -qDj spark-archive-2x-arm.zip arm/*
```

```

rw-r--r-- 2 omm wheel 63 Dec 14 17:57 arm
rw-r--r-- 2 omm wheel 4096 Dec 14 17:57 bin
rwxr-x--- 2 omm ficommon 4096 Dec 14 17:57 carbonlib
rw-r--r-- 1 omm wheel 1403 Dec 15 12:51 child.crt
rw-r--r-- 1 omm wheel 1097 Dec 15 12:51 child.csr
rw-r--r-- 1 omm wheel 6296 Dec 15 12:51 child.keystore
rw-r--r-- 2 omm wheel 326 Dec 14 17:57 conf
rw-r--r-- 3 omm wheel 18 Dec 14 17:54 examples
rwxr-x--- 4 omm ficommon 12288 Dec 14 17:57 jars
rw-r--r-- 1 omm wheel 18045 Dec 14 17:54 LICENSE
rw-r--r-- 1 omm wheel 26366 Dec 14 17:54 NOTICE
rw-r--r-- 5 omm wheel 129 Dec 14 17:57 python
rw-r--r-- 3 omm wheel 17 Dec 14 17:54 R
rw-r--r-- 1 omm wheel 501 Dec 14 17:54 README
rw-r--r-- 1 omm wheel 20 Dec 14 17:54 RELEASE
rw-r--r-- 2 omm wheel 4096 Dec 15 17:14 sbin
rw-r--r-- 1 root root 14904892 Dec 15 17:14 spark-archive-2x-arm.zip
rw-r--r-- 1 root root 13881100 Dec 15 17:15 spark-archive-2x-x86.zip
rw-r--r-- 1 omm wheel 244 Dec 14 17:57 version.properties
rw-r--r-- 2 omm wheel 63 Dec 14 17:57 x86
rw-r--r-- 2 omm wheel 42 Dec 14 17:54 yarn

```

**Step 4** Run the following command to check the .jar package on which the Spark2x of HDFS depends:

```
hdfs dfs -ls /user/spark2x/jars/8.1.0.1
```

 **NOTE**

Change the version number 8.1.0.1 as required.

Run the following commands to move the .jar package files from HDFS, for example, /tmp.

```
hdfs dfs -mv /user/spark2x/jars/8.1.0.1/spark-archive-2x-arm.zip /tmp
```

```
hdfs dfs -mv /user/spark2x/jars/8.1.0.1/spark-archive-2x-x86.zip /tmp
```

**Step 5** Run the following commands to upload the **spark-archive-2x-arm.zip** and **spark-archive-2x-x86.zip** packages in **Step 3** to the **/user/spark2x/jars/8.1.0.1** directory of HDFS:

```
hdfs dfs -put spark-archive-2x-arm.zip /user/spark2x/jars/8.1.0.1/
```

```
hdfs dfs -put spark-archive-2x-x86.zip /user/spark2x/jars/8.1.0.1/
```

After the upload is complete, delete local files **spark-archive-2x-arm.zip** and **spark-archive-2x-x86.zip**.

**Step 6** Perform **Step 1** to **Step 2** for other SparkResource nodes.

**Step 7** Log in to the web UI and restart the jdbcServer instance of Spark2x.

**Step 8** After the restart, update the client configuration. Copy **xx.jar** for the corresponding platform to the Spark2x installation directory **\${install\_home}/Spark2x/spark/jars** on the client according to client server type (x86 or TaiShan). **\$ {install\_home}** indicates the installation path of the client. Replace it with the actual one. If the local installation directory is **/opt/hadoopclient**, copy the corresponding **xx.jar** to the **/opt/hadoopclient/Spark2x/spark/jars** folder.

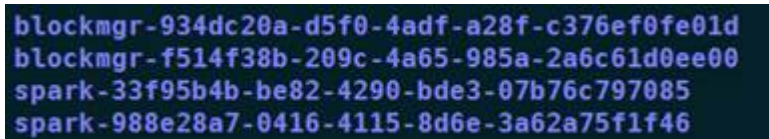
----End

### 30.5.5.14 What Should I Do If a Large Number of Directories Whose Names Start with **blockmgr-** or **spark-** Exist in the **/tmp** Directory on the Client Installation Node?

#### Question

After the system runs for a long time, there are many directories whose names start with **blockmgr-** or **spark-** in the **/tmp** directory on the node where the client is installed.

**Figure 30-50** Residual directory example



```
blockmgr-934dc20a-d5f0-4adf-a28f-c376ef0fe01d
blockmgr-f514f38b-209c-4a65-985a-2a6c61d0ee00
spark-33f95b4b-be82-4290-bde3-07b76c797085
spark-988e28a7-0416-4115-8d6e-3a62a75f1f46
```

#### Answer

During the running of Spark tasks, the driver creates a local temporary directory whose name starts with **spark-** for storing service JAR packages and configuration files. In addition, the driver creates a local temporary directory with the name starting with **blockmgr-** for storing block data. The two directories are automatically deleted when the Spark application running is finished.

The path for storing the two directories is preferentially specified by the environment variable *SPARK\_LOCAL\_DIRS*. If the environment variable is not configured, use the value of **spark.local.dir** as the path for storing the directories. If the environment variable and the preceding parameter both are not configured, use the value of **java.io.tmpdir**. By default, **spark.local.dir** is set to **/tmp** on the client. Therefore, the **/tmp** directory is used by default.

In some special cases, for example, the driver process does not exit normally, for example, the **kill -9** command ends the process, or the Java virtual machine crashes. As a result, the directory cannot be deleted and remains in the system.

Currently, only the driver processes in yarn-client mode and local mode may confront the preceding problem. In yarn-cluster mode, the temporary directory of the process in the container is configured as the temporary directory of the container. When the container exits, the container automatically clears the directory. Therefore, this problem does not occur in yarn-cluster mode.

#### Solution

In Linux, you can configure automatic directory clearing for the **/tmp** temporary directory. Alternatively, you can change the value of **spark.local.dir** in the **spark-defaults.conf** configuration file on the client, specify the temporary directory to a specified directory, and configure a clear mechanism for the directory.

### 30.5.5.15 Error Code 139 Reported When Python Pipeline Runs in the ARM Environment

#### Question

Error code 139 is displayed when the pipeline of the Python plug-in is used on the TaiShan server. The error information is as follows:

```
subprocess exited with status 139
```

#### Answer

The Python program uses both **libcrypto.so** and **libssl.so**. If the native library directory of Hadoop is added to **LD\_LIBRARY\_PATH**, the **libcrypto.so** in the hadoop native library is used and the **libssl.so** provided by the system is used (because the hadoop native directory does not contain this package). The versions of the two libraries do not match. As a result, a segment error occurs during the running of the Python file.

#### Solution

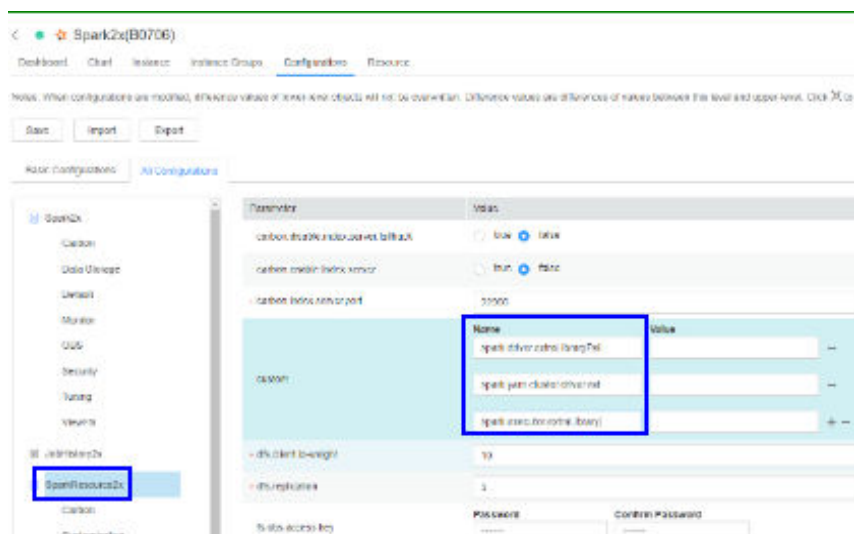
Solution 1:

Modify the **spark-default.conf** file in the **conf** directory of the Spark2x client. Clear the values of **spark.driver.extraLibraryPath**, **spark.yarn.cluster.driver.extraLibraryPath**, and **spark.executor.extraLibraryPath**.

Solution 2:

On the Spark2x page of FusionInsight Manager, modify the preceding three parameters. Restart the Spark2x instance, and download the client again. The procedure is as follows:

1. Log in to FusionInsight Manager, choose **Cluster > Name of the desired cluster > Services > Spark2x > Configurations > All Configurations**, search for the **spark.driver.extraLibraryPath** and **spark.executor.extraLibraryPath** parameters, and clear their values.
2. Choose **All Configurations > SparkResource2x**. In the **custom** area, add the three parameters in solution 1, as shown in the following figure.



3. Click **Save**. Restart the expired spark2x instance. Download and install the client again.

### 30.5.5.16 What Should I Do If the Structured Streaming Task Submission Way Is Changed?

#### Question

When submitting a structured streaming task, users need to run the `--jars` command to specify the Kafka JAR package path, for example, `--jars /kafkadir/kafka-clients-x.x.x.jar,/kafkadir/kafka_2.11-x.x.x.jar`. However, in the current version, users need to configure additional items. Otherwise, an error is reported, indicating that the class is not found.

#### Answer

The Spark kernel of the current version depends on the Kafka JAR package, which is used by the structured streaming. Therefore, when submitting a structured streaming task, you need to add the Kafka JAR package path to the library directory of the driver of this task to ensure that the driver can properly load the Kafka package.

#### Solution

1. The following operations need to be performed additionally when a structured streaming task in Yarn-client mode is submitted:  
Copy the path of `spark.driver.extraClassPath` in the `spark-default.conf` file in the Spark client directory, and add the Kafka JAR package path to its end. When submitting a structured stream task, add the `--conf` statement to combine these two configuration items. For example, if the Kafka JAR package path is `/kafkadir`, you need to add `--conf spark.driver.extraClassPath=/opt/client/Spark2x/spark/conf:/opt/client/Spark2x/spark/jars/*:/opt/client/Spark2x/spark/x86/*:/kafkadir/*` when submitting the task.
2. The following operations need to be performed additionally when a structured streaming task in **Yarn-cluster** mode is submitted:  
Copy the path of `spark.yarn.cluster.driver.extraClassPath` in the `spark-default.conf` file in the Spark client directory, and add relative paths of Kafka JAR packages to its end. When submitting a structured stream task, add the `--conf` statement to combine these two configuration items. For example, if the Kafka JAR package paths are `kafka-clients-x.x.x.jar` and `kafka_2.11-x.x.x.jar`, you need to add `--conf spark.yarn.cluster.driver.extraClassPath=/home/huawei/Bigdata/common/runtime/security:/kafka-clients-x.x.x.jar:/kafka_2.11-x.x.x.jar` when submitting the task.
3. In the current version, the structured streaming of Spark does not support versions earlier than Kafka2.x. In the upgrade scenario, use the client of earlier versions.



### 30.5.5.17 Common JAR File Conflicts

#### Symptom

Spark can interconnect with many third-party tools. Therefore, it depends on a large number of third-party packages. Some packages are provided by MRS. As a result, the versions of the JAR files used by the code may be different from those of the JAR files provided by the cluster. In this case, JAR file conflicts may occur.

Common JAR file conflict errors are as follows:

1. A class cannot be found: **java.lang.NoClassDefFoundError**
2. A method cannot be found: **java.lang.NoSuchMethodError**

#### Cause Analysis

The following uses UDF customization as an example:

```
2021-02-08 10:51:04.249 INFO [main] Total input files to process : 2 | org.apache.hadoop.mapred.FileInputFormat.ListStatus(FileInputFormat.java:256)
Exception in thread "main" java.lang.NoClassDefFoundError: com.huawei.udf.HelloUDF
 at com.huawei.sparkwordcount$.main(SparkWordCount.scala:32)
 at com.huawei.sparkwordcount$.main(SparkWordCount.scala)
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
 at java.lang.reflect.Method.invoke(Method.java:498)
 at org.apache.spark.deploy.SparkSubmit$.org$apache$spark$deploy$SparkSubmit$$runMain(SparkSubmit.scala:813)
 at org.apache.spark.deploy.SparkSubmit$.doRunMain$1(SparkSubmit.scala:187)
 at org.apache.spark.deploy.SparkSubmit$.submit(SparkSubmit.scala:212)
 at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:126)
 at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
Caused by: java.lang.ClassNotFoundException: com.huawei.udf.HelloUDF
 at java.net.URLClassLoader.findClass(URLClassLoader.java:382)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
 ... 11 more
```

If the class cannot be found, do as follows:

1. Check whether the JAR file of the class is in the classpath of the JVM. The JAR files of the Spark are stored in the **Spark client directory/jars/** directory.
2. Check whether multiple JAR files contain the class.
3. Check whether other dependencies are added to the task by using **--jars**.
4. If the dependencies are added to the task but cannot be found, the driver of the configuration file or the classpath of the executor may be incorrectly configured. View logs to check whether the configuration file is loaded to the environment.
5. Check whether the error is caused by a class initialization failure that occurs before using the class.

```
Exception in thread "main" java.lang.NoSuchMethodError: com.huawei.udf.HelloUDF.evaluate(Ljava/lang/String;Ljava/lang/String;
 at com.huawei.sparkwordcount$.main(SparkWordCount.scala:32)
 at com.huawei.sparkwordcount$.main(SparkWordCount.scala)
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
 at java.lang.reflect.Method.invoke(Method.java:498)
 at org.apache.spark.deploy.SparkSubmit$.org$apache$spark$deploy$SparkSubmit$$runMain(SparkSubmit.scala:813)
 at org.apache.spark.deploy.SparkSubmit$.doRunMain$1(SparkSubmit.scala:187)
 at org.apache.spark.deploy.SparkSubmit$.submit(SparkSubmit.scala:212)
 at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:126)
 at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
```

If the method cannot be found, do as follows:

1. Check whether the JAR file of the class corresponding to the method is loaded to the classpath of the JVM. The classes of the Spark are stored in the **Spark client directory/jars/** directory.
2. Check whether multiple JAR files contain the class. (Pay special attention to different versions of the same tool.)
3. If the error is reported for a Hadoop package, the possible cause is that the Hadoop version is inconsistent. As a result, some methods have been modified.

4. If the error is reported for a third-party package, the possible cause is that the Spark has the related JAR file, but the version is different from that in the code.

## Procedure

Solution 1:

Check whether third-party tool packages are required. If the packages can be replaced with the packages of the same version in the cluster, modify dependency versions.

### NOTE

You are advised to use the dependencies provided by the MRS cluster.

Solution 2:

The following shows how to modify a JAR file version:

MRS\_2.1 is used as an example.

1. Add the **<properties>** parameter to the **pom.xml** file and set the variables to facilitate subsequent version modification.

```
<groupId>com.huawei</groupId>
<artifactId>SparkDemo</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 <spark.version>2.3.2-mrs-2.1</spark.version>
 <hbase.version>2.1.1.0101-mrs-2.1</hbase.version>
 <hadoop.version>3.1.1-mrs-2.1</hadoop.version>
</properties>
```

2. Use the preceding defined variables to set the version of each JAR file in the **dependencies** parameter.

```
<dependencies>
 <dependency>
 <groupId>org.apache.spark</groupId>
 <artifactId>spark-core_2.11</artifactId>
 <version>${spark.version}</version>
 </dependency>
 <dependency>
 <groupId>org.apache.hadoop</groupId>
 <artifactId>hadoop-common</artifactId>
 <version>${hadoop.version}</version>
 </dependency>
 <dependency>
 <groupId>org.apache.hbase</groupId>
 <artifactId>hbase-common</artifactId>
 <version>${hbase.version}</version>
 </dependency>
 <dependency>
 <groupId>org.apache.hbase</groupId>
 <artifactId>hbase-client</artifactId>
 <version>${hbase.version}</version>
 </dependency>
</dependencies>
```

If other third-party packages conflict, check whether the same package of different versions exists by querying the dependency relationship. If yes, replace the JAR file version with the version of the JAR file provided by the cluster.

You can [obtain sample projects from Huawei Mirrors](#) by referring to the **pom.xml** file of the MRS sample project.

3. Print the dependency tree.

Run the **mvn dependency:tree** command in the directory where the **pom.xml** file is stored.

# 31 Spark2x Development Guide (Normal Mode)

---

## 31.1 Overview

### 31.1.1 Application Development Overview

#### Spark Introduction

Spark is a distributed batch processing system as well as an analysis and mining engine. It provides an iterative memory computation framework and supports the development in multiple programming languages, including Scala, Java, and Python. The application scenarios of Spark include:

- Data processing: Spark can process data quickly and has fault tolerance and scalability.
- Iterative computation: Spark supports iterative computation to meet the requirements of multi-step data processing logic.
- Data mining: Based on massive data, Spark can handle complex data mining and analysis and support multiple data mining and machine learning algorithms.
- Streaming processing: Spark supports stream processing at a seconds-level delay and supports multiple external data sources.
- Query analysis: Spark supports standard SQL query analysis, provides the DSL (DataFrame), and supports multiple external inputs.

This section focuses on the application development guides of Spark, Spark SQL and Spark Streaming.

#### Spark Development API Introduction

Spark supports the development in multiple programming languages, including Scala, Java, and Python. Since Spark is developed in Scala and Scala is easy to read, users are advised to develop Spark application in Scala.

Divided by different languages, the APIs of Spark are listed in [Table 31-1](#).

**Table 31-1** Spark APIs

Function	Description
Scala API	Indicates the API in Scala. For common APIs of Spark Core, Spark SQL and Spark Streaming, see <a href="#">Scala</a> . Since Scala is easy to read, users are advised to use Scala APIs in the program development.
Java API	Indicates the API in Java. For common APIs of Spark Core, Spark SQL and Spark Streaming, see <a href="#">Java</a> .
Python API	Indicates the API in Python. For common APIs of Spark Core, Spark SQL and Spark Streaming, see <a href="#">Python</a> .

Divided by different modes, APIs listed in the preceding table are used in the development of Spark Core and Spark Streaming. Spark SQL supports CLI and JDBCServer for accessing. There are two ways to access the JDBCServer: Beeline and the JDBC client code. For details, see [JDBCServer Interface](#).

 **NOTE**

For spark-sql, spark-shell and spark-submit (which application contains SQL operations), do not use the **proxy user** parameter to submit a task.

## 31.1.2 Basic Concepts

### Basic Concepts

- **RDD**

Resilient Distributed Dataset (RDD) is a core concept of Spark. It indicates a read-only and partitioned distributed dataset. Partial or all data of this dataset can be cached in the memory and reused between computations.

**RDD Generation**

- An RDD can be generated from the Hadoop file system or other storage systems that are compatible with Hadoop, such as Hadoop Distributed File System (HDFS).
- A new RDD can be transferred from a parent RDD.
- An RDD can be converted from a collection.

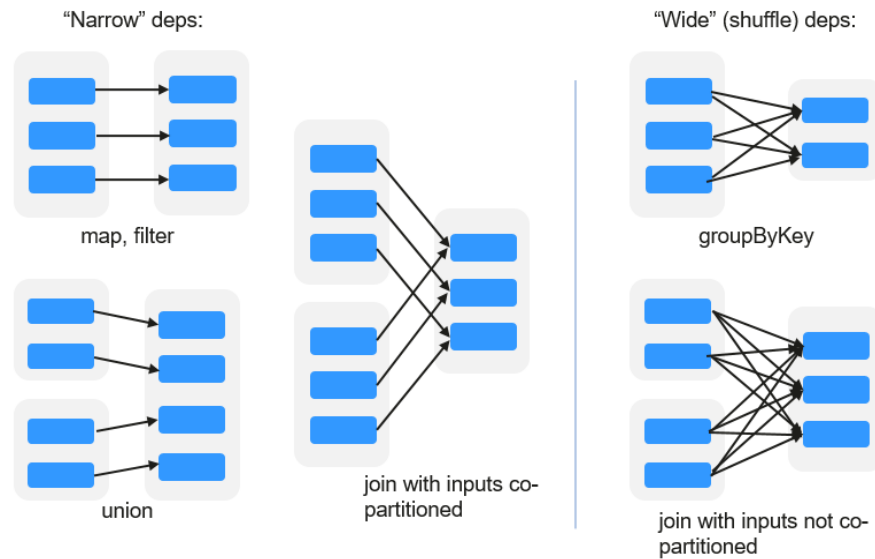
**RDD Storage**

- Users can select different storage levels to store an RDD for reuse. (There are 11 storage levels to store an RDD.)
- The current RDD is stored in the memory by default. When the memory is insufficient, the RDD overflows to the disk.

- **RDD Dependency**

The RDD dependency includes the narrow dependency and wide dependency.

**Figure 31-1** RDD dependency



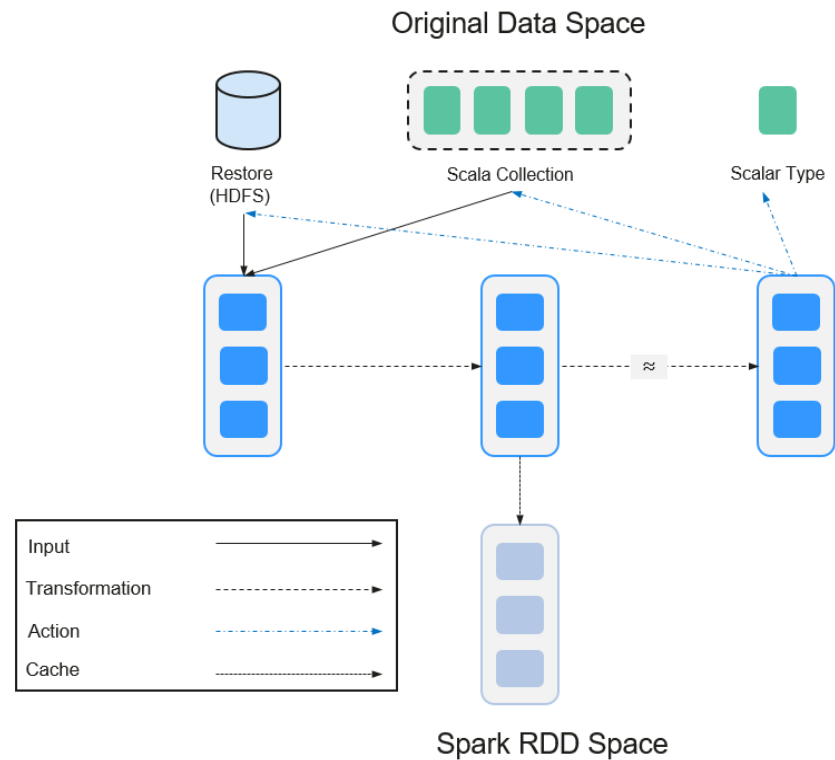
- **Narrow dependency:** Each partition of the parent RDD is used by at most one partition of the child RDD partition.
- **Wide dependency:** Partitions of the child RDD depend on all partitions of the parent RDD due to shuffle operations.

The narrow dependency facilitates the optimization. Logically, each RDD operator is a fork/join process. Fork the RDD to each partition, and then perform the computation. After the computation, join the results, and then perform the fork/join operation on next operator. It takes a long period of time to directly translate the RDD to physical implementation. There are two reasons: Each RDD (even the intermediate results) must be physicalized to the memory or storage, which takes time and space; the partitions can be joined only when the computation of all partitions is complete (if the computation of a partition is slow, the entire join process is slowed down). If the partitions of the child RDD narrowly depend on the partitions of the parent RDD, the two fork/join processes can be combined to optimize the entire process. If the relationship in the continuous operator sequence is narrow dependency, multiple fork/join processes can be combined to reduce the time for waiting and improve the performance. This is called pipeline optimization in Spark.

- **Transformation and Action (RDD Operations)**

Operations on RDD include transformation (the returned value is an RDD) and action (the returned value is not an RDD). [Figure 31-2](#) shows the process. The transformation is lazy, which indicates that the transformation from one RDD to another RDD is not immediately executed. Spark only records the transformation but does not execute it immediately. The real computation is started only when the action is started. The action returns results or writes the RDD data into the storage system. The action is the driving force for Spark to start the computation.

**Figure 31-2** RDD operation



The data and operation model of RDD are quite different from those of Scala.

```
val file = sc.textFile("hdfs://...")
val errors = file.filter(_.contains("ERROR"))
errors.cache()
errors.count()
```

- The **textFile** operator reads log files from the HDFS and returns **file** (as an RDD).
- The filter operator filters rows with ERROR and assigns them to errors (a new RDD). The filter operator is a transformation.
- The cache operator caches errors for future use.
- The count operator returns the number of rows of errors. The count operator is an action.

**Transformation includes the following types:**

- The RDD elements are regarded as simple elements:  
The input and output have the one-to-one relationship, and the partition structure of the result RDD remains unchanged, for example, map.  
The input and output have the one-to-many relationship, and the partition structure of the result RDD remains unchanged, for example, flatMap (one element becomes a sequence containing multiple elements and then flattens to multiple elements).
- The input and output have the one-to-one relationship, but the partition structure of the result RDD changes, for example, union (two RDDs integrates to one RDD, and the number of partitions becomes the sum of the number of partitions of two RDDs) and coalesce (partitions are reduced).

Operators of some elements are selected from the input, such as filter, distinct (duplicate elements are deleted), subtract (elements only exist in this RDD are retained), and sample (samples are taken).

- The RDD elements are regarded as Key-Value pairs.

Perform the one-to-one calculation on the single RDD, such as mapValues (the partition mode of the source RDD is retained, which is different from map).

Sort the single RDD, such as sort and partitionBy (partitioning with consistency, which is important to the local optimization).

Restructure and reduce the single RDD based on key, such as groupByKey and reduceByKey.

Join and restructure two RDDs based on key, such as join and cogroup.

#### NOTE

The later three operations involve sorting and are called shuffle operations.

#### **Action is classified into the following types:**

- Generate scalar configuration items, such as count (the number of elements in the returned RDD), reduce, fold/aggregate (the number of scalar configuration items that are returned), and take (the number of elements before the return).
- Generate the Scala collection, such as collect (import all elements in the RDD to the Scala collection) and lookup (look up all values corresponds to the key).
- Write data to the storage, such as saveAsTextFile (which corresponds to the preceding textFile).
- Check points, such as checkpoint. When Lineage is quite long (which occurs frequently in graphics computation), it takes a long period of time to execute the whole sequence again when a fault occurs. In this case, checkpoint is used as the check point to write the current data to stable storage.

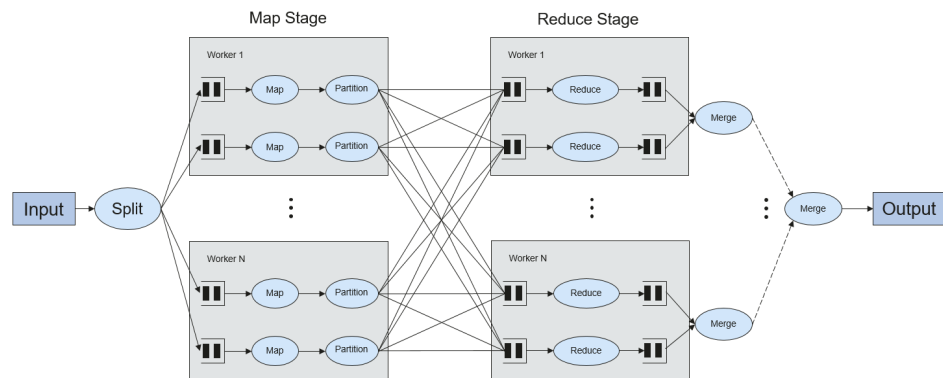
- **Shuffle**

Shuffle is a specific phase in the MapReduce framework, which is located between the Map phase and the Reduce phase. If the output results of Map are to be used by Reduce, each output result must be hashed based on the key and distributed to the corresponding Reducer. This process is called Shuffle. Shuffle involves the read and write of the disk and the transmission of the network, so that the performance of Shuffle directly affects the operation efficiency of the entire program.

The following figure describes the entire process of the MapReduce algorithm.



**Figure 31-3** Algorithm process



Shuffle is a bridge connecting data. The following describes the implementation of shuffle in Spark.

Shuffle divides the Job of a Spark into multiple stages. The former stages contain one or multiple ShuffleMapTasks, and the last stage contains one or multiple ResultTasks.

- **Spark Application Structure**

The Spark application structure includes the initial SparkContext and the main program.

- Initial SparkContext: constructs the operation environment of the Spark application.

Constructs the SparkContext object. For example:

```
new SparkContext(master, appName, [SparkHome], [jars])
```

Parameter description

**master:** indicates the link string. The link modes include local, YARN-cluster, and YARN-client.

**appName:** indicates the application name.

**SparkHome:** indicates the directory where Spark is installed in the cluster.

**jars:** indicates the code and dependency package of the application.

- Main program: processes data.

For submitting applications details, see <https://spark.apache.org/docs/3.1.1/submitting-applications.html>

- **Spark Shell Command**

The basic Spark shell command supports the submitting of the Spark application. The Spark shell command is as follows.

```
./bin/spark-submit \
--class <main-class> \
--master <master-url> \
... # other options \
<application-jar> \
[application-arguments]
```

Parameter description:

--class: indicates the name of the class of the Spark application.

--master: indicates the master that the Spark application links, such as YARN-client and YARN-cluster.

application-jar: indicates the path of the jar package of the Spark application.

application-arguments: indicates the parameter required to submit the Spark application. (This parameter can be empty.)

- **Spark JobHistory Server**

The Spark web UI is used to monitor the details in each phase of the Spark framework of a running or historical Spark job and provide the log display, which helps users to develop, configure, and optimize the job in more fine-grained units.

## Spark SQL Basic Concepts

### DataSet

A DataSet is a strongly typed collection of domain-specific objects that can be transformed in parallel using functional or relational operations. Each Dataset also has an untyped view called a DataFrame, which is a Dataset of Row.

DataFrame is a structured distributed data set composed of several columns, which is similar to a table in the relationship database or the data frame in R/Python. DataFrame is a basic concept in Spark SQL, and can be created by using multiple methods, such as structured data set, Hive table, external database, or RDD.

## Spark Streaming Basic Concepts

### Dstream

The DStream (Discretized Stream) is an abstract concept provided by the Spark Streaming.

The DStream is a continuous data stream which is obtained from the data source or transformed and generated by the inflow. In essence, a DStream is a series of continuous RDDs. The RDD is a distributed dataset which can be read only and divided into partitions.

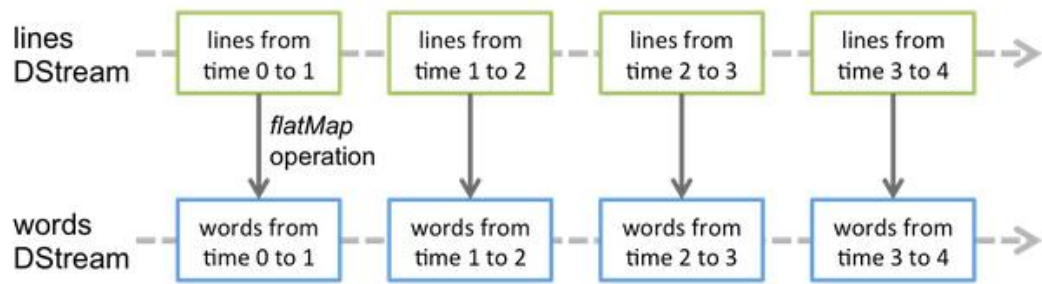
Each RDD in the DStream contains the data of a partition, as shown in [Figure 31-4](#).

**Figure 31-4** Relationship between DStream and RDD



All operators applied in the DStream are translated to the operations in the lower RDD, as shown in [Figure 31-5](#). The transformation of the lower RDDs is calculated by using the Spark engine. Most operation details are concealed in the DStream operators and High-level APIs are provided for developers.

**Figure 31-5** DStream operator transfer



## Structured Streaming Basic Concepts

- **Input Source**

Input data sources. Data sources must support data replay based on the offset. Different data sources have different fault tolerance capabilities.

- **Sink**

Data output. Sink must support idempotence write operations. Different Sinks have different fault tolerance capabilities.

- **outputMode**

Result output mode, which can be:

- Complete Mode: The entire updated result table will be written to the external storage. It is up to the storage connector to decide how to handle writing of the entire table.
- Append Mode: If an interval is triggered, only the new rows appended in the Result Table will be written into an external system. This mode is applicable only to a result set that has already existed and will not be updated.
- Update Mode: If an interval is triggered, only updated data in the result table will be written into an external system, which is the difference between the Complete Mode and Update Mode.

- **Trigger**

Output trigger. Currently, the following trigger types are supported:

- Default: Micro-batch mode. After a batch is complete, the next batch is automatically executed.
- Specific interval: Processing is performed at a specific interval.
- One-time execution: Query is performed only once.
- Continuous mode: This is an experimental feature. In this mode, the stream processing delay can be decreased to 1 ms.

Structured Streaming supports the micro-batch mode and continuous mode. The micro-batch mode cannot ensure low-delay but has a larger throughput within the same time. The continuous mode is suitable for data processing requiring millisecond-level delay, which is an experimental feature.

### NOTE

In the current version, if the stream-to-batch joins function is required, **outputMode** must be set to **Append Mode**.

Figure 31-6 Running process in micro-batch mode

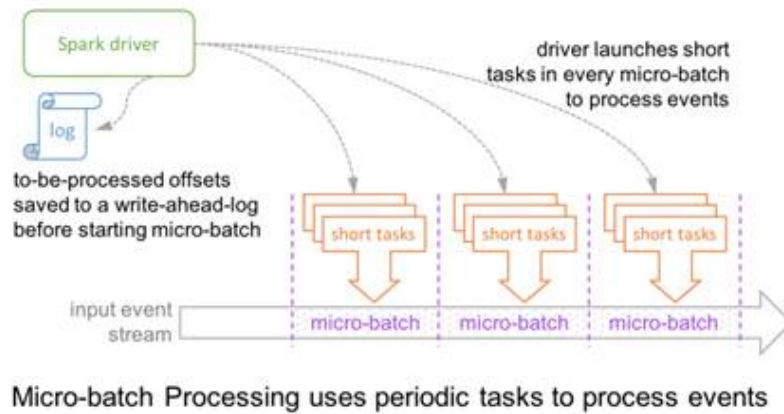
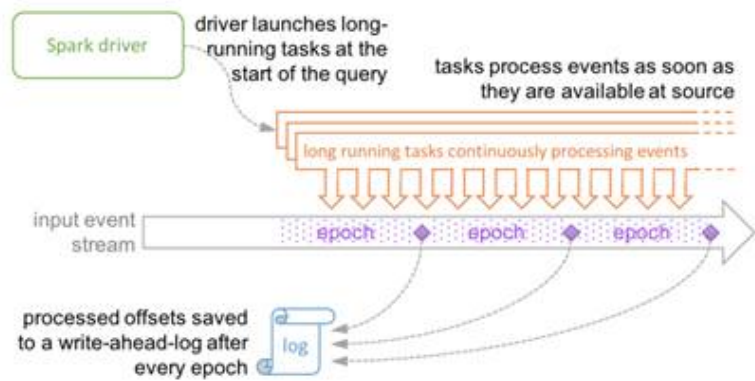


Figure 31-7 Running process in continuous mode

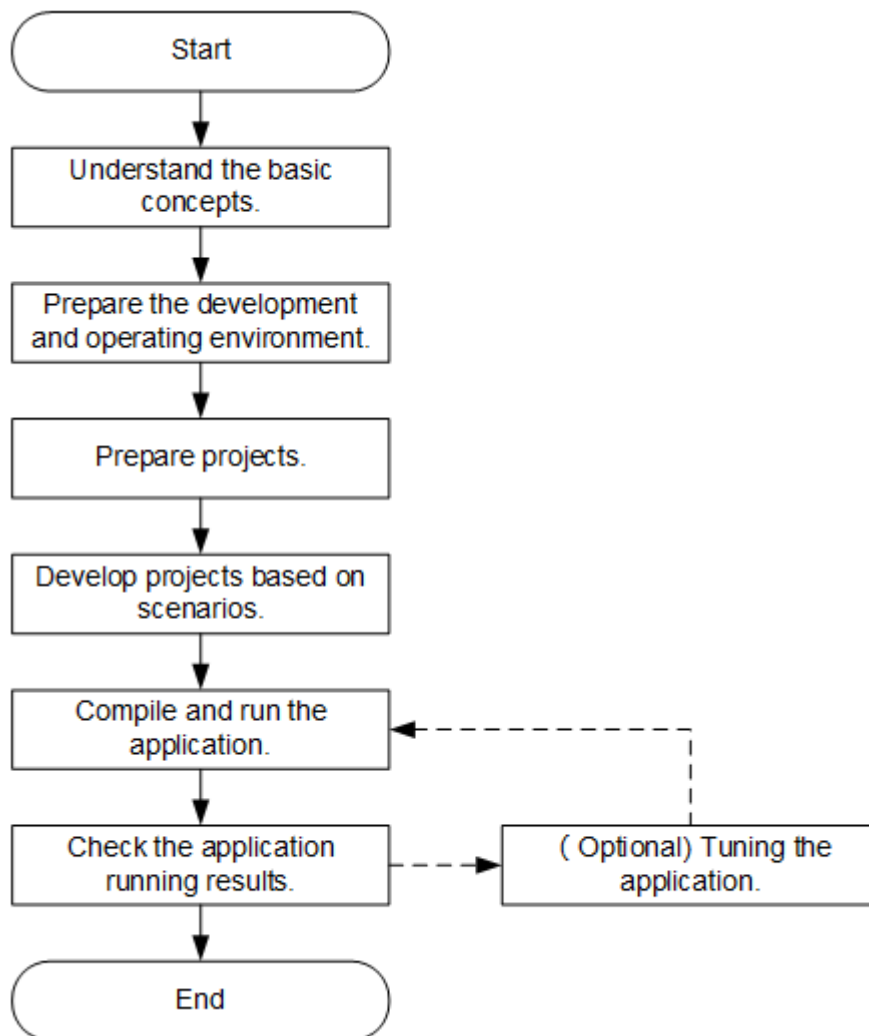


### 31.1.3 Development Process

Spark includes Spark Core, Spark SQL and Spark Streaming, whose development processes are the same.

Figure 31-8 and Table 31-2 describe the stages in the development process.

**Figure 31-8** Spark development process



**Table 31-2** Description of Spark development process

Stage	Description	Reference
Understand the basic concepts.	Before the application development, the basic concepts of Spark are required to be understood. Choose the concepts required to be understood based on the actual scenario. The basic concepts include the basic concept of Spark Core, basic concept of Spark SQL and basic concept of Spark Streaming.	<a href="#">Basic Concepts</a>
Prepare the development and operating environment.	The Spark application is developed in Scala, Java, and Python. The IDEA tool is recommended to prepare development environments in different languages based on the reference. The running environment of Spark is the Spark client. Install and configure the client based on the reference.	<a href="#">Development and Operating Environment</a>

Stage	Description	Reference
Prepare projects.	Spark provides sample projects in various scenarios. sample projects can be imported for studying. Or you can create a new Spark project based on the reference.	<a href="#">Configuring and Importing Sample Projects</a> <a href="#">Creating a New Project (Optional)</a>
Develop projects based on scenarios.	Sample projects in different languages including Scala, Java, and Python are provided. Sample projects in different scenarios including Streaming, SQL, JDBC client program, and Spark on HBase are also provided. This helps users to learn about the programming interfaces of all Spark components quickly.	<a href="#">Developing the Project</a>
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	<a href="#">Compiling and Running the Application</a>
Check the application running results.	Application running results are stored in the directory specified by users. Users can also check the running results through the UI.	<a href="#">Checking the Commissioning Result</a>
Tune the application.	Based on the application running results, tune the application to meet the requirements of the service scenario. After application tuning, compile and run the application again.	<a href="#">Spark2x Performance Optimization</a>

## 31.2 Preparing for the Environment

### 31.2.1 Development and Operating Environment

[Table 31-3](#) describes the environment required for secondary development.

**Table 31-3** Development environment

Preparation Item	Description
OS	<ul style="list-style-type: none"> <li>Development environment: Windows OS. Windows 7 or later is supported.</li> <li>Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the runtime environment must be able to communicate with the cluster service plane network.</li> </ul>
JDK installation	<p>Basic configuration for the Java/Scala development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> <li>For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> <li>Oracle JDK versions: 1.8</li> <li>Supported IBM JDK versions: 1.8.5.11</li> </ul> </li> <li>For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> <li>OpenJDK: 1.8.0_272</li> </ul> </li> </ul> <p><b>NOTE</b> The server supports only TLS V1.2 or later to ensure security. By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls">https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls</a>.</p>
IntelliJ IDEA installation and configuration	<p>It is a tool used to develop Spark applications. Version 2019.1 or other compatible versions are recommended.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li> <li>If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li> <li>If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li> <li>Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li> </ul>
Installation of Maven	<p>Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.</p>

Preparation Item	Description
Scala installation	It is the basic configuration for the Scala development environment. The required version is 2.12.10.
Scala plug-in installation	It is the basic configuration for the Scala development environment. The required version is 2018.2.11 or other compatible versions.
Editra installation	Editra is an editor in the Python development environment and is used to compile Python programs. You can also use other IDEs for Python programming.
7-zip	Used to decompress <b>.zip</b> and <b>.rar</b> packages, 7-Zip 16.04 is supported.
Python installation	Its version must be 3.6 or later.

## Preparing a Runtime Environment

During application development, you need to prepare the environment for code running and commissioning to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.

- a. [Log in to the FusionInsight Manager portal](#) and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Configuration Files Only**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is

**FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain the **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file, and then decompress it to the **D:**

**\FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles\Spark2x\config** and manually import the configuration file to the configuration file directory (usually the **resources** folder) of the Hive sample project.

[Table 31-4](#) describes the main configuration files.



**Table 31-4** Configuration files

File	Function
carbon.properties	CarbonData configuration file
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.
hbase-site.xml	Configures HBase parameters.
hive-site.xml	Configures Hive parameters.
jaas-zk.conf	Java authentication configuration file
log4j-executor.properties	executor log configuration file
mapred-site.xml	Hadoop mapreduce configuration file
ranger-spark-audit.xml	Ranger audit log configuration file
ranger-spark-security.xml	Ranger permission management configuration file
yarn-site.xml	Configures Yarn parameters.
spark-defaults.conf	Configures Spark2x parameters.
spark-env.sh	Spark2x environment variable configuration file

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
  - The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
- a. Install the client in a directory, for example, **/opt/client**, on the node. Ensure that the difference between the client time and the cluster time is less than 5 minutes.
- For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. **Log in to the FusionInsight Manager portal.** Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Spark2x/config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/client/conf**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Spark2x/config/* root@IP address of the client node:/opt/client/conf
```

**Table 31-5** Configuration files

File	Function
carbon.properties	CarbonData configuration file
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.
hbase-site.xml	Configures HBase parameters.
hive-site.xml	Configures Hive parameters.
jaas-zk.conf	Java authentication configuration file
log4j-executor.properties	executor log configuration file
mapred-site.xml	Hadoop mapreduce configuration file
ranger-spark-audit.xml	Ranger audit log configuration file
ranger-spark-security.xml	Ranger permission management configuration file
yarn-site.xml	Configures Yarn parameters.
spark-defaults.conf	Configures Spark2x parameters.
spark-env.sh	Spark2x environment variable configuration file

- c. Check the network connection of the client node.  
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no,

manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

## 31.2.2 Configuring and Importing Sample Projects

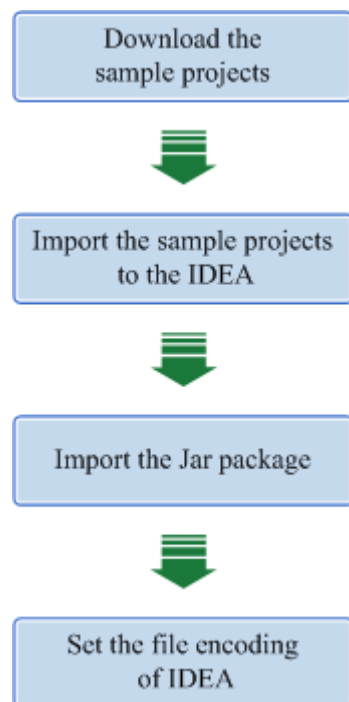
### Scenario

Spark provides sample projects for multiple scenarios, including Java projects and Scala projects. This helps users to learn Spark projects quickly.

Import methods of Java and Scala projects are the same. Sample projects developed by using the Python do not need to be imported, and you only need to open the Python file (\*.py).

The import of Java sample codes is used as a sample in the following procedure. [Figure 31-9](#) shows the procedure of importing sample projects.

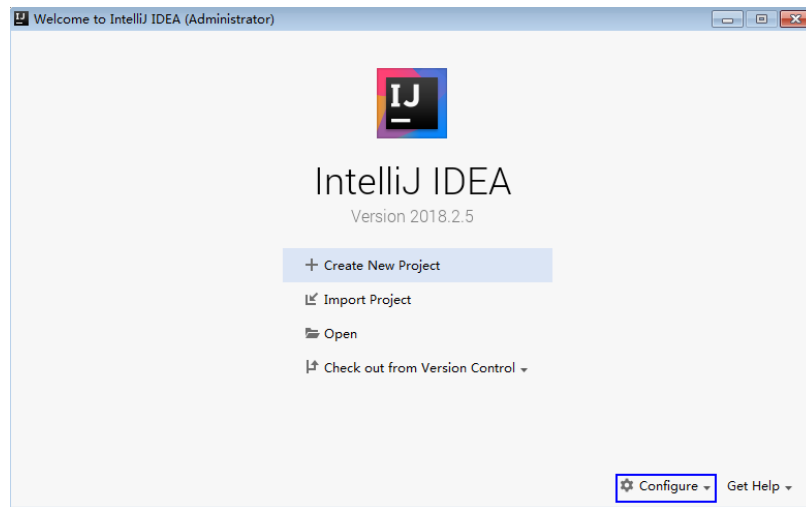
**Figure 31-9** Procedure of importing sample projects



### Procedure

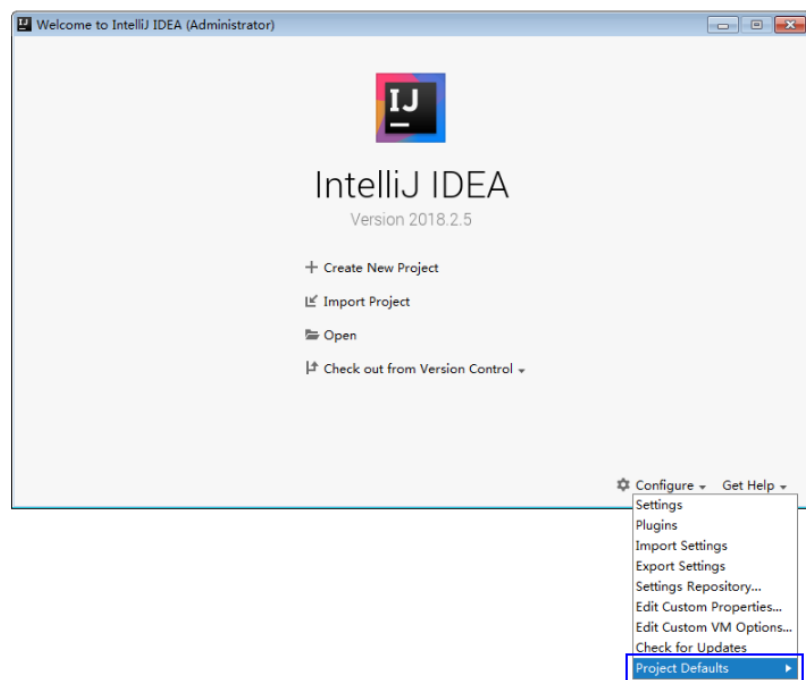
- Step 1** Obtain multiple sample projects such as Scala and Spark Streaming in the **sparknormal-examples** folder in the **spark-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#)
- Step 2** After the IntelliJ IDEA and JDK are installed, configure the JDK in IntelliJ IDEA.
  1. Start the IntelliJ IDEA and select **Configure**.

**Figure 31-10** Quick Start



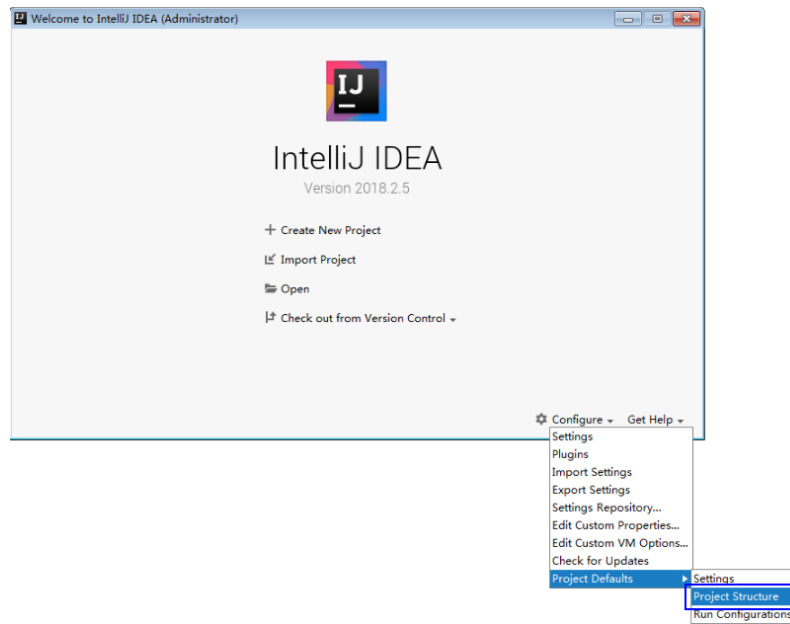
2. Select **Project Defaults** from the **Configure** drop-down list.

**Figure 31-11** Configure



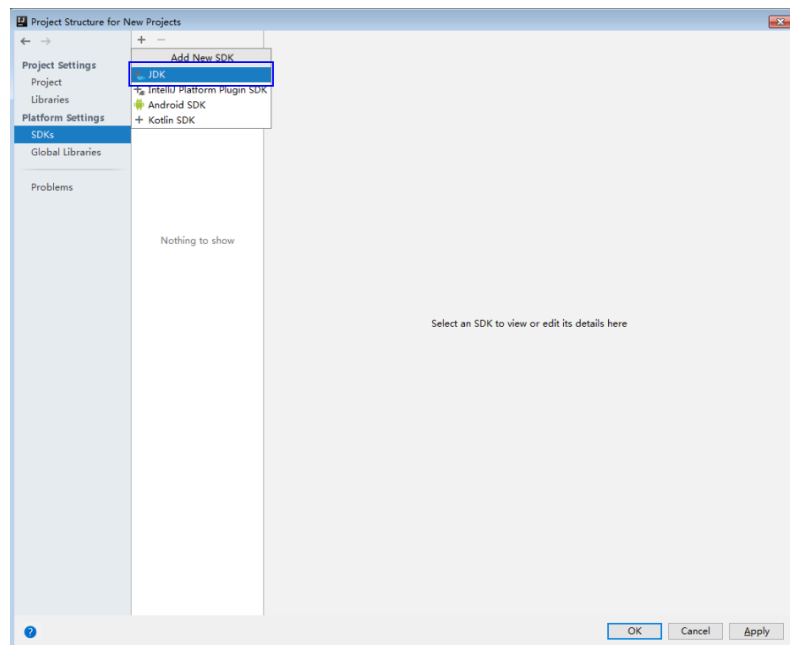
3. Select **Project Structure** from **Project Defaults**.

Figure 31-12 Project Defaults



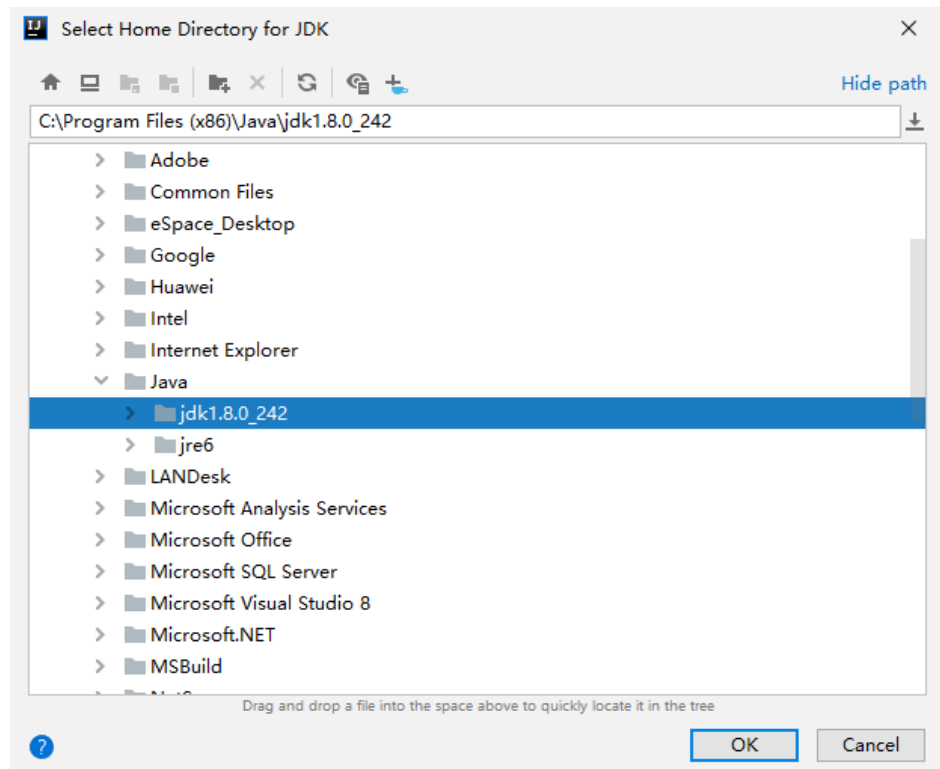
4. On the displayed **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 31-13 Adding the JDK



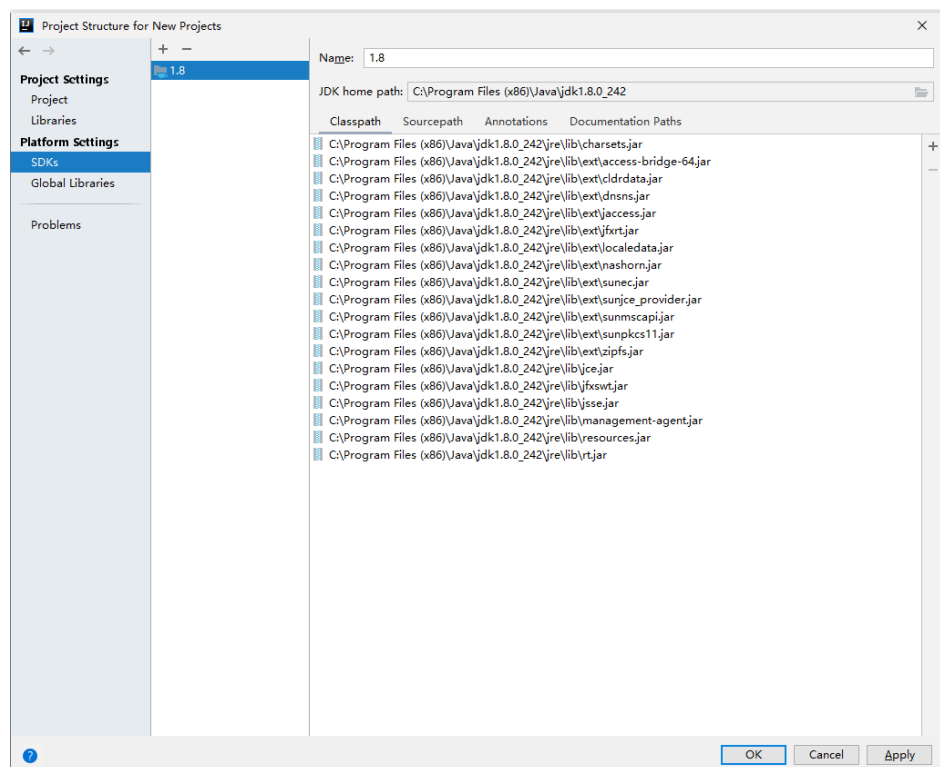
5. In the displayed **Select Home Directory for JDK** window, select a home directory for the JDK and click **OK**.

**Figure 31-14** Selecting a home directory for the JDK



6. After selecting the JDK, click **OK** to complete the configuration.

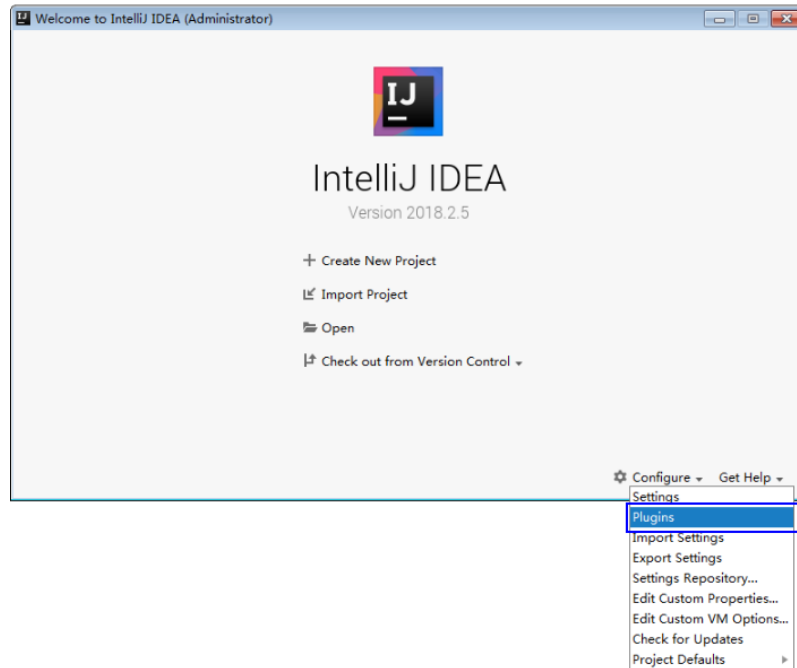
**Figure 31-15** Completing the configuration



**Step 3** (Optional) If the Scala development environment is used, install the Scala plug-in in IntelliJ IDEA.

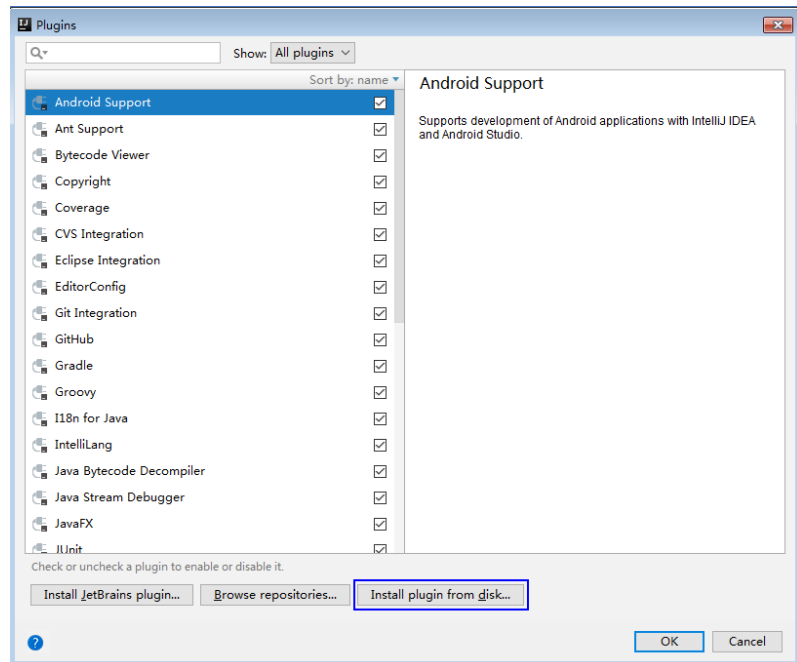
1. Select **Plugins** from the **Configure** drop-down list.

**Figure 31-16** Plugins

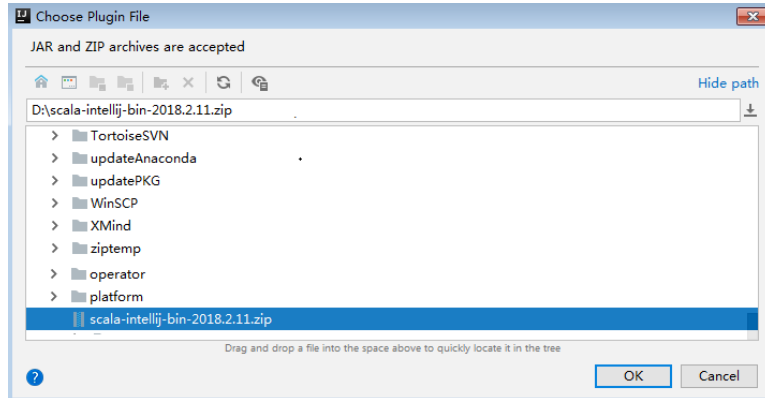


2. On the **Plugins** page, select **Install plugin from disk**.

**Figure 31-17** Install plugin from disk

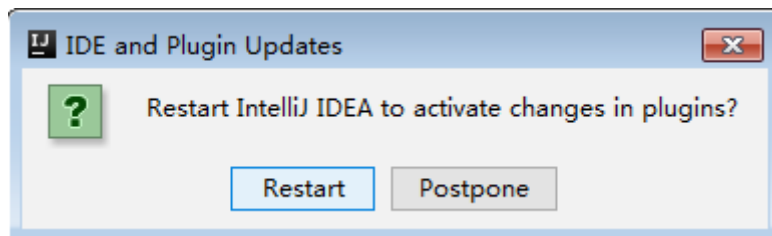


3. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.



4. On the **Plugins** page, click **Apply** to install the Scala plugin.
5. On the displayed **Plugins Changed** page, click **Restart** for the configuration to take effect.

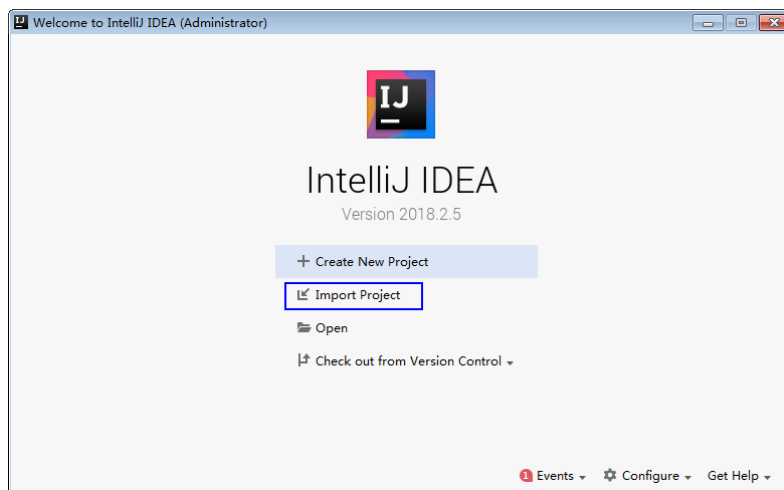
**Figure 31-18** Plugins Changed



**Step 4** Import the Java sample projects to the IDEA.

1. Start the IntelliJ IDEA. On the **Quick Start** page, select **Import Project**.  
Alternatively, for the used IDEA tool, add the project directly from the IDEA home page. Select **File > Import project...** to import projects.

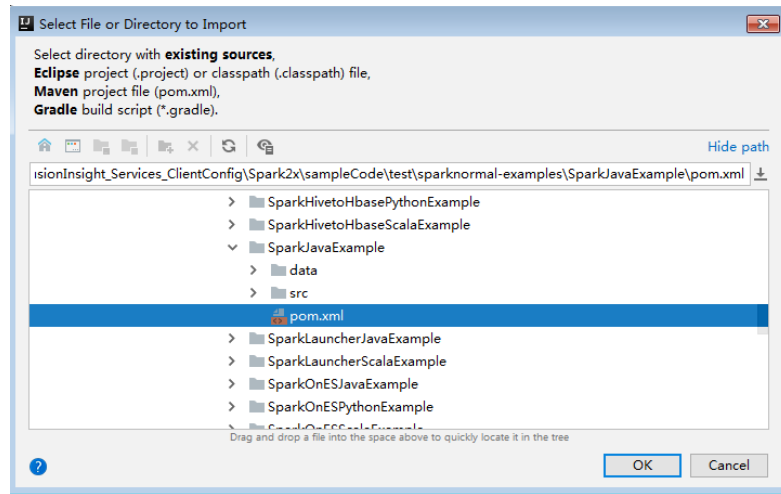
**Figure 31-19** Import Project (on the Quick Start page)



2. Select the directory to store the imported projects and the pom file, and click **OK**.

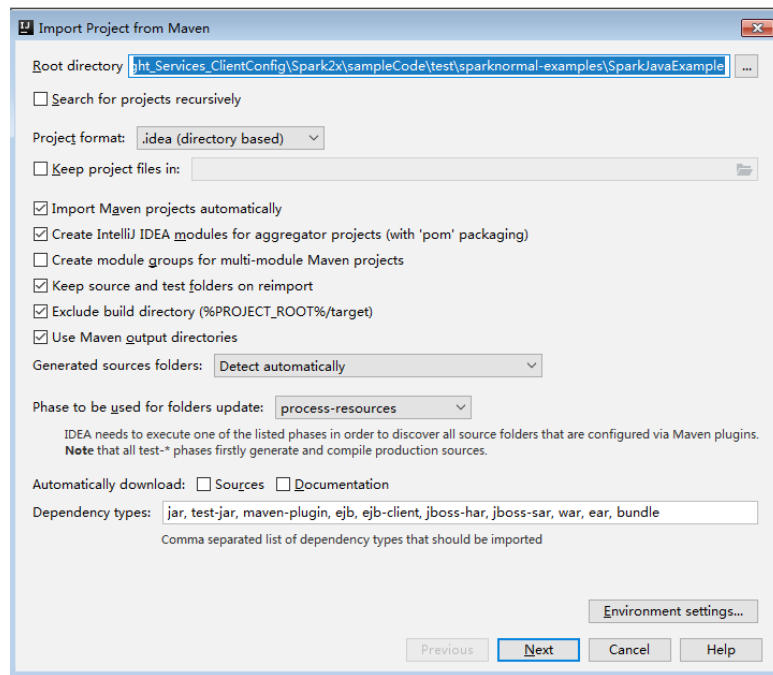


Figure 31-20 Select File or Directory to Import



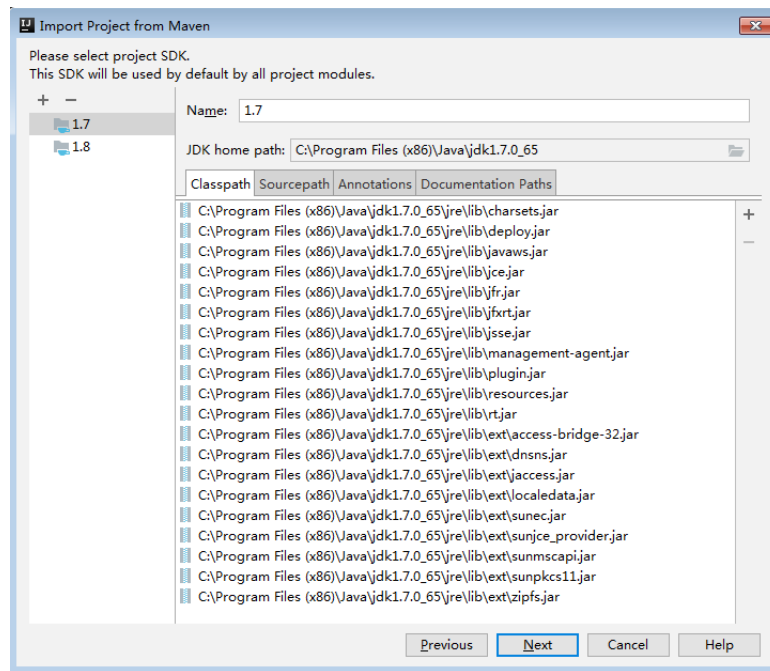
3. Confirm the import directory and project name, and click **Next**.

Figure 31-21 Import Project from Maven



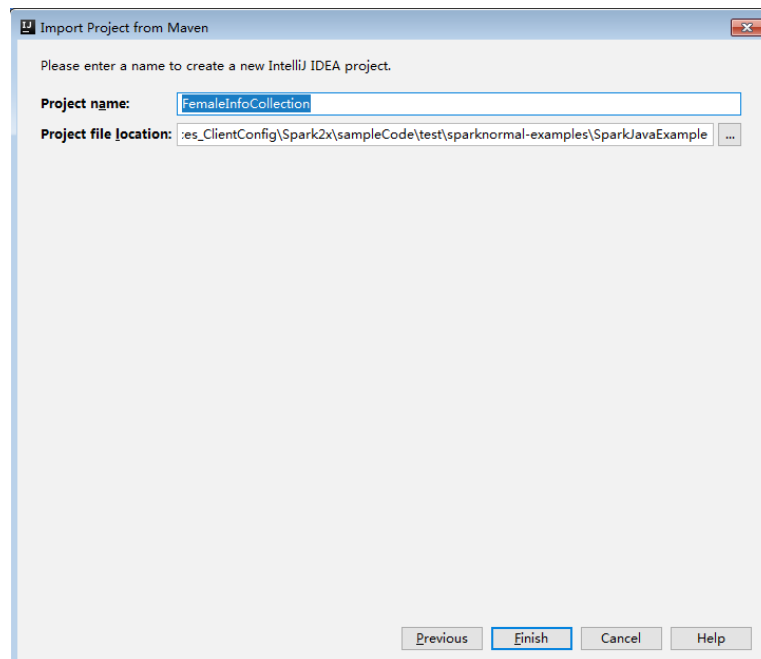
4. Select the projects to import and click **Next**.
5. Confirm the project JDK and click **Next**.

Figure 31-22 Select project SDK



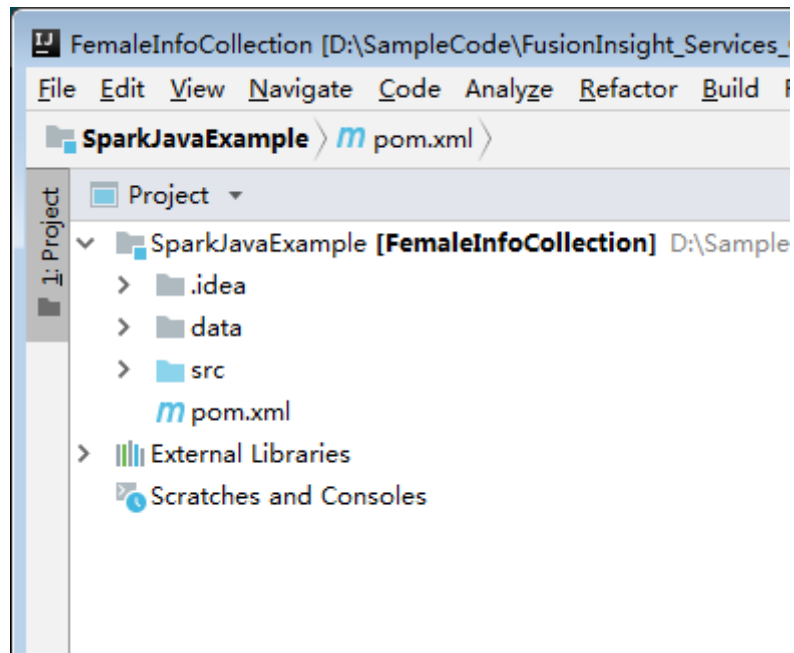
6. Confirm the project name and project file location, and click **Finish** to complete the import.

Figure 31-23 Confirm the project name and file location



7. After the import, the imported projects are displayed on the IDEA home page.

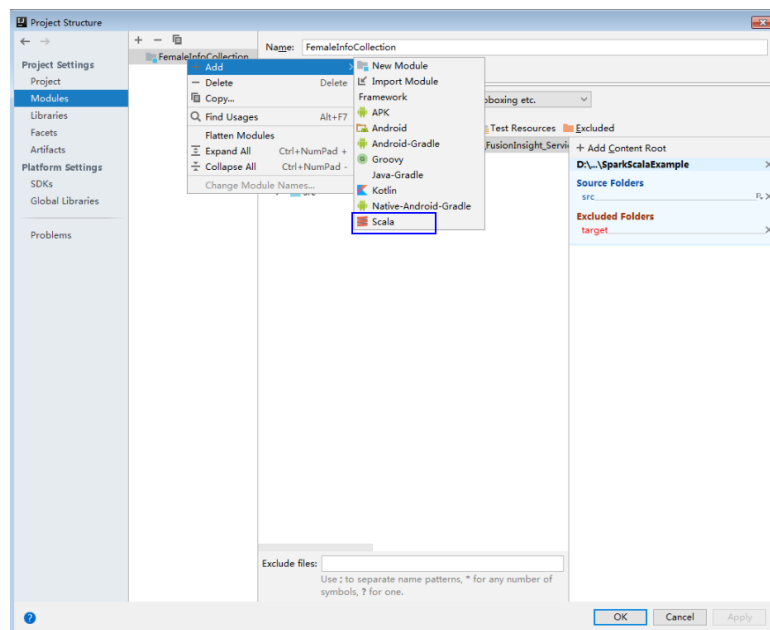
Figure 31-24 Imported projects



**Step 5** (Optional) If a sample application developed in Scala is imported, configure the language for the project.

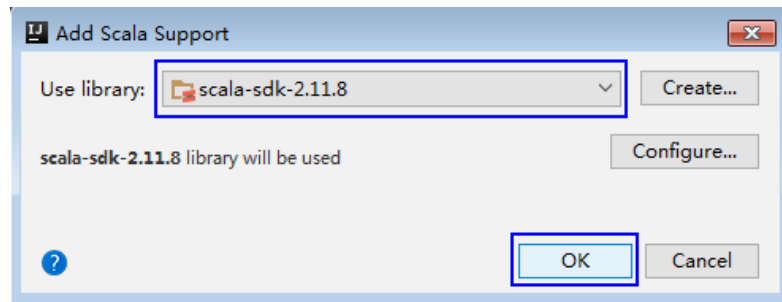
1. On the main page of the IDEA, choose **File > Project Structures...** to access the **Project Structure** page.
2. Choose **Modules**, right-click the project name, and choose **Add > Scala**.

Figure 31-25 Selecting the Scala language



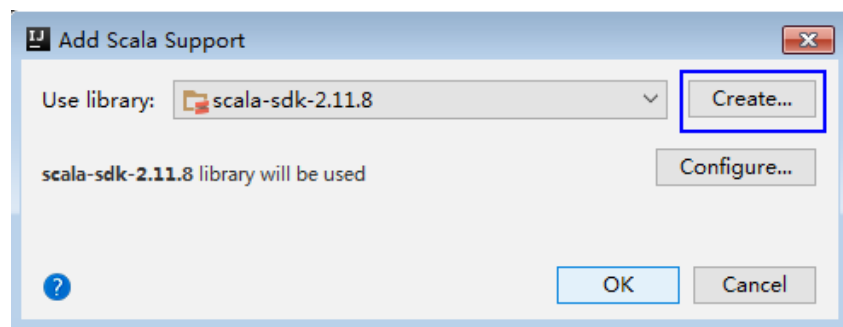
3. Wait until IDEA identifies Scala SDK, select the dependency JAR packages in the **Add Scala Support** dialog box, and then click **OK**.

**Figure 31-26** Add Scala Support



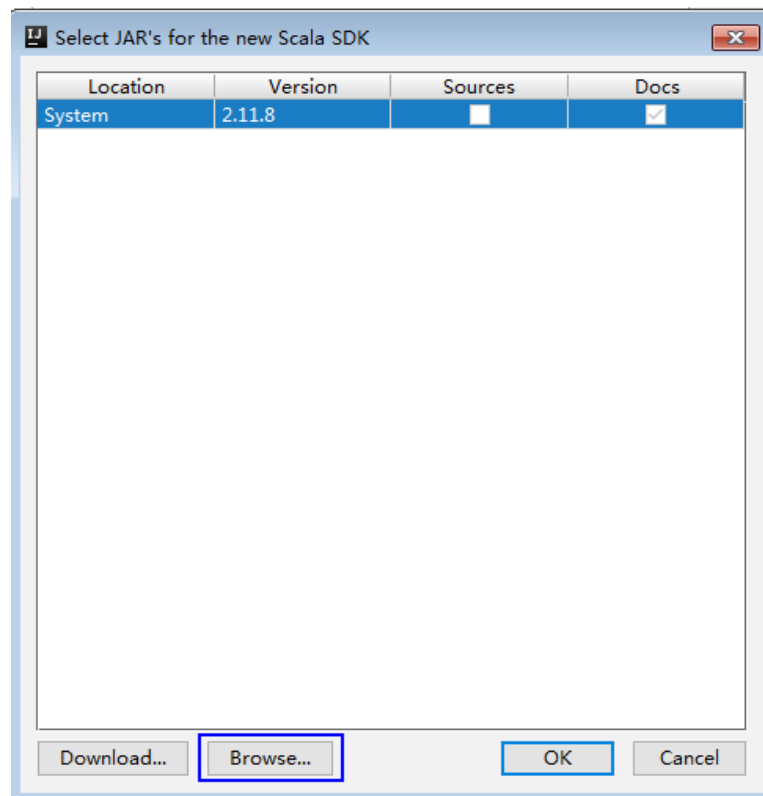
4. If IDEA fails to identify Scala SDK, you are required to create a Scala SDK.
  - a. Click **Create...**

**Figure 31-27** Create...



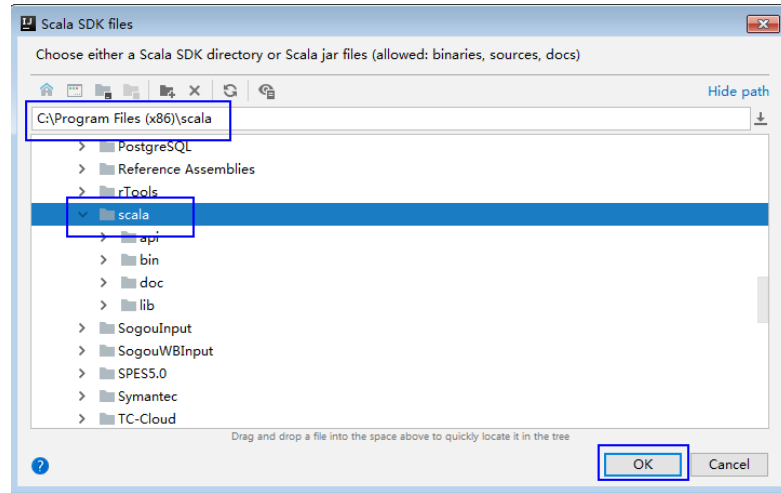
- b. On the **Select JAR's for the new Scala SDK** page, click **Browse...**

**Figure 31-28** Select JAR's for the new Scala SDK



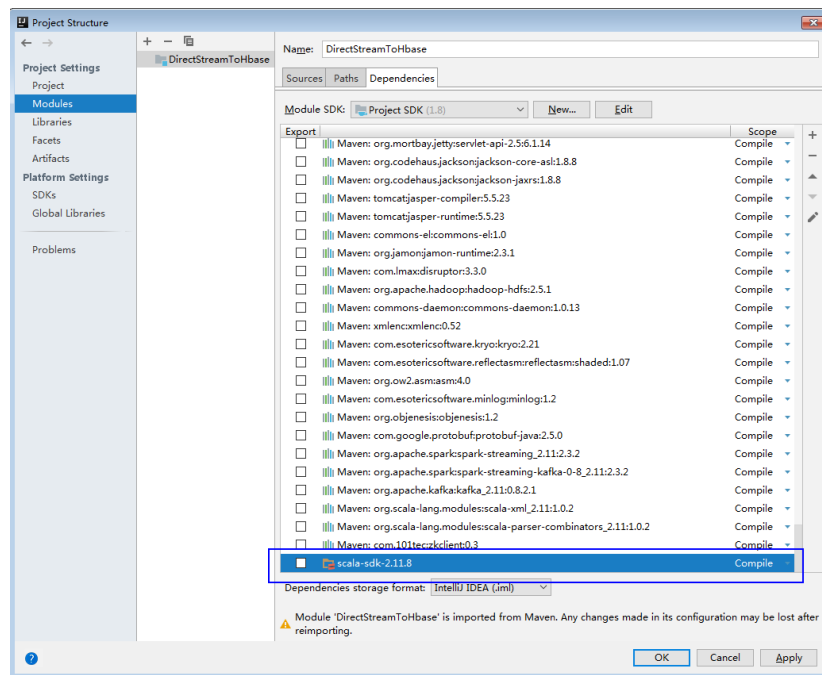
- c. On the **Scala SDK files** page, select the **scala sdk** directory, and then click **OK**.

**Figure 31-29** Scala SDK files



5. Click **OK**.

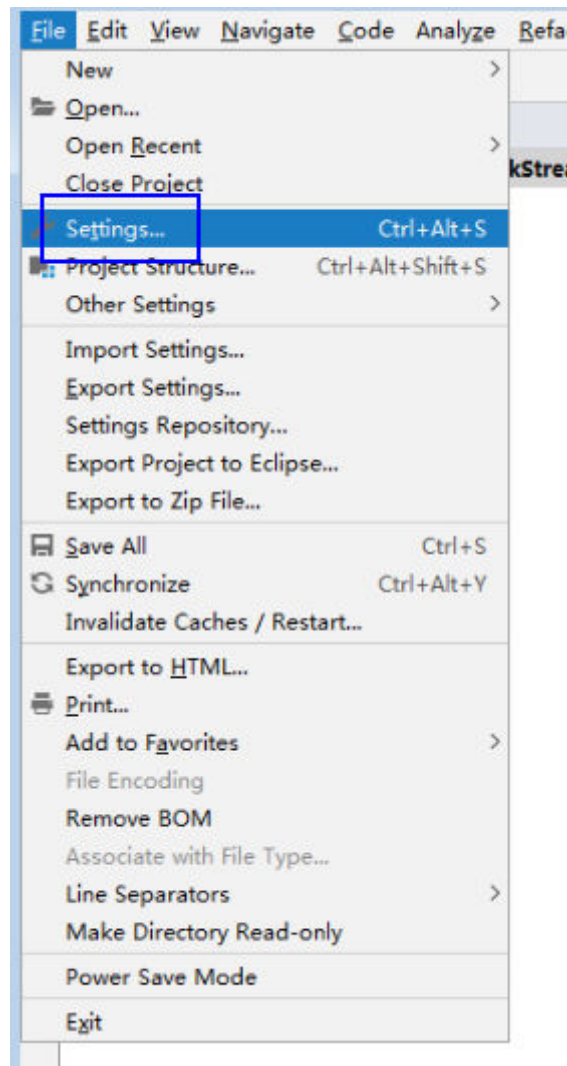
**Figure 31-30** Successful configuration



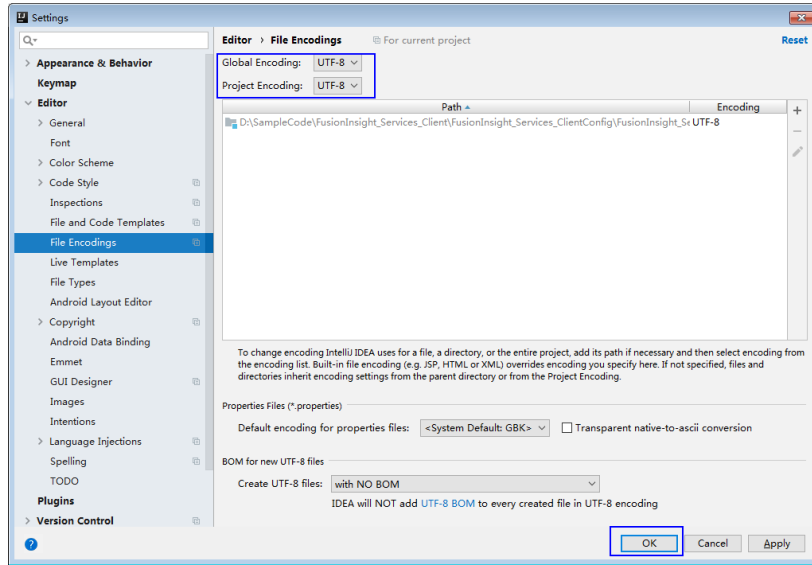
**Step 6** Set the file encoding of IDEA and solve the display of garble characters.

1. On the IDEA home page, choose **File > Settings...**

Figure 31-31 Choosing Settings



2. Configure the encoding.
  - a. On the **Settings** page, choose **Editor > File Encodings**.
  - b. In the **Global Encoding** and **Project Encoding** drop-down lists, select **UTF-8**, respectively.
  - c. Click **Apply**.
  - d. Click **OK** to complete the encoding configuration.



----End

## Sample Code Path Description

Table 31-6 Sample code path description

Sample Code Project	Sample Name	Sample Development Language
SparkJavaExample	Spark Core Project	Java
SparkScalaExample	Spark Core Project	Scala
SparkPythonExample	Spark Core Project	Python
SparkSQLJavaExample	Spark SQL Project	Java
SparkSQLScalaExample	Spark SQL Project	Scala
SparkSQLPythonExample	Spark SQL Project	Python
SparkThriftServerJavaExample	Accessing the Spark SQL Through JDBC	Java
SparkThriftServerScalaExample	Accessing the Spark SQL Through JDBC	Scala
SparkOnHbaseJavaExample-AvroSource	Spark on HBase-Performing Operations on Data in Avro Format	Java
SparkOnHbaseScalaExample-AvroSource	Spark on HBase-Performing Operations on Data in Avro Format	Scala

Sample Code Project	Sample Name	Sample Development Language
SparkOnHbasePythonExample-AvroSource	Spark on HBase-Performing Operations on Data in Avro Format	Python
SparkOnHbaseJavaExample-HbaseSource	Spark on HBase-Performing Operations on the HBase Data Source	Java
SparkOnHbaseScalaExample-HbaseSource	Spark on HBase-Performing Operations on the HBase Data Source	Scala
SparkOnHbasePythonExample-HbaseSource	Spark on HBase-Performing Operations on the HBase Data Source	Python
SparkOnHbaseJavaExample-JavaHBaseBulkPutExample	Spark on HBase-Using the BulkPut Interface	Java
SparkOnHbaseScalaExample-HBaseBulkPutExample	Spark on HBase-Using the BulkPut Interface	Scala
SparkOnHbasePythonExample-HBaseBulkPutExample	Spark on HBase-Using the BulkPut Interface	Python
SparkOnHbaseJavaExample-JavaHBaseBulkGetExample	Spark on HBase-Using the BulkGet Interface	Java
SparkOnHbaseScalaExample-HBaseBulkGetExample	Spark on HBase-Using the BulkGet Interface	Scala
SparkOnHbasePythonExample-HBaseBulkGetExample	Spark on HBase-Using the BulkGet Interface	Python
SparkOnHbaseJavaExample-JavaHBaseBulkDeleteExample	Spark on HBase-Using the BulkDelete Interface	Java
SparkOnHbaseScalaExample-HBaseBulkDeleteExample	Spark on HBase-Using the BulkDelete Interface	Scala
SparkOnHbasePythonExample-HBaseBulkDeleteExample	Spark on HBase-Using the BulkDelete Interface	Python
SparkOnHbaseJavaExample-JavaHBaseBulkLoadExample	Spark on HBase-Using the BulkLoad Interface	Java
SparkOnHbaseScalaExample-HBaseBulkLoadExample	Spark on HBase-Using the BulkLoad Interface	Scala
SparkOnHbasePythonExample-HBaseBulkLoadExample	Spark on HBase-Using the BulkLoad Interface	Python



Sample Code Project	Sample Name	Sample Development Language
SparkOnHbaseJavaExample-JavaHBaseForEachPartitionExample	Spark on HBase-Using the foreachPartition Interface	Java
SparkOnHbaseScalaExample-HBaseForEachPartitionExample	Spark on HBase-Using the foreachPartition Interface	Scala
SparkOnHbasePythonExample-HBaseForEachPartitionExample	Spark on HBase-Using the foreachPartition Interface	Python
SparkOnHbaseJavaExample-JavaHBaseDistributedScanExample	Spark on HBase-Distributedly Scanning HBase Tables	Java
SparkOnHbaseScalaExample-HBaseDistributedScanExample	Spark on HBase-Distributedly Scanning HBase Tables	Scala
SparkOnHbasePythonExample-HBaseDistributedScanExample	Spark on HBase-Distributedly Scanning HBase Tables	Python
SparkOnHbaseJavaExample-JavaHBaseMapPartitionExample	Spark on HBase-Using the mapPartition Interface	Java
SparkOnHbaseScalaExample-HBaseMapPartitionExample	Spark on HBase-Using the mapPartition Interface	Scala
SparkOnHbasePythonExample-HBaseMapPartitionExample	Spark on HBase-Using the mapPartition Interface	Python
SparkOnHbaseJavaExample-JavaHBaseStreamingBulkPutExample	Spark on HBase-Writing Data to HBase Tables In Batches Using SparkStreaming	Java
SparkOnHbaseScalaExample-HBaseStreamingBulkPutExample	Spark on HBase-Writing Data to HBase Tables In Batches Using SparkStreaming	Scala
SparkOnHbasePythonExample-HBaseStreamingBulkPutExample	Spark on HBase-Writing Data to HBase Tables In Batches Using SparkStreaming	Python
SparkHbasetoHbaseJavaExample	Reading Data from HBase and Write It Back to HBase	Java
SparkHbasetoHbaseScalaExample	Reading Data from HBase and Write It Back to HBase	Scala

Sample Code Project	Sample Name	Sample Development Language
SparkHbaseToHbasePythonExample	Reading Data from HBase and Write It Back to HBase	Python
SparkHiveToHbaseJavaExample	Reading Data from Hive and Write It to HBase	Java
SparkHiveToHbaseScalaExample	Reading Data from Hive and Write It to HBase	Scala
SparkHiveToHbasePythonExample	Reading Data from Hive and Write It to HBase	Python
SparkStreamingKafka010JavaExample	Streaming Connecting to Kafka0-10	Java
SparkStreamingKafka010ScalaExample	Streaming Connecting to Kafka0-10	Scala
SparkStructuredStreamingJavaExample	Structured Streaming Project	Java
SparkStructuredStreamingScalaExample	Structured Streaming Project	Scala
SparkStructuredStreamingPythonExample	Structured Streaming Project	Python
StructuredStreamingADScalaExample	Structured Streaming Stream-Stream Join	Scala
StructuredStreamingStateScalaExample	Structured Streaming Status Operation	Scala
SparkOnHudiJavaExample	Using Spark to Perform Basic Hudi Operations	Java
SparkOnHudiPythonExample	Using Spark to Perform Basic Hudi Operations	Python
SparkOnHudiScalaExample	Using Spark to Perform Basic Hudi Operations	Scala

### 31.2.3 Creating a New Project (Optional)

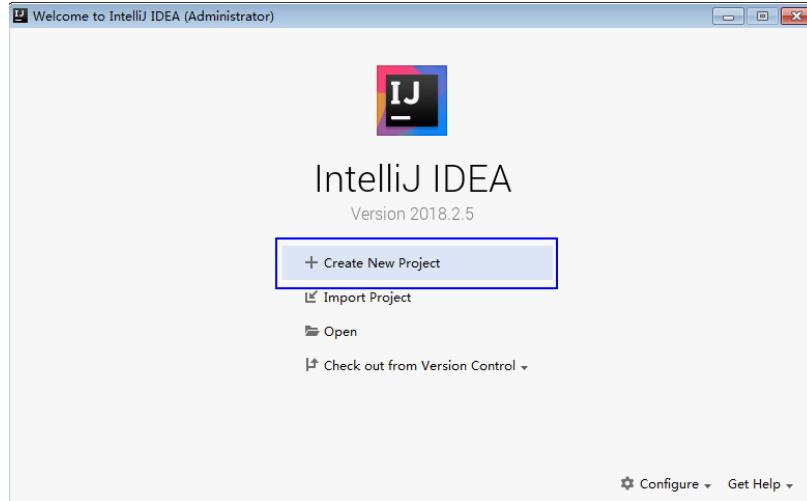
#### Scenario

Besides importing Spark sample projects, the IDEA can be used to create a Spark project. A Scala project is used as an example in the following steps.

## Procedure

**Step 1** Start the IDEA and select **Create New Project**.

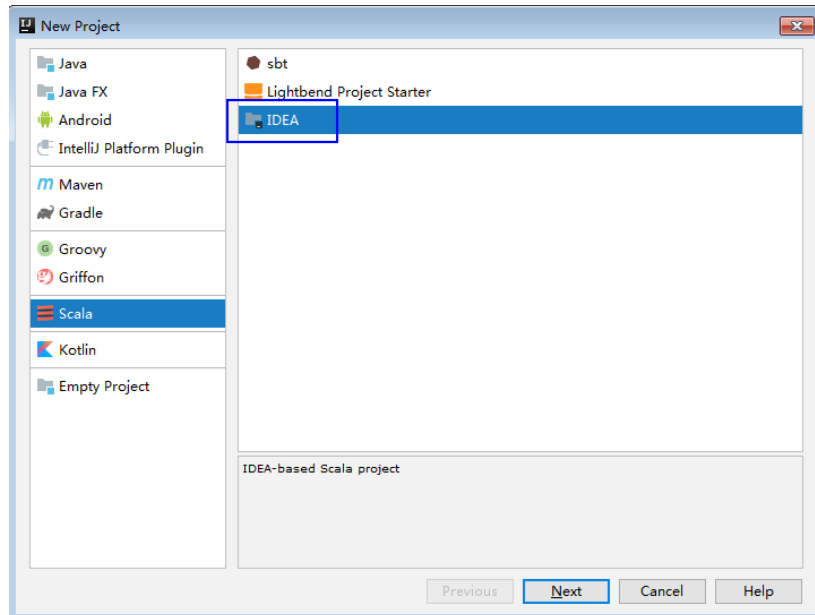
**Figure 31-32** Create a project



**Step 2** On the **New Project** page, select **Scala** as the development environment, select **IDEA**, and click **Next**.

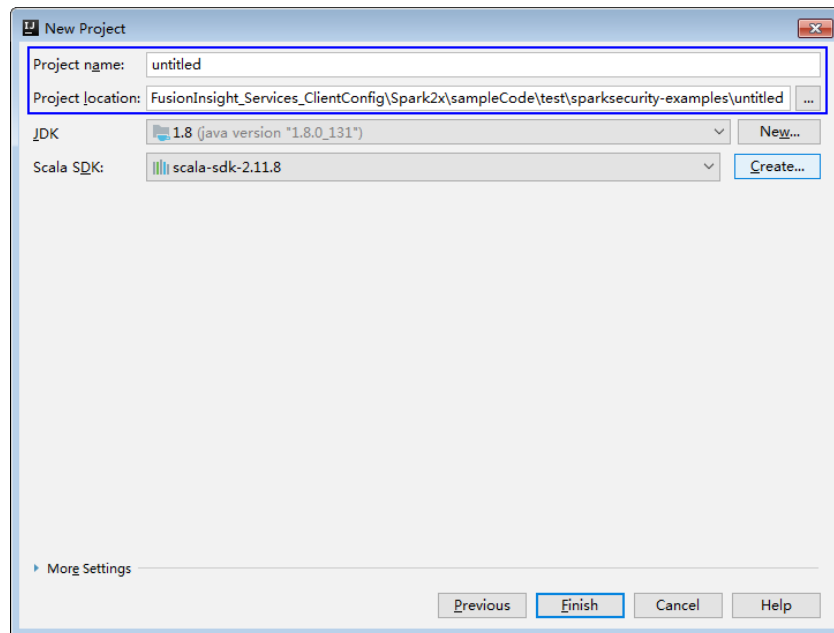
If a new project in Java is required, choose corresponding parameters.

**Figure 31-33** Select the development language



**Step 3** On the project information page, specify **Project name**, **Project location**, **Project JDK**, and **Scala SDK**, and click **Finish** to complete the project creation.

Figure 31-34 Add the project information



----End

## 31.2.4 Configuring the Python3 Sample Project

### Scenario

To run the Python3 interface sample code of the Spark2x component of MRS, perform the following operations.

### Procedure

**Step 1** Install Python3 of 3.6 or a higher version on the client.

The Python version can be viewed by running the **python3** command on the CLI of the client. The following information indicates that the Python version is 3.8.2.

```
Python 3.8.2 (default, Jun 23 2020, 10:26:03)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

**Step 2** Setuptools 47.3.1 must be installed on the client.

To obtain the software, visit the official websites.

<https://pypi.org/project/setuptools/#files>

Copy the downloaded setuptools package to the client, decompress the package, go to the decompressed directory, and run the **python3 setup.py install** command in the CLI of the client.

The following information indicates that setuptools 47.3.1 is installed successfully.

```
Finished processing dependencies for setuptools==47.3.1
```

**Step 3** Install Python on the client.

1. Obtain the sample project folder **python3-examples** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python3-examples** folder.
3. Go to the **dependency\_python3.6**, **dependency\_python3.7**, or **dependency\_python3.8** folder based on the Python3 version.
4. Run the **whereis easy\_install** command to find the path of the **easy\_install** program. If there are multiple paths, run the **easy\_install --version** command to select the **easy\_install** path corresponding to the **setuptools** version, for example, **/usr/local/bin/easy\_install**.
5. Run the **easy\_install** command to install the EGG files in the **dependency\_python3.x** folder in sequence. Example:  

```
/usr/local/bin/easy_install future-0.18.2-py3.8.egg
```

If the following information is displayed, the EGG file is successfully installed.

```
Finished processing dependencies for future==0.18.2
```

----End

## 31.3 Developing the Project

### 31.3.1 Spark Core Project

#### 31.3.1.1 Overview

##### Scenarios

Develop a Spark application to perform the following operations on logs about dwell durations of netizens for shopping online:

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.
- The first column in the log file records names, the second column records gender, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

log1.txt: logs collected on Saturday

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

log2.txt: logs collected on Sunday

```
LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
```

```
FangBo,female,50
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
CaiXuyu,female,50
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
FangBo,female,50
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

## Data Preparation

Save log files in the Hadoop distributed file system (HDFS).

1. Create text files **input\_data1.txt** and **input\_data2.txt** on a local computer, and copy the log contents of **log1.txt** to **input\_data1.txt** and **log2.txt** to **input\_data2.txt** respectively.
2. Create **/tmp/input** on HDFS client path, and run the following commands to upload **input\_data1.txt** and **input\_data2.txt** to **/tmp/input**:
  - a. On the Linux HDFS client, run the **hadoop fs -mkdir /tmp/input** command (a **hdfs dfs** command provides the same function.) to create a directory.
  - b. Go to the **/tmp/input** directory on the HDFS client, on the Linux HDFS client, run the **hadoop fs -put input\_data1.txt /tmp/input** and **hadoop fs -put input\_data2.txt /tmp/input** commands to upload data files.

## Development Idea

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

- Read the source file data.
- Filter the data information of the time that female netizens spend online.
- Aggregate the total time that each female netizen spends online.
- Filter the information of female netizens who spend more than 2 hours online.

## Packaging the Project

1. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
2. Upload the JAR file to any directory (for example, **/opt/female/**) on the server where the Spark client is located

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the **bin/spark-submit** script to run the code (The class name and file name must be the same as those in the actual code. The following is only an example):

- Run the Scala and Java sample programs.
  - **bin/spark-submit --class** *com.huawei.bigdata.spark.examples.FemaleInfoCollection --master yarn --deploy-mode client /opt/female/FemaleInfoCollection-1.0.jar <inputPath>*
  - <inputPath> indicates the input path in HDFS
- Run the Python sample program.
  - **bin/spark-submit --master yarn --deploy-mode client /opt/female/SparkPythonExample/collectFemaleInfo.py <inputPath>**
  - <inputPath> indicates the input path in HDFS.

### 31.3.1.2 Java Example Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

#### Example Code

The following code segment is only an example. For details, see the `com.huawei.bigdata.spark.examples.FemaleInfoCollection` class.

```
// Create a configuration class SparkConf, and then create a SparkContext.
SparkSession spark = SparkSession
 .builder()
 .appName("CollectFemaleInfo")
 .config("spark.some.config.option", "some-value")
 .getOrCreate();

// Read the source file data, and transfer each row of records to an element of the RDD.
JavaRDD<String> data = spark.read()
 .textFile(args[0])
 .javaRDD();

// Split each column of each record, and generate a Tuple.
JavaRDD<Tuple3<String,String,Integer>> person = data.map(new
Function<String,Tuple3<String,String,Integer>>()
{
 private static final long serialVersionUID = -2381522520231963249L;

 public Tuple3<String, String, Integer> call(String s) throws Exception
 {
 // Split a row of data by commas (.).
 String[] tokens = s.split(",");

 // Integrate the three split elements to a ternary Tuple.
 Tuple3<String, String, Integer> person = new Tuple3<String, String, Integer>(tokens[0], tokens[1],
Integer.parseInt(tokens[2]));
 return person;
 }
});

// Use the filter function to filter the data information about the time that female netizens spend online.
JavaRDD<Tuple3<String,String,Integer>> female = person.filter(new
Function<Tuple3<String,String,Integer>, Boolean>()
{
 private static final long serialVersionUID = -4210609503909770492L;

 public Boolean call(Tuple3<String, String, Integer> person) throws Exception
```

```
{
 // Filter the records of which the gender in the second column is female.
 Boolean isFemale = person._2().equals("female");
 return isFemale;
}
});

// Aggregate the total time that each female netizen spends online.
JavaPairRDD<String, Integer> females = female.mapToPair(new PairFunction<Tuple3<String, String,
Integer>, String, Integer>()
{
 private static final long serialVersionUID = 8313245377656164868L;

 public Tuple2<String, Integer> call(Tuple3<String, String, Integer> female) throws Exception
 {
 // Extract the two columns representing the name and online time for the sum of online time by
name during further operations.
 Tuple2<String, Integer> femaleAndTime = new Tuple2<String, Integer>(female._1(), female._3());
 return femaleAndTime;
 }
});
JavaPairRDD<String, Integer> femaleTime = females.reduceByKey(new Function2<Integer, Integer,
Integer>()
{
 private static final long serialVersionUID = -3271456048413349559L;

 public Integer call(Integer integer, Integer integer2) throws Exception
 {
 // Sum two online time durations of the same female netizen.
 return (integer + integer2);
 }
});

// Filter the information about female netizens who spend more than 2 hours online.
JavaPairRDD<String, Integer> rightFemales = females.filter(new Function<Tuple2<String, Integer>,
Boolean>()
{
 private static final long serialVersionUID = -3178168214712105171L;

 public Boolean call(Tuple2<String, Integer> s) throws Exception
 {
 // Extract the total time that female netizens spend online, and determine whether the time is
more than 2 hours.
 if(s._2() > (2 * 60))
 {
 return true;
 }
 return false;
 }
});

// Print the information about female netizens who meet the requirements.
for(Tuple2<String, Integer> d: rightFemales.collect())
{
 System.out.println(d._1() + "," + d._2());
}
```

### 31.3.1.3 Scala Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.



## Sample Code

The following code segment is only an example. For details, see the `com.huawei.bigdata.spark.examples.FemaleInfoCollection` class.

Example: `CollectMapper` class

```
val spark = SparkSession
 .builder()
 .appName("CollectFemaleInfo")
 .config("spark.some.config.option", "some-value")
 .getOrCreate()

// Read data. This code indicates the data path that the input parameter args(0) specifies.
val text = spark.sparkContext.textFile(args(0))
// Filter the data information about the time that female netizens spend online.
val data = text.filter(_ contains("female"))
// Aggregate the time that each female netizen spends online.
val femaleData:RDD[(String,Int)] = data.map{line =>
 val t= line.split(',')
 (t(0),t(2).toInt)
}.reduceByKey(_ + _)
// Filter the information about female netizens who spend more than 2 hours online, and export the results.
val result = femaleData.filter(line => line._2 > 120)
result.collect().map(x => x._1 + ' ' + x._2).foreach(println)
spark.stop()
```

### 31.3.1.4 Python Example Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

#### Example Code

The following code segment is only an example. For details, see `collectFemaleInfo.py`.

```
def contains(str, substr):
 if substr in str:
 return True
 return False

if __name__ == "__main__":
 if len(sys.argv) < 2:
 print "Usage: CollectFemaleInfo <file>"
 exit(-1)

 spark = SparkSession \
 .builder \
 .appName("CollectFemaleInfo") \
 .getOrCreate()

 """
 The following programs are used to implement the following functions:
 1. Read data. This code indicates the data path that the input parameter argv[1] specifies. - text
 2. Filter data about the time that female netizens spend online. - filter
 3. Aggregate the total time that each female netizen spends online. - map/map/reduceByKey
 4. Filter information about female netizens who spend more than 2 hours online. - filter
 """
 inputPath = sys.argv[1]
 result = spark.read.text(inputPath).rdd.map(lambda r: r[0])\
 .filter(lambda line: contains(line, "female")) \
 .map(lambda line: line.split(',')) \
```

```
.map(lambda dataArr: (dataArr[0], int(dataArr[2]))) \
.reduceByKey(lambda v1, v2: v1 + v2) \
.filter(lambda tupleVal: tupleVal[1] > 120) \
.collect() \
for (k, v) in result: \
 print k + "," + str(v) \
 \
Stop SparkContext. \
spark.stop()
```

## 31.3.2 Spark SQL Project

### 31.3.2.1 Overview

#### Scenarios

Develop a Spark application to perform the following operations on logs about dwell durations of netizens for shopping online:

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.
- The first column in the log file records names, the second column records gender, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

log1.txt: logs collected on Saturday

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

log2.txt: logs collected on Sunday

```
LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
FangBo,female,50
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
CaiXuyu,female,50
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
FangBo,female,50
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

#### Data Preparation

Save log files in the Hadoop distributed file system (HDFS).

1. Create text files **input\_data1.txt** and **input\_data2.txt** on a local computer, and copy the log contents of **log1.txt** to **input\_data1.txt** and **log2.txt** to **input\_data2.txt** respectively.

2. Create **/tmp/input** on HDFS client path, and run the following commands to upload **input\_data1.txt** and **input\_data2.txt** to **/tmp/input**:
  - a. On the Linux HDFS client, run the ***hadoop fs -mkdir /tmp/input*** command (a **hdfs dfs** command provides the same function.) to create a directory.
  - b. Go to the **/tmp/input** directory on the HDFS client, on the Linux HDFS client, run the ***hadoop fs -put input\_data1.txt /tmp/input*** and ***hadoop fs -put input\_data2.txt /tmp/input*** commands to upload data files.

## Development Idea

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

- Create a table and import the log files into the table.
- Filter the data information of the time that female netizens spend online.
- Aggregate the total time that each female netizen spends online.
- Filter the information of female netizens who spend more than 2 hours online.

## Packaging the Project

1. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
2. Upload the JAR file to any directory (for example, **/opt/female/**) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the **bin/spark-submit** script to run the code:

- Run the Scala and Java sample programs.
  - **bin/spark-submit --class *com.huawei.bigdata.spark.examples.FemaleInfoCollection --master yarn --deploy-mode client /opt/female/SparkSqlScalaExample-1.0.jar <inputPath>***
  - **<inputPath>** indicates the input path in HDFS.
- Run the Python sample program.
  - **bin/spark-submit --master yarn --deploy-mode client /opt/female/SparkSQLPythonExample/SparkSQLPythonExample.py <inputPath>**
  - **<inputPath>** indicates the input path in HDFS.

### 31.3.2.2 Java Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files (The class name and file name must be the same as those in the actual code. The following is only an example).

#### Sample Code

The following code segment is only an example. For details, see `com.huawei.bigdata.spark.examples.FemaleInfoCollection`.

```
public static void main(String[] args) throws Exception {
 SparkSession spark = SparkSession
 .builder()
 .appName("CollectFemaleInfo")
 .config("spark.some.config.option", "some-value")
 .getOrCreate();

 // Convert RDD to DataFrame through the implicit conversion.
 JavaRDD<FemaleInfo> femaleInfoJavaRDD = spark.read().textFile(args[0]).javaRDD().map(
 new Function<String, FemaleInfo>() {
 @Override
 public FemaleInfo call(String line) throws Exception {
 String[] parts = line.split(",");
 FemaleInfo femaleInfo = new FemaleInfo();
 femaleInfo.setName(parts[0]);
 femaleInfo.setGender(parts[1]);
 femaleInfo.setStayTime(Integer.parseInt(parts[2].trim()));
 return femaleInfo;
 }
 }
);

 // Register table.
 Dataset<ROW> schemaFemaleInfo = spark.createDataFrame(femaleInfoJavaRDD, FemaleInfo.class);
 schemaFemaleInfo.registerTempTable("FemaleInfoTable");

 // Run SQL query
 Dataset<ROW> femaleTimeInfo = spark.sql("select * from " +
 "(select name,sum(stayTime) as totalStayTime from FemaleInfoTable " +
 "where gender = 'female' group by name)" +
 " tmp where totalStayTime >120");

 // Collect the columns of a row in the result.
 List<String> result = femaleTimeInfo.javaRDD().map(new Function<Row, String>() {
 public String call(Row row) {
 return row.getString(0) + "," + row.getLong(1);
 }
 }).collect();
 System.out.println(result);
 spark.stop();
}
```

In the preceding code example, data processing logic is implemented by SQL statements. It can also be implemented by invoking the SparkSQL interface in Scala/Java/Python code. For details, see <http://spark.apache.org/docs/3.1.1/sql-programming-guide.html#running-sql-queries-programmatically>

### 31.3.2.3 Scala Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

#### Sample Code

The following code segment is only an example. For details, see `com.huawei.bigdata.spark.examples.FemaleInfoCollection`.

```
object FemaleInfoCollection
{
 //Table structure, used for mapping the text data to df
 case class FemaleInfo(name: String, gender: String, stayTime: Int)
 def main(args: Array[String]) {
 //Configure the Spark application name.
 val spark = SparkSession
 .builder()
 .appName("FemaleInfo")
 .config("spark.some.config.option", "some-value")
 .getOrCreate()
 import spark.implicits._
 //Convert RDD to DataFrame through the implicit conversion, then register a table.
 spark.sparkContext.textFile(args(0)).map(_._split(","))
 .map(p => FemaleInfo(p(0), p(1), p(2).trim.toInt))
 .toDF.registerTempTable("FemaleInfoTable")
 //Use SQL statements to filter the data information of the time that female netizens spend online, and
 aggregate data of the same name.
 val femaleTimeInfo = spark.sql("select name,sum(stayTime) as stayTime from FemaleInfoTable where
gender = 'female' group by name")
 //Filter the information of female netizens who spend more than 2 hours online and output the result.
 val c = femaleTimeInfo.filter("stayTime >= 120").collect().foreach(println)
 spark.stop()
 }
}
```

In the preceding code example, data processing logic is implemented by SQL statements. It can also be implemented by invoking the SparkSQL interface in Scala/Java/Python code. For details, see <http://spark.apache.org/docs/3.1.1/sql-programming-guide.html#running-sql-queries-programmatically>

### 31.3.2.4 Python Sample Code

#### Function

Collect information about female netizens who have spent more than 2 hours in online shopping on the weekend.

#### Sample Code

The following code segment is only an example. For details, see `SparkSQLPythonExample`.

```
-*- coding:utf-8 -*-

import sys
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext
```

```
def contains(str1, substr1):
 if substr1 in str1:
 return True
 return False

if __name__ == "__main__":
 if len(sys.argv) < 2:
 print "Usage: SparkSQLPythonExample.py <file>"
 exit(-1)

 # Initialize the SparkSession and SQLContext.
 sc = SparkSession.builder.appName("CollectFemaleInfo").getOrCreate()
 sqlCtx = SQLContext(sc)

 #Convert RDD to DataFrame.
 inputPath = sys.argv[1]
 inputRDD = sc.read.text(inputPath).rdd.map(lambda r: r[0])\
 .map(lambda line: line.split(",")\
 .map(lambda dataArr: (dataArr[0], dataArr[1], int(dataArr[2]))))\
 .collect()
 df = sqlCtx.createDataFrame(inputRDD)

 # Register a table.
 df.registerTempTable("FemaleInfoTable")

 # Run SQL query statements and display the result.
 FemaleTimeInfo = sqlCtx.sql("SELECT * FROM " +
 "(SELECT _1 AS Name,SUM(_3) AS totalStayTime FROM FemaleInfoTable " +
 "WHERE _2 = 'female' GROUP BY _1)" +
 " WHERE totalStayTime >120").show()

 sc.stop()
```

## 31.3.3 Accessing the Spark SQL Through JDBC

### 31.3.3.1 Overview

#### Scenarios

Users customize JDBCServer clients and use JDBC connections to create, load data to, query, and delete data tables.

#### Data Preparation

- Step 1** Ensure that the JDBCServer service has been started in multi-active instance HA mode and at least one instance provides connections for client. Create the **/home/data** file on every available instance nodes of the JDBCServer. The file content is as follows:

```
Miranda,32
Karlie,23
Candice,27
```

- Step 2** Ensure that the user whose starts the JDBCServer has the read and write permission on the file.
- Step 3** Ensure that the **hive-site.xml** file exists in **classpath**, and set parameters required for the client connection. For details about parameters required for the JDBCServer, see [JDBCServer Interface](#).

----End

## Development Idea

1. Create a child table in the default database.
2. Add data in **/home/data** to the child table.
3. Query data in the child table.
4. Delete the child table.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR file to any directory (for example, **/opt/female/**) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the **java -cp** command to run the code (The class name and file name must be the same as those in the actual code. The following is only an example).

- Run the Java sample code:

```
java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/hive/*:$SPARK_HOME/
conf:/opt/female/SparkThriftServerJavaExample-1.0.jar
com.huawei.bigdata.spark.examples.ThriftServerQueriesTest $SPARK_HOME/
conf/hive-site.xml $SPARK_HOME/conf/spark-defaults.conf
```

- Run the Scala sample code:

```
java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/hive/*:$SPARK_HOME/
conf:/opt/female/SparkThriftServerExample-1.0.jar
com.huawei.bigdata.spark.examples.ThriftServerQueriesTest $SPARK_HOME/
conf/hive-site.xml $SPARK_HOME/conf/spark-defaults.conf
```

### NOTE

After the SSL feature of ZooKeeper is enabled for the cluster (check the **ssl.enabled** parameter of the ZooKeeper service), add the **-Dzookeeper.client.secure=true -Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty** parameter to the command:

```
java -Dzookeeper.client.secure=true -
Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -cp
$SPARK_HOME/jars/*:$SPARK_HOME/jars/hive/*:$SPARK_HOME/conf:/opt/female/
SparkThriftServerJavaExample-1.0.jar
com.huawei.bigdata.spark.examples.ThriftServerQueriesTest $SPARK_HOME/conf/hive-
site.xml $SPARK_HOME/conf/spark-defaults.conf
```

### 31.3.3.2 Java Example Code

#### Function

The JDBC interface of the user-defined client is used to submit the data analysis task and return the results.

## Example Code

**Step 1** Define an SQL statement. The SQL statement must be a single statement that cannot contain the semicolon (;). For example,

```
ArrayList<String> sqlList = new ArrayList<String>();
sqlList.add("CREATE TABLE CHILD (NAME STRING, AGE INT) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ','");
sqlList.add("LOAD DATA LOCAL INPATH '/home/data' INTO TABLE CHILD");
sqlList.add("SELECT * FROM child");
sqlList.add("DROP TABLE child");
executeSql(url, sqlList);
```

### NOTE

The data file in the sample project must be placed into the Home directory of the host where the JDBCServer is located.

**Step 2** Assemble the JDBC URL.

```
Configuration config = new Configuration();
config.addResource(new Path(args[0]));
String zkUrl = config.get("spark.deploy.zookeeper.url");

String zkNamespace = null;
zkNamespace = fileInfo.getProperty("spark.thriftserver.zookeeper.namespace");
if (zkNamespace != null) {
 //Remove redundant characters from configuration items
 zkNamespace = zkNamespace.substring(1);
}

StringBuilder sb = new StringBuilder("jdbc:hive2://"
+ zkUrl
+ "/;serviceDiscoveryMode=zooKeeper;"
+ "zooKeeperNamespace="
+ zkNamespace + ";");
String url = sb.toString();
```

**Step 3** Load the Hive JDBC driver.

```
Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance();
```

**Step 4** Obtain the JDBC connection, execute the HQL, export the obtained column name and results to the console, and close the JDBC connection.

The **zk.quorum** in the connection string can be replaced by **spark.deploy.zookeeper.url** in the configuration file.

In network congestion, configure the timeout of the connection between the client and JDBCServer to avoid the suspending of the client due to timeless wait of the return from the server.

Before executing the `DriverManager.getConnection` script to obtain the JDBC connection, add the `DriverManager.setLoginTimeout(n)` script to configure the timeout. `n` indicates the timeout length of waiting for the return from the server. The unit is second, the type is `Int`, and the default value is 0 (indicating never timing out).

```
static void executeSql(String url, ArrayList<String> sqls) throws ClassNotFoundException, SQLException {
 try {
 Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance();
 } catch (Exception e) {
 e.printStackTrace();
 }
 Connection connection = null;
 PreparedStatement statement = null;
```



```
try {
 connection = DriverManager.getConnection(url);
 for (int i = 0 ; i < sqls.size(); i++) {
 String sql = sqls.get(i);
 System.out.println("---- Begin executing sql: " + sql + " ----");
 statement = connection.prepareStatement(sql);
 ResultSet result = statement.executeQuery();
 ResultSetMetaData resultMetaData = result.getMetaData();
 Integer colNum = resultMetaData.getColumnCount();
 for (int j = 1; j <= colNum; j++) {
 System.out.println(resultMetaData.getColumnLabel(j) + "\t");
 }
 System.out.println();

 while (result.next()) {
 for (int j = 1; j <= colNum; j++){
 System.out.println(result.getString(j) + "\t");
 }
 System.out.println();
 }
 System.out.println("---- Done executing sql: " + sql + " ----");
 }

} catch (Exception e) {
 e.printStackTrace();
} finally {
 if (null != statement) {
 statement.close();
 }
 if (null != connection) {
 connection.close();
 }
}
}
```

----End

### 31.3.3.3 Scala Example Code

#### Function

The JDBC interface of the user-defined client is used to submit the data analysis task and return the results.

#### Example Code

- Step 1** Define an SQL statement. The SQL statement must be a single statement that cannot contain the semicolon (;). For example,

```
val sqlList = new ArrayBuffer[String]
sqlList += "CREATE TABLE CHILD (NAME STRING, AGE INT) " +
"ROW FORMAT DELIMITED FIELDS TERMINATED BY ','"
sqlList += "LOAD DATA LOCAL INPATH '/home/data' INTO TABLE CHILD"
sqlList += "SELECT * FROM child"
sqlList += "DROP TABLE child"
```

#### NOTE

The data file in the sample project must be placed into the Home directory of the host where the JDBCServer is located.

- Step 2** Assemble the JDBC URL.

```
val config: Configuration = new Configuration()
config.addResource(new Path(args(0)))
```

```
val zkUrl = config.get("spark.deploy.zookeeper.url")

var zkNamespace: String = null
zkNamespace = fileInfo.getProperty("spark.thriftserver.zookeeper.namespace")
//Remove redundant characters from configuration items
if (zkNamespace != null) zkNamespace = zkNamespace.substring(1)

val sb = new StringBuilder("jdbc:hive2://"
+ zkUrl
+ "/;serviceDiscoveryMode=zooKeeper;"
+ "zooKeeperNamespace="
+ zkNamespace + ";");
val url = sb.toString()
```

**Step 3** Load the Hive JDBC driver. Obtain the JDBC connection, execute the HQL, export the obtained column name and results to the console, and close the JDBC connection.

The **zk.quorum** in the connection string can be replaced by **spark.deploy.zookeeper.url** in the configuration file.

In network congestion, configure the timeout of the connection between the client and JDBCServer to avoid the suspending of the client due to timeless wait of the return from the server.

Before executing the `DriverManager.getConnection` script to obtain the JDBC connection, add the `DriverManager.setLoginTimeout(n)` script to configure the timeout. `n` indicates the timeout length of waiting for the return from the server. The unit is second, the type is `Int`, and the default value is 0 (indicating never timing out).

```
def executeSql(url: String, sqls: Array[String]): Unit = {
//Load the Hive JDBC driver.
Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance()

var connection: Connection = null
var statement: PreparedStatement = null
try {
connection = DriverManager.getConnection(url)
for (sql <- sqls) {
println(s"---- Begin executing sql: $sql ----")
statement = connection.prepareStatement(sql)

val result = statement.executeQuery()

val resultMetaData = result.getMetaData
val colNum = resultMetaData.getColumnCount
for (i <- 1 to colNum) {
print(resultMetaData.getColumnLabel(i) + "\t")
}
println()

while (result.next()) {
for (i <- 1 to colNum) {
print(result.getString(i) + "\t")
}
println()
}
println(s"---- Done executing sql: $sql ----")
}
} finally {
if (null != statement) {
statement.close()
}

if (null != connection) {
connection.close()
}
```

```
}
}
}
```

----End

## 31.3.4 Spark on HBase

### 31.3.4.1 Performing Operation on Data in Avro Format

#### Scenario

Users can use HBase as data sources in Spark applications. In this example, data is stored in HBase in Avro format. Data is read from the HBase, and the read data is filtered.

#### Data Planning

On the client, run the **hbase shell** command to go to the HBase command line and run the following commands to create HBase tables to be used in the sample code:

```
create 'ExampleAvrotable','rowkey','cf1'
```

```
create 'ExampleAvrotableInsert','rowkey','cf1'
```

#### Development Guideline

1. Create an RDD.
2. Perform operations on HBase to treat it as the data source and write the generated RDD into HBase tables.
3. Read data from HBase tables and performs simple operations on the data.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, `$SPARK_HOME`) on the server where the Spark client is located.

#### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

#### Submitting Commands

Assume that the JAR package name of the case code is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. Run the following commands in the **\$SPARK\_HOME** directory.

- yarn-client mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode client --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.AvroSource SparkOnHbaseJavaExample-1.0.jar**  
Python version. (The file name must be the same as the actual one. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode client --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample-1.0.jar,/opt/female/protobuf-java-2.5.0.jar AvroSource.py**
- yarn-cluster mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode cluster --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.AvroSource SparkOnHbaseJavaExample-1.0.jar**  
Python version. (The file name must be the same as the actual one. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode cluster --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample-1.0.jar,/opt/female/protobuf-java-2.5.0.jar AvroSource.py**

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the AvroSource file in SparkOnHbaseJavaExample.

```
public static void main(JavaSparkContext jsc) throws IOException {
 SQLContext sqlContext = new SQLContext(jsc);
 Configuration hbaseconf = new HBaseConfiguration().create();
 JavaHBaseContext hBaseContext = new JavaHBaseContext(jsc, hbaseconf);
 List list = new ArrayList<AvroHBaseRecord>();
 for(int i=0; i<=255 ; ++i){
 list.add(AvroHBaseRecord.apply(i));
 }
 try{
 Map<String, String> map = new HashMap<String, String>();
 map.put(HBaseTableCatalog.tableCatalog(), catalog);
 map.put(HBaseTableCatalog.newTable(), "5");
 sqlContext.createDataFrame(list,
AvroHBaseRecord.class).write().options(map).format("org.apache.hadoop.hbase.spark").save();
 Dataset<Row> ds = withCatalog(sqlContext,catalog);
 ds.show();
 ds.printSchema();
 ds.registerTempTable("ExampleAvrotable");
 Dataset<Row> c= sqlContext.sql("select count(1) from ExampleAvrotable");
 c.show();
 Dataset<Row> filtered = ds.select("col0", "col1.favorite_array").where("col0 = 'name1'");
 filtered.show();
 java.util.List<Row> collected = filtered.collectAsList();
 if (collected.get(0).get(1).toString().equals("number1")) {
 throw new UserCustomizedSampleException("value invalid", new Throwable());
 }
 }
}
```

```
 }
 if (collected.get(0).get(1).toString().equals("number2")) {
 throw new UserCustomizedSampleException("value invalid", new Throwable());
 }
 Map avroCatalogInsertMap = new HashMap<String,String>();
 avroCatalogInsertMap.put("avroSchema" , AvroHBaseRecord.schemaString);
 avroCatalogInsertMap.put(HBaseTableCatalog.tableCatalog(), avroCatalogInsert);
 ds.write().options(avroCatalogInsertMap).format("org.apache.hadoop.hbase.spark").save();
 Dataset<Row> newDS = withCatalog(sqlContext,avroCatalogInsert);
 newDS.show();
 newDS.printSchema();
 if (newDS.count() != 256) {
 throw new UserCustomizedSampleException("value invalid", new Throwable());
 }
 ds.filter("col1.name = 'name5' || col1.name <= 'name5'").select("col0","col1.favorite_color",
"col1.favorite_number").show();
 } finally{
 jsc.stop();
 }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the AvroSource file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
 val sparkConf = new SparkConf().setAppName("AvroSourceExample")
 val sc = new SparkContext(sparkConf)
 val sqlContext = new SQLContext(sc)
 val hbaseConf = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc, hbaseConf)
 import sqlContext.implicitly._
 def withCatalog(cat: String): DataFrame = {
 sqlContext
 .read
 .options(Map("avroSchema" -> AvroHBaseRecord.schemaString, HBaseTableCatalog.tableCatalog ->
avroCatalog))
 .format("org.apache.hadoop.hbase.spark")
 .load()
 }
 val data = (0 to 255).map { i =>
 AvroHBaseRecord(i)
 }
 try {
 sc.parallelize(data).toDF.write.options(
 Map(HBaseTableCatalog.tableCatalog -> catalog, HBaseTableCatalog.newTable -> "5"))
 .format("org.apache.hadoop.hbase.spark")
 .save()

 val df = withCatalog(catalog)
 df.show()
 df.printSchema()
 df.registerTempTable("ExampleAvrotable")
 val c = sqlContext.sql("select count(1) from ExampleAvrotable")
 c.show()

 val filtered = df.select($"col0", $"col1.favorite_array").where($"col0" === "name001")
 filtered.show()
 val collected = filtered.collect()
 if (collected(0).getSeq[String](1)(0) != "number1") {
 throw new UserCustomizedSampleException("value invalid")
 }
 if (collected(0).getSeq[String](1)(1) != "number2") {
 throw new UserCustomizedSampleException("value invalid")
 }
 }

 df.write.options(
 Map("avroSchema" -> AvroHBaseRecord.schemaString, HBaseTableCatalog.tableCatalog ->
```

```
avroCatalogInsert,
 HBaseTableCatalog.newTable -> "5")
 .format("org.apache.hadoop.hbase.spark")
 .save()
val newDF = withCatalog(avroCatalogInsert)
newDF.show()
newDF.printSchema()
if (newDF.count() != 256) {
 throw new UserCustomizedSampleException("value invalid")
}
df.filter($"col1.name" === "name005" || $"col1.name" <= "name005")
 .select("col0", "col1.favorite_color", "col1.favorite_number")
 .show()
} finally {
 sc.stop()
}
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the AvroSource file in SparkOnHbasePythonExample.

```
-*- coding:utf-8 -*-
"""
[Note]
Pyspark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("AvroSourceExample")\
 .getOrCreate()
Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.datasources.AvroSource')
Create a class instance, invoke the method, and transfer the sc._jsc parameter.
spark._jvm.AvroSource().execute(spark._jsc)
Stop SparkSession.
spark.stop()
```

### 31.3.4.2 Performing Operations on the HBase Data Source

#### Scenario

Users can use HBase as data sources in Spark applications, write DataFrame to HBase, read data from HBase, and filter the read data.

#### Data Planning

On the client, run the **hbase shell** command to go to the HBase command line and run the following commands to create HBase tables to be used in the sample code:

```
create 'HBaseSourceExampleTable','rowkey','cf1','cf2','cf3','cf4','cf5','cf6','cf7','cf8'
```

#### Development Guideline

1. Create an RDD.

2. Perform operations on HBase to treat it as the data source and write the generated RDD into HBase tables.
3. Read data from HBase tables and performs simple operations on the data.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, `$SPARK_HOME`) on the server where the Spark client is located.

### NOTE

To run the Spark on HBase example program, set `spark.yarn.security.credentials.hbase.enabled` (`false` by default) in the `spark-defaults.conf` file on the Spark client to `true`. Changing the `spark.yarn.security.credentials.hbase.enabled` value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to `false`.) Set the value of the configuration item `spark.inputFormat.cache.enabled` to `false`.

## Submitting Commands

Assume that the JAR package name of the case code is `spark-hbaseContext-test-1.0.jar` that is stored in the `$SPARK_HOME` directory on the client. Run the following commands in the `$SPARK_HOME` directory.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.HBaseSource SparkOnHbaseJavaExample-1.0.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample-1.0.jar,/opt/female/protobuf-java-2.5.0.jar HBaseSource.py
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.HBaseSource SparkOnHbaseJavaExample-1.0.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample-1.0.jar,/opt/female/protobuf-java-2.5.0.jar HBaseSource.py
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseSource file in SparkOnHbaseJavaExample.

```
public static void main(String args[]) throws IOException{
 SparkConf sparkConf = new SparkConf().setAppName("HBaseSourceExample");
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 SQLContext sqlContext = new SQLContext(jsc);

 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc,conf);
 try{
 List<HBaseRecord> list = new ArrayList<HBaseRecord>();
 for(int i=0 ; i<256; i++){
 list.add(new HBaseRecord(i));
 }
 Map map = new HashMap<String, String>();
 map.put(HBaseTableCatalog.tableCatalog(), "cat");
 map.put(HBaseTableCatalog.newTable(), "5");
 System.out.println("Before insert data into hbase table");
 sqlContext.createDataFrame(list,
HBaseRecord.class).write().options(map).format("org.apache.hadoop.hbase.spark").save();
 Dataset<Row> ds = withCatalog(sqlContext, "cat");
 System.out.println("After insert data into hbase table");
 ds.printSchema();
 ds.show();
 ds.filter("key <= 'row5'").select("key","col1").show();
 ds.registerTempTable("table1");
 Dataset<Row> tempDS = sqlContext.sql("select count(col1) from table1 where key < 'row5'");
 tempDS.show();
 } finally {
 jsc.stop();
 }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseSource file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
 val sparkConf = new SparkConf().setAppName("HBaseSourceExample")
 val sc = new SparkContext(sparkConf)
 val sqlContext = new SQLContext(sc)
 val conf = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc,conf)
 import sqlContext.implicits._
 def withCatalog(cat: String): DataFrame = {
 sqlContext
 .read
 .options(Map(HBaseTableCatalog.tableCatalog->cat))
 .format("org.apache.hadoop.hbase.spark")
 .load()
 }
 val data = (0 to 255).map { i =>
 HBaseRecord(i)
 }
 try{
 sc.parallelize(data).toDF.write.options(
 Map(HBaseTableCatalog.tableCatalog -> "cat", HBaseTableCatalog.newTable -> "5"))
 .format("org.apache.hadoop.hbase.spark")
 .save()
 val df = withCatalog("cat")
 df.show()
 df.filter($"col0" <= "row005")
 .select($"col0", $"col1").show
 }
```



```
df.registerTempTable("table1")
val c = sqlContext.sql("select count(col1) from table1 where col0 < 'row050'")
c.show()
} finally {
 sc.stop()
}
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseSource file in SparkOnHbasePythonExample.

```
-*- coding:utf-8 -*-
"""
[Note]
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement the required operations.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("HBaseSourceExample")\
 .getOrCreate()
Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.datasources.HBaseSource')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.HBaseSource().execute(spark._jsc)
Stop the SparkSession instance.
spark.stop()
```

### 31.3.4.3 Using the BulkPut Interface

#### Scenario

Users can use the HBaseContext method to use HBase in Spark applications and write the constructed RDD into HBase.

#### Data Planning

On the client, run the **hbase shell** command to go to the HBase command line and run the following commands to create HBase tables to be used in the sample code:

```
create 'bulktable','cf1'
```

#### Development Guideline

1. Create an RDD.
2. Perform operations on HBase in HBaseContext mode and write the generated RDD into the HBase table.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located

 NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkPutExa
mple SparkOnHbaseJavaExample-1.0.jar bulktable cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars
SparkOnHbaseJavaExample-1.0.jar HBaseBulkPutExample.py bulktable
cf1
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkPutExa
mple SparkOnHbaseJavaExample-1.0.jar bulktable cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars
SparkOnHbaseJavaExample-1.0.jar HBaseBulkPutExample.py bulktable
cf1
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `JavaHBaseBulkPutExample` file in `SparkOnHbaseJavaExample`.

```
public static void main(String[] args) throws Exception{
 if (args.length < 2) {
 System.out.println("JavaHBaseBulkPutExample " +
 "{tableName} {columnFamily}");
 return;
 }
 String tableName = args[0];
 String columnFamily = args[1];
 SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkPutExample " + tableName);
```

```
JavaSparkContext jsc = new JavaSparkContext(sparkConf);
try {
 List<String> list = new ArrayList<String>(5);
 list.add("1," + columnFamily + ",1,1");
 list.add("2," + columnFamily + ",1,2");
 list.add("3," + columnFamily + ",1,3");
 list.add("4," + columnFamily + ",1,4");
 list.add("5," + columnFamily + ",1,5");
 list.add("6," + columnFamily + ",1,6");
 list.add("7," + columnFamily + ",1,7");
 list.add("8," + columnFamily + ",1,8");
 list.add("9," + columnFamily + ",1,9");
 list.add("10," + columnFamily + ",1,10");
 JavaRDD<String> rdd = jsc.parallelize(list);
 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
 hbaseContext.bulkPut(rdd,
 TableName.valueOf(tableName),
 new PutFunction());
 System.out.println("Bulk put into Hbase successfully!");
} finally {
 jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkPutExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
 if (args.length < 2) {
 System.out.println("HBaseBulkPutTimestampExample {tableName} {columnFamily} are missing an argument")
 return
 }
 val tableName = args(0)
 val columnFamily = args(1)
 val sparkConf = new SparkConf().setAppName("HBaseBulkPutTimestampExample " +
 tableName + " " + columnFamily)
 val sc = new SparkContext(sparkConf)
 try {
 val rdd = sc.parallelize(Array(
 (Bytes.toBytes("1"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("1")))),
 (Bytes.toBytes("2"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("2")))),
 (Bytes.toBytes("3"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("3")))),
 (Bytes.toBytes("4"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("4")))),
 (Bytes.toBytes("5"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("5")))),
 (Bytes.toBytes("6"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("6")))),
 (Bytes.toBytes("7"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("7")))),
 (Bytes.toBytes("8"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("8")))),
 (Bytes.toBytes("9"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("9")))),
 (Bytes.toBytes("10"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("10")))))
)
 val conf = HBaseConfiguration.create()
 val timeStamp = System.currentTimeMillis()
 val hbaseContext = new HBaseContext(sc, conf)
 hbaseContext.bulkPut[(Array[Byte], Array[(Array[Byte], Array[Byte], Array[Byte])])](rdd,
 TableName.valueOf(tableName),
 (putRecord) => {
```

```
val put = new Put(putRecord._1)
putRecord._2.foreach((putValue) => put.addColumn(putValue._1, putValue._2,
 timeStamp, putValue._3))
 put
 })
} finally {
 sc.stop()
}
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkPutExample` file in `SparkOnHbasePythonExample`.

```
-*- coding:utf-8 -*-
"""
[Note]
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseBulkPutExample")\
 .getOrCreate()
Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkPutExample')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseBulkPutExample().execute(spark._jsc, sys.argv)
Stop the SparkSession instance.
spark.stop()
```

### 31.3.4.4 Using the BulkGet Interface

#### Scenario

Users can use the `HBaseContext` method to use HBase in Spark applications, construct the rowkey of the data to be obtained into RDDs, and obtain the data corresponding to the rowkey in the HBase tables through the BulkGet interface of `HBaseContext`.

#### Data Planning

Perform operations based on the HBase tables and data in the tables that are created in [Using the BulkPut Interface](#)

#### Development Guideline

1. Create RDDs containing the rowkey to be obtained.
2. Perform operations on HBase in `HBaseContext` mode and obtain data corresponding to rowkey in HBase tables through the BulkGet interface of `HBaseContext`.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

- Upload the JAR package to any directory (for example, `$SPARK_HOME`) on the server where the Spark client is located.

#### NOTE

To run the Spark on HBase example program, set **`spark.yarn.security.credentials.hbase.enabled`** (**false** by default) in the **`spark-defaults.conf`** file on the Spark client to **true**. Changing the **`spark.yarn.security.credentials.hbase.enabled`** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **`spark.inputFormat.cache.enabled`** to **false**.

## Submitting Commands

Assume that the JAR package name is **`spark-hbaseContext-test-1.0.jar`** that is stored in the `$SPARK_HOME` directory on the client. The following commands are executed in the `$SPARK_HOME` directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)  
**`bin/spark-submit --master yarn --deploy-mode client --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkGetExample SparkOnHbaseJavaExample-1.0.jar bulktable`**  
Python version. (The file name must be the same as the actual one. The following is only an example.)  
**`bin/spark-submit --master yarn --deploy-mode client --jars SparkOnHbaseJavaExample-1.0.jar HBaseBulkGetExample.py bulktable`**
- yarn-cluster mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)  
**`bin/spark-submit --master yarn --deploy-mode cluster --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkGetExample SparkOnHbaseJavaExample-1.0.jar bulktable`**  
Python version. (The file name must be the same as the actual one. The following is only an example.)  
**`bin/spark-submit --master yarn --deploy-mode cluster --jars SparkOnHbaseJavaExample-1.0.jar HBaseBulkGetExample.py bulktable`**

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkGetExample` file in `SparkOnHbaseJavaExample`.

```
public static void main(String[] args) throws IOException{
 if (args.length < 1) {
 System.out.println("JavaHBaseBulkGetExample {tableName}");
 return;
 }
 String tableName = args[0];
 SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkGetExample " + tableName);
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 try {
```

```
List<byte[]> list = new ArrayList<byte[]>(5);
list.add(Bytes.toBytes("1"));
list.add(Bytes.toBytes("2"));
list.add(Bytes.toBytes("3"));
list.add(Bytes.toBytes("4"));
list.add(Bytes.toBytes("5"));
JavaRDD<byte[]> rdd = jsc.parallelize(list);
Configuration conf = HBaseConfiguration.create();
JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
List resultList = hbaseContext.bulkGet(tableName.valueOf(tableName), 2, rdd, new GetFunction(),
 new ResultFunction()).collect();
for(int i = 0 ;i<resultList.size();i++){
 System.out.println(resultList.get(i));
}
} finally {
 jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkGetExample` file in `SparkOnHbaseScalaExample`.

```
def main(args: Array[String]) {
 if (args.length < 1) {
 println("HBaseBulkGetExample {tableName} missing an argument")
 return
 }
 val tableName = args(0)
 val sparkConf = new SparkConf().setAppName("HBaseBulkGetExample " + tableName)
 val sc = new SparkContext(sparkConf)
 try {
 //[(Array[Byte])]
 val rdd = sc.parallelize(Array(
 Bytes.toBytes("1"),
 Bytes.toBytes("2"),
 Bytes.toBytes("3"),
 Bytes.toBytes("4"),
 Bytes.toBytes("5"),
 Bytes.toBytes("6"),
 Bytes.toBytes("7")))
 val conf = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc, conf)
 val getRdd = hbaseContext.bulkGet[Array[Byte], String](
 tableName.valueOf(tableName),
 2,
 rdd,
 record => {
 System.out.println("making Get")
 new Get(record)
 },
 (result: Result) => {
 val it = result.listCells().iterator()
 val b = new StringBuilder
 b.append(Bytes.toString(result.getRow) + ":",)
 while (it.hasNext) {
 val cell = it.next()
 val q = Bytes.toString(CellUtil.cloneQualifier(cell))
 if (q.equals("counter")) {
 b.append("(" + q + "," + Bytes.toLong(CellUtil.cloneValue(cell)) + ")")
 } else {
 b.append("(" + q + "," + Bytes.toString(CellUtil.cloneValue(cell)) + ")")
 }
 }
 b.toString()
 })
 getRdd.collect().foreach(v => println(v))
 } finally {
```

```
sc.stop()
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkGetExample` file in `SparkOnHbasePythonExample`.

```
-*- coding:utf-8 -*-
"""
[Note]
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseBulkGetExample")\
 .getOrCreate()
Import required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkGetExample')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseBulkGetExample().execute(spark._jsc, sys.argv)
Stop the SparkSession instance.
spark.stop()
```

### 31.3.4.5 Using the BulkDelete Interface

#### Scenario

Users can use the `HBaseContext` method to use HBase in Spark applications, construct rowkey of the data to be deleted into RDDs, and delete the data corresponding to the rowkey in HBase tables through the BulkDelete interface of `HBaseContext`.

#### Data Planning

Perform operations based on the HBase tables and data in the tables that are created in [Using the BulkPut Interface](#).

#### Development Guideline

1. Create RDDs containing the rowkey to be deleted.
2. Perform operations on the HBase in `HBaseContext` mode and delete the data corresponding to the rowkey in HBase tables through the BulkDelete interface of `HBaseContext`.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, `$SPARK_HOME`) on the server where the Spark client is located.

 NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkDeleteE
xample SparkOnHbaseJavaExample-1.0.jar bulktable
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars
SparkOnHbaseJavaExample-1.0.jar HBaseBulkDeleteExample.py bulktable
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkDeleteE
xample SparkOnHbaseJavaExample-1.0.jar bulktable
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars
SparkOnHbaseJavaExample-1.0.jar HBaseBulkDeleteExample.py bulktable
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the **HBaseBulkDeleteExample** file in **SparkOnHbaseJavaExample**.

```
public static void main(String[] args) throws IOException {
 if (args.length < 1) {
 System.out.println("JavaHBaseBulkDeleteExample {tableName}");
 return;
 }
 String tableName = args[0];
 SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkDeleteExample " + tableName);
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 try {
 List<byte[]> list = new ArrayList<byte[]>(5);
 list.add(Bytes.toBytes("1"));
 list.add(Bytes.toBytes("2"));
 }
}
```



```
list.add(Bytes.toBytes("3"));
list.add(Bytes.toBytes("4"));
list.add(Bytes.toBytes("5"));
JavaRDD<byte[]> rdd = jsc.parallelize(list);
Configuration conf = HBaseConfiguration.create();
JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
hbaseContext.bulkDelete(rdd,
 TableName.valueOf(tableName), new DeleteFunction(), 4);
System.out.println("Bulk Delete successfully!");
} finally {
 jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkDeleteExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
 if (args.length < 1) {
 println("HBaseBulkDeleteExample {tableName} missing an argument")
 return
 }
 val tableName = args(0)
 val sparkConf = new SparkConf().setAppName("HBaseBulkDeleteExample " + tableName)
 val sc = new SparkContext(sparkConf)
 try {
 //[Array[Byte]]
 val rdd = sc.parallelize(Array(
 Bytes.toBytes("1"),
 Bytes.toBytes("2"),
 Bytes.toBytes("3"),
 Bytes.toBytes("4"),
 Bytes.toBytes("5")
))
 val conf = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc, conf)
 hbaseContext.bulkDelete[Array[Byte]](rdd,
 TableName.valueOf(tableName),
 putRecord => new Delete(putRecord),
 4)
 } finally {
 sc.stop()
 }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkDeleteExample file in SparkOnHbasePythonExample.

```
def main(args: Array[String]) {
-*- coding:utf-8 -*-
"""
[Note]
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseBulkDeleteExample")\
 .getOrCreate()
Import the required class to sc._jvm.
```

```
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkDeleteExample')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseBulkDeleteExample().execute(spark._jsc, sys.argv)
Stop the SparkSession instance.
spark.stop()
```

### 31.3.4.6 Using the BulkLoad Interface

#### Scenario

Users can use HBaseContext to perform operations on HBase in Spark applications, construct rowkey of the data to be inserted into RDDs, and write RDDs to HFiles through the BulkLoad interface of HBaseContext. The following command is used to import the generated HFiles to the HBase table and will not be described in this section.

```
hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles
{hfilePath} {tableName}
```

#### Data Planning

1. Run the **hbase shell** command on the client to go to the HBase command line.
2. Run the following command to create HBase tables:  
**create 'bulkload-table-test','f1','f2'**

#### Development Guideline

1. Construct the data to be imported into RDDs.
2. Perform operations on HBase in HBaseContext mode and write RDDs into HFiles through the BulkLoad interface of HBaseContext.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.

#### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

#### Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode client --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkLoadExample SparkOnHbaseJavaExample-1.0.jar /tmp/hfile bulkload-table-test**  
Python version. (The file name must be the same as the actual one. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode client --jars SparkOnHbaseJavaExample-1.0.jar HBaseBulkLoadExample.py /tmp/hfile bulkload-table-test**
- yarn-cluster mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode cluster --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkLoadExample SparkOnHbaseJavaExample-1.0.jar /tmp/hfile bulkload-table-test**  
Python version. (The file name must be the same as the actual one. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode cluster --jars SparkOnHbaseJavaExample-1.0.jar HBaseBulkLoadExample.py /tmp/hfile bulkload-table-test**

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `JavaHBaseBulkLoadExample` file in `SparkOnHbaseJavaExample`.

```
public static void main(String[] args) throws IOException{
 if (args.length < 2) {
 System.out.println("JavaHBaseBulkLoadExample {outputPath} {tableName}");
 return;
 }
 String outputPath = args[0];
 String tableName = args[1];
 String columnFamily1 = "f1";
 String columnFamily2 = "f2";
 SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkLoadExample " + tableName);
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 try {
 List<String> list= new ArrayList<String>();
 // row1
 list.add("1," + columnFamily1 + ",b,1");
 // row3
 list.add("3," + columnFamily1 + ",a,2");
 list.add("3," + columnFamily1 + ",b,1");
 list.add("3," + columnFamily2 + ",a,1");
 /* row2 */
 list.add("2," + columnFamily2 + ",a,3");
 list.add("2," + columnFamily2 + ",b,3");
 JavaRDD<String> rdd = jsc.parallelize(list);
 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
 hbaseContext.bulkLoad(rdd, TableName.valueOf(tableName),new BulkLoadFunction(), outputPath,
 new HashMap<byte[], FamilyHFileWriteOptions>(), false, HConstants.DEFAULT_MAX_FILE_SIZE);
 } finally {
 jsc.stop();
 }
}
```

```
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkLoadExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
 if(args.length < 2) {
 println("HBaseBulkLoadExample {outputPath} {tableName}")
 return
 }
 LoginUtil.loginWithUserKeytab()
 val Array(outputPath, tableName) = args
 val columnFamily1 = "f1"
 val columnFamily2 = "f2"
 val sparkConf = new SparkConf().setAppName("JavaHBaseBulkLoadExample " + tableName)
 val sc = new SparkContext(sparkConf)
 try {
 val arr = Array("1," + columnFamily1 + ",b,1",
 "2," + columnFamily1 + ",a,2",
 "3," + columnFamily1 + ",b,1",
 "3," + columnFamily2 + ",a,1",
 "4," + columnFamily2 + ",a,3",
 "5," + columnFamily2 + ",b,3")

 val rdd = sc.parallelize(arr)
 val config = HBaseConfiguration.create
 val hbaseContext = new HBaseContext(sc, config)
 hbaseContext.bulkLoad[String](rdd,
 TableName.valueOf(tableName),
 (putRecord) => {
 if(putRecord.length > 0) {
 val strArray = putRecord.split(",")
 val kfq = new KeyFamilyQualifier(Bytes.toBytes(strArray(0)), Bytes.toBytes(strArray(1)),
Bytes.toBytes(strArray(2)))
 val ite = (kfq, Bytes.toBytes(strArray(3)))
 val itea = List(ite).iterator
 itea
 } else {
 null
 }
 },
 outputPath)
 } finally {
 sc.stop()
 }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkLoadPythonExample file in SparkOnHbasePythonExample.

```
-*- coding:utf-8 -*-
"""
[Note]
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
```

```
.builder\
 .appName("JavaHBaseBulkLoadExample")\
 .getOrCreate()
Import required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.HBaseBulkLoadPythonExample')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.HBaseBulkLoadPythonExample().hbaseBulkLoad(spark._jsc, sys.argv[1], sys.argv[2])
Stop the SparkSession instance.
spark.stop()
```

### 31.3.4.7 Using the foreachPartition Interface

#### Scenario

Users can use HBaseContext to perform operations on HBase in the Spark application, construct rowkey of the data to be inserted into RDDs, and write RDDs to HBase tables through the mapPartition interface of HBaseContext.

#### Data Planning

1. Run the **hbase shell** command on the client to go to the HBase command line.
2. Run the following command to create an HBase table:

```
create 'table2','cf1'
```

#### Development Guideline

1. Construct the data to be imported into RDDs.
2. Perform operations on HBase in HBaseContext mode and concurrently write data to HBase through the foreachPatition interface of HBaseContext.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.

#### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

#### Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseForEachParti
tionExample SparkOnHbaseJavaExample-1.0.jar table2 cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars
SparkOnHbaseJavaExample-1.0.jar HBaseForEachPartitionExample.py
table2 cf1
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseForEachParti
tionExample SparkOnHbaseJavaExample-1.0.jar table2 cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars
SparkOnHbaseJavaExample-1.0.jar HBaseForEachPartitionExample.py
table2 cf1
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `JavaHBaseForEachPartitionExample` file in `SparkOnHbaseJavaExample`.

```
public static void main(String[] args) throws IOException {
 if (args.length < 1) {
 System.out.println("JavaHBaseForEachPartitionExample {tableName} {columnFamily}");
 return;
 }
 final String tableName = args[0];
 final String columnFamily = args[1];
 SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkGetExample " + tableName);
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 try {
 List<byte[]> list = new ArrayList<byte[]>(5);
 list.add(Bytes.toBytes("1"));
 list.add(Bytes.toBytes("2"));
 list.add(Bytes.toBytes("3"));
 list.add(Bytes.toBytes("4"));
 list.add(Bytes.toBytes("5"));
 JavaRDD<byte[]> rdd = jsc.parallelize(list);
 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
 hbaseContext.foreachPartition(rdd,
 new VoidFunction<Tuple2<Iterator<byte[]>, Connection>>() {
 public void call(Tuple2<Iterator<byte[]>, Connection> t)
 throws Exception {
 Connection con = t._2();
 Iterator<byte[]> it = t._1();
 BufferedMutator buf = con.getBufferedMutator(TableNames.valueOf(tableName));
 while (it.hasNext()) {
 byte[] b = it.next();
 Put put = new Put(b);
 put.add(Bytes.toBytes(columnFamily), Bytes.toBytes("cid"), b);
 buf.mutate(put);
 }
 }
 })
 }
}
```

```
 mutator.flush();
 mutator.close();

 }
 });
} finally {
 jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseForEachPartitionExample` file in `SparkOnHbaseScalaExample`.

```
def main(args: Array[String]) {
 if (args.length < 2) {
 println("HBaseForEachPartitionExample {tableName} {columnFamily} are missing an arguments")
 return
 }
 val tableName = args(0)
 val columnFamily = args(1)
 val sparkConf = new SparkConf().setAppName("HBaseForEachPartitionExample " +
 tableName + " " + columnFamily)
 val sc = new SparkContext(sparkConf)
 try {
 //[(Array[Byte], Array[(Array[Byte], Array[Byte], Array[Byte])])]
 val rdd = sc.parallelize(Array(
 (Bytes.toBytes("1"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("1"))),
 (Bytes.toBytes("2"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("2"))),
 (Bytes.toBytes("3"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("3"))),
 (Bytes.toBytes("4"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("4"))),
 (Bytes.toBytes("5"),
 Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("5"))))
))
 val conf = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc, conf)
 rdd.hbaseForEachPartition(hbaseContext,
 (it, connection) => {
 val m = connection.getBufferedMutator(tableName.valueOf(tableName))
 it.foreach(r => {
 val put = new Put(r._1)
 r._2.foreach((putValue) =>
 put.addColumn(putValue._1, putValue._2, putValue._3))
 m.mutate(put)
 })
 m.flush()
 m.close()
 })
 } finally {
 sc.stop()
 }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseForEachPartitionExample` file in `SparkOnHbasePythonExample`.

```
-*- coding:utf-8 -*-
"""
```

[Note]

Pyspark does not provide HBase-related APIs. In this example, Python is used to invoke Java code.

```
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseForEachPartitionExample")\
 .getOrCreate()
Import the required class to sc._jvm.
java_import(spark._jvm,
'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseForEachPartitionExample')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseForEachPartitionExample().execute(spark._jsc, sys.argv)
Stop the SparkSession instance.
spark.stop()
```

### 31.3.4.8 Distributedly Scanning HBase Tables

#### Scenario

Users can use HBaseContext to perform operations on HBase in Spark applications and use HBase RDDs to scan HBase tables based on specific rules.

#### Data Planning

Use HBase tables created in [Performing Operation on Data in Avro Format](#)

#### Development Guideline

1. Set the scanning rule, for example: setCaching.
2. Use specific rules to scan the HBase table.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.

#### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

#### Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)



```
bin/spark-submit --master yarn --deploy-mode client --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseDistributedS
canExample SparkOnHbaseJavaExample-1.0.jar ExampleAvrotable
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars
SparkOnHbaseJavaExample-1.0.jar HBaseDistributedScanExample.py
ExampleAvrotable
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseDistributedS
canExample SparkOnHbaseJavaExample-1.0.jar ExampleAvrotable
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars
SparkOnHbaseJavaExample-1.0.jar HBaseDistributedScanExample.py
ExampleAvrotable
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `JavaHBaseDistributedScanExample` file in `SparkOnHbaseJavaExample`.

```
public static void main(String[] args) throws IOException{
 if (args.length < 1) {
 System.out.println("JavaHBaseDistributedScan {tableName}");
 return;
 }
 String tableName = args[0];
 SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseDistributedScan " + tableName);
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 try {
 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
 Scan scan = new Scan();
 scan.setCaching(100);
 JavaRDD<Tuple2<ImmutableBytesWritable, Result>> javaRdd =
 hbaseContext.hbaseRDD(tableName.valueOf(tableName), scan);
 List<String> results = javaRdd.map(new ScanConvertFunction()).collect();
 System.out.println("Result Size: " + results.size());
 } finally {
 jsc.stop();
 }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseDistributedScanExample` file in `SparkOnHbaseScalaExample`.

```
def main(args: Array[String]) {
 if (args.length < 1) {
 println("HBaseDistributedScanExample {tableName} missing an argument")
 return
 }
 val tableName = args(0)
```

```
val sparkConf = new SparkConf().setAppName("HBaseDistributedScanExample " + tableName)
val sc = new SparkContext(sparkConf)
try {
 val conf = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc, conf)
 val scan = new Scan()
 scan.setCaching(100)
 val getRdd = hbaseContext.hbaseRDD(tableName.valueOf(tableName), scan)
 getRdd.foreach(v => println(Bytes.toString(v._1.get())))
 println("Length: " + getRdd.map(r => r._1.copyBytes()).collect().length);
} finally {
 sc.stop()
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseDistributedScanExample file in SparkOnHbasePythonExample.

```
-*- coding:utf-8 -*-
-*- coding:utf-8 -*-
"""
[Note]
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseDistributedScan")\
 .getOrCreate()
Import the required class into sc._jvm.
java_import(spark._jvm,
'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseDistributedScanExample')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseDistributedScan().execute(spark._jsc, sys.argv)
Stop the SparkSession instance.
spark.stop()
```

### 31.3.4.9 Using the mapPartition Interface

#### Scenario

Users can use the HBaseContext method to perform operations on HBase in Spark applications and use the mapPartition interface to traverse HBase tables in parallel.

#### Data Planning

Use HBase tables created in [Using the foreachPartition Interface](#).

#### Development Guideline

1. Construct RDDs corresponding to rowkey in HBase tables to be traversed.
2. Use the mapPartition interface to traverse the data corresponding to the rowkey and perform simple operations.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, `$SPARK_HOME`) on the server where the Spark client is located.

### NOTE

To run the Spark on HBase example program, set `spark.yarn.security.credentials.hbase.enabled` (**false** by default) in the `spark-defaults.conf` file on the Spark client to **true**. Changing the `spark.yarn.security.credentials.hbase.enabled` value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item `spark.inputFormat.cache.enabled` to **false**.

## Submitting Commands

Assume that the JAR package name is `spark-hbaseContext-test-1.0.jar` that is stored in the `$SPARK_HOME` directory on the client. The following commands are executed in the `$SPARK_HOME` directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode client --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseMapPartitionExample SparkOnHbaseJavaExample-1.0.jar table2**  
Python version. (The file name must be the same as the actual one. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode client --jars SparkOnHbaseJavaExample-1.0.jar HBaseMapPartitionExample.py table2**
- yarn-cluster mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode cluster --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseMapPartitionExample SparkOnHbaseJavaExample-1.0.jar table2**  
Python version. (The file name must be the same as the actual one. The following is only an example.)  
**bin/spark-submit --master yarn --deploy-mode cluster --jars SparkOnHbaseJavaExample-1.0.jar HBaseMapPartitionExample.py table2**

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `JavaHBaseMapPartitionExample` file in `SparkOnHbaseJavaExample`.

```
public static void main(String args[]) throws IOException {
 if(args.length <1){
 System.out.println("JavaHBaseMapPartitionExample {tableName} is missing an argument");
 return;
 }
}
```

```
}
final String tableName = args[0];
SparkConf sparkConf = new SparkConf().setAppName("HBaseMapPartitionExample " + tableName);
JavaSparkContext jsc = new JavaSparkContext(sparkConf);
try{
 List<byte []> list = new ArrayList();
 list.add(Bytes.toBytes("1"));
 list.add(Bytes.toBytes("2"));
 list.add(Bytes.toBytes("3"));
 list.add(Bytes.toBytes("4"));
 list.add(Bytes.toBytes("5"));
 JavaRDD<byte []> rdd = jsc.parallelize(list);
 Configuration hbaseconf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, hbaseconf);
 JavaRDD getRDD = hbaseContext.mapPartitions(rdd, new
FlatMapFunction<Tuple2<Iterator<byte[]>,Connection>, Object>() {
 public Iterator call(Tuple2<Iterator<byte[]>, Connection> t)
 throws Exception {
 Table table = t._2.getTable(TableName.valueOf(tableName));
 //go through rdd
 List<String> list = new ArrayList<String>();
 while(t._1.hasNext()){
 byte[] bytes = t._1.next();
 Result result = table.get(new Get(bytes));
 Iterator<Cell> it = result.listCells().iterator();
 StringBuilder sb = new StringBuilder();
 sb.append(Bytes.toString(result.getRow()) + "-");
 while(it.hasNext()){
 Cell cell = it.next();
 String column = Bytes.toString(cell.getQualifierArray());
 if(column.equals("counter")){
 sb.append("(" + column + "," + Bytes.toLong(cell.getValueArray()) + ")");
 } else {
 sb.append("(" + column + "," + Bytes.toString(cell.getValueArray()) + ")");
 }
 }
 list.add(sb.toString());
 }
 return list.iterator();
 }
});
List<byte[]> resultList = getRDD.collect();
if(null == resultList || 0 == resultList.size()){
 System.out.println("Nothing matches!");
}else{
 for(int i=0; i< resultList.size(); i++){
 System.out.println(resultList.get(i));
 }
}
} finally {
 jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseMapPartitionExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
 if (args.length < 1) {
 println("HBaseMapPartitionExample {tableName} is missing an argument")
 return
 }
 val tableName = args(0)
 val sparkConf = new SparkConf().setAppName("HBaseMapPartitionExample " + tableName)
 val sc = new SparkContext(sparkConf)
 try {
 //[(Array[Byte])]
 }
}
```

```
val rdd = sc.parallelize(Array(
 Bytes.toBytes("1"),
 Bytes.toBytes("2"),
 Bytes.toBytes("3"),
 Bytes.toBytes("4"),
 Bytes.toBytes("5")))
val conf = HBaseConfiguration.create()
val hbaseContext = new HBaseContext(sc, conf)
val b = new StringBuilder
val getRdd = rdd.hbaseMapPartitions[String](hbaseContext, (it, connection) => {
 val table = connection.getTable(TableName.valueOf(tableName))
 it.map{r =>
 //batching would be faster. This is just an example
 val result = table.get(new Get(r))
 val it = result.listCells().iterator()
 b.append(Bytes.toString(result.getRow) + ":")
 while (it.hasNext) {
 val cell = it.next()
 val q = Bytes.toString(cell.getQualifierArray)
 if (q.equals("counter")) {
 b.append("(" + q + "," + Bytes.toLong(cell.getValueArray) + ")")
 } else {
 b.append("(" + q + "," + Bytes.toString(cell.getValueArray) + ")")
 }
 }
 b.toString()
 }
})
getRdd.collect().foreach(v => println(v))
} finally {
 sc.stop()
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseMapPartitionExample` file in `SparkOnHbasePythonExample`.

```
#!/usr/bin/env python
-*- coding:utf-8 -*-
"""
[Note]
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseMapPartitionExample")\
 .getOrCreate()
Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseMapPartitionExample')
Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseMapPartitionExample().execute(spark._jsc, sys.argv)
Stop the SparkSession instance.
spark.stop()
```

### 31.3.4.10 Writing Data to HBase Tables In Batches Using SparkStreaming

#### Scenario

Users can use `HBaseContext` to perform operations on HBase in Spark applications and write streaming data to HBase tables using the `streamBulkPut` interface.

## Data Planning

1. Create a session connected to the client and run the **hbase shell** command in the session to go to the HBase command line.
2. Run the following command in the HBase command line to create an HBase table:  
**create 'streamingTable','cf1'**
3. In another session of the client, run the Linux command to construct a port for receiving data. The command may be different for servers running different operating systems. For the SUSE operating system, the following command is used: **netcat -lk 9999**.

```
nc -lk 9999
```

### NOTE

To construct a port for receiving data, you need to install netcat on the server where the client is located.

## Development Guideline

1. Use SparkStreaming to continuously read data from a specific port.
2. Write the read Dstream to HBase tables through the streamBulkPut interface.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, `$SPARK_HOME`) on the server where the Spark client is located.

### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the `$SPARK_HOME` directory on the client. The following commands are executed in the `$SPARK_HOME` directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version. (The class name must be the same as the actual code. The following is only an example.) `#{ip}` must be the IP address of the host where the **nc -lk 9999** command is executed.

```
bin/spark-submit --master yarn --deploy-mode client --class
com.huawei.bigdata.spark.examples.streaming.JavaHBaseStreamingBulkPutExample SparkOnHbaseJavaExample-1.0.jar #{ip} 9999 streamingTable
cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --jars SparkOnHbaseJavaExample-1.0.jar
HBaseStreamingBulkPutExample.py ${ip} 9999 streamingTable cf1
```

- yarn-cluster mode:

Java/Scala version. (The class name must be the same as the actual code. The following is only an example.) **\${ip}** must be the IP address of the host where the **nc -lk 9999** command is executed.

```
bin/spark-submit --master yarn --deploy-mode client --deploy-mode
cluster --class
com.huawei.bigdata.spark.examples.streaming.JavaHBaseStreamingBulkP
utExample SparkOnHbaseJavaExample-1.0.jar ${ip} 9999 streamingTable
cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars
SparkOnHbaseJavaExample-1.0.jar HBaseStreamingBulkPutExample.py $
{ip} 9999 streamingTable cf1
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `JavaHBaseStreamingBulkPutExample` file in `SparkOnHbaseJavaExample`.

### NOTE

The `awaitTerminationOrTimeout()` method is used to set the task timeout interval (in milliseconds). You are advised to set this parameter based on the expected task execution time.

```
public static void main(String[] args) throws IOException {
 if (args.length < 4) {
 System.out.println("JavaHBaseBulkPutExample " +
 "{host} {port} {tableName}");
 return;
 }
 String host = args[0];
 String port = args[1];
 String tableName = args[2];
 String columnFamily = args[3];
 SparkConf sparkConf =
 new SparkConf().setAppName("JavaHBaseStreamingBulkPutExample " +
 tableName + ":" + port + ":" + tableName);
 JavaSparkContext jsc = new JavaSparkContext(sparkConf);
 try {
 JavaStreamingContext jssc =
 new JavaStreamingContext(jsc, new Duration(1000));
 JavaReceiverInputDStream<String> javaDstream =
 jssc.socketTextStream(host, Integer.parseInt(port));
 Configuration conf = HBaseConfiguration.create();
 JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
 hbaseContext.streamBulkPut(javaDstream,
 TableName.valueOf(tableName),
 new PutFunction(columnFamily));
 jssc.start();
 jssc.awaitTerminationOrTimeout(60000);
 jssc.stop(false,true);
 } catch (InterruptedException e){
 e.printStackTrace();
 } finally {
```

```
jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseStreamingBulkPutExample` file in `SparkOnHbaseScalaExample`.

### NOTE

The `awaitTerminationOrTimeout()` method is used to set the task timeout interval (in milliseconds). You are advised to set this parameter based on the expected task execution time.

```
def main(args: Array[String]): Unit = {
 val host = args(0)
 val port = args(1)
 val tableName = args(2)
 val columnFamily = args(3)
 val conf = new SparkConf()
 conf.setAppName("HBase Streaming Bulk Put Example")
 val sc = new SparkContext(conf)
 try {
 val config = HBaseConfiguration.create()
 val hbaseContext = new HBaseContext(sc, config)
 val ssc = new StreamingContext(sc, Seconds(1))
 val lines = ssc.socketTextStream(host, port.toInt)
 hbaseContext.streamBulkPut[String](lines,
 TableName.valueOf(tableName),
 (putRecord) => {
 if (putRecord.length() > 0) {
 val put = new Put(Bytes.toBytes(putRecord))
 put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes("foo"), Bytes.toBytes("bar"))
 put
 } else {
 null
 }
 })
 ssc.start()
 ssc.awaitTerminationOrTimeout(60000)
 ssc.stop(stopSparkContext = false)
 } finally {
 sc.stop()
 }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseStreamingBulkPutExample` file in `SparkOnHbasePythonExample`.

```
-*- coding:utf-8 -*-
"""
[Note]
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
Create a SparkSession instance.
spark = SparkSession\
 .builder\
 .appName("JavaHBaseStreamingBulkPutExample")\
 .getOrCreate()
Import required class to sc._jvm.
java_import(spark._jvm,
```



```
'com.huawei.bigdata.spark.examples.streaming.JavaHBaseStreamingBulkPutExample')
Create class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseStreamingBulkPutExample().execute(spark._jsc, sys.argv)
Stop SparkSession.
spark.stop()
```

## 31.3.5 Reading Data from HBase and Write It Back to HBase

### 31.3.5.1 Overview

#### Scenarios

Assume table1 of HBase stores the user consumption amount of the current day and table2 stores the history consumption data.

In table1, key=1 and cf:cid=100 indicate that the consumption amount of user 1 in the current day is 100 CNY.

In table2, key=1 and cf:cid=1000 indicate that the history consumption amount of user 1 is 1000 CNY.

The Spark application shall achieve the following function:

Add the current consumption amount (100) to the history consumption amount (1000).

The running result is that the total consumption amount of user 1 (key=1) in table2 is 1100 CNY (cf:cid=1100).

#### Data Preparation

Use the Spark-Beeline tool to create table1 and table2 (Spark table and HBase table, respectively), and insert data by HBase.

**Step 1** Ensure that JDBCServer is started. On the Spark2x client, perform the following operations using the Spark-Beeline command tool:

**Step 2** Use the Spark-Beeline tool to create Spark table1:

```
create table table1

(
key string,
cid string
)
using org.apache.spark.sql.hbase.HBaseSource
options(
hbaseTableName "table1",
keyCols "key",
colsMapping "cid=cf.cid");
```

**Step 3** Run the following command on HBase to insert data to table1:

```
put 'table1', '1', 'cf:cid', '100'
```

**Step 4** Use the Spark-Beeline tool to create Spark table2:

```
create table table2
(
 key string,
 cid string
)
using org.apache.spark.sql.hbase.HBaseSource
options(
 hbaseTableName "table2",
 keyCols "key",
 colsMapping "cid=cf.cid");
```

**Step 5** Run the following command on HBase to insert data to table2 :

```
put 'table2', '1', 'cf:cid', '1000'
```

----End

## Development Idea

1. Query the data in table1.
2. Query the data in table2 using the key value of table1.
3. Add up the queried data.
4. Write the results of the preceding step to table2.

## Packaging the Project

1. Use the Maven tool provided by IDEA to pack the project and generate a JAR file (The class name and file name must be the same as those in the actual code. The following is only an example). For details, see [Compiling and Running the Application](#).
2. Upload the JAR file to any directory (for example, `/opt/female/`) on the server where the Spark client is located.

### NOTE

Before running the sample program, set the `spark.yarn.security.credentials.hbase.enabled` configuration item to `true` in the `spark-defaults.conf` configuration file of Spark client. (The default value is `false`. Changing the value to `true` does not affect existing services. If you want to uninstall the HBase service, change the value back to `false` first.)

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the `bin/spark-submit` script to run the code:

- Run Java or Scala sample code.  
**bin/spark-submit --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.SparkHbasetoHbase --master yarn --deploy-mode client /opt/female/SparkHbasetoHbase-1.0.jar**
- Run the Python sample program.

#### NOTE

- PySpark does not provide HBase-related APIs. Therefore, Python is used to invoke Java code in this sample. Use Maven to pack the provided Java code into a JAR package and place it in the same driver class directory. When running the Python program, configure `--jars` to load the JAR package to the directory where the Python file resides.

```
bin/spark-submit --master yarn --deploy-mode client --conf spark.yarn.user.classpath.first=true --jars /opt/female/SparkHbasetoHbasePythonExample/SparkHbasetoHbase-1.0.jar,/opt/female/protobuf-java-2.5.0.jar /opt/female/SparkHbasetoHbasePythonExample/SparkHbasetoHbasePythonExample.py
```

## 31.3.5.2 Java Example Code

### Function

Call HBase API using Spark to operate HBase table1, analyze the data, and then write the analyzed data to HBase table2.

### Example Code

For details about the code, see the class `com.huawei.bigdata.spark.examples.SparkHbasetoHbase`.

Example code:

```
/**
 * calculate data from hbase1/hbase2,then update to hbase2
 */
public class SparkHbasetoHbase {

 public static void main(String[] args) throws Exception {

 SparkConf conf = new SparkConf().setAppName("SparkHbasetoHbase");
 conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer");
 conf.set("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrator");
 JavaSparkContext jsc = new JavaSparkContext(conf);
 // Create the configuration parameter to connect the HBase. The hbase-site.xml must be included in the
 classpath.
 Configuration hbConf = HBaseConfiguration.create(jsc.hadoopConfiguration());

 // Declare the information of the table to be queried.
 Scan scan = new org.apache.hadoop.hbase.client.Scan();
 scan.addFamily(Bytes.toBytes("cf")); //column family
 org.apache.hadoop.hbase.protobuf.generated.ClientProtos.Scan proto = ProtobufUtil.toScan(scan);
 String scanToString = Base64.encodeBytes(proto.toByteArray());
 hbConf.set(TableInputFormat.INPUT_TABLE, "table1"); //table name
 hbConf.set(TableInputFormat.SCAN, scanToString);

 // Obtain the data in the table through the Spark interface.
 JavaPairRDD rdd = jsc.newAPIHadoopRDD(hbConf, TableInputFormat.class,
 ImmutableBytesWritable.class, Result.class);
```

```

// Traverse every Partition in the HBase table1 and update the HBase table2
//If less data, you can use rdd.foreach()

rdd.foreachPartition(
 new VoidFunction<Iterator<Tuple2<ImmutableBytesWritable, Result>>>() {
 public void call(Iterator<Tuple2<ImmutableBytesWritable, Result>> iterator) throws Exception {
 hBaseWriter(iterator);
 }
 }
);

jsc.stop();
}

/**
 * write to table2 in exetutor
 *
 * @param iterator partition data from table1
 */
private static void hBaseWriter(Iterator<Tuple2<ImmutableBytesWritable, Result>> iterator) throws
IOException {
 //read hbase
 String tableName = "table2";
 String columnFamily = "cf";
 String qualifier = "cid";
 Configuration conf = HBaseConfiguration.create();

 Connection connection = null;
 Table table = null;

 try {
 connection = ConnectionFactory.createConnection(conf);
 table = connection.getTable(TableName.valueOf(tableName));

 List<Get> rowList = new ArrayList<Get>();
 List<Tuple2<ImmutableBytesWritable, Result>> table1List = new
ArrayList<Tuple2<ImmutableBytesWritable, Result>>();
 while (iterator.hasNext()) {
 Tuple2<ImmutableBytesWritable, Result> item = iterator.next();
 Get get = new Get(item._2().getRow());
 table1List.add(item);
 rowList.add(get);
 }

 //get data from hbase table2
 Result[] resultDataBuffer = table.get(rowList);

 //set data for hbase
 List<Put> putList = new ArrayList<Put>();
 for (int i = 0; i < resultDataBuffer.length; i++) {
 Result resultData = resultDataBuffer[i]; //hbase2 row
 if (!resultData.isEmpty()) {
 //query hbase1Value
 String hbase1Value = "";
 Iterator<Cell> it = table1List.get(i)._2().listCells().iterator();
 while (it.hasNext()) {
 Cell c = it.next();
 // query table1 value by column family and column qualifier
 if (columnFamily.equals(Bytes.toString(CellUtil.cloneFamily(c)))
 && qualifier.equals(Bytes.toString(CellUtil.cloneQualifier(c)))) {
 hbase1Value = Bytes.toString(CellUtil.cloneValue(c));
 }
 }

 String hbase2Value = Bytes.toString(resultData.getValue(columnFamily.getBytes(),
qualifier.getBytes()));
 Put put = new Put(table1List.get(i)._2().getRow());

```

```
 //calculate result value
 int resultValue = Integer.parseInt(hbase1Value) + Integer.parseInt(hbase2Value);
 //set data to put
 put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes(qualifier),
Bytes.toBytes(String.valueOf(resultValue)));
 putList.add(put);
 }
}

if (putList.size() > 0) {
 table.put(putList);
}
} catch (IOException e) {
 e.printStackTrace();
} finally {
 if (table != null) {
 try {
 table.close();
 } catch (IOException e) {
 e.printStackTrace();
 }
 }
 if (connection != null) {
 try {
 // Close the HBase connection
 connection.close();
 } catch (IOException e) {
 e.printStackTrace();
 }
 }
}
}
}
```

### 31.3.5.3 Scala Example Code

#### Function

Call HBase API using Spark to operate HBase table1, analyze the data, and then write the analyzed data to HBase table2.

#### Example Code

For details about code, see [com.huawei.bigdata.spark.examples.SparkHbasetoHbase](#).

Example code:

```
/**
 * calculate data from hbase1/hbase2,then update to hbase2
 */
object SparkHbasetoHbase {

 case class FemaleInfo(name: String, gender: String, stayTime: Int)

 def main(args: Array[String]) {

 val conf = new SparkConf().setAppName("SparkHbasetoHbase")
 conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
 conf.set("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrator")
 val sc = new SparkContext(conf)
 // Create the configuration parameter to connect the HBase. The hbase-site.xml must be included in the
 classpath.
 val hbConf = HBaseConfiguration.create(sc.hadoopConfiguration)
```

```
// Declare the information of the table to be queried.
val scan = new Scan()
scan.addFamily(Bytes.toBytes("cf")) //column family
val proto = ProtobufUtil.toScan(scan)
val scanToString = Base64.encodeBytes(proto.toByteArray)
hbConf.set(TableInputFormat.INPUT_TABLE, "table1") //table name
hbConf.set(TableInputFormat.SCAN, scanToString)

// Obtain the data in the table through the Spark interface.
val rdd = sc.newAPIHadoopRDD(hbConf, classOf[TableInputFormat], classOf[ImmutableBytesWritable],
classOf[Result])

// Traverse every Partition in the HBase table1 and update the HBase table2
//If less data, you can use rdd.foreach()
rdd.foreachPartition(x => hBaseWriter(x))

sc.stop()
}

/**
 * write to table2 in exetutor
 *
 * @param iterator partition data from table1
 */
def hBaseWriter(iterator: Iterator[(ImmutableBytesWritable, Result]]): Unit = {
//read hbase
val tableName = "table2"
val columnFamily = "cf"
val qualifier = "cid"
val conf = HBaseConfiguration.create()

var table: Table = null
var connection: Connection = null

try {
connection = ConnectionFactory.createConnection(conf)
table = connection.getTable(TableName.valueOf(tableName))

val iteratorArray = iterator.toArray
val rowList = new util.ArrayList[Get]()
for (row <- iteratorArray) {
val get = new Get(row._2.getRow)
rowList.add(get)
}

//get data from hbase table2
val resultDataBuffer = table.get(rowList)

//set data for hbase
val putList = new util.ArrayList[Put]()
for (i <- 0 until iteratorArray.size) {
val resultData = resultDataBuffer(i) //hbase2 row
if (!resultData.isEmpty) {
//query hbase1Value
var hbase1Value = ""
val it = iteratorArray(i)._2.listCells().iterator()
while (it.hasNext) {
val c = it.next()
// query table1 value by column family and column qualifier
if (columnFamily.equals(Bytes.toString(CellUtil.cloneFamily(c)))
&& qualifier.equals(Bytes.toString(CellUtil.cloneQualifier(c)))) {
hbase1Value = Bytes.toString(CellUtil.cloneValue(c))
}
}
}
val hbase2Value = Bytes.toString(resultData.getValue(columnFamily.getBytes, qualifier.getBytes))
val put = new Put(iteratorArray(i)._2.getRow)

//calculate result value
```



```
-*- coding:utf-8 -*-

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession

Create the SparkContext and set kryo serialization.
spark = SparkSession\
 .builder\
 .appName("SparkHbaseToHbase") \
 .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
 .config("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrator") \
 .getOrCreate()

Import the class that will run into sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.SparkHbaseToHbase')

Create a class instance and invoke the method.
spark._jvm.SparkHbaseToHbase().hbaseToHbase(spark._jsc)

Stop the SparkSession
spark.stop()
```

## 31.3.6 Reading Data from Hive and Write It to HBase

### 31.3.6.1 Overview

#### Scenarios

Assume that table **person** of Hive stores the user consumption amount of the current day and table2 of HBase stores the history consumption data.

In table **person**, name=1 and account=100 indicates that the consumption amount of user 1 in the current day is 100 CNY.

In table2, key=1 and cf:cid=1000 indicate that the history consumption amount of user 1 is 1000 CNY.

The Spark application shall achieve the following function:

Add the current consumption amount (100) to the history consumption amount (1000).

The running result is that the total consumption amount of user 1 (key=1) in table2 is 1100 CNY (cf:cid=1100).

#### Data Preparation

Before develop the application, create the Hive table **person** and insert data to it. Create HBase table2.

**Step 1** Place the source log file to HDFS.

1. Create a blank file **log1.txt** in the local and write the following content to the file:  
1,100
2. Create a directory **/tmp/input** in HDFS and copy the **log1.txt** file to the directory.



- a. On the Linux HDFS client, run the **hadoop fs -mkdir /tmp/input** command (a `hdfs dfs` command provides the same function.) to create a directory.
- b. On the Linux HDFS client, run the **hadoop fs -put log1.txt /tmp/input** command to upload the data file.

**Step 2** Store the imported data to the Hive table.

Ensure that JDBCServer is started. Use the Beeline tool to create a Hive table and insert data to it.

1. Run the following commands to create the Hive table person:

```
create table person
(
 name STRING,
 account INT
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ESCAPED BY '\\'
STORED AS TEXTFILE;
```

2. Run the following command to insert data to the table:

```
load data inpath '/tmp/input/log1.txt' into table person;
```

**Step 3** Create a HBase table:

Ensure that JDBCServer is started. Use the Spark-beeline command tool to create a HBase table and insert data to it.

1. Run the following commands to create the HBase table table2:

```
create table table2
(
 key string,
 cid string
)
using org.apache.spark.sql.hbase.HBaseSource
options(
 hbaseTableName "table2",
 keyCols "key",
 colsMapping "cid=cf.cid");
```

2. Run the following command to insert data to the table:

```
put 'table2', '1', 'cf:cid', '1000'
```

----End

## Development Idea

1. Query the data in Hive table person.
2. Query the data in table2 using the key value of table person.
3. Add the queried data.

4. Write the results of the preceding step to table2.

## Packaging the Project

1. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
2. Upload the JAR file to any directory (for example, `/opt/female/`) on the server where the Spark client is located.

### NOTE

Before running the sample program, set the `spark.yarn.security.credentials.hbase.enabled` configuration item to `true` in the `spark-defaults.conf` configuration file of Spark client. (The default value is `false`. Changing the value to `true` does not affect existing services. If you want to uninstall the HBase service, change the value back to `false` first.)

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the `bin/spark-submit` script to run the code (The class name and file name must be the same as those in the actual code. The following is only an example):

- Run Java or Scala sample code.
  - `bin/spark-submit --class com.huawei.bigdata.spark.examples.SparkHivetoHbase --master yarn --deploy-mode client /opt/female/SparkHivetoHbase-1.0.jar`

- Run the Python sample program

### NOTE

- PySpark does not provide HBase-related APIs. Therefore, Python is used to invoke Java code in this sample. Use Maven to pack the provided Java code into a JAR file and place it in the same directory. When running the Python program, use `--jars` to load the JAR file to `classpath`
  - `bin/spark-submit --master yarn --deploy-mode client --jars /opt/female/SparkHivetoHbasePythonExample/SparkHivetoHbase-1.0.jar /opt/female/SparkHivetoHbasePythonExample/SparkHivetoHbasePythonExample.py`

### 31.3.6.2 Java Example Code

## Function

In Spark application, call Hive API using Spark to operate Hive table, analyze the data, and then write the analyzed data to the HBase table.

## Example Code

For details about code, see `com.huawei.bigdata.spark.examples.SparkHbaseToHbase`.

Example code

```
/**
 * calculate data from hive/hbase,then update to hbase
```

```

*/
public class SparkHivetoHbase {

 public static void main(String[] args) throws Exception {

 // Obtain the data in the table through the Spark interface.
 SparkConf conf = new SparkConf().setAppName("SparkHivetoHbase");
 JavaSparkContext jsc = new JavaSparkContext(conf);
 HiveContext sqlContext = new org.apache.spark.sql.hive.HiveContext(jsc);

 SparkSession spark = SparkSession
 .builder()
 .appName("SparkHivetoHbase")
 .config("spark.sql.warehouse.dir", "spark-warehouse")
 .enableHiveSupport()
 .getOrCreate();

 Dataset<Row>dataFrame = spark.sql("select name, account from person");

 // Traverse every Partition in the hive table and update the hbase table
 // If less data, you can use rdd.foreach()

 dataFrame.toJavaRDD().foreachPartition(
 new VoidFunction<Iterator<Row>>() {
 public void call(Iterator<Row> iterator) throws Exception {
 hBaseWriter(iterator);
 }
 }
);

 spark.stop();
 }

 /**
 * write to hbase table in exetutor
 *
 * @param iterator partition data from hive table
 */
 private static void hBaseWriter(Iterator<Row> iterator) throws IOException {
 // read hbase
 String tableName = "table2";
 String columnFamily = "cf";
 Configuration conf = HBaseConfiguration.create();

 Connection connection = null;
 Table table = null;
 try {
 connection = ConnectionFactory.createConnection(conf);
 table = connection.getTable(TableName.valueOf(tableName));

 List<Row> table1List = new ArrayList<Row>();
 List<Get> rowList = new ArrayList<Get>();
 while (iterator.hasNext()) {
 Row item = iterator.next();
 Get get = new Get(item.getString(0).getBytes());
 table1List.add(item);
 rowList.add(get);
 }

 // get data from hbase table
 Result[] resultDataBuffer = table.get(rowList);

 // set data for hbase
 List<Put> putList = new ArrayList<Put>();
 for (int i = 0; i < resultDataBuffer.length; i++) {
 // hbase row
 Result resultData = resultDataBuffer[i];
 if (!resultData.isEmpty()) {
 // get hiveValue

```



```

val spark = SparkSession
 .builder()
 .appName("SparkHiveHbase")
 .config("spark.sql.warehouse.dir", "spaek-warehouse")
 .enableHiveSupport()
 .getOrCreate()

import spark.implicits._
val dataframe = spark.sql("select name, account from person")

// Traverse every Partition in the hive table and update the hbase table
// If less data, you can use rdd.foreach()
dataFrame.rdd.foreachPartition(x => hBaseWriter(x))

spark.stop()
}

/**
 * write to hbase table in exetutor
 *
 * @param iterator partition data from hive table
 */
def hBaseWriter(iterator: Iterator[Row]): Unit = {
 // read hbase
 val tableName = "table2"
 val columnFamily = "cf"
 val conf = HBaseConfiguration.create()

 var table: Table = null
 var connection: Connection = null

 try {
 connection = ConnectionFactory.createConnection(conf)
 table = connection.getTable(TableName.valueOf(tableName))

 val iteratorArray = iterator.toArray
 val rowList = new util.ArrayList[Get]()
 for (row <- iteratorArray) {
 val get = new Get(row.getString(0).getBytes)
 rowList.add(get)
 }

 // get data from hbase table
 val resultDataBuffer = table.get(rowList)

 // set data for hbase
 val putList = new util.ArrayList[Put]()
 for (i <- 0 until iteratorArray.size) {
 // hbase row
 val resultData = resultDataBuffer(i)
 if (!resultData.isEmpty) {
 // get hiveValue
 var hiveValue = iteratorArray(i).getInt(1)

 // get hbaseValue by column Family and colomn qualifier
 val hbaseValue = Bytes.toString(resultData.getValue(columnFamily.getBytes, "cid".getBytes))
 val put = new Put(iteratorArray(i).getString(0).getBytes)

 // calculate result value
 val resultValue = hiveValue + hbaseValue.toInt

 // set data to put
 put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes("cid"),
Bytes.toBytes(resultValue.toString))
 putList.add(put)
 }
 }

 if (putList.size() > 0) {

```

```
 table.put(putList)
 }
} catch {
 case e: IOException =>
 e.printStackTrace();
} finally {
 if (table != null) {
 try {
 table.close()
 } catch {
 case e: IOException =>
 e.printStackTrace();
 }
 }
 if (connection != null) {
 try {
 //Close the HBase connection.
 connection.close()
 } catch {
 case e: IOException =>
 e.printStackTrace()
 }
 }
}
}
```

### 31.3.6.4 Python Example Code

#### Function

In a Spark application, use Spark to call a Hive API to operate a Hive table, and write the data analysis result of the Hive table to an HBase table.

#### Example Code

PySpark does not provide HBase related APIs. In this example, Python with the Java programming language invoked is used.

The following code snippets are used as an example. For complete codes, see [SparkHivetoHbasePythonExample](#).

```
-*- coding:utf-8 -*-

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession

Create the SparkSession
spark = SparkSession\
 .builder\
 .appName("SparkHivetoHbase") \
 .getOrCreate()

Import the class that will run into sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.SparkHivetoHbase')

Create a class instance and invoke the method.
spark._jvm.SparkHivetoHbase().hivetohbase(spark._jsc)

Stop the SparkSession
spark.stop()
```

### 31.3.7 Streaming Connecting to Kafka0-10

### 31.3.7.1 Overview

#### Scenarios

Assume that the Kafka component receives one word record every 1 second.

The developed Spark application needs to achieve the following function:

Calculate the sum of records for each word in real time.

**log1.txt** example file:

```
LiuYang
YuanJing
GuoYijun
CaiXuyu
Liyuan
FangBo
LiuYang
YuanJing
GuoYijun
CaiXuyu
FangBo
```

#### Data Planning

Spark Streaming sample project data is stored in the Kafka component. A user with Kafka permission sends data to Kafka component.

1. Ensure that the cluster, including HDFS, Yarn, Spark, and Kafka is successfully installed.

2. Create the **input\_data1.txt** file in the local and copy the content of the **log1.txt** file to the **input\_data1.txt** file.

On the client installation node, create the **/home/data** directory and upload the **input\_data1.txt** file to the **/home/data** directory.

3. Create a topic.

{zkQuorum} indicates ZooKeeper cluster information. The format is IP:port.

```
$KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper {zkQuorum}/
kafka --replication-factor 1 --partitions 3 --topic {Topic}
```

4. Start the Producer of Kafka and send data to Kafka.

```
java -cp {ClassPath}
com.huawei.bigdata.spark.examples.StreamingExampleProducer
{BrokerList} {Topic}
```

In this command, **ClassPath** must contain the absolute path of the Kafka JAR package on the Spark client in addition to the path of the sample JAR package, for example: **/opt/client/Spark2x/spark/jars/\*:/opt/client/Spark2x/spark/jars/streamingClient010/\*:{ClassPath}**.

#### Development Approach

1. Receive data from Kafka and generate DStream.
2. Collect the statistics of word records by category.
3. Calculate and print the result.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, `/opt`) on the server where the Spark client is located.
- Prepare dependency packages and upload the following JAR packages to the `$$SPARK_HOME/jars/streamingClient010` directory on the server where the Spark client is located.
  - `spark-streaming-kafkaWriter-0-10_2.12-3.1.1-hw-ei-311001.jar`
  - `kafka-clients-xxx.jar`
  - `spark-sql-kafka-0-10_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar`
  - `spark-streaming-kafka-0-10_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar`
  - `spark-token-provider-kafka-0-10_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar`

### NOTE

- For the dependency package whose version number contains "hw-ei", download from the Huawei open-source image site. For details, see [Configuring Huawei Open-Source Mirrors](#).
- For the dependency package whose version number does not contain "hw-ei", obtain them from the Maven central repository. For details, see [Obtaining Sample Projects](#).

## Running Tasks

When running the sample program, you need to specify `<checkpointDir>`, `<brokers>`, `<topic>`, and `<batchTime>`. `<checkPointDir>` indicates the path for storing the program result backup in HDFS. `<brokers>` indicates the Kafka address for obtaining metadata. `<topic>` indicates the topic name read from Kafka. `<batchTime>` indicates the interval for Streaming processing in batches.

### NOTE

The path of Spark Streaming's Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is `$$SPARK_HOME/jars`, and the path of the Kafka dependency package is `$$SPARK_HOME/jars/streamingClient010`. Therefore, when running an application, you need to add a configuration item to the `spark-submit` command to specify the path of the dependency package of Spark Streaming Kafka, for example, `--jars $(files=($$SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "${files[*]}")`.

Go to the Spark client directory and run the following commands to invoke the `bin/spark-submit` script to run the code (The class name and file name must be the same as those in the actual code. The following is only an example.):

- Sample code (**Spark Streaming read Kafka 0-10 Write To Print**)  

```
bin/spark-submit --master yarn --deploy-mode client --jars $(files=($$SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "${files[*]}") --class com.huawei.bigdata.spark.examples.KafkaWordCount /opt/SparkStreamingKafka010JavaExample-1.0.jar <checkpointDir> <brokers> <topic> <batchTime>
```
- Sample code (**Spark Streaming Write To Kafka 0-10**)  

```
bin/spark-submit --master yarn --deploy-mode client --jars $(files=($$SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "${files[*]}")
```



```
{files[*]}") --class
com.huawei.bigdata.spark.examples.JavaDstreamKafkaWriter /opt/
SparkStreamingKafka010JavaExample-1.0.jar <groupId> <brokers> <topics>
```

### 31.3.7.2 Java Example Code

#### Function

In Spark applications, use Streaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word, and write the data to Kafka 0-10.

#### Code Sample (Streaming Read Data from Kafka 0-10)

The following code is an example. For details, see `com.huawei.bigdata.spark.examples.KafkaWordCount`.

```
/**
 * Consumes messages from one or more topics in Kafka.
 * <checkPointDir> is the Spark Streaming checkpoint directory.
 * <brokers> is for bootstrapping and the producer will only use it for getting metadata
 * <topics> is a list of one or more kafka topics to consume from
 * <batchTime> is the Spark Streaming batch duration in seconds.
 */
public class KafkaWordCount
{
 public static void main(String[] args) {
 JavaStreamingContext ssc = createContext(args);
 //The Streaming system starts.
 ssc.start();
 try {
 ssc.awaitTermination();
 } catch (InterruptedException e) {
 }
 }
 private static JavaStreamingContext createContext(String[] args) {
 String checkPointDir = args[0];
 String brokers = args[1];
 String topics = args[2];
 String batchSize = args[3];

 // Create a Streaming startup environment.
 SparkConf sparkConf = new SparkConf().setAppName("KafkaWordCount");
 JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, new
 Duration(Long.parseLong(batchTime) * 1000));

 //Configure the CheckPoint directory for the Streaming.
 //This parameter is mandatory because of existence of the window concept.
 ssc.checkpoint(checkPointDir);

 // Get the list of topic used by kafka
 String[] topicArr = topics.split(",");
 Set<String> topicSet = new HashSet<String>(Arrays.asList(topicArr));
 Map<String, Object> kafkaParams = new HashMap();
 kafkaParams.put("bootstrap.servers", brokers);
 kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
 kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
 kafkaParams.put("group.id", "DemoConsumer");

 LocationStrategy locationStrategy = LocationStrategies.PreferConsistent();
 ConsumerStrategy consumerStrategy = ConsumerStrategies.Subscribe(topicSet, kafkaParams);

 // Create direct kafka stream with brokers and topics
 // Receive data from the Kafka and generate the corresponding DStream
```

```
JavaInputDStream<ConsumerRecord<String, String>> messages = KafkaUtils.createDirectStream(ssc,
locationStrategy, consumerStrategy);

// Obtain field properties in each row.
JavaDStream<String> lines = messages.map(new Function<ConsumerRecord<String, String>, String>() {
 @Override
 public String call(ConsumerRecord<String, String> tuple2) throws Exception {
 return tuple2.value();
 }
});

// Aggregate the total time that calculate word count
JavaPairDStream<String, Integer> wordCounts = lines.mapToPair(
 new PairFunction<String, String, Integer>() {
 @Override
 public Tuple2<String, Integer> call(String s) {
 return new Tuple2<String, Integer>(s, 1);
 }
 }).reduceByKey(new Function2<Integer, Integer, Integer>() {
 @Override
 public Integer call(Integer i1, Integer i2) {
 return i1 + i2;
 }
}).updateStateByKey(
 new Function2<List<Integer>, Optional<Integer>, Optional<Integer>>() {
 @Override
 public Optional<Integer> call(List<Integer> values, Optional<Integer> state) {
 int out = 0;
 if (state.isPresent()) {
 out += state.get();
 }
 for (Integer v : values) {
 out += v;
 }
 return Optional.of(out);
 }
 });

// print the results
wordCounts.print();
return ssc;
}
```

## Example Code (Streaming Write To Kafka 0-10)

The following code segment is only an example. For details, see [com.huawei.bigdata.spark.examples.DstreamKafkaWriter](#).

### NOTE

It is advisable to use the new API `createDirectStream` instead of the old API `createStream` for application development. The old API continues to exist, but its performance and stability are worse than the new API.

```
/**
 * Parameter description:
 * <groupId> is the group ID for the consumer.
 * <brokers> is for bootstrapping and the producer will only use
 * <topic> is a kafka topic to consume from.
 */
public class JavaDstreamKafkaWriter {

 public static void main(String[] args) throws InterruptedException {
 if (args.length != 3) {
 System.err.println("Usage: JavaDstreamKafkaWriter <groupId> <brokers> <topic>");
 System.exit(1);
 }
 }
}
```

```
final String groupId = args[0];
final String brokers = args[1];
final String topic = args[2];

SparkConf sparkConf = new SparkConf().setAppName("KafkaWriter");

// Populate Kafka properties
Map<String, Object> kafkaParams = new HashMap<String, Object>();
kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
kafkaParams.put("value.serializer", "org.apache.kafka.common.serialization.ByteArraySerializer");
kafkaParams.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
kafkaParams.put("bootstrap.servers", brokers);
kafkaParams.put("group.id", groupId);
kafkaParams.put("auto.offset.reset", "smallest");

// Create Spark Java streaming context
JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.milliseconds(500));

// Populate data to write to kafka
List<String> sendData = new ArrayList();
sendData.add("kafka_writer_test_msg_01");
sendData.add("kafka_writer_test_msg_02");
sendData.add("kafka_writer_test_msg_03");

// Create Java RDD queue
Queue<JavaRDD<String>> sent = new LinkedList();
sent.add(ssc.sparkContext().parallelize(sendData));

// Create java Dstream with the data to be written
JavaDStream wStream = ssc.queueStream(sent);

// Write to kafka
JavaDStreamKafkaWriterFactory.fromJavaDStream(wStream).writeToKafka(
 JavaConverters.mapAsScalaMapConverter(kafkaParams).asScala(),
 new Function<String, ProducerRecord<String, byte[]>>() {
 public ProducerRecord<String, byte[]> call(String s) throws Exception {
 return new ProducerRecord(topic, s.toString().getBytes());
 }
 });

ssc.start();
ssc.awaitTermination();
}
```

### 31.3.7.3 Scala Example Code

#### Function

In Spark applications, use Streaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word, and write the data to Kafka 0-10.

#### Code Sample (Streaming Read Data from Kafka 0-10)

The following code is an example. For details, see `com.huawei.bigdata.spark.examples.KafkaWordCount`.

```
/**
 * Consumes messages from one or more topics in Kafka.
 * <checkPointDir> is the Spark Streaming checkpoint directory.
 * <brokers> is for bootstrapping and the producer will only use it for getting metadata
 * <topics> is a list of one or more kafka topics to consume from
 */
```

```
* <batchTime> is the Spark Streaming batch duration in seconds.
*/
public class KafkaWordCount
{
 public static void main(String[] args) {
 JavaStreamingContext ssc = createContext(args);
 //The Streaming system starts.
 ssc.start();
 try {
 ssc.awaitTermination();
 } catch (InterruptedException e) {
 }
 }
 private static JavaStreamingContext createContext(String[] args) {
 String checkPointDir = args[0];
 String brokers = args[1];
 String topics = args[2];
 String batchSize = args[3];
 // Create a Streaming startup environment.
 SparkConf sparkConf = new SparkConf().setAppName("KafkaWordCount");
 JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, new
Duration(Long.parseLong(batchSize) * 1000));
 //Configure the CheckPoint directory for the Streaming.
 //This parameter is mandatory because of existence of the window concept.
 ssc.checkpoint(checkPointDir);
 // Get the list of topic used by kafka
 String[] topicArr = topics.split(",");
 Set<String> topicSet = new HashSet<String>(Arrays.asList(topicArr));
 Map<String, Object> kafkaParams = new HashMap();
 kafkaParams.put("bootstrap.servers", brokers);
 kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
 kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
 kafkaParams.put("group.id", "DemoConsumer");
 LocationStrategy locationStrategy = LocationStrategies.PreferConsistent();
 ConsumerStrategy consumerStrategy = ConsumerStrategies.Subscribe(topicSet, kafkaParams);
 // Create direct kafka stream with brokers and topics
 // Receive data from the Kafka and generate the corresponding DStream
 JavaInputDStream<ConsumerRecord<String, String>> messages = KafkaUtils.createDirectStream(ssc,
locationStrategy, consumerStrategy);
 // Obtain field properties in each row.
 JavaDStream<String> lines = messages.map(new Function<ConsumerRecord<String, String>, String>() {
 @Override
 public String call(ConsumerRecord<String, String> tuple2) throws Exception {
 return tuple2.value();
 }
 });
 // Aggregate the total time that calculate word count
 JavaPairDStream<String, Integer> wordCounts = lines.mapToPair(
new PairFunction<String, String, Integer>() {
 @Override
 public Tuple2<String, Integer> call(String s) {
 return new Tuple2<String, Integer>(s, 1);
 }
 }).reduceByKey(new Function2<Integer, Integer, Integer>() {
 @Override
 public Integer call(Integer i1, Integer i2) {
 return i1 + i2;
 }
 });
 wordCounts.updateStateByKey(
new Function2<List<Integer>, Optional<Integer>, Optional<Integer>>() {
 @Override
 public Optional<Integer> call(List<Integer> values, Optional<Integer> state) {
 int out = 0;
 if (state.isPresent()) {
 out += state.get();
 }
 for (Integer v : values) {
 out += v;
 }
 }
 });
 }
}
```

```
 return Optional.of(out);
 }
 });
 // print the results
 wordCounts.print();
 return ssc;
}
}
```

## Example Code (Streaming Write To Kafka 0-10)

The following code segment is only an example. For details, see `com.huawei.bigdata.spark.examples.DstreamKafkaWriter`.

### NOTE

After updates to Spark, it is advisable to use the new API `createDirectStream` instead of the old API `createStream` for application development. The old API continues to exist, but its performance and stability are worse than the new API.

```
package com.huawei.bigdata.spark.examples

import scala.collection.mutable
import scala.language.postfixOps

import com.huawei.spark.streaming.kafka010.KafkaWriter._
import org.apache.kafka.clients.producer.ProducerRecord
import org.apache.spark.SparkConf
import org.apache.spark.rdd.RDD
import org.apache.spark.streaming.{Milliseconds, StreamingContext}

/**
 * Exaple code to demonstrate the usage of dstream.writeToKafka API
 *
 * Parameter description:
 * <groupId> is the group ID for the consumer.
 * <brokers> is for bootstrapping and the producer will only use
 * <topic> is a kafka topic to consume from.
 */
object DstreamKafkaWriter {
 def main(args: Array[String]) {

 if (args.length != 3) {
 System.err.println("Usage: DstreamKafkaWriter <groupId> <brokers> <topic>")
 System.exit(1)
 }

 val Array(groupId, brokers, topic) = args
 val sparkConf = new SparkConf().setAppName("KafkaWriter")

 // Populate Kafka properties
 val kafkaParams = Map[String, String](
 "bootstrap.servers" -> brokers,
 "value.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
 "key.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
 "value.serializer" -> "org.apache.kafka.common.serialization.ByteArraySerializer",
 "key.serializer" -> "org.apache.kafka.common.serialization.StringSerializer",
 "group.id" -> groupId,
 "auto.offset.reset" -> "latest"
)

 // Create Spark streaming context
 val ssc = new StreamingContext(sparkConf, Milliseconds(500));

 // Populate data to write to kafka
 val sentData = Seq("kafka_writer_test_msg_01", "kafka_writer_test_msg_02",
 "kafka_writer_test_msg_03")
```

```
// Create RDD queue
val sent = new mutable.Queue[RDD[String]]()
sent.enqueue(ssc.sparkContext.makeRDD(sentData))

// Create Dstream with the data to be written
val wStream = ssc.queueStream(sent)

// Write to kafka
wStream.writeToKafka(kafkaParams,
(x: String) => new ProducerRecord[String, Array[Byte]](topic, x.getBytes))

ssc.start()
ssc.awaitTermination()
}
```

## 31.3.8 Structured Streaming Project

### 31.3.8.1 Overview

#### Scenarios

In Spark applications, use StructuredStreaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word.

#### Data Planning

Sample project data of StructuredStreaming is stored in Kafka. A user with Kafka permission sends data to Kafka.

1. Ensure that the cluster, including HDFS, Yarn, Spark, and Kafka is successfully installed.
2. Create a topic.

zkQuorum} indicates ZooKeeper cluster information. The format is IP:port.

```
$KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper {zkQuorum}/
kafka --replication-factor 1 --partitions 1 --topic {Topic}
```

3. Start the Producer of Kafka and send data to Kafka.

{ClassPath} indicates the storage path of engineer JAR packages and is specified by the user. For details, see [Compiling and Running the Application](#).

```
java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/streamingClient010/*:
{ClassPath}
com.huawei.bigdata.spark.examples.KafkaWordCountProducer
{BrokerList} {Topic} {messagesPerSec} {wordsPerMessage}
```

#### Development Approach

1. Receive data from Kafka and generate DataStreamReader.
2. Collect the statistics of word records.
3. Calculate and print the result.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, `/opt`) on the server where the Spark client is located.
- Upload `commons-pool2-xxx.jar` to the `$$SPARK_HOME/jars/streamingClient010/` directory. The JAR package can be obtained from the `$$SPARK_HOME/tool/carbonPrequery` directory.

## Running Tasks

- When running the sample application, you need to specify `<brokers>`, `<subscribe-type>`, `<topic>`, and `<checkpointDir>`. `<brokers>` indicates the Kafka address for obtaining metadata (port `21007` is required), `<subscribe-type>` indicates the Kafka subscription type (for example, `subscribe`), and `<topic>` indicates the name of the topic read from Kafka. `<checkpointDir>` indicates the path for storing the checkpoint file.

### NOTE

The path of the Spark Structured Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is `$$SPARK_HOME/jars`. Whereas the path of the Spark Structured Streaming Kafka dependency package is `$$SPARK_HOME/jars/streamingClient010`. Therefore, when running an application, you need to add a configuration item to the `spark-submit` command to specify the path of the dependency package of Spark Streaming Kafka, for example, `--jars $(files=($$SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "${files[*]}")`

---

### CAUTION

When submitting a structured stream task, you need to run the `--jars` command to specify the path of the Kafka-related JAR file. For the current version, you need to copy the `kafka-clients.jar` file from the `$$SPARK_HOME/jars/streamingClient010` directory to the `$$SPARK_HOME/jars` directory. Otherwise, the "class not found" error is reported.

---

Go to the Spark client directory and run the following commands to invoke the `bin/spark-submit` script to run the code (The class name and file name must be the same as those in the actual code. The following is only an example).

- Run Java or Scala sample code:  

```
bin/spark-submit --master yarn --deploy-mode client --jars $(files=($$SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "${files[*]}") --class com.huawei.bigdata.spark.examples.KafkaWordCount /opt/SparkStructuredStreamingScalaExample-1.0.jar <brokers> <subscribe-type> <topic> <checkpointDir>
```

The configuration example is as follows:

*If an error indicating that the user does not have the permission to read and write the local directory is reported, the `spark.sql.streaming.checkpointLocation` parameter must be specified, and the user must have the read and write permissions on the directory specified by this parameter.*

- Run the Python sample code:

 NOTE

When running the Python sample code, you need to add the JAR package of the Java project to the **streamingClient/** directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars $(
files=$(SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "${
files[*]}") /opt/female/SparkStructuredStreamingPythonExample/
KafkaWordCount.py <brokers> <subscribe-type> <topic> <checkpointDir>
```

### 31.3.8.2 Java Sample Code

#### Function

In Spark applications, use StructuredStreaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word.

#### Code Sample

The following code is an example. For details, see [com.huawei.bigdata.spark.examples.KafkaWordCount](http://com.huawei.bigdata.spark.examples.KafkaWordCount).

 NOTE

When new data is available in Streaming DataFrame/Dataset, **outputMode** is used for configuring data written to the Streaming receptor.

```
public class KafkaWordCount
{
 public static void main(String[] args) throws Exception {
 if (args.length < 3) {
 System.err.println("Usage: KafkaWordCount <bootstrap-servers> " +
 "<subscribe-type> <topics>");
 System.exit(1);
 }

 String bootstrapServers = args[0];
 String subscribeType = args[1];
 String topics = args[2];

 SparkSession spark = SparkSession
 .builder()
 .appName("KafkaWordCount")
 .getOrCreate();

 // Create DataSet representing the stream of input lines from kafka
 Dataset<String> lines = spark
 .readStream()
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option(subscribeType, topics)
 .load()
 .selectExpr("CAST(value AS STRING)")
 .as(Encoders.STRING());

 // Generate running word count
 Dataset<Row> wordCounts = lines.flatMap(new FlatMapFunction<String, String>() {
 @Override
 public Iterator<String> call(String x) {
 return Arrays.asList(x.split(" ")).iterator();
 }
 }, Encoders.STRING()).groupBy("value").count();
```



```
// Start running the query that prints the running counts to the console
StreamingQuery query = wordCounts.writeStream()
 .outputMode("complete")
 .format("console")
 .start();

query.awaitTermination();
}
}
```

### 31.3.8.3 Scala Sample Code

#### Function

In Spark applications, use StructuredStreaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word.

#### Code Sample

The following code is an example. For details, see `com.huawei.bigdata.spark.examples.KafkaWordCount`.

#### NOTE

When new data is available in Streaming DataFrame/Dataset, **outputMode** is used for configuring data written to the Streaming receptor.

```
object KafkaWordCount {
 def main(args: Array[String]): Unit = {
 if (args.length < 3) {
 System.err.println("Usage: KafkaWordCount <bootstrap-servers> " +
 "<subscribe-type> <topics>")
 System.exit(1)
 }

 val Array(bootstrapServers, subscribeType, topics) = args

 val spark = SparkSession
 .builder
 .appName("KafkaWordCount")
 .getOrCreate()

 import spark.implicits._

 // Create DataSet representing the stream of input lines from kafka
 val lines = spark
 .readStream
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option(subscribeType, topics)
 .load()
 .selectExpr("CAST(value AS STRING)")
 .as[String]

 // Generate running word count
 val wordCounts = lines.flatMap(_.split(" ")).groupBy("value").count()

 // Start running the query that prints the running counts to the console
 val query = wordCounts.writeStream
 .outputMode("complete")
 .format("console")
 .start()

 query.awaitTermination()
 }
}
```

### 31.3.8.4 Python Sample Code

#### Function

In Spark applications, use StructuredStreaming to invoke Kafka APIs to obtain word records. Classify word records to obtain the number of records of each word.

#### Sample Code

The following code segment is only an example. For details, see SecurityKafkaWordCount.

#### NOTE

When there is new available data in Streaming DataFrame/Dataset, **outputMode** indicates data written to the Streaming receiver. The default value is append. To change the output mode, see the outputMode description of Spark2x > Scala in [conref:text]Target does not exist .

```
#!/usr/bin/python
-*- coding: utf-8 -*-

import sys
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, split

if __name__ == "__main__":
 if len(sys.argv) < 3:
 print("Usage: <bootstrapServers> <subscribeType> <topics>")
 exit(-1)

 bootstrapServers = sys.argv[1]
 subscribeType = sys.argv[2]
 topics = sys.argv[3]

 # Initialize the SparkSession.
 spark = SparkSession.builder.appName("KafkaWordCount").getOrCreate()

 # Create the DataFrame of input lines stream from Kafka.
 # In security mode, set spark/conf/jaas.conf and jaas-zk.conf to KafkaClient.
 lines = spark.readStream.format("kafka")\
 .option("kafka.bootstrap.servers", bootstrapServers)\
 .option(subscribeType, topics)\
 .load()\
 .selectExpr("CAST(value AS STRING)")

 # Split lines into words.
 words = lines.select(explode(split(lines.value, " ")).alias("word"))
 # Generate the running word count.
 wordCounts = words.groupBy("word").count()

 # Start to query whether the running counts are printed to the console.
 query = wordCounts.writeStream\
 .outputMode("complete")\
 .format("console")\
 .start()

 query.awaitTermination()
```

### 31.3.9 Structured Streaming Stream-Stream Join

### 31.3.9.1 Overview

#### Scenarios

An advertisement service involves advertisement request events, advertisement display events, and advertisement click events. The advertiser needs to collect statistics on valid advertisement displays and advertisement click data.

Known conditions are as follows:

1. Each time a user requests an advertisement, an advertisement request event is generated and saved to the adRequest topic of Kafka.
2. After an advertisement is requested, the advertisement may be displayed for multiple times. Each time the advertisement is displayed, an advertisement display event is generated and saved to the adShow topic of Kafka.
3. Each advertisement may be clicked for multiple times. Each time it is clicked, an advertisement click event is generated and saved to the adClick topic of Kafka.
4. For advertisement display:
  - a. If the duration from the time when a request is generated to the time when the advertisement is displayed exceeds A minutes, the display is invalid.
  - b. Advertisement display events generated during A minutes are valid events.
5. For advertisement click:
  - a. If the duration from the display event to the click event exceeds B minutes, the click is invalid.
  - b. If there are multiple click events within B minutes, only the first click event is valid.

The simple data structure in this scenario is as follows:

- Advertisement request event  
Data structure: adID^reqTime
- Advertisement display event  
Data structure: adID^showID^showTime
- Advertisement click event  
Data structure: adID^showID^clickTime

Data association relationships are as follows:

- The advertisement request event is associated with the advertisement display event through the adID.
- The advertisement display event is associated with the advertisement click event through the adID and showID.

Data requirements:

- The delay from the time when data is generated to the time when the data reaches the stream processing engine does not exceed two hours.

- The time for advertisement request events, advertisement display events, and advertisement click events reach the stream processing engine may not be in sequence or be aligned with each other.

## Data Planning

1. Enable users with the Kafka permission to generate simulation data in Kafka.

```
java -cp $SPARK_HOME/conf:$SPARK_HOME/jars/*:$SPARK_HOME/jars/
streamingClient010/*:{ClassPath}
com.huawei.bigdata.spark.examples.KafkaADEventProducer {BrokerList}
{timeOfProduceReqEvent} {eventTimeBeforeCurrentTime} {reqTopic}
{reqEventCount} {showTopic} {showEventMaxDelay} {clickTopic}
{clickEventMaxDelay}
```

### NOTE

- Ensure that the clusters are installed, including HDFS, Yarn, Spark2x and Kafka.
- Set the **allow.everyone.if.no.acl.found** parameter of Kafka Broker to **true**.
- Start Kafka Producer and enable it to send data to Kafka.
- {ClassPath} indicates the storage path of engineer JAR packages and is specified by the user. For details, see [Compiling and Running the Application](#).

Command example:

```
java -cp /opt/client/Spark2x/spark/conf:/opt/
StructuredStreamingADScalaExample-1.0.jar:/opt/client/Spark2x/spark/
jars/*:/opt/client/Spark2x/spark/jars/streamingClient010/*
com.huawei.bigdata.spark.examples.KafkaADEventProducer
10.132.190.170:21005,10.132.190.165:21005 2h 1h req 10000000 show 5m
click 5m
```

This command creates three topics on Kafka: req, show, and click. Ten million data records of request events are generated in two hours. The time range of the request events is from one hour ahead of the current time to the current time, and up to five display events are randomly generated for each request event. The time range of the display events is from the request event time to five minutes after the request event time. Up to five click events are randomly generated for each display event. The time range of the click events is from the display event time to five minutes after the display event time.

## Development Guideline

1. Use structured streaming to receive data from Kafka and generate request flows, display flows, and click flows.
2. Perform join query of data in request flows, display flows, and click flows.
3. Write the statistics result to Kafka.
4. Monitor the flow processing task status in the application.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, /opt) on the server where the Spark client is located.

- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.

## Running Tasks

When running the sample program, you need to specify **<kafkaBootstrapServers>**, **<maxEventDelay>**, **<reqTopic>**, **<showTopic>**, **<maxShowDelay>**, **<clickTopic>**, **<maxClickDelay>**, **<triggerInterver>**, and **<checkpointDir>**. **<kafkaBootstrapServers>** indicates the Kafka address (port 21005 is required) for obtaining metadata, and **<maxEventDelay>** indicates the maximum delay from data generation to stream processing. **<reqTopic>** indicates the topic name of the request event. **<showTopic>** indicates the topic name of the display event. **<maxShowDelay>** indicates the maximum delay for effectively displaying the event, and **<clickTopic>** indicates the topic name of the click event. **<maxClickDelay>** indicates the maximum delay of a valid click event. **<triggerInterver>** indicates the interval for triggering a stream processing task. **<checkpointDir>** indicates the path for storing the checkpoint file.

### NOTE

The path of the Spark Structured Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is **\$SPARK\_HOME/jars**. Whereas the path of the Spark Structured Streaming Kafka dependency package is **\$SPARK\_HOME/jars/streamingClient010**. Therefore, when running an application, you need to add a configuration item to the **spark-submit** command to specify the path of the dependency package of Spark Streaming Kafka, for example, **--jars \$(files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar); IFS=,; echo "\${files[\*]}")**

### CAUTION

When submitting a structured stream task, you need to run the **--jars** command to specify the path of the Kafka-related JAR file. For the current version, you need to cope the **kafka-clients.jar** file from the **\$SPARK\_HOME/jars/streamingClient010** directory to the **\$SPARK\_HOME/jars** directory. Otherwise, the "class not found" error is reported.

Go to the Spark client directory and run the following command to invoke the **bin/spark-submit** script to run the code (The class name and file name must be the same as those in the actual code. The following is only an example):

```
bin/spark-submit --master yarn --deploy-mode client --jars $(files=($SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "${files[*]}")
--conf "spark.sql.streaming.statefulOperator.checkCorrectness.enabled=false"
--class com.huawei.bigdata.spark.examples.KafkaADCount /opt/StructuredStreamingADScalaExample-1.0.jar <kafkaBootstrapServers>
<maxEventDelay> <reqTopic> <showTopic> <maxShowDelay> <clickTopic>
<maxClickDelay> <triggerInterver> <checkpointDir>
```

## 31.3.9.2 Scala Example Code

### Function

Structured Streaming is used to read advertisement request data, display data, and click data from Kafka, obtain effective display statistics and click statistics in real time, and write the statistics to Kafka.

### Example code

The following code snippets are used as an example. For complete codes, see `com.huawei.bigdata.spark.examples.KafkaADCount`.

```
/**
 * Run the Structured Streaming task to collect statistics on valid advertisement display and click data. The
 * result is written into Kafka.
 */
object KafkaADCount {
 def main(args: Array[String]): Unit = {
 if (args.length < 12) {
 System.err.println("Usage: KafkaWordCount <bootstrap-servers> " +
 "<maxEventDelay> <reqTopic> <showTopic> <maxShowDelay> " +
 "<clickTopic> <maxClickDelay> <triggerInterver> " +
 "<checkpointLocation> <protocol> <service> <domain>")
 System.exit(1)
 }

 val Array(bootstrapServers, maxEventDelay, reqTopic, showTopic,
 maxShowDelay, clickTopic, maxClickDelay, triggerInterver, checkpointLocation,
 protocol, service, domain) = args

 val maxEventDelayMills = JavaUtils.timeStringAs(maxEventDelay, TimeUnit.MILLISECONDS)
 val maxShowDelayMills = JavaUtils.timeStringAs(maxShowDelay, TimeUnit.MILLISECONDS)
 val maxClickDelayMills = JavaUtils.timeStringAs(maxClickDelay, TimeUnit.MILLISECONDS)
 val triggerMills = JavaUtils.timeStringAs(triggerInterver, TimeUnit.MILLISECONDS)

 val spark = SparkSession
 .builder
 .appName("KafkaADCount")
 .getOrCreate()

 spark.conf.set("spark.sql.streaming.checkpointLocation", checkpointLocation)

 import spark.implicits._

 // Create DataSet representing the stream of input lines from kafka
 val reqDf = spark
 .readStream
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option("kafka.security.protocol", protocol)
 .option("kafka.sasl.kerberos.service.name", service)
 .option("kafka.kerberos.domain.name", domain)
 .option("subscribe", reqTopic)
 .load()
 .selectExpr("CAST(value AS STRING)")
 .as[String]
 .map{
 _.split('^') match {
 case Array(reqAdID, reqTime) => ReqEvent(reqAdID,
 Timestamp.valueOf(reqTime))
 }
 }
 .as[ReqEvent]
 .withWatermark("reqTime", maxEventDelayMills +
```

```
maxShowDelayMills + " millisecond")

val showDf = spark
 .readStream
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option("kafka.security.protocol", protocol)
 .option("kafka.sasl.kerberos.service.name", service)
 .option("kafka.kerberos.domain.name", domain)
 .option("subscribe", showTopic)
 .load()
 .selectExpr("CAST(value AS STRING)")
 .as[String]
 .map{
 _._split('^') match {
 case Array(showAdID, showID, showTime) => ShowEvent(showAdID,
 showID, Timestamp.valueOf(showTime))
 }
 }
 .as[ShowEvent]
 .withWatermark("showTime", maxEventDelayMills +
 maxShowDelayMills + maxClickDelayMills + " millisecond")

val clickDf = spark
 .readStream
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option("kafka.security.protocol", protocol)
 .option("kafka.sasl.kerberos.service.name", service)
 .option("kafka.kerberos.domain.name", domain)
 .option("subscribe", clickTopic)
 .load()
 .selectExpr("CAST(value AS STRING)")
 .as[String]
 .map{
 _._split('^') match {
 case Array(clickAdID, clickShowID, clickTime) => ClickEvent(clickAdID,
 clickShowID, Timestamp.valueOf(clickTime))
 }
 }
 .as[ClickEvent]
 .withWatermark("clickTime", maxEventDelayMills + " millisecond")

val showStaticsQuery = reqDf.join(showDf,
 expr(s"""
 reqAdID = showAdID
 AND showTime >= reqTime + interval ${maxShowDelayMills} millisecond
 """))
 .selectExpr("concat_ws('^', showAdID, showID, showTime) as value")
 .writeStream
 .queryName("showEventStatics")
 .outputMode("append")
 .trigger(Trigger.ProcessingTime(triggerMills.millis))
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option("kafka.security.protocol", protocol)
 .option("kafka.sasl.kerberos.service.name", service)
 .option("kafka.kerberos.domain.name", domain)
 .option("topic", "showEventStatics")
 .start()

val clickStaticsQuery = showDf.join(clickDf,
 expr(s"""
 showAdID = clickAdID AND
 showID = clickShowID AND
 clickTime >= showTime + interval ${maxClickDelayMills} millisecond
 """), joinType = "rightouter")
 .dropDuplicates("showAdID")
 .selectExpr("concat_ws('^', clickAdID, clickShowID, clickTime) as value")
```

```
.writeStream
.queryName("clickEventStatics")
.outputMode("append")
.trigger(Trigger.ProcessingTime(triggerMills.millis))
.format("kafka")
.option("kafka.bootstrap.servers", bootstrapServers)
.option("kafka.security.protocol", protocol)
.option("kafka.sasl.kerberos.service.name", service)
.option("kafka.kerberos.domain.name", domain)
.option("topic", "clickEventStatics")
.start()

new Thread(new Runnable {
 override def run(): Unit = {
 while(true) {
 println("-----get showStatic progress-----")
 //println(showStaticsQuery.lastProgress)
 }
 }
}).start()

spark.streams.awaitAnyTermination()
}
```

## 31.3.10 Structured Streaming Status Operation

### 31.3.10.1 Overview

#### Scenarios

Assume that you need to collect statistics on the number of events in each session and the start and end timestamp of the sessions.

You need to export the sessions that are in the updated state in this batch.

#### Data Planning

1. Generate simulated data in Kafka (the Kafka permission is required).
2. Ensure that the cluster has been installed, including the HDFS, Yarn, Spark2x, and Kafka services.
3. Create a topic.

**{zkQuorum}** indicates ZooKeeper cluster information in the *IP address.Port number* format.

***\$KAFKA\_HOME/bin/kafka-topics.sh --create --zookeeper {zkQuorum}/kafka --replication-factor 1 --partitions 1 --topic {Topic}***

4. Start the Producer of Kafka and send data to Kafka.

**{ClassPath}** indicates the storage path of the project JAR package that is specified by the user. For details, see [Compiling and Running the Application](#).

```
java -cp $SPARK_HOME/conf:$SPARK_HOME/jars/*:$SPARK_HOME/jars/streamingClient010/*:{ClassPath}
```



```
com.huawei.bigdata.spark.examples.KafkaProducer {brokerlist} {topic}
{number of events produce every 0.02s}
```

Example:

```
java -cp /opt/client/Spark2x/spark/conf:/opt/
StructuredStreamingState-1.0.jar:/opt/client/Spark2x/spark/jars*/:/opt/client/
Spark2x/spark/jars/streamingClient010/*
com.huawei.bigdata.spark.examples.KafkaProducer xxx.xxx.xxx.xxx:
21005,xxx.xxx.xxx.xxx:21005,xxx.xxx.xxx.xxx:21005 mytopic 10
```

## Development Guideline

1. Receive data from Kafka and generate the corresponding DataStreamReader.
2. Collect statistics by category.
3. Calculate the result and print it.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file.
- Upload the JAR package to any directory (for example, **/opt**) on the server where the Spark client is located.

## Running Tasks

When running the example program, you need to specify **<brokers>**, **<subscribe-type>**, **<topic>**, and **<checkpointLocation>**, where **<brokers>** indicates the Kafka address for obtaining metadata (port 21005 is required). **<subscribe-type>** indicates the Kafka consumption mode, **<topic>** indicates the Kafka topic to be consumed, and **<checkpointLocation>** indicates the path for storing the checkpoint of the Spark task.

### NOTE

The path of the Spark Structured Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is **\$SPARK\_HOME/jars**. Whereas the path of the Spark Streaming Structured Kafka dependency package is **\$SPARK\_HOME/jars/streamingClient010**. Therefore, when running an application, you need to add a configuration item to the **spark-submit** command to specify the path of the dependency package of Spark Streaming Kafka, for example, **--jars \$(files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar); IFS=,; echo "\${files[\*]}")**

Go to the Spark client directory and run the following command to invoke the **bin/spark-submit** script to run the code (The class name and file name must be the same as those in the actual code. The following is only an example):

- **bin/spark-submit --master yarn --deploy-mode client --jars \$(files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar); IFS=,; echo "\${files[\*]}") --class com.huawei.bigdata.spark.examples.kafkaSessionization /opt/StructuredStreamingState-1.0.jar <brokers> <subscribe-type> <topic> <checkpointLocation>**

 **CAUTION**

When submitting a structured stream task, you need to run the `--jars` command to specify the path of the Kafka-related JAR file. For the current version, you need to copy the `kafka-clients.jar` file from the `$SPARK_HOME/jars/streamingClient010` directory to the `$SPARK_HOME/jars` directory. Otherwise, the "class not found" error is reported.

## 31.3.10.2 Scala Sample Code

### Function

In the Spark structure flow application, the number of events in each session and the start and end timestamp of the sessions are collected in different batches. At the same time, the system exports the sessions that are in the updated state in this batch.

### Code Example

The following code snippets are used as an example. For complete codes, see [com.huawei.bigdata.spark.examples.kafkaSessionization](#).

 **NOTE**

When new data is available in Streaming DataFrame/Dataset, **outputMode** is used for configuring data written to the Streaming receiver.

```
object kafkaSessionization {
 def main(args: Array[String]): Unit = {
 if (args.length < 7) {
 System.err.println("Usage: kafkaSessionization <bootstrap-servers> " +
 "<subscribe-type> <protocol> <service> <domain> <topics> <checkpointLocation>")
 System.exit(1)
 }

 val Array(bootstrapServers, subscribeType, protocol, service, domain, topics, checkpointLocation) = args

 val spark = SparkSession
 .builder
 .appName("kafkaSessionization")
 .getOrCreate()

 spark.conf.set("spark.sql.streaming.checkpointLocation", checkpointLocation)

 spark.streams.addListener(new StreamingQueryListener {

 @volatile private var startTime: Long = 0L
 @volatile private var endTime: Long = 0L
 @volatile private var numRecs: Long = 0L

 override def onQueryStarted(event: StreamingQueryListener.QueryStartedEvent): Unit = {
 println("Query started: " + event.id)
 startTime = System.currentTimeMillis
 }

 override def onQueryProgress(event: StreamingQueryListener.QueryProgressEvent): Unit = {
 println("Query made progress: " + event.progress)
 numRecs += event.progress.numInputRows
 }

 override def onQueryTerminated(event: StreamingQueryListener.QueryTerminatedEvent): Unit = {
```

```

println("Query terminated: " + event.id)
endTime = System.currentTimeMillis
}
})

import spark.implicits._

val df = spark
 .readStream
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option("kafka.security.protocol", protocol)
 .option("kafka.sasl.kerberos.service.name", service)
 .option("kafka.kerberos.domain.name", domain)
 .option(subscribeType, topics)
 .load()
 .selectExpr("CAST(value AS STRING)")
 .as[String]
 .map { x =>
 val splitStr = x.split(",")
 (splitStr(0), Timestamp.valueOf(splitStr(1)))
 }.as[(String, Timestamp)].flatMap { case (line, timestamp) =>
 line.split(" ").map(word => Event(sessionId = word, timestamp))}

// Sessionize the events. Track number of events, start and end timestamps of session, and
// and report session updates.
val sessionUpdates = df
 .groupByKey(event => event.sessionId)
 .mapGroupsWithState[SessionInfo, SessionUpdate](GroupStateTimeout.ProcessingTimeTimeout) {

 case (sessionId: String, events: Iterator[Event], state: GroupState[SessionInfo]) =>

 // If timed out, then remove session and send final update
 if (state.hasTimedOut) {
 val finalUpdate =
 SessionUpdate(sessionId, state.get.durationMs, state.get.numEvents, expired = true)
 state.remove()
 finalUpdate
 } else {
 // Update start and end timestamps in session
 val timestamps = events.map(_.timestamp.getTime).toSeq
 val updatedSession = if (state.exists) {
 val oldSession = state.get
 SessionInfo(
 oldSession.numEvents + timestamps.size,
 oldSession.startTimestampMs,
 math.max(oldSession.endTimestampMs, timestamps.max))
 } else {
 SessionInfo(timestamps.size, timestamps.min, timestamps.max)
 }
 state.update(updatedSession)

 // Set timeout such that the session will be expired if no data received for 10 seconds
 state.setTimeoutDuration("10 seconds")
 SessionUpdate(sessionId, state.get.durationMs, state.get.numEvents, expired = false)
 }
 }

// Start running the query that prints the session updates to the console
val query = sessionUpdates
 .writeStream
 .outputMode("update")
 .format("console")
 .start()

query.awaitTermination()
}

```

```

object kafkaSessionization {
 def main(args: Array[String]): Unit = {

```

```

if (args.length < 7) {
 System.err.println("Usage: kafkaSessionization <bootstrap-servers> " +
 "<subscribe-type> <protocol> <service> <domain> <topics> <checkpointLocation>")
 System.exit(1)
}

val Array(bootstrapServers, subscribeType, protocol, service, domain, topics, checkpointLocation) = args

val spark = SparkSession
 .builder
 .appName("kafkaSessionization")
 .getOrCreate()

spark.conf.set("spark.sql.streaming.checkpointLocation", checkpointLocation)

spark.streams.addListener(new StreamingQueryListener {

 @volatile private var startTime: Long = 0L
 @volatile private var endTime: Long = 0L
 @volatile private var numRecs: Long = 0L

 override def onQueryStarted(event: StreamingQueryListener.QueryStartedEvent): Unit = {
 println("Query started: " + event.id)
 startTime = System.currentTimeMillis
 }

 override def onQueryProgress(event: StreamingQueryListener.QueryProgressEvent): Unit = {
 println("Query made progress: " + event.progress)
 numRecs += event.progress.numInputRows
 }

 override def onQueryTerminated(event: StreamingQueryListener.QueryTerminatedEvent): Unit = {
 println("Query terminated: " + event.id)
 endTime = System.currentTimeMillis
 }
})

import spark.implicits._

val df = spark
 .readStream
 .format("kafka")
 .option("kafka.bootstrap.servers", bootstrapServers)
 .option("kafka.security.protocol", protocol)
 .option("kafka.sasl.kerberos.service.name", service)
 .option("kafka.kerberos.domain.name", domain)
 .option(subscribeType, topics)
 .load()
 .selectExpr("CAST(value AS STRING)")
 .as[String]
 .map { x =>
 val splitStr = x.split(",")
 (splitStr(0), Timestamp.valueOf(splitStr(1)))
 }.as[(String, Timestamp)].flatMap { case (line, timestamp) =>
 line.split(" ").map(word => Event(sessionId = word, timestamp))}

// Sessionize the events. Track number of events, start and end timestamps of session, and
// and report session updates.
val sessionUpdates = df
 .groupByKey(event => event.sessionId)
 .mapGroupsWithState[SessionInfo, SessionUpdate](GroupStateTimeout.ProcessingTimeTimeout) {

 case (sessionId: String, events: Iterator[Event], state: GroupState[SessionInfo]) =>

 // If timed out, then remove session and send final update
 if (state.hasTimedOut) {
 val finalUpdate =
 SessionUpdate(sessionId, state.get.durationMs, state.get.numEvents, expired = true)
 }
 }
 }

```

```
state.remove()
finalUpdate
} else {
// Update start and end timestamps in session
val timestamps = events.map(_.timestamp.getTime).toSeq
val updatedSession = if (state.exists) {
val oldSession = state.get
SessionInfo(
oldSession.numEvents + timestamps.size,
oldSession.startTimestampMs,
math.max(oldSession.endTimestampMs, timestamps.max))
} else {
SessionInfo(timestamps.size, timestamps.min, timestamps.max)
}
state.update(updatedSession)

// Set timeout such that the session will be expired if no data received for 10 seconds
state.setTimeoutDuration("10 seconds")
SessionUpdate(sessionId, state.get.durationMs, state.get.numEvents, expired = false)
}
}
// Start running the query that prints the session updates to the console
val query = sessionUpdates
.writeStream
.outputMode("update")
.format("console")
.start()

query.awaitTermination()
}
```

## 31.3.11 Synchronizing HBase Data from Spark to CarbonData

### 31.3.11.1 Overview

#### Scenarios

Data is written to HBase in real time for point query services and is synchronized to CarbonData tables in batches at a specified interval for analytical query services.

#### Data Preparation

##### NOTE

Before running the sample program, set the `spark.yarn.security.credentials.hbase.enabled` configuration item to `true` in the `spark-defaults.conf` configuration file of Spark client. (The default value is `false`. Changing the value to `true` does not affect existing services.) If you want to uninstall the HBase service, change the value back to `false` first.

1. Create an HBase table and construct data with **key**, **modify\_time**, and **valid** columns. **key** of each data record is unique in the table. **modify\_time** indicates the modification time, and **valid** indicates whether the data is valid. In this example, **1** indicates that the data is valid, and **0** indicates that the data is invalid.

For example, go to HBase Shell and run the following commands:

```
create 'hbase_table','key','info'
```

```
put 'hbase_table','1','info:modify_time','2019-11-22 23:28:39'
```

```
put 'hbase_table','1','info:valid','1'
```

```
put 'hbase_table','2','info:modify_time','2019-11-22 23:28:39'
put 'hbase_table','2','info:valid','1'
put 'hbase_table','3','info:modify_time','2019-11-22 23:28:39'
put 'hbase_table','3','info:valid','0'
put 'hbase_table','4','info:modify_time','2019-11-22 23:28:39'
put 'hbase_table','4','info:valid','1'
```

 NOTE

The values of **modify\_time** in the preceding information can be set to the time earlier than the current time.

```
put 'hbase_table','5','info:modify_time','2021-03-03 15:20:39'
put 'hbase_table','5','info:valid','1'
put 'hbase_table','6','info:modify_time','2021-03-03 15:20:39'
put 'hbase_table','6','info:valid','1'
put 'hbase_table','7','info:modify_time','2021-03-03 15:20:39'
put 'hbase_table','7','info:valid','0'
put 'hbase_table','8','info:modify_time','2021-03-03 15:20:39'
put 'hbase_table','8','info:valid','1'
put 'hbase_table','4','info:valid','0'
put 'hbase_table','4','info:modify_time','2021-03-03 15:20:39'
```

 NOTE

The values of **modify\_time** in the preceding information can be set to the time within 30 minutes after the sample program is started. (30 minutes is the default synchronization interval of the sample program and can be modified.)

```
put 'hbase_table','9','info:modify_time','2021-03-03 15:32:39'
put 'hbase_table','9','info:valid','1'
put 'hbase_table','10','info:modify_time','2021-03-03 15:32:39'
put 'hbase_table','10','info:valid','1'
put 'hbase_table','11','info:modify_time','2021-03-03 15:32:39'
put 'hbase_table','11','info:valid','0'
put 'hbase_table','12','info:modify_time','2021-03-03 15:32:39'
put 'hbase_table','12','info:valid','1'
```

 NOTE

The values of **modify\_time** in the preceding information can be set to the time from 30 minutes to 60 minutes after the sample program is started, that is, the second synchronization period.

2. Run the following commands to create a Hive foreign table for HBase in SparkSQL:

```
create table external_hbase_table(key string ,modify_time STRING, valid STRING)
```

```
using org.apache.spark.sql.hbase.HBaseSource
```

```
options(hbaseTableName "hbase_table", keyCols "key", colsMapping "modify_time=info.modify_time,valid=info.valid");
```

3. Run the following command to create a CarbonData table in SparkSQL:  
**create table** *carbon01* (*key string, modify\_time STRING, valid STRING*) stored as carbondata;
4. Initialize and load all data in the current HBase table to the CarbonData table.  
**insert into table** *carbon01* select \* from *external\_hbase\_table* where *valid='1'*;
5. Run the following **spark-submit** command:  
spark-submit --master yarn --deploy-mode client --class  
com.huawei.bigdata.spark.examples.HBaseExternalHivetoCarbon /opt/example/  
HBaseExternalHivetoCarbon-1.0.jar

### 31.3.11.2 Java Example Code

The following code snippets are used as an example. For complete code, see **com.huawei.spark.examples.HBaseExternalHivetoCarbon**.

```
public static void main(String[] args) throws Exception {
 spark = SparkSession.builder().appName("HBaseExternalHiveToCarbon").getOrCreate();

 Timer timer = new Timer();
 timer.schedule(new TimerTask() {
 public void run() {
 timeEnd = timeStart + TIMEWINDOW;

 queryTimeStart = transferDateToStr(timeStart);
 queryTimeEnd = transferDateToStr(timeEnd);

 //run delete logic
 cmdsb = new StringBuilder();
 cmdsb.append("delete from ")
 .append(carbonTableName)
 .append(" where key in (select key from ")
 .append(externalHiveTableName)
 .append(" where modify_time>")
 .append(queryTimeStart)
 .append(" and modify_time<")
 .append(queryTimeEnd)
 .append(" and valid='0')");
 spark.sql(cmdsb.toString());

 //run insert logic
 cmdsb = new StringBuilder();
 cmdsb.append("insert into ")
 .append(carbonTableName)
 .append(" select * from ")
 .append(externalHiveTableName)
 .append(" where modify_time>")
 .append(queryTimeStart)
 .append(" and modify_time<")
 .append(queryTimeEnd)
 .append(" and valid='1'");
 spark.sql(cmdsb.toString());

 timeStart = timeEnd;
 }
 }, TIMEWINDOW, TIMEWINDOW);
}
```

## 31.3.12 Using Spark to Perform Basic Hudi Operations

### 31.3.12.1 Overview

#### Scenarios

This section describes how to use Spark to perform operations such as data insertion, query, update, incremental query, query at a specific time point, and data deletion on Hudi.

For details, see the sample code.

#### Packaging the Project

1. Use the Maven tool provided by IDEA to package the project and generate the JAR file. For details, see [Compiling and Running the Application](#).

##### NOTE

The Python sample code does not need to be packaged using Maven.

2. Upload the generated JAR file to any directory (for example, `/opt/example/`) on the server where the Spark client is located.

#### Running Tasks

After compiling and building the sample code, you can use the `spark-submit` command to perform the write, update, query, and delete operations in sequence.

- Run the Java sample program.

```
spark-submit --class
com.huawei.bigdata.hudi.examples.HoodieWriteClientExample /opt/
example/hudi-java-examples-1.0.jar hdfs://hacluster/tmp/example/
hoodie_java hoodie_java
```

`/opt/example/hudi-java-examples-1.0.jar` indicates the JAR file path, `hdfs://hacluster/tmp/example/hoodie_java` indicates the storage path of the Hudi table, and `hoodie_java` indicates the name of the Hudi table.

- Run the Scala sample program.

```
spark-submit --class
com.huawei.bigdata.hudi.examples.HoodieDataSourceExample /opt/
example/hudi-scala-examples-1.0.jar hdfs://hacluster/tmp/example/
hoodie_scala hoodie_scala
```

`/opt/example/hudi-scala-examples-1.0.jar` indicates the JAR file path, `hdfs://hacluster/tmp/example/hoodie_scala` indicates the storage path of the Hudi table, and `hoodie_Scala` indicates the name of the Hudi table.

- Run the Python sample program.

```
spark-submit /opt/example/HudiPythonExample.py hdfs://hacluster/tmp/
huditest/example/python hoodie_trips_cow
```

`hdfs://hacluster/tmp/huditest/example/python` indicates the storage path of the Hudi table, and `hoodie_trips_cow` indicates the name of the Hudi table.

### 31.3.12.2 Java Example Code

The following code snippets are used as an example. For complete code, see `com.huawei.bigdata.hudi.examples.HoodieWriteClientExample`.



Create a client object to operate Hudi:

```
String tablePath = args[0];
String tableName = args[1];
SparkConf sparkConf = HoodieExampleSparkUtils.defaultSparkConf("hoodie-client-example");
JavaSparkContext jsc = new JavaSparkContext(sparkConf);
// Generator of some records to be loaded in.
HoodieExampleDataGenerator<HoodieAvroPayload> dataGen = new HoodieExampleDataGenerator<>();
// initialize the table, if not done already
Path path = new Path(tablePath);
FileSystem fs = FSUtils.getFs(tablePath, jsc.hadoopConfiguration());
if (!fs.exists(path)) {
 HoodieTableMetaClient.initTableType(jsc.hadoopConfiguration(), tablePath,
 HoodieTableType.valueOf(tableType),
 tableName, HoodieAvroPayload.class.getName());
}

// Create the write client to write some records in
HoodieWriteConfig cfg = HoodieWriteConfig.newBuilder().withPath(tablePath)
 .withSchema(HoodieExampleDataGenerator.TRIP_EXAMPLE_SCHEMA).withParallelism(2, 2)
 .withDeleteParallelism(2).forTable(tableName)
 .withIndexConfig(HoodieIndexConfig.newBuilder().withIndexType(HoodieIndex.IndexType.BLOOM).build()
)
 .withCompactionConfig(HoodieCompactionConfig.newBuilder().archiveCommitsWith(20,
 30).build()).build();
SparkRDDWriteClient<HoodieAvroPayload> client = new SparkRDDWriteClient<>(new
HoodieSparkEngineContext(jsc), cfg);
```

Insert data:

```
String newCommitTime = client.startCommit();
LOG.info("Starting commit " + newCommitTime);
List<HoodieRecord<HoodieAvroPayload>> records = dataGen.generateInserts(newCommitTime, 10);
List<HoodieRecord<HoodieAvroPayload>> recordsSoFar = new ArrayList<>(records);
JavaRDD<HoodieRecord<HoodieAvroPayload>> writeRecords = jsc.parallelize(records, 1);
client.upsert(writeRecords, newCommitTime);
```

Update data:

```
newCommitTime = client.startCommit();
LOG.info("Starting commit " + newCommitTime);
List<HoodieRecord<HoodieAvroPayload>> toBeUpdated = dataGen.generateUpdates(newCommitTime, 2);
records.addAll(toBeUpdated);
recordsSoFar.addAll(toBeUpdated);
writeRecords = jsc.parallelize(records, 1);
client.upsert(writeRecords, newCommitTime);
```

Delete data:

```
newCommitTime = client.startCommit();
LOG.info("Starting commit " + newCommitTime);
// just delete half of the records
int numToDelete = recordsSoFar.size() / 2;
List<HoodieKey> toBeDeleted =
recordsSoFar.stream().map(HoodieRecord::getKey).limit(numToDelete).collect(Collectors.toList());
JavaRDD<HoodieKey> deleteRecords = jsc.parallelize(toBeDeleted, 1);
client.delete(deleteRecords, newCommitTime);
```

Compress data.

```
if (HoodieTableType.valueOf(tableType) == HoodieTableType.MERGE_ON_READ) {
 Option<String> instant = client.scheduleCompaction(Option.empty());
 JavaRDD<WriteStatus> writeStatuses = client.compact(instant.get());
 client.commitCompaction(instant.get(), writeStatuses, Option.empty());
}
```

### 31.3.12.3 Scala Example Code

The following code snippets are used as an example. For complete code, see [com.huawei.bigdata.hudi.examples.HoodieDataSourceExample](#).

### Insert data:

```
def insertData(spark: SparkSession, tablePath: String, tableName: String, dataGen:
HoodieExampleDataGenerator[HoodieAvroPayload]): Unit = {
val commitTime: String = System.currentTimeMillis().toString
val inserts = dataGen.convertToStringList(dataGen.generateInserts(commitTime, 20))
spark.sparkContext.parallelize(inserts, 2)
val df = spark.read.json(spark.sparkContext.parallelize(inserts, 1))df.write.format("org.apache.hudi").
options(getQuickstartWriteConfigs).
option(PRECOMBINE_FIELD_OPT_KEY, "ts").
option(RECORDKEY_FIELD_OPT_KEY, "uuid").
option(PARTITIONPATH_FIELD_OPT_KEY, "partitionpath").
option(TABLE_NAME, tableName).
mode(Overwrite).
save(tablePath)}
```

### Query data.

```
def queryData(spark: SparkSession, tablePath: String, tableName: String, dataGen:
HoodieExampleDataGenerator[HoodieAvroPayload]): Unit = {
val roViewDF = spark.
read.
format("org.apache.hudi").
load(tablePath + "/*/*/*/*")
roViewDF.createOrReplaceTempView("hudi_ro_table")
spark.sql("select fare, begin_lon, begin_lat, ts from hudi_ro_table where fare > 20.0").show()
// +-----+-----+-----+-----+
// | fare| begin_lon| begin_lat| ts|
// +-----+-----+-----+-----+
// |98.88075495133515|0.39556048623031603|0.17851135255091155|0.0|
// ...
spark.sql("select _hoodie_commit_time, _hoodie_record_key, _hoodie_partition_path, rider, driver, fare from
hudi_ro_table").show()
// +-----+-----+-----+-----+-----+
// |_hoodie_commit_time|_hoodie_record_key|hoodie_partition_path| rider|
// | driver| fare|
// +-----+-----+-----+-----+-----+
// | 20191231181501|31cafb9f-0196-4b1...| 2020/01/02|rider-1577787297889|
// |driver-1577787297889|98.88075495133515|
// ...
}
```

### Update data:

```
def updateData(spark: SparkSession, tablePath: String, tableName: String, dataGen:
HoodieExampleDataGenerator[HoodieAvroPayload]): Unit = {
val commitTime: String = System.currentTimeMillis().toString
val updates = dataGen.convertToStringList(dataGen.generateUpdates(commitTime, 10))
val df = spark.read.json(spark.sparkContext.parallelize(updates, 1))
df.write.format("org.apache.hudi").
options(getQuickstartWriteConfigs).
option(PRECOMBINE_FIELD_OPT_KEY, "ts").
option(RECORDKEY_FIELD_OPT_KEY, "uuid").
option(PARTITIONPATH_FIELD_OPT_KEY, "partitionpath").
option(TABLE_NAME, tableName).
mode(Append).
save(tablePath)}
```

### Incremental query:

```
def incrementalQuery(spark: SparkSession, tablePath: String, tableName: String) {
import spark.implicits._
val commits = spark.sql("select distinct(_hoodie_commit_time) as commitTime from hudi_ro_table order by
commitTime").map(k => k.getString(0)).take(50)
val beginTime = commits(commits.length - 2)

val incViewDF = spark.
read.
```

```
format("org.apache.hudi").
option(QUERY_TYPE_OPT_KEY, QUERY_TYPE_INCREMENTAL_OPT_VAL).
option(BEGIN_INSTANTTIME_OPT_KEY, beginTime).
load(tablePath)
incViewDF.createOrReplaceTempView("hudi_incr_table")
spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_incr_table where fare >
20.0").show()}
```

Query at a specific time point:

```
def pointInTimeQuery(spark: SparkSession, tablePath: String, tableName: String) {
import spark.implicits._
val commits = spark.sql("select distinct(_hoodie_commit_time) as commitTime from hudi_ro_table order by
commitTime").map(k => k.getString(0)).take(50)
val beginTime = "000"
// Represents all commits > this time.
val endTime = commits(commits.length - 2)
// commit time we are interested in
//incrementally query data
val incViewDF = spark.read.format("org.apache.hudi").
option(QUERY_TYPE_OPT_KEY, QUERY_TYPE_INCREMENTAL_OPT_VAL).
option(BEGIN_INSTANTTIME_OPT_KEY, beginTime).
option(END_INSTANTTIME_OPT_KEY, endTime).
load(tablePath)
incViewDF.createOrReplaceTempView("hudi_incr_table")
spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_incr_table where fare >
20.0").show()}
```

### 31.3.12.4 Python Sample Code

#### Using Python to Write Data to a Hudi Table

The following code snippets are used as an example. For complete code, see [sparknormal-examples.SparkOnHudiPythonExample.hudi\\_python\\_write\\_example](#).

Insert data:

```
#insert
inserts = sc._jvm.org.apache.hudi.QuickstartUtils.convertToStringList(dataGen.generateInserts(10))
df = spark.read.json(spark.sparkContext.parallelize(inserts, 2))
hudi_options = {
'hoodie.table.name': tableName,
'hoodie.datasource.write.recordkey.field': 'uuid',
'hoodie.datasource.write.partitionpath.field': 'partitionpath',
'hoodie.datasource.write.table.name': tableName,
'hoodie.datasource.write.operation': 'insert',
'hoodie.datasource.write.precombine.field': 'ts',
'hoodie.upsert.shuffle.parallelism': 2,
'hoodie.insert.shuffle.parallelism': 2
}
df.write.format("hudi"). \
options(**hudi_options). \
mode("overwrite"). \
save(basePath)
```

Query data.

```
tripsSnapshotDF = spark. \
read. \
format("hudi"). \
load(basePath + "/*/*/*")
tripsSnapshotDF.createOrReplaceTempView("hudi_trips_snapshot")
spark.sql("select fare, begin_lon, begin_lat, ts from hudi_trips_snapshot where fare > 20.0").show()
spark.sql("select _hoodie_commit_time, _hoodie_record_key, _hoodie_partition_path, rider, driver, fare from
hudi_trips_snapshot").show()
```

### Update data:

```
updates = sc._jvm.org.apache.hudi.QuickstartUtils.convertToStringList(dataGen.generateUpdates(10))
df = spark.read.json(spark.sparkContext.parallelize(updates, 2))
df.write.format("hudi"). \
 options(**hudi_options). \
 mode("append"). \
 save(basePath)
```

### Query incremental data:

```
spark. \
 read. \
 format("hudi"). \
 load(basePath + "/*/*/*/*"). \
 createOrReplaceTempView("hudi_trips_snapshot")
incremental_read_options = {
 'hoodie.datasource.query.type': 'incremental',
 'hoodie.datasource.read.begin.instanttime': beginTime,
}
tripsIncrementalDF = spark.read.format("hudi"). \
 options(**incremental_read_options). \
 load(basePath)
tripsIncrementalDF.createOrReplaceTempView("hudi_trips_incremental")
spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_trips_incremental where
fare > 20.0").show()
```

### Query data at a specific time point:

```
Represents all commits > this time.
beginTime = "000"
endTime = commits[len(commits) - 2]
point_in_time_read_options = {
 'hoodie.datasource.query.type': 'incremental',
 'hoodie.datasource.read.end.instanttime': endTime,
 'hoodie.datasource.read.begin.instanttime': beginTime
}

tripsPointInTimeDF = spark.read.format("hudi"). \
 options(**point_in_time_read_options). \
 load(basePath)

tripsPointInTimeDF.createOrReplaceTempView("hudi_trips_point_in_time")
spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_trips_point_in_time where
fare > 20.0").show()
```

### Delete data:

```
Obtain the total number of records.
spark.sql("select uuid, partitionpath from hudi_trips_snapshot").count()
Obtain two records to be deleted.
ds = spark.sql("select uuid, partitionpath from hudi_trips_snapshot").limit(2)
Delete the records.
hudi_delete_options = {
 'hoodie.table.name': tableName,
 'hoodie.datasource.write.recordkey.field': 'uuid',
 'hoodie.datasource.write.partitionpath.field': 'partitionpath',
 'hoodie.datasource.write.table.name': tableName,
 'hoodie.datasource.write.operation': 'delete',
 'hoodie.datasource.write.precombine.field': 'ts',
 'hoodie.upsert.shuffle.parallelism': 2,
 'hoodie.insert.shuffle.parallelism': 2
}
from pyspark.sql.functions import lit
deletes = list(map(lambda row: (row[0], row[1]), ds.collect()))
df = spark.sparkContext.parallelize(deletes).toDF(['uuid', 'partitionpath']).withColumn('ts', lit(0.0))
df.write.format("hudi"). \
 options(**hudi_delete_options). \
 mode("append"). \
 save(basePath)
```

```
Perform the query in the same way.
roAfterDeleteViewDF = spark. \
 read. \
 format("hudi"). \
 load(basePath + "/*/*/*")
roAfterDeleteViewDF.registerTempTable("hudi_trips_snapshot")
Return (total - 2) records.
spark.sql("select uuid, partitionpath from hudi_trips_snapshot").count()
spark.sql("select uuid, partitionpath from hudi_trips_snapshot").show()
```

## 31.3.13 Compiling User-defined Configuration Items for Hudi

### 31.3.13.1 HoodieDeltaStreamer

Compile a user-defined conversion class for **Transformer**.

Compile a user-defined schema for **SchemaProvider**.

Add the following parameters when running **HoodieDeltaStreamer**:

```
--schemaprovider-class Defined schema class --transformer-class Defined transform class
```

Examples of **Transformer** and **SchemaProvider**:

```
public class TransformerExample implements Transformer, Serializable {
 @Override
 public Dataset<Row> apply(JavaSparkContext jsc, SparkSession sparkSession, Dataset<Row> rowDataset,
 TypedProperties properties) {
 JavaRDD<Row> rowJavaRdd = rowDataset.toJavaRDD();
 List<Row> rowList = new ArrayList<>();
 for(Row row: rowJavaRdd.collect()){
 rowList.add(buildRow(row));
 }
 JavaRDD<Row> stringJavaRdd = jsc.parallelize(rowList);
 List<StructField> fields = new ArrayList<>();
 builFields(fields);
 StructType schema = DataTypes.createStructType(fields);
 Dataset<Row> dataFrame = sparkSession.createDataFrame(stringJavaRdd, schema);
 return dataFrame;
 }

 private void builFields(List<StructField> fields) {
 fields.add(DataTypes.createStructField("age", DataTypes.StringType, true));
 fields.add(DataTypes.createStructField("id", DataTypes.StringType, true));
 fields.add(DataTypes.createStructField("name", DataTypes.StringType, true));
 fields.add(DataTypes.createStructField("job", DataTypes.StringType, true));
 }

 private Row buildRow(Row row){
 String age = row.getString(0);
 String id = row.getString(1);
 String job = row.getString(2);
 String name = row.getString(3);
 Row returnRow = RowFactory.create(age, id, job, name);
 return returnRow;
 }
}

public class DataSchemaProviderExample extends SchemaProvider {

 public DataSchemaProviderExample(TypedProperties props, JavaSparkContext jssc) {
 super(props, jssc);
 }

 @Override
 public Schema getSourceSchema() {
 Schema avroSchema = new Schema.Parser().parse(
```

```
 "{ \"type\": \"record\", \"name\": \"hoodie_source\", \"fields\": [{ \"name\": \"age\", \"type\": \"string\" },\n { \"name\": \"id\", \"type\": \"string\" }, { \"name\": \"job\", \"type\": \"string\" }, { \"name\": \"name\", \"type\":\n \"string\" }] }";\n return avroSchema;\n }\n\n @Override\n public Schema getTargetSchema() {\n Schema avroSchema = new Schema.Parser().parse(\n "{ \"type\": \"record\", \"name\": \"mytest_record\", \"namespace\": \"hoodie.mytest\", \"fields\":\n [{ \"name\": \"age\", \"type\": \"string\" }, { \"name\": \"id\", \"type\": \"string\" }, { \"name\": \"job\", \"type\": \"string\n \" }, { \"name\": \"name\", \"type\": \"string\" }] }";\n return avroSchema;\n }\n}
```

### 31.3.13.2 User-defined Partitioner

Compile a user-defined partitioner class that inherits **BulkInsertPartitioner** and add the following configuration when writing data to Hudi:

```
.option(BULKINSERT_USER_DEFINED_PARTITIONER_CLASS, <User-defined partitioner class package name + class name>)
```

Example of the user-defined partitioner:

```
public class HoodieSortExample<T> extends HoodieRecordPayload<T>\n implements BulkInsertPartitioner<JavaRDD<HoodieRecord<T>>> {\n @Override\n public JavaRDD<HoodieRecord<T>> repartitionRecords(JavaRDD<HoodieRecord<T>> records, int\n outputSparkPartitions) {\n JavaPairRDD<String,\n HoodieRecord<T>> stringHoodieRecordJavaPairRDD = records.coalesce(outputSparkPartitions)\n .mapToPair(record -> new Tuple2<>(new StringBuilder().append(record.getPartitionPath())\n .append("+")\n .append(record.getRecordKey())\n .toString(), record));\n JavaRDD<HoodieRecord<T>> hoodieRecordJavaRDD =\n stringHoodieRecordJavaPairRDD.mapPartitions(partition -> {\n List<Tuple2<String, HoodieRecord<T>>> recordList = new ArrayList<>();\n for (; partition.hasNext();) {\n recordList.add(partition.next());\n }\n Collections.sort(recordList, (o1, o2) -> {\n if (o1._1().split("+")[0] == o2._1().split("+")[0]) {\n return Integer.parseInt(o1._1().split("+")[1]) - Integer.parseInt(o2._1().split("+")[1]);\n } else {\n return o1._1().split("+")[0].compareTo(o2._1().split("+")[0]);\n }\n });\n return recordList.stream().map(e -> e._2).iterator();\n });\n return hoodieRecordJavaRDD;\n }\n\n @Override\n public boolean arePartitionRecordsSorted() {\n return true;\n }\n}
```

## 31.4 Commissioning the Application

## 31.4.1 Commissioning Applications on Windows

### 31.4.1.1 Spark Access Configuration on Windows Using EIPs

#### Scenario

This section describes how to bind Elastic IP addresses (EIPs) to a cluster and configure Spark files so that sample files can be compiled locally.

This section uses SparkScalaExample as an example.

#### Procedure

**Step 1** Apply for an EIP for each node in the cluster and add public IP addresses and corresponding host domain names of all nodes to the Windows local **hosts** file. (If a host name contains uppercase letters, change them to lowercase letters.)

1. On the VPC console, apply for EIPs (the number of EIPs you buy should be equal to the number of nodes in the cluster), click the name of each node in the MRS cluster, and bind an EIP to each node on the **EIPs** page.

For details, see **Virtual Private Cloud > User Guide > EIP > Assigning an EIP and Binding It to an ECS**.

2. Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.

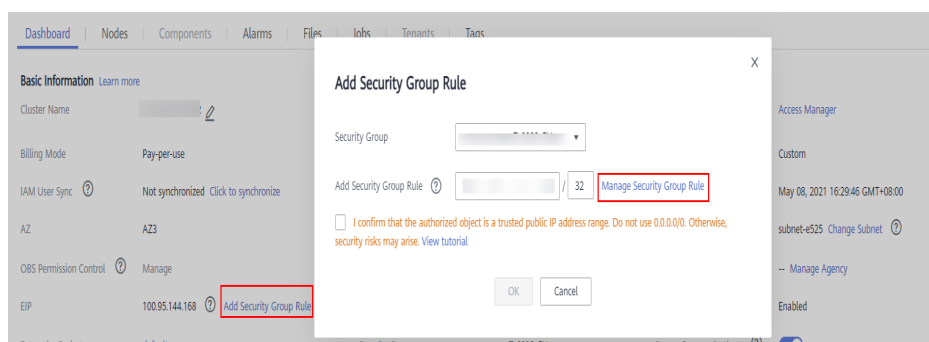
```

1 Mapping between public IP addresses and private IP addresses
2 100.95.144.168 172.16.0.120
3 100.95.144.168 172.16.0.42
4 100.95.144.168 172.16.0.62
5 100.95.144.168 172.16.0.200
6 100.95.144.168 172.16.0.139
7 100.95.144.168 172.16.0.214
8
9 hosts file in the cluster
10 172.16.0.120 node-group-1XZI0002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
11 172.16.0.42 node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com.
12 172.16.0.62 node-group-1XZI0003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
13 172.16.0.200 node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
14 172.16.0.139 node-group-1XZI0001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
15 172.16.0.214 node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.
16
17 hosts file that should be added to Windows
18 100.95.144.168 node-group-1xzi0002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
19 100.95.144.168 node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3Vint.ead64699-185a-4290-bbef-1a07e2f0459b.com.
20 100.95.144.168 node-group-1xzi0003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
21 100.95.144.168 node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
22 100.95.144.168 node-group-1xzi0001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZI0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
23 100.95.144.168 node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVnu.ead64699-185a-4290-bbef-1a07e2f0459b.com.

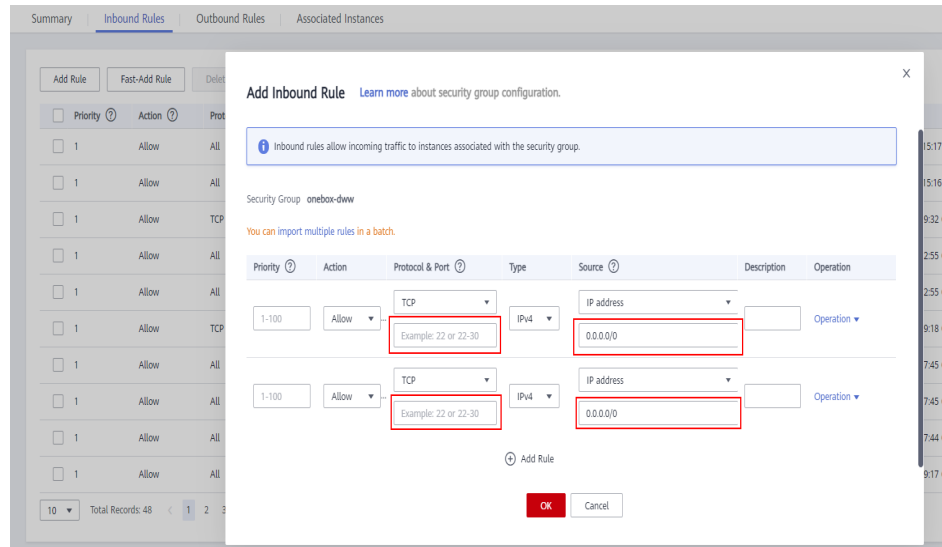
```

**Step 2** Configure security group rules for the cluster.

1. On the **Dashboard** page, choose **Add Security Group Rule > Manage Security Group Rule**.



2. On the **Inbound Rules** tab page, click **Add Rule**. In the **Add Inbound Rule** dialog box, configure the Windows IP addresses and ports 8020 and 9866.



- 3 On Manager, choose **Cluster > Services > HDFS > More > Download Client**, and copy the **core-site.xml** and **hdfs-site.xml** files on the client to the **conf** directory of the sample project.

Add the following content to the **hdfs-site.xml** file:

```
<property>
 <name>dfs.client.use.datanode.hostname</name>
 <value>>true</value>
</property>
```

Add the following content to the **pom.xml** file:

```
<dependency>
 <groupId>com.huawei.mrs</groupId>
 <artifactId>hadoop-plugins</artifactId>
 <version>Component package version-302002</version>
</dependency>
```

- 4 Before running the sample code, add **.master("local").config("spark.driver.host", "localhost")** to **SparkSession** to set the local running mode for Spart.

```
7 ▶ Object FemaleInfoCollection {
8 ▶ def main (args: Array[String]) {
9 // if (args.length < 1) {
10 // System.err.println("Usage: CollectFemaleInfo <file>")
11 // System.exit(1)
12 // }
13
14 // Configure the Spark application name.
15 val spark = SparkSession
16 .builder()
17 .appName(name = "CollectFemaleInfo")
18 .master("master:local")
19 .config("spark.driver.host", "localhost")
20 .getOrCreate()
21 // Initializing Spark
22
23
24
25 // Read data. This code indicates the data path that the input parameter args(0) specifies.
26 val text = spark.sparkContext.textFile(path = "/tmp/sparkText/")
27 // Filter the data information about the time that female netizens spend online.
28 val data = text.filter(_.contains("female"))
```

----End



## 31.4.1.2 Compiling and Running Applications

### Scenario

You can run applications in the Windows environment after application code development is complete. The procedures for running applications developed using Scala or Java are the same on IDEA.

#### NOTE

- In the Windows environment, only the sample code for accessing Spark SQL using JDBC is provided.
- Ensure that the Maven image repository of the SDK in the Huawei image site has been configured for Maven. For details, see [Configuring Huawei Open-Source Mirrors](#).

### Procedure

#### Step 1 Obtain the sample code.

Download the Maven project source code and configuration file of the sample project. For details, see [Obtaining Sample Projects](#).

Import the sample code to IDEA.

#### Step 2 Obtain configuration files.

Obtain the files from the cluster client. Download the **hive-site.xml** and **spark-defaults.conf** files from **\$SPARK\_HOME/conf** to a local directory.

#### Step 3 Upload data to HDFS.

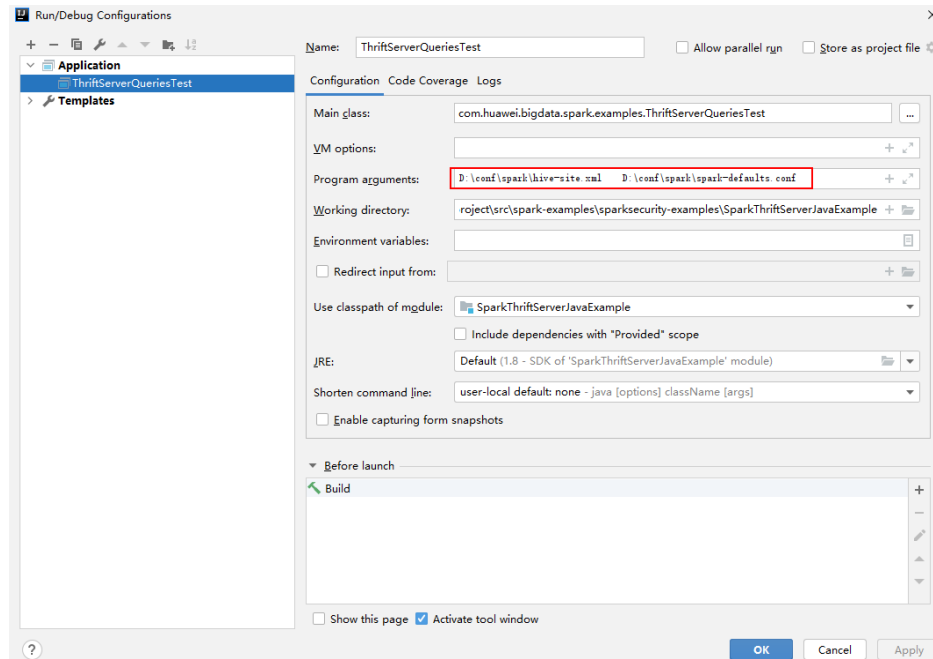
1. Create a data text file on Linux and save the following data to the data file:  
Miranda,32  
Karlle,23  
Candice,27
2. On the HDFS client running the Linux OS, run the **hadoop fs -mkdir /data** command (or the **hdfs dfs** command) to create a directory.
3. On the HDFS client running the Linux OS, run the **hadoop fs -put data /data** command to upload the data file.

#### Step 4 Configure related parameters in the sample code.

Change the SQL statement for loading data to **LOAD DATA INPATH 'hdfs:/data/data' INTO TABLE CHILD**.

```
ArrayList<String> sqlList = new ArrayList<>();
sqlList.add("CREATE TABLE IF NOT EXISTS CHILD (NAME STRING, AGE INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY"
 + " ','");
sqlList.add("LOAD DATA INPATH 'hdfs:/data/data' INTO TABLE CHILD");
sqlList.add("SELECT * FROM child");
sqlList.add("DROP TABLE child");
executeSql(url, sqlList);
```

#### Step 5 Add running parameters to the **hive-site.xml** and **spark-defaults.conf** files when the application is running.



**Step 6** Run the application.

----End

### 31.4.1.3 View Debugging Results

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/D:/mavenlocal/org/apache/logging/log4j/log4j-slf4j-impl/2.6.2/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/D:/mavenlocal/org/slf4j/slf4j-log4j12/1.7.30/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to the console.
---- Begin executing sql: CREATE TABLE IF NOT EXISTS CHILD (NAME STRING, AGE INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ----
Result
---- Done executing sql: CREATE TABLE IF NOT EXISTS CHILD (NAME STRING, AGE INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ----
---- Begin executing sql: LOAD DATA INPATH 'hdfs:/data/data' INTO TABLE CHILD ----
Result
---- Done executing sql: LOAD DATA INPATH 'hdfs:/data/data' INTO TABLE CHILD ----
---- Begin executing sql: SELECT * FROM child ----
NAME AGE
Miranda 32
Karlie 23
Candice 27
---- Done executing sql: SELECT * FROM child ----
---- Begin executing sql: DROP TABLE child ----
Result
---- Done executing sql: DROP TABLE child ----

Process finished with exit code 0
```

## 31.4.2 Commissioning an Application in Linux

### 31.4.2.1 Compiling and Running the Application

#### Scenario

After the program codes are developed, you can upload the codes to the Linux client for running. The running procedures of applications developed in Scala or Java are the same.

#### NOTE

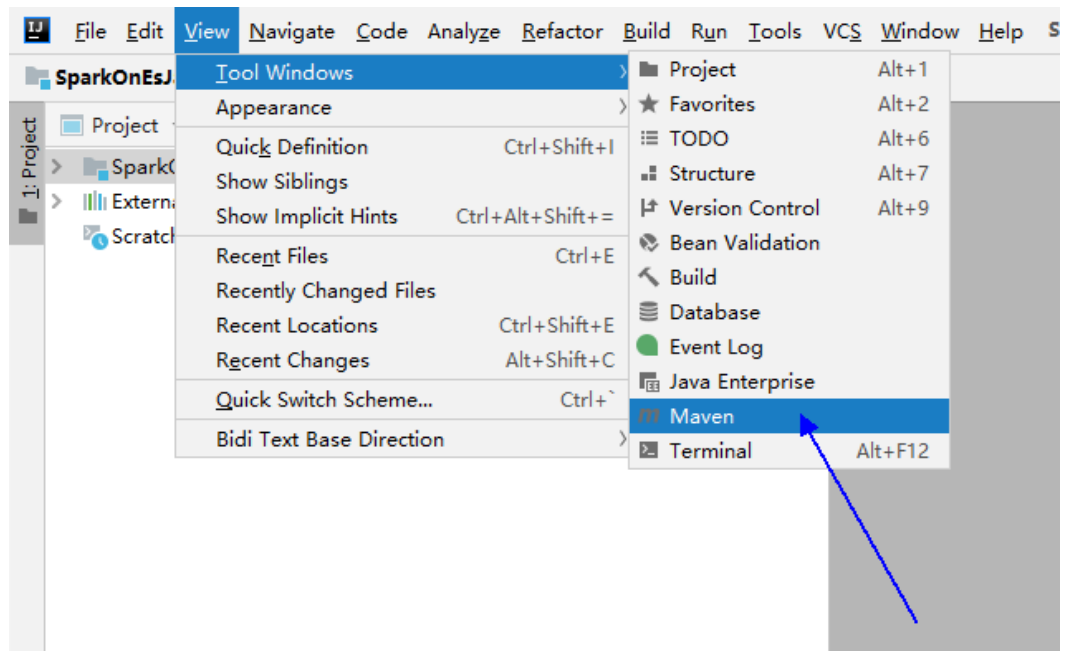
- The Spark application developed in Python does not need to build Artifacts as a jar. You just need to copy the sample projects to the compiler.
- It is needed to ensure that the version of Python installed on the worker and driver is consistent, otherwise the following error will be reported: "Python in worker has different version %s than that in driver %s."
- Ensure that Maven image repository of the SDK in the Huawei image site has been configured in Maven. For details, see [Configuring Huawei Open-Source Mirrors](#).

#### Procedure

**Step 1** In the IntelliJ IDEA, open the Maven tool window.

On the main page of the IDEA, choose **View > Tool Windows > Maven** to open the Maven tool window.

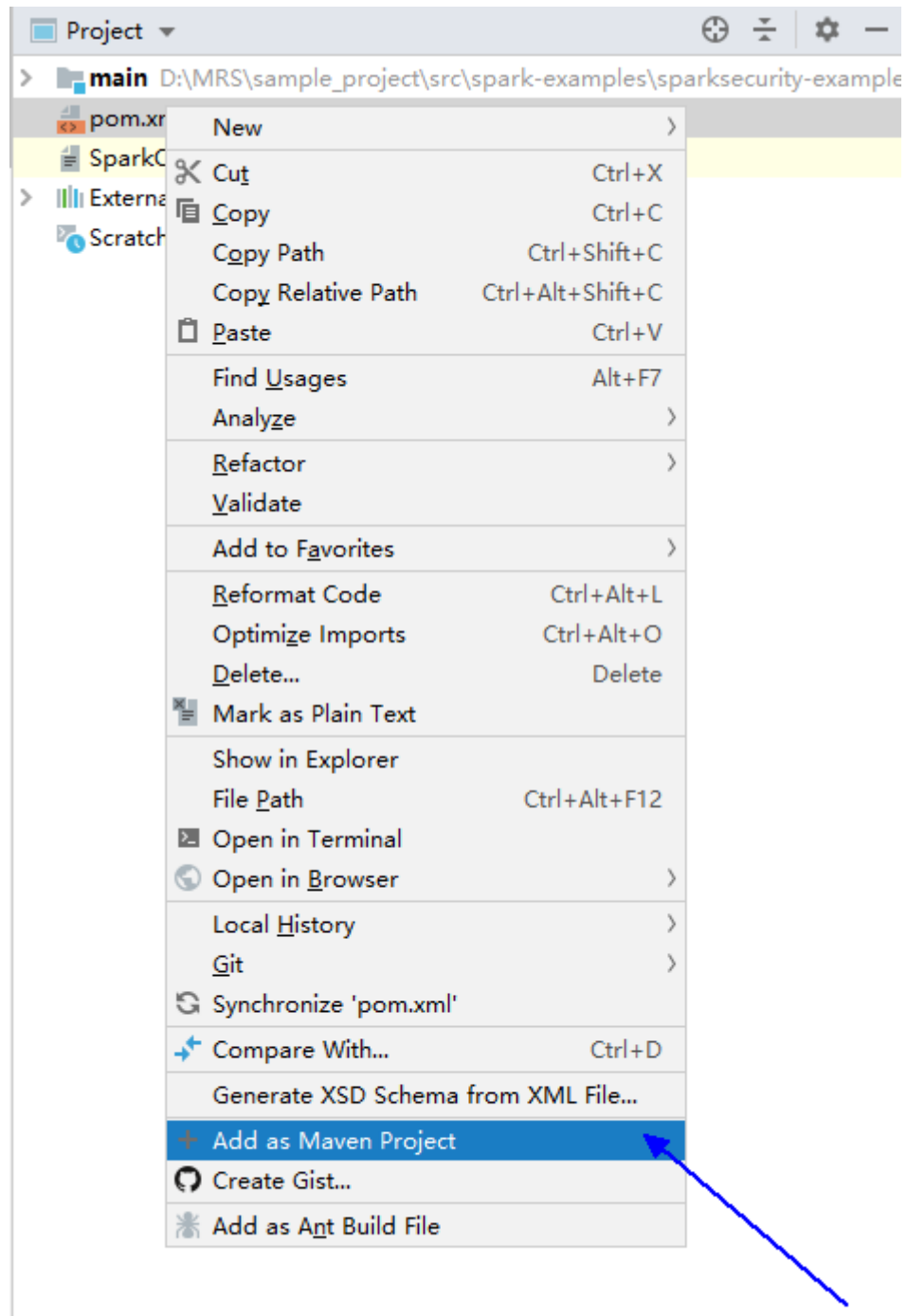
Figure 31-35 Opening the Maven tool window



If the project is not imported using Maven, perform the following operations:

Right-click the **pom** file in the sample code project and choose **Add as Maven Project** from the shortcut menu to add a Maven project.

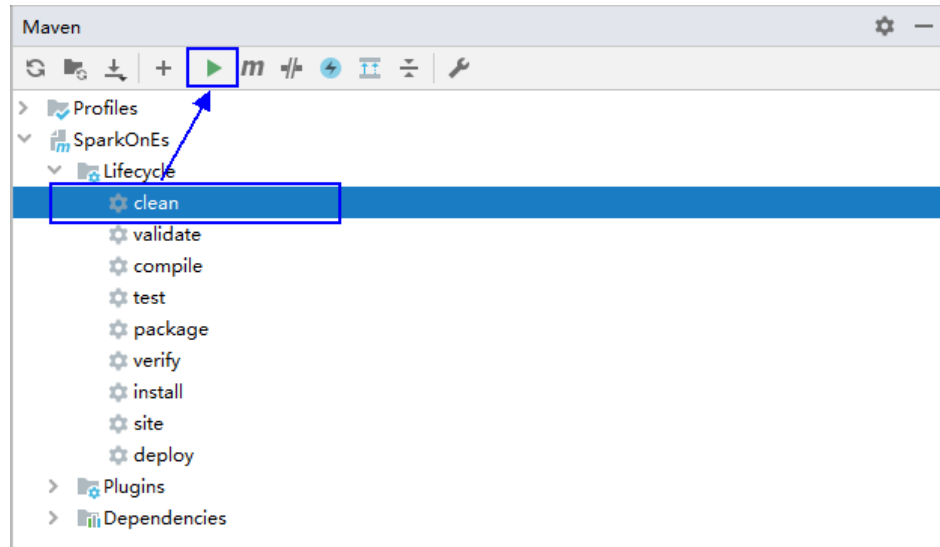
Figure 31-36 Adding a Maven project



**Step 2** Use Maven to generate a JAR file.

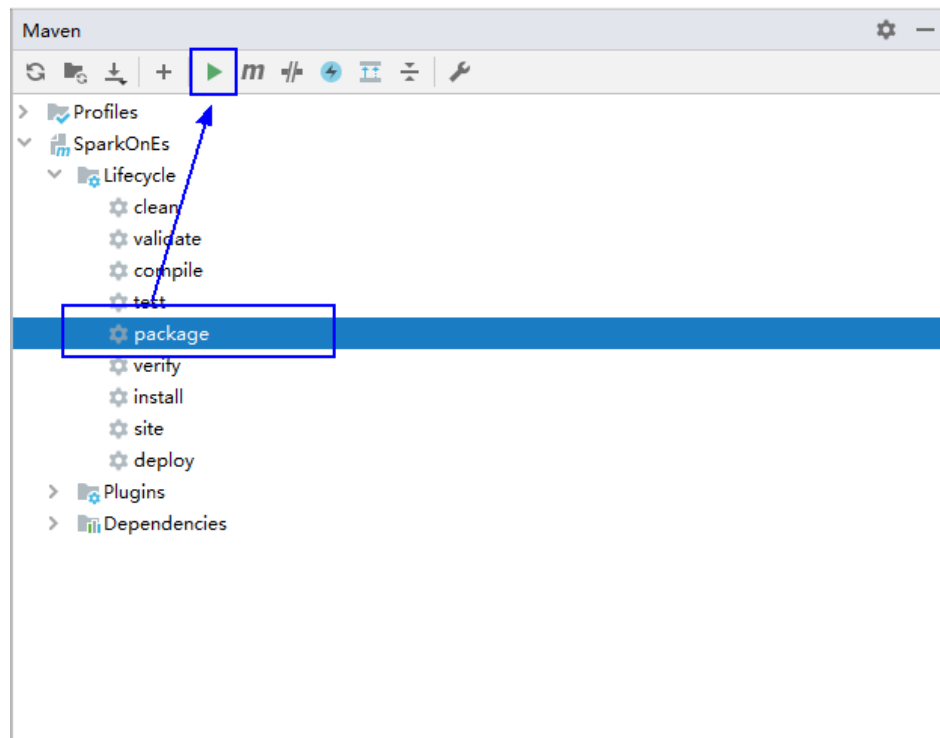
1. In the Maven tool window, select **clean** from **Lifecycle** to execute the Maven building process.

**Figure 31-37** Selecting **clean** from **Lifecycle** and execute the Maven building process



2. In the Maven tool window, select **package** from **Lifecycle** and execute the Maven building process.

**Figure 31-38** Selecting **package** from **Lifecycle** and execute the Maven build process.



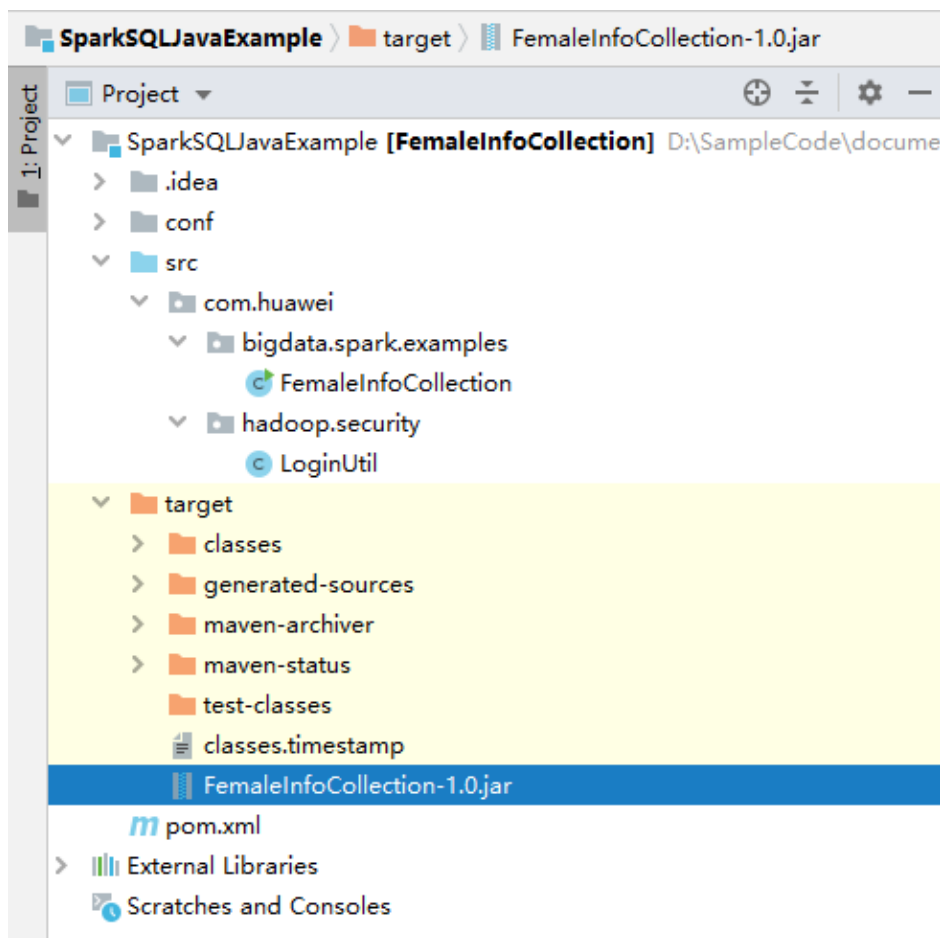
If the following information is displayed in **Run:**, the packaging is successful.

**Figure 31-39** Packaging success message

```
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ FemaleInfoCollection ---
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ FemaleInfoCollection ---
[INFO] Building jar: D:\SampleCode\document\VT\code\sparksecurity-examples\SparkSQLJavaExample\target\FemaleInfoCollection-1.0.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 19.427 s
[INFO] Finished at: 2020-09-21T11:17:31+08:00
[INFO] -----
```

- 3. You can obtain the JAR package from the target folder in the project directory.

**Figure 31-40** Obtaining the JAR Package



**Step 3** Copy the JAR file generated in **Step 2** (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. Run the Spark application. For details about the example application, see [Developing the Project](#).

**CAUTION**

Do not restart the HDFS service or all DataNode instances during Spark job running. Otherwise, the job may fail and some JobHistory data may be lost.

----End

## 31.4.2.2 Checking the Commissioning Result

### Scenario

After a Spark application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Logging in to the Spark web UI.
- Viewing Spark logs.

### Procedure

- **Check the operating result data of the Spark application.**

The data storage directory and format are specified by users in the Spark application. You can obtain the data in the specified file.

- **Check the status of the Spark application.**

The Spark contains the following two web UIs:

- The Spark UI displays the status of applications being executed.

The Spark UI contains the **Spark Jobs**, **Spark Stages**, **Storage**, **Environment**, and **Executors** parts. Besides these parts, **Streaming** is displayed for the Streaming application.

On the YARN web UI, find the Spark application. Click **ApplicationMaster** in the last column of the application information. The Spark UI page is displayed.

- The History Server UI displays the status of all Spark applications.

The History Server UI displays information such as the application ID, application name, start time, end time, execution time, and user to whom the application belongs. After the application ID is clicked, the Spark UI of the application is displayed.

- **View Spark logs to learn application running conditions.**

The logs of Spark offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 31.5 More Information

### 31.5.1 Common APIs

#### 31.5.1.1 Java

To avoid API compatibility or reliability issues after updates to the open-source Spark, it is advisable to use APIs of the version you are currently using.

### Spark Core Common Interfaces

Spark mainly uses the following classes:

- **JavaSparkContext**: external interface of Spark, which is used to provide the functions of Spark for Java applications that invoke this class, for example, connecting Spark clusters and generating RDDs, accumulations, and broadcasts. It functions as a container.
- **SparkConf**: Spark application configuration class, which is used to configure the application name, execution model, and executor memory.
- **JavaRDD**: class used to define the JavaRDD in the Java application, which functions like the RDD (Resilient Distributed Dataset) class of Scala.
- **JavaPairRDD**: indicates the JavaRDD in the key-value format. This class provides methods such as `groupByKey` and `reduceByKey`.
- **Broadcast**: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- **StorageLevel**: data storage levels, including memory (`MEMORY_ONLY`), disk (`DISK_ONLY`), and memory+disk (`MEMORY_AND_DISK`).

The JavaRDD supports two types of operations, transformation and action. [Table 31-7](#) and [Table 31-8](#) show the common methods.

**Table 31-7** Transformation

Method	Description
<code>&lt;R&gt; JavaRDD&lt;R&gt; map(Function&lt;T,R&gt; f)</code>	Returns a new RDD by applying a function to all elements of this RDD.
<code>JavaRDD&lt;T&gt; filter(Function&lt;T,Boolean&gt; f)</code>	Invokes Function on all elements of the RDD and returns the element that is <b>true</b> .
<code>&lt;U&gt; JavaRDD&lt;U&gt; flatMap(FlatMapFunction&lt;T,U&gt; f)</code>	Returns a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
<code>JavaRDD&lt;T&gt; sample(boolean withReplacement, double fraction, long seed)</code>	Returns a sampled subset of this RDD.
<code>JavaRDD&lt;T&gt; distinct(int numPartitions)</code>	Returns a new RDD containing the distinct elements in this RDD.
<code>JavaPairRDD&lt;K,Iterable&lt;V&gt;&gt; groupByKey(int numPartitions)</code>	Returns <code>(K,Seq[V])</code> and combines the values of the same key to a set.
<code>JavaPairRDD&lt;K,V&gt; reduceByKey(Function2&lt;V,V,V&gt; func, int numPartitions)</code>	Invokes Function on the values of the same key.



Method	Description
JavaPairRDD<K,V> sortByKey(boolean ascending, int numPartitions)	Sorts by key in ascending or descending order. Ascending is of the boolean type.
JavaPairRDD<K,scala.Tuple2<V,W>> join(JavaPairRDD<K,W> other)	Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. <b>numTasks</b> indicates the number of concurrent tasks.
JavaPairRDD<K,scala.Tuple2<Iterable<V>,Iterable<W>>> cogroup(JavaPairRDD<K,W> other, int numPartitions)	Returns the dataset of <K,scala.Tuple2<Iterable<V>,Iterable<W>>> when the (K,V) and (K,W) datasets exist. <b>numTasks</b> indicates the number of concurrent tasks.
JavaPairRDD<T,U> cartesian(JavaRDDLike<U,?> other)	Returns the Cartesian product of the RDD and other RDDs.

**Table 31-8** Action

Method	Description
T reduce(Function2<T,T,T> f)	Invokes Function2 on elements of the RDD.
java.util.List<T> collect()	Returns an array that contains all of the elements in this RDD.
long count()	Returns the number of elements in the RDD.
T first()	Returns the first element in this RDD.
java.util.List<T> take(int num)	Returns the first <i>n</i> elements in this RDD.
java.util.List<T> takeSample(boolean withReplacement, int num, long seed)	Samples the dataset randomly and returns a dataset of num elements. <b>withReplacement</b> indicates whether replacement is used.
void saveAsTextFile(String path, Class<? extends org.apache.hadoop.io.compress.CompressionCodec> codec)	Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file.

Method	Description
<code>java.util.Map&lt;K, Object&gt; countByKey()</code>	Counts the appearance times of each key.
<code>void foreach(VoidFunction &lt;T&gt; f)</code>	Applies a function f to all elements of this RDD.
<code>java.util.Map&lt;T, Long&gt; countByValue()</code>	Counts the times that each element of the RDD occurs.

**Table 31-9** New APIs of Spark core

API	Description
<code>public java.util.concurrent.atomic.AtomicBoolean isSparkContextDown()</code>	<p>Determines whether sparkContext is shut down completely. The initial value is <b>false</b>.</p> <p>The value <b>true</b> indicates that sparkContext is shut down completely.</p> <p>The value <b>false</b> indicates that sparkContext is not shut down.</p> <p>For example, <code>jsc.sc().isSparkContextDown().get() == true</code> indicates that sparkContext is shut down completely.</p>

## Spark Streaming Common Interfaces

Spark Streaming mainly uses the following classes:

- `JavaStreamingContext`: main entrance of the Spark Streaming, which is used to provide methods for creating the DStream. Intervals by batch need to be set in the input parameter.
- `JavaDStream`: a type of data which indicates the RDDs continuous sequence. It indicates the continuous data flow.
- `JavaPairDStream`: interface of KV DStream, which is used to provide the `reduceByKey` and `join` operations.
- `JavaReceiverInputDStream<T>`: specifies any inflow accepting data from the network.

Common methods of Spark Streaming are the same as those of Spark Core. The following table describes some special Spark Streaming methods.

**Table 31-10** Spark Streaming methods

Method	Description
JavaReceiverInputD-Stream<java.lang.String> socketStream(java.lang.String hostname,int port)	Creates an inflow to receive data from the corresponding hostname and port through the TCP socket. Received data is resolved to the UTF8 format. The default storage level is the Memory+Disk.
JavaDStream<java.lang.String> textFileStream(java.lang.String directory)	Creates an inflow to detect new files compatible with the Hadoop file system, and read it as a text file. The directory of the input parameter is an HDFS directory.
void start()	Starts the Spark Streaming computing.
void awaitTermination()	Terminates the await of the process, which is similar to pressing <b>Ctrl+C</b> .
void stop()	Stops the Spark Streaming computing.
<T> JavaDStream<T> transform(java.util.List<JavaDStream<?>> dstreams,Function2<java.util.List<JavaRDD<?>>,Time,JavaRDD<T>> transformFunc)	Performs the Function operation on each RDD to obtain a new DStream. In this function, the sequence of the JavaRDDs must be the same as the corresponding DStreams.
<T> JavaDStream<T> union(JavaDStream<T> > first,java.util.List<JavaDStream<T>> rest)	Creates a unified Dstream from multiple DStreams with the same type and sliding window.

**Table 31-11** Spark Streaming enhancement interface

Method	Description
JAVADStreamKafkaWriter.writeToKafka()	Writes data from the DStream into Kafka in batch.
JAVADStreamKafkaWriter.writeToKafkaBySingle()	Writes data from DStream into Kafka one by one.

## Spark SQL Common Interfaces

Spark SQL mainly uses the following classes:

- SQLContext: main entrance of the Spark SQL function and DataFrame.

- DataFrame: a distributed dataset organized by naming columns.
- DataFrameReader: interface for loading the DataFrame from external storage systems.
- DataFrameStatFunctions: implementation the statistic function of the DataFrame.
- UserDefinedFunction: function defined by users.

Common Actions methods are described in the following table.

**Table 31-12** Spark SQL methods

Method	Description
Row[] collect()	Returns an array containing all DataFrame columns.
long count()	Returns the number of DataFrame rows.
DataFrame describe(java.lang.String... cols)	Counts the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
Row first()	Returns the first row.
Row[] head(int n)	Returns the first <i>n</i> rows.
void show()	Displays the first 20 rows in table.
Row[] take(int n)	Returns the first <i>n</i> rows in the DataFrame.

**Table 31-13** Basic DataFrame Functions

Method	Description
void explain(boolean extended)	Prints the logical plan and physical plan of the SQL.
void printSchema()	Prints the schema information to the console.
registerTempTable	Registers the DataFrame as a temporary table, whose period is bound to the SQLContext.
DataFrame toDF(java.lang.String... colNames)	Returns a DataFrame whose columns are renamed.
DataFrame sort(java.lang.String sortCol,java.lang.String... sortCols)	Sorts columns in ascending or descending orders based on different columns.
GroupedData rollup(Column... cols)	Performs multiple-dimension crankback on the specified columns in the DataFrame.

### 31.5.1.2 Scala

To avoid API compatibility or reliability issues after updates to the open-source Spark, it is advisable to use APIs of the version you are currently using. For details about APIs, see [conref:text]Target does not exist.

## Spark Core Common Interfaces

Spark mainly uses the following classes:

- **SparkContext**: external interface of Spark, which is used to provide the functions of Spark for Scala applications that invoke this class, for example, connecting Spark clusters and generating RDDs.
- **SparkConf**: Spark application configuration class, which is used to configure the application name, execution model, and executor memory.
- **Resilient Distributed Dataset (RDD)**: defines the RDD class in the Spark application. The class provides the data collection operation methods, such as map and filter.
- **PairRDDFunctions**: provides computation operations for the RDD data of the key-value pair, such as groupByKey.
- **Broadcast**: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- **StorageLevel**: data storage levels, including memory (MEMORY\_ONLY), disk (DISK\_ONLY), and memory+disk (MEMORY\_AND\_DISK).

RDD supports two types of operations, transformation and action. [Table 31-14](#) and [Table 31-15](#) describe the common methods.

**Table 31-14** Transformation

Method	Description
map[U](f: (T => U): RDD[U]	Returns a new RDD by applying a function to all elements of this RDD.
filter(f: (T => Boolean): RDD[T]	Invokes the f method for all RDD elements to generate a satisfied data set that is returned in the form of RDD.
flatMap[U](f: (T => TraversableOnce[U]) (implicit arg0: ClassTag[U]): RDD[U]	Returns a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
sample(withReplacement: Boolean, fraction: Double, seed: Long = Utils.random.nextLong): RDD[T]	Returns a sampled subset of this RDD.
union(other: RDD[T]): RDD[T]	Returns a new RDD, contains source RDD and the group of RDD's elements.
distinct([numPartitions: Int]): RDD[T]	Returns a new RDD containing the distinct elements in this RDD.

Method	Description
groupByKey(): RDD[(K, Iterable[V])]	Returns (K,Iterable[V]) and combines the values of the same key to a set.
reduceByKey(func: (V, V) => V[, numPartitions: Int]): RDD[(K, V)]	Invokes func on the values of the same key.
sortByKey(ascending: Boolean = true, numPartitions: Int = self.partitions.length): RDD[(K, V)]	Sorts by key in ascending or descending order. Ascending is of the boolean type.
join[W](other: RDD[(K, W)][, numPartitions: Int]): RDD[(K, (V, W))]	Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. <b>numPartitions</b> indicates the number of concurrent tasks.
cogroup[W](other: RDD[(K, W)], numPartitions: Int): RDD[(K, (Iterable[V], Iterable[W]))]	Returns the dataset of (K, (Iterable[V], Iterable[W])) when the (K,V) and (K,W) datasets of two key-value pairs exist. <b>numPartitions</b> indicates the number of concurrent tasks.
cartesian[U](other: RDD[U])(implicit arg0: ClassTag[U]): RDD[(T, U)]	Returns the Cartesian product of the RDD and other RDDs.

**Table 31-15** Action

API	Description
reduce(f: (T, T) => T):	Invokes f on elements of the RDD.
collect(): Array[T]	Returns an array that contains all of the elements in this RDD.
count(): Long	Returns the number of elements in the dataset.
first(): T	Returns the first element in this RDD.
take(num: Int): Array[T]	Returns the first <i>n</i> elements in this RDD.
takeSample(withReplacement: Boolean, num: Int, seed: Long = Utils.random.nextLong): Array[T]	Samples the dataset randomly and returns a dataset of num elements. <b>withReplacement</b> indicates whether replacement is used.

API	Description
saveAsTextFile(path: String): Unit	Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file.
saveAsSequenceFile(path: String, codec: Option[Class[_ <: CompressionCodec]] = None): Unit	This API can be used only on the key-value pair, and then it generates SequenceFile and writes the file to the local or Hadoop file system.
countByKey(): Map[K, Long]	Counts the appearance times of each key.
foreach(func: (T) => Unit): Unit	Applies a function f to all elements of this RDD.
countByValue()(implicit ord: Ordering[T] = null): Map[T, Long]	Counts the times that each element of the RDD occurs.

**Table 31-16** New APIs of Spark core

API	Description
isSparkContextDown: AtomicBoolean	<p>Determines whether sparkContext is shut down completely. The initial value is <b>false</b>.</p> <p>The value <b>true</b> indicates that sparkContext is shut down completely.</p> <p>The value <b>false</b> indicates that sparkContext is not shut down.</p> <p>For example, <b>sc.isSparkContextDown.get() == true</b> indicates that sparkContext is shut down completely.</p>

## Spark Streaming Common Interfaces

Spark Streaming mainly uses the following classes:

- StreamingContext: main entrance of the Spark Streaming, which is used to provide methods for creating the DStream. Intervals by batch need to be set in the input parameter.
- dstream.DStream: a type of data which indicates the RDDs continuous sequence. It indicates the continuous data flow.
- dstream.PariDStreamFunctions: the Dstream of key-value, common operations are groupByKey and reduceByKey.

The cooperated Java APIs of Spark Streaming are `JavaStreamingContext`, `JavaDStream`, `JavaPairDStream`.

Common methods of Spark Streaming are the same as those of Spark Core. The following table describes some special Spark Streaming methods.

**Table 31-17** Spark Streaming methods

Method	Description
<code>socketTextStream(hostname: String, port: Int, storageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK_SER_2): ReceiverInputDStream[String]</code>	Creates an input stream from the TCP source host:port.
<code>start():Unit</code>	Starts Spark Streaming computing.
<code>awaitTermination(timeout: long):Unit</code>	Terminates the await of the process, which is similar to pressing <b>Ctrl+C</b> .
<code>stop(stopSparkContext: Boolean, stopGracefully: Boolean): Unit</code>	Stops the Spark Streaming computing.
<code>transform[T](dstreams: Seq[DStream[_]], transformFunc: (Seq[RDD[_]], Time) ? RDD[T])(implicit arg0: ClassTag[T]): DStream[T]</code>	Performs the function operation on each RDD to obtain a new DStream.
<code>updateStateByKey(func)</code>	Updates the status of DStream. To use this method, you need to define the state and state update functions.
<code>window(windowLength, slideInterval)</code>	Generates a new DStream by batch calculating according to the window of the source DStream.
<code>countByWindow(windowLength, slideInterval)</code>	Returns the number of sliding window elements in the stream.
<code>reduceByWindow(func, windowLength, slideInterval)</code>	When the key-value pair of DStream is invoked, a new key-value pair of DStream is returned. The value of each key is obtained by aggregating the reduce function in batches in the sliding window.
<code>join(otherStream, [numTasks])</code>	Performs a join operation between different Spark Streamings.



Method	Description
DStreamKafkaWriter.writeToKafka()	Writes data from the DStream into Kafka in batch.
DStreamKafkaWriter.writeToKafkaBySingle()	Writes data from the DStream into Kafka one by one.

**Table 31-18** Spark Streaming enhancement interface

Method	Description
DStreamKafkaWriter.writeToKafka()	Writes data from the DStream into Kafka in batch.
DStreamKafkaWriter.writeToKafkaBySingle()	Writes data from the DStream into Kafka one by one.

## SparkSQL Common Interfaces

Spark SQL mainly uses the following classes:

- SQLContext: main entrance of the Spark SQL function and DataFrame.
- DataFrame: a distributed dataset organized by naming columns.
- HiveContext: main entrance for obtaining data stored in Hive.

**Table 31-19** Common Actions methods

Method	Description
collect(): Array[Row]	Returns an array containing all DataFrame columns.
count(): Long	Returns the number of DataFrame rows.
describe(cols: String*): DataFrame	Counts the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
first(): Row	Returns the first row.
Head(n:Int): Row	Returns the first <i>n</i> rows.
show(numRows: Int, truncate: Boolean): Unit	Displays DataFrame in a table.
take(n:Int): Array[Row]	Returns the first <i>n</i> rows in the DataFrame.

**Table 31-20** Basic DataFrame Functions

Method	Description
explain(): Unit	Prints the logical plan and physical plan of the SQL.
printSchema(): Unit	Prints the schema information to the console.
registerTempTable(tableName: String): Unit	Registers the DataFrame as a temporary table, whose period is bound to the SQLContext.
toDF(colNames: String*): DataFrame	Returns a DataFrame whose columns are renamed.

### 31.5.1.3 Python

To avoid API compatibility or reliability issues after updates to the open-source Spark, it is advisable to use APIs of the version you are currently using. For details about APIs, see [conref:text]Target does not exist.

## Spark Core Common Interfaces

Spark mainly uses the following classes:

- `pyspark.SparkContext`: external API of Spark. It provides the functions of Spark for Python applications that invoke this class, for example, connecting Spark clusters, creating RDDs, and broadcasting variables.
- `pyspark.SparkConf`: Spark application configuration class. It is used to set an application name, execution mode, and executor memory.
- `pyspark.RDD`: class used to define the RDD in the Spark application. The class provides the data collection operation methods, such as map and filter.
- `pyspark.Broadcast`: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- `pyspark.StorageLevel`: data storage levels, including memory (`MEMORY_ONLY`), disk (`DISK_ONLY`), and memory+disk (`MEMORY_AND_DISK`).
- `pyspark.sql.SQLContext`: Main entry point for SparkSQL functionality. It can be used to create DataFrame, register DataFrame as a table, and execute SQL on a table.
- `pyspark.sql.DataFrame`: A distributed collection of data grouped into named columns. A DataFrame is equivalent to a relational table in Spark SQL, and can be created using various functions in SQLContext.
- `pyspark.sql.DataFrameNaFunctions`: Functionality for working with missing data in DataFrame.
- `pyspark.sql.DataFrameStatFunctions`: function in DataFrame for statistics. It calculates the variance between columns and sample covariance.

The RDD supports two types of operations, transformation and action. [Table 31-21](#) and [Table 31-22](#) show the common methods.

**Table 31-21** Transformation

Method	Description
map(f, preservesPartitioning=False)	Returns a new RDD by applying a function to all elements of this RDD.
filter(f)	Invokes the Func method for all RDD elements to generate a satisfied data set that is returned in the form of RDD.
flatMap(f, preservesPartitioning=False)	Returns a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
sample(withReplacement, fraction, seed=None)	Returns a sampled subset of this RDD.
union(rdds)	Returns a new RDD, contains source RDD and the group of RDD's elements.
distinct([numPartitions: Int]): RDD[T]	Returns a new RDD containing the distinct elements in this RDD.
groupByKey(): RDD[(K, Iterable[V])]	Returns (K,Iterable[V]) and combines the values of the same key to a set.
reduceByKey(func, numPartitions=None)	Invokes Func on the values of the same key.
sortByKey(ascending=True, numPartitions=None, keyfunc=function <lambda>)	Sorts by key in ascending or descending order. Ascending is of the boolean type.
join(other, numPartitions)	Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. <b>numPartitions</b> indicates the number of concurrent tasks.
cogroup(other, numPartitions)	Returns the dataset of (K, (Iterable[V], Iterable[W])) when the (K,V) and (K,W) datasets of two key-value pairs exist. <b>numPartitions</b> indicates the number of concurrent tasks.
cartesian(other)	Returns the Cartesian product of the RDD and other RDDs.

**Table 31-22** Action

API	Description
reduce(f)	Invokes Func on elements of the RDD.

API	Description
collect()	Returns an array that contains all of the elements in this RDD.
count()	Returns the number of elements in the dataset.
first()	Returns the first element in this RDD.
take(num)	Returns the first num elements of the RDD.
takeSample(withReplacement, num, seed)	Samples the dataset randomly and returns a dataset of num elements. <b>withReplacement</b> indicates whether replacement is used.
saveAsTextFile(path, compressionCodecClass)	Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file.
saveAsSequenceFile(path, compressionCodecClass=None)	This API can be used only on the key-value pair, and then it generates SequenceFile and writes the file to the local or Hadoop file system.
countByKey()	Counts the appearance times of each key.
foreach(func)	Runs the function on each element of the dataset.
countByValue()	Counts the times that each value of the RDD occurs.

## Spark Streaming Common APIs

Spark Streaming mainly uses the following classes:

- `pyspark.streaming.StreamingContext`: main entrance of Spark Streaming. It provides methods for creating the DStream. A batch interval needs to be set in the input parameter.
- `pyspark.streaming.DStream`: a type of data which indicates the RDDs continuous sequence. It indicates the continuous data flow.
- `dstream.PariDStreamFunctions`: the Dstream of key-value, common operations are `groupByKey` and `reduceByKey`.  
The cooperated Java APIs of Spark Streaming are `JavaStreamingContext`, `JavaDStream`, `JavaPairDStream`.

Common methods of Spark Streaming are the same as those of Spark Core. The following table describes some special Spark Streaming methods.

**Table 31-23** Spark Streaming common interfaces

Method	Description
socketTextStream(hostname, port, storageLevel)	Creates an input stream from the TCP source host:port.
start()	Starts Spark Streaming computing.
awaitTermination(timeout)	Terminates the await of the process, which is similar to pressing <b>Ctrl+C</b> .
stop(stopSparkContext, stopGracefully)	Stops Spark Streaming computing. <b>stopSparkContext</b> is used to determine whether SparkContext needs to be terminated. <b>StopGracefully</b> is used to determine whether to wait for all the received data to be processed.
updateStateByKey(func)	Updates the status of DStream. To use this method, you need to define the state and state update functions.
window(windowLength, slideInterval)	Generates a new DStream by batch calculating according to the window of the source DStream.
countByWindow(windowLength, slideInterval)	Returns the number of sliding window elements in the stream.
reduceByWindow(func, windowLength, slideInterval)	When the key-value pair of DStream is invoked, a new key-value pair of DStream is returned. The value of each key is obtained by aggregating the reduce function in batches in the sliding window.
join(other, numPartitions)	Performs a join operation between different Spark Streamings.

## SparkSQL Common Interfaces

Spark SQL mainly uses the following classes:

- pyspark.sql.SQLContext: main entrance of the Spark SQL function and DataFrame.
- pyspark.sql.DataFrame: a distributed dataset organized by naming columns.
- pyspark.sql.HiveContext: main entrance for obtaining data stored in Hive.
- pyspark.sql.DataFrameStatFunctions: Functionality for statistic functions with DataFrame.
- pyspark.sql.functions: A collection of builtin functions.
- pyspark.sql.Window: Utility functions for defining window in DataFrames.

**Table 31-24** Spark SQL common Actions

Method	Description
collect()	Returns an array containing all DataFrame columns.
count()	Returns the number of DataFrame rows.
describe()	Counts the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
first()	Returns the first row.
head(n)	Returns the first <i>n</i> rows.
show()	Displays DataFrame in a table.
take(num)	Returns the first num rows in the DataFrame.

**Table 31-25** Basic DataFrame Functions

Method	Description
explain()	Prints the logical plan and physical plan of the SQL.
printSchema()	Prints the schema information to the console.
registerTempTable(name)	Registers the DataFrame as a temporary table, whose period is bound to the SQLContext.
toDF()	Returns a DataFrame whose columns are renamed.

## 31.5.2 Common CLIs

For details about how to use the Spark CLIs, visit the official website <http://spark.apache.org/docs/3.1.1/quick-start.html>.

### Common CLI

Common Spark CLIs are described as follows:

- **spark-shell**

It provides an easy way to learn APIs, which is similar to the tool for interactive data analysis. It supports two languages including Scala and Python. In the Spark directory, run the `./bin/spark-shell` command to log in the interactive interface of Scala, obtain data from the HDFS, and perform the RDD.

For example: a row of codes can count all words in a file.

```
scala> sc.textFile("hdfs://10.96.1.57:9000//wordcount_data.txt").flatMap(l => l.split(" ")).map(w => (w,1)).reduceByKey(_+_).collect()
```

- **spark-submit**

It is used to submit the Spark application to the Spark cluster for running and return the running results. The class, master, jar and input parameter need to be specified.

For example: Run the GroupByTest example in the jar. There are four input parameters and the specified running mode of the cluster is local single platform.

```
./bin/spark-submit --class org.apache.spark.examples.GroupByTest --
master local[1] examples/jars/spark-examples_2.12-3.1.1-hw-ei-311001-
SNAPSHOT.jar 6 10 10 3
```

- **spark-sql**

It is used to run the Hive metadata service and query command lines in the local or cluster mode. If its logical plan needs to be queried, add "explain extended" before the SQL statement.

For example:

```
Select key from src group by key
```

- **run-example**

It is used to run or debug the default example in the Spark open-source community.

For example: Run the SparkPi.

```
./run-example SparkPi 100
```

## 31.5.3 JDBCServer Interface

### Overview

JDBCServer is another HiveServer2 implementation in Hive. It uses the Spark structured query language (SQL) to process the SQL statements, providing higher performance than Hive.

JDBCServer is a JDBC interface. Users can use JDBC to connect to JDBCServer to access SparkSQL data. When JDBCServer is started, a SparkSQL application is started, and the clients connected through JDBC can share the resources of the SparkSQL application, that is, different users can share data. When JDBCServer is started, a listener is also enabled to wait for the JDBC client to submit the connection and query requests. Therefore, when configuring JDBCServer, you need to configure at least the host name and port number of JDBCServer. If you want to use Hive data, you also need to provide the URIs of Hive MetaStore.

JDBCServer starts a JDBC service on port 10000 of the installation node by default. Users can connect to JDBCServer using Beeline or running the JDBC client code to run SQL statements.

For more information about JDBCServer, visit the Spark official website at <http://spark.apache.org/docs/3.1.1/sql-programming-guide.html#distributed-sql-engine>.

## Beeline

For details about the Beeline connection modes provided by the open source community, visit <https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients>.

```
sh CLIENT_HOME/spark/bin/beeline -u "jdbc:hive2://
<zkNode_IP>:<zkNode_Port>;serviceDiscoveryMode=zooKeeper;zooKeeperNa
mespace=sparkthriftserver2x;"
```

### NOTE

- In the preceding information, `<zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>` indicates the ZooKeeper URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181.
- `sparkthriftserver2x` indicates the ZooKeeper directory, where a random JDBCServer instance is connected to the client.

## JDBC Client Codes

Log in to JDBCServer by using the JDBC client codes and access the Spark SQL data. For details, see [Accessing the Spark SQL Through JDBC](#).

## Enhanced Features

Compared with the open source community, Huawei provides two enhanced features: JDBCServer HA solution and setting of the JDBCServer connection timeout interval.

- In the JDBCServer HA solution, multiple active JDBCServer nodes provide services at the same time. When one node is faulty, new client connections are allocated to other active nodes to ensure uninterrupted services for the cluster. The operations of using Beeline or JDBC client code for connection are the same.
- Set the timeout interval for the connection between the client and JDBCServer.

- Beeline

In the case of network congestion, this feature prevents the Beeline from being suspended due to the infinite waiting for the response of the server. The configuration method is as follows:

When the Beeline is started, append `--socketTimeout=n`, where **n** indicates the timeout interval (unit: second) for waiting for the service response. The default value is **0**, indicating that the beeline never times out. You are advised to set this parameter to the maximum value allowed based on the service scenario.

- JDBC Client Codes

In the case of network congestion, this feature prevents the client from being suspended due to the infinite waiting for the response of the server. The method is as follows:

Before using the `DriverManager.getConnection` method to obtain the JDBC connection, add the `DriverManager.setLoginTimeout(n)` method to configure a timeout interval. **n** indicates the timeout interval for



waiting for the return result from the server. The unit is second, the type is **Int**, and the default value is **0** (indicating never timing out). You are advised to set this parameter to the maximum value allowed based on the service scenario.

## 31.5.4 Structured Streaming Functions and Reliability

### Functions Supported by Structured Streaming

1. ETL operations on streaming data are supported.
2. Schema inference and partitioning of streaming DataFrames or Datasets are supported.
3. Operations on the streaming DataFrames or Datasets are supported, including SQL-like operations without types (such as select, where, and groupBy) and RDD operations with types (such as map, filter, and flatMap).
4. Aggregation calculation based on Event Time and processing of delay data are supported.
5. Deduplication of streaming data is supported.
6. Status computing is supported.
7. Stream processing tasks can be monitored.
8. Batch-stream join and stream join are supported.

The following table lists the supported join operations.

Left Table	Right Table	Supported Join Type	Description
Static	Static	All types	In stream processing, join operations with no streaming data involved are supported.
Stream	Static	Inner	Supported, but stateless.
		Left Outer	Supported, but stateless.
		Right Outer	Not supported.
		Full Outer	Not supported.
Stream	Stream	Inner	Supported. The watermark or time range can be used to clear status of the left and right tables.
		Left Outer	Conditionally supported. The watermark can be used to clear status of the left table, and watermark and time range must be used for the right table.

Left Table	Right Table	Supported Join Type	Description
		Right Outer	Conditionally supported. The watermark can be used to clear status of the right table, and watermark and time range must be used for the left table.
		Full Outer	Not supported.

## Functions Not Supported by Structured Streaming

1. Multi-stream aggregation is not supported.
2. The following operations of obtaining multiple rows are not supported: limit, first, and take.
3. Distinct is not supported.
4. Sorting is supported only when output mode is complete.
5. The external connection between streams and static data sets is conditionally supported.
6. Immediate query and result return on some data sets are not supported.
  - count(): Instead of returning a single count from streaming data sets, it uses ds.groupBy().count() to return a streaming data set containing the running count.
  - foreach(): The ds.writeStream.foreach(...) is used to replace it.
  - show(): The output console sink is used to replace it.

## Structured Streaming Reliability

Based on the checkpoint and WAL mechanisms, Structured Streaming provides end-to-end exactly-once error tolerance semantics for the sources that can be replayed and the idempotent sinks that support repeated processing.

1. You can enable the checkpoint function by setting option ("checkpointLocation", "checkpoint path") in the program.

When data is restored from checkpoint, the application or configuration may change. Some changes may result in the failure in restoring data from the checkpoint. The restrictions are as follows:

- a. The number or type of sources cannot be changed.
- b. The source type and query statement determine whether the source parameter can change. For example:
  - The parameters related to rate control can be added, deleted, or modified. For example:  
spark.readStream.format("kafka").option("subscribe", "topic") is changed to spark.readStream.format("kafka").option("subscribe", "topic").option("maxOffsetsPerTrigger", ...).
  - Unexpected problems may occur when the topic/file of the consumption is modified. For example:

- `spark.readStream.format("kafka").option("subscribe", "topic")` is changed to `spark.readStream.format("kafka").option("subscribe", "newTopic")`.
- c. The type of sink changes. Specific sinks can be combined. The specific scenarios need to be verified. For example:
    - File sink can be changed to kafka sink. Kafka processes only new data.
    - Kafka sink cannot be changed to file sink.
    - Kafka sink can be changed to foreach sink, and vice versa.
  - d. The sink type and query statement determine whether the sink parameter can change. For example:
    - The output path of file sink cannot be changed.
    - The output topic of Kafka sink can be changed.
    - The custom operator code in foreach sink can be changed, but the change result depends on the user code.
  - e. The projection, filter, and map-like operations can be changed in some scenarios. For example:
    - Filters can be added and deleted. For example: `sdf.selectExpr("a")` is changed to `sdf.where(...).selectExpr("a").filter(...)`.
    - When Output schema is the same, projections can be changed. For example: `sdf.selectExpr("stringColumn AS json").writeStream` is changed to `sdf.select(to_json(...).as("json")).writeStream`.
    - If Output schema is different, projections can be changed in some conditions. For example: when `sdf.selectExpr("a").writeStream` is changed to `sdf.selectExpr("b").writeStream`, no error occurs only when the sink supports schema conversion from a to b.
  - f. In some scenarios, the status restoration will fail after status is changed.
    - Streaming aggregation: For example, in the `sdf.groupBy("a").agg(...)` operation, the type or quantity of the grouping key or the aggregation key cannot be changed.
    - Streaming deduplication: For example, in the `sdf.dropDuplicates("a")` operation, the type or quantity of the grouping key or the aggregation key cannot be changed.
    - Stream-stream join: For example, in the `sdf1.join(sdf2, ...)` operation, the schema of the association key cannot be changed, and the join type cannot be changed. Change of other join conditions may lead to uncertainty results.
    - Any status computing: For example, in the `sdf.groupByKey(...).mapGroupsWithState(...)` or `sdf.groupByKey(...).flatMapGroupsWithState(...)` operation, the schema or timeout type of the user-defined status cannot be changed. Users can customize the state-mapping function change,

but the change result depends on the user code. If schema changes are required, users can encode or decode the status data into binary data to support schema migration.

2. Fault tolerance list of Source

Sources	Supported Options	Fault Tolerance Supported	Description
File source	<p><b>path</b>: file path, which is mandatory.</p> <p><b>maxFilesPerTrigger</b>: maximum number of files in each trigger. The default value is infinity.</p> <p><b>latestFirst</b>: whether to process limited number of new files. The default value is <b>false</b>.</p> <p><b>fileNameOnly</b>: whether the file name is used as the new file for verification instead of using the complete path. (Default value: <b>false</b>)</p>	Supported	<p>Paths with wildcard are supported, but multiple paths separated by commas (,) are not supported.</p> <p>Files must be placed in a given directory in atomic mode, which can be implemented through file movement in most file systems.</p>
Socket Source	<p><b>host</b>: IP address of the connected node, which is mandatory.</p> <p><b>port</b>: connected port, which is mandatory.</p>	Not supported.	-
Rate Source	<p><b>rowsPerSecond</b>: number of rows generated per second. The default value is <b>1</b>.</p> <p><b>rampUpTime</b>: rising time before the speed specified by <b>rowsPerSecond</b> is reached.</p> <p><b>numPartitions</b>: concurrency degree for generating data rows.</p>	Supported	-
Kafka Source	For details, see <a href="https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html">https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html</a> .	Supported	-

3. Fault tolerance list of Sink

Sinks	Supported Output Mode	Supported Options	Fault Tolerance	Description
File Sink	Append	<b>Path:</b> The specified file format must be specified. For details, see APIs in DataFrameWriter .	exactly-once	Data can be written to partition tables. Time-based partition is better.
Kafka Sink	Append, Update, Complete	For details, see <a href="https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html">https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html</a> .	at-least-once	For details, see <a href="https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html">https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html</a> .
Foreach Sink	Append, Update, Complete	None	Depends on Foreach Writer.	For details, see <a href="https://spark.apache.org/docs/3.1.1/structured-streaming-programming-guide.html#using-foreach">https://spark.apache.org/docs/3.1.1/structured-streaming-programming-guide.html#using-foreach</a> .
ForeachBatch Sink	Append, Update, Complete	None	Depends on operator.	For details, see <a href="https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#using-foreach-and-foreachbatch">https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#using-foreach-and-foreachbatch</a> .
Console Sink	Append, Update, Complete	<b>numRows:</b> number of rows printed in each round. The default value is <b>20</b> . <b>truncate:</b> whether to clear the output when the output is too long. The default value is <b>true</b> .	Not supported	-

Sinks	Supported Output Mode	Supported Options	Fault Tolerance	Description
Memory Sink	Append, Complete	None	Not supported. In complete mode, the entire table is rebuilt after the query is restarted.	-

## 31.5.5 FAQ

### 31.5.5.1 How to Add a User-Defined Library

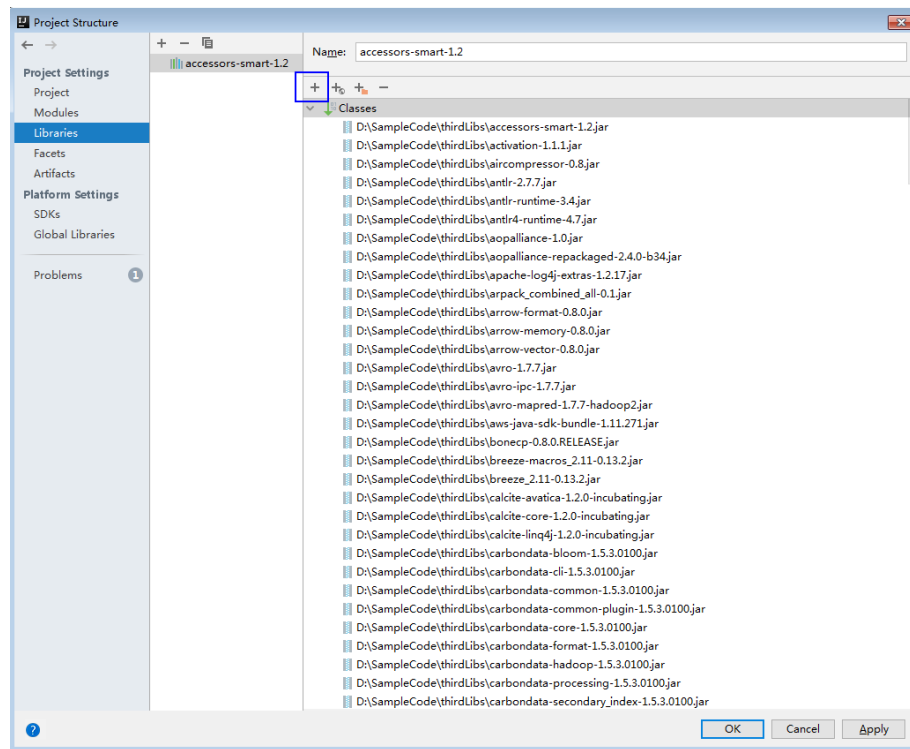
#### Question

In the development of the Spark application, users may add a user-defined dependent library which is different from the sample project. This section describes how to add a dependent library with user-defined codes into the project.

#### Answer

- Step 1** On the main page of the IDEA, choose **File > Project Structures...** to access the **Project Structure** page.
- Step 2** Select the **Libraries** tab, click **+** on the following page, and add the local dependent library.

Figure 31-41 Add the Dependent Library

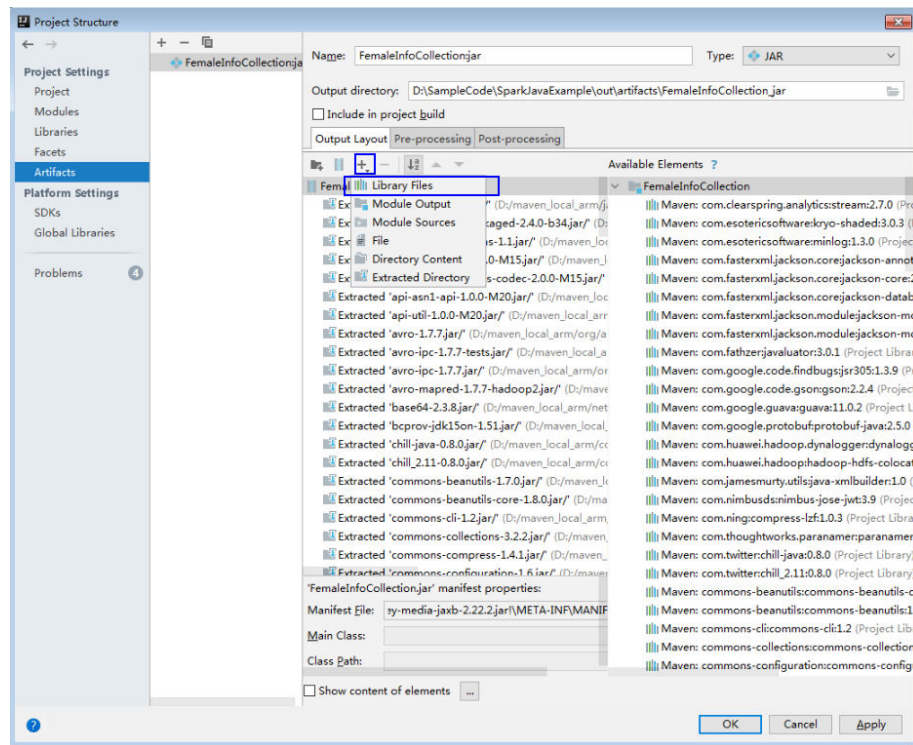


**Step 3** Click **Apply** to load the dependent library and click **OK** to complete the configuration.

**Step 4** Add the dependent library when building Artifacts. This is because the user-defined dependent library does not exist in the operating environment. By adding the library, the created jar contains the user-defined dependent library, which ensures that the Spark application runs properly.

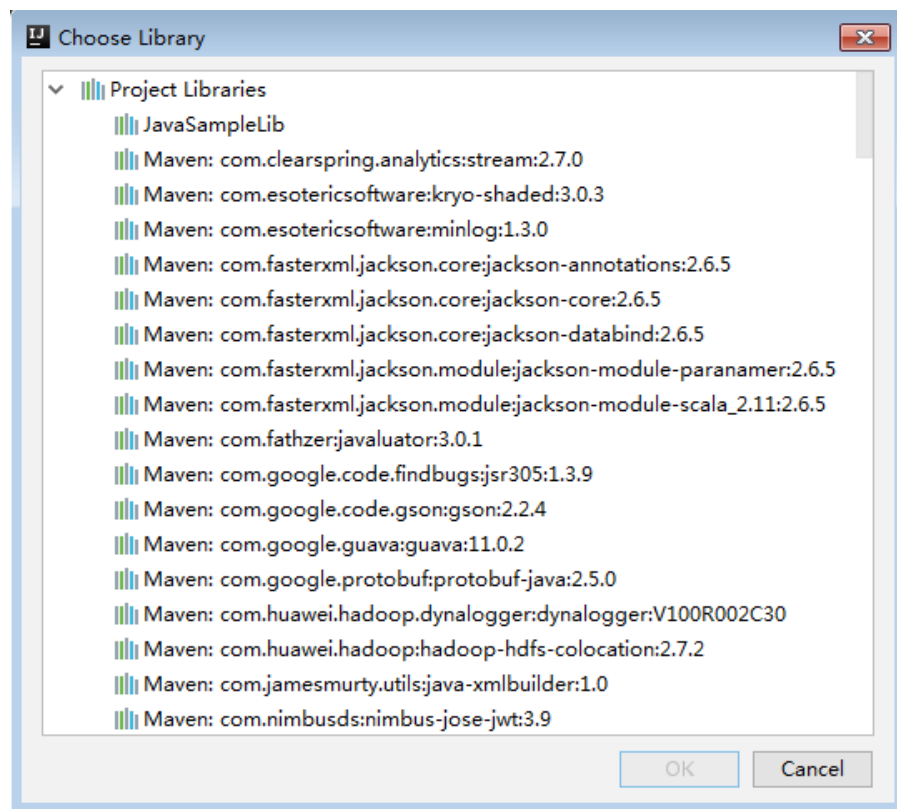
1. On the **Project Structure** page, select the **Artifacts** tab.
2. Click **+** and select **Library Files** in the right window to add the dependent library.

Figure 31-42 Add Library Files



3. Choose the dependent library to be added and click **OK**.

Figure 31-43 Choose Libraries





4. Click **Apply** to load the dependent library and click **OK** to complete the configuration.

----End

### 31.5.5.2 How to Automatically Load Jars Packages?

#### Question

Before importing a project by using the IDEA tool, if Maven has been configured in the IDEA tool, the IDEA tool will automatically load the Jars packages in the Maven configuration. When the automatically loaded Jars packages do not match with the application, the project fails to be built. How to Automatically Load Jars Packages?

#### Answer

After a project is imported, to manually delete the automatically loaded Jars packages, perform the following steps:

1. On the IDEA tool, choose **File > Project Structures...**
2. Select **Libraries** and select the Jars packages that are automatically imported. Right-click the mouse and select **Delete**.

### 31.5.5.3 Why the "Class Does not Exist" Error Is Reported While the SparkStreamingKafka Project Is Running?

#### Question

While the KafkaWordCount task (org.apache.spark.examples.streaming.KafkaWordCount) is being submitted by running the spark-submit script, the log file shows that the Kafka-related class does not exist. The KafkaWordCount sample is provided by the Spark open-source community.

#### Answer

When Spark is deployed, the following jar packages are saved in the **\$ {SPARK\_HOME}/jars/streamingClient010** directory on the client and the **\$ {BIGDATA\_HOME}/FusionInsight\_Spark2x\_[conref:text]undefined/install/FusionInsight-Spark2x-3.1.1/spark/jars/streamingClient010** directory on the server.

- kafka-clients-xxx.jar
- kafka\_2.12-xxx.jar
- spark-streaming-kafka-0-10\_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar
- spark-token-provider-kafka-0-10\_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar

Because **\$SPARK\_HOME/jars/streamingClient010/\*** is not added in to classpath by default, add "spark.executor.extraClassPath" and "spark.driver.extraClassPath" in the command.

When the application is submitted and run, add following parameters in the command. See [Compiling and Running the Application](#).

```
--jars $SPARK_CLIENT_HOME/jars/streamingClient010/kafka-client-2.4.0.jar,
$SPARK_CLIENT_HOME/jars/streamingClient010/kafka_2.12-*.jar,
$SPARK_CLIENT_HOME/jars/streamingClient010/spark-streaming-
kafka-0-10_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar
```

You can run the preceding command to submit the self-developed applications and sample projects.

To submit the sample projects such as KafkaWordCount provided by Spark open source community, you need to add other parameters in addition to `--jars`. Otherwise, the `ClassNotFoundException` error will occur. The configurations in `yarn-client` and `yarn-cluster` modes are as follows:

- `yarn-client` mode  
In the configuration file `spark-defaults.conf` on the client, add the path of the client dependency package, for example `$SPARK_HOME/jars/streamingClient010/*`, (in addition to `--jars`) to the `spark.driver.extraClassPath` parameter.
- `yarn-cluster` mode  
Perform any one of the following configurations in addition to `--jars`:
  - In the configuration file `spark-defaults.conf` on the client, add the path of the server dependency package, for example `/${BIGDATA_HOME}/FusionInsight_Spark2x_8.1.0.1/install/FusionInsight-Spark2x-3.1.1/spark/jars/streamingClient010/*`, to the `spark.yarn.cluster.driver.extraClassPath` parameter.
  - Delete the `original-spark-examples_2.12-3.1.1-xxx.jar` packages from all the server nodes.
  - In the `spark-defaults.conf` configuration file on the client, modify (or add and modify) the parameter `spark.driver.userClassPathFirst` to `true`.

### 31.5.5.4 Why Does Kafka Fail to Receive the Data Written Back by Spark Streaming?

#### Question

While a running Spark Streaming task is writing data back to Kafka, Kafka cannot receive the written data and Kafka logs contain the following error information:

```
2016-03-02 17:46:19,017 | INFO | [kafka-network-thread-21005-1] | Closing socket connection to /
10.91.8.208 due to invalid request: Request of length
122371301 is not valid, it is larger than the maximum size of 104857600 bytes. | kafka.network.Processor
(Logging.scala:68)
2016-03-02 17:46:19,155 | INFO | [kafka-network-thread-21005-2] | Closing socket connection to /
10.91.8.208. | kafka.network.Processor (Logging.scala:68)
2016-03-02 17:46:19,270 | INFO | [kafka-network-thread-21005-0] | Closing socket connection to /
10.91.8.208 due to invalid request:
Request of length 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. |
kafka.network.Processor (Logging.scala:68)
2016-03-02 17:46:19,513 | INFO | [kafka-network-thread-21005-1] | Closing socket connection to /
10.91.8.208 due to invalid request:
Request of length 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. |
kafka.network.Processor (Logging.scala:68)
2016-03-02 17:46:19,763 | INFO | [kafka-network-thread-21005-2] | Closing socket connection to /
```

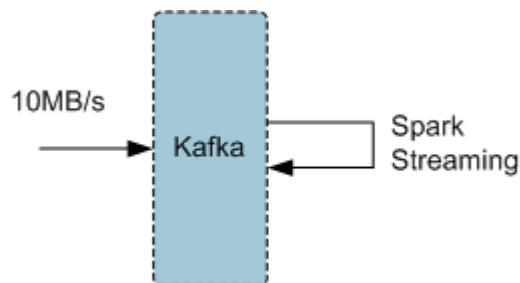
```
10.91.8.208 due to invalid request:
Request of length 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. |
kafka.network.Processor (Logging.scala:68)
53393 [main] INFO org.apache.hadoop.mapreduce.Job - Counters: 50
```

## Answer

As shown in the figure below, the logic defined in Spark Streaming applications is as follows: reading data from Kafka -> executing processing -> writing result data back to Kafka.

Imagine that data is written into Kafka at a data rate of 10 MB/s, the interval (defined in Spark Streaming) between write-back operations is 60s, and a total of 600-MB data needs to be written back into Kafka. If Kafka defines that a maximum of 500-MB data can be received at a time, then the size of written-back data exceeds the threshold, triggering the error information.

Figure 31-44 Application scenario



Solution:

- Method 1: On Spark Streaming, reduce the interval between write-back operations to avoid the size of written-back data exceeding the threshold defined by Kafka. The recommended interval is 5-10 seconds.
- Method 2: Increase the threshold defined in Kafka. It is advisable to increase the threshold by adjusting the `socket.request.max.bytes` parameter of Kafka service on FusionInsight Manager.

### 31.5.5.5 Why a Spark Core Application Is Suspended Instead of Being Exited When Driver Memory Is Insufficient to Store Collected Intensive Data?

#### Question

A Spark Core application is attempting to collect intensive data and store it into the Driver. When the Driver runs out of memory, the Spark Core application is suspended. Why does the Spark Core application not exit?

The following is the log information displayed at the time of out-of-memory (OOM) error.

```
16/04/19 15:56:22 ERROR Utils: Uncaught exception in thread task-result-getter-2
java.lang.OutOfMemoryError: Java heap space
at java.lang.reflect.Array.newInstance(Native Method)
at java.lang.reflect.Array.newInstance(Array.java:75)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1671)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
```

```
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1707)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
at org.apache.spark.serializer.JavaDeserializationStream.readObject(JavaSerializer.scala:71)
at org.apache.spark.serializer.JavaSerializerInstance.deserialize(JavaSerializer.scala:91)
at org.apache.spark.scheduler.DirectTaskResult.value(TaskResult.scala:94)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3$$$anonfunrun1.apply$mcV
 $sp(TaskResultGetter.scala:66)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3$$$anonfunrun1.apply(TaskResultGetter.scala:57)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3$$$anonfunrun1.apply(TaskResultGetter.scala:57)
at org.apache.spark.util.Utils$.logUncaughtExceptions(Utils.scala:1716)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3.run(TaskResultGetter.scala:56)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)
Exception in thread "task-result-getter-2" java.lang.OutOfMemoryError: Java heap space
at java.lang.reflect.Array.newInstance(Native Method)
at java.lang.reflect.Array.newInstance(Array.java:75)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1671)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1707)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
at org.apache.spark.serializer.JavaDeserializationStream.readObject(JavaSerializer.scala:71)
at org.apache.spark.serializer.JavaSerializerInstance.deserialize(JavaSerializer.scala:91)
at org.apache.spark.scheduler.DirectTaskResult.value(TaskResult.scala:94)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3$$$anonfunrun1.apply$mcV
 $sp(TaskResultGetter.scala:66)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3$$$anonfunrun1.apply(TaskResultGetter.scala:57)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3$$$anonfunrun1.apply(TaskResultGetter.scala:57)
at org.apache.spark.util.Utils$.logUncaughtExceptions(Utils.scala:1716)
at org.apache.spark.scheduler.TaskResultGetter$$$anon$3.run(TaskResultGetter.scala:56)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)
```

## Answer

If memory of the Driver is insufficient to store the intensive data that has been collected, the OOM error is reported and the Driver performs garbage collection repeatedly to reclaim the memory occupied by garbage. The Spark Core application remains suspended while garbage collection is under way.

If you expect the Spark Core application to exit forcibly in the event of OOM error, add the following information to the configuration option **spark.driver.extraJavaOptions** in the Spark client configuration file **\$SPARK\_HOME/conf/spark-defaults.conf** when you start the Spark Core application for the first time:

```
-XX:OnOutOfMemoryError='kill -9 %p'
```

### 31.5.5.6 Why the Name of the Spark Application Submitted in Yarn-Cluster Mode Does not Take Effect?

#### Question

The name of the Spark application submitted in yarn-cluster mode does not take effect, whereas the Spark application name submitted in yarn-client mode takes effect. In the following figure, the first application is submitted in yarn-client mode and the application name Spark Pi takes effect. However, the setAppName execution sequence of a task submitted in yarn-client mode is different from that submitted in yarn-cluster mode.

application_1463550673865_0007	zwm	Spark Pi	SPARK	tenant_zwm	Sat May 28 11:52:27 +0800 2016	Sat May 28 11:52:51 +0800 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A	History	N/A
application_1463550673865_0006	zwm	org.apache.spark.examples.SparkPi	SPARK	tenant_zwm	Sat May 28 11:52:29 +0800 2016	Sat May 28 11:52:00 +0800 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A	History	N/A

#### Answer

The reason is that the setAppName execution sequence of a task submitted in yarn-client mode is different from that submitted in yarn-cluster mode. In yarn-client mode, the setAppName is read before the application is registered in yarn. However, in yarn-cluster mode, the setAppName is read after the application registers with yarn, so the name of the second application does not take effect.

Solution:

When submitting tasks using the spark-submit script, set **--name** the same as the application name in sparkconf.setAppName(appname).

For example, if the application name is Spark Pi, in sparkconf.setAppName(appname) in yarn-cluster mode, run the following command:

```
./spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode cluster --name SparkPi jars/original-spark-examples*.jar 10
```


### 31.5.5.7 How to Perform Remote Debugging Using IDEA?

#### Question

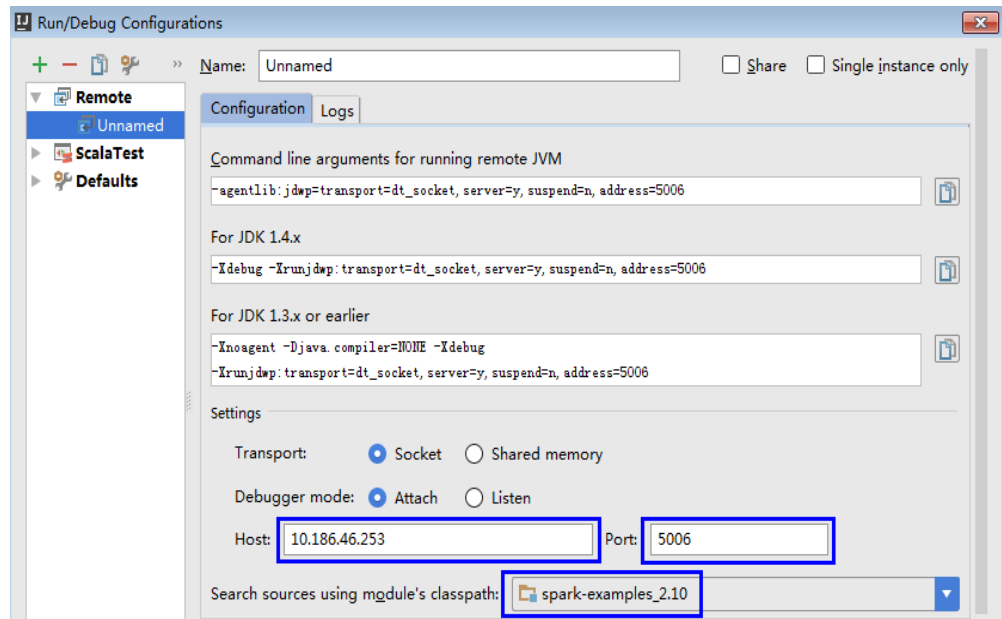
How to perform remote debugging using IDEA during Spark secondary development?

#### Answer

The SparkPi application is used as an example here to illustrate how to perform remote debugging using IDEA.

1. Open the project and choose **Run > Edit Configurations**.
2. In the displayed window, click  at the upper left corner, and then choose **Remote** on the drop-down menu, as shown in [Figure 31-45](#).

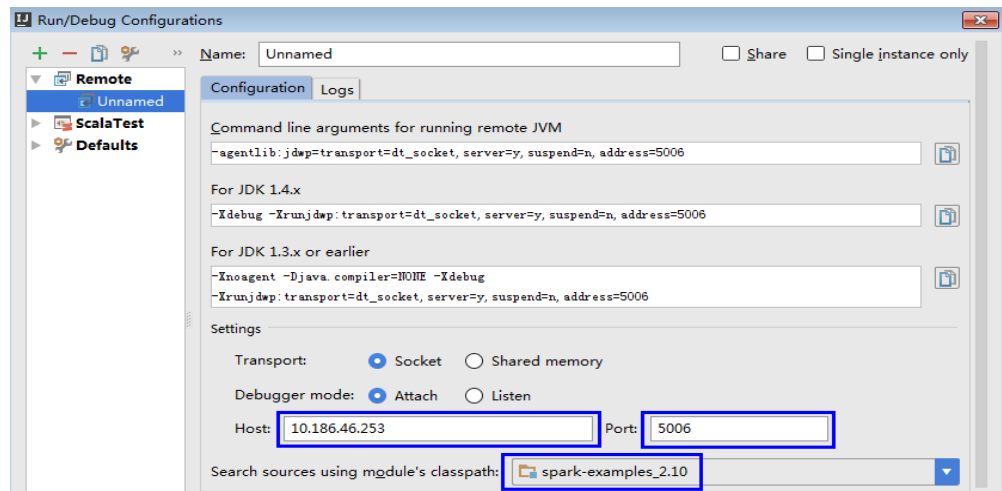
**Figure 31-45** Choosing Remote



3. Configure the **Host**, **Port**, and **Search source using module's classpath**, as shown in [Figure 31-46](#).

**Host** indicates the IP address of the Spark client and **Port** indicates the debugging port. Ensure that the port is available on the VM.

**Figure 31-46** Configuring parameters



**NOTE**

If the value of **Port** is changed, the debugging command of **For JDK1.4.x** must be changed accordingly. For example, if the value of **Port** is changed to **5006**, the debugging command must be changed to **-Xdebug -Xrunjdwp:transport=dt\_socket,server=y,suspend=y,address=5006**, which will be used during the startup of Spark.

4. Run the following command to remotely start SparkPi on the Spark client:  
`./spark-submit --master yarn-client --driver-java-options "-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5006" --class`

```
org.apache.spark.examples.SparkPi /opt/FI-Client/Spark2x/spark/examples/
jars/spark-examples_2.12-3.1.1-xxx.jar
```

Change the **--class** and JAR package in the preceding command to the **--class** and JAR package of the actual application. Change the **-Xdebug -Xrunjdpw:transport=dt\_socket,server=y,suspend=y,address=5006** to the **For JDK1.4.x** debugging command obtained in **3**.

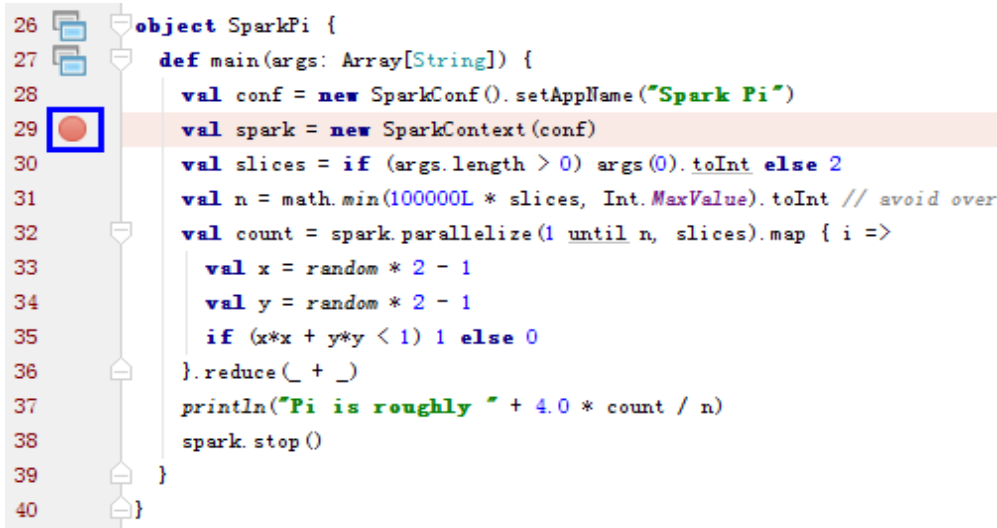
**Figure 31-47** Command for running Spark

```
[root@host2 bin]# ./spark-submit --master yarn-client --driver-java-options "-Xdebug
-Xrunjdpw:transport=dt_socket,server=y,suspend=y,address=5006" --class org.a
pache.spark.examples.SparkPi /opt/client/Spark2x/spark/examples/jars/spark-examp
les_2.11-2.1.0.jar
Listening for transport dt_socket at address: 5006
```

5. Set the debugging breakpoint.

Click the blank area on the left of the code editing window to select the breakpoint of code. **Figure 31-48** illustrates how to select the breakpoint of the code in row 29 of **SparkPi.scala**.

**Figure 31-48** Setting the breakpoint

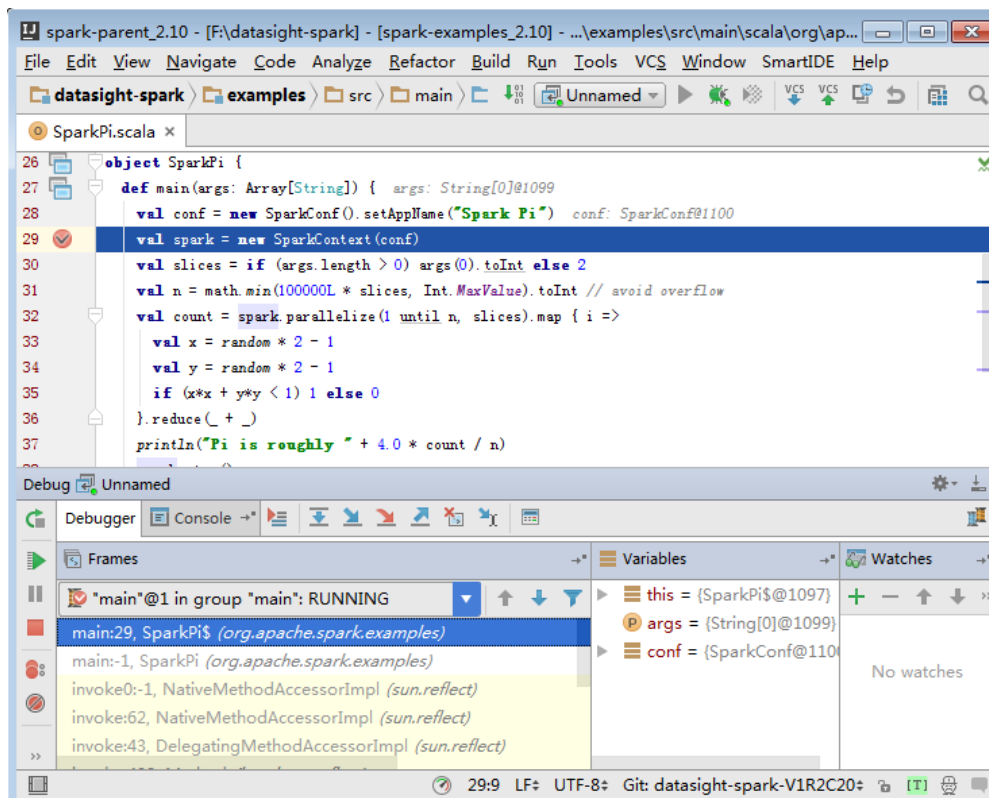


```
26 object SparkPi {
27 def main(args: Array[String]) {
28 val conf = new SparkConf().setAppName("Spark Pi")
29 val spark = new SparkContext(conf)
30 val slices = if (args.length > 0) args(0).toInt else 2
31 val n = math.min(100000L * slices, Int.MaxValue).toInt // avoid over
32 val count = spark.parallelize(1 until n, slices).map { i =>
33 val x = random * 2 - 1
34 val y = random * 2 - 1
35 if (x*x + y*y < 1) 1 else 0
36 }.reduce(_ + _)
37 println("Pi is roughly " + 4.0 * count / n)
38 spark.stop()
39 }
40 }
```

6. Start the debugging.

On the menu bar of IDEA, choose **Run > Debug 'Unnamed'** to open a debugging window. Start the debugging of **SparkPi**, for example, performing step-by-step debugging, checking call stack information, and tracking variable values, as shown in **Figure 31-49**.

Figure 31-49 Debugging



### 31.5.5.8 How to Submit the Spark Application Using Java Commands?

#### Question

How to submit the Spark application using Java commands in addition to spark-submit commands?

#### Answer

Use the org.apache.spark.launcher.SparkLauncher class and run Java command to submit the Spark application. The procedure is as follows:

**Step 1** Define the org.apache.spark.launcher.SparkLauncher class. The SparkLauncherJavaExample and SparkLauncherScalaExample are provided by default as sample codes. You can modify the input parameters of sample codes as required.

- If you use Java as the development language, you can compile the SparkLauncher class by referring to the following code:

```
public static void main(String[] args) throws Exception {
 System.out.println("com.huawei.bigdata.spark.examples.SparkLauncherExample <mode>
<jarParh> <app_main_class> <appArgs>");
 SparkLauncher launcher = new SparkLauncher();
 launcher.setMaster(args[0])
 .setAppResource(args[1]) // Specify user app jar path
 .setMainClass(args[2]);
 if (args.length > 3) {
 String[] list = new String[args.length - 3];
 for (int i = 3; i < args.length; i++) {
 list[i-3] = args[i];
 }
 }
}
```



```
 }
 // Set app args
 launcher.addAppArgs(list);
 }

 // Launch the app
 Process process = launcher.launch();
 // Get Spark driver log
 new Thread(new ISRRunnable(process.getErrorStream())).start();
 int exitCode = process.waitFor();
 System.out.println("Finished! Exit code is " + exitCode);
}
```

- If you use Scala as the development language, you can compile the SparkLauncher class by referring to the following code:

```
def main(args: Array[String]) {
 println(s"com.huawei.bigdata.spark.examples.SparkLauncherExample <mode> <jarParh>
<app_main_class> <appArgs>")
 val launcher = new SparkLauncher()
 launcher.setMaster(args(0))
 .setAppResource(args(1)) // Specify user app jar path
 .setMainClass(args(2))
 if (args.drop(3).length > 0) {
 // Set app args
 launcher.addAppArgs(args.drop(3): _*)
 }

 // Launch the app
 val process = launcher.launch()
 // Get Spark driver log
 new Thread(new ISRRunnable(process.getErrorStream())).start()
 val exitCode = process.waitFor()
 println(s"Finished! Exit code is $exitCode")
}
```

**Step 2** Develop the Spark application based on the service logic and configure constant values such as the main class of the user-compiled Spark application. For details about different scenarios, see [Developing the Project](#).

- If you use the security mode, you are advised to prepare the security authentication code, service application code, and related configurations according to the security requirements.

#### NOTE

In yarn-cluster mode, security authentication cannot be added to the Spark project because security authentication must be completed before the application is started. Therefore, users need to add security authentication code or run commands to perform security authentication. There is security authentication code in the sample code. In yarn-cluster mode, modify the corresponding security code before running the operation.

- In normal mode, prepare the service application code and related configurations.

**Step 3** Call the `org.apache.spark.launcher.SparkLauncher.launch()` function to submit user applications.

1. Generate jar packages from the SparkLauncher application and user applications, and upload the jar packages to the Spark node of the application. For details about how to generate jar packages, see [Compiling and Running the Application](#).
  - The compilation dependency package of SparkLauncher is **spark-launcher\_2.12-3.1.1-hw-ei-311001-SNAPSHOT.jar**. Please obtain it from the **jars** directory of **FusionInsight\_Spark2x\_8.1.0.1.tar.gz** in **Software**.

- The compilation dependency packages of user applications vary with the code. You need to load the dependency package based on the compiled code.
- 2. Upload the dependency jar package of the application to a directory, for example, `$SPARK_HOME/jars` (the node where the application will run).

Upload the dependency packages of the SparkLauncher class and the application to the `jars` directory on the client. The dependency package of the sample code has existed in the `jars` directory on the client.

 **NOTE**

If you want to use the Spark Launcher class, the node where the application runs must have the Spark client installed. The running of the Spark Launcher class is dependent on the configured environment variables, running dependency package, and configuration files.

- 3. In the node where the Spark application is running, run the following command to submit the application. Then you can check the running situation through Spark web UI and check the result by obtaining specified files. See [Checking the Commissioning Result](#) for details.

```
java -cp $SPARK_HOME/conf:$SPARK_HOME/jars/
*:SparkLauncherExample.jar
com.huawei.bigdata.spark.examples.SparkLauncherExample yarn-
client /opt/female/FemaleInfoCollection.jar
com.huawei.bigdata.spark.examples.FemaleInfoCollection <inputPath>
```

----End

### 31.5.5.9 A Message Stating "Problem performing GSS wrap" Is Displayed When IBM JDK Is Used

#### Question

A Message Stating "Problem performing GSS wrap" Is Displayed When IBM JDK Is Used.

#### Answer

##### Possible Causes

The authentication fails because the duration for creating a JDBC connection on IBM JDK exceeds the timeout duration for user authentication (one day by default).

 **NOTE**

The authentication mechanism of IBM JDK differs from that of Oracle JDK. IBM JDK checks time but does not detect external time update. Therefore, time is not updated even though `relogin` is called.

##### Solution

When one JDBC connection fails, disable this connection, and create a new connection to continue performing previous operations.

### 31.5.5.10 Application Fails When ApplicationManager Is Terminated During Data Processing in the Cluster Mode of Structured Streaming

#### Question

If ApplicationManager is terminated during data processing in the cluster mode of Structured Streaming, the following information is displayed when the application is executed, indicating an error:

```
2017-05-09 20:46:02,393 | INFO | main |
client token: Token { kind: YARN_CLIENT_TOKEN, service: }
diagnostics: User class threw exception: org.apache.spark.sql.AnalysisException: This query does not
support recovering from checkpoint location. Delete hdfs://hacluster/structuredtest/checkpoint/offsets to
start over.;
ApplicationMaster host: 10.96.101.170
ApplicationMaster RPC port: 0
queue: default
start time: 1494333891969
final status: FAILED
tracking URL: https://9-96-101-191:8090/proxy/application_1493689105146_0052/
user: spark2x | org.apache.spark.internal.Logging$class.logInfo(Logging.scala:54)
Exception in thread "main" org.apache.spark.SparkException: Application application_1493689105146_0052
finished with failed status
```

#### Answer

**Possible causes:** The error occurs because **recoverFromCheckpointLocation** is determined as **false** but the checkpoint directory is configured.

The value of the **recoverFromCheckpointLocation** parameter is the result of the **outputMode == OutputMode.Complete()** statement in the code. (The default **outputMode** is **append**.)

**Solution:** When compiling an application, you can change the data output mode based on the actual conditions.

When the output mode is changed to **complete**, the value of **recoverFromCheckpointLocation** is determined as **true**. No error would be indicated if the checkpoint directory is configured at the time.

### 31.5.5.11 Restrictions on Restoring the Spark Application from the checkpoint

#### Question

The Spark application can be restored from the checkpoint and continues to execute the task from the breakpoint of the last task, ensuring that data is not lost. However, in some cases, the Spark application fails to be restored from the checkpoint.

#### Answer

The checkpoint contains the object serialization information, task execution status information, and configuration information of the Spark application. Therefore, the Spark application cannot be restored from the checkpoint if the following problems exist:

1. The service code is changed and the SerialVersionUID is not specified in the changed class.
2. The internal Spark class is changed and the SerialVersionUID is not specified in the changed class.

Besides, some configuration items are stored in the checkpoint. Therefore, if some configuration items of the service are modified, the configuration items may remain unchanged when the service is restored from the checkpoint. Currently, only the following configurations are reloaded when the service is restored from the checkpoint.

```
"spark.yarn.app.id",
"spark.yarn.app.attemptId",
"spark.driver.host",
"spark.driver.bindAddress",
"spark.driver.port",
"spark.master",
"spark.yarn.jars",
"spark.yarn.keytab",
"spark.yarn.principal",
"spark.yarn.credentials.file",
"spark.yarn.credentials.renewalTime",
"spark.yarn.credentials.updateTime",
"spark.ui.filters",
"spark.mesos.driver.frameworkId",
"spark.yarn.jars"
```

## Solution

Manually delete the checkpoint directory and restart the service program.

### NOTE

Deleting a file or folder is a high-risk operation. Ensure that the file or folder is no longer required before performing this operation.

## 31.5.5.12 Support for Third-party JAR Packages on x86 and TaiShan Platforms

### Question

How to enable Spark2x to support the third-party JAR packages (for example, custom UDF packages) if these packages have two versions (x86 and TaiShan)?

### Answer

If the third-party JAR packages (for example, custom UDF packages) has two versions (x86 and TaiShan), the following hybrid solution is used:

- Step 1** Go to the installation directory of the Spark2x SparkResource on the server. The cluster may be installed on multiple nodes. You can go to any SparkResource node to go to the installation directory of the SparkResource.
- Step 2** Prepare the .jar package, for example, xx.jar packages for the x86 and TaiShan platforms. Copy the xx.jar packages of x86 and TaiShan to the x86 and TaiShan folders respectively.

**Step 3** Run the following commands in the current directory to compress the JAR packages:

```
zip -qDj spark-archive-2x-x86.zip x86/*
```

```
zip -qDj spark-archive-2x-arm.zip arm/*
```

```

drwxr-xr-x 2 omm wheel 63 Dec 14 17:57 arm
drwxr-xr-x 2 omm wheel 4096 Dec 14 17:57 bin
drwxr-xr-x 2 omm ficommon 4096 Dec 14 17:57 carbonlib
-rw-r--r-- 1 omm wheel 1403 Dec 15 12:51 child.crt
-rw-r--r-- 1 omm wheel 1097 Dec 15 12:51 child.csr
-rw-r--r-- 1 omm wheel 6296 Dec 15 12:51 child.keystore
drwxr-xr-x 2 omm wheel 326 Dec 14 17:57 conf
drwxr-xr-x 3 omm wheel 18 Dec 14 17:54 examples
drwxr-xr-x 4 omm ficommon 12288 Dec 14 17:57 jars
-rw-r--r-- 1 omm wheel 18045 Dec 14 17:54 LICENSE
-rw-r--r-- 1 omm wheel 26366 Dec 14 17:54 NOTICE
drwxr-xr-x 5 omm wheel 129 Dec 14 17:57 python
drwxr-xr-x 3 omm wheel 17 Dec 14 17:54 R
-rw-r--r-- 1 omm wheel 501 Dec 14 17:54 README
-rw-r--r-- 1 omm wheel 20 Dec 14 17:54 RELEASE
drwxr-xr-x 2 omm wheel 496 Dec 15 17:14 sbin
-rw-r--r-- 1 root root 14904892 Dec 15 17:14 spark-archive-2x-arm.zip
-rw-r--r-- 1 root root 13881100 Dec 15 17:15 spark-archive-2x-x86.zip
-rw-r--r-- 1 omm wheel 244 Dec 14 17:57 version.properties
drwxr-xr-x 2 omm wheel 63 Dec 14 17:57 x86
drwxr-xr-x 2 omm wheel 42 Dec 14 17:54 yarn

```

**Step 4** Run the following command to check the .jar package on which the Spark2x of HDFS depends:

```
hdfs dfs -ls /user/spark2x/jars/8.1.0.1
```

NOTE

Change the version number 8.1.0.1 as required.

Run the following commands to move the .jar package files from HDFS, for example, /tmp.

```
hdfs dfs -mv /user/spark2x/jars/8.1.0.1/spark-archive-2x-arm.zip /tmp
```

```
hdfs dfs -mv /user/spark2x/jars/8.1.0.1/spark-archive-2x-x86.zip /tmp
```

**Step 5** Run the following commands to upload the spark-archive-2x-arm.zip and spark-archive-2x-x86.zip packages in Step 3 to the /user/spark2x/jars/8.1.0.1 directory of HDFS:

```
hdfs dfs -put spark-archive-2x-arm.zip /user/spark2x/jars/8.1.0.1/
```

```
hdfs dfs -put spark-archive-2x-x86.zip /user/spark2x/jars/8.1.0.1/
```

After the upload is complete, delete local files spark-archive-2x-arm.zip and spark-archive-2x-x86.zip.

**Step 6** Perform Step 1 to Step 2 for other SparkResource nodes.

**Step 7** Log in to the web UI and restart the jdbcServer instance of Spark2x.

**Step 8** After the restart, update the client configuration. Copy xx.jar for the corresponding platform to the Spark2x installation directory \${install\_home}/Spark2x/spark/jars on the client according to client server type (x86 or TaiShan). \${install\_home} indicates the installation path of the client. Replace it with the actual one. If the local installation directory is /opt/hadoopclient, copy the corresponding xx.jar to the /opt/hadoopclient/Spark2x/spark/jars folder.

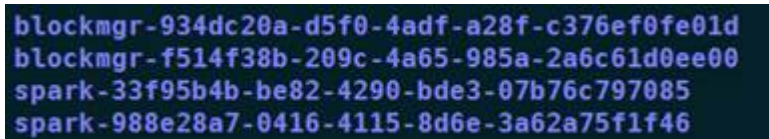
----End

### 31.5.5.13 What Should I Do If a Large Number of Directories Whose Names Start with **blockmgr-** or **spark-** Exist in the **/tmp** Directory on the Client Installation Node?

#### Question

After the system runs for a long time, there are many directories whose names start with **blockmgr-** or **spark-** in the **/tmp** directory on the node where the client is installed.

**Figure 31-50** Residual directory example



```
blockmgr-934dc20a-d5f0-4adf-a28f-c376ef0fe01d
blockmgr-f514f38b-209c-4a65-985a-2a6c61d0ee00
spark-33f95b4b-be82-4290-bde3-07b76c797085
spark-988e28a7-0416-4115-8d6e-3a62a75f1f46
```

#### Answer

During the running of Spark tasks, the driver creates a local temporary directory whose name starts with **spark-** for storing service JAR packages and configuration files. In addition, the driver creates a local temporary directory with the name starting with **blockmgr-** for storing block data. The two directories are automatically deleted when the Spark application running is finished.

The path for storing the two directories is preferentially specified by the environment variable *SPARK\_LOCAL\_DIRS*. If the environment variable is not configured, use the value of **spark.local.dir** as the path for storing the directories. If the environment variable and the preceding parameter both are not configured, use the value of **java.io.tmpdir**. By default, **spark.local.dir** is set to **/tmp** on the client. Therefore, the **/tmp** directory is used by default.

In some special cases, for example, the driver process does not exit normally, for example, the **kill -9** command ends the process, or the Java virtual machine crashes. As a result, the directory cannot be deleted and remains in the system.

Currently, only the driver processes in yarn-client mode and local mode may confront the preceding problem. In yarn-cluster mode, the temporary directory of the process in the container is configured as the temporary directory of the container. When the container exits, the container automatically clears the directory. Therefore, this problem does not occur in yarn-cluster mode.

#### Solution

In Linux, you can configure automatic directory clearing for the **/tmp** temporary directory. Alternatively, you can change the value of **spark.local.dir** in the **spark-defaults.conf** configuration file on the client, specify the temporary directory to a specified directory, and configure a clear mechanism for the directory.

### 31.5.5.14 Error Code 139 Reported When Python Pipeline Runs in the ARM Environment

#### Question

Error code 139 is displayed when the pipeline of the Python plug-in is used on the TaiShan server. The error information is as follows:

```
subprocess exited with status 139
```

#### Answer

The Python program uses both **libcrypto.so** and **libssl.so**. If the native library directory of Hadoop is added to **LD\_LIBRARY\_PATH**, the **libcrypto.so** in the hadoop native library is used and the **libssl.so** provided by the system is used (because the hadoop native directory does not contain this package). The versions of the two libraries do not match. As a result, a segment error occurs during the running of the Python file.

#### Solution

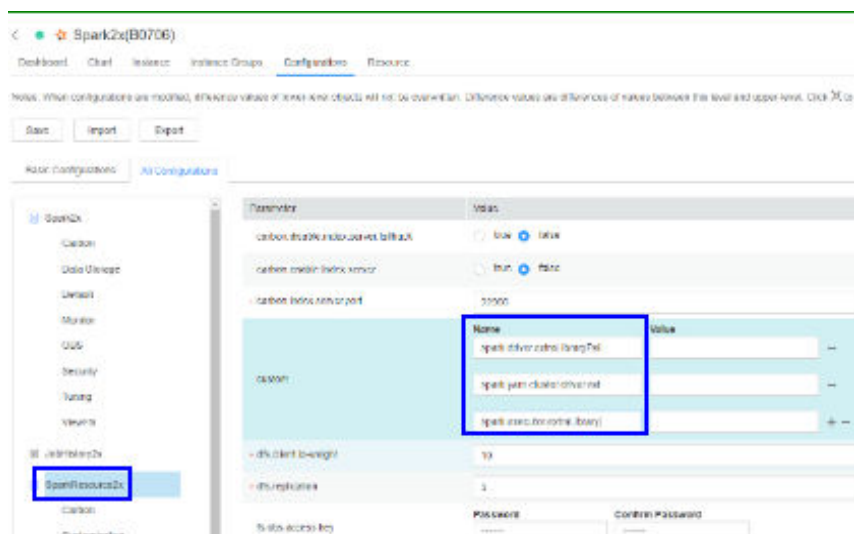
Solution 1:

Modify the **spark-default.conf** file in the **conf** directory of the Spark2x client. Clear the values of **spark.driver.extraLibraryPath**, **spark.yarn.cluster.driver.extraLibraryPath**, and **spark.executor.extraLibraryPath**.

Solution 2:

On the Spark2x page of FusionInsight Manager, modify the preceding three parameters. Restart the Spark2x instance, and download the client again. The procedure is as follows:

1. Log in to FusionInsight Manager, choose **Cluster > Name of the desired cluster > Services > Spark2x > Configurations > All Configurations**, search for the **spark.driver.extraLibraryPath** and **spark.executor.extraLibraryPath** parameters, and clear their values.
2. Choose **All Configurations > SparkResource2x**. In the **custom** area, add the three parameters in solution 1, as shown in the following figure.



3. Click **Save**. Restart the expired spark2x instance. Download and install the client again.

### 31.5.5.15 What Should I Do If the Structured Streaming Task Submission Way Is Changed?

#### Question

When submitting a structured streaming task, users need to run the `--jars` command to specify the Kafka JAR package path, for example, `--jars /kafkadir/kafka-clients-x.x.x.jar,/kafkadir/kafka_2.11-x.x.x.jar`. However, in the current version, users need to configure additional items. Otherwise, an error is reported, indicating that the class is not found.

#### Answer

The Spark kernel of the current version depends on the Kafka JAR package, which is used by the structured streaming. Therefore, when submitting a structured streaming task, you need to add the Kafka JAR package path to the library directory of the driver of this task to ensure that the driver can properly load the Kafka package.

#### Solution

1. The following operations need to be performed additionally when a structured streaming task in Yarn-client mode is submitted:  
Copy the path of `spark.driver.extraClassPath` in the `spark-default.conf` file in the Spark client directory, and add the Kafka JAR package path to its end. When submitting a structured stream task, add the `--conf` statement to combine these two configuration items. For example, if the Kafka JAR package path is `/kafkadir`, you need to add `--conf spark.driver.extraClassPath=/opt/client/Spark2x/spark/conf:/opt/client/Spark2x/spark/jars/*:/opt/client/Spark2x/spark/x86/*:/kafkadir/*` when submitting the task.
2. The following operations need to be performed additionally when a structured streaming task in **Yarn-cluster** mode is submitted:  
Copy the path of `spark.yarn.cluster.driver.extraClassPath` in the `spark-default.conf` file in the Spark client directory, and add relative paths of Kafka JAR packages to its end. When submitting a structured stream task, add the `--conf` statement to combine these two configuration items. For example, if the Kafka JAR package paths are `kafka-clients-x.x.x.jar` and `kafka_2.11-x.x.x.jar`, you need to add `--conf spark.yarn.cluster.driver.extraClassPath=/home/huawei/Bigdata/common/runtime/security:/kafka-clients-x.x.x.jar:/kafka_2.11-x.x.x.jar` when submitting the task.
3. In the current version, the structured streaming of Spark does not support versions earlier than Kafka2.x. In the upgrade scenario, use the client of earlier versions.



### 31.5.5.16 Common JAR File Conflicts

#### Symptom

Spark can interconnect with many third-party tools. Therefore, it depends on a large number of third-party packages. Some packages are provided by MRS. As a result, the versions of the JAR files used by the code may be different from those of the JAR files provided by the cluster. In this case, JAR file conflicts may occur.

Common JAR file conflict errors are as follows:

1. A class cannot be found: **java.lang.NoClassDefFoundError**
2. A method cannot be found: **java.lang.NoSuchMethodError**

#### Cause Analysis

The following uses UDF customization as an example:

```
2021-02-08 10:51:04.249 INFO [main] Total input files to process : 2 | org.apache.hadoop.mapred.FileInputFormat.ListStatus(FileInputFormat.java:256)
Exception in thread "main" java.lang.NoClassDefFoundError: com.huawei.udf.HelloUDF
 at com.huawei.sparkwordcount$.main(SparkWordCount.scala:32)
 at com.huawei.sparkwordcount$.main(SparkWordCount.scala)
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
 at java.lang.reflect.Method.invoke(Method.java:498)
 at org.apache.spark.deploy.SparkSubmit$.org$apache$spark$deploy$SparkSubmit$$runMain(SparkSubmit.scala:813)
 at org.apache.spark.deploy.SparkSubmit$.doRunMain$1(SparkSubmit.scala:187)
 at org.apache.spark.deploy.SparkSubmit$.submit(SparkSubmit.scala:212)
 at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:126)
 at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
Caused by: java.lang.ClassNotFoundException: com.huawei.udf.HelloUDF
 at java.net.URLClassLoader.findClass(URLClassLoader.java:382)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
 ... 11 more
```

If the class cannot be found, do as follows:

1. Check whether the JAR file of the class is in the classpath of the JVM. The JAR files of the Spark are stored in the **Spark client directory/jars/** directory.
2. Check whether multiple JAR files contain the class.
3. Check whether other dependencies are added to the task by using **--jars**.
4. If the dependencies are added to the task but cannot be found, the driver of the configuration file or the classpath of the executor may be incorrectly configured. View logs to check whether the configuration file is loaded to the environment.
5. Check whether the error is caused by a class initialization failure that occurs before using the class.

```
Exception in thread "main" java.lang.NoSuchMethodError: com.huawei.udf.HelloUDF.evaluate(Ljava/lang/String;Ljava/lang/String;
 at com.huawei.sparkwordcount$.main(SparkWordCount.scala:32)
 at com.huawei.sparkwordcount$.main(SparkWordCount.scala)
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
 at java.lang.reflect.Method.invoke(Method.java:498)
 at org.apache.spark.deploy.SparkSubmit$.org$apache$spark$deploy$SparkSubmit$$runMain(SparkSubmit.scala:813)
 at org.apache.spark.deploy.SparkSubmit$.doRunMain$1(SparkSubmit.scala:187)
 at org.apache.spark.deploy.SparkSubmit$.submit(SparkSubmit.scala:212)
 at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:126)
 at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
```

If the method cannot be found, do as follows:

1. Check whether the JAR file of the class corresponding to the method is loaded to the classpath of the JVM. The classes of the Spark are stored in the **Spark client directory/jars/** directory.
2. Check whether multiple JAR files contain the class. (Pay special attention to different versions of the same tool.)
3. If the error is reported for a Hadoop package, the possible cause is that the Hadoop version is inconsistent. As a result, some methods have been modified.

4. If the error is reported for a third-party package, the possible cause is that the Spark has the related JAR file, but the version is different from that in the code.

## Procedure

Solution 1:

Check whether third-party tool packages are required. If the packages can be replaced with the packages of the same version in the cluster, modify dependency versions.

### NOTE

You are advised to use the dependencies provided by the MRS cluster.

Solution 2:

The following shows how to modify a JAR file version:

MRS\_2.1 is used as an example.

1. Add the **<properties>** parameter to the **pom.xml** file and set the variables to facilitate subsequent version modification.

```
<groupId>com.huawei</groupId>
<artifactId>SparkDemo</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 <spark.version>2.3.2-mrs-2.1</spark.version>
 <hbase.version>2.1.1.0101-mrs-2.1</hbase.version>
 <hadoop.version>3.1.1-mrs-2.1</hadoop.version>
</properties>
```

2. Use the preceding defined variables to set the version of each JAR file in the **dependencies** parameter.

```
<dependencies>
 <dependency>
 <groupId>org.apache.spark</groupId>
 <artifactId>spark-core_2.11</artifactId>
 <version>${spark.version}</version>
 </dependency>
 <dependency>
 <groupId>org.apache.hadoop</groupId>
 <artifactId>hadoop-common</artifactId>
 <version>${hadoop.version}</version>
 </dependency>
 <dependency>
 <groupId>org.apache.hbase</groupId>
 <artifactId>hbase-common</artifactId>
 <version>${hbase.version}</version>
 </dependency>
 <dependency>
 <groupId>org.apache.hbase</groupId>
 <artifactId>hbase-client</artifactId>
 <version>${hbase.version}</version>
 </dependency>
</dependencies>
```

If other third-party packages conflict, check whether the same package of different versions exists by querying the dependency relationship. If yes, replace the JAR file version with the version of the JAR file provided by the cluster.

For details, see the **pom.xml** file of the MRS sample project in [Obtaining Sample Projects from Huawei Mirrors](#).

3. Print the dependency tree.

Run the **mvn dependency:tree** command in the directory where the **pom.xml** file is stored.

# 32 Storm Development Guide (Security Mode)

---

## 32.1 Overview

### 32.1.1 Application Development Overview

#### Intended Audience

This document is provided for users who want to implement Storm secondary development. This document is intended for development personnel who are experienced in Java development.

#### Introduction

Storm is a distributed, reliable, and fault-tolerant data stream processing system. Storm delegates work tasks to components of different types, and each component processes a specific simple task. Storm processes big data streams in real time and unlimited data streams in reliable mode.

Storm applies to real-time analytic, online machine learning, continuous computation, and distributed Extract, Transform, and Load (ETL). It is scalable and fault-tolerant, and is easy to set up and operate. This ensures data processing.

Storm has the following features:

- **Wide application**
- **High scalability**
- **Free from data loss**
- **High fault tolerance**
- **Language independence**
- **Ease of construction and control**

## 32.1.2 Common Concepts

### Topology

A computing stream chart, in which each node contains processing logic and the lines between nodes specify data flow between nodes.

### Spout

A component that generates source data streams in a Topology. A Spout reads data from an external data source and converts the data into source data inside a Topology.

### Bolt

A component that receives data from a Topology and then processes data. A Bolt can perform operations, such as filtering, executing functions, combination, and writing data into a database.

### Tuple

Basic unit for transferring messages once.

### Stream

A set of (infinite) elements, each of which belongs to a same schema. Each element is related to the logic time, that is, a Stream has the Tuple and Time attributes. Any elements can be expressed in the format of Element<Tuple,Time>, in which Tuple includes the data structure and content, and Time is the logic time of data.

### keytab file

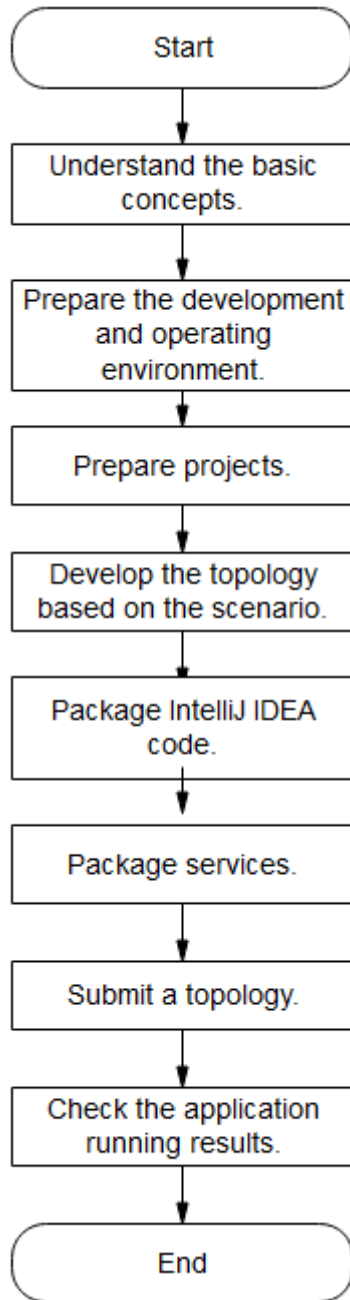
A key file used for storing user information. Applications use the key file for API authentication on service.

## 32.1.3 Development Process

This document describes Storm topology development based on the Java API.

[Figure 32-1](#) shows the development process.

**Figure 32-1** Topology development process



**Table 32-1** Description about Storm application development process

Phase	Description	Reference Document
Understand the basic concepts.	Before application development, understand common concepts about Storm, understand the scenario requirements, and topologies.	<a href="#">Common Concepts</a>

Phase	Description	Reference Document
Prepare the development and operating environment.	The Java language is recommended for Storm application development. The IntelliJ IDEA tool can be used.  The Storm running environment is a Storm client. Install and configure the client according to the guide.	<a href="#">Preparing for Development and Operating Environment</a>
Prepare projects.	Storm provides sample projects for different scenarios. You can import a sample project to learn the application.	<a href="#">Configuring and Importing Sample Projects</a>
Develop topologies based on scenarios.	The Storm topology construction and Spout/Bolt development process are provided.	<a href="#">Developing an Application</a>
Package the IntelliJ IDEA code.	Storm example programs are running in the Linux OS. IntelliJ IDEA code needs to be packaged into a JAR package.	<a href="#">Packaging IntelliJ IDEA Code</a>
Package services.	Combine the JAR package generated from the IntelliJ IDEA code and the project dependent JAR package to generate the <b>source.jar</b> package that can be submitted.	<a href="#">Packaging Services</a>
Submit a topology.	Submit the developed application for running.	<a href="#">Submitting a Topology</a>
View application running results.	Check the program running result after submitting topologies.	<a href="#">Viewing Results</a>

## 32.2 Environment Preparation

## 32.2.1 Environment Preparation Overview

Prepare clients for developing and submitting applications. Generally, applications are developed in Windows and submitted in Linux.

## 32.2.2 Preparing for Development and Operating Environment

### Preparing Development Environment

**Table 32-2** describes the environment required for secondary development.

**Table 32-2** Development environment

Item	Description
OS	<ul style="list-style-type: none"> <li>Development environment: Windows OS. Windows 7 or later is supported.</li> <li>Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li> </ul>
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows:</p> <p>The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed.</p> <p>Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> <li>For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> <li>Oracle JDK versions: 1.8</li> <li>IBM JDK versions: 1.8.5.11</li> </ul> </li> <li>For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> <li>OpenJDK: 1.8.0_272</li> </ul> </li> </ul> <p><b>NOTE</b></p> <p>The server supports only TLS V1.2 or later to ensure security.</p> <p>By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls">https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls</a>.</p>



Item	Description
IntelliJ IDEA installation and configuration	<p>Tool used for developing Storm applications. Requirements are as follows:</p> <p>JDK 1.8 is used. IntelliJ IDEA 2019.1 or other compatible versions are used.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li> <li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li> <li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li> </ul>
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
Developer account preparation	See <a href="#">Preparing a Developer Account</a> for configuration.
7-zip	It is a tool used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-Zip 16.04 is supported.

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. [Log in to the FusionInsight Manager portal](#) and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
 For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar file**. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar file** to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Storm\config**. Obtain the Storm-related configuration file.  
[Table 32-3](#) describes the main configuration files.

**Table 32-3** Configuration file

Document Name	Function
storm.yaml	Configures Storm Cluster Information.
user.keytab	Provides user information for Kerberos security authentication.
krb5.conf	Kerberos server configuration information
streaming-site.xml	Configures Storm parameters.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, C:\WINDOWS\system32\drivers\etc\hosts.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client on the node. For example, the client installation directory is /opt/client.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. [Log in to the FusionInsight Manager portal](#). Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Storm/config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, /opt/client/conf.

For example, if the client software package is

**FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is /tmp/FusionInsight-Client on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
```

```
scp Storm/config/* root@IP address of the client node:/opt/client/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 32-4](#) describes the main configuration files.

**Table 32-4** Configuration file

Document Name	Function
storm.yaml	Configures Storm Cluster Information.
user.keytab	Provides user information for Kerberos security authentication.
krb5.conf	Kerberos server configuration information
streaming-site.xml	Configures Storm parameters.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

## 32.2.3 Configuring and Importing Sample Projects

### Background

The Storm development example project is included in the Storm client installation directory. Import the project to IntelliJ IDEA for learning.

### Prerequisites

Ensure that the difference between the local PC time and the cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the cluster time in the lower-right corner on FusionInsight Manager.

### Procedure

- Step 1** Obtain the sample project folder **storm-examples** in the **src\storm-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the files **user.keytab** and **krb5.conf** files obtained in [Preparing a Developer Account](#) section and the cluster configuration file obtained in section [Preparing an Operating Environment](#) to the **storm-examples\src\main\resources** directory of the sample project.

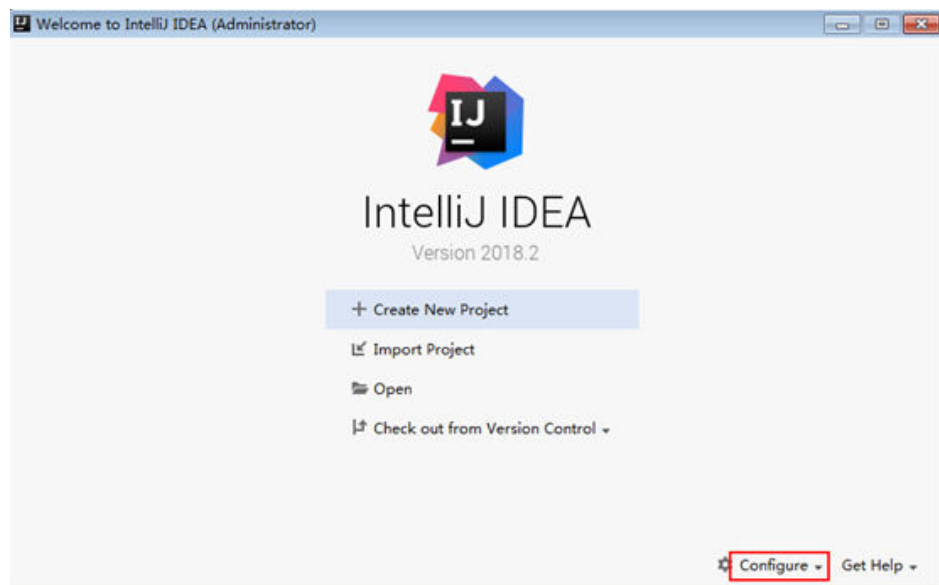
 **NOTE**

- To submit a topology when no client is installed in Windows or Linux, you need to place both streaming-site.xml and storm.yaml in the storm-examples\src\main\resources directory of the sample project.
- To submit a topology when the client is installed in Linux, you only need to place streaming-site.xml in the storm-examples\src\main\resources directory of the sample project.

**Step 3** After installing the IntelliJ IDEA and JDK tools, configure JDK in IntelliJ IDEA.

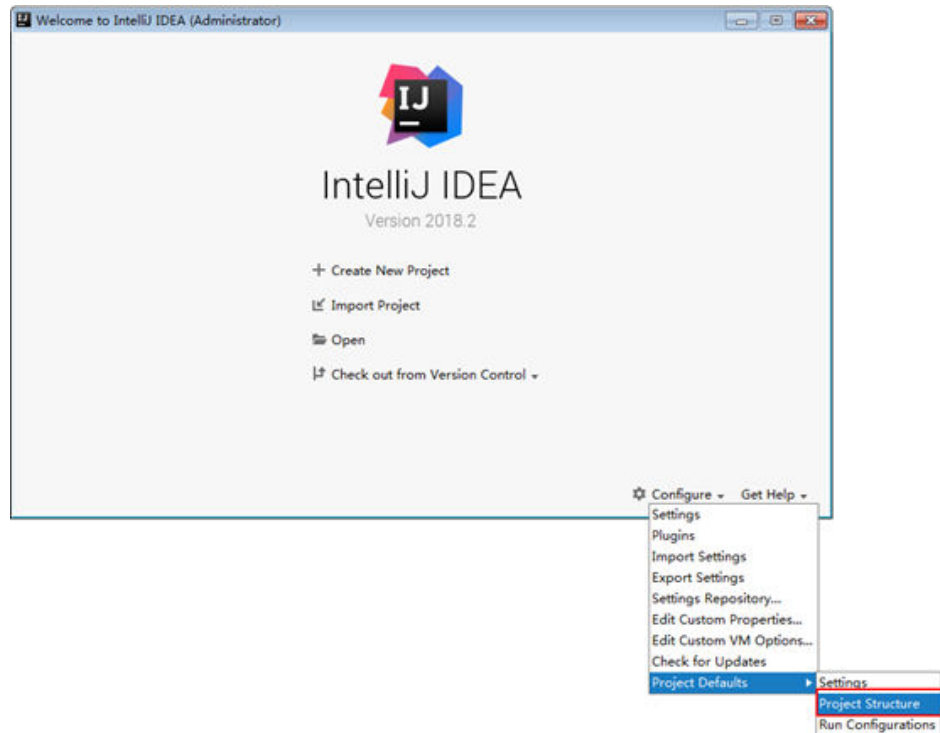
1. Open IntelliJ IDEA and click **Configure**.

**Figure 32-2** Quick Start



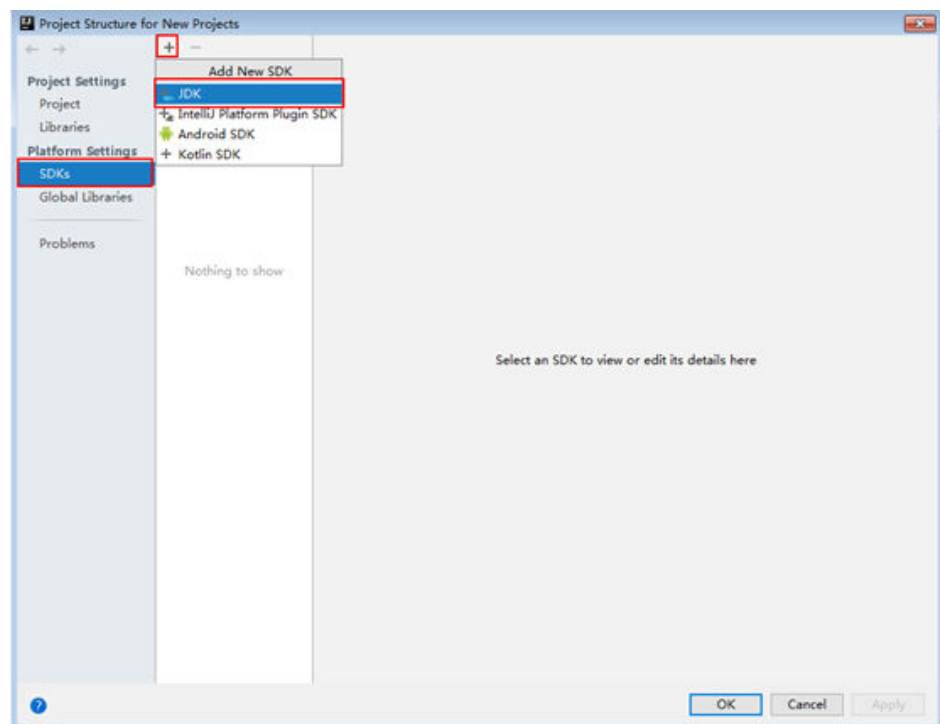
2. Choose **Project Defaults > Project Structure**.

Figure 32-3 Configure



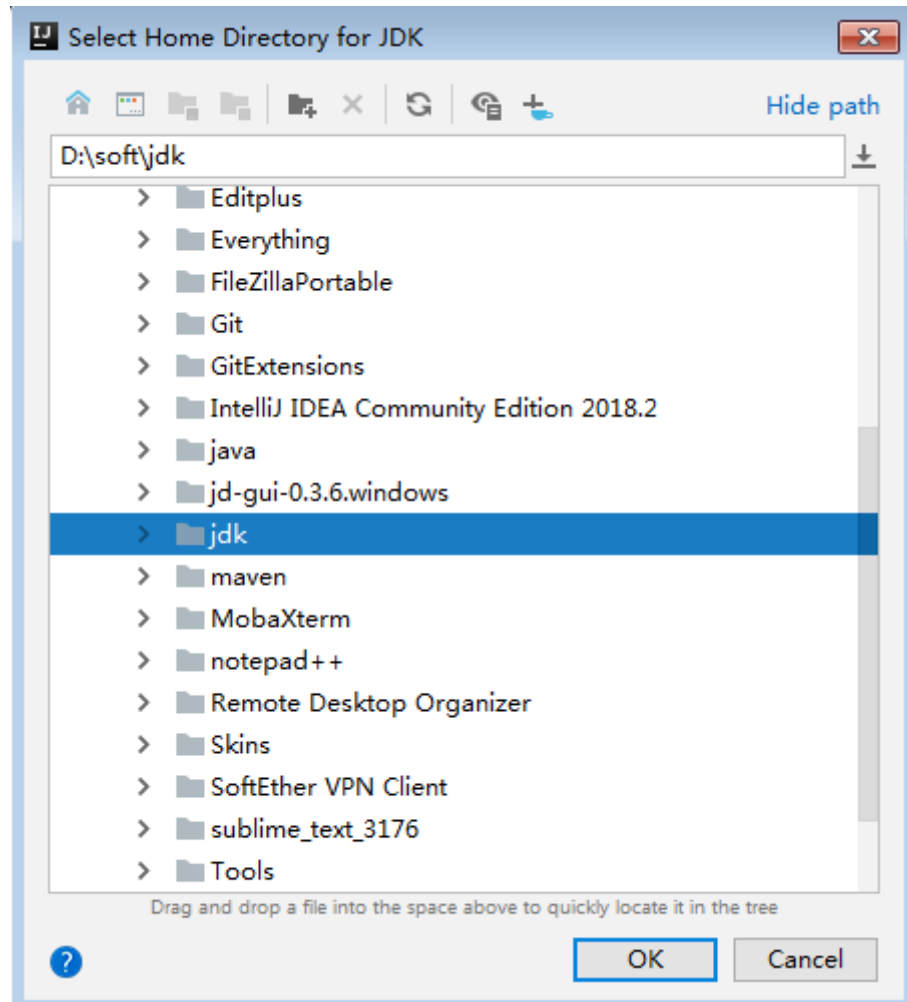
3. In the displayed **Project Structure for New Projects** interface, click **SDKs**. Then click the plus sign (+) and choose **JDK**.

Figure 32-4 Project Structure for New Projects



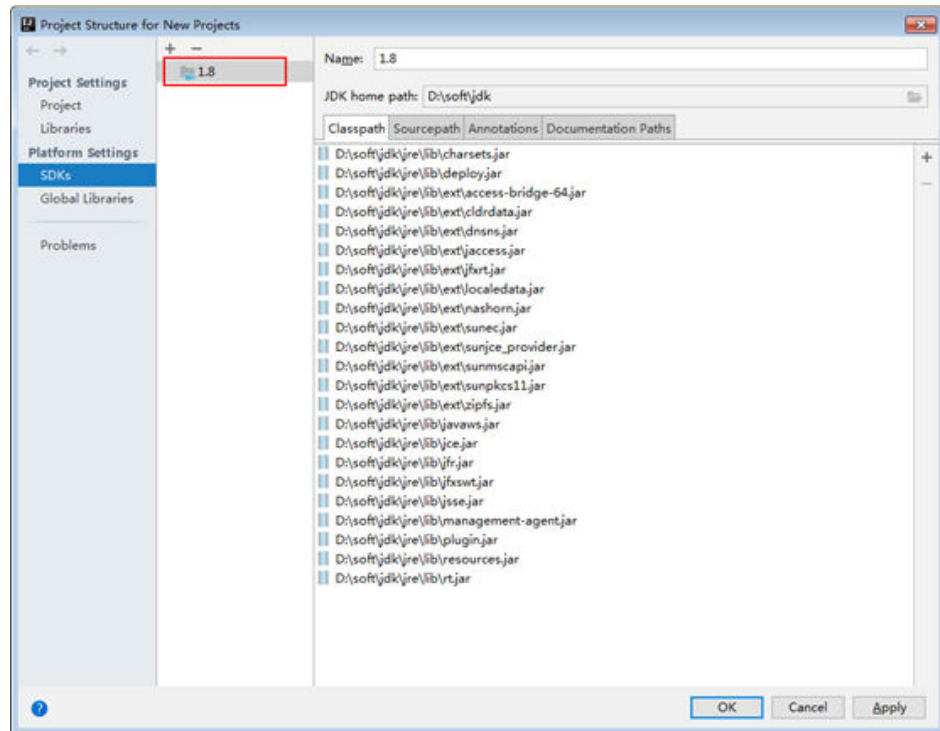
4. In the displayed **Select Home Directory for JDK** interface, select the JDK directory, and click **OK**.

**Figure 32-5** Select Home Directory for JDK



5. Click **OK**.

Figure 32-6 Completing the JDK configuration

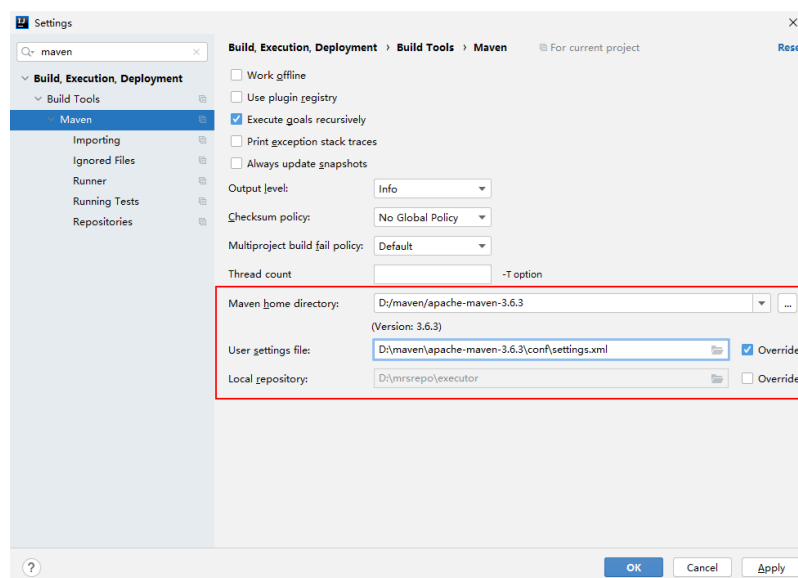


**Step 4** In the application development environment, import the sample project to the IntelliJ IDEA development environment.

1. Configure the IntelliJ IDEA Maven project environment.

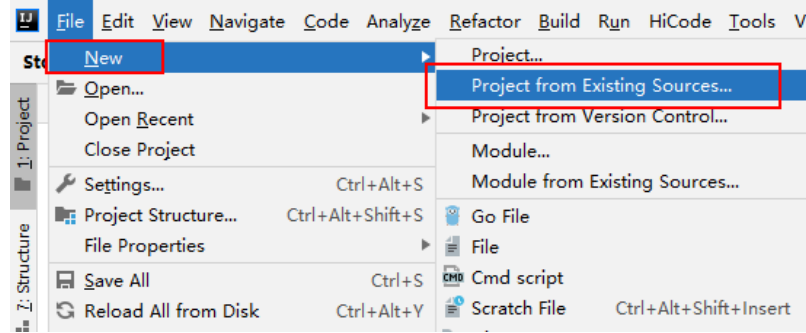
Choose **File > Settings** and search for **Maven**. Configure the Maven project, and choose **Apply > OK**.

Figure 32-7 Configuring the IntelliJ IDEA Maven project environment



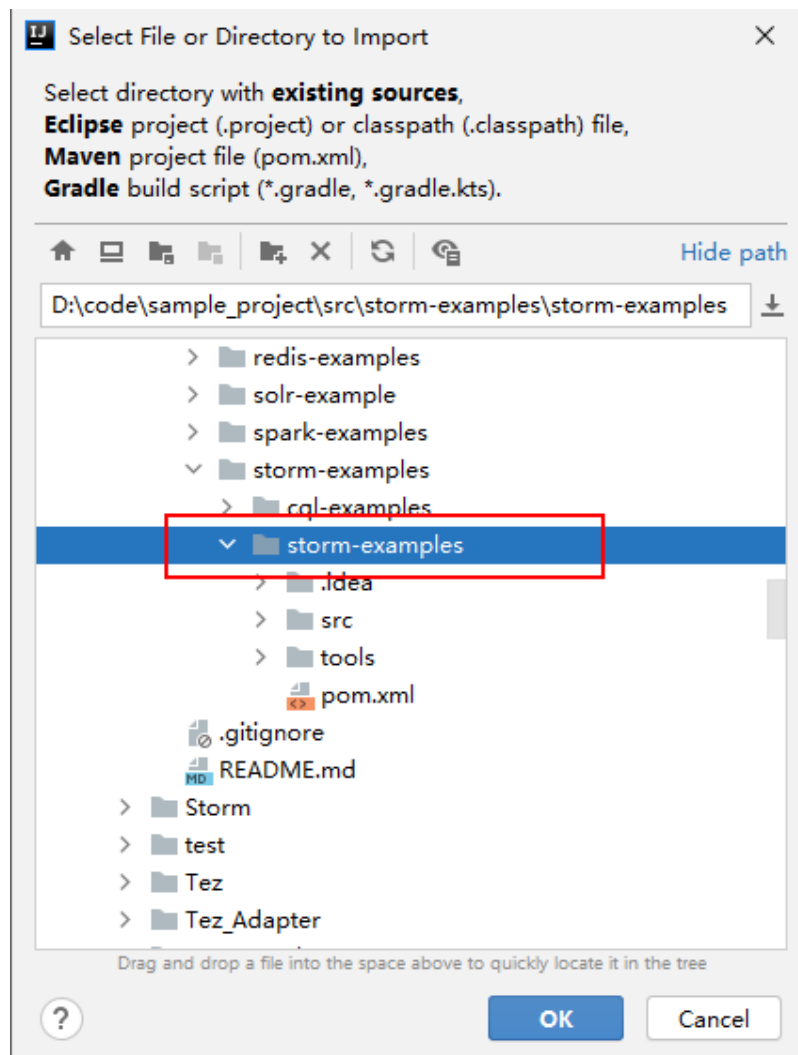
2. Choose **File > New > Project from Existing Sources....**

Figure 32-8 The Project from Existing Sources.... page is displayed



3. Select the sample project to be imported, for example, **storm-examples**.

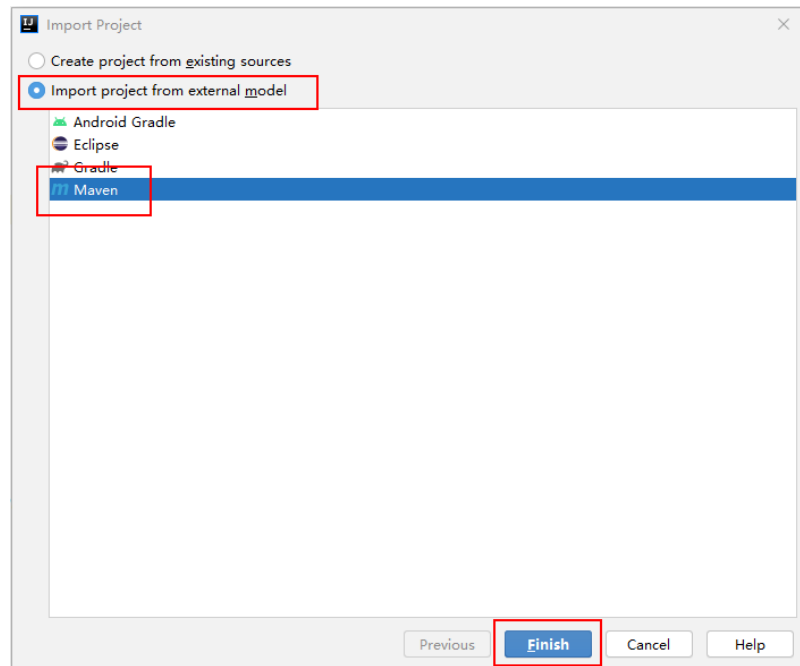
Figure 32-9 Selecting the sample project to be imported



4. Import the project as a Maven project.



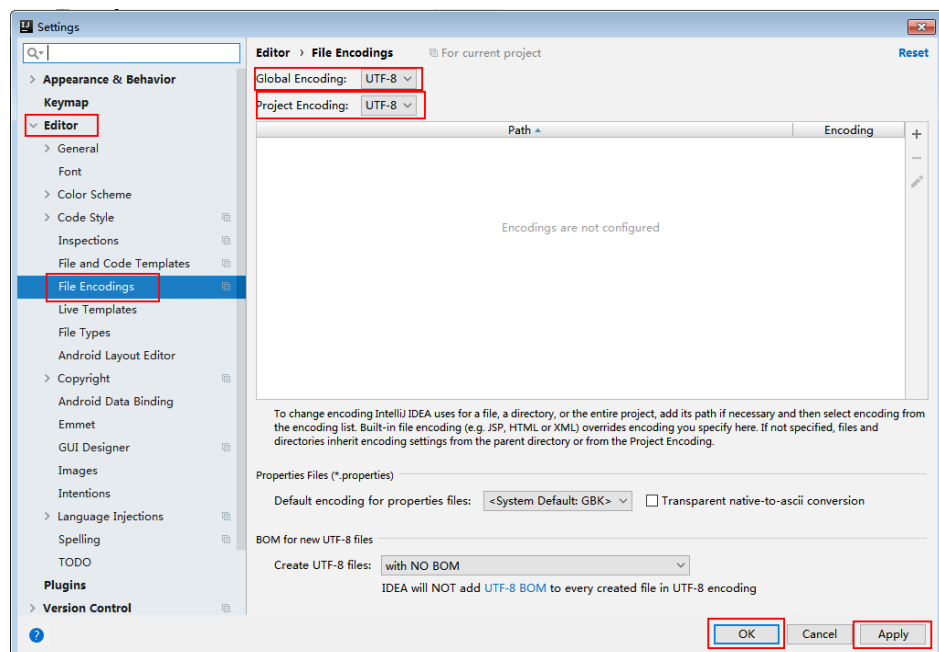
**Figure 32-10** Importing the project as a Maven project.



**Step 5** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.  
The **Settings** window is displayed.
2. In the navigation tree, choose **Editor > File Encodings**. In the **Project Encoding** area and **Global Encoding** area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in **Figure 32-11**.

**Figure 32-11** Setting the IntelliJ IDEA encoding format



----End

## 32.3 Developing an Application

### 32.3.1 Typical Scenario Description

A typical scenario can help you quickly learn and master the Storm topology structure and Spout/Bolt development process.

#### Scenario Description

Service processing of a dynamic word counting system is described as follows. The data source is a logical unit that produces random text continuously.

- The data source continuously sends random text, such as "apple orange apple", to the text splitting logic.
- The word splitting logic splits each text entry sent by the data source by space, such as "apple", "orange", "apple", and then sends each word to the word counting logic.
- The word counting logic increases the number of times that a specific word occurs by one when receiving the word, and prints the real-time results. For example:

apple: 1

orange: 1

apple: 2

### 32.3.2 Development Idea

#### Function Decomposition

The procedure for a user to develop an application to calculate the number of times that each word appears in random text is as [Table 32-5](#):

**Table 32-5** Functions to be developed

No.	Procedure	Code Sample
1	Create a Spout to generate random text.	For details, see <a href="#">Creating a Spout</a> .
2	Create a Bolt to split the random text into words.	For details, see <a href="#">Creating a Bolt</a> .
3	Create a Bolt to calculate the number of times that each word appears.	For details, see <a href="#">Creating a Bolt</a> .
4	Create a Topology.	For details, see <a href="#">Creating a Topology</a> .

For details about certain code, see [Code Samples](#). For details about complete code, see the storm-examples project.

## 32.3.3 Example Code Description

### 32.3.3.1 Creating a Spout

#### Function Description

A Spout is a message source of Storm and message producer of Topology. Generally, a message source reads data from an external source and sends messages (Tuple) to Topology.

One message source can send multiple message streams, and therefore, `OutputFieldsDeclarer.declarerStream` can be used to define multiple streams, and then `SpoutOutputCollector` emits specific streams.

#### Code Sample

The following code snippet belongs to the `nextTuple` method in the `RandomSentenceSpout` class of the `com.huawei.storm.example.common` package, and these code snippets are used to split strings into words.

```
/**
 * {@inheritDoc}
 */
@Override
public void nextTuple()
{
 Utils.sleep(100);
 String[] sentences =
 new String[] {"the cow jumped over the moon",
 "an apple a day keeps the doctor away",
 "four score and seven years ago",
 "snow white and the seven dwarfs",
 "i am at two with nature"};
 String sentence = sentences[random.nextInt(sentences.length)];
 collector.emit(new Values(sentence));
}
```

### 32.3.3.2 Creating a Bolt

#### Function Description

All message processing logic is encapsulated in Bolts. Bolts provide multiple functions, such as filtering and aggregation.

If other topology operators, except for Bolts, exist, `OutputFieldsDeclarer.declareStream` can be used to define streams, and `OutputCollector.emit` can be used to select streams to be emitted.

#### Code Sample

The following code snippet belongs to the `execute` method in the `SplitSentenceBolt` class of the `com.huawei.storm.example.common` package, and these code snippets are used to split a statement into words and send the words.

```
/**
 * {@inheritDoc}
 */
@Override
public void execute(Tuple input, BasicOutputCollector collector)
{
 String sentence = input.getString(0);
 String[] words = sentence.split(" ");
 for (String word : words)
 {
 word = word.trim();
 if (!word.isEmpty())
 {
 word = word.toLowerCase();
 collector.emit(new Values(word));
 }
 }
}
```

The following code snippets are in the `com.huawei.storm.example.wordcount.WordCountBolt` class, and these code snippets are used to calculate the number of received words.

```
@Override
public void execute(Tuple tuple, BasicOutputCollector collector)
{
 String word = tuple.getString(0);
 Integer count = counts.get(word);
 if (count == null)
 {
 count = 0;
 }
 count++;
 counts.put(word, count);
 System.out.println("word: " + word + ", count: " + count);
}
```

### 32.3.3.3 Creating a Topology

#### Function Description

A topology is a directed acyclic graph (DAG) consisting of Spouts and Bolts.

Applications are submitted in storm jar mode. Therefore, a function for creating a topology must be invoked in the main function, and the class to which the main function belongs must be specified in storm jar parameters.

#### Code Sample

The following code snippet belongs to the `main` method in the `WordCountTopology` class of the `com.huawei.storm.example.wordcount` package, and these code snippets are used to create and submit applications.

```
public static void main(String[] args)
 throws Exception
{
 TopologyBuilder builder = buildTopology();

 /*
 * Tasks can be submitted in the following three modes:
 * 1. Command line submitting. In this mode, a user must copy an application JAR package to a client
 and run related commands on the client.
 * 2. Remote submitting. In this mode, a user must package application JAR packages and execute the
 main method in IntelliJ IDEA.
 */
}
```

```

* 3. Local submitting. In this mode, a user must run an application for test on a local computer.
* The command line submitting and remote submitting modes support both security and normal
modes.
* The local submitting mode supports the normal mode only.
*
* A user can select only one mode for submitting a task. By default, the command line submitting
mode is used. To use another mode, delete code comments.
*/

cmdSubmit(builder, null);

//Remote submitting mode
//remoteSubmit(builder);

//Local submitting mode, used for test
//localSubmit(builder);
}

private static void cmdSubmit(TopologyBuilder builder, Config conf)
throws AlreadyAliveException, InvalidTopologyException, NotALeaderException,
AuthorizationException
{
 if (conf == null)
 {
 conf = new Config();
 }
 /**
 * Command line submitting mode
 * The procedures are as follows:
 * Package a JAR package and then submit the task in the client CLI.
 * In remote submitting mode, package application JAR packages and other externally dependent
JAR packages into one JAR package. Other externally dependent JAR packages are depended by user
programs, not provided by the example project.
 * Run the storm -jar command on the Storm client to submit the task.
 *
 * In a security environment, before submitting the task in the client CLI, run the kinit command to
log in in security mode.
 *
 * Run the following command:
 * ./storm jar ../example/example.jar com.huawei.storm.example.WordCountTopology
 */
 conf.setNumWorkers(1);
 StormSubmitter.submitTopology(TOPOLOGY_NAME, conf, builder.createTopology());
}

private static void localSubmit(TopologyBuilder builder)
throws InterruptedException
{
 Config conf = new Config();
 conf.setDebug(true);
 conf.setMaxTaskParallelism(3);
 LocalCluster cluster = new LocalCluster();
 cluster.submitTopology(TOPOLOGY_NAME, conf, builder.createTopology());
 Thread.sleep(10000);
 cluster.shutdown();
}

private static void remoteSubmit(TopologyBuilder builder)
throws AlreadyAliveException, InvalidTopologyException, NotALeaderException,
AuthorizationException,
IOException
{
 Config config = createConf();

 //Preparations in security mode
 if (isSecurityModel())
 {
 securityPrepare();
 }
}

```

```
}
String userJarFilePath = "User JAR package address";
System.setProperty(STORM_SUBMIT_JAR_PROPERTY, userJarFilePath);

cmdSubmit(builder, config);
}

private static TopologyBuilder buildTopology()
{
 TopologyBuilder builder = new TopologyBuilder();
 builder.setSpout("spout", new RandomSentenceSpout(), 5);
 builder.setBolt("split", new SplitSentenceBolt(), 8).shuffleGrouping("spout");
 builder.setBolt("count", new WordCountBolt(), 12).fieldsGrouping("split", new Fields("word"));
 return builder;
}
```

 **NOTE**

If ack is enabled for the topology, it is recommended that the number of ackers be less than or equal to the configured number of workers.

## 32.4 Running an Application

### 32.4.1 Packaging IntelliJ IDEA Code

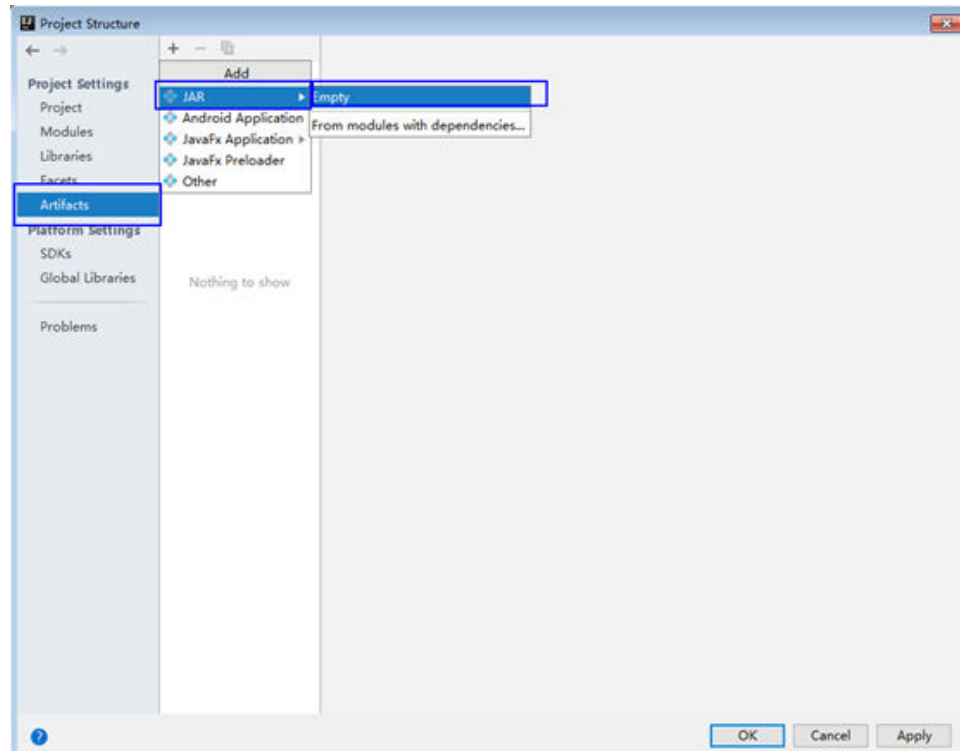
#### Scenario

Export a JAR package using IntelliJ IDEA and specify the name of the JAR package, for example, **storm-examples.jar**.

#### Procedure

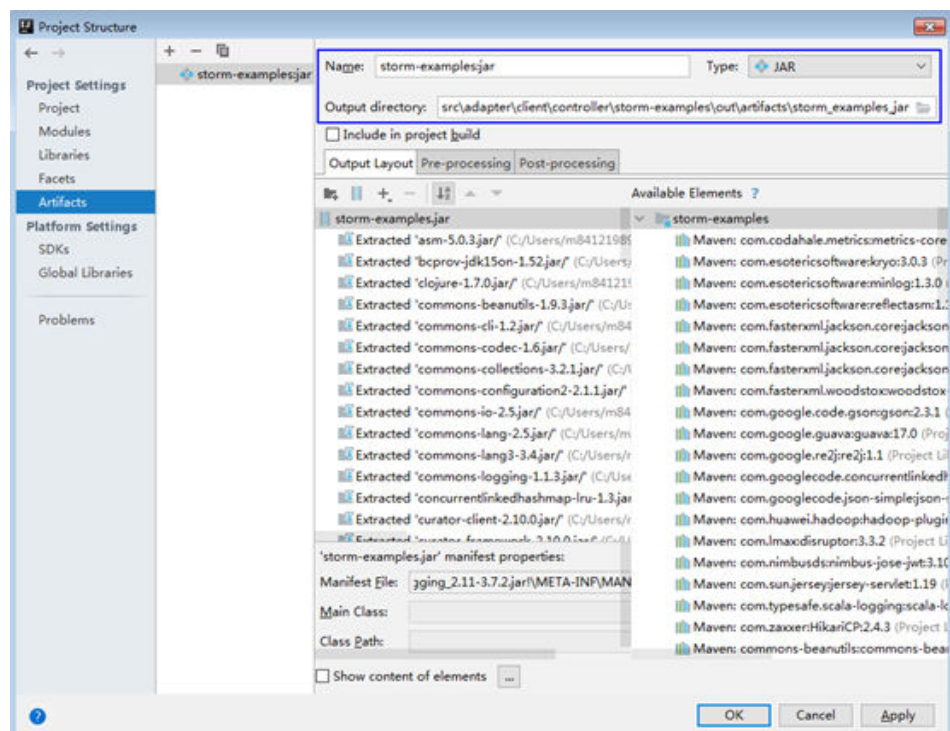
- Step 1** In IntelliJ IDEA, configure the Artifacts information for the project before the JAR package is generated.
1. Enter IntelliJ IDEA and choose **File > Project Structure**.
  2. On the **Project Structure** page, select **Artifacts**, click the plus sign (+), and choose **JAR > Empty**.

Figure 32-12 Adding Artifacts



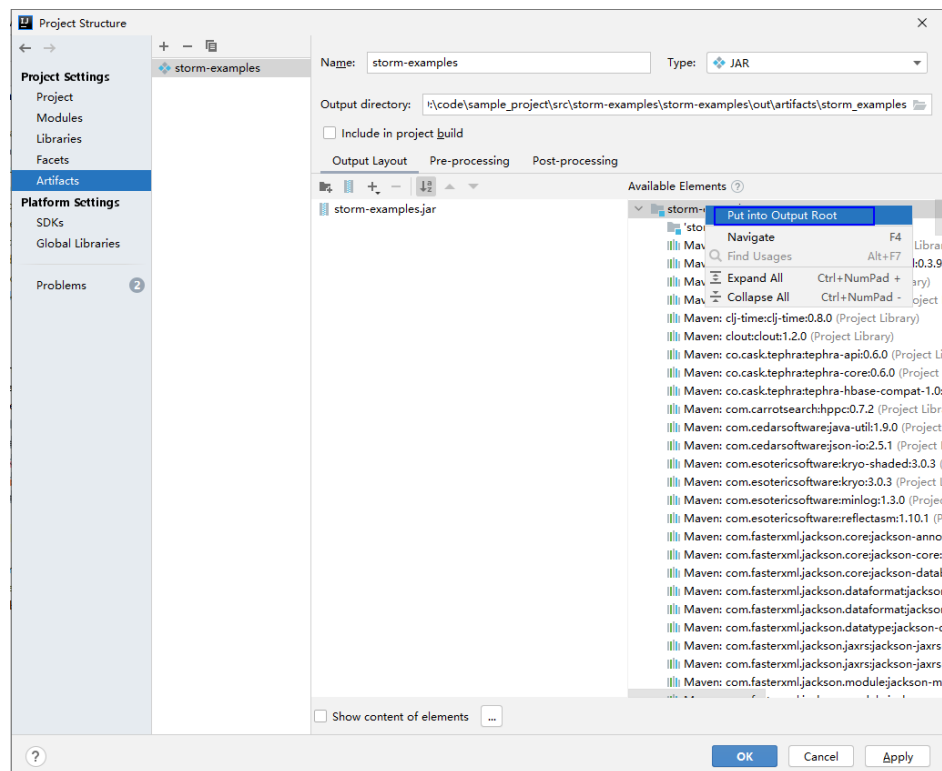
3. Set the name, type, and output path of the JAR package based on site requirements.

Figure 32-13 Configuring basic information



4. Choose 'storm-examples', right-click Put into Output Root, and click Apply.

Figure 32-14 Put into Output Root

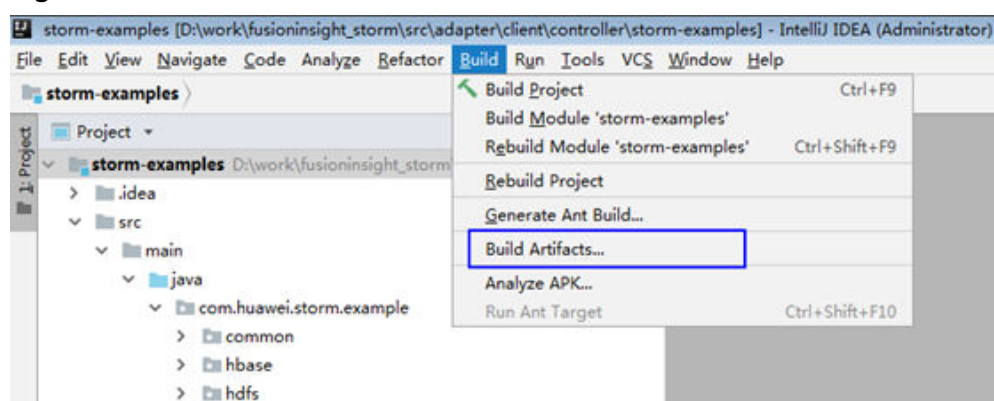


5. Click **OK** to complete the configuration.

**Step 2** Generate the JAR package.

1. Enter IntelliJ IDEA and choose **Build > Build Artifacts....**

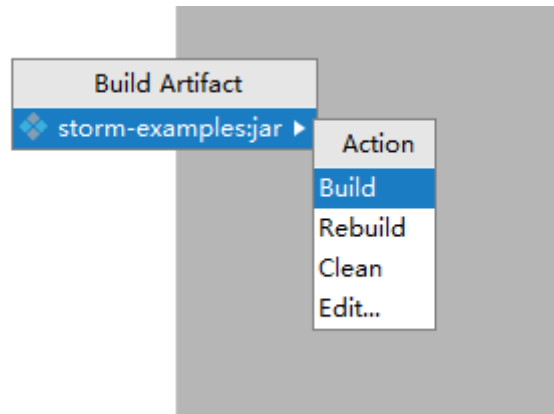
Figure 32-15 Build Artifacts



2. In the displayed interface, choose **storm-examples:jar > Build** to generate the JAR package.



Figure 32-16 Build



3. If log similar to the following is displayed in event logs, the JAR package is successfully generated. Obtain the JAR package from the **Output directory** configured shown in [Figure 32-13](#).

```
14:37 Compilation completed successfully in 25 s 991 ms
```

----End

## 32.4.2 Packaging Services

### 32.4.2.1 Overview

Packaging services is to combine the JAR package generated by IntelliJ IDEA code and the dependency JAR package of the project to export the **source.jar** package that can be submitted. The storm-jartool tool is required for service packaging. The tool can run on a Windows or Linux OS.

### 32.4.2.2 Packaging Services on a Linux OS

#### Scenario

Storm supports packaging in a Linux environment. You can upload JAR packages exported from IntelliJ IDEA and other related JAR packages to a Linux environment and perform packaging.

#### Prerequisites

- You have installed the Storm client.
- [Packaging IntelliJ IDEA Code](#) is complete.
- When the host where the Linux OS runs is not a node of the cluster, you are required to set the mapping between the host name and IP address in the **hosts** file of the node where the Linux OS runs. The host name must be correctly mapped to the IP address.

#### Procedure

- Step 1** Copy the JAR package exported from IntelliJ IDEA to a specified directory on the Linux client (for example, `/opt/jarsource`) .

- Step 2** If the service needs to access external components, obtain the configuration files and place them in the directory specified in [Step 1](#) For details about the dependency configuration files, see related development guidelines.
- Step 3** If the service needs to access external components, obtain the JAR packages and place them in the directory specified in [Step 1](#) For details about the dependency JAR packages, see the related development guidelines.
- Step 4** Run the packaging command in the Storm client installation directory **Storm/storm-1.2.1/bin** to package the preceding JAR packages into a complete service JAR package and place the package in the **/opt/jartarget** directory (which can be any empty directory). After the **sh storm-jartool.sh /opt/jarsource/ /opt/jartarget** command is executed, **source.jar** is generated in the **/opt/jartarget** directory.

----End

### 32.4.2.3 Packaging Services on a Windows OS

#### Scenario

Storm supports packaging in a Windows environment.

#### Prerequisites

[Packaging IntelliJ IDEA Code](#) is complete.

#### Procedure

- Step 1** Place the JAR package exported from IntelliJ IDEA to a specified directory (for example, **D:\source**).
- Step 2** Create the lib directory in the sample code directory **src/storm-examples/storm-examples**, copy the JAR package exported from IntelliJ IDEA to the lib directory, and decompress the package.
- Step 3** If the service needs to access external components, obtain the configuration files and place them in the lib directory specified in [Step 1](#) For details about the configuration files, see related development guidelines.
- Step 4** If the service needs to access external components, obtain the JAR packages and place them in the lib directory specified in [Step 1](#) For details about the dependency JAR packages, see the related development guidelines.
- Step 5** Find **storm-jartool.cmd** in the **tools** example project directory.
- Step 6** Double-click the packaging tool, enter the directory where the JAR packages to be packaged locate (**D:\source**) and press **Enter**, and enter the directory where the package to be generated locates (**D:\target**). The **source.jar** file is generated in the **D:\target** directory.

----End

### 32.4.3 Submitting a Topology

### 32.4.3.1 Submitting a Topology When a Client Is Installed on a Linux OS

#### Scenario

You can use storm commands to submit topologies in a Linux environment.

#### Prerequisites

- You have installed the Storm client.
- If the host where the client is installed is not a node in the cluster, set the mapping between the host name and the IP address in the **hostsfile** on the node where the client locates. The host name and IP address must be in one-to-one mapping.
- [Packaging Services](#) has been performed and **source.jar** has been generated.

#### Procedure

**Step 1** In security mode, perform security authentication first.

1. Initialize client environment variables.

Go to the installation directory **/opt/Storm\_client**, and run the following command to import environment variables:

```
source bigdata_env
```

2. Use the developer account created in *Preparing the Developer Account* section for secure login.

Run the **kinit** command to log in with security mode as a human-machine user.

```
kinit Username
```

For example, run the following command:

```
kinit developuser
```

Enter a password as prompted. If no error message appears, the Kerberos authentication is completed for the user.

**Step 2** Submit a topology. (Wordcount is used as an example. For details about other topologies, see related development guide.) Go to the **Storm/storm-1.2.1/bin** directory on the Streaming client, and run the **storm jar /opt/jartarget/source.jar com.huawei.storm.example.wordcount.WordCountTopology** command to submit the generated **source.jar** package. (If the package is generated on a Windows OS, upload the **source.jar** package to a specified directory on the Linux server, such as **/opt/jartarget**.)

**Step 3** Run the **storm list** command to view the submitted applications. If the word-count application can be viewed, the task is submitted successfully.

#### NOTE

If a service is set to the local mode and is submitted by using commands, ensure that the submitting environment is a normal one. Currently, services in local mode cannot be submitted by using commands in a security environment.

----End

## 32.4.3.2 Submitting a Topology When No Client Is Installed on a Linux OS

### Scenario

Storm supports topology running in a Linux environment where no Storm client is installed.

### Prerequisites

- Ensure that the time difference between the client and the MRS cluster is less than 5 minutes.
- If the host where the client is installed is not a node in the cluster, set the mapping between the host name and the IP address in the **hosts** file on the node where the client locates. The host name and IP address must be in one-to-one mapping.

### Procedure

**Step 1** Prepare for dependency JAR packages and configuration files.

In the Linux environment, create a directory, such as **/opt/test**, and create subdirectories **lib** and **src/main/resources/**. Upload the JAR packages in the **lib** folder of the example project to the **lib** directory in the Linux environment. Upload the configuration files in the **src/main/resources** folder of the example project to the **src/main/resources** directory in the Linux environment.

**Step 2** Modify the **WordCountTopology.java** class in the IntelliJ IDEA project and change the application submission mode to the **remoteSubmit** mode. Replace the user keytab file name, user principal name, and the JAR file address.

- Submit the application in **remoteSubmit** mode.

```
public static void main(String[] args)
 throws Exception
 {
 TopologyBuilder builder = buildTopology();

 /*
 * The task submitting modes are as follows:
 * 1. Command submitting mode: In this mode, you need to copy an application JAR package to
 the client machine and run related commands for submitting.
 * 2. Remote submitting mode: In this mode, you need to package related content into an
 application JAR package and run the main method in IntelliJ IDEA for submitting.
 * 3. Local submitting mode: In this mode, you need to run an application locally. This mode is
 usually used for tests.
 * The command submitting mode and remote submitting mode are supported in both the
 security and normal modes.
 * The local submitting mode is supported only in the normal mode.
 *
 * The user can only select one mode for task submitting. The command submitting mode is
 used by default. If another mode is used, delete the code comments.
 */

 submitTopology(builder, SubmitType.REMOTE);
 }
```

- Change the value of **userJarFilePath** to the specified directory **/opt/test/lib/example.jar** in the Linux environment.

```
private static void remoteSubmit(TopologyBuilder builder)
 throws AlreadyAliveException, InvalidTopologyException, NotALeaderException,
 AuthorizationException,
 IOException
```

```
{
 Config config = createConf();

 String userJarFilePath = "/opt/test/lib/example.jar ";
 System.setProperty(STORM_SUBMIT_JAR_PROPERTY, userJarFilePath);

 //Preparations in security mode
 if (isSecurityModel())
 {
 securityPrepare(config);
 }
 config.setNumWorkers(1);
 StormSubmitter.submitTopologyWithProgressBar(TOPOLOGY_NAME, config,
builder.createTopology());
}
```

- In security mode, make security authentication preparations, and change the values of **userKeyTablePath** and **userPrincipal** to the user keytab file name and principal obtained in step 2 of [Configuring and Importing Sample Projects](#) based on the actual situation.

```
private static void securityPrepare(Config config)
 throws IOException
{
 String userKeyTablePath =
 System.getProperty("user.dir") + File.separator + "src" + File.separator + "main" +
File.separator + "resources" + File.separator + "user.keytab";
 String userPrincipal = "StreamingDeveloper";
 String krbFilePath = System.getProperty("user.dir") + File.separator + "src" + File.separator +
"main" + File.separator + "resources" + File.separator + "krb5.conf";

 //Separators need to be replaced in Windows paths.
 userKeyTablePath = userKeyTablePath.replace("\\", "\\");
 krbFilePath = krbFilePath.replace("\\", "\\");

 String principalInstance =
String.valueOf(config.get(Config.STORM_SECURITY_PRINCIPAL_INSTANCE));
 LoginUtil.setKrb5Config(krbFilePath);
 LoginUtil.setZookeeperServerPrincipal("zookeeper/" + principalInstance);
 LoginUtil.setJaasFile(userPrincipal, userKeyTablePath);
}
```

**Step 3** Export the JAR package to the Linux environment.

- Package the IntelliJ IDEA code by referring to [Packaging IntelliJ IDEA Code](#) and name the JAR package example.jar.
- Copy the exported JAR package to the **/opt/test/lib** directory in the Linux environment.

**Step 4** Go to the **/opt/test** directory, and run the following command to run the JAR package:

```
java -classpath /opt/test/lib/*:/opt/test/src/main/resources
com.huawei.storm.example.wordcount.WordCountTopology
```

----End

### 32.4.3.3 Submitting a Topology in IntelliJ IDEA Remotely

#### Scenario

Storm supports remote topology submitting in IntelliJ IDEA. In current example code, only WordCountTopology can be submitted remotely. If you want to submit other topologies remotely, implement the remote submission function first. For details, see WordCountTopology.

## Prerequisites

- **Packaging IntelliJ IDEA Code** is complete.
- Ensure that the time difference between the client and the Storm cluster is less than 5 minutes.
- When the host where the Linux OS runs is not a node of the cluster, you are required to set the mapping between the host name and IP address in the **hosts** file of the node where the Linux OS runs. The host name must be correctly mapped to the IP address.

## Procedure

**Step 1** Modify the `WordCountTopology.java` class and change the application submission mode to the `remoteSubmit` mode. Replace the user keytab file name, user principal name, and the JAR file address.

- Submit the application in `remoteSubmit` mode.

```
public static void main(String[] args)
 throws Exception
{
 TopologyBuilder builder = buildTopology();

 /*
 * The task submitting modes are as follows:
 * 1. Command submitting mode: In this mode, you need to copy an application JAR package to
 * the client machine and run related commands for submitting.
 * 2. Remote submitting mode: In this mode, you need to package related content into an
 * application JAR package and run the main method in IntelliJ IDEA for submitting.
 * 3. Local submitting mode: In this mode, you need to run an application locally. This mode is
 * usually used for tests.
 * The command submitting mode and remote submitting mode are supported in both the
 * security and normal modes.
 * The local submitting mode is supported only in the normal mode.
 *
 * The user can only select one mode for task submitting. The command submitting mode is
 * used by default. If another mode is used, delete the code comments.
 */

 submitTopology(builder, SubmitType.REMOTE);
}
```

- Change the value of **userJarFilePath** to the actual topology JAR package address based on the actual situation.

```
private static void remoteSubmit(TopologyBuilder builder)
 throws AlreadyAliveException, InvalidTopologyException, NotALeaderException,
 AuthorizationException,
 IOException
{
 Config config = createConf();

 String userJarFilePath = "D:\\example.jar";
 System.setProperty(STORM_SUBMIT_JAR_PROPERTY, userJarFilePath);

 //Preparations in security mode
 if (isSecurityModel())
 {
 securityPrepare(config);
 }
 config.setNumWorkers(1);
 StormSubmitter.submitTopologyWithProgressBar(TOPOLOGY_NAME, config,
 builder.createTopology());
}
```

- In security mode, make security authentication preparations, and change the values of **userKeyTablePath** and **userPrincipal** to the user keytab file path

and principal obtained in step 2 of [Configuring and Importing Sample Projects](#) based on the actual situation.

```
private static void securityPrepare(Config config)
 throws IOException
{
 String userKeyTablePath =
 System.getProperty("user.dir") + File.separator + "src" + File.separator + "main" +
 File.separator + "resources" + File.separator + "user.keytab";
 String userPrincipal = "StreamingDeveloper";
 String krbFilePath = System.getProperty("user.dir") + File.separator + "src" + File.separator +
 "main" + File.separator + "resources" + File.separator + "krb5.conf";

 //Separators need to be replaced in Windows paths.
 userKeyTablePath = userKeyTablePath.replace("\\", "\\");
 krbFilePath = krbFilePath.replace("\\", "\\");

 String principlInstance =
 String.valueOf(config.get(Config.STORM_SECURITY_PRINCIPAL_INSTANCE));
 LoginUtil.setKrb5Config(krbFilePath);
 LoginUtil.setZookeeperServerPrincipal("zookeeper/" + principlInstance);
 LoginUtil.setJaasFile(userPrincipal, userKeyTablePath);
}
```

**Step 2** Execute the **Main** method of the **WordCountTopology.java** class to submit the application.

----End

## 32.4.4 Viewing Results

### Scenario

After a Storm application is run, you can log in to the Storm WebUI to check the running result.

### Procedure

**Step 1** Log in to FusionInsight Manager.

Enter the address in the address box of your browser. The address format is `https://FusionInsight Manager WebService floating IP address:28443/web`.

For example, enter `https://10.0.0.1:28443/web`.

**Step 2** Choose **Cluster > Name of the desired cluster > Services > Storm** to go to the Streaming WebUI.

**Step 3** On the Storm UI, click the word-count application to view the application running status, as shown in [Figure 32-17](#).

**Figure 32-17** Storm application execution page

### Topology summary

Name	Id	Owner	Status	Uptime	Num workers	Num executors
word-count	word-count-6-1536579754	stormkafka1	ACTIVE	1m 24s	1	26

### Topology actions

Activate	Deactivate	Rebalance	Kill	Debug	Stop Debug	Change Log Level
----------	------------	-----------	------	-------	------------	------------------

### Topology stats

Window	Emitted	Transferred	Complete latency (ms)
10m 0s	22500	22500	0
3h 0m 0s	22500	22500	0
1d 0h 0m 0s	22500	22500	0
All time	22500	22500	0

In **Topology stats**, the total volume of data transferred between operators in different time periods is displayed.

In **Spouts**, the total number of messages sent by the spout operator from the moment the operator is started till now is displayed. In **Bolts**, the total number of messages sent by the Count operator and the split operator is displayed, as shown in [Figure 32-18](#).

**Figure 32-18** Total volume of data sent by the Storm application operators

### Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Last error
spout	5	5	20940	20940	0.000	0	0	

### Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)
count	12	12	0	0	0.006	0.105	133920	0.086
split	8	8	133880	133880	0.005	0.670	20940	0.648

----End

## 32.5 More Information



## 32.5.1 Storm-Kafka Development Guideline

### Scenario

This topic describes how to use the Storm-Kafka toolkit to implement the interaction between Storm and Kafka. KafkaSpout and KafkaBolt are included. KafkaSpout enables Storm to read data from Kafka. KafkaBolt enables Storm to write data into Kafka.

The code example in this section is based on the new Kafka API and corresponds to `com.huawei.storm.example.kafka.NewKafkaTopology.java` in the IntelliJ IDEA project.

This topic applies only to the access between the Storm component and the Kafka component of MRS. Determine the versions of the jar packages described in this chapter based on the actual situation.

### Procedure for Developing an Application

**Step 1** Verify that the Storm and Kafka components have been installed and are running properly.

**Step 2** Obtain the sample project folder `storm-examples` in the `src\storm-examples` directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#). Import `storm-examples` to the IntelliJ IDEA development environment. For details, see [Environment Preparation](#).

**Step 3** Install the Storm client in Linux OS.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

**Step 4** If security services are enabled in the cluster, obtain a human-machine user from the administrator for login to the FusionInsight Manager platform and authentication, and obtain the keytab file of the user.

#### NOTE

- The obtained user must belong to the `storm` group and `kafka` group.
- The default validity period of a user password is 90 days. Therefore, the validity period of the obtained keytab file is 90 days. To prolong the validity period of the keytab file, modify the user password policy and obtain the keytab file again.

**Step 5** Download and install the Kafka client.

----End

### Code Sample

Create a topology. (Change the IP addresses and ports to the actual ones.)

```
public static void main(String[] args) throws Exception {

 // Set the topology
 Config conf = new Config();
```

```
// Configure the security plug-in.
setSecurityPlugin(conf);

if (args.length >= 2) {
// If the default keytab file name has been changed, configure the new keytab file name.
conf.put(Config.TOPOLOGY_KEYTAB_FILE, args[1]);
}

// Define KafkaSpout
KafkaSpout kafkaSpout = new KafkaSpout<String, String>(
getKafkaSpoutConfig(getKafkaSpoutStreams()));

// CountBolt
CountBolt countBolt = new CountBolt();
//SplitBolt
SplitSentenceBolt splitBolt = new SplitSentenceBolt();

// KafkaBolt configuration information
KafkaBolt<String, String> kafkaBolt = new KafkaBolt<String, String>();
kafkaBolt.withTopicSelector(new DefaultTopicSelector(OUTPUT_TOPIC))
.withTupleToKafkaMapper(
new FieldNameBasedTupleToKafkaMapper("word", "count"));
kafkaBolt.withProducerProperties(getKafkaProducerProps());

// Define the topology.
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("kafka-spout", kafkaSpout, 10);
builder.setBolt("split-bolt", splitBolt, 10).shuffleGrouping("kafka-spout", STREAMS[0]);
builder.setBolt("count-bolt", countBolt, 10).fieldsGrouping(
"split-bolt", new Fields("word"));
builder.setBolt("kafka-bolt", kafkaBolt, 10).shuffleGrouping("count-bolt");

// Submit the topology.
StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
}
```

#### NOTE

If the cluster domain name is modified, set the Kerberos domain name in the Kafka consumer/producer attribute to the actual domain name of the cluster, for example, `props.put(KERBEROS_DOMAIN_NAME, "hadoop.hadoop1.com")`.

## Running the Application and Viewing Results

**Step 1** Export the local JAR package. For details, see [Packaging IntelliJ IDEA Code](#).

**Step 2** Obtain the related configuration files. The methods are described as follows:

- Security mode: See [Step 4](#) to obtain the keytab file.
- Normal mode: None

**Step 3** Obtain the following JAR packages.

Go to the **Kafka/kafka/libs** directory on the installed Kafka client, and obtain the following JAR packages:

- kafka\_<version>.jar
- scala-library-<version>.jar
- scala-logging\_2.11-3.7.2.jar
- metrics-core-<version>.jar
- kafka-clients-<version>.jar
- zookeeper-<version>.jar

Obtain the following JAR file from the **streaming-cql-*<HD-Version>/lib*** directory on the Storm client:

- storm-kafka-client-*<version>.jar*
- storm-kafka-*<version>.jar*
- slf4j-api-*<version>.jar*
- guava-*<version>.jar*
- json-simple-*<version>.jar*
- curator-client-*<version>.jar*
- curator-framework-*<version>.jar*
- curator-recipes-*<version>.jar*

**Step 4** Combine the JAR packages obtained in the preceding steps and export a complete service JAR package. For details, see [Packaging Services](#).

**Step 5** Go to the directory where the Kafka client locates in the Linux system, use the Kafka client to create the topic used by the topology in the **Kafka/kafka/bin** directory. Run the following command:

```
./kafka-topics.sh --create --topic input --partitions 2 --replication-factor 2 --zookeeper {ip:port}/kafka
```

```
./kafka-topics.sh --create --topic output --partitions 2 --replication-factor 2 --zookeeper {ip:port}/kafka
```

 **NOTE**

- The variable appended to **--zookeeper** specifies the ZooKeeper address. You must set the ZooKeeper address to the ZooKeeper address configured during cluster installation.
- Safe mode requires Kafka administrator user to create Topic.

**Step 6** Submit the topology on a Linux OS.

The submission command example is as follows (the topology name is **kafka-test**):

```
storm jar /opt/jartarget/source.jar
com.huawei.storm.example.kafka.SimpleKafkaTopology kafka-test
```

 **NOTE**

- Before submitting **source.jar** in security mode, ensure that the Kerberos security login is implemented, and the login user and the user of the uploaded keytab file are the same user in keytab mode.
- In security mode, Kafka requires the user to have access permission on related topics. Therefore, you need to log in to the server from the cluster where Kafka is located as the Kafka administrator, and then run the **kafka-acls.sh** command to assign permission to the user. After successful permission assignment, you need to log in to the server as the user who submits the task and submit the topology. For details about Kafka user permission assignment, see [Kafka Development Guide > More Information](#).

**Step 7** After the topology is successfully submitted, send data to Kafka and check whether related information is generated.

Go to the directory where the Kafka client locates in the Linux system, start the consumer in the **Kafka/kafka/bin** directory, and check whether data is generated. Run the following command:

```
./kafka-console-consumer.sh --bootstrap-server {ip:port} --topic output --
consumer.config ../config/consumer.properties
```

Go to the directory where the Kafka client is located in the Linux system, start the producer in the **Kafka/kafka/bin** directory, and write data into Kafka. Run the following command:

```
./kafka-console-producer.sh --broker-list {ip:port} --topic input --
producer.config ../config/producer.properties
```

Write test data into input, and check whether related data is generated in output. If yes, the Storm-Kafka topology is executed successfully.

----End

## 32.5.2 Storm-JDBC Development Guideline

### Scenario

This topic describes how to use the open-source Storm-JDBC toolkit to implement the interaction between Storm and JDBC. `JdbcInsertBolt` and `JdbcLookupBolt` are included. `JdbcLookupBolt` is used to query data from the database. `JdbcInsertBolt` is used to store data to the database. In addition, `JdbcLookupBolt` and `JdbcInsertBolt` can be used to process data using the data processing logic.

This topic applies only to the access between the Storm component and the JDBC component. Determine the versions of the jar packages described in this chapter based on the actual situation.

### Procedure for Developing an Application

**Step 1** Verify that the Storm component has been installed and is running properly.

**Step 2** Obtain the sample project folder `storm-examples` in the `src\storm-examples` directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#). Import `storm-examples` to the IntelliJ IDEA development environment. For details, see [Environment Preparation](#).

**Step 3** Install the Storm client in Linux OS.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

----End

### Configuring the Database - Configuring the Derby Database

**Step 1** Download a database. Select the most appropriate database based on the actual scenario.

In this topic, the Derby database is used as an example. The Derby database is a Java-based small-sized open-source database that is easy to use and suitable to most applications.

**Step 2** Obtain the Derby database. Download the Derby database package of the latest version from the official website, upload the database package to the Linux client, for example, `/opt`, and decompress the package.

**Step 3** In the Derby installation directory, go to the `bin` directory, and run the following commands:

```
export DERBY_INSTALL=/opt/db-derby-10.12.1.1-bin
export CLASSPATH=$DERBY_INSTALL/lib/derbytools.jar:$DERBY_INSTALL/lib\derbynet.jar:
export DERBY_HOME=/opt/db-derby-10.12.1.1-bin
. setNetworkServerCP
./startNetworkServer -h Host name
```

**Step 4** Run the `./ij` command, and enter `connect 'jdbc:derby://Host name:1527/example;create=true'`; to create the connection.

**Step 5** After the database is connected, run SQL statements to create table ORIGINAL and table GOAL and insert a group of data into table ORIGINAL. The statement examples are as follows (the table names can be customized):

```
CREATE TABLE GOAL(WORD VARCHAR(12),COUNT INT);
CREATE TABLE ORIGINAL(WORD VARCHAR(12),COUNT INT);
INSERT INTO ORIGINAL VALUES('orange',1),('pineapple',1),('banana',1),
('watermelon',1);
----End
```

## Code Sample

SimpleJDBCTopology code sample: (Change the IP addresses and ports to the actual ones.)

```
public class SimpleJDBCTopology
{
 private static final String WORD_SPOUT = "WORD_SPOUT";
 private static final String COUNT_BOLT = "COUNT_BOLT";
 private static final String JDBC_INSERT_BOLT = "JDBC_INSERT_BOLT";
 private static final String JDBC_LOOKUP_BOLT = "JDBC_LOOKUP_BOLT";
 @SuppressWarnings ("unchecked")
 public static void main(String[] args) throws Exception{
 //connectionProvider configuration
 Map hikariConfigMap = Maps.newHashMap();
 hikariConfigMap.put("dataSourceClassName", "org.apache.derby.jdbc.ClientDataSource");
 hikariConfigMap.put("dataSource.serverName", "192.168.0.1");//Set this parameter to the actual IP
 address.
 hikariConfigMap.put("dataSource.portNumber", "1527");//Set this parameter to the actual port number.

 hikariConfigMap.put("dataSource.databaseName", "example");
 hikariConfigMap.put("dataSource.user", "app");
 hikariConfigMap.put("dataSource.password", "mine");
 hikariConfigMap.put("connectionTestQuery", "select COUNT from GOAL"); //The table name must be
 consistent with that during table creation.
 Config conf = new Config();

 ConnectionProvider connectionProvider = new HikariCPCConnectionProvider(hikariConfigMap);
 //JdbcLookupBolt instantiation
 Fields outputFields = new Fields("WORD", "COUNT");
```

```
List<Column> queryParamColumns = Lists.newArrayList(new Column("WORD", Types.VARCHAR));
SimpleJdbcLookupMapper jdbcLookupMapper = new SimpleJdbcLookupMapper(outputFields,
queryParamColumns);
String selectSql = "select COUNT from ORIGINAL where WORD = ?";
JdbcLookupBolt wordLookupBolt = new JdbcLookupBolt(connectionProvider, selectSql,
jdbcLookupMapper);
//JdbcInsertBolt instantiation
String tableName = "GOAL";
JdbcMapper simpleJdbcMapper = new SimpleJdbcMapper(tableName, connectionProvider);
JdbcInsertBolt userPersistenceBolt = new JdbcInsertBolt(connectionProvider,
simpleJdbcMapper).withTableName("GOAL").withQueryTimeoutSecs(30);
WordSpout wordSpout = new WordSpout();TopologyBuilder builder = new TopologyBuilder();
builder.setSpout(WORD_SPOUT, wordSpout);
builder.setBolt(JDBC_LOOKUP_BOLT, wordLookupBolt, 1).fieldsGrouping(WORD_SPOUT,new
Fields("WORD"));
builder.setBolt(JDBC_INSERT_BOLT, userPersistenceBolt,1).fieldsGrouping(JDBC_LOOKUP_BOLT,new
Fields("WORD"));StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
}
}
```

## Running and Submitting the Topology

**Step 1** Export the local JAR package. For details, see [Packaging IntelliJ IDEA Code](#).

**Step 2** Obtain the following JAR packages.

- Go to the **lib** directory in the installed database, and obtain the following JAR packages:
  - derbyclient.jar
  - derby.jar
- Obtain the following JAR file from the **streaming-cql-*<HD-Version>/lib*** directory on the Storm client:
  - storm-jdbc-*<version>*.jar
  - guava-*<version>*.jar
  - commons-lang3-*<version>*.jar
  - commons-lang-*<version>*.jar
  - HikariCP-*<version>*.jar

**Step 3** Combine the JAR packages obtained in the preceding steps and export a complete service JAR package. For details, see [Packaging Services](#).

**Step 4** Run a command to submit the topology. The submission command example is as follows (the topology name is **jdbc-test**):

```
storm jar /opt/jartarget/source.jar
com.huawei.storm.example.jdbc.SimpleJDBCTopology jdbc-test
```

----End

## Viewing Results

After the topology is submitted, go to the database and check whether data is inserted into the related tables.

Run the **select \* from goal;** statement to query data in table **goal**. If data is inserted into table GOAL, the topology is executed successfully.

## 32.5.3 Storm-HDFS Development Guideline

### Scenario

This topic applies only to the interaction between Storm and HDFS. Determine the versions of the jar packages described in this chapter based on the actual situation.

Log in in security mode is classified into ticket login and keytab file login, and the procedures for these two login modes are the same. The ticket login mode is an open-source capability and requires manual ticket uploading, which may cause reliability and usability problems. Therefore, the keytab file login mode is recommended.

### Procedure for Developing an Application

- Step 1** Verify that the Storm and HDFS components have been installed and are running properly.
- Step 2** Import storm-examples to the IntelliJ IDEA development environment. For details, see [Environment Preparation](#).
- Step 3** If security services are enabled in the cluster, perform the related configuration based on the login mode.
  - keytab mode: Obtain a human-machine user from the administrator for login to the FusionInsight Manager platform and authentication, and obtain the keytab file of the user.
  - Ticket mode: Obtain a human-machine user from the administrator for subsequent secure login, enable the renewable and forwardable functions of the Kerberos service, set the ticket update interval, and restart Kerberos and related components.

#### NOTE

- The obtained user must belong to the storm group.
- The default validity period of a user password is 90 days. Therefore, the validity period of the obtained keytab file is 90 days. To prolong the validity period of the keytab file, modify the user password policy and obtain the keytab file again.
- The parameters for enabling the renewable and forwardable functions and setting the ticket update interval are on the **System** tab of the Kerberos service configuration page. The ticket update interval can be set to **kdc\_renew\_lifetime** or **kdc\_max\_renewable\_life** based on the actual situation.

**Step 4** Download and install the HDFS client.

**Step 5** Obtain the related configuration files using the following method.

Go to the **/opt/clientHDFS/HDFS/hadoop/etc/hadoop** directory on the installed HDFS client, and obtain configuration files **core-site.xml** and **hdfs-site.xml**.

In keytab mode, obtain the keytab file by following [Step 3](#). In ticket mode, no extra configuration file is required.

 NOTE

The obtained keytab file is named as **user.keytab** by default. A user can directory change the file name as required. However, the user must upload the changed file name as a parameter when submitting a task.

**Step 6** Obtain the related JAR packages.

- Go to the **HDFS/hadoop/share/hadoop/common/lib** directory on the installed HDFS client, and obtain the following JAR packages:
  - commons-cli-<version>.jar
  - commons-io-<version>.jar
  - commons-lang-<version>.jar
  - commons-lang3-<version>.jar
  - commons-collections-<version>.jar
  - commons-configuration2-<version>.jar
  - commons-logging-<version>.jar
  - guava-<version>.jar
  - hadoop-\*.jar
  - protobuf-java-<version>.jar
  - jackson-databind-<version>.jar
  - jackson-core-<version>.jar
  - jackson-annotations-<version>.jar
  - re2j-<version>.jar
  - jaeger-core-<version>.jar
  - opentracing-api-<version>.jar
  - opentracing-noop-<version>.jar
  - opentracing-tracerresolver-<version>.jar
  - opentracing-util-<version>.jar
- Go to the **HDFS/hadoop/share/hadoop/common** directory on the installed HDFS client, and obtain the **hadoop-\*.jar** package.
- Go to the **HDFS/hadoop/share/hadoop/client** directory on the installed HDFS client, and obtain the **hadoop-\*.jar** package.
- Go to the **HDFS/hadoop/share/hadoop/hdfs** directory on the installed HDFS client, obtain the **hadoop-hdfs-\*.jar** package.
- Obtain the following JAR packages from the sample project `/src/storm-examples/storm-examples/lib`:
  - storm-hdfs-<version>.jar
  - storm-autocreds-<version>.jar

----End

## IntelliJ IDEA Code Sample

Create a topology.

```
public static void main(String[] args) throws Exception
{
```



```
TopologyBuilder builder = new TopologyBuilder();

// Separator. Use | to replace the default , to separate fields in tuple.
// Mandatory HdfsBolt parameter
RecordFormat format = new DelimitedRecordFormat()
 .withFieldDelimiter("|");

// Synchronize policy. Synchronize the file system for every 1000 tuples.
// Mandatory HdfsBolt parameter

SyncPolicy syncPolicy = new CountSyncPolicy(1000);

// File size cyclic policy. If the size of a file reaches 5 MB, the file is written from the beginning.
// Mandatory HdfsBolt parameter
FileRotationPolicy rotationPolicy = new FileSizeRotationPolicy(5.0f, Units.MB);

// Objective file written to hdfs
// Mandatory HdfsBolt parameter
FileNameFormat fileNameFormat = new DefaultFileNameFormat()
 .withPath("/user/foo/");

//Create HdfsBolt.
HdfsBolt bolt = new HdfsBolt()
 .withFsUrl(DEFAULT_FS_URL)
 .withFileNameFormat(fileNameFormat)
 .withRecordFormat(format)
 .withRotationPolicy(rotationPolicy)
 .withSyncPolicy(syncPolicy);

//Spout generates a random statement.
builder.setSpout("spout", new RandomSentenceSpout(), 1);
builder.setBolt("split", new SplitSentence(), 1).shuffleGrouping("spout");
builder.setBolt("count", bolt, 1).fieldsGrouping("split", new Fields("word"));

//Add the plugin required for kerberos authentication to the list. The security mode is mandatory.

setSecurityConf(conf,AuthenticationType.KEYTAB);

Config conf = new Config();
//Write the plugin list configured on the client to a specific config item. The security mode is mandatory.
conf.put(Config.TOPOLOGY_AUTO_CREDENTIALS, auto_tgt);

if(args.length >= 2)
{
 //The default keytab file name is changed by the user. Specify the new keytab file name as a
parameter.
 conf.put(Config.STORM_CLIENT_KEYTAB_FILE, args[1]);
}

//Run a command to submit the topology.
StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
}
```

#### NOTE

The target file path of Storm cannot be in an SM4 encrypted HDFS partition.

## Running the Application and Viewing Results

**Step 1** Export the local JAR package. For details, see [Packaging IntelliJ IDEA Code](#).

**Step 2** Combine the configuration files and JAR packages obtained respectively in [Step 5](#) and [Step 6](#), and export a complete service JAR package. For details, see [Packaging Services](#).

**Step 3** Run a command to submit the topology.

In keytab mode, if the user changes the keytab file name, for example, **huawei.keytab**, the changed keytab file name must be added to the command as a parameter for description. The submission command examples are as follows (the topology name is **hdfs-test**):

```
storm jar /opt/jartarget/source.jar
com.huawei.storm.example.hdfs.SimpleHDFSTopology hdfs-test
huawei.keytab
```

 **NOTE**

Before submitting **source.jar**, ensure that the Kerberos security login is implemented, and the login user and the user of the uploaded keytab file are the same user in keytab mode.

**Step 4** After the topology is submitted successfully, log in to the HDFS cluster to view the topology.

**Step 5** To log in with ticket mode, perform the following operations to regularly upload a ticket. The interval for uploading the ticket depends on the deadline for updating the ticket.

1. Add the following content to a new line at the end of the **Storm/storm-1.2.1/conf/storm.yaml** file in the Storm client installation directory:  
**topology.auto-credentials:**  
**- org.apache.storm.security.auth.kerberos.AutoTGT**
2. Run the **./storm upload-credentials hdfs-test** command.

----End

## 32.5.4 Storm-HBase Development Guideline

### Scenario

This topic applies only to the interaction between Storm and HBase of MRS. Determine the versions of the jar packages described in this chapter based on the actual situation.

Security login modes include ticket login and keytab file login, and the procedures for these two login modes are the same. The ticket login mode is an open-source capability and requires manual ticket uploading to prevent ticket expiration, which may cause reliability and usability problems. Therefore, the keytab file login mode is recommended.

### Procedure for Developing an Application

- Step 1** Verify that the Storm and HBase components have been installed and are running properly.
- Step 2** Import storm-examples to the IntelliJ IDEA development environment. For details, see [Environment Preparation](#).
- Step 3** If security services are enabled in the cluster, perform the related configuration based on the login mode.

- keytab mode: Obtain a **Human-Machine** user from the administrator for login to the FusionInsight Manager platform and authentication, and obtain the keytab file of the user.
- Ticket mode: Obtain a **Human-Machine** user from the administrator for subsequent secure login, enable the renewable and forwardable functions of the Kerberos service, set the ticket update interval, and restart Kerberos and related components.

 **NOTE**

- The obtained user must belong to the storm group.
- The default validity period of a user password is 90 days. Therefore, the validity period of the obtained keytab file is 90 days. To prolong the validity period of the keytab file, modify the user password policy and obtain the keytab file again.
- The parameters for enabling the renewable and forwardable functions and setting the ticket update interval are on the **System** tab of the Kerberos service configuration page. The ticket update interval can be set to **kdc\_renew\_lifetime** or **kdc\_max\_renewable\_life** based on the actual situation.

**Step 4** Download and install the HBase client.

**Step 5** Obtain the related configuration files using the following method.

Go to the **/opt/clientHbase/HBase/hbase/conf** directory on the installed HBase client, and obtain configuration files **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml**.

In keytab mode, obtain the keytab file by following [Step 3](#). In ticket mode, no extra configuration file is required.

 **NOTE**

The obtained keytab file is named as **user.keytab** by default. A user can directory change the file name as required. However, the user must upload the changed file name as a parameter when submitting a task.

**Step 6** Obtain the related JAR packages.

- Go to the **HBase/hbase/lib** directory on the installed HBase client, and obtain the following JAR packages:
  - hbase-\*.jar
  - hadoop-\*.jar
  - jackson-core-asl-<version>.jar
  - jackson-mapper-asl-<version>.jar
  - commons-cli-<version>.jar
  - commons-io-<version>.jar
  - commons-lang-<version>.jar
  - commons-lang3-<version>.jar
  - commons-collections-<version>.jar
  - commons-configuration2-<version>.jar
  - guava-<version>.jar
  - protobuf-java-<version>.jar
  - netty-all-<version>.jar

- zookeeper-<version>.jar
- zookeeper-jute-<version>.jar
- metrics-core-<version>.jar
- commons-validator-<version>.jar
- Go to the **HBase/hbase/lib/client-facing-thirdparty** directory in the HBase client installation directory, and obtain the **commons-logging-<version>.jar** package.
- Go to the **HBase/hbase/lib/jdbc** directory in the HBase client installation directory, and obtain the **htrace-core-<version>-incubating.jar** package and the **htrace-core4-<version>-incubating.jar** package.
- Obtain the following JAR packages from the sample project /src/storm-examples/storm-examples/lib:
  - storm-hdfs-<version>.jar
  - storm-autocreds-<version>.jar

----End

## IntelliJ IDEA Code Sample

Create a topology.

```
public static void main(String[] args) throws Exception
{
 Config conf = new Config();
 //Add the plugin required for kerberos authentication to the list. The security mode is mandatory.
 setSecurityConf(conf,AuthenticationType.KEYTAB);

 if(args.length >= 2)
 {
 //The default keytab file name is changed by the user. Specify a new keytab file name as a parameter.
 conf.put(Config.STORM_CLIENT_KEYTAB_FILE, args[1]);
 }
 //hbase client configuration. Only hbase.rootdir configuration item is provided, which is optional.
 Map<String, Object> hbConf = new HashMap<String, Object>();
 if(args.length >= 3)
 {
 hbConf.put("hbase.rootdir", args[2]);
 }
 //Mandatory parameter. If it is not set, it is left blank.
 conf.put("hbase.conf", hbConf);

 //Spout generates a random word.
 WordSpout spout = new WordSpout();
 WordCounter bolt = new WordCounter();

 //HbaseMapper, which is used for parsing tuple content.
 SimpleHBaseMapper mapper = new SimpleHBaseMapper()
 .withRowKeyField("word")
 .withColumnFields(new Fields("word"))
 .withCounterFields(new Fields("count"))
 .withColumnFamily("cf");

 //HBaseBolt, the first parameter is a table name.
 //withConfigKey("hbase.conf")Transfer the hbase client configuration to HBaseBolt.
 HBaseBolt hbase = new HBaseBolt("WordCount", mapper).withConfigKey("hbase.conf");

 // wordSpout ==> countBolt ==> HBaseBolt
 TopologyBuilder builder = new TopologyBuilder();

 builder.setSpout(WORD_SPOUT, spout, 1);
```

```
builder.setBolt(COUNT_BOLT, bolt, 1).shuffleGrouping(WORD_SPOUT);
builder.setBolt(HBASE_BOLT, hbase, 1).fieldsGrouping(COUNT_BOLT, new Fields("word"));
//Run a command to submit the topology.
StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
}
```

## Running the Application and Viewing Results

**Step 1** Export the local JAR package. For details, see [Packaging IntelliJ IDEA Code](#).

**Step 2** Combine the configuration files and JAR packages obtained respectively in [Step 5](#) and [Step 6](#), and export a complete service JAR package. For details, see [Packaging Services](#).

**Step 3** Run a command to submit the topology.

In keytab mode, if the user changes the keytab file name, for example, **huawei.keytab**, the changed keytab file name must be added to the command as a parameter for description. The submission command example is as follows (the topology name is **hbase-test**):

```
storm jar /opt/jartarget/source.jar
com.huawei.storm.example.hbase.SimpleHBaseTopology hbase-test
huawei.keytab
```

### NOTE

- Before submitting **source.jar** in security mode, ensure that the Kerberos security login is implemented, and the login user and the user of the uploaded keytab file are the same user in keytab mode.
- HBaseBolt in the preceding example does not provide the function for creating tables. Therefore, you must verify that necessary tables exist in HBase. If the tables do not exist, run the **create 'WordCount', 'cf'** statement to manually create HBase shell tables.
- In HBase security mode, a user must have the permission to access related tables, column families, and columns. Therefore, the user must log in to the HBase cluster as an HBase administrator, run the **grant** command in HBase shell to apply for table access permission, such as WordCount, for the user, and submit the topology as the user.

**Step 4** After the topology is submitted successfully, log in to the HBase cluster to view the topology.

**Step 5** To log in with ticket mode, perform the following operations to regularly upload a ticket. The interval for uploading the ticket depends on the deadline for updating the ticket.

1. Add the following content to a new line at the end of the **Storm/storm-1.2.1/conf/storm.yaml** file in the Storm client installation directory:

```
topology.auto-credentials:
- org.apache.storm.security.auth.kerberos.AutoTGT
```

2. Run the **./storm upload-credentials hbase-test** command.

----End

## 32.5.5 Flux Development Guideline

### Scenario

This topic applies only to the scenario of submitting and deploying a topology using the Flux framework in the Storm component of MRS. Determine the

versions of the jar packages described in this chapter based on the actual situation.

The Flux framework is a framework provided by Storm. This framework is used to improve the topology deployment usability. Using the Flux framework, users can use a YAML file to define and deploy a topology and run the **storm jar** command to submit the topology. This mode facilitates topology deployment and submitting, and reduces the service development cycle.

## Basic Syntax Description

Using Flux to define a topology can be classified into two scenarios: defining a new topology and defining an existing topology.

### 1. Using Flux to define a new topology

Using Flux to define a topology indicates using a YAML file to describe a topology. A complete topology definition must contain the following parts:

- Topology name
- List of components used for defining the topology
- Topology configuration
- Topology definition, including the spout list, bolt list, and stream list

Code sample for defining the topology name:

```
name: "yaml-topology"
```

Code sample for defining the component list:

```
#Simple component definition
components:
- id: "stringScheme"
 className: "org.apache.storm.kafka.StringScheme"

#Use a constructor to define a component.
- id: "defaultTopicSelector"
 className: "org.apache.storm.kafka.bolt.selector.DefaultTopicSelector"
 constructorArgs:
 - "output"

#Reference parameters as input arguments in a constructor, and use the `ref` tag to describe the
reference.
#When using a reference, ensure that the referenced object has been defined.
- id: "stringMultiScheme"
 className: "org.apache.storm.spout.SchemeAsMultiScheme"
 constructorArgs:
 - ref: "stringScheme"

#Reference the configuration items in the specified properties file as input arguments in a constructor,
and use the `${}` tag to describe the reference.
#If the properties file is referenced, use the --filter my-prop.properties mode to specify the path of the
properties file when you run the storm jar command to submit the topology.
- id: "zkHosts"
 className: "org.apache.storm.kafka.ZkHosts"
 constructorArgs:
 - "${kafka.zookeeper.root.list}"

#Reference environment variables as input arguments in a constructor, and use the `${ENV-[NAME]}`
tag to describe the reference.
#NAME must be a defined environment variable.
- id: "zkHosts"
 className: "org.apache.storm.kafka.ZkHosts"
 constructorArgs:
 - "${ENV-ZK_HOSTS}"
```

```
#Use the `properties` keyword to initialize the internal private variables.
- id: spoutConfig
className: "org.apache.storm.kafka.SpoutConfig"
constructorArgs:
- ref: "zkHosts"
- "input"
- "/kafka/input"
- "myId"
properties:
- name: "scheme"
ref: "stringMultiScheme"
```

#### Code sample for defining the topology configuration:

```
config:
#Simple configuration item
topology.workers: 1

#If the configuration item value is a list, use `[]` to indicate it.
topology.auto-credentials: ["class1","class2"]

#The configuration item value is in the map structure.
kafka.broker.properties:
metadata.broker.list: "${metadata.broker.list}"
producer.type: "async"
request.required.acks: "0"
serializer.class: "kafka.serializer.StringEncoder"
```

#### Code sample for defining the spout/bolt list:

```
#Define the spout list.
spouts:
- id: "spout1"
className: "org.apache.storm.kafka.KafkaSpout"
constructorArgs:
- ref: "spoutConfig"
parallelism: 1

#Define the bolt list.
bolts:
- id: "bolt1"
className: "com.huawei.storm.example.hbase.WordCounter"
parallelism: 1

#Use a method to initialize an object, and the keyword is `configMethods`.
- id: "bolt2"
className: "org.apache.storm.hbase.bolt.HBaseBolt"
constructorArgs:
- "WordCount"
- ref: "mapper"
configMethods:
- name: "withConfigKey"
args: ["hbase.conf"]
parallelism: 1
```

#### Code sample for defining the stream list:

```
#To define the stream mode, you must specify the grouping mode. The keyword is `grouping`, and
keywords for grouping methods provided currently are as follows:
#`ALL`, `CUSTOM`, `DIRECT`, `SHUFFLE`, `LOCAL_OR_SHUFFLE`, `FIELDS`, `GLOBAL`, and `NONE`.
#`CUSTOM` is used for a customized group.

#For the definition of a simple stream, the grouping mode is SHUFFLE.
streams:
- name: "spout1 --> bolt1"
from: "spout1"
to: "bolt1"
grouping:
type: SHUFFLE

#If the grouping mode is FIELDS, parameters must be entered.
- name: "bolt1 --> bolt2"
from: "bolt1"
```

```

to: "bolt2"
grouping:
type: FIELDS
args: ["word"]

#If the grouping mode is CUSTOM, you must specify a customized grouping class.
- name: "bolt-1 --> bolt2"
from: "bolt-1"
to: "bolt-2"
grouping:
type: CUSTOM
customClass:
className: "org.apache.storm.testing.NGrouping"
constructorArgs:
- 1

```

## 2. Using Flux to define an existing topology

If a topology already exists (for example, a topology has already been defined by Java code), you can still use the Flux framework to submit and deploy the topology. In this situation, you must use the `getTopology()` method in the current topology definition (for example, `MyTopology.java`). The definition in Java is as follows:

```

public StormTopology getTopology(Config config)
or
public StormTopology getTopology(Map<String, Object> config)

```

In this situation, you can use the following YAML file to define the topology:

```

name: "existing-topology"#You can specify the topology name to any value.
topologySource:
className: "custom-class"#Specify the client class.

```

You can specify another method name to obtain `StormTopology` (non-`getTopology()` method). The YAML file example is as follows:

```

name: "existing-topology"
topologySource:
className: "custom-class "
methodName: "getTopologyWithDifferentMethodName"

```

### NOTE

The specified method must accept an input parameter of the `Map<String, Object>` type or the `Config` type and return an object of the `org.apache.storm.generated.StormTopology` type. This method is the same as the `getTopology()` method.

## Procedure for Developing an Application

- Step 1** Verify that the Storm component has been installed and is running properly. If the services need to connect to other components, install the required components and ensure that the components are running properly.
- Step 2** Import `storm-examples` to the IntelliJ IDEA development environment. For details, see [Configuring and Importing Sample Projects](#).
- Step 3** Develop client services. For details, see the related YAML application examples in the `src/main/resources/flux-examples` directory of the `storm-examples` project.
- Step 4** Obtain the related configuration files.



 NOTE

For details, this step applies only to the scenarios when other components, such as HDFS and HBase, need to be accessed for service requirements. For details about how to obtain the related configuration files, see [Storm-HDFS Development Guideline](#) or [Storm-HBase Development Guideline](#). If the services do not require the related configuration files, skip this step.

**Step 5** Obtain the related JAR packages.

- Obtain the following JAR file from the **streaming-cql-*<HD-Version>/lib*** directory on the Storm client:  
flux-core-*<version>*.jar  
flux-wrappers-*<version>*.jar
- Obtain other related JAR packages for the services. For example, to obtain the related JAR packages required for accessing HDFS, see [Step 6](#). Other scenarios are similar.

----End

**The Flux configuration file example is as follows:**

It is a complete YAML file example for accessing the Kafka service.

```
name: "simple_kafka"

components:
- id: "zkHosts" #Object name
 className: "org.apache.storm.kafka.ZkHosts" #Complete class name
 constructorArgs: #Constructor
 - "${kafka.zookeeper.root.list}" #Constructor parameter

- id: "stringScheme"
 className: "org.apache.storm.kafka.StringScheme"

- id: "stringMultiScheme"
 className: "org.apache.storm.spout.SchemeAsMultiScheme"
 constructorArgs:
 - ref: "stringScheme" #A reference is used, and the value is stringScheme that has been defined.

- id: spoutConfig
 className: "org.apache.storm.kafka.SpoutConfig"
 constructorArgs:
 - ref: "zkHosts" #A reference is used.
 - "input"
 - "/kafka/input"
 - "myId"
 properties: #Use properties to set the private variable whose name is "scheme" in this object.
 - name: "scheme"
 ref: "stringMultiScheme"

- id: "defaultTopicSelector"
 className: "org.apache.storm.kafka.bolt.selector.DefaultTopicSelector"
 constructorArgs:
 - "output"

- id: "fieldNameBasedTupleToKafkaMapper"
 className: "org.apache.storm.kafka.bolt.mapper.FieldNameBasedTupleToKafkaMapper"
 constructorArgs:
 - "words" #The first input argument in the constructor
 - "count" #The second input argument in the constructor

config:
topology.workers: 1 #Set the number of workers of the topology to 1.
```

```
kafka.broker.properties: #Set the parameters related to Kafka, and the values are in the map structure.
metadata.broker.list: "${metadata.broker.list}"
producer.type: "async"
request.required.acks: "0"
serializer.class: "kafka.serializer.StringEncoder"

spouts:
- id: "kafkaSpout" #Spout name
 className: "org.apache.storm.kafka.KafkaSpout" #Spout class name
 constructorArgs: #Use a constructor to perform initialization.
- ref: "spoutConfig" #Reference parameters as input arguments in a constructor.
 parallelism: 1 #Set the concurrency of the spout to 1.

bolts:
- id: "splitBolt"
 className: "com.huawei.storm.example.common.SplitSentenceBolt"
 parallelism: 1

- id: "countBolt"
 className: "com.huawei.storm.example.kafka.CountBolt"
 parallelism: 1

- id: "kafkaBolt"
 className: "org.apache.storm.kafka.bolt.KafkaBolt"
 configMethods: #Invoke an internal method of the object to initialize the object.
- name: "withTopicSelector" #Name of the invoked internal method
 args: #Parameter required by the internal method
- ref: "defaultTopicSelector" #Only one input argument is set, and it is referenced.
- name: "withTupleToKafkaMapper" #Invoke the second internal method.
 args:
- ref: "fieldNameBasedTupleToKafkaMapper"

#Define the data stream.
streams:
- name: "kafkaSpout --> splitBolt" #Name of the first data stream, which is only for display
 from: "kafkaSpout" #Data stream start, whose value is kafkaSpout defined in spouts
 to: "splitBolt" #Data stream end, whose value is splitBolt defined in bolts
 grouping: #Define the grouping mode.
 type: LOCAL_OR_SHUFFLE #The grouping mode is local_or_shuffle.

- name: "splitBolt --> countBolt" #The second data stream
 from: "splitBolt"
 to: "countBolt"
 grouping:
 type: FIELDS #The grouping mode is fields.
 args: ["word"] #Parameters must be entered for the fields mode.

- name: "countBolt --> kafkaBolt" #The third data stream
 from: "countBolt"
 to: "kafkaBolt"
 grouping:
 type: SHUFFLE #The grouping mode is shuffle, and no parameter needs to be entered.
```

## Running the Application and Viewing Results

- Step 1** Export the local JAR package. For details, see [Packaging IntelliJ IDEA Code](#).
- Step 2** Combine the configuration files and JAR packages obtained respectively in [Step 4](#) and [Step 5](#), and export a complete service JAR package. For details, see [Packaging Services](#).
- Step 3** Copy the developed YAML file and the related **properties** file to any directory on the host where the Storm client is located, for example, **/opt**.
- Step 4** Run the following command to submit the topology:
 

```
storm jar /opt/jartarget/source.jar org.apache.storm.flux.Flux --
remote /opt/my-topology.yaml
```

If the services are set to be started locally, run the following command to submit the topology:

```
storm jar /opt/jartarget/source.jar org.apache.storm.flux.Flux --local /opt/my-topology.yaml
```

#### NOTE

Submit local services by running commands only in a normal mode environment.

If the **properties** file is used, run the following command to submit the topology:

```
storm jar /opt/jartarget/source.jar org.apache.storm.flux.Flux --remote /opt/my-topology.yaml --filter /opt/my-prop.properties
```

**Step 5** After the topology is submitted successfully, log in to the Storm UI to view the topology.

----End

## 32.5.6 External Interfaces

- Versions of interfaces adopted by Storm-HDFS are consistent with those in the open-source community. For details, see: <https://github.com/apache/storm/tree/v1.2.1/external/storm-hdfs>.
- Versions of interfaces adopted by Storm-HBase are consistent with those in the open-source community. For details, see: <https://github.com/apache/storm/tree/v1.2.1/external/storm-hbase>.
- Versions of interfaces adopted by Storm-Kafka are consistent with those in the open-source community. For details, see: <https://github.com/apache/storm/tree/v1.2.1/external/storm-kafka>.
- Versions of interfaces adopted by Storm-JDBC are consistent with those in the open-source community. For details, see: <https://github.com/apache/storm/tree/v1.2.1/external/storm-jdbc>.
- Versions of interfaces adopted by Storm-Solr are consistent with those in the open-source community. For details, see: <https://github.com/apache/storm/tree/v1.2.1/external/storm-solr>.

## 32.5.7 FAQ

### 32.5.7.1 How Do I Use IDEA to Remotely Debug Services?

#### Question

How do I use IDEA to remotely debug services after submitting services by the Storm client?

#### Answer

The following uses WordCount as an example to demonstrate how to use IDEA to remotely debug services.

**Step 1** Log in to FusionInsight Manager and choose **Cluster** > *Name of the desired cluster* > **Services** > **Storm**. On the page that is displayed, click the

**Configurations** tab. In the search box, search for **nimbus.task.timeout.secs** and **supervisor.worker.start.timeout.secs** and increase their values. You are advised to adjust the values to the maximum ones. Add **-Xdebug -Xrunjwp:transport=dt\_socket,address=5055,suspend=n,server=y** right after the existing value of **WORKER\_GC\_OPTS**. Save the configuration and restart related instances.

 **NOTE**

Before debugging the Storm program, you need to modify specified parameters on the server side. These parameters take effect after you restart the server. You are advised to debug the program in the test environment.

**Step 2** After submitting a topology, enter the **Topology** page on the Storm UI. In the **Topology** page that is displayed, click the component to be debugged to enter its debugging page.

**Figure 32-19** Component debugging interface

Id	Executors	Tasks	Emitted
spout	5	5	591500

Showing 1 to 1 of 1 entries

## Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred
count	12	12	3771500	0
split	8	8	3776500	3776500

**Step 3** On the component debugging interface, obtain the IP address of the host where the worker process runs. If the host has multiple IP addresses, select any one of them.

**Figure 32-20** Obtaining the IP addresses of the host where worker runs

### Executors (All time)

Id	Uptime	Host	Port	Actions
[24-24]	4h 44m 50s	189-39-172-183	29300	files
[25-25]	4h 40m 59s	189-39-172-217	29300	files
[26-26]	4h 44m 53s	189-39-173-167	29300	files
[27-27]	4h 44m 50s	189-39-172-183	29300	files
[28-28]	4h 40m 59s	189-39-172-217	29300	files

Showing 1 to 5 of 5 entries

**Step 4** Open the IDEA project. On the menu bar, choose **Run > Edit Configurations**.

**Step 5** In the window that is displayed, click the plus sign (+) in the upper left corner. In the drop-down list, select **Remote**. Then select the path where the source code module to be debugged resides and configure the **Host** and **Port** parameters as shown in [Figure 32-21](#).

**Host** indicates the IP address of the host where worker runs and **Port** indicates the number of the port to be debugged. Ensure that the port is not occupied on the host.

**Figure 32-21** Configuring parameters

### Executors (All time)

Id	Uptime	Host	Port	Actions
[24-24]	4h 44m 50s	192-168-172-183	29300	files
[25-25]	4h 40m 59s	192-168-172-217	29300	files
[26-26]	4h 44m 53s	192-168-172-168	29300	files
[27-27]	4h 44m 50s	192-168-172-183	29300	files
[28-28]	4h 40m 59s	192-168-172-217	29300	files

Showing 1 to 5 of 5 entries

#### NOTE

When the port number changes, the value added to **WORKER\_GC\_OPTS** changes too. For example, if the port number is **8011**, the value changes to **-Xdebug -Xrunjdpw:transport=dt\_socket,address=8011,suspend=n,server=y**.

**Step 6** Set the debugging breakpoint.

Click in the blank area on the left of the IDEA code editing window to set corresponding code line breakpoint.

Figure 32-22 Setting the breakpoint

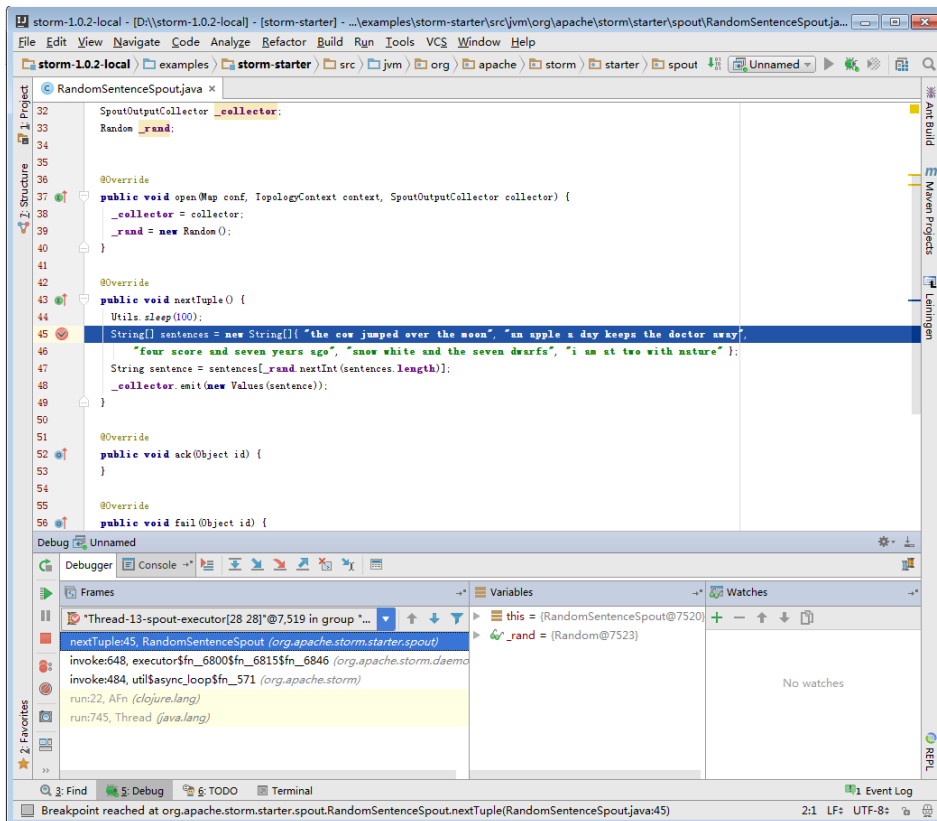
```
@Override
public void nextTuple() {
 Utils.sleep(100);
 String[] sentences = new String[]{"the cow jumped over the moon", "an apple
 "four score and seven years ago", "snow white and the seven dwarfs:
 String sentence = sentences[_rand.nextInt(sentences.length)];
 _collector.emit(new Values(sentence));
}

@Override
public void ack(Object id) {
}
```

**Step 7** Start debugging.

On the menu bar, choose **Run > Debug'Unnamed'** to open the debugging window and start debugging, including step-by-step debugging, checking the call stack, and tracing the variable value as shown in **Figure 32-23**.

Figure 32-23 Debugging



----End

## 32.5.7.2 How Do I Handle the Error "Command line is too long" Reported When Main Is Executed for Remote Topology Submission in IntelliJ IDEA

### Question

When the **Main** method is executed to remotely submit a topology in IntelliJ IDEA, the following error is reported:

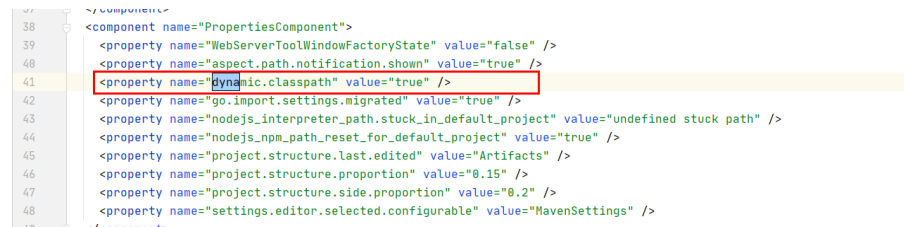
Command line is too long. Shorten command line for ServiceStarter or also for Application default configuration.

### Answer

**Step 1** Open the `.idea\workspace.xml` file in the project.

**Step 2** Find the `<component name="PropertiesComponent">` tag and add `<property name="dynamic.classpath" value="true" />`, as shown in [Figure 32-24](#).

**Figure 32-24** Modifying the `.idea\workspace.xml` file



```
38 <component>
39 <component name="PropertiesComponent">
40 <property name="WebServerToolWindowFactoryState" value="false" />
41 <property name="aspect.path.notification.shown" value="true" />
42 <property name="dynamic.classpath" value="true" />
43 <property name="go.import.settings.migrated" value="true" />
44 <property name="nodejs_interpreter_path.stuck_in_default_project" value="undefined stuck path" />
45 <property name="nodejs_npm_path_reset_for_default_project" value="true" />
46 <property name="project.structure.last.edited" value="Artifacts" />
47 <property name="project.structure.proportion" value="0.15" />
48 <property name="project.structure.side.proportion" value="0.2" />
49 <property name="settings.editor.selected.configurable" value="MavenSettings" />
</component>
```

----End

# 33 Storm Development Guide (Normal Mode)

---

## 33.1 Overview

### 33.1.1 Application Development Overview

#### Intended Audience

This document is provided for users who want to implement Storm secondary development. This document is intended for development personnel who are experienced in Java development.

#### Introduction

Storm is a distributed, reliable, and fault-tolerant data stream processing system. Storm delegates work tasks to components of different types, and each component processes a specific simple task. Storm processes big data streams in real time and unlimited data streams in reliable mode.

Storm applies to real-time analytic, online machine learning, continuous computation, and distributed Extract, Transform, and Load (ETL). It is scalable and fault-tolerant, and is easy to set up and operate. This ensures data processing.

Storm has the following features:

- **Wide application**
- **High scalability**
- **Free from data loss**
- **High fault tolerance**
- **Language independence**
- **Ease of construction and control**



## 33.1.2 Common Concepts

### Topology

A computing stream chart, in which each node contains processing logic and the lines between nodes specify data flow between nodes.

### Spout

A component that generates source data streams in a Topology. A Spout reads data from an external data source and converts the data into source data inside a Topology.

### Bolt

A component that receives data from a Topology and then processes data. A Bolt can perform operations, such as filtering, executing functions, combination, and writing data into a database.

### Tuple

Basic unit for transferring messages once.

### Stream

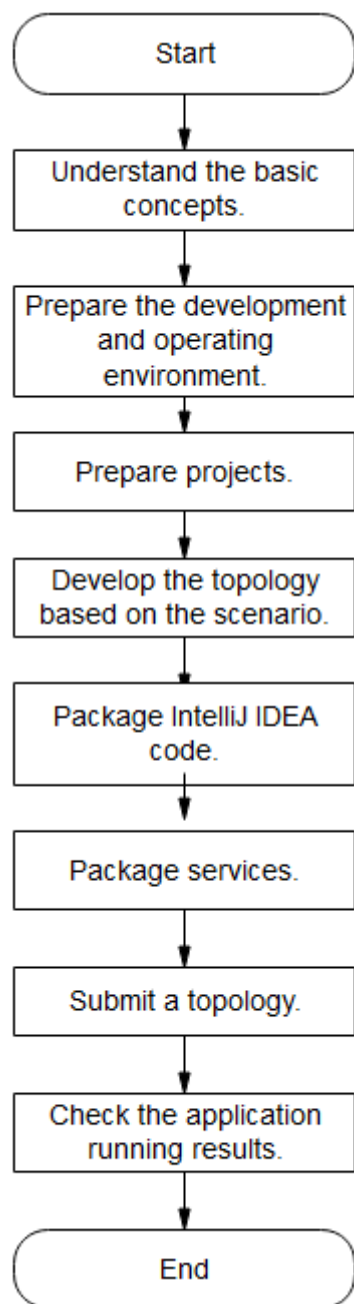
A set of (infinite) elements, each of which belongs to a same schema. Each element is related to the logic time, that is, a Stream has the Tuple and Time attributes. Any elements can be expressed in the format of `Element<Tuple,Time>`, in which Tuple includes the data structure and content, and Time is the logic time of data.

## 33.1.3 Development Process

This document describes Storm topology development based on the Java API.

[Figure 33-1](#) shows the development process.

**Figure 33-1** Topology development process



**Table 33-1** Description about Storm application development process

Phase	Description	Reference Document
Understand the basic concepts.	Before application development, understand common concepts about Storm, understand the scenario requirements, and topologies.	<a href="#">Common Concepts</a>

Phase	Description	Reference Document
Prepare the development and operating environment.	The Java language is recommended for Storm application development. The IntelliJ IDEA tool can be used.  The Storm running environment is a Storm client. Install and configure the client according to the guide.	<a href="#">Preparing for Development and Operating Environment</a>
Prepare projects.	Storm provides sample projects for different scenarios. You can import a sample project to learn the application.	<a href="#">Configuring and Importing Sample Projects</a>
Develop topologies based on scenarios.	The Storm topology construction and Spout/Bolt development process are provided.	<a href="#">Developing an Application</a>
Package the IntelliJ IDEA code.	Storm example programs are running in the Linux OS. IntelliJ IDEA code needs to be packaged into a JAR package.	<a href="#">Packaging IntelliJ IDEA Code</a>
Package services.	Combine the JAR package generated from the IntelliJ IDEA code and the project dependent JAR package to generate the <b>source.jar</b> package that can be submitted.	<a href="#">Packaging Services</a>
Submit a topology.	Submit the developed application for running.	<a href="#">Submitting a Topology</a>
View application running results.	Check the program running result after submitting topologies.	<a href="#">Viewing Results</a>

## 33.2 Environment Preparation

## 33.2.1 Environment Preparation Overview

Prepare clients for developing and submitting applications. Generally, applications are developed in Windows and submitted in Linux.

## 33.2.2 Preparing for Development and Operating Environment

### Preparing Development Environment

**Table 33-2** describes the environment required for secondary development.

**Table 33-2** Development environment

Item	Description
OS	<ul style="list-style-type: none"> <li>• Development environment: Windows OS. Windows 7 or later is supported.</li> <li>• Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li> </ul>
JDK installation	<p>Basic configuration for the development and operating environment. The version requirements are as follows: The server and client support only built-in OpenJDK, version: 1.8.0_272, and therefore a JDK replacement is not allowed. Customers' applications that need to reference the JAR packages of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK.</p> <ul style="list-style-type: none"> <li>• For x86 nodes that run clients, the following JDKs can be used: <ul style="list-style-type: none"> <li>- Oracle JDK versions: 1.8</li> <li>- IBM JDK versions: 1.8.5.11</li> </ul> </li> <li>• For nodes that run TaiShan and clients, the following JDK can be used: <ul style="list-style-type: none"> <li>- OpenJDK: 1.8.0_272</li> </ul> </li> </ul> <p><b>NOTE</b> The server supports only TLS V1.2 or later to ensure security. By default, IBM JDK supports only TLS V1.0. If IBM JDK is used, setting <code>com.ibm.jsse2.overrideDefaultTLS</code> to true enables TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls">https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/jsse2Docs/matchsslcontext_tls.html#matchsslcontext_tls</a>.</p>

Item	Description
IntelliJ IDEA installation and configuration	<p>Tool used for developing Storm applications. Requirements are as follows: JDK 1.8 is used. IntelliJ IDEA 2019.1 or other compatible versions are used.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li> <li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li> <li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li> </ul>
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
7-zip	It is a tool used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-Zip 16.04 is supported.

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. and choose **Cluster > Dashboard > More > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Storm\config**. Obtain the Storm-related configuration file.

**Table 33-3** describes the main configuration files.

**Table 33-3** Configuration file

Document Name	Function
storm.yaml	Configures Storm Cluster Information.
streaming-site.xml	Configures Storm parameters.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, C:\WINDOWS\system32\drivers\etc\hosts.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client on the node. For example, the client installation directory is /opt/client.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

- b. [Log in to the FusionInsight Manager portal](#). Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Storm/config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, /opt/client/conf.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is /tmp/FusionInsight-Client on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
scp Storm/config/* root@IP address of the client node:/opt/client/conf
```

[Table 33-4](#) describes the main configuration files.

**Table 33-4** Configuration file

Document Name	Function
storm.yaml	Configures Storm Cluster Information.
streaming-site.xml	Configures Storm parameters.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

## 33.2.3 Configuring and Importing Sample Projects

### Background

The Storm development example project is included in the Storm client installation directory. Import the project to IntelliJ IDEA for learning.

### Prerequisites

Ensure that the difference between the local PC time and the cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the cluster time in the lower-right corner on FusionInsight Manager.

### Procedure

- Step 1** Obtain the sample project folder **storm-examples** in the `src\storm-examples` directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the cluster configuration file obtained in section [Preparing an Operating Environment](#) to the `storm-examples\src\main\resources` directory of the sample project.

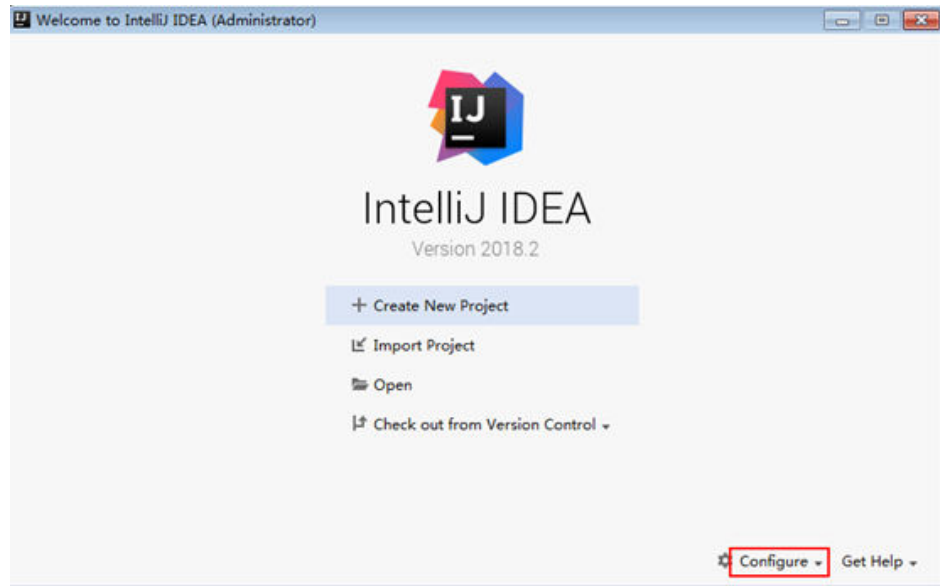
 **NOTE**

- To submit a topology when no client is installed in Windows or Linux, you need to place both `streaming-site.xml` and `storm.yaml` in the `storm-examples\src\main\resources` directory of the sample project.
- To submit a topology when the client is installed in Linux, you only need to place `streaming-site.xml` in the `storm-examples\src\main\resources` directory of the sample project.

- Step 3** After installing the IntelliJ IDEA and JDK tools, configure JDK in IntelliJ IDEA.

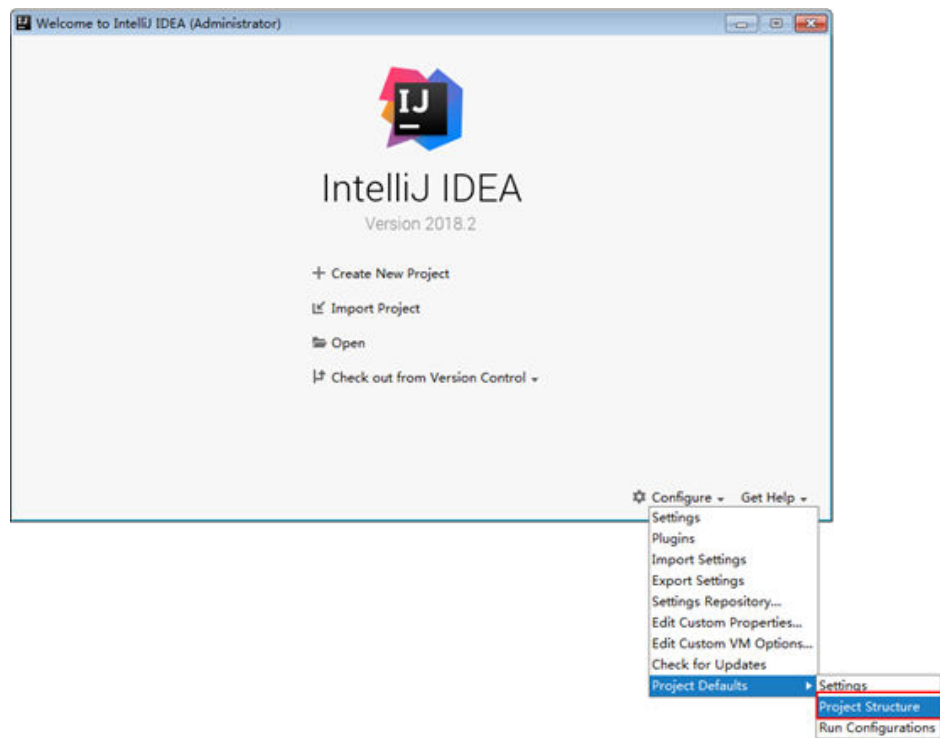
1. Open IntelliJ IDEA and click **Configure**.

Figure 33-2 Quick Start



2. Choose **Project Defaults > Project Structure**.

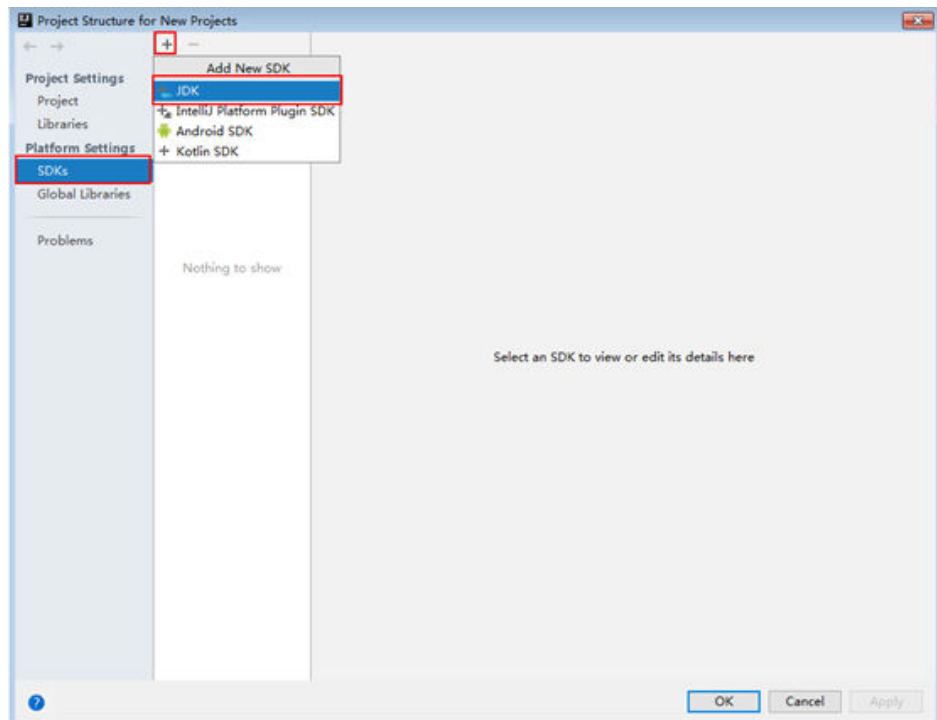
Figure 33-3 Configure



3. In the displayed **Project Structure for New Projects** interface, click **SDKs**. Then click the plus sign (+) and choose **JDK**.

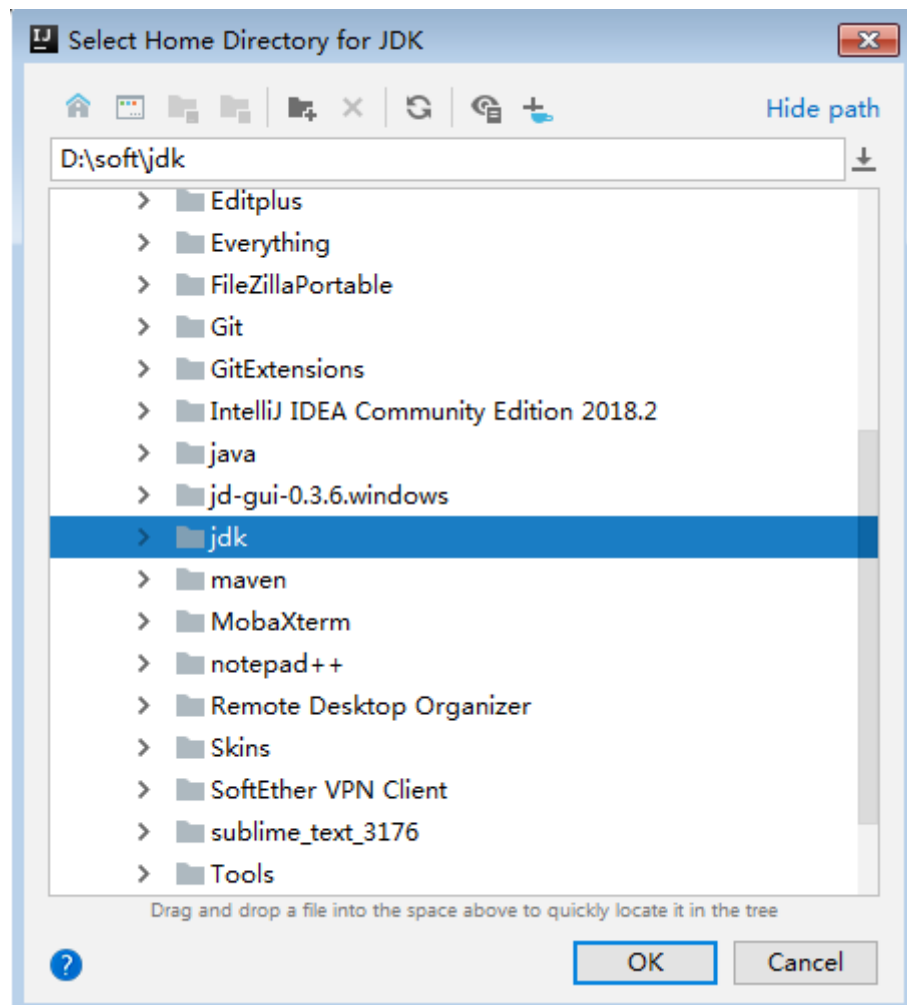


**Figure 33-4** Project Structure for New Projects



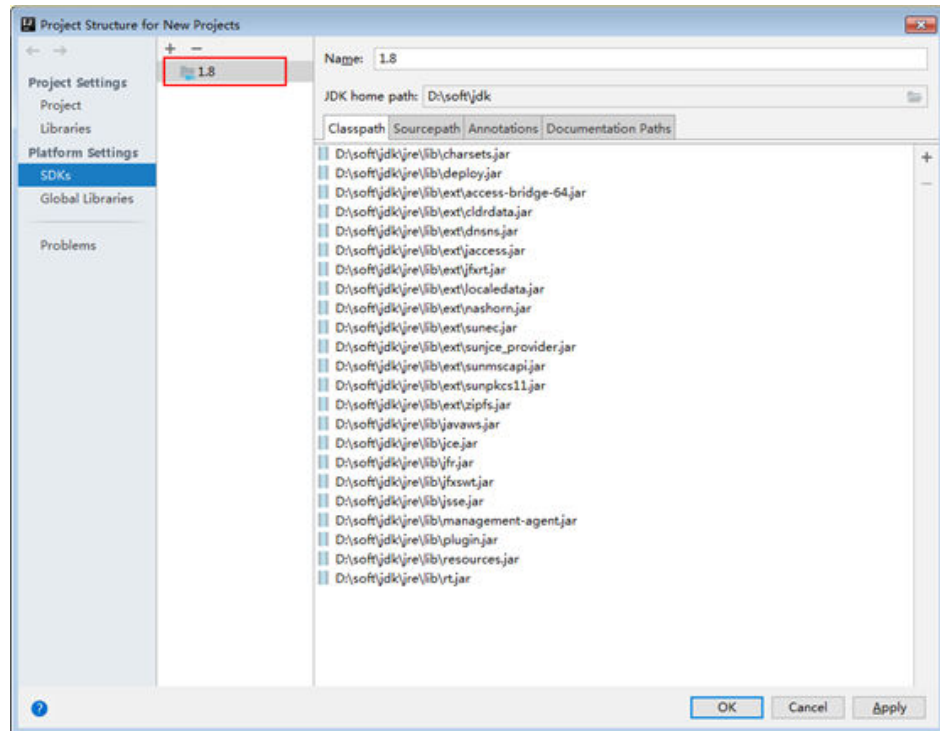
4. In the displayed **Select Home Directory for JDK** interface, select the JDK directory, and click **OK**.

**Figure 33-5** Select Home Directory for JDK



5. Click **OK**.

Figure 33-6 Completing the JDK configuration

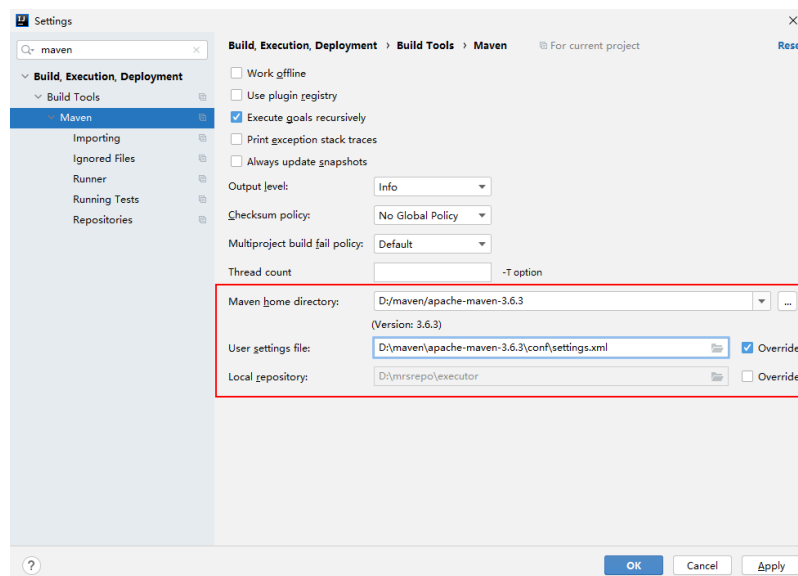


**Step 4** In the application development environment, import the sample project to the IntelliJ IDEA development environment.

1. Configure the IntelliJ IDEA Maven project environment.

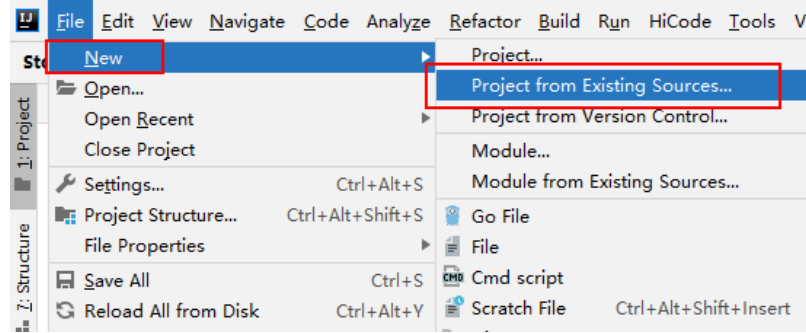
Choose **File > Settings** and search for **Maven**. Configure the Maven project, and choose **Apply > OK**.

Figure 33-7 Configuring the IntelliJ IDEA Maven project environment



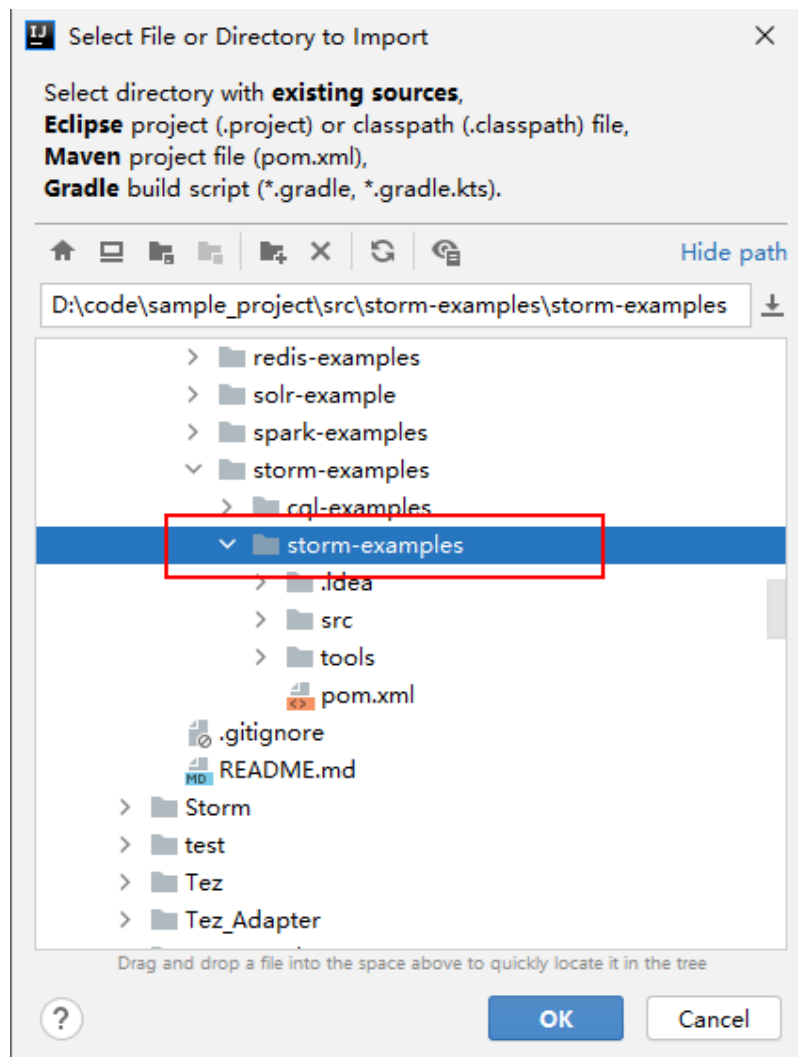
2. Choose **File > New > Project from Existing Sources....**

Figure 33-8 The Project from Existing Sources.... page is displayed



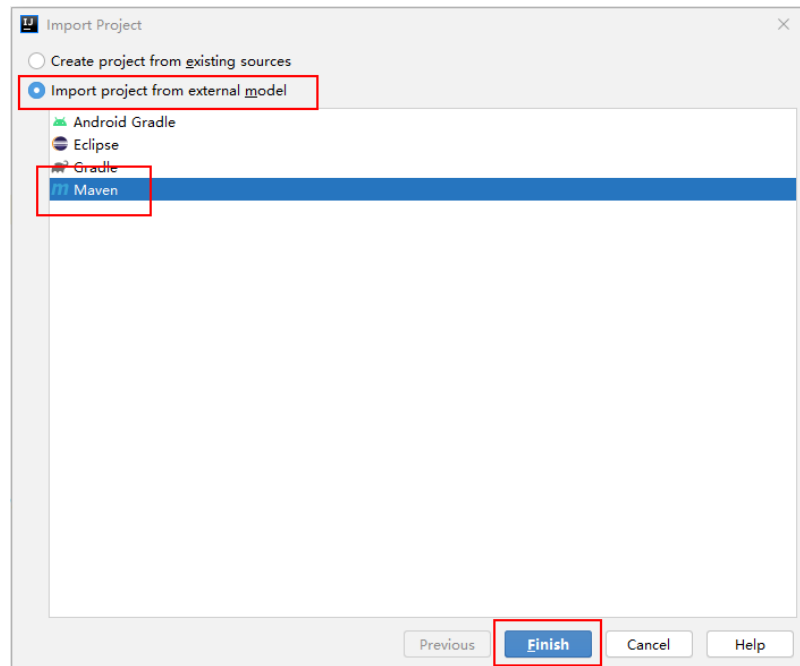
3. Select the sample project to be imported, for example, **storm-examples**.

Figure 33-9 Selecting the sample project to be imported



4. Import the project as a Maven project.

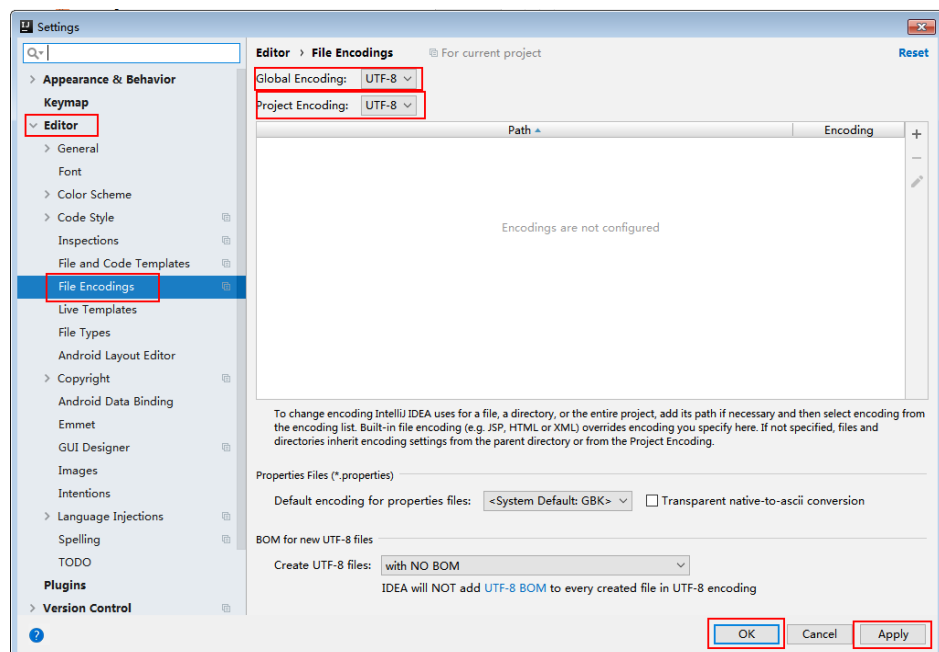
**Figure 33-10** Importing the project as a Maven project.



**Step 5** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.  
The **Settings** window is displayed.
2. In the navigation tree, choose **Editor > File Encodings**. In the **Project Encoding** area and **Global Encoding** area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in **Figure 33-11**.

**Figure 33-11** Setting the IntelliJ IDEA encoding format



----End

## 33.3 Developing an Application

### 33.3.1 Typical Scenario Description

A typical scenario can help you quickly learn and master the Storm topology structure and Spout/Bolt development process.

#### Scenario Description

Service processing of a dynamic word counting system is described as follows. The data source is a logical unit that produces random text continuously.

- The data source continuously sends random text, such as "apple orange apple", to the text splitting logic.
- The word splitting logic splits each text entry sent by the data source by space, such as "apple", "orange", "apple", and then sends each word to the word counting logic.
- The word counting logic increases the number of times that a specific word occurs by one when receiving the word, and prints the real-time results. For example:

```
apple: 1
orange: 1
apple: 2
```

### 33.3.2 Development Idea

#### Function Decomposition

The procedure for a user to develop an application to calculate the number of times that each word appears in random text is as [Table 33-5](#):

**Table 33-5** Functions to be developed

No.	Procedure	Code Sample
1	Create a Spout to generate random text.	For details, see <a href="#">Creating a Spout</a> .
2	Create a Bolt to split the random text into words.	For details, see <a href="#">Creating a Bolt</a> .
3	Create a Bolt to calculate the number of times that each word appears.	For details, see <a href="#">Creating a Bolt</a> .
4	Create a Topology.	For details, see <a href="#">Creating a Topology</a> .

For details about certain code, see [Code Samples](#). For details about complete code, see the Storm-examples sample project.

## 33.3.3 Example Code Description

### 33.3.3.1 Creating a Spout

#### Function Description

A Spout is a message source of Storm and message producer of Topology. Generally, a message source reads data from an external source and sends messages (Tuple) to Topology.

One message source can send multiple message streams, and therefore, `OutputFieldsDeclarer.declarerStream` can be used to define multiple streams, and then `SpoutOutputCollector` emits specific streams.

#### Code Sample

The following code snippet belongs to the `nextTuple` method in the `RandomSentenceSpout` class of the `com.huawei.storm.example.common` package, and these code snippets are used to split strings into words.

```
/**
 * {@inheritDoc}
 */
@Override
public void nextTuple()
{
 Utils.sleep(100);
 String[] sentences =
 new String[] {"the cow jumped over the moon",
 "an apple a day keeps the doctor away",
 "four score and seven years ago",
 "snow white and the seven dwarfs",
 "i am at two with nature"};
 String sentence = sentences[random.nextInt(sentences.length)];
 collector.emit(new Values(sentence));
}
```

### 33.3.3.2 Creating a Bolt

#### Function Description

All message processing logic is encapsulated in Bolts. Bolts provide multiple functions, such as filtering and aggregation.

If other topology operators, except for Bolts, exist, `OutputFieldsDeclarer.declareStream` can be used to define streams, and `OutputCollector.emit` can be used to select streams to be emitted.

#### Code Sample

The following code snippet belongs to the `execute` method in the `SplitSentenceBolt` class of the `com.huawei.storm.example.common` package, and these code snippets are used to split a statement into words and send the words.

```
/**
 * {@inheritDoc}
 */
@Override
public void execute(Tuple input, BasicOutputCollector collector)
{
 String sentence = input.getString(0);
 String[] words = sentence.split(" ");
 for (String word : words)
 {
 word = word.trim();
 if (!word.isEmpty())
 {
 word = word.toLowerCase();
 collector.emit(new Values(word));
 }
 }
}
```

The following code snippets are in the `com.huawei.storm.example.wordcount.WordCountBolt` class, and these code snippets are used to calculate the number of received words.

```
@Override
public void execute(Tuple tuple, BasicOutputCollector collector)
{
 String word = tuple.getString(0);
 Integer count = counts.get(word);
 if (count == null)
 {
 count = 0;
 }
 count++;
 counts.put(word, count);
 System.out.println("word: " + word + ", count: " + count);
}
```

### 33.3.3.3 Creating a Topology

#### Function Description

A topology is a directed acyclic graph (DAG) consisting of Spouts and Bolts.

Applications are submitted in storm jar mode. Therefore, a function for creating a topology must be invoked in the main function, and the class to which the main function belongs must be specified in storm jar parameters.

#### Code Sample

The following code snippet belongs to the `main` method in the `WordCountTopology` class of the `com.huawei.storm.example.wordcount` package, and these code snippets are used to create and submit applications.

```
public static void main(String[] args)
 throws Exception
{
 TopologyBuilder builder = buildTopology();

 /*
 * Tasks can be submitted in the following three modes:
 * 1. Command line submitting. In this mode, a user must copy an application JAR package to a client
 and run related commands on the client.
 * 2. Remote submitting. In this mode, a user must package application JAR packages and execute the
 main method in IntelliJ IDEA.
 */
}
```



```

 * 3. Local submitting. In this mode, a user must run an application for test on a local computer.
 * The command line submitting and remote submitting modes support both security and normal
modes.
 * The local submitting mode supports the normal mode only.
 *
 * A user can select only one mode for submitting a task. By default, the command line submitting
mode is used. To use another mode, delete code comments.
 */

 cmdSubmit(builder, null);

 //Remote submitting mode
 //remoteSubmit(builder);

 //Local submitting mode, used for test
 //localSubmit(builder);
}

private static void cmdSubmit(TopologyBuilder builder, Config conf)
 throws AlreadyAliveException, InvalidTopologyException, NotALeaderException,
AuthorizationException
{
 if (conf == null)
 {
 conf = new Config();
 }
 /**
 * Command line submitting mode
 * The procedures are as follows:
 * Package a JAR package and then submit the task in the client CLI.
 * In remote submitting mode, package application JAR packages and other externally dependent
JAR packages into one JAR package. Other externally dependent JAR packages are depended by user
programs, not provided by the example project.
 * Run the storm -jar command on the Storm client to submit the task.
 *
 * In a security environment, before submitting the task in the client CLI, run the kinit command to
log in in security mode.
 *
 * Run the following command:
 * ./storm jar ../example/example.jar com.huawei.storm.example.WordCountTopology
 */
 conf.setNumWorkers(1);
 StormSubmitter.submitTopology(TOPOLOGY_NAME, conf, builder.createTopology());
}

private static void localSubmit(TopologyBuilder builder)
 throws InterruptedException
{
 Config conf = new Config();
 conf.setDebug(true);
 conf.setMaxTaskParallelism(3);
 LocalCluster cluster = new LocalCluster();
 cluster.submitTopology(TOPOLOGY_NAME, conf, builder.createTopology());
 Thread.sleep(10000);
 cluster.shutdown();
}

private static void remoteSubmit(TopologyBuilder builder)
 throws AlreadyAliveException, InvalidTopologyException, NotALeaderException,
AuthorizationException,
IOException
{
 Config config = createConf();

 //Preparations in security mode
 if (isSecurityModel())
 {
 securityPrepare();
 }
}

```

```
}
String userJarFilePath = "User JAR package address";
System.setProperty(STORM_SUBMIT_JAR_PROPERTY, userJarFilePath);

cmdSubmit(builder, config);
}

private static TopologyBuilder buildTopology()
{
 TopologyBuilder builder = new TopologyBuilder();
 builder.setSpout("spout", new RandomSentenceSpout(), 5);
 builder.setBolt("split", new SplitSentenceBolt(), 8).shuffleGrouping("spout");
 builder.setBolt("count", new WordCountBolt(), 12).fieldsGrouping("split", new Fields("word"));
 return builder;
}
```

 **NOTE**

If ack is enabled for the topology, it is recommended that the number of ackers be less than or equal to the configured number of workers.

## 33.4 Running an Application

### 33.4.1 Packaging IntelliJ IDEA Code

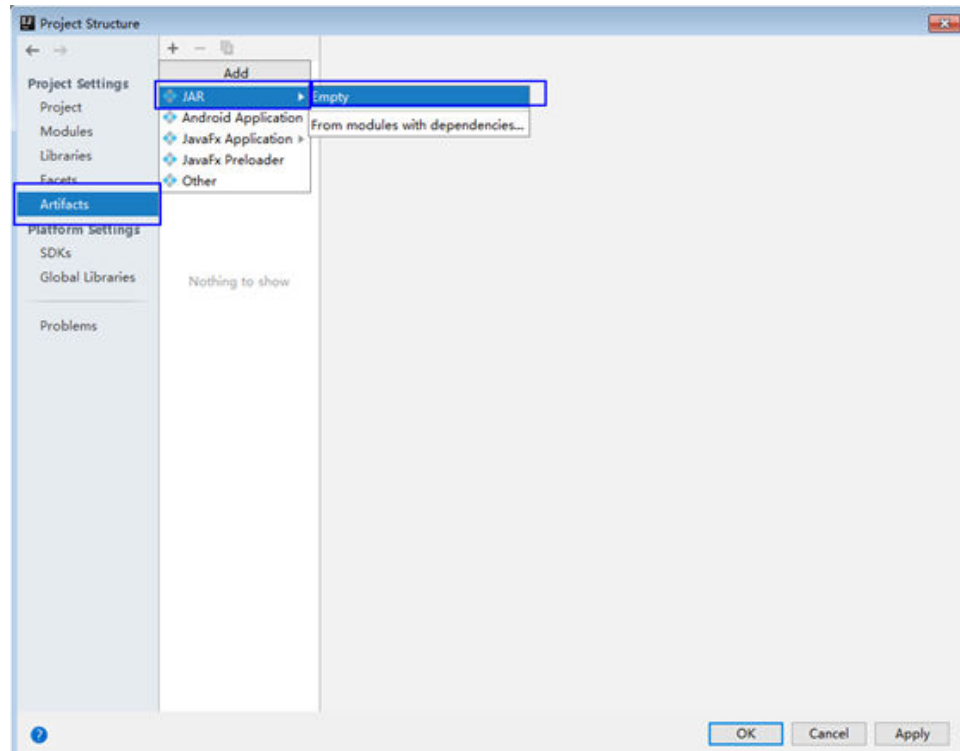
#### Scenario

Export a JAR package using IntelliJ IDEA and specify the name of the JAR package, for example, **storm-examples.jar**.

#### Procedure

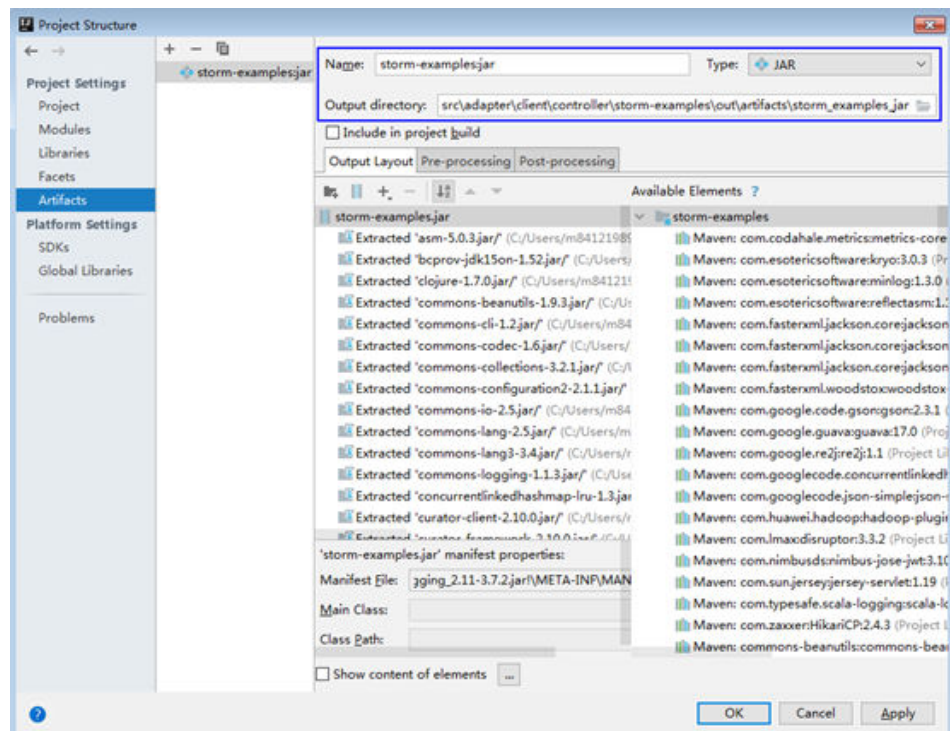
- Step 1** In IntelliJ IDEA, configure the Artifacts information for the project before the JAR package is generated.
1. Enter IntelliJ IDEA and choose **File > Project Structure**.
  2. On the **Project Structure** page, select **Artifacts**, click the plus sign (+), and choose **JAR > Empty**.

Figure 33-12 Adding Artifacts



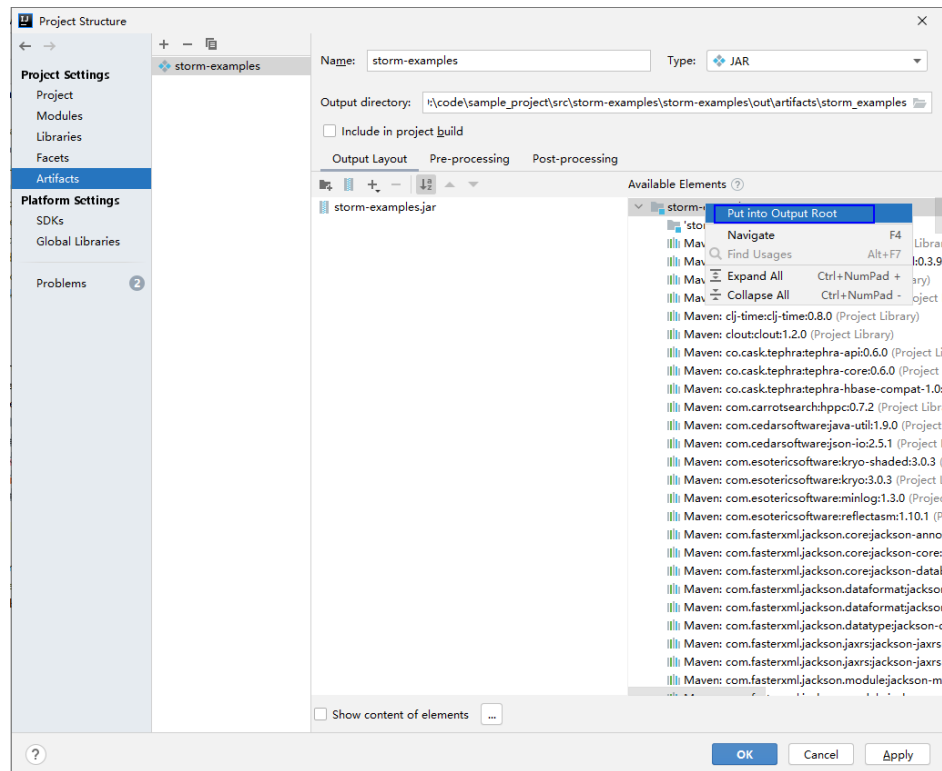
3. Set the name, type, and output path of the JAR package based on site requirements.

Figure 33-13 Configuring basic information



4. Choose 'storm-examples', right-click **Put into Output Root**, and click **Apply**.

Figure 33-14 Put into Output Root

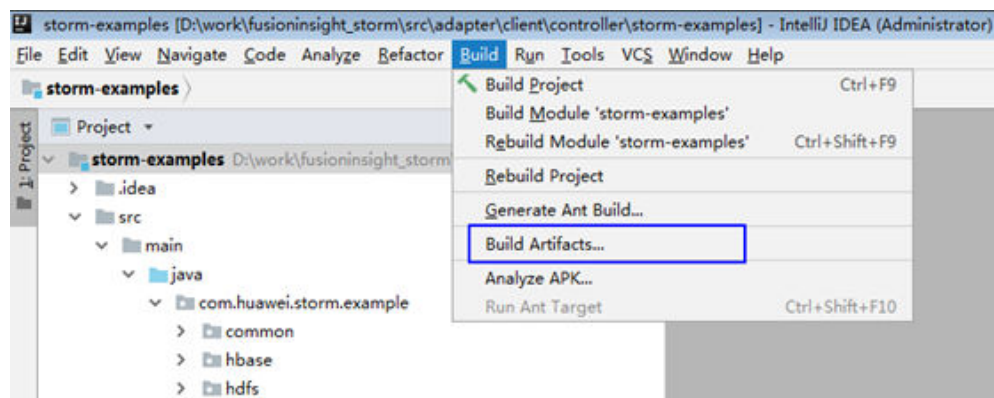


5. Click **OK** to complete the configuration.

**Step 2** Generate the JAR package.

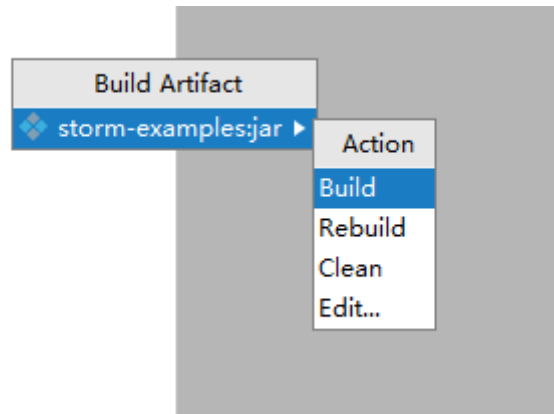
1. Enter IntelliJ IDEA and choose **Build > Build Artifacts...**

Figure 33-15 Build Artifacts



2. In the displayed interface, choose **storm-examples:jar > Build** the JAR package.

Figure 33-16 Build



3. If log similar to the following is displayed in event logs, the JAR package is successfully generated. Obtain the JAR package from the **Output directory** configured shown in [Figure 33-13](#)

```
14:37 Compilation completed successfully in 25 s 991 ms
```

----End

## 33.4.2 Packaging Services

### 33.4.2.1 Overview

Packaging services is to combine the JAR package generated by IntelliJ IDEA code and the dependency JAR package of the project to export the **source.jar** package that can be submitted. The storm-jartool tool is required for service packaging. The tool can run on a Windows or Linux OS.

### 33.4.2.2 Packaging Services on a Linux OS

#### Scenario

Storm supports packaging in a Linux environment. You can upload JAR packages exported from IntelliJ IDEA and other related JAR packages to a Linux environment and perform packaging.

#### Prerequisites

- You have installed the Storm client.
- [Packaging IntelliJ IDEA Code](#) is complete.
- When the host where the Linux OS runs is not a node of the cluster, you are required to set the mapping between the host name and IP address in the **hosts** file of the node where the Linux OS runs. The host name must be correctly mapped to the IP address.

#### Procedure

- Step 1** Copy the JAR package exported from IntelliJ IDEA to a specified directory on the Linux client(for example, **/opt/jarsource**) .

- Step 2** If the service needs to access external components, obtain the configuration files and place them in the directory specified in [Step 1](#) For details about the dependency configuration files, see related development guidelines.
- Step 3** If the service needs to access external components, obtain the JAR packages and place them in the directory specified in [Step 1](#) For details about the dependency JAR packages, see the related development guidelines.
- Step 4** Run the packaging command in the Storm client installation directory **Storm/storm-1.2.1/bin** to package the preceding JAR packages into a complete service JAR package and place the package in the **/opt/jartarget** directory (which can be any empty directory). After the **sh storm-jartool.sh /opt/jarsource/ /opt/jartarget** command is executed, **source.jar** is generated in the **/opt/jartarget** directory.

----End

### 33.4.2.3 Packaging Services on a Windows OS

#### Scenario

Storm supports packaging in a Windows environment.

#### Prerequisites

[Packaging IntelliJ IDEA Code](#) is complete.

#### Procedure

- Step 1** Place the JAR package exported from IntelliJ IDEA to a specified directory (for example, **D:\source**).
- Step 2** Create the lib directory in the sample code directory **src/storm-examples/storm-examples**, copy the JAR package exported from IntelliJ IDEA to the lib directory, and decompress the package.
- Step 3** If the service needs to access external components, obtain the configuration files and place them in the lib directory specified in [Step 1](#) For details about the dependency configuration files, see related development guidelines.
- Step 4** If the service needs to access external components, obtain the JAR packages and place them in the lib directory specified in [Step 1](#) For details about the dependency JAR packages, see the related development guidelines.
- Step 5** Find **storm-jartool.cmd** in the **tools** example project directory.
- Step 6** Double-click the packaging tool, enter the directory where the JAR packages to be packaged locate (**D:\source**) and press **Enter**, and enter the directory where the package to be generated locates (**D:\target**). The **source.jar** file is generated in the **D:\target** directory.

----End

### 33.4.3 Submitting a Topology

### 33.4.3.1 Submitting a Topology When a Client Is Installed on a Linux OS

#### Scenario

You can use storm commands to submit topologies in a Linux environment.

#### Prerequisites

- You have installed the Storm client.
- If the host where the client is installed is not a node in the cluster, set the mapping between the host name and the IP address in the **hosts** file on the node where the client locates. The host name and IP address must be in one-to-one mapping.
- [Packaging Services](#) has been performed and **source.jar** has been generated.

#### Procedure

- Step 1** Submit a topology. (Wordcount is used as an example. For details about other topologies, see related development guide.) Go to the **Storm/storm-1.2.1/bin** directory on the Storm client, and run the **storm jar /opt/jartarget/source.jar com.huawei.storm.example.wordcount.WordCountTopology** command to submit the generated **source.jar** package. (If the package is generated on a Windows OS, upload the **source.jar** package to a specified directory on the Linux server, such as **/opt/jartarget**.)
- Step 2** Run the **storm list** command to view the submitted applications. If the word-count application can be viewed, the task is submitted successfully.

#### NOTE

If a service is set to the local mode and is submitted by using commands, ensure that the submitting environment is a normal one. Currently, services in local mode cannot be submitted by using commands in a security environment.

----End

### 33.4.3.2 Submitting a Topology When No Client Is Installed on a Linux OS

#### Scenario

Storm supports topology running in a Linux environment where no Storm client is installed.

#### Prerequisites

- Ensure that the time difference between the client and the MRS cluster is less than 5 minutes.
- If the host where the client is installed is not a node in the cluster, set the mapping between the host name and the IP address in the **hosts** file on the node where the client locates. The host name and IP address must be in one-to-one mapping.

## Procedure

### Step 1 Prepare for dependency JAR packages and configuration files.

In the Linux environment, create a directory, such as **/opt/test**, and create subdirectories **lib** and **src/main/resources/**. Upload the JAR packages in the **lib** folder of the example project to the **lib** directory in the Linux environment. Upload the configuration files in the **src/main/resources** folder of the example project to the **src/main/resources** directory in the Linux environment.

### Step 2 Modify the **WordCountTopology.java** class in the IntelliJ IDEA project and change the application submission mode to the remoteSubmit mode. Replace the JAR file address.

- Submit the application in remoteSubmit mode.

```
public static void main(String[] args)
 throws Exception
{
 TopologyBuilder builder = buildTopology();

 /*
 * The task submitting modes are as follows:
 * 1. Command submitting mode: In this mode, you need to copy an application JAR package to
 the client machine and run related commands for submitting.
 * 2. Remote submitting mode: In this mode, you need to package related content into an
 application JAR package and run the main method in IntelliJ IDEA for submitting.
 * 3. Local submitting mode: In this mode, you need to run an application locally. This mode is
 usually used for tests.
 * The command submitting mode and remote submitting mode are supported in both the
 security and normal modes.
 * The local submitting mode is supported only in the normal mode.
 *
 * The user can only select one mode for task submitting. The command submitting mode is
 used by default. If another mode is used, delete the code comments.
 */

 submitTopology(builder, SubmitType.REMOTE);
}
```

- Change the value of **userJarFilePath** to the specified directory **/opt/test/lib/example.jar** in the Linux environment.

```
private static void remoteSubmit(TopologyBuilder builder)
 throws AlreadyAliveException, InvalidTopologyException, NotALeaderException,
 AuthorizationException,
 IOException
{
 Config config = createConf();

 String userJarFilePath = "/opt/test/lib/example.jar ";
 System.setProperty(STORM_SUBMIT_JAR_PROPERTY, userJarFilePath);

 //Preparations in security mode
 if (isSecurityModel())
 {
 securityPrepare(config);
 }
 config.setNumWorkers(1);
 StormSubmitter.submitTopologyWithProgressBar(TOPOLOGY_NAME, config,
 builder.createTopology());
}
```

### Step 3 Export the JAR package to the Linux environment.

- Package the IntelliJ IDEA code by referring to [Packaging IntelliJ IDEA Code](#) and name the JAR package **example.jar**.
- Copy the exported JAR package to the **/opt/test/lib** directory in the Linux environment.



**Step 4** Go to the `/opt/test` directory, and run the following command to run the JAR package:

```
java -classpath /opt/test/lib/*:/opt/test/src/main/resources
com.huawei.storm.example.wordcount.WordCountTopology

----End
```

### 33.4.3.3 Submitting a Topology in IntelliJ IDEA Remotely

#### Scenario

Storm supports remote topology submitting in IntelliJ IDEA. In current example code, only `WordCountTopology` can be submitted remotely. If you want to submit other topologies remotely, implement the remote submission function first. For details, see `WordCountTopology`.

#### Prerequisites

- [Packaging IntelliJ IDEA Code](#) is complete.
- Ensure that the time difference between the client and the Storm cluster is less than 5 minutes.
- When the host where the Linux OS runs is not a node of the cluster, you are required to set the mapping between the host name and IP address in the `hosts` file of the node where the Linux OS runs. The host name must be correctly mapped to the IP address.

#### Procedure

**Step 1** Modify the `WordCountTopology.java` class and change the application submission mode to the `remoteSubmit` mode. Replace the user keytab file name, user principal name, and the JAR file address.

- Submit the application in `remoteSubmit` mode.

```
public static void main(String[] args)
 throws Exception
 {
 TopologyBuilder builder = buildTopology();

 /*
 * The task submitting modes are as follows:
 * 1. Command submitting mode: In this mode, you need to copy an application JAR package to
 * the client machine and run related commands for submitting.
 * 2. Remote submitting mode: In this mode, you need to package related content into an
 * application JAR package and run the main method in IntelliJ IDEA for submitting.
 * 3. Local submitting mode: In this mode, you need to run an application locally. This mode is
 * usually used for tests.
 * The command submitting mode and remote submitting mode are supported in both the
 * security and normal modes.
 * The local submitting mode is supported only in the normal mode.
 *
 * The user can only select one mode for task submitting. The command submitting mode is
 * used by default. If another mode is used, delete the code comments.
 */

 submitTopology(builder, SubmitType.REMOTE);
 }
```

- Change the value of `userJarFilePath` to the actual topology JAR package address based on the actual situation.

```
private static void remoteSubmit(TopologyBuilder builder)
 throws AlreadyAliveException, InvalidTopologyException, NotALeaderException,
 AuthorizationException,
 IOException
 {
 Config config = createConf();

 String userJarFilePath = "D:\\example.jar";
 System.setProperty(STORM_SUBMIT_JAR_PROPERTY, userJarFilePath);

 //Preparations in security mode
 if (isSecurityModel())
 {
 securityPrepare(config);
 }
 config.setNumWorkers(1);
 StormSubmitter.submitTopologyWithProgressBar(TOPOLOGY_NAME, config,
 builder.createTopology());
 }
```

**Step 2** Execute the **Main** method of the **WordCountTopology.java** class to submit the application.

----End

## 33.4.4 Viewing Results

### Scenario

After a Storm application is run, you can log in to the Storm WebUI to check the running result.

### Procedure

**Step 1** Log in to FusionInsight Manager.

Enter the address in the address box of your browser. The address format is `https://FusionInsight Manager WebService floating IP address.28443/web`.

For example, enter `https://10.0.0.1:28443/web`.

**Step 2** Choose **Cluster > Name of the desired cluster > Services > Storm** to go to the Storm WebUI.

**Step 3** On the Storm UI, click the word-count application to view the application running status, as shown in [Figure 33-17](#).

**Figure 33-17** Storm application execution page

### Topology summary

Name	Id	Owner	Status	Uptime	Num workers	Num executors
word-count	word-count-6-1536579754	stormkafka1	ACTIVE	1m 24s	1	26

### Topology actions

Activate	Deactivate	Rebalance	Kill	Debug	Stop Debug	Change Log Level
----------	------------	-----------	------	-------	------------	------------------

### Topology stats

Window	Emitted	Transferred	Complete latency (ms)
10m 0s	22500	22500	0
3h 0m 0s	22500	22500	0
1d 0h 0m 0s	22500	22500	0
All time	22500	22500	0

In **Topology stats**, the total volume of data transferred between operators in different time periods is displayed.

In **Spouts**, the total number of messages sent by the spout operator from the moment the operator is started till now is displayed. In **Bolts**, the total number of messages sent by the Count operator and the split operator is displayed, as shown in [Figure 33-18](#).

**Figure 33-18** Total volume of data sent by the Storm application operators

### Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Last error
spout	5	5	20940	20940	0.000	0	0	

### Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)
count	12	12	0	0	0.006	0.105	133920	0.086
split	8	8	133880	133880	0.005	0.670	20940	0.648

----End

## 33.5 More Information

## 33.5.1 Storm-Kafka Development Guideline

### Scenario

This topic describes how to use the Storm-Kafka toolkit to implement the interaction between Storm and Kafka. KafkaSpout and KafkaBolt are included. KafkaSpout enables Storm to read data from Kafka. KafkaBolt enables Storm to write data into Kafka.

The code example in this section is based on the new Kafka API and corresponds to `com.huawei.storm.example.kafka.NewKafkaTopology.java` in the IntelliJ IDEA project.

This topic applies only to the access between the Storm component and the Kafka component of MRS. Determine the versions of the jar packages described in this chapter based on the actual situation.

### Procedure for Developing an Application

**Step 1** Verify that the Storm and Kafka components of Huawei MRS have been installed and are running properly.

**Step 2** Obtain the sample project folder `storm-examples` in the `src\storm-examples` directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#). Import `storm-examples` to the IntelliJ IDEA development environment. For details, see [Environment Preparation](#).

**Step 3** Install the Storm client in Linux OS.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

**Step 4** Download and install the Kafka client.

----End

### Code Sample

```
1. Create a topology
public static void main(String[] args) throws Exception {

 // Set the topology
 Config conf = new Config();

 // Configure the security plug-in.
 setSecurityPlugin(conf);

 if (args.length >= 2) {
 // If the default keytab file name has been changed, configure the new keytab file name.
 conf.put(Config.TOPOLOGY_KEYTAB_FILE, args[1]);
 }

 // Define KafkaSpout
 KafkaSpout kafkaSpout = new KafkaSpout<String, String>(
 getKafkaSpoutConfig(getKafkaSpoutStreams()));

 // CountBolt
 CountBolt countBolt = new CountBolt();
```

```
//SplitBolt
SplitSentenceBolt splitBolt = new SplitSentenceBolt();

// KafkaBolt configuration information
KafkaBolt<String, String> kafkaBolt = new KafkaBolt<String, String>();
kafkaBolt.withTopicSelector(new DefaultTopicSelector(OUTPUT_TOPIC))
.withTupleToKafkaMapper(
new FieldNameBasedTupleToKafkaMapper("word", "count"));
kafkaBolt.withProducerProperties(getKafkaProducerProps());

// Define the topology.
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("kafka-spout", kafkaSpout, 10);
builder.setBolt("split-bolt", splitBolt, 10).shuffleGrouping("kafka-spout", STREAMS[0]);
builder.setBolt("count-bolt", countBolt, 10).fieldsGrouping(
"split-bolt", new Fields("word"));
builder.setBolt("kafka-bolt", kafkaBolt, 10).shuffleGrouping("count-bolt");

// Submit the topology.
StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
}
```

## 2. **getKafkaConsumerProps()** in the NewKafkaTopology class

```
private static Map<String, Object> getKafkaConsumerProps() throws Exception {
 Map<String, Object> props = new HashMap<String, Object>();
 props.put(GROUP_ID, DEFAULT_GROUP_ID);
 props.put(SASL_KERBEROS_SERVICE_NAME, DEFAULT_SERVICE_NAME);
 props.put(SEcurity_PROTOCOL, DEFAULT_SECURITY_PROTOCOL);
 props.put(KEY_DESERIALIZER, DEFAULT_DESERIALIZER);
 props.put(VALUE_DESERIALIZER, DEFAULT_DESERIALIZER);
 //props.put(KERBEROS_DOMAIN_NAME, "hadoop." +
 KerberosUtil.getDefaultRealm().toLowerCase());
 return props;
}
```

## 3. **getKafkaProducerProps()** in the NewKafkaTopology class

```
private static Properties getKafkaProducerProps() throws Exception {
 Properties props = new Properties();
 props.put(BOOTSTRAP_SERVERS, KAFKA_BROKER_LIST);
 props.put(SECURITY_PROTOCOL, DEFAULT_SECURITY_PROTOCOL);
 props.put(KEY_SERIALIZER, DEFAULT_SERIALIZER);
 props.put(VALUE_SERIALIZER, DEFAULT_SERIALIZER);
 props.put(SASL_KERBEROS_SERVICE_NAME, DEFAULT_SERVICE_NAME);
 //props.put(KERBEROS_DOMAIN_NAME, "hadoop." +
 KerberosUtil.getDefaultRealm().toLowerCase());
 return props;
}
```

### NOTE

If the cluster domain name is modified, set the Kerberos domain name in the Kafka consumer/producer attribute to the actual domain name of the cluster, for example, `props.put(KERBEROS_DOMAIN_NAME, "hadoop.hadoop1.com")`.

## Running the Application and Viewing Results

**Step 1** Export the local JAR package. For details, see [Packaging IntelliJ IDEA Code](#).

**Step 2** Obtain the following JAR packages.

Go to the **Kafka/kafka/libs** directory on the installed Kafka client, and obtain the following JAR packages:

- kafka\_<version>.jar
- scala-library-<version>.jar

- scala-logging\_2.11-3.7.2.jar
- metrics-core-<version>.jar
- kafka-clients-<version>.jar
- zookeeper-<version>.jar

Obtain the following JAR file from the **streaming-cql-<HD-Version>/lib** directory on the Storm client:

- storm-kafka-client-<version>.jar
- storm-kafka-<version>.jar
- slf4j-api-<version>.jar
- guava-<version>.jar
- json-simple-<version>.jar
- curator-client-<version>.jar
- curator-framework-<version>.jar
- curator-recipes-<version>.jar

**Step 3** Combine the JAR packages obtained in the preceding steps and export a complete service JAR package. For details, see [Packaging Services](#).

**Step 4** Go to the directory where the Kafka client locates in the Linux system, use the Kafka client to create the topic used by the topology in the **Kafka/kafka/bin** directory. Run the following command:

```
./kafka-topics.sh --create --topic input --partitions 2 --replication-factor 2 --zookeeper {ip:port}/kafka
```

```
./kafka-topics.sh --create --topic output --partitions 2 --replication-factor 2 --zookeeper {ip:port}/kafka
```

 NOTE

The variable appended to **--zookeeper** specifies the ZooKeeper address. You must set the ZooKeeper address to the ZooKeeper address configured during cluster installation.

**Step 5** Submit the topology on a Linux OS. The submission command example is as follows (the topology name is **kafka-test**):

```
storm jar /opt/jartarget/source.jar
com.huawei.storm.example.kafka.SimpleKafkaTopology kafka-test
```

**Step 6** After the topology is successfully submitted, send data to Kafka and check whether related information is generated.

Go to the directory where the Kafka client locates in the Linux system, start the consumer in the **Kafka/kafka/bin** directory, and check whether data is generated. Run the following command:

```
./kafka-console-consumer.sh --bootstrap-server {ip:port} --topic output --consumer.config ../config/consumer.properties
```

Go to the directory whether the Kafka client locates in the Linux system, start the producer in the **Kafka/kafka/bin** directory, and write data into Kafka. Run the following command:

```
./kafka-console-producer.sh --broker-list {ip:port} --topic input --
producer.config ../config/producer.properties
```

Write test data into input, and check whether related data is generated in output. If yes, the Storm-Kafka topology is executed successfully.

----End

## 33.5.2 Storm-JDBC Development Guideline

### Scenario

This topic describes how to use the open-source Storm-JDBC toolkit to implement the interaction between Storm and JDBC. `JdbcInsertBolt` and `JdbcLookupBolt` are included. `JdbcLookupBolt` is used to query data from the database. `JdbcInsertBolt` is used to store data to the database. In addition, `JdbcLookupBolt` and `JdbcInsertBolt` can be used to process data using the data processing logic.

This topic applies only to the access between the Storm component and the JDBC component. Determine the versions of the jar packages described in this chapter based on the actual situation.

### Procedure for Developing an Application

**Step 1** Verify that the Storm component has been installed and is running properly.

**Step 2** Obtain the sample project folder `storm-examples` in the `src\storm-examples` directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#). Import `storm-examples` to the IntelliJ IDEA development environment. For details, see [Environment Preparation](#).

**Step 3** Install the Storm client in Linux OS.

For details about how to use the client on a Master or Core node in the cluster, see [Using an MRS Client on Nodes Inside a Cluster](#). For details about how to install the client outside the MRS cluster, see [Using an MRS Client on Nodes Outside a Cluster](#).

----End

### Configuring the Database - Configuring the Derby Database

**Step 1** Download a database. Select the most appropriate database based on the actual scenario.

In this topic, the Derby database is used as an example. The Derby database is a Java-based small-sized open-source database that is easy to use and suitable to most applications.

**Step 2** Obtain the Derby database. Download the Derby database package of the latest version from the official website, use WinSCP to upload the database package to the Linux client, for example, `/opt`, and decompress the package.

**Step 3** In the Derby installation directory, go to the `bin` directory, and run the following commands:

```
export DERBY_INSTALL=/opt/db-derby-10.12.1.1-bin
```

```
export CLASSPATH=$DERBY_INSTALL/lib/derbytools.jar:$DERBY_INSTALL/lib
\derbynet.jar:.
```

```
export DERBY_HOME=/opt/db-derby-10.12.1.1-bin
```

```
. setNetworkServerCP
```

```
./startNetworkServer -h Host name
```

**Step 4** Run the `./ij` command, and enter `connect 'jdbc:derby://Host name:1527/example;create=true'`; to create the connection.

**Step 5** After the database is connected, run SQL statements to create table ORIGINAL and table GOAL and insert a group of data into table ORIGINAL. The statement examples are as follows (the table names can be customized):

```
CREATE TABLE GOAL(WORD VARCHAR(12),COUNT INT);
```

```
CREATE TABLE ORIGINAL(WORD VARCHAR(12),COUNT INT);
```

```
INSERT INTO ORIGINAL VALUES('orange',1),('pineapple',1),('banana',1),
('watermelon',1);
```

```
----End
```

## Code Sample

SimpleJDBCTopology code sample: (Change the IP addresses and ports to the actual ones.)

```
public class SimpleJDBCTopology
{
 private static final String WORD_SPOUT = "WORD_SPOUT";
 private static final String COUNT_BOLT = "COUNT_BOLT";
 private static final String JDBC_INSERT_BOLT = "JDBC_INSERT_BOLT";
 private static final String JDBC_LOOKUP_BOLT = "JDBC_LOOKUP_BOLT";
 @SuppressWarnings ("unchecked")
 public static void main(String[] args) throws Exception{
 //connectionProvider configuration
 Map hikariConfigMap = Maps.newHashMap();
 hikariConfigMap.put("dataSourceClassName", "org.apache.derby.jdbc.ClientDataSource");
 hikariConfigMap.put("dataSource.serverName", "192.168.0.1");//Set this parameter to the actual IP
 address.
 hikariConfigMap.put("dataSource.portNumber", "1527");//Set this parameter to the actual port number.

 hikariConfigMap.put("dataSource.databaseName", "example");
 hikariConfigMap.put("dataSource.user", "app");
 hikariConfigMap.put("dataSource.password", "mine");
 hikariConfigMap.put("connectionTestQuery", "select COUNT from GOAL"); //The table name must be
 consistent with that during table creation.
 Config conf = new Config();

 ConnectionProvider connectionProvider = new HikariCPCConnectionProvider(hikariConfigMap);
 //JdbcLookupBolt instantiation
 Fields outputFields = new Fields("WORD", "COUNT");
 List<Column> queryParamColumns = Lists.newArrayList(new Column("WORD", Types.VARCHAR));
 SimpleJdbcLookupMapper jdbcLookupMapper = new SimpleJdbcLookupMapper(outputFields,
 queryParamColumns);
 String selectSql = "select COUNT from ORIGINAL where WORD = ?";
 JdbcLookupBolt wordLookupBolt = new JdbcLookupBolt(connectionProvider, selectSql,
 jdbcLookupMapper);
 //JdbcInsertBolt instantiation
 String tableName = "GOAL";
 JdbcMapper simpleJdbcMapper = new SimpleJdbcMapper(tableName, connectionProvider);
 JdbcInsertBolt userPersistenceBolt = new JdbcInsertBolt(connectionProvider,
```



```
simpleJdbcMapper).withTableName("GOAL").withQueryTimeoutSecs(30);
 WordSpout wordSpout = new WordSpout();TopologyBuilder builder = new TopologyBuilder();
 builder.setSpout(WORD_SPOUT, wordSpout);
 builder.setBolt(JDBC_LOOKUP_BOLT, wordLookupBolt, 1).fieldsGrouping(WORD_SPOUT,new
Fields("WORD"));
 builder.setBolt(JDBC_INSERT_BOLT, userPersistenceBolt,1).fieldsGrouping(JDBC_LOOKUP_BOLT,new
Fields("WORD"));StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
}
}
```

## Running and Submitting the Topology

**Step 1** Export the local JAR package. For details, see [Packaging IntelliJ IDEA Code](#).

**Step 2** Obtain the following JAR packages.

- Go to the **lib** directory in the installed database, and obtain the following JAR packages:
  - derbyclient.jar
  - derby.jar
- Obtain the following JAR file from the **streaming-cql-*<HD-Version>/lib*** directory on the Storm client:
  - storm-jdbc-*<version>*.jar
  - guava-*<version>*.jar
  - commons-lang3-*<version>*.jar
  - commons-lang-*<version>*.jar
  - HikariCP-*<version>*.jar

**Step 3** Combine the JAR packages obtained in the preceding steps and export a complete service JAR package. For details, see [Packaging Services](#).

**Step 4** Run a command to submit the topology. The submission command example is as follows (the topology name is **jdbc-test**):

```
storm jar /opt/jartarget/source.jar
com.huawei.storm.example.jdbc.SimpleJDBCTopology jdbc-test
```

----End

## Viewing Results

After the topology is submitted, go to the database and check whether data is inserted into the related tables.

Run the **select \* from goal;** statement to query data in table **goal**. If data is inserted into table GOAL, the topology is executed successfully.

## 33.5.3 Storm-HDFS Development Guideline

### Scenario

This topic applies only to the interaction between Storm and HDFS. Determine the versions of the jar packages described in this chapter based on the actual situation.

## Procedure for Developing an Application

- Step 1** Verify that the Storm and HDFS components have been installed and are running properly.
- Step 2** Import storm-examples to the IntelliJ IDEA development environment. For details, see [Environment Preparation](#).
- Step 3** Download and install the HDFS client.
- Step 4** Obtain the related configuration files using the following method.

Go to the **/opt/clientHDFS/HDFS/hadoop/etc/hadoop** directory on the installed HDFS client, and obtain configuration files **core-site.xml** and **hdfs-site.xml**.

- Step 5** Obtain the related JAR packages.
  - Go to the **HDFS/hadoop/share/hadoop/common/lib** directory on the installed HDFS client, and obtain the following JAR packages:
    - commons-cli-<version>.jar
    - commons-io-<version>.jar
    - commons-lang-<version>.jar
    - commons-lang3-<version>.jar
    - commons-collections-<version>.jar
    - commons-configuration2-<version>.jar
    - commons-logging-<version>.jar
    - guava-<version>.jar
    - hadoop-\*.jar
    - protobuf-java-<version>.jar
    - jackson-databind-<version>.jar
    - jackson-core-<version>.jar
    - jackson-annotations-<version>.jar
    - re2j-<version>.jar
    - jaeger-core-<version>.jar
    - opentracing-api-<version>.jar
    - opentracing-noop-<version>.jar
    - opentracing-tracerresolver-<version>.jar
    - opentracing-util-<version>.jar
  - Go to the **HDFS/hadoop/share/hadoop/common** directory on the installed HDFS client, and obtain the **hadoop-\*.jar** package.
  - Go to the **HDFS/hadoop/share/hadoop/client** directory on the installed HDFS client, and obtain the **hadoop-\*.jar** package.
  - Go to the **HDFS/hadoop/share/hadoop/hdfs** directory on the installed HDFS client, obtain the **hadoop-hdfs-\*.jar** package.
  - Obtain the following JAR packages from the sample project **/src/storm-examples/storm-examples/lib**:
    - storm-hdfs-<version>.jar

```
- storm-autocreds-<version>.jar
----End
```

## IntelliJ IDEA Code Sample

Create a topology.

```
public static void main(String[] args) throws Exception
{
 TopologyBuilder builder = new TopologyBuilder();

 // Separator. Use ; to replace the default | to separate fields in tuple.
 // Mandatory HdfsBolt parameter
 RecordFormat format = new DelimitedRecordFormat()
 .withFieldDelimiter("|");

 // Synchronize policy. Synchronize the file system for every 1000 tuples.
 // Mandatory HdfsBolt parameter

 SyncPolicy syncPolicy = new CountSyncPolicy(1000);

 // File size cyclic policy. If the size of a file reaches 5 MB, the file is written from the beginning.
 // Mandatory HdfsBolt parameter
 FileRotationPolicy rotationPolicy = new FileSizeRotationPolicy(5.0f, Units.MB);

 // Objective file written to hdfs
 // Mandatory HdfsBolt parameter
 FileNameFormat fileNameFormat = new DefaultFileNameFormat()
 .withPath("/user/foo/");

 //Create HdfsBolt.
 HdfsBolt bolt = new HdfsBolt()
 .withFsUrl(DEFAULT_FS_URL)
 .withFileNameFormat(fileNameFormat)
 .withRecordFormat(format)
 .withRotationPolicy(rotationPolicy)
 .withSyncPolicy(syncPolicy);

 //Spout generates a random statement.
 builder.setSpout("spout", new RandomSentenceSpout(), 1);
 builder.setBolt("split", new SplitSentence(), 1).shuffleGrouping("spout");
 builder.setBolt("count", bolt, 1).fieldsGrouping("split", new Fields("word"));

 //Add the plugin required for kerberos authentication to the list. The security mode is mandatory.

 setSecurityConf(conf, AuthenticationType.KEYTAB);

 Config conf = new Config();
 //Write the plugin list configured on the client to a specific config item. The security mode is mandatory.
 conf.put(Config.TOPOLOGY_AUTO_CREDENTIALS, auto_tgts);

 if(args.length >= 2)
 {
 //The default keytab file name is changed by the user. Specify the new keytab file name as a
parameter.
 conf.put(Config.STORM_CLIENT_KEYTAB_FILE, args[1]);
 }

 //Run a command to submit the topology.
 StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
}
```

### NOTE

The target file path of Storm cannot be in a SM4 encrypted HDFS partition.

## Running the Application and Viewing Results

- Step 1** Export the local JAR package. For details, see [Packaging IntelliJ IDEA Code](#).
- Step 2** Combine the configuration files and JAR packages obtained respectively in [Step 4](#) and [Step 5](#), and export a complete service JAR package. For details, see [Packaging Services](#).
- Step 3** Run a command to submit the topology.
- ```
storm jar /opt/jartarget/source.jar  
  
com.huawei.storm.example.hdfs.SimpleHDFSTopology hdfs-test
```
- Step 4** After the topology is submitted successfully, log in to the HDFS cluster to view the topology.
- End

33.5.4 Storm-HBase Development Guideline

Scenario

This topic applies only to the interaction between Storm and HBase. Determine the versions of the jar packages described in this chapter based on the actual situation.

Procedure for Developing an Application

- Step 1** Verify that the Storm and HBase components have been installed and are running properly.
- Step 2** Import storm-examples to the IntelliJ IDEA development environment. For details, see [Environment Preparation](#).
- Step 3** Download and install the HBase client.
- Step 4** Obtain the related configuration files using the following method.
- Go to the `/opt/clientHbase/HBase/hbase/conf` directory on the installed HBase client, and obtain configuration files `core-site.xml`, `hdfs-site.xml`, and `hbase-site.xml`.
- Step 5** Obtain the related JAR packages.
- Go to the `HBase/hbase/lib` directory on the installed HBase client, and obtain the following JAR packages:
 - hbase-*.jar
 - hadoop-*.jar
 - jackson-core-asl-<version>.jar
 - jackson-mapper-asl-<version>.jar
 - commons-cli-<version>.jar
 - commons-io-<version>.jar
 - commons-lang-<version>.jar

- commons-lang3-<version>.jar
- commons-collections-<version>.jar
- commons-configuration2-<version>.jar
- guava-<version>.jar
- protobuf-java-<version>.jar
- netty-all-<version>.jar
- zookeeper-<version>.jar
- zookeeper-<version>.jar
- zookeeper-jute-<version>.jar
- metrics-core-<version>.jar
- commons-validator-<version>.jar
- Go to the **HBase/hbase/lib/client-facing-thirdparty** directory in the HBase client installation directory, and obtain the **commons-logging-<version>.jar** package.
- Go to the **HBase/hbase/lib/jdbc** directory in the HBase client installation directory, and obtain the **htrace-core-<version>-incubating.jar** package and the **htrace-core4-<version>-incubating.jar** package.
- Obtain the following JAR packages from the sample project `/src/storm-examples/storm-examples/lib`:
 - storm-hdfs-<version>.jar
 - storm-autocreds-<version>.jar

----End

IntelliJ IDEA Code Sample

Create a topology.

```
public static void main(String[] args) throws Exception
{
    Config conf = new Config();
    //Add the plugin required for kerberos authentication to the list. The security mode is mandatory.
    setSecurityConf(conf,AuthenticationType.KEYTAB);

    if(args.length >= 2)
    {
        //The default keytab file name is changed by the user. Specify a new keytab file name as a parameter.
        conf.put(Config.STORM_CLIENT_KEYTAB_FILE, args[1]);
    }
    //hbase client configuration. Only hbase.rootdir configuration item is provided, which is optional.
    Map<String, Object> hbConf = new HashMap<String, Object>();
    if(args.length >= 3)
    {
        hbConf.put("hbase.rootdir", args[2]);
    }
    //Mandatory parameter. If it is not set, it is left blank.
    conf.put("hbase.conf", hbConf);

    //Spout generates a random word.
    WordSpout spout = new WordSpout();
    WordCounter bolt = new WordCounter();

    //HbaseMapper, which is used for parsing tuple content.
    SimpleHBaseMapper mapper = new SimpleHBaseMapper()
        .withRowKeyField("word")
        .withColumnFields(new Fields("word"))
}
```

```
.withCounterFields(new Fields("count"))
.withColumnFamily("cf");

//HBaseBolt, the first parameter is a table name.
//withConfigKey("hbase.conf")Transfer the hbase client configuration to HBaseBolt.
HBaseBolt hbase = new HBaseBolt("WordCount", mapper).withConfigKey("hbase.conf");

// wordSpout ==> countBolt ==> HBaseBolt
TopologyBuilder builder = new TopologyBuilder();

builder.setSpout(WORD_SPOUT, spout, 1);
builder.setBolt(COUNT_BOLT, bolt, 1).shuffleGrouping(WORD_SPOUT);
builder.setBolt(HBASE_BOLT, hbase, 1).fieldsGrouping(COUNT_BOLT, new Fields("word"));
//Run a command to submit the topology.
StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
}
```

Running the Application and Viewing Results

- Step 1** Export the local JAR package. For details, see [Packaging IntelliJ IDEA Code](#).
- Step 2** Combine the configuration files and JAR packages obtained respectively in [Step 4](#) and [Step 5](#), and export a complete service JAR package. For details, see [Packaging Services](#).
- Step 3** Run a command to submit the topology.

```
storm jar /opt/jartarget/source.jar
com.huawei.storm.example.hbase.SimpleHBaseTopology hbase-test
```

NOTE

HBaseBolt in the preceding example does not provide the function for creating tables. Therefore, you must verify that necessary tables exist in HBase. If the tables do not exist, run the **create 'WordCount', 'cf'** statement to manually create HBase shell tables.

- Step 4** After the topology is submitted successfully, log in to the HBase cluster to view the topology.

----End

33.5.5 Flux Development Guideline

Scenario

This topic applies only to the scenario of submitting and deploying a topology using the Flux framework in the Storm component. Determine the versions of the jar packages described in this chapter based on the actual situation.

The Flux framework is a framework provided by Storm. This framework is used to improve the topology deployment usability. Using the Flux framework, users can use a YAML file to define and deploy a topology and run the **storm jar** command to submit the topology. This mode facilitates topology deployment and submitting, and reduces the service development cycle.

Basic Syntax Description

Using Flux to define a topology can be classified into two scenarios: defining a new topology and defining an existing topology.

1. Using Flux to define a new topology

Using Flux to define a topology indicates using a YAML file to describe a topology. A complete topology definition must contain the following parts:

- Topology name
- List of components used for defining the topology
- Topology configuration
- Topology definition, including the spout list, bolt list, and stream list

Code sample for defining the topology name:

```
name: "yaml-topology"
```

Code sample for defining the component list:

```
#Simple component definition
components:
- id: "stringScheme"
  className: "org.apache.storm.kafka.StringScheme"

#Use a constructor to define a component.
- id: "defaultTopicSelector"
  className: "org.apache.storm.kafka.bolt.selector.DefaultTopicSelector"
  constructorArgs:
  - "output"

#Reference parameters as input arguments in a constructor, and use the `ref` tag to describe the
reference.
#When using a reference, ensure that the referenced object has been defined.
- id: "stringMultiScheme"
  className: "org.apache.storm.spout.SchemeAsMultiScheme"
  constructorArgs:
  - ref: "stringScheme"

#Reference the configuration items in the specified properties file as input arguments in a constructor,
and use the `${}` tag to describe the reference.
#If the properties file is referenced, use the --filter my-prop.properties mode to specify the path of the
properties file when you run the storm jar command to submit the topology.
- id: "zkHosts"
  className: "org.apache.storm.kafka.ZkHosts"
  constructorArgs:
  - "${kafka.zookeeper.root.list}"

#Reference environment variables as input arguments in a constructor, and use the `${ENV-[NAME]}`
tag to describe the reference.
#NAME must be a defined environment variable.
- id: "zkHosts"
  className: "org.apache.storm.kafka.ZkHosts"
  constructorArgs:
  - "${ENV-ZK_HOSTS}"

#Use the `properties` keyword to initialize the internal private variables.
- id: spoutConfig
  className: "org.apache.storm.kafka.SpoutConfig"
  constructorArgs:
  - ref: "zkHosts"
  - "input"
  - "/kafka/input"
  - "myId"
  properties:
  - name: "scheme"
  ref: "stringMultiScheme"
```

Code sample for defining the topology configuration:

```
config:
#Simple configuration item
topology.workers: 1
```

```
#If the configuration item value is a list, use `[]` to indicate it.  
topology.auto-credentials: ["class1","class2"]
```

```
#The configuration item value is in the map structure.  
kafka.broker.properties:  
metadata.broker.list: "${metadata.broker.list}"  
producer.type: "async"  
request.required.acks: "0"  
serializer.class: "kafka.serializer.StringEncoder"
```

Code sample for defining the spout/bolt list:

```
#Define the spout list.  
spouts:  
- id: "spout1"  
  className: "org.apache.storm.kafka.KafkaSpout"  
  constructorArgs:  
  - ref: "spoutConfig"  
  parallelism: 1  
  
#Define the bolt list.  
bolts:  
- id: "bolt1"  
  className: "com.huawei.storm.example.hbase.WordCounter"  
  parallelism: 1  
  
#Use a method to initialize an object, and the keyword is `configMethods`.  
- id: "bolt2"  
  className: "org.apache.storm.hbase.bolt.HBaseBolt"  
  constructorArgs:  
  - "WordCount"  
  - ref: "mapper"  
  configMethods:  
  - name: "withConfigKey"  
  args: ["hbase.conf"]  
  parallelism: 1
```

Code sample for defining the stream list:

```
#To define the stream mode, you must specify the grouping mode. The keyword is `grouping`, and  
keywords for grouping methods provided currently are as follows:  
#`ALL`, `CUSTOM`, `DIRECT`, `SHUFFLE`, `LOCAL_OR_SHUFFLE`, `FIELDS`, `GLOBAL`, and `NONE`.  
#`CUSTOM` is used for a customized group.
```

```
#For the definition of a simple stream, the grouping mode is SHUFFLE.
```

```
streams:  
- name: "spout1 --> bolt1"  
  from: "spout1"  
  to: "bolt1"  
  grouping:  
  type: SHUFFLE
```

```
#If the grouping mode is FIELDS, parameters must be entered.
```

```
- name: "bolt1 --> bolt2"  
  from: "bolt1"  
  to: "bolt2"  
  grouping:  
  type: FIELDS  
  args: ["word"]
```

```
#If the grouping mode is CUSTOM, you must specify a customized grouping class.
```

```
- name: "bolt-1 --> bolt2"  
  from: "bolt-1"  
  to: "bolt-2"  
  grouping:  
  type: CUSTOM  
  customClass:  
  className: "org.apache.storm.testing.NGrouping"  
  constructorArgs:  
  - 1
```


2. Using Flux to define an existing topology

If a topology already exists (for example, a topology has already been defined by Java code), you can still use the Flux framework to submit and deploy the topology. In this situation, you must use the `getTopology()` method in the current topology definition (for example, `MyTopology.java`). The definition in Java is as follows:

```
public StormTopology getTopology(Config config)
or
public StormTopology getTopology(Map<String, Object> config)
```

In this situation, you can use the following YAML file to define the topology:

```
name: "existing-topology"#You can specify the topology name to any value.
topologySource:
className: "custom-class"#Specify the client class.
```

You can specify another method name to obtain `StormTopology` (non-`getTopology()` method). The YAML file example is as follows:

```
name: "existing-topology"
topologySource:
className: "custom-class "
methodName: "getTopologyWithDifferentMethodName"
```

NOTE

The specified method must accept an input parameter of the `Map<String, Object>` type or the `Config` type and return an object of the `org.apache.storm.generated.StormTopology` type. This method is the same as the `getTopology()` method.

Procedure for Developing an Application

- Step 1** Verify that the Storm component has been installed and is running properly. If the services need to connect to other components, install the required components and ensure that the components are running properly.
- Step 2** Import storm-examples to the IntelliJ IDEA development environment. For details, see [Configuring and Importing Sample Projects](#).
- Step 3** Develop client services. For details, see the related YAML application examples in the `src/main/resources/flux-examples` directory of the storm-examples project.
- Step 4** Obtain the related configuration files.

NOTE

For details, this step applies only to the scenarios when other components, such as HDFS and HBase, need to be accessed for service requirements. For details about how to obtain the related configuration files, see [Storm-HDFS Development Guideline](#) or [Storm-HBase Development Guideline](#). If the services do not require the related configuration files, skip this step.

- Step 5** Obtain the related JAR packages.
 - Obtain the following JAR file from the `streaming-cql-<HD-Version>/lib` directory on the Storm client:
 - flux-core-<version>.jar
 - flux-wrappers-<version>.jar

- Obtain other related JAR packages for the services. For example, to obtain the related JAR packages required for accessing HDFS, see [Step 5](#). Other scenarios are similar.

----End

The Flux configuration file example is as follows.

It is a complete YAML file example for accessing the Kafka service.

```
name: "simple_kafka"

components:
- id: "zkHosts"#Object name
  className: "org.apache.storm.kafka.ZkHosts" #Complete class name
  constructorArgs:#Constructor
  - "${kafka.zookeeper.root.list}" #Constructor parameter

- id: "stringScheme"
  className: "org.apache.storm.kafka.StringScheme"

- id: "stringMultiScheme"
  className: "org.apache.storm.spout.SchemeAsMultiScheme"
  constructorArgs:
  - ref: "stringScheme" #A reference is used, and the value is stringScheme that has been defined.

- id: spoutConfig
  className: "org.apache.storm.kafka.SpoutConfig"
  constructorArgs:
  - ref: "zkHosts" #A reference is used.
  - "input"
  - "/kafka/input"
  - "myId"
  properties: #Use properties to set the private variable whose name is "scheme" in this object.
  - name: "scheme"
  ref: "stringMultiScheme"

- id: "defaultTopicSelector"
  className: "org.apache.storm.kafka.bolt.selector.DefaultTopicSelector"
  constructorArgs:
  - "output"

- id: "fieldNameBasedTupleToKafkaMapper"
  className: "org.apache.storm.kafka.bolt.mapper.FieldNameBasedTupleToKafkaMapper"
  constructorArgs:
  - "words" #The first input argument in the constructor
  - "count" #The second input argument in the constructor

config:
topology.workers: 1 #Set the number of workers of the topology to 1.
kafka.broker.properties: #Set the parameters related to Kafka, and the values are in the map structure.
metadata.broker.list: "${metadata.broker.list}"
producer.type: "async"
request.required.acks: "0"
serializer.class: "kafka.serializer.StringEncoder"

spouts:
- id: "kafkaSpout" #Spout name
  className: "org.apache.storm.kafka.KafkaSpout" #Spout class name
  constructorArgs: #Use a constructor to perform initialization.
  - ref: "spoutConfig" #Reference parameters as input arguments in a constructor.
  parallelism: 1 #Set the concurrency of the spout to 1.

bolts:
- id: "splitBolt"
  className: "com.huawei.storm.example.common.SplitSentenceBolt"
  parallelism: 1
```

```

- id: "countBolt"
className: "com.huawei.storm.example.kafka.CountBolt"
parallelism: 1

- id: "kafkaBolt"
className: "org.apache.storm.kafka.bolt.KafkaBolt"
configMethods: #Invoke an internal method of the object to initialize the object.
- name: "withTopicSelector" #Name of the invoked internal method
args: #Parameter required by the internal method
- ref: "defaultTopicSelector" #Only one input argument is set, and it is referenced.
- name: "withTupleToKafkaMapper" #Invoke the second internal method.
args:
- ref: "fieldNameBasedTupleToKafkaMapper"

#Define the data stream.
streams:
- name: "kafkaSpout --> splitBolt" #Name of the first data stream, which is only for display
from: "kafkaSpout" #Data stream start, whose value is kafkaSpout defined in spouts
to: "splitBolt" #Data stream end, whose value is splitBolt defined in bolts
grouping: #Define the grouping mode.
type: LOCAL_OR_SHUFFLE #The grouping mode is local_or_shuffle.

- name: "splitBolt --> countBolt" #The second data stream
from: "splitBolt"
to: "countBolt"
grouping:
type: FIELDS #The grouping mode is fields.
args: ["word"] #Parameters must be entered for the fields mode.

- name: "countBolt --> kafkaBolt" #The third data stream
from: "countBolt"
to: "kafkaBolt"
grouping:
type: SHUFFLE #The grouping mode is shuffle, and no parameter needs to be entered.

```

Running the Application and Viewing Results

- Step 1** Export the local JAR package. For details, see [Packaging IntelliJ IDEA Code](#).
- Step 2** Combine the configuration files and JAR packages obtained respectively in [Step 4](#) and [Step 5](#), and export a complete service JAR package. For details, see [Packaging Services](#).
- Step 3** Copy the developed YAML file and the related **properties** file to any directory on the host where the Storm client is located, for example, **/opt**.
- Step 4** Run the following command to submit the topology:

```
storm jar /opt/jartarget/source.jar org.apache.storm.flux.Flux --remote /opt/my-topology.yaml
```

If the services are set to be started locally, run the following command to submit the topology:

```
storm jar /opt/jartarget/source.jar org.apache.storm.flux.Flux --local /opt/my-topology.yaml
```

NOTE

Submit local services by running commands only in a normal mode environment.

If the **properties** file is used, run the following command to submit the topology:

```
storm jar /opt/jartarget/source.jar org.apache.storm.flux.Flux --remote /opt/my-topology.yaml --filter /opt/my-prop.properties
```

Step 5 After the topology is submitted successfully, log in to the Storm UI to view the topology.

----End

33.5.6 External Interfaces

- Versions of interfaces adopted by Storm-HDFS are consistent with those in the open-source community. For details, see: <https://github.com/apache/storm/tree/v1.2.1/external/storm-hdfs>.
- Versions of interfaces adopted by Storm-HBase are consistent with those in the open-source community. For details, see: <https://github.com/apache/storm/tree/v1.2.1/external/storm-hbase>.
- Versions of interfaces adopted by Storm-Kafka are consistent with those in the open-source community. For details, see: <https://github.com/apache/storm/tree/v1.2.1/external/storm-kafka>.
- Versions of interfaces adopted by Storm-JDBC are consistent with those in the open-source community. For details, see: <https://github.com/apache/storm/tree/v1.2.1/external/storm-jdbc>.
- Versions of interfaces adopted by Storm-Solr are consistent with those in the open-source community. For details, see: <https://github.com/apache/storm/tree/v1.2.1/external/storm-solr>.

33.5.7 FAQ

33.5.7.1 How Do I Use IDEA to Remotely Debug Services?

Question

How do I use IDEA to remotely debug services after submitting services by the Storm client?

Answer

The following uses WordCount as an example to demonstrate how to use IDEA to remotely debug services.

Step 1 Log in to FusionInsight Manager and choose **Cluster > Name of the desired cluster > Services > Storm**. On the page that is displayed, click the **Configuration** tab. In the search box, search for **nimbus.task.timeout.secs** and **supervisor.worker.start.timeout.secs** and increase their values. You are advised to adjust the values to the maximum ones. Add **-Xdebug -Xrunjdpw:transport=dt_socket,address=5055,suspend=n,server=y** right after the existing value of **WORKER_GC_OPTS**. Save the configuration and restart related instances.

NOTE

Before debugging the Storm program, you need to modify specified parameters on the server side. These parameters take effect after you restart the server. You are advised to debug the program in the test environment.

Step 2 After submitting a topology, enter the **Topology** page on the Storm UI. In the **Topology** page that is displayed, click the component to be debugged to enter its debugging page.

Figure 33-19 Component debugging interface

| Id | Executors | Tasks | Emitted |
|-------|-----------|-------|---------|
| spout | 5 | 5 | 591500 |

Showing 1 to 1 of 1 entries

Bolts (All time)

| Id | Executors | Tasks | Emitted | Transferred |
|-------|-----------|-------|---------|-------------|
| count | 12 | 12 | 3771500 | 0 |
| split | 8 | 8 | 3776500 | 3776500 |

Step 3 On the component debugging interface, obtain the IP address of the host where the worker process runs. If the host has multiple IP addresses, select any one of them.

Figure 33-20 Obtaining the IP addresses of the host where worker runs

Executors (All time)

| Id | Uptime | Host | Port | Actions |
|---------|------------|----------------|-------|---------|
| [24-24] | 4h 44m 50s | 189-39-172-183 | 29300 | files |
| [25-25] | 4h 40m 59s | 189-39-172-217 | 29300 | files |
| [26-26] | 4h 44m 53s | 189-39-173-167 | 29300 | files |
| [27-27] | 4h 44m 50s | 189-39-172-183 | 29300 | files |
| [28-28] | 4h 40m 59s | 189-39-172-217 | 29300 | files |

Showing 1 to 5 of 5 entries

Step 4 Open the IDEA project. On the menu bar, choose **Run > Edit Configurations**.

Step 5 In the window that is displayed, click the plus sign (+) in the upper left corner. In the drop-down list, select **Remote**. Then select the path where the source code module to be debugged resides and configure the **Host** and **Port** parameters as shown in **Figure 33-21**.

Host indicates the IP address of the host where worker runs and **Port** indicates the number of the port to be debugged. Ensure that the port is not occupied on the host.

Figure 33-21 Configuring parameters
Executors (All time)

| Id | Uptime | Host | Port | Actions |
|---------|------------|-----------------|-------|---------|
| [24-24] | 4h 44m 50s | 192-168-172-183 | 29300 | files |
| [25-25] | 4h 40m 59s | 192-168-172-217 | 29300 | files |
| [26-26] | 4h 44m 53s | 192-168-172-168 | 29300 | files |
| [27-27] | 4h 44m 50s | 192-168-172-183 | 29300 | files |
| [28-28] | 4h 40m 59s | 192-168-172-217 | 29300 | files |

Showing 1 to 5 of 5 entries

NOTE

When the port number changes, the value added to **WORKER_GC_OPTS** changes too. For example, if the port number is **8011**, the value changes to **-Xdebug -Xrunjdpw:transport=dt_socket,address=8011,suspend=n,server=y**.

Step 6 Set the debugging breakpoint.

Click in the blank area on the left of the IDEA code editing window to set corresponding code line breakpoint.

Figure 33-22 Setting the breakpoint

```

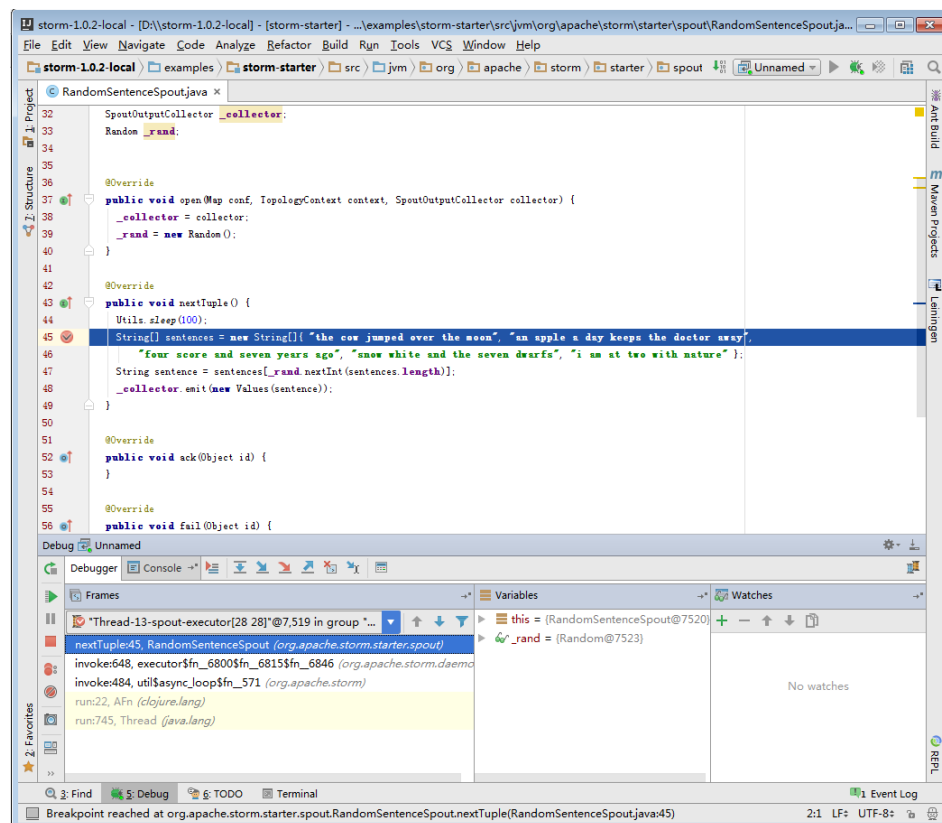
@Override
public void nextTuple() {
    Utils.sleep(100);
    String[] sentences = new String[]{ "the cow jumped over the moon", "an apple
    "four score and seven years ago", "snow white and the seven dwarfs" };
    String sentence = sentences[_rand.nextInt(sentences.length)];
    _collector.emit(new Values(sentence));
}

@Override
public void ack(Object id) {
}
    
```

Step 7 Start debugging.

On the menu bar, choose **Run > Debug'Unnamed'** to open the debugging window and start debugging, including step-by-step debugging, checking the call stack, and tracing the variable value as shown in **Figure 33-23**.

Figure 33-23 Debugging



----End

33.5.7.2 How Do I Set the Offset Correctly when Using the Old Plug-in storm-kafka?

Question

Currently, storm-kafka-client plug-in is recommended by default to achieve a safe connection with Kafka. However, the old plug-in storm-kafka is still preferred for some users and scenarios. This section describes how to correctly specify the to-be-consumed offset to avoid consuming from start point again when topology is restarted.

Answer

KafkaSpout in the old plug-in storm-kafka uses the SimpleConsumer interface of Kafka. The offset needs to be managed independently. The KafkaSpout stores the offset of each Partition in the Topic in ZooKeeper based on user-defined fields. The definition is as follows:

```
public SpoutConfig(BrokerHosts hosts, String topic, String zkRoot, String id) {
    super(hosts, topic);
    this.zkRoot = zkRoot;
    this.id = id;
}
```

In this definition, **hosts** indicates a connection string of ZooKeeper, such as `192.168.0.1:2181/kafka`. **topic** indicates the name of the topic to be consumed.

zkRoot indicates the root path for storing data in ZooKeeper, which is generally / **kafka/ {topic}**. **id** indicates the application ID, such as: app1. There are two scenarios where offset is read:

- Scenario 1

When the topology is run, the KafkaSpout stores the offset in following path of ZooKeeper: / **{zkRoot}** / **{id}** / **{partitionId}**. **zkRoot** and **id** are specified by the user, and **partitionId** is automatically obtained. By default, after the topology is started, the history offset is read from the offset storage path on ZooKeeper and is used as the start point of the current consumption. Therefore, **zkRoot** and **id** avoid consuming from start point again.

- Scenario 2

There is no such fixed **zkRoot** or **id** as set in scenario 1. As a result, the history offset cannot be read. This way, history data is re-consumed every time the topology is submitted. In this case, you can manually specify **zkRoot** or **id** using the following manner:

```
SpoutConfig spoutConfig = new SpoutConfig(hosts, inputTopicName, zkRoot, appId);
spoutConfig.ignoreZkOffsets = true;
spoutConfig.startOffsetTime = kafka.api.OffsetRequest.LatestTime();
```

You can specify **ignoreZkOffsets** and **startOffsetTime** in SpoutConfig to enable latest data to be forcibly consumed.

 **NOTE**

In practice, you are advised to use the method in scenario 1. In scenario 2, consumption does not start from the point when the commitment is successful. As a result, some data may be missing.

33.5.7.3 How Do I Handle the Error "Command line is too long" Reported When Main Is Executed for Remote Topology Submission in IntelliJ IDEA

Question

When the **Main** method is executed to remotely submit a topology in IntelliJ IDEA, the following error is reported:

Command line is too long. Shorten command line for ServiceStarter or also for Application default configuration.

Answer

Step 1 Open the `.idea\workspace.xml` file in the project.

Step 2 Find the `<component name="PropertiesComponent">` tag and add `<property name="dynamic.classpath" value="true" />`, as shown in [Figure 33-24](#).

Figure 33-24 Modifying the .idea\workspace.xml file

```
38 <component name="PropertiesComponent">
39   <property name="WebServerToolWindowFactoryState" value="false" />
40   <property name="aspect.path.notification.shown" value="true" />
41   <property name="dynamic.classpath" value="true" />
42   <property name="go.import.settings.migrated" value="true" />
43   <property name="nodejs_interpreter_path.stuck_in_default_project" value="undefined stuck path" />
44   <property name="nodejs_npm_path_reset_for_default_project" value="true" />
45   <property name="project.structure.last.edited" value="Artifacts" />
46   <property name="project.structure.proportion" value="0.15" />
47   <property name="project.structure.side.proportion" value="0.2" />
48   <property name="settings.editor.selected.configurable" value="MavenSettings" />
49 </component>
```

----End

34 YARN Development Guide (Security Mode)

34.1 Overview

Intended Audience

This document is intended for development personnel who are experienced in Java development and want to develop YARN applications.

YARN Introduction

YARN is a distributed resource management system that is used to improve resource usage in the distributed cluster environment. Resources include memory, I/O, network, and disk resources. YARN is developed to address the shortage of the original MapReduce framework. At the beginning, MapReduce committers periodically modified existing codes. As codes increase and because the original MapReduce framework was designed improperly, modification on the original MapReduce framework becomes more difficult. Therefore, MapReduce committers decided to re-design the MapReduce framework to provide a next-generation MapReduce (MRv2/Yarn) framework that supports high scalability, availability, reliability, backward compatibility, and resource usage. The next-generation MapReduce (MRv2/Yarn) framework supports more computing frameworks in addition to the MapReduce framework.

Basic Concepts

- **ResourceManager (RM)**
ResourceManager is a global resource manager that manages and allocates resources in the system. ResourceManager consists of two components: Scheduler and Applications Manager.
- **ApplicationMaster (AM)**
Each application submitted by users includes an ApplicationMaster. The ApplicationMaster provides the following functions:
 - Negotiates with the ResourceManager Scheduler to obtain resources (represented by Containers).

- Allocates resource to internal tasks.
 - Communicates with NodeManager to start or stop tasks.
 - Monitors all tasks with the running status, and applies for resources again for tasks when tasks fail to run to restart the tasks.
- **NodeManager (NM)**

NodeManager is the resource and task manager of each node. On one hand, NodeManager periodically reports resource usage of the local node and the running status of each Container to ResourceManager. On the other hand, NodeManager receives and processes requests from ApplicationMaster for starting or stopping Containers.
 - **Container**

Container is a resource abstract in YARN. Container encapsulates multidimensional resources of a node, such as memory, CPU, disk, and network resources. When ApplicationMaster applies for resources from ResourceManager, ResourceManager returns resources in a Container to ApplicationMaster.

34.2 Interfaces

34.2.1 Command

You can use YARN commands to perform operations on YARN clusters, such as starting ResourceManager, submitting applications, killing applications, querying node status, and downloading container logs.

For details about the commands, see:

<http://hadoop.apache.org/docs/r3.1.1/hadoop-yarn/hadoop-yarn-site/YarnCommands.html>

Common Commands

YARN commands can be used by common users and administrators. Common users can run a few of YARN commands, such as **jar** and **logs**. Administrators have the permission to run most YARN commands.

Users can run the following command to query usage of YARN.

yarn --help

Usage: Go to any directory on the YARN client, execute source command to import the environment variables, and run the command. The command format is as follows.

yarn [--config *confdir*] *COMMAND*

Table 34-1 describes the commands.

NOTE

The version 8.0.2.1 is used as an example. Replace it with the actual version number.

Table 34-1 Common commands

| Command | Description |
|------------------------------|---|
| resourcemanager | <p>Runs a ResourceManager.</p> <p>Notice: Before running commands, for example, the following two commands, on the server as user omm (the client must have the execute permission of user omm), export environment variables first.</p> <ul style="list-style-type: none"> • export YARN_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.1.0.1/1_10_ResourceManager/etc • export HADOOP_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.1.0.1/1_10_ResourceManager/etc |
| nodemanager | <p>Runs a NodeManager.</p> <p>Notice: Before running commands, for example, the following two commands, on the server as user omm (the client must have the execute permission of user omm), export environment variables first.</p> <ul style="list-style-type: none"> • export YARN_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.1.0.1/1_10_NodeManager/etc • export export HADOOP_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.1.0.1/1_10_NodeManager/etc |
| rmdadmin | Runs administrator's tool for dynamically updating information. |
| version | Displays the version. |
| jar <jar> | Runs a JAR file. |
| logs | Obtains container logs. |
| classpath | Displays the class path of the Hadoop JAR package and other library files. |
| daemonlog | Obtains or sets the service log level. |
| CLASSNAME | Runs a class named by <i>CLASSNAME</i> . |
| top | Runs the cluster usage monitor tool. |
| -Dmapreduce.job.hdfs-servers | <p>If OBS is connected, but the server still uses HDFS, you need to explicitly use this parameter in the command line to specify the HDFS address. The format is hdfs://{NAMESERVICE}. Replace <i>NAMESERVICE</i> with the HDFS NameService name.</p> <p>If the current HDFS has multiple NameService instances, you need to specify all NameService instances and separate them with commas (,), for example, hdfs://nameservice1,hdfs://nameservice2.</p> |

Superior Scheduler Command

Superior Scheduler Engine provides a CLI that display detail information of Superior Scheduler Engine. For executing superior command we need to use the "<HADOOP_HOME>/bin/superior" script.

Here is the format of "superior" command.

```
<HADOOP_HOME>/bin/superior
```

```
Usage: superior [COMMAND | -help]
  Where COMMAND is one of:
  resourcepool      prints resource pool status
  queue             prints queue status
  application        prints application status
  policy            prints policy status
```

Most commands print help when invoked without parameters.

- Superior **resourcepool** command:

This command displays resourcepool and associated policy related status and configuration information.

NOTE

Superior resourcepool command can be used only by admin user and user with yarn admin rights.

Usage output:

```
>superior resourcepool
```

```
Usage: resourcepool [-help]
      [-list]
      [-status <resourcepoolname>]
-help          prints resource pool usage
-list          prints all resource pool summary report
-status <resourcepoolname> prints status and configuration of specified
              resource pool
```

- **resourcepool -list** prints resource pool summary in a table format. Here is an example:

```
> superior resourcepool -list
NAME      NUMBER_MEMBER  TOTAL_RESOURCE      AVAILABLE_RESOURCE
Pool1     4               vcores 30,memory 1000  vcores 21,memory 80
Pool2     100             vcores 100,memory 12800 vcores 30,memory 1000
default   2               vcores 64,memory 128   vcores 40,memory 28
```

- **resourcepool -status <resourcepoolname>** prints resource pool detail information in a list format. Here is an example:

```
> superior resourcepool -status default
NAME: default
DESCRIPTION: System generated resource pool
TOTAL_RESOURCE: vcores 64,memory 128
AVAILABLE_RESOURCE: vcores 40,memory 28
NUMBER_MEMBER: 2
MEMBERS: node1,node2
CONFIGURATION:
|-- RESOURCE_SELECT:
|_ RESOURCES:
```

- Superior **queue** command

This command displays hierarchy queue information.

Usage output:

```
>superior queue

Usage: queue [-help]
        [-list] [-e] [[-name <queue_name>] [-r|-c]]
        [-status <queue_name>]
-c          only work with -name <queue_name> option. If this
           option is used, command will print information of
           specified queue and its direct children.
-e          only work with -list or -list -name option. If
           this option is used, command will print effective
           state of specified queue and all of its
           descendants.
-help      prints queue sub command usage
-list      prints queue summary report. This option can work
           with -name <queue_name> and -r options.
-name <queue_name> print specified queue, this can work with -r
           option. By default, it will print queue's own
           information. When -r is defined, command will
           print all of its descendant queues. When -c is
           defined, it will print its direct children queues.
-r         only work with -name <queue_name> option. If this
           option is used, command will print information of
           specified queue and all of its descendants.
-status <queue_name> prints status of specified queue
```

- **queue -list** prints a table format of queue summary information. When command displays, command will display them based on queue hierarchy fashion. With SUBMIT ACL or ADMIN ACL rights for queue, user will be able to see queues. Here is an example:

```
> superior queue -list
NAME      STATE      NRUN_APP  NPEND_APP  NRUN_CONTAINER
NPEND_REQUEST RES_INUSE  RES_REQUEST
root      OPEN|ACTIVE  10      20      100      200      vcores 100,memory
1000     vcores 200,memory 2000
root.Q1   OPEN|ACTIVE  5       10      50      100      vcores 50,memory
500     vcores 100,memory 1000
root.Q1.Q11 OPEN|ACTIVE  5       10      50      100      vcores 50,memory
500     vcores 100,memory 1000
root.Q1.Q12 CLOSE|INACTIVE 0       0       0       0       vcores 0,memory
0       vcores 0,memory 0
root.Q2   OPEN|INACTIVE 5       10      50      100      vcores 50,memory
500     vcores 100,memory 1000
root.Q2.Q21 OPEN|ACTIVE  5       10      50      100      vcores 50,memory
500     vcores 100,memory 1000
```

- **queue -list -name root.Q1** will display root.Q1 only.

```
> superior queue -list -name root.Q1
NAME      STATE      NRUN_APP  NPEND_APP  NRUN_CONTAINER
NPEND_REQUEST RES_INUSE  RES_REQUEST
root.Q1   OPEN|ACTIVE  5       10      50      100      vcores 50,memory
500     vcores 100,memory 1000
```

- **queue -list -name root.Q1 -r** will print out root.Q1 and all of its descendents.

```
> superior queue -list -name root.Q1 -r
NAME      STATE      NRUN_APP  NPEND_APP  NRUN_CONTAINER
NPEND_REQUEST RES_INUSE  RES_REQUEST
root.Q1   OPEN|ACTIVE  5       10      50      100      vcores 50,memory
500     vcores 100,memory 1000
root.Q1.Q11 OPEN|ACTIVE  5       10      50      100      vcores 50,memory
500     vcores 100,memory 1000
root.Q1.Q12 CLOSE|INACTIVE 0       0       0       0       vcores 0,memory
0       vcores 0,memory 0
```

- **queue -list -name root -c** will print out root and all of its direct children

```
> superior queue -list -name root -c
NAME      STATE      NRUN_APP  NPEND_APP  NRUN_CONTAINER
NPEND_REQUEST RES_INUSE  RES_REQUEST
root      OPEN|ACTIVE  10      20      100      200      vcores
100,memory 1000 vcores 200,memory 2000
```

| | | | | | | |
|---------------|------------------------|---|----|----|-----|--------|
| root.Q1 | OPEN ACTIVE | 5 | 10 | 50 | 100 | vcores |
| 50,memory 500 | vcores 100,memory 1000 | | | | | |
| root.Q2 | OPEN INACTIVE | 5 | 10 | 50 | 100 | vcores |
| 50,memory 500 | vcores 100,memory 1000 | | | | | |

- **queue -status <queue_name>** will display detail queue status and configuration.

With SUBMIT ACL, user will be able to see details except ACLS of queue. User with ADMIN ACL rights for queue will be able to see queue details including ACL.

```
> superior queue -status root.Q1
NAME: root.Q1
OPEN_STATE:CLOSED
ACTIVE_STATE: INACTIVE
EOPEN_STATE: CLOSED
EACTIVE_STATE: INACTIVE
LEAF_QUEUE: Yes
NUMBER_PENDING_APPLICATION: 100
NUMBER_RUNNING_APPLICATION: 10
NUMBER_PENDING_REQUEST: 10
NUMBER_RUNNING_CONTAINER: 10
NUMBER_RESERVED_CONTAINER: 0
RESOURCE_REQUEST: vcores 3,memory 3072
RESOURCE_INUSE: vcores 2,memory 2048
RESOURCE_RESERVED: vcores 0,memory 0
CONFIGURATION:
|-- DESCRIPTION: Spark session queue
|-- MAX_PENDING_APPLICATION: 10000
|--MAX_RUNNING_APPLICATION: 1000
|--ALLOCATION_ORDER_POLICY: FIFO
|--DEFAULT_RESOURCE_SELECT: label1
|--MAX_MASTER_SHARE: 10%
|--MAX_RUNNING_APPLICATION_PER_USER : -1
|--MAX_ALLOCATION_UNIT: vcores 32,memory 12800
|--ACL_USERS: user1,user2
|--ACL_USERGROUPS: usergroup1,usergroup2
|-- ACL_ADMINS: user1
|--ACL_ADMININGROUPS: usergroup1
```

- Superior **application** command

This command displays application related information.

Usage output:

```
>superior application

Usage: application [-help]
           [-list]
           [-status <application_id>]
-help           prints application sub command usage
-list          prints all application summary report
-status <application_id> prints status of specified application
```

With the view access rights to application user can see application related information.

- **application -list** provides summary information of all application in a table format. Here is an example:

```
> superior application -list
ID          QUEUE          USER          NRUN_CONTAINER
NPEND_REQUEST  NRSV_CONTAINER  RES_INUSE
RES_REQUEST  RES_RESERVED
application_1482743319705_0005          root.SEQ.queueB  hbase  1
100          0          vcores 1,memory 1536          vcores 2000,memory
409600          vcores 0,memory 0
application_1482743319705_0006          root.SEQ.queueB  hbase  0
1          0          vcores 0,memory 0          vcores 1,memory
1536          vcores 0,memory 0
```

- ***application -status <app_id>*** command displays detail information of specified application. Here is an example:

```
> superior application -status application_1443067302606_0609
ID: application_1443067302606_0609
QUEUE: root.Q1.Q11
USER: cchen
RESOURCE_REQUEST: vcores 3,memory 3072
RESOURCE_INUSE: vcores 2,memory 2048
RESOURCE_RESERVED:vcores 1, memory 1024
NUMBER_RUNNING_CONTAINER: 2
NUMBER_PENDING_REQUEST: 3
NUMBER_RESERVED_CONTAINER: 1
MASTER_CONTAINER_ID: application_1443067302606_0609_01
MASTER_CONTAINER_RESOURCE: node1.domain.com
BLACKLIST: node5,node8
DEMANDS:
|-- PRIORITY: 20
|-- MASTER: true
|-- CAPABILITY: vcores 2, memory 2048
|-- COUNT: 1
|-- RESERVED_RES : vcores 1, memory 1024
|-- RELAXLOCALITY: true
|-- LOCALITY: node1/1
|-- RESOURCESELECT: label1
|-- PENDINGREASON: "application limit reached"
|-- ID: application_1443067302606_0609_03
|-- RESOURCE: node1.domain.com
|-- RESERVED_RES: vcores 1, memory 1024
|
|--PRIORITY: 1
|-- MASTER: false
|-- CAPABILITY: vcores 1,memory 1024
|-- COUNT: 2
|-- RESERVED_RES: vcores 0, memory 0
|-- RELAXLOCLITY: true
|--LOCALITY: node1/1,node2/1,rackA/2
|-- RESOURCESELECT: label1
|-- PENDINGREASON: "no available resource"
CONTAINERS:
|-- ID: application_1443067302606_0609_01
|-- RESOURCE: node1.domain.com
|-- CAPABILITY: vcores 1,memory 1024
|
|-- ID: application_1443067302606_0609_02
|-- RESOURCE: node2.domain.com
|-- CAPABILITY: vcores 1,memory 1024
```

- Superior ***policy*** command

This command displays policy related information.

NOTE

Superior policy command can be used only by admin user and user with yarn admin rights.

Usage output:

```
>superior policy

Usage: policy [-help]
        [-list <resourcepoolname>] [-u] [-detail]
        [-status <resourcepoolname>]
-detail          only work with -list option to show a
                  summary information of resource pool
                  distribution on queues, including reserve,
                  minimum and maximum
-help           prints policy sub command usage
-list <resourcepoolname> prints a summary information of resource
                  pool distribution on queue
```



```
-status <resourcepoolname> prints pool distribution policy
configuration and status of specified
resource pool
-u only work with -list option to show a
summary information of resource pool
distribution on queues and also user
accounts
```

- ***policy -list <resourcepoolname>*** print out a summary of queue distribution information. Here is an example:

```
>superior policy -list default
NAME: default
TOTAL_RESOURCE: vcores 16,memory 16384
AVAILABLE_RESOURCE: vcores 16,memory 16384

NAMERES_INUSERES_REQUEST
root.defaultvcores 0,memory 0vcores 0,memory 0
root.productionvcores 0,memory 0vcores 0,memory 0
root.production.BU1vcores 0,memory 0vcores 0,memory 0
root.production.BU2 vcores 0,memory 0vcores 0,memory 0
```

- ***policy -list <resourcepoolname> -u*** prints out also user level summary information

```
> superior policy -list default -u
NAME: default
TOTAL_RESOURCE: vcores 16,memory 16384
AVAILABLE_RESOURCE: vcores 16,memory 16384

NAMERES_INUSERES_REQUEST
root.defaultvcores 0,memory 0vcores 0,memory 0
root.default.[_others_]vcores 0,memory 0vcores 0,memory 0
root.productionvcores 0,memory 0vcores 0,memory 0
root.production.BU1vcores 0,memory 0vcores 0,memory 0
root.production.BU1.[_others_]vcores 0,memory 0vcores 0,memory 0
root.production.BU2vcores 0,memory 0vcores 0,memory 0
root.production.BU2.[_others_]vcores 0,memory 0vcores 0,memory 0
```

- ***policy -status <resourcepoolname>*** print out policy detail of specified resource pool. Here is an example:

```
> superior policy -status pool1
NAME: pool1
TOTAL_RESOURCE: vcores 64,memory 128
AVAILABLE_RESOURCE: vcores 40,memory 28
QUEUES:
|-- NAME: root.Q1
|-- RESOURCE_USE: vcores 20, memory 1000
|-- RESOURCE_REQUEST: vcores 2,memory 100
|--RESERVE: vcores 10, memory 4096
|--MINIMUM: vcore 11, memory 4096
|--MAXIMUM: vcores 500, memory 100000
|--CONFIGURATION:
|-- SHARE: 50%
|-- RESERVE: vcores 10, memory 4096
|-- MINIMUM: vcores 11, memory 4096
|-- MAXIMUM: vcores 500, memory 100000
|-- QUEUES:
|-- NAME: root.Q1.Q11
|-- RESOURCE_USE: vcores 15, memory, 500
|-- RESOURCE_REQUEST: vcores 1, memory 50
|-- RESERVE: vcores 0, memory 0
|-- MINIMUM: vcores 0, memory 0
|-- MAXIMUM: vcores -1, memory -1
|-- USER_ACCOUNTS:
|-- NAME: user1
|-- RESOURCE_USE: vcores 1, memory 10
|-- RESOURCE_REQUEST: vcores 1, memory 50
|
|-- NAME: OTHERS
|--RESOURCE_USE: vcores 0, memory 0
|-- RESOURCE_REQUEST: vcores 0, memory 0
```

```

|-- CONFIGURATION:
|-- SHARE: 100%
|-- USER_POLICY:
|-- NAME: user1
|-- WEIGHT: 10
|
|-- NAME: OTHERS
|-- WEIGHT: 1
|-- MAXIMUM: vcores 10, memory 1000
    
```

34.2.2 Java API

For details about YARN application programming interfaces (APIs), see:

<http://hadoop.apache.org/docs/r3.1.1/api/index.html>.

Common Interfaces

Common YARN Java classes are as follows:

- **ApplicationClientProtocol**

This class is used between the client and ResourceManager. The client submits applications to ResourceManager, queries the running status of applications, and kills applications using this protocol.

Table 34-2 Common interfaces of ApplicationClientProtocol

| Interface | Description |
|---|---|
| forceKillApplication(KillApplicationRequest request) | The client requests ResourceManager to stop a submitted task through this interface. |
| getApplicationAttemptReport(GetApplicationAttemptReportRequest request) | The client obtains the report of specified application attempts from ResourceManager through this interface. |
| getApplicationAttempts(GetApplicationAttemptsRequest request) | The client obtains the report of all application attempts from ResourceManager through this interface. |
| getApplicationReport(GetApplicationReportRequest request) | The client obtains an application report from ResourceManager through this interface. |
| getApplications(GetApplicationsRequest request) | The client obtains information about applications that meet filtering conditions from ResourceManager through this interface. |
| getClusterMetrics(GetClusterMetricsRequest request) | The client obtains metrics of clusters from ResourceManager through this interface. |
| getClusterNodes(GetClusterNodesRequest request) | The client obtains information about all nodes in a cluster from ResourceManager through this interface. |

| Interface | Description |
|---|---|
| getContainerReport(GetContainerReportRequest request) | The client obtains the report of a Container from ResourceManager through this interface. |
| getContainers(GetContainersRequest request) | The client obtains the report of all Containers of an application attempt from ResourceManager through this interface. |
| getDelegationToken(GetDelegationTokenRequest request) | The client obtains the delegation token through this interface. The delegation token is used by the container to access related services. |
| getNewApplication(GetNewApplicationRequest request) | The client obtains a new application ID through this interface for submitting a new application. |
| getQueueInfo(GetQueueInfoRequest request) | The client obtains queue information from ResourceManager through this interface. |
| getQueueUserAcls(GetQueueUserAclsInfoRequest request) | The client obtains queue access permission information about the current user from ResourceManager through this interface. |
| moveApplicationAcrossQueues(MoveApplicationAcrossQueuesRequest request) | This interface is used to move an application to a new queue. |
| submitApplication(SubmitApplicationRequest request) | The client submits a new application to ResourceManager through this interface. |

- ApplicationMasterProtocol

This class is used between ApplicationMaster and ResourceManager. ApplicationMaster registers with ResourceManager, and applies for resources and obtains the running status of each task from ResourceManager using this protocol.

Table 34-3 Common interfaces of ApplicationMasterProtocol

| Interface | Description |
|---|--|
| allocate(AllocateRequest request) | ApplicationMaster submits a resource allocation request through this interface. |
| finishApplicationMaster(FinishApplicationMasterRequest request) | ApplicationMaster notifies ResourceManager of running success or failure through this interface. |
| registerApplicationMaster(RegisterApplicationMasterRequest request) | ApplicationMaster registers with ResourceManager through this interface. |

- ContainerManagementProtocol
This class is used between ApplicationMaster and NodeManager. ApplicationMaster requests NodeManager to start or terminate a Container or queries the Container running status using this protocol.

Table 34-4 Common interfaces of ContainerManagementProtocol

| Interface | Description |
|---|---|
| getContainerStatuses(GetContainerStatusesRequest request) | ApplicationMaster obtains the current status of the Containers from NodeManager through this interface. |
| startContainers(StartContainersRequest request) | ApplicationMaster requests NodeManager to start Containers through this interface. |
| stopContainers(StopContainersRequest request) | ApplicationMaster requests NodeManager to stop a series of allocated Containers through this interface. |

34.2.3 REST API

Function Description

Users can use the application programming interface (API) of Representational State Transfer (REST) to query more information about YARN jobs. Currently, on the REST API provided by YARN, you can only query some resources or jobs. For details of the HTTP REST API, see the following official guidelines:

<http://hadoop.apache.org/docs/r3.1.1/hadoop-yarn/hadoop-yarn-site/WebServicesIntro.html>

Preparing Running Environment

1. Install a client on the node. For example, install a client in the **/opt/client** directory. See details in "Software Installation > Initial Configuration > Configuring Client > Installing a Client".
2. Go to the **/opt/client** directory where the client is installed and run the following commands to initiate environment variables:

```
source bigdata_env
```

```
kinit component service user
```

NOTE

The validity duration of kinit authentication is 24 hours. After 24 hours, you need to re-authenticate the sample with the kinit to restart the sample.

3. HTTPS-based access is different from HTTP-based access. When you access Yarn using HTTPS, you must ensure that the SSL protocol supported by the **curl** command is supported by the cluster because SSL security encryption is used. If the cluster does not support the SSL protocol, change the SSL protocol in the cluster. For example, if the **Curl** command only supports the TLSv1 protocol, modify the protocol configuration performing following measures:

Log in to the FusionInsight Manager portal and choose **Cluster > Name of the desired cluster > Service > Yarn > Configuration > All Configurations**. Type **hadoop.ssl.enabled.protocols** in the research box, check whether the parameter value contains **TLSv1**. If the parameter value does not contain **TLSv1**, add **TLSv1** in the **hadoop.ssl.enabled.protocols** configuration item, and clear the value of **ssl.server.exclude.cipher.list**. Otherwise, Yarn cannot be accessed by using HTTPS. Then click **Save Configuration** and select **Restart the affected services or instances**. Restart the service.

 **NOTE**

TLSv1 has security vulnerabilities. Exercise caution when using it.

Procedure

1. Query the information of the jobs that run on the Yarn.

- Command:

```
curl -k -i --negotiate -u : "https://10-120-85-2:8090/ws/v1/cluster/apps/"
```

 **NOTE**

- **10-120-85-2**: host name of the active ResourceManager node.
You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Instance**, and view the **Host Name** of **ResourceManager(Active)**.
- **8090**: port number of the ResourceManager.
You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Configurations > All Configurations**, search for and obtain the value of **yarn.resourcemanager.webapp.https.port**.

- If users have the admin permission, they can check jobs in some columns.

 **NOTE**

If the current component uses Ranger for permission control, you need to configure permission management policies based on Ranger.

- The result is displayed as follows:

```
{
  "apps": {
    "app": [
      {
        "id": "application_1461743120947_0001",
        "user": "spark",
        "name": "Spark-JDBCServer",
        "queue": "default",
        "state": "RUNNING",
        "finalStatus": "UNDEFINED",
        "progress": 10,
        "trackingUI": "ApplicationMaster",
        "trackingUrl": "https://10-120-85-2:8090/proxy/application_1461743120947_0001/",
        "diagnostics": "AM is launched. ",
        "clusterId": 1461743120947,
        "applicationType": "SPARK",
        "applicationTags": "",
        "startedTime": 1461804906260,
        "finishedTime": 0,
        "elapsedTime": 6888848,
        "amContainerLogs": "https://10-120-85-2:8044/node/containerlogs/
container_e12_1461743120947_0001_01_000001/spark",
        "amHostHttpAddress": "10-120-85-2:8044",
        "allocatedMB": 1024,
        "allocatedVCores": 1,
        "runningContainers": 1,
```

```

"memorySeconds": 7053309,
"vcoreSeconds": 6887,
"preemptedResourceMB": 0,
"preemptedResourceVCores": 0,
"numNonAMContainerPreempted": 0,
"numAMContainerPreempted": 0,
"resourceRequests": [
  {
    "capability": {
      "memory": 1024,
      "virtualCores": 1
    },
    "nodeLabelExpression": "",
    "numContainers": 0,
    "priority": {
      "priority": 0
    },
    "relaxLocality": true,
    "resourceName": "*"
  }
],
"logAggregationStatus": "NOT_START",
"amNodeLabelExpression": ""
},
{
  "id": "application_1461722876897_0002",
  "user": "admin",
  "name": "QuasiMonteCarlo",
  "queue": "default",
  "state": "FINISHED",
  "finalStatus": "SUCCEEDED",
  "progress": 100,
  "trackingUI": "History",
  "trackingUrl": "https://10-120-85-2:8090/proxy/application_1461722876897_0002/",
  "diagnostics": "Attempt recovered after RM restart",
  "clusterId": 1461743120947,
  "applicationType": "MAPREDUCE",
  "applicationTags": "",
  "startedTime": 1461741052993,
  "finishedTime": 1461741079483,
  "elapsedTime": 26490,
  "amContainerLogs": "https://10-120-85-2:8044/node/containerlogs/
container_e11_1461722876897_0002_01_000001/admin",
  "amHostHttpAddress": "10-120-85-2:8044",
  "allocatedMB": -1,
  "allocatedVCores": -1,
  "runningContainers": -1,
  "memorySeconds": 158664,
  "vcoreSeconds": 52,
  "preemptedResourceMB": 0,
  "preemptedResourceVCores": 0,
  "numNonAMContainerPreempted": 0,
  "numAMContainerPreempted": 0,
  "amNodeLabelExpression": ""
}
]
}

```

– Result analysis:

On the interface, you can query the information of jobs that are running on the YARN and obtain the common information that is displayed in [Table 1](#).

Table 34-5 Common information of jobs running on Yarn

| Information | Description |
|-----------------|--|
| user | Indicates the user who runs the job. |
| applicationType | Indicates the application types, such as MAPREDUCE or SPARK. |
| finalStatus | Indicates whether a job is executed successfully. |
| elapsedTime | Indicates the time to run a job. |

2. Obtain the overall information of YARN resources.

- Command:

```
curl -k -i --negotiate -u : "https://10-120-85-102:8090/ws/v1/cluster/metrics"
```

- The result is displayed as follows:

```
{
  "clusterMetrics": {
    "appsSubmitted": 2,
    "appsCompleted": 1,
    "appsPending": 0,
    "appsRunning": 1,
    "appsFailed": 0,
    "appsKilled": 0,
    "reservedMB": 0,
    "availableMB": 23552,
    "allocatedMB": 1024,
    "reservedVirtualCores": 0,
    "availableVirtualCores": 23,
    "allocatedVirtualCores": 1,
    "containersAllocated": 1,
    "containersReserved": 0,
    "containersPending": 0,
    "totalMB": 24576,
    "totalVirtualCores": 24,
    "totalNodes": 3,
    "lostNodes": 0,
    "unhealthyNodes": 0,
    "decommissionedNodes": 0,
    "rebootedNodes": 0,
    "activeNodes": 3,
    "rmMainQueueSize": 0,
    "schedulerQueueSize": 0,
    "stateStoreQueueSize": 0
  }
}
```

- Result analysis:

On the interface, users can query the common information of jobs that are running in the cluster, as displayed in [Table 2](#).

Table 34-6 Common information of jobs running in the cluster

| Information | Description |
|---------------|--|
| appsSubmitted | Indicates the number of jobs that have been submitted. |

| Information | Description |
|-------------------|--|
| appsCompleted | Indicates the number of jobs that have been completed. |
| appsPending | Indicates the number of jobs that have been suspended. |
| appsRunning | Indicates the number of jobs that are running. |
| appsFailed | Indicates the number of jobs that have failed. |
| appsKilled | Indicates the number of jobs that have been killed. |
| totalMB | Indicates the total memory of Yarn resources. |
| totalVirtualCores | Indicates the total VCores of Yarn resources. |

34.2.4 REST APIs of Superior Scheduler

Function Description

The REST/HTTP server is part of Superior Scheduler on YARN Resource Manager host and leverage existing YARN resource manager web service port. In the section below, we will denote this YARN *address:port* as *SS_REST_SERVER*.

Below uses HTTPS as part of URL and only HTTPS will be supported.

Superior Scheduler Interfaces

- **Query Application**
 - Query **all** application within scheduler engine.
 - URL
GET https://<SS_REST_SERVER>/ws/v1/sscheduler/applications/list
 - 📖 **NOTE**
SS_REST_SERVER indicates *ResourceManager IP address:Port number*.
 - ResourceManager IP address: You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Instance**, and view the service IP address of any ResourceManager.
 - Port: HTTPS port number of the ResourceManager. You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Configurations > All Configurations**, search for and view the value of **yarn.resourcemanager.webapp.https.port**.
 - Input
None.

■ **Output**

JSON Response:

```
{
  "applicationlist": [
    {
      "id": "1020201_0123_12",
      "queue": "root.Q1.Q11",
      "user": "cchen",
      "resource_request": {
        "vcores": 10,
        "memory": 100
      },
      "resource_inuse": {
        "vcores": 100,
        "memory": 2000
      },
      "number_running_container": 100,
      "number_pending_request": 10
    },
    {
      "id": "1020201_0123_15",
      "queue": "root.Q2.Q21",
      "user": "Jason",
      "resource_request": {
        "vcores": 4,
        "memory": 100
      },
      "resource_inuse": {
        "vcores": 20,
        "memory": 200
      },
      "resource_reserved": {
        "vcores": 10,
        "memory": 100
      },
      "number_running_container": 20,
      "number_pending_container": 4,
      "number_reserved_container": 2
    }
  ]
}
```

Table 34-7 Parameters of all application

| Attribute | Type | Description |
|--------------------------|--------|---|
| applicationlist | array | Array of application IDs. |
| queue | String | Name of queue application. |
| user | String | Name of user who submits application. |
| resource_request | object | Currently requested resource, including vcores, memory, and so on. |
| resource_inuse | object | Currently resource in use, including vcores, memory, and so on. |
| resource_reserved | object | Currently reserved resource, including vcores, memory, and so on. |
| number_running_container | int | Total number of running containers. This maps to superior engine decisions. |

| Attribute | Type | Description |
|---------------------------|--------|---|
| number_pending_request | int | Total number of pending requests, this is sum of all counts in all demands of allocation. |
| number_reserved_container | int | Total number of reserved containers, this maps to superior engine decisions. |
| id | String | Application ID. |

- Query **single** application within scheduler engine.
 - URL
GET https://<SS_REST_SERVER>/ws/v1/sscheduler/applications/{application_id}

- Input
None.

- Output

JSON Response:

```
{
  "applicationlist": [
    {
      "id": "1020201_0123_12",
      "queue": "root.Q1.Q11",
      "user": "cchen",
      "resource_request": {
        "vcores" : 3,
        "memory" : 3072
      },
      "resource_inuse": {
        "vcores" : 100,
        "memory" : 2048
      },
      "number_running_container": 2,
      "number_pending_request": 3,
      "number_reserved_container": 1,
      "master_container_id": 23402_3420842
      "master_container_resource": node1.domain.com
      "blacklist": [
        {
          "resource": "node5"
        },
        {
          "resource": "node8"
        }
      ],
      "demand": [
        {
          "priority": 1,
          "ismaster": true,
          "capability": {
            "vcores": 2,
            "memory": 2048
          },
          "count": 1,
          "relaxlocality": true,
          "locality": [
            {
              "target": "node1",
              "count": 1,

```

```
"strict": false
}
],
"resourceselect": "label1",
"pending_reason": "application limit reached",
"reserved_resource": {
  "vcores": 1,
  "memory": 1024
},
"reservations": [
  {
    "id": "23402_3420878",
    "resource": "node1.domain.com",
    "reservedAmount": {
      "vcores": 1,
      "memory": 1024
    }
  }
]
},
{
  "priority": 1,
  "ismaster": false,
  "capability": {
    "vcores": 1,
    "memory": 1024
  },
  "count": 2,
  "relaxlocality": true,
  "locality": [
    {
      "target": "node1",
      "count": 1,
      "strict": false
    },
    {
      "target": "node2",
      "count": 1,
      "strict": false
    },
    {
      "target": "rackA",
      "count": 2,
      "strict": false
    }
  ],
  "resourceselect": "label1",
  "pending_reason": "no available resource"
}
],
"containers": [
  {
    "id": "23402_3420842",
    "resource": "node1.domain.com",
    "capability": {
      "vcores": 1,
      "memory": 1024
    }
  },
  {
    "id": "23402_3420853",
    "resource": "node2.domain.com",
    "capability": {
      "vcores": 1,
      "memory": 1024
    }
  }
]
}
}
```

- Exceptions
 - Application not found.

Table 34-8 Parameters of single application

| Attribute | Type | Description |
|---------------------------|---------|---|
| application | object | Application object. |
| id | String | Application ID. |
| queue | String | Name of queue application. |
| user | String | Name of user who submits application. |
| resource_request | object | Currently requested resource, including vcores, memory, and so on. |
| resource_inuse | object | Currently resource in use, including vcores, memory, and so on. |
| resource_reserved | object | Currently reserved resource, including vcores, memory, and so on. |
| number_running_container | int | Total number of running containers. This maps to superior engine decisions. |
| number_pending_request | int | Total number of pending requests, this is sum of all counts in all demands of allocation. |
| number_reserved_container | int | Total number of reserved containers, this maps to superior engine decisions. |
| master_container_id | String | The master container ID. |
| master_container_resource | String | The host name that master container running on. |
| demand | array | Array of demand objects. |
| priority | int | Priority of demand. |
| ismaster | boolean | Is the demand corresponding to "application master". |
| capability | object | Capability object. |
| vcores, memory, .. | int | Numeric consumable resource attributes, defining the allocation "unit" for this demand. |
| count | int | Number of unit required. |
| relaxlocality | boolean | Locality requirement is preference, and not mandatory if it cannot be satisfied. |

| Attribute | Type | Description |
|------------------------------|---------|---|
| locality | object | Locality object. |
| target | string | Locality target's name (that is, node1, rack1..). |
| count | int | Number of resource "unit" required with this locality requirement. |
| strict | boolean | If this particular locality is mandatory or not. |
| resourceselect | String | Resource selection expression for the demand. |
| pending_reason | String | Reason why this demand is outstanding. |
| resource_reserved | object | Currently reserved resource for this demand, including vcores, memory, and so on. |
| reservations | array | Array of reserved container objects. |
| reservations:id | String | Reserved container ID. |
| reservations:resource | String | Where the container is allocated. |
| reservations:reserveAmount | object | The reserved amount of this reservation. |
| containers | array | Array of allocated container objects. |
| containers:id | String | Container ID. |
| containers:resource | String | Where the container is allocated. |
| containers:capability | object | Capability object. |
| containers:vcores, memory... | int | Numeric consumable resource attributes allocated to this container. |

- Query Queue
 - Query **all** queues within scheduler engine, including leaf and all middle queues.
 - URL
GET https://<SS_REST_SERVER>/ws/v1/scheduler/queues/list
 - Input
None.
 - Output
JSON Response:

```

{
  "queuelist": [
    {
      "name": "root.default",
      "eopen_state": "OPEN",
      "eactive_state": "ACTIVE",
      "open_state": "OPEN",
      "active_state": "ACTIVE",
      "number_pending_application": 2,
      "number_running_application": 10,
      "number_pending_request": 2,
      "number_running_container": 10,
      "number_reserved_container": 1,
      "resource_inuse" {
        "vcores": 10,
        "memory": 10240
      },
      "resource_request" {
        "vcores": 2,
        "memory": 2048
      },
      "resource_reserved" {
        "vcores": 1
        "memory": 1024
      }
    },
    {
      "name": "root.dev",
      "eopen_state": "OPEN",
      "eactive_state": "INACTIVE",
      "open_state": "OPEN",
      "active_state": "INACTIVE",
      "number_pending_application": 2,
      "number_running_application": 10,
      "number_pending_request": 2,
      "number_running_container": 10,
      "number_reserved_container": 0,
      "resource_inuse" {
        "vcores": 10,
        "memory": 10240
      },
      "resource_request" {
        "vcores": 2,
        "memory": 2048
      },
      "resource_reserved" {
        "vcores": 0
        "memory": 0
      }
    },
    {
      "name": "root.qa",
      "eopen_state": "CLOSED",
      "eactive_state": "ACTIVE",
      "open_state": "CLOSED",
      "active_state": "ACTIVE",
      "number_pending_application": 2,
      "number_running_application": 10,
      "number_pending_request": 2,
      "number_running_container": 10,
      "number_reserved_container": 0,
      "resource_inuse" {
        "vcores": 10,
        "memory": 10240
      },
      "resource_request" {
        "vcores": 2,
        "memory": 2048
      }
    }
  ]
}

```

```

    },
    "resource_reserved" {
      "vcores": 1
      "memory": 1024
    }
  },
]
}

```

Table 34-9 Parameters of all queues

| Attribute | Type | Description |
|----------------------------|--------|--|
| queuelist | array | Array of queue names. |
| name | String | Queue name. |
| open_state | String | This is inner state (self state) of queue. Indicate either "OPEN" or "CLOSED" effective state of the queue. A "CLOSED" queue does not accept any new allocation request. |
| eopen_state | String | This is outer state of queue. An effective state is combination of queue own state and its ancestors. A "CLOSED" queue does not accept any new allocation request. |
| active_state | String | This is inner state (self state) of queue. Indicate either "ACTIVE" or "INACTIVE" state of the queue. An "INACTIVE" queue does not schedule any application. |
| eactive_state | String | This is outer state of queue. An effective state is combination of queue own state and its ancestors. An "INACTIVE" queue does not schedule any application. |
| number_pending_application | int | Total number of pending applications. |
| number_running_application | int | Total number of running applications. |
| number_pending_request | int | Total number of pending request. |
| number_running_container | int | Total number of running containers. |
| number_reserved_container | int | Total number of reserved containers. |
| resource_request | object | Pending resource requests within queue in a form of vcores, memory, and so on. |

| Attribute | Type | Description |
|-------------------|--------|--|
| resource_inuse | object | In use resource within queue in a form of vcores, memory, and so on. |
| resource_reserved | object | Reserved resource within queue in form of vcores, memory, and so on. |
| active_state | String | Indicate either "ACTIVE" or "INACTIVE" state of the queue. An "INACTIVE" queue does not schedule any allocation. |

- Query **single** queues within scheduler engine, including leaf and all middle queues.

- URL
GET https://<SS_REST_SERVER>/ws/v1/sscheduler/queues/{queuename}
- Input
None.
- Output

JSON Response:

```
{
  "queue": {
    "name": "root.default",
    "eopen_state": "CLOSED",
    "eactive_state": "INACTIVE",
    "open_state": "CLOSED",
    "active_state": "INACTIVE",
    "leaf_queue" : yes,
    "number_pending_application": 100,
    "number_running_application": 10,
    "number_pending_request": 10,
    "number_running_container": 10,
    "number_reserved_container": 1,
    "resource_inuse" {
      "vcores": 10,
      "memory": 10240
    },
    "resource_request" {
      "vcores": 2,
      "memory": 2048
    },
    "resource_reserved" {
      "vcores": 1,
      "memory": 1024
    }
  },
  "configuration": {
    "description": "Production spark queue",
    "max_pending_application": 10000,
    "max_running_application": 1000,
    "allocation_order_policy": "FIFO",
    "default_resource_select": "label1",
    "max_master_share": 10%,
    "max_running_application_per_user": -1,
    "max_allocation_unit": {
      "vcores": 32,
      "memory": 128000
    }
  }
}
```



```

"user_acl": [
  {
    "user": "user1"
  },
  {
    "group": "group1"
  }
],
"admin_acl": [
  {
    "user": "user2"
  },
  {
    "group": "group2"
  }
]
}
}
}

```

- Exceptions
Queue not found.

Table 34-10 Parameters of single queues

| Attribute | Type | Description |
|----------------------------|---------|--|
| queue | object | A queue object. |
| name | String | Queue name. |
| description | String | Purpose of queue. |
| open_state | String | This is inner state (self state) of queue. Indicate either "OPEN" or "CLOSED" effective state of the queue. A "CLOSED" queue does not accept any new allocation request. |
| eopen_state | String | This is outer state of queue. An effective state is combination of queue own state and its ancestors. A "CLOSED" queue does not accept any new allocation request. |
| active_state | String | This is inner state (self state) of queue. Indicate either "ACTIVE" or "INACTIVE" state of the queue. An "INACTIVE" queue does not schedule any application. |
| eactive_state | String | This is outer state of queue. An effective state is combination of queue own state and its ancestors. An "INACTIVE" queue does not schedule any application. |
| leaf_queue | boolean | Indicate whether queue is leaf or middle. Yes means leaf queue. |
| number_pending_application | int | Number of pending application currently. In case of middle or parent queue, this is the aggregated number of all children queue. |

| Attribute | Type | Description |
|----------------------------------|--------|--|
| number_running_application | int | Number of running application currently. In case of middle or parent queue, this is the aggregated number of all children queue. |
| number_pending_request | int | Number of pending demand; sum of count from each outstanding demand. In case of middle/parent queue, this is the aggregated number of all children queues. |
| number_running_container | int | Number of running containers. In case of middle or parent queue, this is the aggregated number of all children queues. |
| number_reserved_container | int | Number of reserved containers. In case of middle or parent queue, this is the aggregated number of children queues. |
| resource_request | object | Pending resource requests within queue in a form of vcores, memory etc. |
| resource_inuse | object | In use resource within queue in a form of vcores and memory etc. |
| resource_reserved | object | Reserved resources within queue in a form of vcores and memory etc. |
| configuration | object | A queue configuration object. |
| max_pending_application | int | Max number of pending application. In case of middle or parent queue, this is the aggregated number or all children queue. |
| max_running_application | int | Max number of running application. In case of middle/parent queue, this is the aggregated number or all children queue. |
| allocation_order_policy | String | Allocation policy, can be FIFO, PRIORITY, or FAIR. |
| max_running_application_per_user | int | Maximum number of running application per user. |
| max_master_share | String | Percentage of steady share of this queue. |
| max_allocation_unit | object | Maximum allowed resource per container in vcores and memory format. |
| default_resource_select | String | Default resource selection expression. It is used when an application does not specify one during its submission. |
| user_acl | array | Array of user, who have been given "user" rights on this queue. |

| Attribute | Type | Description |
|-----------|--------|--|
| admin_acl | array | Array of user, who have been given "admin" rights on this queue. |
| group | String | User group name. |
| user | String | User name. |

- Query Resource Pool
 - Query *all* resource pools within scheduler engine.

- URL

GET https://<SS_REST_SERVER>/ws/v1/sscheduler/resourcepools/list

- Input

None.

- Output

JSON Response:

```
{
  "resourcepool_list": [
    {
      "name": "pool1",
      "description": "resource pool for crc",
      "number_member": 5,
      "members": [
        {
          "resource": "node1"
        },
        {
          "resource": "node2"
        },
        {
          "resource": "node3"
        },
        {
          "resource": "node4"
        },
        {
          "resource": "node5"
        }
      ],
      "available_resource": {
        "vcores": 60,
        "memory": 60000
      },
      "total_resource": {
        "vcores": 100,
        "memory": 128000
      },
      "configuration": {
        "resources": [
          {
            "resource": "node1"
          },
          {
            "resource": "node[2-5]"
          }
        ],
        "resource_select": "label1"
      }
    }
  ],
  {
```

```

"name": "pool2",
"description": "resource pool for erc",
"number_member": 4
"members": [
  {
    "resource": "node6"
  },
  {
    "resource": "node7"
  },
  {
    "resource": "node8"
  },
  {
    "resource": "node9"
  }
],
"available_resource": {
  "vcores": 56,
  "memory": 48000
},
"total_resource": {
  "vcores": 100,
  "memory": 128000
},
"configuration": {
  "resources": [
    {
      "resource": "node6"
    },
    {
      "resource": "node[7-9]"
    }
  ],
  "resource_select": "label1"
}
},
{
  "name": "default",
  "description": "system-generated",
  "number_member": 1,
  "members": [
    {
      "resource": "node0"
    }
  ],
  "available_resource": {
    "vcores": 8,
    "memory": 8192
  },
  "total_resource": {
    "vcores": 16,
    "memory": 12800
  }
}
]
}

```

Table 34-11 Parameters of all resource pool

| Attribute | Type | Description |
|-------------------|--------|---------------------------------|
| resourcepool_list | array | Array of resource pool objects. |
| name | String | Resource pool name. |

| Attribute | Type | Description |
|--------------------|--------|---|
| number_member | int | Number of members in resource pool. |
| description | string | Description of the resource pool. |
| members | array | Array of resource name, which is current members of the resource pool. |
| resource | String | Resource name. |
| available_resource | object | Current available resource in this pool. |
| vcores, memory | int | Numeric consumable resource attributes, available via this resource pool now. |
| total_resource | object | total resource in this pool. |
| vcores, memory | int | Numeric consumable resource attributes, total via this resource pool now. |
| configuration | object | Configuration object. |
| resources | array | Array of resource name pattern configured. |
| resource | String | Resource name pattern. |
| resource_select | String | Resource selection expression. |

- Query **single** resource pools within scheduler engine.
 - URL
GET https://<SS_REST_SERVER>/ws/v1/sscheduler/resourcepools/{resourcepoolname}

- Input
None.

- Output

JSON Response:

```
{
  "resourcepool": {
    "name": "pool1",
    "description": "resource pool for crc",
    "number_member": 5
    "members": [
      {
        "resource": "node1"
      },
      {
        "resource": "node2"
      },
      {
        "resource": "node3"
      },
      {
        "resource": "node4"
      }
    ]
  }
}
```

```

    "resource": "node5"
  }
],
"available_resource": {
  "vcores": 60,
  "memory": 60000
},
"total_resource": {
  "vcores": 100,
  "memory": 128000
},
"configuration": {
  "resources": [
    {
      "resource": "node6"
    },
    {
      "resource": "node[7-9]"
    }
  ],
  "resource_select": "label1"
}
}
}

```

- Exceptions
Resource pool not found.

Table 34-12 Parameters of single resource pool

| Attribute | Type | Description |
|--------------------|--------|---|
| resourcepool | object | Resource pool object. |
| name | String | Resource pool name. |
| description | String | Description of the resource pool. |
| number_member | int | Number of members in resource pool. |
| members | array | Array of resource name, which is current members of the resource pool. |
| resource | String | Resource name. |
| available_resource | object | Current available resource in this pool. |
| vcores, memory | int | Numeric consumable resource attributes, available via this resource pool now. |
| total_resource | object | total resource in this pool. |
| vcores, memory | int | Numeric consumable resource attributes, total via this resource pool now. |
| configuration | object | Configuration object. |
| resources | array | Array of resource name pattern configured. |
| resource | String | Resource name pattern. |

| Attribute | Type | Description |
|-----------------|--------|--------------------------------|
| resource_select | String | Resource selection expression. |

- Query **policiesxmlconf**

- URL

GET https://<SS_REST_SERVER>/ws/v1/sscheduler/policiesxmlconf/list

- Input

None.

- Output

```
<policies>
  <policlist>
    <resourcepool>default</resourcepool>
    <queues>
      <name>default</name>
      <fullname>root.default</fullname>
      <share>20.0</share>
      <reserve>memory 0,vcores 0 : 0.0%</reserve>
      <minimum>memory 0,vcores 0 : 20.0%</minimum>
      <maximum>memory 0,vcores 0 : 100.0%</maximum>
      <defaultuser>
        <maximum>100.0%</maximum>
        <weight>1.0</weight>
      </defaultuser>
    </queues>
  </policlist>
</policies>
```

35 YARN Development Guide (Normal Mode)

35.1 Overview

Intended Audience

This document is intended for development personnel who are experienced in Java development and want to develop YARN applications.

YARN Introduction

YARN is a distributed resource management system that is used to improve resource usage in the distributed cluster environment. Resources include memory, I/O, network, and disk resources. YARN is developed to address the shortage of the original MapReduce framework. At the beginning, MapReduce committers periodically modified existing codes. As codes increase and because the original MapReduce framework was designed improperly, modification on the original MapReduce framework becomes more difficult. Therefore, MapReduce committers decided to re-design the MapReduce framework to provide a next-generation MapReduce (MRv2/Yarn) framework that supports high scalability, availability, reliability, backward compatibility, and resource usage. The next-generation MapReduce (MRv2/Yarn) framework supports more computing frameworks in addition to the MapReduce framework.

Basic Concepts

- **ResourceManager (RM)**
ResourceManager is a global resource manager that manages and allocates resources in the system. ResourceManager consists of two components: Scheduler and Applications Manager.
- **ApplicationMaster (AM)**
Each application submitted by users includes an ApplicationMaster. The ApplicationMaster provides the following functions:
 - Negotiates with the ResourceManager Scheduler to obtain resources (represented by Containers).

- Allocates resource to internal tasks.
 - Communicates with NodeManager to start or stop tasks.
 - Monitors all tasks with the running status, and applies for resources again for tasks when tasks fail to run to restart the tasks.
- **NodeManager (NM)**

NodeManager is the resource and task manager of each node. On one hand, NodeManager periodically reports resource usage of the local node and the running status of each Container to ResourceManager. On the other hand, NodeManager receives and processes requests from ApplicationMaster for starting or stopping Containers.
 - **Container**

Container is a resource abstract in YARN. Container encapsulates multidimensional resources of a node, such as memory, CPU, disk, and network resources. When ApplicationMaster applies for resources from ResourceManager, ResourceManager returns resources in a Container to ApplicationMaster.

35.2 Interfaces

35.2.1 Command

You can use YARN commands to perform operations on YARN clusters, such as starting ResourceManager, submitting applications, killing applications, querying node status, and downloading container logs.

For details about the commands, see:

<http://hadoop.apache.org/docs/r3.1.1/hadoop-yarn/hadoop-yarn-site/YarnCommands.html>

Common Commands

YARN commands can be used by common users and administrators. Common users can run a few of YARN commands, such as **jar** and **logs**. Administrators have permissions to run most YARN commands.

Users can run the following command to query usage of YARN:

yarn --help

Usage: Go to any directory on the YARN client, execute source command to import the environment variables, and run the command. The command format is as follows:

yarn [--config *confdir*] *COMMAND*

Table 35-1 describes the commands.

NOTE

The version 8.1.0.1 is used as an example. Replace it with the actual version number.

Table 35-1 Common commands

Command	Description
resourcemanager	<p>Runs a ResourceManager.</p> <p>Notice: Before running commands, for example, the following two commands, on the server as user omm (the client must have the execute permission of user omm), export environment variables first.</p> <ul style="list-style-type: none"> • export YARN_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.1.0.1/1_10_ResourceManager/etc • export HADOOP_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.1.0.1/1_10_ResourceManager/etc
nodemanager	<p>Runs a NodeManager.</p> <p>Notice: Before running commands, for example, the following two commands, on the server as user omm (the client must have the execute permission of user omm), export environment variables first.</p> <ul style="list-style-type: none"> • export YARN_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.1.0.1/1_10_NodeManager/etc • export export HADOOP_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.1.0.1/1_10_NodeManager/etc
rmdadmin	Runs administrator's tool for dynamically updating information.
version	Displays the version.
jar <jar>	Runs a JAR file.
logs	Obtains container logs.
classpath	Displays the class path of the Hadoop JAR package and other library files.
daemonlog	Obtains or sets the service log level.
CLASSNAME	Runs a class named by <i>CLASSNAME</i> .
top	Runs the cluster usage monitor tool.
-Dmapreduce.job.hdfs-servers	<p>If OBS is connected, but the server still uses HDFS, you need to explicitly use this parameter in the command line to specify the HDFS address. The format is hdfs://{NAMESERVICE}. Replace <i>NAMESERVICE</i> with the HDFS NameService name.</p> <p>If the current HDFS has multiple NameService instances, you need to specify all NameService instances and separate them with commas (,), for example, hdfs://nameservice1,hdfs://nameservice2.</p>

Superior Scheduler Command

Superior Scheduler Engine provides a CLI that display detail information of Superior Scheduler Engine. For executing superior command we need to use the "<HADOOP_HOME>/bin/superior" script.

Here is the format of "superior" command.

```
<HADOOP_HOME>/bin/superior
```

```
Usage: superior [COMMAND | -help]
Where COMMAND is one of:
resourcepool      prints resource pool status
queue             prints queue status
application       prints application status
policy           prints policy status
```

Most commands print help when invoked without parameters.

- Superior **resourcepool** command :

This command displays resourcepool and associated policy related status and configuration information.

NOTE

Superior resourcepool command can be used only by admin user and user with yarn admin rights.

Usage output:

```
>superior resourcepool
```

```
Usage: resourcepool [-help]
        [-list]
        [-status <resourcepoolname>]
-help          prints resource pool usage
-list         prints all resource pool summary report
-status <resourcepoolname> prints status and configuration of specified
resource pool
```

- **resourcepool -list** prints resource pool summary in a table format. Here is an example:

```
> superior resourcepool -list
NAME      NUMBER_MEMBER  TOTAL_RESOURCE      AVAILABLE_RESOURCE
Pool1     4               vcores 30,memory 1000  vcores 21,memory 80
Pool2     100            vcores 100,memory 12800 vcores 30,memory 1000
default   2               vcores 64,memory 128   vcores 40,memory 28
```

- **resourcepool -status <resourcepoolname>** prints resource pool detail information in a list format. Here is an example:

```
> superior resourcepool -status default
NAME: default
DESCRIPTION: System generated resource pool
TOTAL_RESOURCE: vcores 64,memory 128
AVAILABLE_RESOURCE: vcores 40,memory 28
NUMBER_MEMBER: 2
MEMBERS: node1,node2
CONFIGURATION:
|-- RESOURCE_SELECT:
|_ RESOURCES:
```

- Superior **queue** command

This command displays hierarchy queue information.

Usage output:

```
>superior queue
Usage: queue [-help]
      [-list] [-e] [[-name <queue_name>] [-r|-c]]
      [-status <queue_name>]
-c          only work with -name <queue_name> option. If this
            option is used, command will print information of
            specified queue and its direct children.
-e          only work with -list or -list -name option. If
            this option is used, command will print effective
            state of specified queue and all of its
            descendants.
-help      prints queue sub command usage
-list      prints queue summary report. This option can work
            with -name <queue_name> and -r options.
-name <queue_name> print specified queue, this can work with -r
            option. By default, it will print queue's own
            information. When -r is defined, command will
            print all of its descendant queues. When -c is
            defined, it will print its direct children queues.
-r          only work with -name <queue_name> option. If this
            option is used, command will print information of
            specified queue and all of its descendants.
-status <queue_name> prints status of specified queue
```

- **queue -list** prints a table format of queue summary information. When command displays, command will display them based on queue hierarchy fashion. With SUBMIT ACL or ADMIN ACL rights for queue, user will be able to see queues. Here is an example:

```
> superior queue -list
NAME      STATE      NRUN_APP  NPEND_APP  NRUN_CONTAINER
NPEND_REQUEST RES_INUSE  RES_REQUEST
root      OPEN|ACTIVE  10      20      100      200      vcores 100,memory
1000     vcores 200,memory 2000
root.Q1   OPEN|ACTIVE  5        10      50      100      vcores 50,memory
500     vcores 100,memory 1000
root.Q1.Q11 OPEN|ACTIVE  5        10      50      100      vcores 50,memory
500     vcores 100,memory 1000
root.Q1.Q12 CLOSE|INACTIVE 0        0        0        0        vcores 0,memory
0        vcores 0,memory 0
root.Q2   OPEN|INACTIVE 5        10      50      100      vcores 50,memory
500     vcores 100,memory 1000
root.Q2.Q21 OPEN|ACTIVE  5        10      50      100      vcores 50,memory
500     vcores 100,memory 1000
```

- **queue -list -name root.Q1** will display root.Q1 only.

```
> superior queue -list -name root.Q1
NAME      STATE      NRUN_APP  NPEND_APP  NRUN_CONTAINER
NPEND_REQUEST RES_INUSE  RES_REQUEST
root.Q1   OPEN|ACTIVE  5        10      50      100      vcores 50,memory
500     vcores 100,memory 1000
```

- **queue -list -name root.Q1 -r** will print out root.Q1 and all of its descendents.

```
> superior queue -list -name root.Q1 -r
NAME      STATE      NRUN_APP  NPEND_APP  NRUN_CONTAINER
NPEND_REQUEST RES_INUSE  RES_REQUEST
root.Q1   OPEN|ACTIVE  5        10      50      100      vcores 50,memory
500     vcores 100,memory 1000
root.Q1.Q11 OPEN|ACTIVE  5        10      50      100      vcores 50,memory
500     vcores 100,memory 1000
root.Q1.Q12 CLOSE|INACTIVE 0        0        0        0        vcores 0,memory
0        vcores 0,memory 0
```

- **queue -list -name root -c** will print out root and all of its direct children

```
> superior queue -list -name root -c
NAME      STATE      NRUN_APP  NPEND_APP  NRUN_CONTAINER
NPEND_REQUEST RES_INUSE  RES_REQUEST
root      OPEN|ACTIVE  10      20      100      200      vcores
100,memory 1000 vcores 200,memory 2000
```

root.Q1	OPEN ACTIVE	5	10	50	100	vcores
50,memory 500	vcores 100,memory 1000					
root.Q2	OPEN INACTIVE	5	10	50	100	vcores
50,memory 500	vcores 100,memory 1000					

- **queue -status <queue_name>** will display detail queue status and configuration.

With SUBMIT ACL, user will be able to see details except ACLS of queue. User with ADMIN ACL rights for queue will be able to see queue details including ACL.

```
> superior queue -status root.Q1
NAME: root.Q1
OPEN_STATE:CLOSED
ACTIVE_STATE: INACTIVE
EOPEN_STATE: CLOSED
EACTIVE_STATE: INACTIVE
LEAF_QUEUE: Yes
NUMBER_PENDING_APPLICATION: 100
NUMBER_RUNNING_APPLICATION: 10
NUMBER_PENDING_REQUEST: 10
NUMBER_RUNNING_CONTAINER: 10
NUMBER_RESERVED_CONTAINER: 0
RESOURCE_REQUEST: vcores 3,memory 3072
RESOURCE_INUSE: vcores 2,memory 2048
RESOURCE_RESERVED: vcores 0,memory 0
CONFIGURATION:
|-- DESCRIPTION: Spark session queue
|-- MAX_PENDING_APPLICATION: 10000
|--MAX_RUNNING_APPLICATION: 1000
|--ALLOCATION_ORDER_POLICY: FIFO
|--DEFAULT_RESOURCE_SELECT: label1
|--MAX_MASTER_SHARE: 10%
|--MAX_RUNNING_APPLICATION_PER_USER : -1
|--MAX_ALLOCATION_UNIT: vcores 32,memory 12800
|--ACL_USERS: user1,user2
|--ACL_USERGROUPS: usergroup1,usergroup2
|-- ACL_ADMINS: user1
|--ACL_ADMININGROUPS: usergroup1
```

- Superior **application** command

This command displays application related information.

Usage output:

```
>superior application

Usage: application [-help]
                [-list]
                [-status <application_id>]
-help           prints application sub command usage
-list          prints all application summary report
-status <application_id> prints status of specified application
```

With the view access rights to application user can see application related information.

- **application -list** provides summary information of all application in a table format. Here is an example:

```
> superior application -list
ID          QUEUE          USER          NRUN_CONTAINER
NPEND_REQUEST  NRSV_CONTAINER  RES_INUSE
RES_REQUEST  RES_RESERVED
application_1482743319705_0005      root.SEQ.queueB  hbase  1
100          0          vcores 1,memory 1536      vcores 2000,memory
409600
application_1482743319705_0006      root.SEQ.queueB  hbase  0
1          0          vcores 0,memory 0          vcores 1,memory
1536          vcores 0,memory 0
```

- ***application -status <app_id>*** command displays detail information of specified application. Here is an example:

```
> superior application -status application_1443067302606_0609
ID: application_1443067302606_0609
QUEUE: root.Q1.Q11
USER: cchen
RESOURCE_REQUEST: vcores 3,memory 3072
RESOURCE_INUSE: vcores 2,memory 2048
RESOURCE_RESERVED:vcores 1, memory 1024
NUMBER_RUNNING_CONTAINER: 2
NUMBER_PENDING_REQUEST: 3
NUMBER_RESERVED_CONTAINER: 1
MASTER_CONTAINER_ID: application_1443067302606_0609_01
MASTER_CONTAINER_RESOURCE: node1.domain.com
BLACKLIST: node5,node8
DEMANDS:
|-- PRIORITY: 20
|-- MASTER: true
|-- CAPABILITY: vcores 2, memory 2048
|-- COUNT: 1
|-- RESERVED_RES : vcores 1, memory 1024
|-- RELAXLOCALITY: true
|-- LOCALITY: node1/1
|-- RESOURCESELECT: label1
|-- PENDINGREASON: "application limit reached"
|-- ID: application_1443067302606_0609_03
|-- RESOURCE: node1.domain.com
|-- RESERVED_RES: vcores 1, memory 1024
|
|--PRIORITY: 1
|-- MASTER: false
|-- CAPABILITY: vcores 1,memory 1024
|-- COUNT: 2
|-- RESERVED_RES: vcores 0, memory 0
|-- RELAXLOCLITY: true
|--LOCALITY: node1/1,node2/1,rackA/2
|-- RESOURCESELECT: label1
|-- PENDINGREASON: "no available resource"
CONTAINERS:
|-- ID: application_1443067302606_0609_01
|-- RESOURCE: node1.domain.com
|-- CAPABILITY: vcores 1,memory 1024
|
|-- ID: application_1443067302606_0609_02
|-- RESOURCE: node2.domain.com
|-- CAPABILITY: vcores 1,memory 1024
```

- Superior ***policy*** command

This command displays policy related information.

NOTE

Superior policy command can be used only by admin user and user with yarn admin rights.

Usage output:

```
>superior policy
Usage: policy [-help]
        [-list <resourcepoolname>] [-u] [-detail]
        [-status <resourcepoolname>]
-detail    only work with -list option to show a
            summary information of resource pool
            distribution on queues, including reserve,
            minimum and maximum
-help      prints policy sub command usage
-list <resourcepoolname> prints a summary information of resource
            pool distribution on queue
```

```
-status <resourcepoolname> prints pool distribution policy
configuration and status of specified
resource pool
-u only work with -list option to show a
summary information of resource pool
distribution on queues and also user
accounts
```

- ***policy -list <resourcepoolname>*** print out a summary of queue distribution information. Here is an example:

```
>superior policy -list default
NAME: default
TOTAL_RESOURCE: vcores 16,memory 16384
AVAILABLE_RESOURCE: vcores 16,memory 16384

NAMERES_INUSERES_REQUEST
root.defaultvcores 0,memory 0vcores 0,memory 0
root.productionvcores 0,memory 0vcores 0,memory 0
root.production.BU1vcores 0,memory 0vcores 0,memory 0
root.production.BU2 vcores 0,memory 0vcores 0,memory 0
```

- ***policy -list <resourcepoolname> -u*** prints out also user level summary information

```
> superior policy -list default -u
NAME: default
TOTAL_RESOURCE: vcores 16,memory 16384
AVAILABLE_RESOURCE: vcores 16,memory 16384

NAMERES_INUSERES_REQUEST
root.defaultvcores 0,memory 0vcores 0,memory 0
root.default.[_others_]vcores 0,memory 0vcores 0,memory 0
root.productionvcores 0,memory 0vcores 0,memory 0
root.production.BU1vcores 0,memory 0vcores 0,memory 0
root.production.BU1.[_others_]vcores 0,memory 0vcores 0,memory 0
root.production.BU2vcores 0,memory 0vcores 0,memory 0
root.production.BU2.[_others_]vcores 0,memory 0vcores 0,memory 0
```

- ***policy -status <resourcepoolname>*** print out policy detail of specified resource pool. Here is an example:

```
> superior policy -status pool1
NAME: pool1
TOTAL_RESOURCE: vcores 64,memory 128
AVAILABLE_RESOURCE: vcores 40,memory 28
QUEUES:
|-- NAME: root.Q1
|-- RESOURCE_USE: vcores 20, memory 1000
|-- RESOURCE_REQUEST: vcores 2,memory 100
|--RESERVE: vcores 10, memory 4096
|--MINIMUM: vcore 11, memory 4096
|--MAXIMUM: vcores 500, memory 100000
|--CONFIGURATION:
|-- SHARE: 50%
|-- RESERVE: vcores 10, memory 4096
|-- MINIMUM: vcores 11, memory 4096
|-- MAXIMUM: vcores 500, memory 100000
|-- QUEUES:
|-- NAME: root.Q1.Q11
|-- RESOURCE_USE: vcores 15, memory, 500
|-- RESOURCE_REQUEST: vcores 1, memory 50
|-- RESERVE: vcores 0, memory 0
|-- MINIMUM: vcores 0, memory 0
|-- MAXIMUM: vcores -1, memory -1
|-- USER_ACCOUNTS:
|-- NAME: user1
|-- RESOURCE_USE: vcores 1, memory 10
|-- RESOURCE_REQUEST: vcores 1, memory 50
|
|-- NAME: OTHERS
|--RESOURCE_USE: vcores 0, memory 0
|-- RESOURCE_REQUEST: vcores 0, memory 0
```

```

|-- CONFIGURATION:
|-- SHARE: 100%
|-- USER_POLICY:
|-- NAME: user1
|-- WEIGHT: 10
|
|-- NAME: OTHERS
|-- WEIGHT: 1
|-- MAXIMUM: vcores 10, memory 1000
    
```

35.2.2 Java API

For details about YARN application programming interfaces (APIs), see

<http://hadoop.apache.org/docs/r3.1.1/api/index.html>.

Common Interfaces

Common YARN Java classes are as follows:

- **ApplicationClientProtocol**

This class is used between the client and ResourceManager. The client submits applications to ResourceManager, queries the running status of applications, and kills applications using this protocol.

Table 35-2 Common interfaces of ApplicationClientProtocol

Interface	Description
forceKillApplication(KillApplicationRequest request)	The client requests ResourceManager to stop a submitted task through this interface.
getApplicationAttemptReport(GetApplicationAttemptReportRequest request)	The client obtains the report of specified ApplicationAttempt from ResourceManager through this interface.
getApplicationAttempts(GetApplicationAttemptsRequest request)	The client obtains the report of all ApplicationAttempts from ResourceManager through this interface.
getApplicationReport(GetApplicationReportRequest request)	The client obtains an application report from ResourceManager through this interface.
getApplications(GetApplicationsRequest request)	The client obtains information about applications that meet filtering conditions from ResourceManager through this interface.
getClusterMetrics(GetClusterMetricsRequest request)	The client obtains metrics of clusters from ResourceManager through this interface.
getClusterNodes(GetClusterNodesRequest request)	The client obtains information about all nodes in a cluster from ResourceManager through this interface.

Interface	Description
getContainerReport(GetContainerReportRequest request)	The client obtains the report of a Container from ResourceManager through this interface.
getContainers(GetContainersRequest request)	The client obtains the report of all Containers of an ApplicationAttempt from ResourceManager through this interface.
getDelegationToken(GetDelegationTokenRequest request)	The client obtains the delegation token through this interface. The delegation token is used by the container to access related services.
getNewApplication(GetNewApplicationRequest request)	The client obtains a new application ID through this interface for submitting a new application.
getQueueInfo(GetQueueInfoRequest request)	The client obtains queue information from ResourceManager through this interface.
getQueueUserAcls(GetQueueUserAclsInfoRequest request)	The client obtains queue access permission information about the current user from ResourceManager through this interface.
moveApplicationAcrossQueues(MoveApplicationAcrossQueuesRequest request)	This interface is used to move an application to a new queue.
submitApplication(SubmitApplicationRequest request)	The client submits a new application to ResourceManager through this interface.

- ApplicationMasterProtocol

This class is used between ApplicationMaster and ResourceManager. ApplicationMaster registers with ResourceManager, and applies for resources and obtains the running status of each task from ResourceManager using this protocol.

Table 35-3 Common interfaces of ApplicationMasterProtocol

Interface	Description
allocate(AllocateRequest request)	ApplicationMaster submits a resource allocation request through this interface.
finishApplicationMaster(FinishApplicationMasterRequest request)	ApplicationMaster notifies ResourceManager of running success or failure through this interface.
registerApplicationMaster(RegisterApplicationMasterRequest request)	ApplicationMaster registers with ResourceManager through this interface.

- ContainerManagementProtocol
This class is used between ApplicationMaster and NodeManager. ApplicationMaster requests NodeManager to start or terminate a Container or queries the Container running status using this protocol.

Table 35-4 Common interfaces of ContainerManagementProtocol

Interface	Description
getContainerStatuses(GetContainerStatusesRequest request)	ApplicationMaster obtains the current status of the Containers from NodeManager through this interface.
startContainers(StartContainersRequest request)	ApplicationMaster requests NodeManager to start Containers through this interface.
stopContainers(StopContainersRequest request)	ApplicationMaster requests NodeManager to stop a series of allocated Containers through this interface.

35.2.3 REST API

Function Description

You can use HTTP REST APIs to query more information about Yarn jobs. Currently, you can only query some resources or jobs using REST APIs provided by Yarn. For details of HTTP REST APIs, see the following official guidelines:

<http://hadoop.apache.org/docs/r3.1.1/hadoop-yarn/hadoop-yarn-site/WebServicesIntro.html>

Preparing Running Environment

1. Install a client on the node. For example, install a client in the **/opt/client** directory. See details in "Installing a Client."
2. Go to the **/opt/client** directory where the client is installed and run the following commands to initiate environment variables:

```
source bigdata_env
```

Procedure

1. Query the information about jobs that run on the Yarn.
 - Command:
`curl -k -i --negotiate -u : "http://10-120-85-2:8088/ws/v1/cluster/apps/"`

 NOTE

- **10-120-85-2**: host name of the active ResourceManager node.
You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Instance**, and view the **Host Name of ResourceManager(Active)**.
- **8088**: port number of the ResourceManager.
You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Configurations > All Configurations**, search for and obtain the value of **yarn.resourcemanager.webapp.port**.

- The running result is displayed as follows:

```
{
  "apps": {
    "app": [
      {
        "id": "application_1461743120947_0001",
        "user": "spark",
        "name": "Spark-JDBCServer",
        "queue": "default",
        "state": "RUNNING",
        "finalStatus": "UNDEFINED",
        "progress": 10,
        "trackingUI": "ApplicationMaster",
        "trackingUrl": "http://10-120-85-2:8088/proxy/application_1461743120947_0001/",
        "diagnostics": "AM is launched. ",
        "clusterId": 1461743120947,
        "applicationType": "SPARK",
        "applicationTags": "",
        "startedTime": 1461804906260,
        "finishedTime": 0,
        "elapsedTime": 6888848,
        "amContainerLogs": "https://10-120-85-2:8088/node/containerlogs/
container_e12_1461743120947_0001_01_000001/spark",
        "amHostHttpAddress": "10-120-85-2:8088",
        "allocatedMB": 1024,
        "allocatedVCores": 1,
        "runningContainers": 1,
        "memorySeconds": 7053309,
        "vcoreSeconds": 6887,
        "preemptedResourceMB": 0,
        "preemptedResourceVCores": 0,
        "numNonAMContainerPreempted": 0,
        "numAMContainerPreempted": 0,
        "resourceRequests": [
          {
            "capability": {
              "memory": 1024,
              "virtualCores": 1
            },
            "nodeLabelExpression": "",
            "numContainers": 0,
            "priority": {
              "priority": 0
            },
            "relaxLocality": true,
            "resourceName": "*"
          }
        ],
        "logAggregationStatus": "NOT_START",
        "amNodeLabelExpression": ""
      }
    ],
    {
      "id": "application_1461722876897_0002",
      "user": "admin",
      "name": "QuasiMonteCarlo",
      "queue": "default",
      "state": "FINISHED",
    }
  }
}
```

```

        "finalStatus": "SUCCEEDED",
        "progress": 100,
        "trackingUI": "History",
        "trackingUrl": "http://10-120-85-2:8088/proxy/application_1461722876897_0002/",
        "diagnostics": "Attempt recovered after RM restart",
        "clusterId": 1461743120947,
        "applicationType": "MAPREDUCE",
        "applicationTags": "",
        "startedTime": 1461741052993,
        "finishedTime": 1461741079483,
        "elapsedTime": 26490,
        "amContainerLogs": "http://10-120-85-2:8088/node/containerlogs/
container_e11_1461722876897_0002_01_000001/admin",
        "amHostHttpAddress": "10-120-85-2:8088",
        "allocatedMB": -1,
        "allocatedVCores": -1,
        "runningContainers": -1,
        "memorySeconds": 158664,
        "vcoreSeconds": 52,
        "preemptedResourceMB": 0,
        "preemptedResourceVCores": 0,
        "numNonAMContainerPreempted": 0,
        "numAMContainerPreempted": 0,
        "amNodeLabelExpression": ""
    }
}
}
}

```

- Result analysis:

On the interface, you can query the information of jobs that are running on the Yarn and obtain the common information that is displayed in [Table 1](#).

Table 35-5 Common information of jobs running on Yarn

Information	Description
user	Indicates the user who runs the job.
applicationType	Indicates the application types, such as MAPREDUCE or SPARK.
finalStatus	Indicates whether a job is executed successfully.
elapsedTime	Indicates the time to run a job.

2. Obtain the overall information of Yarn resources.

- Command:

```
curl -k -i --negotiate -u : "http://10-120-85-102:8088/ws/v1/cluster/metrics"
```

- The running result is displayed as follows:

```

{
  "clusterMetrics": {
    "appsSubmitted": 2,
    "appsCompleted": 1,
    "appsPending": 0,
    "appsRunning": 1,
    "appsFailed": 0,
    "appsKilled": 0,
    "reservedMB": 0,
    "availableMB": 23552,

```

```

    "allocatedMB": 1024,
    "reservedVirtualCores": 0,
    "availableVirtualCores": 23,
    "allocatedVirtualCores": 1,
    "containersAllocated": 1,
    "containersReserved": 0,
    "containersPending": 0,
    "totalMB": 24576,
    "totalVirtualCores": 24,
    "totalNodes": 3,
    "lostNodes": 0,
    "unhealthyNodes": 0,
    "decommissionedNodes": 0,
    "rebootedNodes": 0,
    "activeNodes": 3,
    "rmMainQueueSize": 0,
    "schedulerQueueSize": 0,
    "stateStoreQueueSize": 0
  }
}

```

– Result analysis:

On the interface, users can query the common information of jobs that are running in the cluster, as displayed in [Table 2](#).

Table 35-6 Common information of jobs running in the cluster

Information	Description
appsSubmitted	Indicates the number of jobs that have been submitted.
appsCompleted	Indicates the number of jobs that have been completed.
appsPending	Indicates the number of jobs that have been suspended.
appsRunning	Indicates the number of jobs that are running.
appsFailed	Indicates the number of jobs that have failed.
appsKilled	Indicates the number of jobs that have been killed.
totalMB	Indicates the total memory of Yarn resources.
totalVirtualCores	Indicates the total VCores of Yarn resources.

35.2.4 REST APIs of Superior Scheduler

Function Description

The REST/HTTP server is part of Superior Scheduler on YARN Resource Manager host and leverage existing YARN resource manager web service port. In the section below, we will denote this YARN *address:port* as *SS_REST_SERVER*.

Below uses HTTP as part of URL and only HTTP will be supported.

Superior Scheduler Interfaces

- **Query Application**

- Query **all** application within scheduler engine.

- **URL**

GET `http://<SS_REST_SERVER>/ws/v1/sscheduler/applications/list`

 **NOTE**

SS_REST_SERVER indicates *ResourceManager IP address:Port number*.

- ResourceManager IP address: You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Instance**, and view the service IP address of any ResourceManager.
- Port: HTTPS port number of the ResourceManager. You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Configurations > All Configurations**, search for and view the value of **yarn.resourcemanager.webapp.port**.

- **Input**

None.

- **Output**

JSON Response:

```
{
  "applicationlist": [
    {
      "id": "1020201_0123_12",
      "queue": "root.Q1.Q11",
      "user": "cchen",
      "resource_request": {
        "vcores" : 10,
        "memory" : 100
      },
      "resource_inuse": {
        "vcores" : 100,
        "memory" : 2000
      },
      "number_running_container": 100,
      "number_pending_request": 10
    },
    {
      "id": "1020201_0123_15",
      "queue": "root.Q2.Q21",
      "user": "Jason",
      "resource_request": {
        "vcores" : 4,
        "memory" : 100
      },
      "resource_inuse": {
        "vcores" : 20,
```

```

"memory" : 200
},
"resource_reserved": {
  "vcores" : 10,
  "memory" : 100
},
"number_running_container": 20,
"number_pending_container": 4,
"number_reserved_container":2
}
]
}

```

Table 35-7 Parameters of all application

Attribute	Type	Description
applicationlist	array	Array of application IDs.
queue	String	Name of queue application.
user	String	Name of user who submits application.
resource_request	object	Currently requested resource, including vcores, memory, and so on.
resource_inuse	object	Currently resource in use, including vcores, memory, and so on.
resource_reserved	object	Currently reserved resource, including vcores, memory, and so on.
number_running_container	int	Total number of running containers. This maps to superior engine decisions.
number_pending_request	int	Total number of pending requests, this is sum of all counts in all demands of allocation.
number_reserved_container	int	Total number of reserved containers, this maps to superior engine decisions.
id	String	Application ID.

- Query **single** application within scheduler engine.
 - URL
GET `http://<SS_REST_SERVER>/ws/v1/sscheduler/applications/{application_id}`

- Input
None.

- Output
JSON Response:

```

{
  "applicationlist": [
    {
      "id": "1020201_0123_12",

```

```
"queue": "root.Q1.Q11",
"user": "cchen",
"resource_request": {
  "vcores": 3,
  "memory": 3072
},
"resource_inuse": {
  "vcores": 100,
  "memory": 2048
},
"number_running_container": 2,
"number_pending_request": 3,
"number_reserved_container": 1
"master_container_id": 23402_3420842
"master_container_resource": node1.domain.com
"blacklist": [
  {
    "resource": "node5"
  },
  {
    "resource": "node8"
  }
],
"demand": [
  {
    "priority": 1,
    "ismaster": true,
    "capability": {
      "vcores": 2,
      "memory": 2048
    },
    "count": 1,
    "relaxlocality": true,
    "locality": [
      {
        "target": "node1",
        "count": 1,
        "strict": false
      }
    ],
    "resourceselect": "label1",
    "pending_reason": "application limit reached",
    "reserved_resource": {
      "vcores": 1,
      "memory": 1024
    },
    "reservations": [
      {
        "id": "23402_3420878",
        "resource": "node1.domain.com",
        "reservedAmount": {
          "vcores": 1,
          "memory": 1024
        }
      }
    ]
  },
  {
    "priority": 1,
    "ismaster": false,
    "capability": {
      "vcores": 1,
      "memory": 1024
    },
    "count": 2,
    "relaxlocality": true,
    "locality": [
      {
        "target": "node1",
        "count": 1,
        "strict": false
      }
    ]
  }
]
```



```

    },
    {
      "target": "node2",
      "count": 1,
      "strict": false
    },
    {
      "target": "rackA",
      "count": 2,
      "strict": false
    },
  ],
  "resourceselect": "label1",
  "pending_reason": "no available resource"
}
],
"containers": [
  {
    "id": "23402_3420842",
    "resource": "node1.domain.com",
    "capability": {
      "vcores": 1,
      "memory": 1024
    }
  },
  {
    "id": "23402_3420853",
    "resource": "node2.domain.com",
    "capability": {
      "vcores": 1,
      "memory": 1024
    }
  }
]
}
}

```

- Exceptions
Application not found.

Table 35-8 Parameters of single application

Attribute	Type	Description
application	object	Application object.
id	String	Application ID.
queue	String	Name of queue application.
user	String	Name of user who submits application.
resource_request	object	Currently requested resource, including vcores, memory, and so on.
resource_inuse	object	Currently resource in use, including vcores, memory, and so on.
resource_reserved	object	Currently reserved resource, including vcores, memory, and so on.
number_running_container	int	Total number of running containers. This maps to superior engine decisions.

Attribute	Type	Description
number_pending_request	int	Total number of pending requests, this is sum of all counts in all demands of allocation.
number_reserved_container	int	Total number of reserved containers, this maps to superior engine decisions.
master_container_id	String	The master container ID.
master_container_resource	String	The host name that master container running on.
demand	array	Array of demand objects.
priority	int	Priority of demand.
ismaster	boolean	Is the demand corresponding to "application master".
capability	object	Capability object.
vcores, memory, ..	int	Numeric consumable resource attributes, defining the allocation "unit" for this demand.
count	int	Number of unit required.
relaxlocality	boolean	Locality requirement is preference, and not mandatory if cannot be satisfied.
locality	object	Locality object.
target	string	Locality target's name (that is, node1, rack1..).
count	int	Number of resource "unit" required with this locality requirement.
strict	boolean	If this particular locality is mandatory or not.
resourceselect	String	Resource selection expression for the demand.
pending_reason	String	Reason why this demand is outstanding.
resource_reserved	object	Currently reserved resource for this demand, including vcores, memory, and so on.
reservations	array	Array of reserved container objects.
reservations:id	String	Reserved container ID.
reservations:resource	String	Where the container is allocated.

Attribute	Type	Description
reservations:reserveAmount	object	The reserved amount of this reservation.
containers	array	Array of allocated container objects.
containers:id	String	Container ID.
containers:resource	String	Where the container is allocated.
containers:capability	object	Capability object.
containers:vcores, memory...	int	Numeric consumable resource attributes allocated to this container.

- Query Queue
 - Query **all** queues within scheduler engine, including leaf and all middle queues.

- URL
GET http://<SS_REST_SERVER>/ws/v1/sscheduler/queues/list

- Input
None.

- Output

JSON Response:

```
{
  "queuelist": [
    {
      "name": "root.default",
      "eopen_state": "OPEN",
      "eactive_state": "ACTIVE",
      "open_state": "OPEN",
      "active_state": "ACTIVE",
      "number_pending_application": 2,
      "number_running_application": 10,
      "number_pending_request": 2,
      "number_running_container": 10,
      "number_reserved_container": 1,
      "resource_inuse" {
        "vcores": 10,
        "memory": 10240
      },
      "resource_request" {
        "vcores": 2,
        "memory": 2048
      },
      "resource_reserved" {
        "vcores": 1,
        "memory": 1024
      }
    },
    {
      "name": "root.dev",
      "eopen_state": "OPEN",
      "eactive_state": "INACTIVE",

```

```

"open_state": "OPEN",
"active_state": "INACTIVE",
"number_pending_application": 2,
"number_running_application": 10,
"number_pending_request": 2,
"number_running_container": 10,
"number_reserved_container": 0,
"resource_inuse" {
  "vcores": 10,
  "memory": 10240
},
"resource_request" {
  "vcores": 2,
  "memory": 2048
},
"resource_reserved" {
  "vcores": 0,
  "memory": 0
}
},
{
  "name": "root.qa",
  "eopen_state": "CLOSED",
  "eactive_state": "ACTIVE",
  "open_state": "CLOSED",
  "active_state": "ACTIVE",
  "number_pending_application": 2,
  "number_running_application": 10,
  "number_pending_request": 2,
  "number_running_container": 10,
  "number_reserved_container": 0,
  "resource_inuse" {
    "vcores": 10,
    "memory": 10240
  },
  "resource_request" {
    "vcores": 2,
    "memory": 2048
  },
  "resource_reserved" {
    "vcores": 1,
    "memory": 1024
  }
},
]
}

```

Table 35-9 Parameters of all queues

Attribute	Type	Description
queuelist	array	Array of queue names.
name	String	Queue name.
open_state	String	This is inner state (self state) of queue. Indicate either "OPEN" or "CLOSED" effective state of the queue. A "CLOSED" queue does not accept any new allocation request.

Attribute	Type	Description
eopen_state	String	This is outer state of queue. An effective state is combination of queue own state and its ancestors. A "CLOSED" queue does not accept any new allocation request.
active_state	String	This is inner state (self state) of queue. Indicate either "ACTIVE" or "INACTIVE" state of the queue. A "INACTIVE" queue does not schedule any application.
eactive_state	String	This is outer state of queue. An effective state is combination of queue own state and its ancestors. A "INACTIVE" queue does not schedule any application.
number_pending_application	int	Total number of pending applications.
number_running_application	int	Total number of running applications.
number_pending_request	int	Total number of pending request.
number_running_container	int	Total number of running containers.
numbert_reserved_container	int	Total number of reserved containers.
resource_request	object	Pending resource requests within queue in a form of vcores, memory, and so on.
resource_inuse	object	In use resource within queue in a form of vcores, memory, and so on.
resource_reserved	object	Reserved resource within queue in form of vcores, memory, and so on.
active_state	String	Indicate either "ACTIVE" or "INACTIVE" state of the queue. A "INACTIVE" queue does not schedule any allocation.

- Query **single** queues within scheduler engine, including leaf and all middle queues.
 - URL
GET `http://<SS_REST_SERVER>/ws/v1/sscheduler/queues/{queuename}`
 - Input
None.

- **Output**

JSON Response:

```
{
  "queue": {
    "name": "root.default",
    "eopen_state": "CLOSED",
    "eactive_state": "INACTIVE",
    "open_state": "CLOSED",
    "active_state": "INACTIVE",
    "leaf_queue" : yes,
    "number_pending_application": 100,
    "number_running_application": 10,
    "number_pending_request": 10,
    "number_running_container": 10,
    "number_reserved_container": 1,
    "resource_inuse" {
      "vcores": 10,
      "memory": 10240
    },
    "resource_request" {
      "vcores": 2,
      "memory": 2048
    },
    "resource_reserved" {
      "vcores": 1,
      "memory": 1024
    }
  }
  "configuration": {
    "description": "Production spark queue",
    "max_pending_application": 10000,
    "max_running_application": 1000,
    "allocation_order_policy": "FIFO",
    "default_resource_select": "label1",
    "max_master_share": 10%,
    "max_running_application_per_user": -1,
    "max_allocation_unit": {
      "vcores": 32,
      "memory": 128000
    },
    "user_acl": [
      {
        "user": "user1"
      },
      {
        "group": "group1"
      }
    ],
    "admin_acl": [
      {
        "user": "user2"
      },
      {
        "group": "group2"
      }
    ]
  }
}
```

- **Exceptions**

Queue not found.

Table 35-10 Parameters of single queues

Attribute	Type	Description
queue	object	A queue object.
name	String	Queue name.
description	String	Purpose of queue.
open_state	String	This is inner state (self state) of queue. Indicate either "OPEN" or "CLOSED" effective state of the queue. A "CLOSED" queue does not accept any new allocation request.
eopen_state	String	This is outer state of queue. An effective state is combination of queue own state and its ancestors. A "CLOSED" queue does not accept any new allocation request.
active_state	String	This is inner state (self state) of queue. Indicate either "ACTIVE" or "INACTIVE" state of the queue. A "INACTIVE" queue does not schedule any application.
eactive_state	String	This is outer state of queue. An effective state is combination of queue own state and its ancestors. A "INACTIVE" queue does not schedule any application.
leaf_queue	boolean	Indicate whether queue is leaf or middle. Yes means leaf queue.
number_pending_application	int	Number of pending application currently. In case of middle or parent queue, this is the aggregated number of all children queue.
number_running_application	int	Number of running application currently. In case of middle or parent queue, this is the aggregated number of all children queue.
number_pending_request	int	Number of pending demand; sum of count from each outstanding demand. In case of middle/parent queue, this is the aggregated number of all children queues.
number_running_container	int	Number of running containers. In case of middle or parent queue, this is the aggregated number of all children queues.
number_reserved_container	int	Number of reserved containers. In case of middle or parent queue, this is the aggregated number of children queues.
resource_request	object	Pending resource requests within queue in a form of vcores, memory etc.

Attribute	Type	Description
resource_inuse	object	In use resource within queue in a form of vcores and memory etc.
resource_reserved	object	Reserved resources within queue in a form of vcores and memory etc.
configuration	object	A queue configuration object.
max_pending_application	int	Max number of pending application. In case of middle or parent queue, this is the aggregated number or all children queue.
max_running_application	int	Max number of running application. In case of middle/parent queue, this is the aggregated number or all children queue.
allocation_order_policy	String	Allocation policy, can be FIFO, PRIORITY, or FAIR.
max_running_application_per_user	int	Maximum number of running application per user.
max_master_share	string	Percentage of steady share of this queue.
max_allocation_unit	object	Maximum allowed resource per container in vcores and memory format.
default_resource_select	String	Default resource selection expression. It is used when an application does not specify one during its submission.
user_acl	array	Array of user, who have been given "user" rights on this queue.
admin_acl	array	Array of user, who have been given "admin" rights on this queue.
group	String	User group name.
user	String	User name.

- Query Resource Pool
 - Query *all* resource pools within scheduler engine.
 - URL
GET http://<SS_REST_SERVER>/ws/v1/sscheduler/resourcepools/list
 - Input
None.
 - Output
JSON Response:
{


```
"resourcepool_list": [  
  {  
    "name": "pool1",  
    "description": "resource pool for crc",  
    "number_member": 5,  
    "members": [  
      {  
        "resource": "node1"  
      },  
      {  
        "resource": "node2"  
      },  
      {  
        "resource": "node3"  
      },  
      {  
        "resource": "node4"  
      },  
      {  
        "resource": "node5"  
      }  
    ],  
    "available_resource": {  
      "vcores": 60,  
      "memory": 60000  
    },  
    "total_resource": {  
      "vcores": 100,  
      "memory": 128000  
    },  
    "configuration": {  
      "resources": [  
        {  
          "resource": "node1"  
        },  
        {  
          "resource": "node[2-5]"  
        }  
      ],  
      "resource_select": "label1"  
    }  
  },  
  {  
    "name": "pool2",  
    "description": "resource pool for erc",  
    "number_member": 4  
    "members": [  
      {  
        "resource": "node6"  
      },  
      {  
        "resource": "node7"  
      },  
      {  
        "resource": "node8"  
      },  
      {  
        "resource": "node9"  
      }  
    ],  
    "available_resource": {  
      "vcores": 56,  
      "memory": 48000  
    },  
    "total_resource": {  
      "vcores": 100,  
      "memory": 128000  
    },  
    "configuration": {
```

```

"resources": [
  {
    "resource": "node6"
  },
  {
    "resource": "node[7-9]"
  }
],
"resource_select": "label1"
}
},
{
  "name": "default",
  "description": "system-generated",
  "number_member": 1,
  "members": [
    {
      "resource": "node0"
    }
  ],
  "available_resource": {
    "vcores": 8,
    "memory": 8192
  },
  "total_resource": {
    "vcores": 16,
    "memory": 12800
  }
}
]
}

```

Table 35-11 Parameters of all resource pool

Attribute	Type	Description
resourcepool_list	array	Array of resource pool objects.
name	String	Resource pool name.
number_member	int	Number of members in resource pool.
description	String	Description of the resource pool.
members	array	Arrays of resource name, which are current members of the resource pool.
resource	String	Resource name.
available_resource	object	Current available resource in this pool.
vcores, memory, ..	int	Numeric consumable resource attributes, available via this resource pool now.
total_resource	object	total resource in this pool.
vcores, memory, ..	int	Numeric consumable resource attributes, total via this resource pool now.
configuration	object	Configuration object.

Attribute	Type	Description
resources	array	Array of resource name pattern configured.
resource	String	Resource name pattern.
resource_select	String	Resource select expression.

- Query **single** resource pools within scheduler engine.
 - URL
GET `http://<SS_REST_SERVER>/ws/v1/sscheduler/resourcepools/{resourcepoolname}`

- Input
None.

- Output

JSON Response:

```
{
  "resourcepool": {
    "name": "pool1",
    "description": "resource pool for crc",
    "number_member": 5
  },
  "members": [
    {
      "resource": "node1"
    },
    {
      "resource": "node2"
    },
    {
      "resource": "node3"
    },
    {
      "resource": "node4"
    },
    {
      "resource": "node5"
    }
  ],
  "available_resource": {
    "vcores": 60,
    "memory": 60000
  },
  "total_resource": {
    "vcores": 100,
    "memory": 128000
  },
  "configuration": {
    "resources": [
      {
        "resource": "node6"
      },
      {
        "resource": "node[7-9]"
      }
    ],
    "resource_select": "label1"
  }
}
```

- Exceptions
 - Resource pool not found.

Table 35-12 Parameters of single resource pool

Attribute	Type	Description
resourcepool	object	Resource pool object.
name	String	Resource pool name.
description	String	Description of the resource pool.
number_member	int	Number of members in resource pool.
members	array	Arrays of resource name, which are current members of the resource pool.
resource	String	Resource name.
available_resource	object	Current available resource in this pool.
vcores, memory, ..	int	Numeric consumable resource attributes, available via this resource pool now.
total_resource	object	total resource in this pool.
vcores, memory, ..	int	Numeric consumable resource attributes, total via this resource pool now.
configuration	object	Configuration object.
resources	array	Array of resource name pattern configured.
resource	String	Resource name pattern.
resource_select	String	Resource select expression.

- Query **policiesxmlconf**
 - URL
GET http://<SS_REST_SERVER>/ws/v1/sscheduler/policiesxmlconf/list
 - Input
None.
 - Output

```
<policies>
  <polliclist>
    <resourcepool>default</resourcepool>
    <queues>
      <name>default</name>
      <fullname>root.default</fullname>
      <share>20.0</share>
      <reserve>memory 0,vcores 0 : 0.0%</reserve>
      <minimum>memory 0,vcores 0 : 20.0%</minimum>
      <maximum>memory 0,vcores 0 : 100.0%</maximum>
      <defaultuser>
```

```
<maximum>100.0%</maximum>  
<weight>1.0</weight>  
</defaultuser>  
</queues>  
</policlist>  
</policies>
```

36 Development Specifications

36.1 Development Environment Construction

36.1.1 Rules

The development environment should be constructed strictly following the development guide. The required components should be downloaded from the client instead of open-source websites.

Based on Hadoop open-source components, MRS enhances its functionality and security. Huawei does not update the versions of these components because component dependencies are complicated. Components sharing the same name with those downloaded from the open-source website cannot run on MRS.

The time difference between the client and the cluster must be confirmed.

Ensure that the time difference between the client and the MRS cluster is less than 5 minutes.

The IBM JDK environment must be prepared.

If the application is developed using IBM JDK and the keytab file of the cluster needs to be used for authentication, check whether the IBM JDK version meets the requirements. Otherwise, exceptions may occur.

36.2 Security Authentication

36.2.1 Rules

Only one account is used in each process and a process needs explicit authentication only once.

This rule must be met in the following scenarios (pay attention to this in the design phase):

1. A process can access multiple clusters at the same time (Each cluster has independent KrbServer and LdapServer.)
2. All applications running in a container (for example, Tomcat) belong to the same process.

IBM JDK version requirements

The IBM JDK version must be 1.7.0 or later:

```
java version "1.7.0"  
Java(TM) SE Runtime Environment (build pxa6470sr9fp30-20160112_01(SR9 FP30))  
IBM J9 VM (build 2.6, JRE 1.7.0 Linux amd64-64 Compressed References 20151222_283043 (JIT enabled,  
AOT enabled)  
J9VM - R26_Java726_SR9_20151222_1626_B283043  
JIT - tr.r11_20151209_107111.01  
GC - R26_Java726_SR9_20151222_1626_B283043_CMPRSS  
J9CL - 20151222_283043)  
JCL - 20160112_01 based on Oracle jdk7u95-b13
```

36.2.2 Suggestions

Account Management Principles

1. Service application should apply for new accounts instead of using original system accounts.
2. The new accounts should meet the principle of least privilege.

36.3 ClickHouse

36.3.1 Rules

Ensure That the Time on the Client Is the Same as That on the Server If the Cluster Is Installed in the Security Mode

If the cluster is of the security edition and Kerberos authentication is required, the time on the server must be the same as that on the client. Pay attention to the time difference conversion between time zones. If the time is inconsistent, the client authentication fails and subsequent service processes cannot be executed.

Use partition fields and index fields of data tables properly

The MergeTree engine organizes and stores data in partition directories. During data query, partitions can be used to effectively skip useless data files and reduce data reading.

The MergeTree engine sorts data based on the index field and generates sparse indexes based on the **index_granularity** configuration. Data can be quickly filtered based on index fields, reducing data reading and improving query performance.

Insert a large volume of data at a low frequency

Each time data is inserted in ClickHouse, one or more part files are generated. If there are too many data parts, the pressure on merging increases and an exception may occur, affecting data insertion. You are advised to insert 100,000 rows at a time and ensure the frequency is no more than once per second.

Do not use the character type to store data of the time, date, or numeric type

Especially when the time, date, or numeric field needs to be calculated or compared.

The number of records in a single table (distributed table) cannot exceed trillions, and the number of records in a single table (local table) cannot exceed ten billions

The performance of querying trillions of tables is poor, and the cluster maintenance is difficult.

Data lifecycle management must be considered during table design

The disk space is limited, and data lifecycle management needs to be considered. The MergeTree engine supports column fields and table-level TTL when creating tables. When the values in a column field expire, ClickHouse replaces them with the default values of the data type. If all values of a column in a partition have expired, ClickHouse deletes the column files in the partition directory from the file system. When the data in a table expires, the ClickHouse deletes all the corresponding rows.

The external component ensures the idempotence of imported data

ClickHouse does not support transactions for data write. Use the external import module to control data idempotence. For example, if data of a batch fails to be imported, drop the corresponding partition data. After the fault is rectified, import the partition data again.

When a local ClickHouse table is created, the partition by keyword must be carried. Otherwise, the table cannot be migrated on the ClickHouse data migration page of Manager

The ClickHouse data migration page depends on the partition field of the table during table data migration. If partition by is not used to create partitions when the table is created, the table cannot be migrated on the ClickHouse data migration page of Manager.

Place a small table on the right for join query

When two tables are joined, the data in the right table is loaded to the memory, and then the data in the left table is traversed based on the data in the right table for matching. Placing the small table on the right reduces the number of match queries. According to the usage, the performance of joining a large table to a small table is improved by several orders of magnitude compared with that of joining a small table to a large table.

36.3.2 Suggestions

Properly configure the maximum number of concurrent operations

ClickHouse has a high processing speed because it uses the parallel processing mechanism. Even if a query is performed, half of the CPU of the server is used by default. Therefore, the ClickHouse does not support high-concurrency query scenarios. By default, the maximum number of concurrent connections is 100. You can adjust the number of concurrent connections based on the site requirements. In practice, the number of concurrent connections is 150. It is recommended that the number of concurrent connections be less than or equal to 200.

Deploy the load balancing component. The query is performed based on the load balancing component to prevent the performance from being affected due to heavy single-point query pressure

ClickHouse can connect to any node in the cluster for query. If the query is performed on one node, the node may be overloaded and the reliability is low. You are advised to use ClickHouseBalancer or other load balancing services to balance the query load and improve reliability.

Properly set the partition key, ensure that the number of partitions is less than 1000, and use the integer type for the partition field

1. You are advised to use `toYYYYMMDD` (table field `pt_d`) as the partition key. The table field `pt_d` is of the date type.
2. If hourly partitioning is required in the service scenario, use `toYYYYMMDD` (table field `pt_d`) and `toYYYYMMDD` (table field `pt_h`) as the joint partitioning key. `toYYYYMMDD` (table field `pt_h`) is an integer number of hours.
3. If data needs to be stored for many years, you are advised to create partitions by month, for example, `toYYYYMM` (table field `pt_d`).
4. Properly control the number of parts based on factors such as the data partition granularity, volume of data submitted in each batch, and data storage period.

During query, the most frequently used and most filtered fields are used as the primary keys. The fields are sorted in descending order of access frequency and dimension cardinality

Data is sorted and stored based on primary keys. When querying data, you can quickly filter data based on primary keys. Setting primary keys properly during

table creation can greatly reduce the amount of data to be read and improve query performance. For example, if the service ID needs to be specified for all analysis, the service ID field can be used as the first field of the primary key.

Properly set the sparse index granularity based on service scenarios

The primary key index of ClickHouse is stored by using a sparse index. The default sampling granularity of the sparse index is 8192 rows, that is, one record is selected from every 8192 rows in the index file.

Suggestions:

1. The smaller the index granularity is, the more effective the query in a small range is. This avoids the waste of query resources.
2. The larger the index granularity is, the smaller the index file is, and the faster the index file is processed.
3. If the table index granularity exceeds 1 billion, set this parameter to **16384**. Otherwise, set this parameter to **8192** or a smaller value.

Local Table Creation Reference

Reference:

```
CREATE TABLE mybase_local.mytable
(
  `did` Int32,
  `app_id` Int32,
  `region` Int32,
  `pt_d` Date
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/mybase_local/mytable', '{replica}')
PARTITION BY toYYYYMMDD(pt_d)
ORDER BY (app_id, region)
SETTINGS index_granularity = 8192, use_minimalistic_part_header_in_zookeeper = 1;
```

Instructions:

1. Select a table engine:
ReplicatedMergeTree: MergeTree engine that supports the replica feature. It is the most commonly used engine.
2. Table information registration path on ZooKeeper, which is used to distinguish different configurations in the cluster:
/clickhouse/tables/{shard}/{databaseName}/{tableName}: {shard}
indicates the shard name, **{databaseName}** indicates the database name, and **{tableName}** indicates the replicated table name.
3. **order by** primary key field:
The most frequently used and most filterable field is used as the primary key. The dimensions are sorted in ascending order of access frequency and dimension cardinality. It is recommended that the number of sorting fields be less than or equal to 4. Otherwise, the merge pressure is high. The sorting field cannot be null. If the sorting field is null, data conversion is required.
4. **partition by** field
The partition key cannot be null. If the field contains a null value, data conversion is required.

5. Table-level parameter configuration:
index_granularity: sparse index granularity. The default value is **8192**.
use_minimalistic_part_header_in_zookeeper: whether to enable the optimized storage mode of the new version for data storage in the ZooKeeper.
6. For details about how to create a table, visit <https://clickhouse.tech/docs/en/engines/table-engines/mergetree-family/mergetree/>.

Distributed Table Creation Reference

Reference:

```
CREATE TABLE mybase.mytable AS mybase_local.mytable  
ENGINE = Distributed(cluster_3shards_2replicas, mybase_local, mytable, rand());
```

Instructions:

1. Name of the distributed table: **mybase.mytable**.
2. Name of the local table: **mybase_local.mytable**.
3. Use **AS** to associate the distributed table with the local table to ensure that the field definitions of the distributed table are the same as those of the local table.
4. Parameter description of the distributed table engine:
cluster_3shards_2replicas: name of a logical cluster.
mybase_local: name of the database where the local table is located.
mytable: local table name.
rand(): (optional) sharding key, which can be the raw data (such as did) of a column in the table or the result of a function call, such as rand(). Note that data must be evenly distributed in this key. Another common operation is to use the hash value of a column with a large difference, for example, **intHash64(user_id)**.

Select the minimum type that meets the requirements based on the fields in the service scenario table

Numerical type, such as UInt8/UInt16/UInt32/UInt64, Int8/Int16/Int32/Int64, Float32/Float64. The performance varies according to the length.

Perform data analysis based on large and wide tables. Do not join large tables. Convert distributed join queries into join queries of local tables to improve performance

The performance of ClickHouse distributed join is poor. You are advised to aggregate data into a wide table on the model side and then import the table to ClickHouse. Queries in distributed join mode are converted to join queries on local tables. This eliminates the transmission of a large volume of data between nodes and reduces the volume of data involved in the calculation of local tables. The service layer summarizes data based on the local join results of all shards. The performance is improved remarkably.

Properly set the part size

The `min_bytes_to_rebalance_partition_over_jbod` parameter indicates the minimum size of the part involved in automatic balancing and distribution among disks in a JBOD array. The value must be appropriately set.

If the value is smaller than `max_bytes_to_merge_at_max_space_in_pool/1024`, the ClickHouse server process fails to be started and unnecessary parts move between disks.

If the value of `min_bytes_to_rebalance_partition_over_jbod` is greater than that of `max_data_part_size_bytes` (maximum size of parts that can be stored on disks in one array), no part can meet the condition for automatic balancing.

36.4 Flink

36.4.1 Applicable Scenarios

Flink is a unified computing framework that supports both batch processing and stream processing. It provides a stream data processing engine that supports data distribution and parallel computing. Flink features stream processing and is a top open-source stream processing engine in the industry.

Flink provides high-concurrency pipeline data processing, millisecond-level latency, and high reliability, making it suitable for low-latency data processing.

36.4.2 Rules

Delete Residual Directories If a Flink Task Stops Unexpectedly

After a FlinkServer instance is installed, the residual Flink directories are automatically deleted.

By default, only residual directories in the `/flink_base` directory of ZooKeeper and those in the `/flink/recovery` directory of HDFS are deleted.

36.4.3 Suggestions

FlinkServer Usage

You are advised to submit Flink jobs using FlinkServer. FlinkServer supports the submission of Flink SQL jobs and Flink Jar jobs. .

36.5 GraphBase

36.5.1 Rules

The token validity period in REST API authentication is 20 minutes by default, and can be customized

The token validity period in REST API authentication is 20 minutes. If the token is not used to interact with the server within 20 minutes, the token becomes invalid. To ensure the validity of the token, the invoker must interact with the server within 20 minutes using the token or obtain a new token after the token expires.

The token period can be customized in the GraphBase configuration interface to search for "graphserver.session.timeout" to set custom values.

Proper Index Design

A proper index mechanism has a great impact on data query, especially in full graph query scenarios. The index design must be designed based on service query requirements. If no valid index is designed, all data in the entire graph may be forcibly scanned during data query, resulting in low query performance and even request timeout.

Typical application scenarios of indexes are as follows:

- Query an entity with a same property value (string or numeric type) as the specified one.
For a large graph, if the service requires a query for entities that have the same property value with a specified property value, create indexes for the property (composite or mixed index) first. Otherwise, you need to perform a full graph query for the entities with the same property value as specified by the service in GraphBase, and perform full graph scan, which may cause query timeout.
- Query an entity by comparing property values (numeric type) with a specified one.
In the case of a large graph, if the service requires the query of entities by comparing property values with a specified one, you need to create mixed indexes for the property first. You can use this property index to efficiently lock the database entity objects that meet the query conditions.
- Full-text Search by Condition
- Currently, only mixed indexes support full-text search in GraphBase. For queries similar to the following one, create mixed indexes:
Perform full graph query to query the entity with a particular area or qualifier (such as Xi'an or BEIDAJIE) contained in the value of the address property.

Supplementary Indexes

If no index is created for a specified property at first but new indexes need to be created for the property after the property data is saved, the newly added indexes are supplementary indexes. This process involves recreating indexes.

After an index is created, index data is not automatically updated for the data imported previously. In this case, the user needs to invoke the interface for recreating an index to initiate an index data update.

Using the API for Deleting a Graph Cautiously

The interface for deleting a graph not only deletes the definition of the graph, but deletes all the data in the graph. Before invoking this API, ensure that the data is no longer used.

User Rights Description

Users who log in to GraphBase must have the permissions of the graphbaseadmin, graphbaseoperator, or graphbasedeveloper user group.

Client Currency

The load sharing between multiple GraphServer in GraphBase is based on the HTTP session mode. Multiple requests of the same HTTP session are sent to the same GraphServer for processing. Therefore, in the high concurrency scenario, the client code should use multiple HTTP clients to share loading using multiple GraphServers and to increase the number of concurrent requests.

has Usage Specifications in Gremlin Graph Query

1. Query the information about a vertex with a specified vertex label.

```
gremlin> g.V().hasLabel('person')
```

This query requires full graph query for vertices. If the number of vertices meeting this search condition is large or the graph is large, the query times out.

It is recommended that the output be limited.

```
gremlin> g.V().hasLabel('person').limit(10)
```

2. Perform full graph query for vertices using specified property conditions.

```
gremlin> g.V().has('key', 'value')
```

This query requires full graph query for vertices. If a large graph is accessed and no index is created for the property **key**, the query times out.

To sum up, exercise caution when performing full graph query.

36.5.2 Recommendations

Do Not Use REST APIs to Import a Large Amount of Data

If a large amount of data needs to be imported to GraphBase, use the batch import tool or real-time import tool based on site requirements. The REST API is designed based on HTTP requests and is not suitable for importing a large amount of data.

REST APIs for Periodical Data Import In Batches

If a large amount of data needs to be periodically imported to GraphBase in batches, you can periodically invoke the specified REST API to initiate a batch import task. The timer is provided by invoker.

36.6 HBase

36.6.1 Application Scenarios

Hadoop database (HBase) is a reliable, high-performance, column-oriented and scalable distributed storage system. Different from traditional relational databases, HBase is suitable for massive data process.

The HBase is applicable to the following scenarios:

- Massive data processing (higher than the TB or PB level).
- High-throughput demanding scenarios
- Scenarios that require efficient random read of massive data.
- Good-scalability demanding scenarios
- Concurrent processing of structured and unstructured data.
- Scenarios that do not require the Atomicity, Consistency, Isolation, Durability (ACID) feature provided by traditional relational databases.

HBase tables have the following features:

- Large: Each table contains a hundred million rows and one million columns.
- Column-oriented: Storage and rights control is implemented based on columns (families), and columns (families) are independently retrieved.
- Sparse: Null columns do not occupy storage space.

36.6.2 Rules

Create a Configuration instance

Call the create() method of HBaseConfiguration to instantiate this class. Otherwise, the HBase configurations cannot be successfully loaded.

Correct:

```
//This part is declared in the class member variable declaration.  
private Configuration hbaseConfig = null;  
//Instantiate this class using its constructor function or initialization method.  
hbaseConfig = HBaseConfiguration.create();
```

Incorrect:

```
hbaseConfig = new Configuration();
```

Share the Configuration instance

The HBase client codes obtain rights to interact with an HBase cluster by creating an HConnection with Zookeeper. Each HConnection has a Configuration instance. The created HConnection instances are cached. That is, if the HBase client needs to communicate with an HBase cluster, the client sends a Configuration instance to the cluster. Then, the HBase client checks for an HConnection instance for the Configuration instance in the cache. If a match is found, the HConnection instance is returned. If no match is found, an HConnection instance will be created.

If the Configuration instance is frequently created, a lot of unnecessary HConnection instances will be created, causing the number of connections to Zookeeper to reach the upper limit.

Therefore, it is recommended that the client codes share the same **Configuration** instance.

Create an Table instance

```
public abstract class TableOperationImpl {
    private static Configuration conf = null;
    private static Connection connection = null;
    private static Table table = null;
    private static TableName tableName = TableName.valueOf("sample_table");

    public TableOperationImpl() {
        init();
    }
    public void init() {
        conf = ConfigurationSample.getConfiguration();
        try {
            connection = ConnectionFactory.createConnection(conf);
            table = conn.getTable(tableName);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public void close() {
        if (table != null) {
            try {
                table.close();
            } catch (IOException e) {
                System.out.println("Can not close table.");
            } finally {
                table = null;
            }
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (IOException e) {
                System.out.println("Can not close connection.");
            } finally {
                connection = null;
            }
        }
    }
    public void operate() {
        init();
        process();
        close();
    }
}
```

An Table instance cannot be used by multiple threads at the same time

Table is not thread safe for reads or write. If an Table instance is used by multiple threads at the same time, exceptions will occur.

Cache a frequently used Table instance

Cache the Table instance that will be frequently used by a thread for a long period of time. A cached instance, however, will not be necessarily used by a thread permanently. In special circumstances, you need to rebuild an Table instance. See the next rule for details.

Correct:

 NOTE

In this example, the Table instance is cached by Map. This method applies when multiple threads and Table instances are required. If an Table instance is used by only one thread and the thread has only one Table instance, Map need not be used.

```
//In this Map, TableName is the Key value. Cache all Table instances.
private Map<String, Table> demoTables = new HashMap<String, Table>();
//All Table instances share this Configuration instance.
private Configuration demoConf = null;
/**
 * <Initialize an HTable class>
 * <Detailed function description>
 * @param tableName
 * @return
 * @throws IOException
 * @see [class, class#method, class#member]
 */
private Table initNewTable(String tableName) throws IOException
{
    try (Connection conn = ConnectionFactory.createConnection(demoConf)){
        return conn.getTable(tableName);
    }
}
/**
 * <Obtain Table instances>
 * <Detailed function description>
 * @see [class, class#method, class#member]
 */
private Table getTable(String tableName)
{
    if (demoTables.containsKey(tableName))
    {
        return demoTables.get(tableName);
    } else {
        Table table = null;
        try
        {
            table = initNewTable(tableName);
            demoTables.put(tableName, table);
        }
        catch (IOException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return table;
    }
}
/**
 * <Write data>
 * <Multi-thread multi-Table instance design optimization is not involved. The synchronization method is used
 * because the Table is not thread safe. It is recommended that an Table instance be used by only one data
 * write thread at the same
 * time.>
 * @param dataList
 * @param tableName
 * @see [class, class#method, class#member]
 */
public void putData(List<Put> dataList, String tableName)
{Table table = getTable(tableName);
//Synchronization is not required if the Table instance is not shared by multiple threads.
//Note that Table is not thread safe.
synchronized (table)
{
    try
    {
        table.put(dataList);
        table.notifyAll();
    }
}
```

```

}
catch (IOException e)
{
    // When IOE is detected, the cached instance needs to be re-created.
try {
    // Close the Connection.
    table.close();
    // Re-create the instance.
    table = initNewTable(tableName);
} catch (IOException e1) {
// TODO
}
}
}
}
}
}

```

Incorrect:

```

public void putDataIncorrect(List<Put> dataList, String tableName)
{Table table = null;
try
{
//Create an HTable instance each time when data is written.
table = initNewTable(tableName);
table.put(dataList);
}
catch (IOException e1)
{
// TODO Auto-generated catch block
e1.printStackTrace();
}
finally
{
table.close();
}
}
}

```

Rebuild an Table instance

Rebuild a cached Table when IOException is detected. See the example of the previous rule.

Do not call the following methods unless necessary:

- **Configuration#clear**

Do not call this method if a Configuration is used by an object or a thread. The Configuration#clear method clears all attributes loaded. If this method is called for a Configuration used by Table, all the parameters of this Configuration will be deleted from Table. As a result, an exception occurs when Table uses the Configuration the next time.

Therefore, avoid calling this method each time you rebuild an Table instance. Call this method when all the threads need to quit.

- **HConnectionManager#deleteAllConnections**

This method deletes all connections from the **Connection set**. As the Table stores the links to the connections, the connections being used cannot be stopped after the HConnectionManager#deleteAllConnections method is called, which eventually causes information leakage.

Handle the data failed to write

Some data write operations may fail due to instant exceptions or process failures. Therefore, the data must be recorded so that it can be written to the HBase when the cluster is restored.

The failed data returned by the HBase client will not be automatically rewritten. The interface caller is only informed of the data failed to be written. To prevent data loss, measures must be taken to temporarily save the data in a file or in memory.

Correct:

```
private List<Row> errorList = new ArrayList<Row>();
/**
 * <Insert data in PutList mode. >
 * <Synchronization is not required if the method is not called by multiple threads.>
 * @param put a data record
 * @throws IOException
 * @see [class, class#method, class#member]
 */
public synchronized void putData(Put put)
{
    // Temporarily cache data in this List.
    dataList.add(put);
    // Perform a Put operation when the dataList size reaches PUT_LIST_SIZE.
    if (dataList.size() >= PUT_LIST_SIZE)
    {
        try
        {
            demoTable.put(dataList);
        }
        catch (IOException e)
        {
            // If RetriesExhaustedWithDetailsException occurs,
            // certain data failed to be written, which
            // is caused by process errors in the HBase cluster or migration of a large number of
            // Regions.
            if (e instanceof RetriesExhaustedWithDetailsException)
            {
                RetriesExhaustedWithDetailsException ree =
                    (RetriesExhaustedWithDetailsException)e;
                int failures = ree.getNumExceptions();
                for (int i = 0; i < failures; i++)
                {
                    errorList.add(ree.getRow(i));
                }
            }
        }
        dataList.clear();
    }
}
```

Release resources

Call the Close method to release resources when the ResultScanner and Table instances are not required. To enable the Close method to be called, add the Close method to the **finally** block.

Correct:

```
ResultScanner scanner = null;
try
{
    scanner = demoTable.getScanner(s);
    //Do Something here.
```

```
}  
finally  
{  
  scanner.close();  
}
```

Incorrect:

1. The code does not call the scanner.close() method to release resources.
2. The scanner.close() method is not placed in the **finally** block.

```
ResultScanner scanner = null;  
scanner = demoTable.getScanner(s);  
//Do Something here.  
scanner.close();
```

Add fault-tolerance mechanism for Scan

Exceptions, such as lease expiration, may occur when Scan is performed. Retry operations need to be performed when exceptions occur.

Retry operations can be applied in HBase-related interface methods to improve fault tolerance capabilities.

Stop Admin as soon as it is not required

Stop Admin as soon as possible. Do not cache the same Admin instance for an extended period of time.

36.6.3 Suggestions

Do not call the closeRegion method of Admin to close a Region

Admin interface provides an API to close a Region:

public void closeRegion(final String regionname, final String serverName)

When this method is used to close a Region, the HBase Client sends an RPC request to the RegionServer of the Region to be closed. The Master is unaware of the whole process. That is, the Master does not know even if the Region is closed. If the closeRegion method is called when the Master determines to migrate the Region based on the execution result of Balance, the Region cannot be closed or migrated. (In the current HBase version, this issue has not been resolved).

Therefore, do not call the closeRegion method of Admin to close a Region.

Write data in PutList mode

Table provides two data write interfaces:

- public void put(final Put put) throws IOException
- public void put(final List<Put> puts) throws IOException

The second one is recommended because it provides better performance than the first one.

Specify StartKey and EndKey for a Scan

A Scan with a specific range offers higher performance than a Scan without specific range.

Example:

```
Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("familyname"), Bytes.toBytes("columnname"));
scan.setStartRow( Bytes.toBytes("rowA")); // StartKey is rowA.
scan.setStopRow( Bytes.toBytes("rowB")); // EndKey is rowB.
for(Result result : demoTable.getScanner(scan)) {
    // process Result instance
}
```

Do not disable WAL

Write-Ahead-Log (WAL) allows data to be written in a log file before being stored in the database.

WAL is enabled by default. The Put class provides an interface to disable WAL:

```
public void setWriteToWAL(boolean write)
```

If WAL is disabled (writeToWAL is set to False), data of the last 1s (The time can be specified by the **hbase.regionserver.optionallogflushinterval** parameter on the RegionServer. It is 1s by default) will be lost. WAL can be disabled only when high data write speed is required and data loss of the last 1s is allowed.

Set blockcache to true when creating a table or when Scan is performed

Set blockcache to true when a table is created or when Scan is performed on the HBase client. If there are a large number of repeated records, setting this parameter to true can improve efficiency.

By default, blockcache is true. Avoid setting this parameter to false forcibly, for example:

```
HColumnDescriptor fieldADesc = new HColumnDescriptor("value".getBytes());
fieldADesc.setBlockCacheEnabled(false);
```

The HBase does not support query by Orderby or with the search criteria specified. It is based on the lexicographic order and can only be read by Rowkey.

HBase should not be used in scenarios of random query and sequencing.

Suggestions on Services List Design

1. Pre-allocate regions in a balanced manner in order to improve concurrency capabilities.
2. Avoid excessive hotspot regions. Import the time factor to Rowkey if necessary.
3. It is preferred that concurrently accessed data be stored continuously. Concurrently read data should be stored nearby, on the same row and in the same cell.

4. Put frequently queried attributes property before Rowkey. Rowkey should be designed to match the main query criteria in terms of criterion sequencing.
5. Attributes with high dispersions should be contained in RowKey. Design the services list based on data dispersion and query scenarios.
6. Store redundant information to enhance indexing performance. Use secondary index to adapt to more query scenarios.
7. Enable automatic deletion of expired data by setting the expiration time and version quantity.

 **NOTE**

In the HBase, Regions busy writing data are called hotspot Region.

36.6.4 Examples

Set Configuration parameters

To set up a connection between an HBase Client and the HBase Server, set the following parameters:

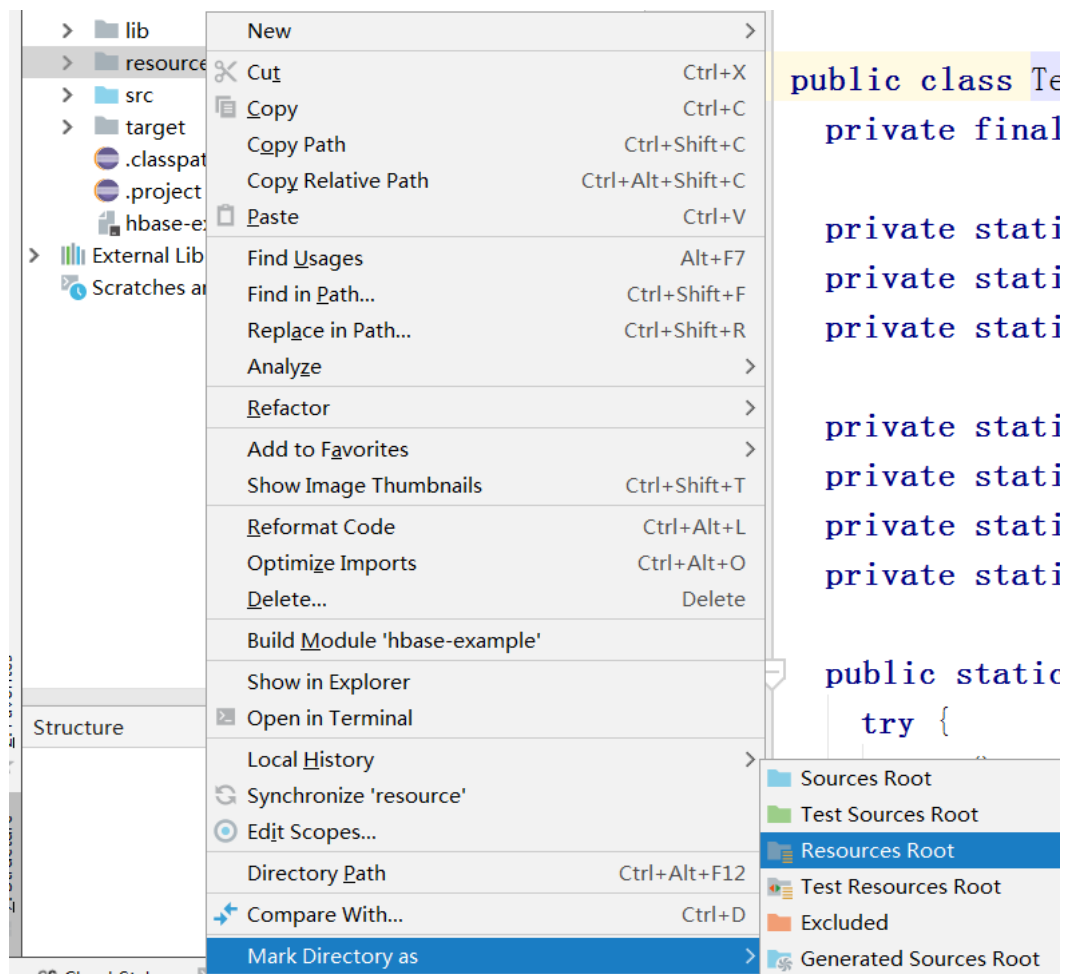
- `hbase.zookeeper.quorum`: IP address of Zookeeper. If there are multiple Zookeeper nodes, separate multiple IP addresses by a comma (,).
- `hbase.zookeeper.property.clientPort`: Port of Zookeeper.

 **NOTE**

The Configuration instance created by using `HBaseConfiguration.create()` will be automatically loaded with the configuration items in the following files:

- `core-default.xml`
- `core-site.xml`
- `hbase-default.xml`
- `hbase-site.xml`

Save these configuration files in **Source Folder**. To create a **Source Folder**, create a **resource** folder in the project, right-click the folder, and choose **Mark Directory as > Resources Root**.



The following table describes the parameters to be configured on the client.

NOTE

Do not change the values of these parameters.

Parameter	Description
hbase.client.pause	Specifies the time to wait before a retry is performed when an exception occurs. The actual time is calculated based on this value and the number of retries.
hbase.client.retries.number	Specifies the number of retries to be performed when an exception occurs.
hbase.client.retries.longer.multiplier	This parameter is related to the number of retries.
hbase.client.rpc.maxattempts	Specifies the number of retries when the RPC request is not successfully sent.
hbase.regionserver.lease.period	This parameter (in ms) is related to the Scanner timeout period.

Parameter	Description
hbase.client.write.buffer	This parameter is invalid if AutoFlush is enabled. If AutoFlush is disabled, the HBase Client caches the data to be written. When the size of the data cached reaches the specified limit, the HBase Client initiates a write operation to the HBase cluster.
hbase.client.scanner.caching	Specifies the number of rows allowed for a next request during a Scan.
hbase.client.keyvalue.maxsize	Specifies the maximum value of a keyvalue.
hbase.htable.threads.max	Specifies the maximum number of threads related to data operations in an HTable instance.
hbase.client.prefetch.limit	Before reading or writing data, the client must obtain the address of the Region. Therefore, a client can have some Region addresses pre-cached. This parameter is related to the configuration of the number of Region addresses pre-cached.

Example:

```
hbaseConfig = HBaseConfiguration.create();
//You do not need to set the following parameters if they are specified in the configuration files.
hbaseConfig.set("hbase.zookeeper.quorum", "172.16.100.1,172.16.100.2,172.16.100.3");
hbaseConfig.set("hbase.zookeeper.property.clientPort", "2181");
```

Use HTablePool in multi-thread write operations

Use HTablePool for multiple data write threads. Observe the following when using HTablePool to perform multi-thread write operations:

1. Enable multiple data write threads to share the same HTablePool instance.
2. Specify maxSize of the HTableInterface instance when instantiating HTablePool. That is, instantiate the class using the following constructor function:

```
public HTablePool(final Configuration config, final int maxSize)
```

The value of maxSize can be determined based on Threads (the number of data write threads) and Tables (the number of user tables). Generally, maxSize cannot be greater than the product of Threads and Tables. (maxSize <= Threads x Tables)

3. The client thread obtains an HTableInterface instance with the table name of tableName using HTablePool#getTable(tableName).
4. An HTableInterface instance can be used by only one thread at a time.
5. If HTableInterface is not used, call HTablePool#putTable(HTableInterface table) to release it.

Example:

```
/**
 * A certain number of retries is required after a data write failure. The time to wait before each retry is
```



```

determined based on the number of retries performed.
*/
private static final int[] RETRIES_WAITTIME = {1, 1, 1, 2, 2, 4, 4, 8, 16, 32};
/**
 * Specify the number of retries.
 */
private static final int RETRIES = 10;
/**
 * The unit of the time to wait after a failure.
 */
private static final int PAUSE_UNIT = 1000;
private static Configuration hadoopConfig;
private static HTablePool tablePool;
private static String[] tables;
/**
 * <Initialize HTablePool>
 * <Function description>
 * @param config
 * @see [class, class#method, class#member]
 */
public static void initTablePool()
{
    DemoConfig config = DemoConfig.getInstance();
    if (hadoopConfig == null)
    {
        hadoopConfig = HBaseConfiguration.create();
        hadoopConfig.set("hbase.zookeeper.quorum", config.getZookeepers());
        hadoopConfig.set("hbase.zookeeper.property.clientPort", config.getZookeeperPort());
    }
    if (tablePool == null)
    {
        tablePool = new HTablePool(hadoopConfig, config.getTablePoolMaxSize());
        tables = config.getTables().split(",");
    }
}
public void run()
{
    // Initialize HTablePool. Initialize this instance only once because it is shared by multiple threads.
    initTablePool();
    for (;;)
    {
        Map<String, Object> data = DataStorage.takeList();
        String tableName = tables[(Integer)data.get("table")];
        List<Put> list = (List)data.get("list");
        // Use Row as the Key and save all puts in the List. This set is used only for querying the data failed to be
        // written when a write operation fails, because the Server only returns the Row of the data failed.
        Map<byte[], Put> rowPutMap = null;
        // Perform the operation again if it fails (even if some of the data failed to be written). Only the data failed
        // to be written is submitted each time.
        INNER_LOOP :
        for (int retry = 0; retry < RETRIES; retry++)
        {
            // Obtain an HTableInterface instance from HTablePool. Release the instance if it is not required.
            HTableInterface table = tablePool.getTable(tableName);
            try
            {
                table.put(list);
                // The operation is successful.
                break INNER_LOOP;
            }
            catch (IOException e)
            {
                // If the exception type is RetriesExhaustedWithDetailsException, some of the data failed to be written. The
                // exception occurs because the processes in the HBase cluster are abnormal or a large number of Regions are
                // being migrated.
                // If the exception type is not RetriesExhaustedWithDetailsException, insert all the data in the list again.
                if (e instanceof RetriesExhaustedWithDetailsException)
                {
                    RetriesExhaustedWithDetailsException ree =

```

```
(RetriesExhaustedWithDetailsException)e;
int failures = ree.getNumExceptions();
System.out.println("In this operation, [" + failures + "] data records failed to be inserted.");
// Instantiate the Map when a retry is performed upon the first failure.
if (rowPutMap == null)
{
rowPutMap = new HashMap<byte[], Put>(failures);
for (int m = 0; m < list.size(); m++)
{
Put put = list.get(m);
rowPutMap.put(put.getRow(), put);
}
}
//Clear the original data and then add the data failed to be written.
list.clear();
for (int m = 0; m < failures; m++)
{
list.add(rowPutMap.get(ree.getRow(m)));
}
}
}
finally
{
// Release the instance after using it.
tablePool.putTable(table);
}
// If an exception occurs, wait some time after releasing the HTableInterface instance.
try
{
sleep(getWaitTime(retry));
}
catch (InterruptedException e1)
{
System.out.println("Interrupted");
}
}
}
}
```

Create a Put instance

HBase is a column-oriented database. One column of data may correspond to multiple column families, and one column family may correspond to multiple columns. Before data is written, the column (column family name and column name) must be specified.

	ColumnFamily01					ColumnFamily02			
	column01	column02	column03	column04	column05	column01	column02	column03	column04
Row--01									
Row--02									
Row--03									
Row--04									
Row--05									
Row--06									
Row--07									
Row--08									

A Put instance must be created before a row of data is written in an HBase table. The Put instance data consists of (Key, Value). The Value can contain multiple columns of values.

When a (Key, Value) record is added to a Put instance, the family, qualifier, and value added are byte sets. Use the Bytes.toBytes method to convert character strings to byte sets. Do not use the String.toBytes method, because this method

cannot ensure correct data coding. Errors occur when the Key or Value contains Chinese characters.

Example:

```
//The column family name is privateInfo.
private final static byte[] FAMILY_PRIVATE = Bytes.toBytes("privateInfo");
//The privateInfo column family has two columns: "name" and "address".
private final static byte[] COLUMN_NAME = Bytes.toBytes("name");
private final static byte[] COLUMN_ADDR = Bytes.toBytes("address");
/**
 * <Create a Put instance. >
 * <A put instance with one column family and two columns of data is created. >
 * @param rowKey Key key value
 * @param name name
 * @param address address
 * @return
 * @see [class, class#method, class#member]
 */
public Put createPut(String rowKey, String name, String address)
{
    Put put = new Put(Bytes.toBytes(rowKey));
    put.add(FAMILY_PRIVATE, COLUMN_NAME, Bytes.toBytes(name));
    put.add(FAMILY_PRIVATE, COLUMN_ADDR, Bytes.toBytes(address));
    return put;
}
```

Create an HBaseAdmin instance

Example:

```
private Configuration demoConf = null;
private HBaseAdmin hbaseAdmin = null;
/**
 * <Constructor function>
 * Import the Configuration instances.
 */
public HBaseAdminDemo(Configuration conf)
{
    this.demoConf = conf;
    try
    {
        // Instantiate HBaseAdmin
        hbaseAdmin = new HBaseAdmin(this.demoConf);
    }
    catch (MasterNotRunningException e)
    {
        e.printStackTrace();
    }
    catch (ZooKeeperConnectionException e)
    {
        e.printStackTrace();
    }
}
/**
 * <Examples of method using>
 * <For details about more methods, see the HBase interface documents. >
 * @throws IOException
 * @throws ZooKeeperConnectionException
 * @throws MasterNotRunningException
 * @see [Class, class#method, class#member]
 */
public void demo() throws MasterNotRunningException, ZooKeeperConnectionException, IOException
{
    byte[] regionName = Bytes.toBytes("mrtest,jjj,1315449869513.fc41d70b84e9f6e91f9f01affdb06703.");
    byte[] encodeName = Bytes.toBytes("fc41d70b84e9f6e91f9f01affdb06703");
    // Reallocate a Region.
    hbaseAdmin.unassign(regionName, false);
}
```

```
// Actively initiate Balance.
hbaseAdmin.balancer();
// Move a Region. The second parameter is HostName+StartCode of RegionServer, for example,
// host187.example.com,60020,1289493121758. If this parameter is set to null, the Region will be moved at
// random.
hbaseAdmin.move(encodeName, null);
// Check whether a table exists.
hbaseAdmin.tableExists("tableName");
// Check whether a table is activated.
hbaseAdmin.isTableEnabled("tableName");
}
/**
 * <Method used to rapidly create a table >
 * <Create an HTableDescriptor instance, which contains description of the HTable to be created. Create the
 * column families, which are associated with the HColumnDescriptor instance. In this example, the column
 * family name is "columnName">.
 * @param tableName table name
 * @return
 * @see [Class, class#method, class#member]
 */
public boolean createTable(String tableName)
{
    try {
        if (hbaseAdmin.tableExists(tableName)) {
            return false;
        }
        HTableDescriptor tableDesc = new HTableDescriptor(tableName);
        HColumnDescriptor fieldADesc = new HColumnDescriptor("columnName".getBytes());
        fieldADesc.setBlocksize(640 * 1024);
        tableDesc.addFamily(fieldADesc);
        hbaseAdmin.createTable(tableDesc);
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
    return true;
}
```

36.6.5 Appendix

Parameters Batch and Caching for Scan

Batch: specifies the maximum number of data records returned each time when scan calls the next interface. It is related to the number of **columns** read each time.

Caching: specifies the maximum number of next values returned for an RPC request. It is related to the number of **rows** obtained by each RPC.

The following examples explain the functions of these two parameters in Scan:

A Region contains two rows (rowkey) of data in table A. Each row has 1000 columns, and each column has only one version, that is, each row has 1000 key values.

-	Colu A1	Colu A2	Colu A3	Colu A4	...	Colu N1	Colu N2	Colu N3	Colu N4
Row1	-	-	-	-	...	-	-	-	-
Row2	-	-	-	-	...	-	-	-	-

- **Example 1:** If Batch is not specified and Caching is 2, 2000 (Key, Value) records will be returned for each RPC request.
- **Example 2:** If Batch is set to 500 and Caching is 2, 1000 (Key, Value) records will be returned for each RPC request.
- **Example 3:** If Batch is set to 300 and Caching is 4, 1000 (Key, Value) records will be returned for each RPC request.

Further explanation of Batch and Caching

- Each Caching indicates a chance of data request.
- The value of Batch determines whether a row of data can be read in a Caching. If the value of Batch is smaller than the total columns in a row, this row of data can be read in at least two Caching operations (the next Caching starts from the data where the previous caching stops).
- Each Caching cannot cross rows. That is, if the value of Batch is not reached after a row of data is read, data of the next row will not be read.

This can further explain the results of the previous examples.

- **Example 1:**
Since Batch is not set, all columns of that row will be read by default. As Caching is 2, 2000 (Key, Value) records will be returned for each RPC request.
- **Example 2:**
Because Batch is 500 and Caching is 2, a maximum of 500 columns of data will be read in each Caching. Therefore, 1000 (Key, Value) records will be returned after two times of caching.
- **Example 3:**
Because Batch is 300 and Caching is 4, four times of caching are required to read 1000 data records. Therefore, only 1000 (Key, Value) records will be returned.

Code example:

```
Scan s = new Scan();
//Set the start and end keys for data query.
s.setStartRow(Bytes.toBytes("01001686138100001"));
s.setStopRow(Bytes.toBytes("01001686138100002"));
s.setBatch(1000);
s.setCaching(100);
ResultScanner scanner = null;
try {
    scanner = tb.getScanner(s);
    for (Result rr = scanner.next(); rr != null; rr = scanner.next()) {
        for (KeyValue kv : rr.raw()) {
            //Display the query results.
            System.out.println("key:" + Bytes.toString(kv.getRow())
                + "getQualifier:" + Bytes.toString(kv.getQualifier())
                + "value" + Bytes.toString(kv.getValue()));
        }
    }
} catch (IOException e) {
    System.out.println("error!" + e.toString());
} finally {
    scanner.close();
}
```

36.7 HDFS

36.7.1 Application Scenarios

Hadoop distribute file system (HDFS) runs on commodity hardware. It provides high error tolerance. In addition, it supports high data access throughput and is suitable for applications that involve large-scale data sets.

The HDFS is applicable to the following scenarios:

- Massive data processing (higher than the TB or PB level).
- High-throughput demanding scenarios.
- High-reliability demanding scenarios.
- Good-scalability demanding scenarios.

The HDFS is not applicable to the scenarios that involve a large number of small files, random write, and low-latency read.

36.7.2 Rules

Set the HDFS NameNode metadata storage path

NameNode metadata is stored in `#{BIGDATA_DATA_HOME}/namenode/data` by default. This parameter sets the storage path of HDFS metadata.

Enable NameNode image backup for the HDFS

`fs.namenode.image.backup.enable` specifies whether to enable the NameNode image backup function. You need to set this parameter to **true**. Then the system can periodically back up the NameNode data.

Set the HDFS DataNode data storage path

DataNode data is stored in `#{BIGDATA_DATA_HOME}/hadoop/dataN/dn/datadir` by default. *N* indicates the number of directories is greater than or equal to 1.

For example, `#{BIGDATA_DATA_HOME}/hadoop/data1/dn/datadir`, `#{BIGDATA_DATA_HOME}/hadoop/data2/dn/datadir`.

After the storage path is set, data is stored in the corresponding directory of each mounted disk on a node.

Improve HDFS read/write performance

The data write process is as follows:

After receiving service data and obtaining the data block number and location from the NameNode, the HDFS client contacts DataNodes and establishes a pipeline with the DataNodes to be written. Then, the HDFS client writes data to DataNode1 using a proprietary protocol, and DataNode1 writes data to

DataNode2 and DataNode3 (three duplicates). After data is written, a message is returned to the HDFS client.

1. Set a proper block size. For example, set **dfs.blocksize** to **268435456** (256 MB).
2. It is not necessary to cache the big data that is not reused. In this case, set the following parameters to **false**:
dfs.datanode.drop.cache.behind.reads and
dfs.datanode.drop.cache.behind.writes

Set the MapReduce intermediate file storage path

Only one default path is provided for storing MapReduce intermediate files, that is, **`\${hadoop.tmp.dir}/mapred/local**. It is recommended that intermediate files be stored on each disk.

For example, **`\${hadoop}/hdfs/data1/mapred/local**, **`\${hadoop}/hdfs/data2/mapred/local**, **`\${hadoop}/hdfs/data3/mapred/local**. Directories that do not exist are automatically ignored.

Release applied resources in finally during Java development.

Applied HDFS resources are released in try/finally and cannot be released outside the try statement only. Otherwise, resource leakage occurs.

HDFS file operation APIs

Almost all Hadoop file operation classes are in the **org.apache.hadoop.fs** package. These APIs support operations such as opening, reading, writing, and deleting a file. `FileSystem` is the interface class provided for users in the Hadoop class library. `FileSystem` is an abstract class. Concrete classes can be obtained only using the `get` method. The `get` method has multiple overload versions, and the following `get` method is often used.

```
static FileSystem get(Configuration conf);
```

This class encapsulates almost all file operations, such as `mkdir` and `delete`. The program library framework for file operations is as follows:

```
operator()  
{  
    Obtain the Configuration object.  
    Obtain the FileSystem object.  
    Perform file operations.  
}
```

HDFS initialization method

HDFS initialization is a prerequisite for using APIs provided by HDFS.

To initialize HDFS, load the HDFS service configuration file, implement Kerberos security authentication, and instantiate `FileSystem`. Obtain keytab files for Kerberos security authentication in advance.

Example:

```
private void init() throws IOException {  
    Configuration conf = new Configuration();
```

```
// Read a configuration file.
conf.addResource("user-hdfs.xml");
// Implement security authentication in security mode.
if ("kerberos".equalsIgnoreCase(conf.get("hadoop.security.authentication"))) {
String PRINCIPAL = "username.client.kerberos.principal";
String KEYTAB = "username.client.keytab.file";
// Set the keytab key file.
conf.set(KEYTAB, System.getProperty("user.dir") + File.separator + "conf" + File.separator +
conf.get(KEYTAB));
// Set the Kerberos configuration file path. */
String krbfilepath = System.getProperty("user.dir") + File.separator + "conf" + File.separator + "krb5.conf";
System.setProperty("java.security.krb5.conf", krbfilepath);
// Implement login authentication. */
SecurityUtil.login(conf, KEYTAB, PRINCIPAL);
}
// Instantiate FileSystem.
fSystem = FileSystem.get(conf);
}
```

Upload local files to the HDFS

FileSystem.copyFromLocalFile (Path src, Patch dst) is used to upload local files to a specified directory in the HDFS. *src* and *dst* indicate complete file paths.

Example:

```
public class CopyFile {
    public static void main(String[] args) throws Exception {
        Configuration conf=new Configuration();
        FileSystem hdfs=FileSystem.get(conf);
        //Local file
        Path src =new Path("D:\\HebutWinOS");
        //To the HDFS
        Path dst =new Path("/");
        hdfs.copyFromLocalFile(src, dst);
        System.out.println("Upload to"+conf.get("fs.default.name"));
        FileStatus files[]=hdfs.listStatus(dst);
        for(FileStatus file:files){
            System.out.println(file.getPath());
        }
    }
}
```

Create files on the HDFS

FileSystem.mkdirs (Path f) is used to create folders on HDFS. *f* indicates a complete folder path.

Example:

```
public class CreateDir {
    public static void main(String[] args) throws Exception{
        Configuration conf=new Configuration();
        FileSystem hdfs=FileSystem.get(conf);
        Path dfs=new Path("/TestDir");
        hdfs.mkdirs(dfs);
    }
}
```

Query the modification time of an HDFS file

FileSystem.getModificationTime() is used to query the modification time of a specified HDFS file.

Example:


```
public static void main(String[] args) throws Exception {
    Configuration conf=new Configuration();
    FileSystem hdfs=FileSystem.get(conf);
    Path fpath =new Path("/user/hadoop/test/file1.txt");
    FileStatus fileStatus=hdfs.getFileStatus(fpath);
    long modiTime=fileStatus.getModificationTime();
    System.out.println("file1.txt modification time is"+modiTime);
}
```

Read all files in an HDFS directory

FileStatus.getPath() is used to query all files in an HDFS directory.

Example:

```
public static void main(String[] args) throws Exception {
    Configuration conf=new Configuration();
    FileSystem hdfs=FileSystem.get(conf);
    Path listf =new Path("/user/hadoop/test");

    FileStatus stats[]=hdfs.listStatus(listf);
    for(int i = 0; i < stats.length; ++i) {
        System.out.println(stats[i].getPath().toString());
    }
    hdfs.close();
}
```

Query the location of a specified file in an HDFS cluster

FileSystem.getFileBlockLocation (FileStatus file, long start, long len) is used to query the location of a specified file in an HDFS cluster. *file* indicates a complete file path, and *start* and *len* specify the file path.

Example:

```
public static void main(String[] args) throws Exception {
    Configuration conf=new Configuration();
    FileSystem hdfs=FileSystem.get(conf);
    Path fpath=new Path("/user/hadoop/cygwin");

    FileStatus filestatus = hdfs.getFileStatus(fpath);
    BlockLocation[] blkLocations = hdfs.getFileBlockLocations(filestatus, 0, filestatus.getLen());

    int blockLen = blkLocations.length;
    for(int i=0;i < blockLen;i++){
        String[] hosts = blkLocations[i].getHosts();
        System.out.println("block_"+i+"_location:"+hosts[0]);
    }
}
```

Obtain all node names in an HDFS cluster

DatanodeInfo.getHostNames() is used to obtain all node names in an HDFS cluster.

Example:

```
public static void main(String[] args) throws Exception {
    Configuration conf=new Configuration();
    FileSystem fs=FileSystem.get(conf);

    DistributedFileSystem hdfs = (DistributedFileSystem)fs;

    DatanodeInfo[] dataNodeStats = hdfs.getDataNodeStats();
}
```

```
for(int i=0;i < dataNodeStats.length;i++){
    System.out.println("DataNode_"+i+"_Name:"+dataNodeStats[i].getHostName());
}
}
```

Multithread security login mode

If multiple threads are performing login operations, the relogin mode must be used for the subsequent logins of all threads after the first successful login of an application.

login example code:

```
private Boolean login(Configuration conf){
    boolean flag = false;
    UserGroupInformation.setConfiguration(conf);
    try {
        UserGroupInformation.loginUserFromKeytab(conf.get(PRINCIPAL), conf.get(KEYTAB));
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "
+UserGroupInformation.isLoginKeytabBased());
        flag = true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return flag;
}
```

relogin example code:

```
public Boolean relogin(){
    boolean flag = false;
    try {
        UserGroupInformation.getLoginUser().reloginFromKeytab();
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "
+UserGroupInformation.isLoginKeytabBased());
        flag = true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return flag;
}
```

NOTICE

Repetitive logins will cause a newly created session to overwrite the previous session. As a result, the previous session cannot be maintained or monitored, and some functions are unavailable after the previous session expires.

36.7.3 Suggestions

Notes for reading and writing HDFS files

The HDFS does not support random read/write.

Data can be appended only to the end of an HDFS file.

Only data stored in the HDFS supports append. **edit.log** and metadata files do not support append. When using the append function, set **dfs.support.append** in *hdfs-site.xml* to **true**.

 NOTE

- **dfs.support.append** is disabled by default in open-source versions but enabled by default in FusionInsight versions.
- This parameter is a server parameter. You are advised to enable this parameter to use the append function.
- Store data in other modes, such as HBase, if the HDFS is not applicable.

The HDFS is not suitable for storing a large number of small files

The HDFS is not suitable for storing a large number of small files because the metadata of small files will consume excessive memory resources of the NameNode.

Back up HDFS data in three duplicates

Three duplicates are enough for DataNode data backup. System data security is improved when more duplicates are generated but system efficiency is reduced. When a node is faulty, data on the node is balanced to other nodes.

Periodical HDFS Image Back-up

The system can back up the data on NameNode periodically after the image back-up parameter **fs.namenode.image.backup.enable** is set to **true**.

Provide operations to ensure data reliability

When you invoke the write function to write data, HDFS client does not write the data to HDFS but caches it in the client memory. If the client is abnormal, power-off, the data will be lost. For high-reliability demanding data, invoke `hflush` to refresh the data to HDFS after writing finishes.

36.8 Hive

36.8.1 Application Scenarios

Hive is an open-source data warehouse framework built on Hadoop. It provides storage of structured data and basic data analysis services using the Hive query language (HQL), a language like the structured query language (SQL). Hive converts HQL statements to MapReduce or Spark tasks to query and analyze massive data stored in Hadoop clusters.

Hive provides the following features:

- Extracts, transforms, and loads (ETL) data using HQL.
- Analyzes massive structured data using HQL.
- Supports multiple data storage formats, including JavaScript object notation (JSON), comma separated values (CSV), TextFile, RCFile, ORCFILE, and SequenceFile, and supports custom extensions.
- Multiple client connection modes. JDBC interfaces are supported.

Hive is applicable to offline massive data analysis (such as log and cluster status analysis), large scale data mining (such as user behavior analysis, interest region analysis, and region display), and other scenarios.

To ensure Hive high availability (HA), user data security, and service access security, Huawei MRS incorporates the following features based on Hive 3.1.0:

- Kerberos security authentication.
- Data file encryption.
- Comprehensive rights management.

36.8.2 Rules

Load the Hive JDBC driver

The client software connects to HiveServer using Java database connectivity (JDBC). Therefore, you must load the JDBC driver class `org.apache.hive.jdbc.HiveDriver` for Hive.

Use the current class loader to load the driver class.

If there is no jar package in classpath, the client software throws "Class Not Found" and exits.

Example:

```
Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance();
```

Set up a database connection

The driver management class `java.sql.DriverManager` of JDK is used to obtain a connection to the Hive database.

```
The Hive database URL is url="jdbc:hive2://xxx10.64xxx.22xxx.231xxx:2181,10xxx.64xxx.22xxx.232xxx:2181,10xxx.64xxx.22xxx.233xxx:2181;/serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2;sasl.qop=auth-conf;auth=KERBEROS;principal=hive/hadoop.hadoop.com@HADOOP.COM;user.principal=hive/hadoop.hadoop.com;user.keytab=conf/hive.keytab";
```

In this example, ZooKeeper is deployed on three nodes and the default port is 2181. `xxx.xxx.xxx.xxx` indicates each of the IP addresses of the three nodes. The user name and password are null or empty because authentication has been performed successfully.

Example:

```
// Set up a connection.  
connection = DriverManager.getConnection(url, "", "");
```

Execute HQL

Note that the HQL statement cannot end with a semicolon (;).

Correct:

```
String sql = "SELECT COUNT(*) FROM employees_info";  
Connection connection = DriverManager.getConnection(url, "", "");
```

```
PreparedStatement statement = connection.prepareStatement(sql);  
resultSet = statement.executeQuery();
```

Incorrect:

```
String sql = "SELECT COUNT(*) FROM employees_info";  
Connection connection = DriverManager.getConnection(url, "", "");  
PreparedStatement statement = connection.prepareStatement(sql);  
resultSet = statement.executeQuery();
```

Close a database connection

After the client executes the HQL, close the database connection to prevent memory leakage.

Close the statement and connection objects of the JDK.

Example:

```
finally {  
    if (null != statement) {  
        statement.close();  
    }  
  
    // Close the JDBC connection.  
    if (null != connection) {  
        connection.close();  
    }  
}
```

HQL syntax used to check for null values

Use **is null** to check whether a field is empty, that is, the field has no value. Use **is not null** to check whether a field is not null, that is, the field has a value.

If you use **is null** for a character whose type is String and length is 0, False is returned. Use **col = ''** to check for null values, and use **col != ''** to check for non-null values.

Correct:

```
select * from default.tbl_src where id is null;  
select * from default.tbl_src where id is not null;  
select * from default.tbl_src where name = '';  
select * from default.tbl_src where name != '';
```

Incorrect:

```
select * from default.tbl_src where id = null;  
select * from default.tbl_src where id != null;  
select * from default.tbl_src where name is null;  
select * from default.tbl_src where name is not null;
```

Note that the type of the id field in the tbl_src table is Int, and the type of the name field is String.

The client configuration parameters must be consistent with the server configuration parameters

If the configuration parameters of the Hive, YARN, and HDFS servers of the cluster are modified, the related parameter in a client program will be modified. You need to check whether the configuration parameters submitted to the HiveServer before the configuration parameters are modified are consistent with those on the

servers. If the configuration parameters are inconsistent, modify them on the client and submit them to the HiverServer. In the following example, if the parameter of YARN in the cluster is modified, the parameter submitted to the HiverServer from the Hive client and sample program before the modification must be reviewed and modified.

Initial state:

The parameter configuration of YARN in the cluster is as follows:

```
mapreduce.reduce.java.opts=-Xmx2048M
```

The parameter configuration on the client is as follows:

```
mapreduce.reduce.java.opts=-Xmx2048M
```

The parameter configuration of YARN in the cluster after the modification is as follows:

```
mapreduce.reduce.java.opts=-Xmx1024M
```

If the parameter in the client program is not changed, the parameter is still valid. This will result in insufficient memory for reducer and lead to MR running failure.

Multithread security login mode

If multiple threads are performing login operations, the relogin mode must be used for the subsequent logins of all threads after the first successful login of an application.

login example code:

```
private Boolean login(Configuration conf){
    boolean flag = false;
    UserGroupInformation.setConfiguration(conf);

    try {
        UserGroupInformation.loginUserFromKeytab(conf.get(PRINCIPAL), conf.get(KEYTAB));
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "
+UserGroupInformation.isLoginKeytabBased());
        flag = true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return flag;
}
```

relogin example code:

```
public Boolean relogin(){
    boolean flag = false;
    try {

        UserGroupInformation.getLoginUser().reloginFromKeytab();
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "
+UserGroupInformation.isLoginKeytabBased());
        flag = true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return flag;
}
```

Prerequisites for using the REST interface of WebHCat to submit an MR task in Streaming mode

The REST interface depends on the streaming packages of Hadoop. Before submitting an MR task to WebHCat in Streaming mode, upload **hadoop-streaming-2.7.0.jar** to the specified path of the HDFS: **hdfs:///apps/templeton/hadoop-streaming-2.7.0.jar**. Log in to the node where the client and Hive service are installed. Assume that the client installation path is **/opt/client**.

```
source /opt/client/bigdata_env
```

Run the **kinit** command to log in to the node as the human-machine or machine-machine user.

```
hdfs dfs -put ${BIGDATA_HOME}/FusionInsight_HD_8.1.0.1/FusionInsight-Hadoop-*/hadoop/share/hadoop/tools/lib/hadoop-streaming-*.jar /apps/templeton/
```

/apps/templeton/ need to be modified based on different instances. The default instance uses **/apps/templeton/** and the Hive1 instance uses **/apps1/templeton/**. The others follow the same rule

Read and write operations cannot be performed on the same table at the same time

Currently, Hive does not support concurrent operations. Read and write operations cannot be performed on the same table at the same time. Otherwise, query results may be inaccurate and tasks may fail.

A bucket table does not support insert into

A bucket table does not support insert into, and only supports insert overwrite; otherwise, the number of files and the number of buckets will be inconsistent.

Prerequisites for using some REST interfaces of WebHCat

Some REST interfaces of WebHCat depend on the JobHistoryServer instance of MapReduce. The interfaces are as follows:

- `mapreduce/jar`(POST)
- `mapreduce/streaming`(POST)
- `hive`(POST)
- `jobs`(GET)
- `jobs/:jobid`(GET)
- `jobs/:jobid`(DELETE)

Hive Authorization Description

It is recommended that Hive authorization (databases, tables, or views) be performed on the Manager authorization page. Authorization in command-line interface is not recommended except in the **alter databases databases_name set owner='user_name'** scenario.

Hive on HBase partition tables cannot be created

Data of Hive on HBase tables is stored on HBase. Because HBase tables are divided into multiple partitions that are scattered on RegionServer, Hive on HBase partition tables cannot be created on Hive.

A Hive on HBase table does not support insert overwrite

HBase uses a RowKey to uniquely identify a record. If data to be inserted has the same RowKey as the existing data, HBase will use the new data to overwrite the existing data. If insert overwrite is performed for a Hive on HBase table on Hive, only data with the same RowKey will be overwritten.

36.8.3 Suggestions

HQL - implicit type conversion

If the query statements use the field value for filtering, do not use the implicit type conversion of Hive to compile HQL. The reason is that the implicit type conversion is not conducive to code reading and migration.

Correct:

```
select * from default.tbl_src where id = 10001;  
select * from default.tbl_src where name = 'TestName';
```

Incorrect:

```
select * from default.tbl_src where id = '10001';  
select * from default.tbl_src where name = TestName;
```

NOTE

Note that the type of the id field in the tbl_src table is Int, and the type of the name field is String.

HQL - object name length

The HQL object names include table names, field names, view names, and index names. It is recommended that the object name not exceed 30 bytes.

An error is reported if an object name of Oracle exceeds 30 bytes. PT also limits object names to 30 bytes.

Excessive long object names are not conducive to code reading, migration, and maintenance.

HQL - statistics of data records

To count the total number of records in a table, use `select count(1) from table_name`.

To count the number of valid records for a field in a table, use `select count(column_name) from table_name`.

JDBC - timeout limit

The JDBC provided by Hive supports timeout limit. The default value is 5 minutes. Users can use `java.sql.DriverManager.setLoginTimeout(int seconds)` to change the value. The unit of *seconds* is second.

UDF Management

It is recommended that the administrator creates permanent UDF. This is done to avoid repeated execution of the add jar statement and UDF redefining.

UDF of Hive has some default properties. For example, the default value of **deterministic** is **true** (indicating that the same result will be returned for the same input), and the default value of **stateful** is **true**. Corresponding annotations should be added when user-defined UDF conducts an internal data summary. The following is an example:

```
@UDFType(deterministic = false)
Public class MyGenericUDAFEvaluator implements Closeable {
```

Suggestions on Optimizing Table Partitions

1. It is advised to use partition tables and store data by day when data volume is large and statistics need to be collected on a daily basis.
2. In order to avoid excessive small files, add **distribute by** to the partition field during dynamic partition data insertion.

Suggestions on Optimizing Storage File Formats

Hive supports multiple storage formats, including TextFile, RCFile, ORC, Sequence, and Parquet. If you want to save storage space or query certain fields for the most of time, use columnar storage, for example, ORC files, to create tables.

36.8.4 Examples

JDBC Secondary Development Code Example 1

The following code example provides the following functions:

1. Provides the username and key file path in the JDBC URL address so that programs can automatically perform security logins and create Hive connections.
2. Runs HQL statements for creating, querying, and deleting tables.

```
package com.huawei.bigdata.hive.example;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.Properties;
```

```

import org.apache.hadoop.conf.Configuration;
import com.huawei.bigdata.security.LoginUtil;

public class JDBCExample {
    private static final String HIVE_DRIVER = "org.apache.hive.jdbc.HiveDriver";

    private static final String ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME = "Client";
    private static final String ZOOKEEPER_SERVER_PRINCIPAL_KEY = "zookeeper.server.principal";
    private static final String ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL = "zookeeper/hadoop.hadoop.com";

    private static Configuration CONF = null;
    private static String KRB5_FILE = null;
    private static String USER_NAME = null;
    private static String USER_KEYTAB_FILE = null;

    private static String zkQuorum = null;//IP address and port list of a ZooKeeper node
    private static String auth = null;
    private static String sasl_qop = null;
    private static String zooKeeperNamespace = null;
    private static String serviceDiscoveryMode = null;
    private static String principal = null;
    private static void init() throws IOException{
        CONF = new Configuration();

        Properties clientInfo = null;
        String userdir = System.getProperty("user.dir") + File.separator
            + "conf" + File.separator;
        InputStream fileInputStream = null;
        try{
            clientInfo = new Properties();
            //"hiveclient.properties" is the client configuration file. If the multi-instance feature is used, you need to
            replace the file with "hiveclient.properties" of the corresponding instance client.
            //"hiveclient.properties" file is stored in the config directory of the decompressed installation package
            of the corresponding instance client.
            String hiveclientProp = userdir + "hiveclient.properties" ;
            File propertiesFile = new File(hiveclientProp);
            fileInputStream = new FileInputStream(propertiesFile);
            clientInfo.load(fileInputStream);
        }catch (Exception e) {
            throw new IOException(e);
        }finally{
            if(fileInputStream != null){
                fileInputStream.close();
                fileInputStream = null;
            }
        }
        //The format of zkQuorum is "xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181";
        //"xxx.xxx.xxx.xxx" of zkQuorum indicates the IP address of the node where ZooKeeper locates. The
        default port is 2181.
        zkQuorum = clientInfo.getProperty("zk.quorum");
        auth = clientInfo.getProperty("auth");
        sasl_qop = clientInfo.getProperty("sasl.qop");
        zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
        serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
        principal = clientInfo.getProperty("principal");
        // Set a user name for the newly created user, where xxx indicates the username created previously. For
        example, if the created user is user, USER_NAME is user.
        USER_NAME = "xxx";

        if ("KERBEROS".equalsIgnoreCase(auth)) {
            // Set the keytab and krb5 file path on the client.
            USER_KEYTAB_FILE = userdir + "user.keytab";
            KRB5_FILE = userdir + "krb5.conf";
            System.setProperty("java.security.krb5.conf", KRB5_FILE);
            System.setProperty(ZOOKEEPER_SERVER_PRINCIPAL_KEY, ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);
        }
    }
}

```

```

/**
 * This example shows how to use the Hive JDBC interface to run the HQL command <br>.
 * <br>
 *
 * @throws ClassNotFoundException
 * @throws IllegalAccessException
 * @throws InstantiationException
 * @throws SQLException
 * @throws IOException
 */
public static void main(String[] args) throws InstantiationException,
    IllegalAccessException, ClassNotFoundException, SQLException, IOException{
    // Parameter Initialization
    init();

    // Define HQL. HQL must be a single statement and cannot contain ";".
    String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT,name STRING)",
        "SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};

    // Build JDBC URL
    StringBuilder sBuilder = new StringBuilder(
        "jdbc:hive2://").append(zkQuorum).append("/")

    if ("KERBEROS".equalsIgnoreCase(auth)) {
        sBuilder.append(";serviceDiscoveryMode=")
            .append(serviceDiscoveryMode)
            .append(";zooKeeperNamespace=")
            .append(zooKeeperNamespace)
            .append(";sasL.qop=")
            .append(sasL_qop)
            .append(";auth=")
            .append(auth)
            .append(";principal=")
            .append(principal)
            .append(";user.principal=")
            .append(USER_NAME)
            .append(";user.keytab=")
            .append(USER_KEYTAB_FILE)
            .append(";");
    } else {
        //Normal mode
        sBuilder.append(";serviceDiscoveryMode=")
            .append(serviceDiscoveryMode)
            .append(";zooKeeperNamespace=")
            .append(zooKeeperNamespace)
            .append(";auth=none");
    }
    String url = sBuilder.toString();

    // Load the Hive JDBC driver.
    Class.forName(HIVE_DRIVER);

    Connection connection = null;
    try {
        // Obtain the JDBC connection.
        // If the normal mode is used, the second parameter needs to be set to a correct username. Otherwise,
        the anonymous user will be used for login.
        connection = DriverManager.getConnection(url, "", "");

        // Create a table.
        // If data needs to be imported to the table, use the load statement to import data to the table, for
        example, import data from the HDFS to the table.
        // load data inpath '/tmp/employees.txt' overwrite into table employees_info;
        execDDL(connection,sqls[0]);
        System.out.println("Create table success!");

        // Query the table.
        execDML(connection,sqls[1]);
    }

```

```

// Delete the table
execDDL(connection,sqls[2]);
System.out.println("Delete table success!");
}catch (Exception e) {
    System.out.println("Create connection failed : " + e.getMessage());
}
finally {
    // Close the JDBC connection.
    if (null != connection) {
        connection.close();
    }
}
}

public static void execDDL(Connection connection, String sql)
throws SQLException {
    PreparedStatement statement = null;
    try {
        statement = connection.prepareStatement(sql);
        statement.execute();
    }
    finally {
        if (null != statement) {
            statement.close();
        }
    }
}

public static void execDML(Connection connection, String sql) throws SQLException {
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    ResultSetMetaData resultMetaData = null;

    try {
        // Execute HQL.
        statement = connection.prepareStatement(sql);
        resultSet = statement.executeQuery();

        // Export the queried column names to the console.
        resultMetaData = resultSet.getMetaData();
        int columnCount = resultMetaData.getColumnCount();
        for (int i = 1; i <= columnCount; i++) {
            System.out.print(resultMetaData.getColumnLabel(i) + '\t');
        }
        System.out.println();

        // Export the query results to the console.
        while (resultSet.next()) {
            for (int i = 1; i <= columnCount; i++) {
                System.out.print(resultSet.getString(i) + '\t');
            }
            System.out.println();
        }
    }
    finally {
        if (null != resultSet) {
            resultSet.close();
        }

        if (null != statement) {
            statement.close();
        }
    }
}
}

```

JDBC Secondary Development Code Example 2

The following code example provides the following functions:

1. Does not provide the username and key file path in the JDBC URL address to create Hive connections. Users perform security logins by themselves.
2. Runs HQL statements for creating, querying, and deleting tables.

 NOTE

When accessing ZooKeeper, programs need to use the jaas configuration file, for example, **user.hive.jaas.conf**. The details are as follows:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="D:\\workspace\\jdbc-examples\\conf\\user.keytab"
  principal="xxx@HADOOP.COM"
  useTicketCache=false
  storeKey=true
  debug=true;
};
```

You need to modify the keyTab path (absolute path) and principal in the configuration file based on the actual environment, and set environment variable **java.security.auth.login.config** to the file path.

```
package com.huawei.bigdata.hive.example;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.Properties;

import org.apache.hadoop.conf.Configuration;
import com.huawei.bigdata.security.LoginUtil;

public class JDBCExamplePreLogin {
  private static final String HIVE_DRIVER = "org.apache.hive.jdbc.HiveDriver";

  private static final String ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME = "Client";
  private static final String ZOOKEEPER_SERVER_PRINCIPAL_KEY = "zookeeper.server.principal";
  private static final String ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL = "zookeeper/hadoop";

  private static Configuration CONF = null;
  private static String KRB5_FILE = null;
  private static String USER_NAME = null;
  private static String USER_KEYTAB_FILE = null;

  private static String zkQuorum = null; //IP address and port list of a ZooKeeper node
  private static String auth = null;
  private static String sasl_qop = null;
  private static String zooKeeperNamespace = null;
  private static String serviceDiscoveryMode = null;
  private static String principal = null;
  private static void init() throws IOException{
    CONF = new Configuration();

    Properties clientInfo = null;
    String userdir = System.getProperty("user.dir") + File.separator
      + "conf" + File.separator;
    InputStream fileInputStream = null;
    try{
      clientInfo = new Properties();
      // "hiveclient.properties" is the client configuration file. If the multi-instance feature is used, you
      need to replace the file with "hiveclient.properties" of the corresponding instance client.
      // "hiveclient.properties" file is stored in the config directory of the decompressed installation
      package of the corresponding instance client.
      String hiveclientProp = userdir + "hiveclient.properties";
      File propertiesFile = new File(hiveclientProp);
```

```

fileInputStream = new FileInputStream(propertiesFile);
clientInfo.load(fileInputStream);
} catch (Exception e) {
    throw new IOException(e);
} finally {
    if (fileInputStream != null) {
        fileInputStream.close();
        fileInputStream = null;
    }
}
//The format of zkQuorum is "xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181";
//"xxx.xxx.xxx.xxx" of zkQuorum indicates the IP address of the node where ZooKeeper locates. The
default port is 2181.
zkQuorum = clientInfo.getProperty("zk.quorum");
auth = clientInfo.getProperty("auth");
sasL_qop = clientInfo.getProperty("sasL.qop");
zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
principal = clientInfo.getProperty("principal");
// Set a user name for the newly created user, where xxx indicates the username created previously.
For example, if the created user is user, USER_NAME is user.
USER_NAME = "xxx";

if ("KERBEROS".equalsIgnoreCase(auth)) {
    // Set the keytab and krb5 file path on the client.
    USER_KEYTAB_FILE = userdir + "user.keytab";
    KRB5_FILE = userdir + "krb5.conf";

    LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, USER_NAME,
USER_KEYTAB_FILE);
    LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY,
ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);

    // Security mode
    // Zookeeper Login Authentication
    LoginUtil.login(USER_NAME, USER_KEYTAB_FILE, KRB5_FILE, CONF);
}
}

/**
 * This example shows how to use the Hive JDBC interface to run the HQL command <br>
 * <br>
 *
 * @throws ClassNotFoundException
 * @throws IllegalAccessException
 * @throws InstantiationException
 * @throws SQLException
 * @throws IOException
 */
public static void main(String[] args) throws InstantiationException,
IllegalAccessException, ClassNotFoundException, SQLException, IOException {
    // Parameter Initialization
    init();

    // Define HQL. HQL must be a single statement and cannot contain ";".
    String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT,name STRING)",
        "SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};

    // Build JDBC URL
    StringBuilder sBuilder = new StringBuilder(
        "jdbc:hive2://").append(zkQuorum).append("/");

    if ("KERBEROS".equalsIgnoreCase(auth)) {
        sBuilder.append(";serviceDiscoveryMode=")
            .append(serviceDiscoveryMode)
            .append(";zooKeeperNamespace=")
            .append(zooKeeperNamespace)
            .append(";sasL.qop=")
            .append(sasL_qop)

```

```

        .append(";auth=")
        .append(auth)
        .append(";principal=")
        .append(principal)
        .append(";");
    } else {
        // Normal mode
        sBuilder.append(";serviceDiscoveryMode=")
            .append(serviceDiscoveryMode)
            .append(";zooKeeperNamespace=")
            .append(zooKeeperNamespace)
            .append(";auth=none");
    }
    String url = sBuilder.toString();

    // Load the Hive JDBC driver.
    Class.forName(HIVE_DRIVER);

    Connection connection = null;
    try {
        // Obtain the JDBC connection.
        // If the normal mode is used, the second parameter needs to be set to a correct username.
        // Otherwise, the anonymous user will be used for login.
        connection = DriverManager.getConnection(url, "", "");

        // Create a table.
        // If data needs to be imported to the table, use the load statement to import data to the table,
        // for example, import data from the HDFS to the table.
        // load data inpath '/tmp/employees.txt' overwrite into table employees_info;
        execDDL(connection,sqls[0]);
        System.out.println("Create table success!");

        // Query the table.
        execDML(connection,sqls[1]);

        // Delete the table
        execDDL(connection,sqls[2]);
        System.out.println("Delete table success!");
    }
    finally {
        // Close the JDBC connection.
        if (null != connection) {
            connection.close();
        }
    }
}

public static void execDDL(Connection connection, String sql)
throws SQLException {
    PreparedStatement statement = null;
    try {
        statement = connection.prepareStatement(sql);
        statement.execute();
    }
    finally {
        if (null != statement) {
            statement.close();
        }
    }
}

public static void execDML(Connection connection, String sql) throws SQLException {
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    ResultSetMetaData resultMetaData = null;

    try {
        // Execute HQL.

```



```
statement = connection.prepareStatement(sql);
resultSet = statement.executeQuery();

// Export the queried column names to the console.
resultMetaData = resultSet.getMetaData();
int columnCount = resultMetaData.getColumnCount();
for (int i = 1; i <= columnCount; i++) {
    System.out.print(resultMetaData.getColumnLabel(i) + '\t');
}
System.out.println();

// Export the query results to the console.
while (resultSet.next()) {
    for (int i = 1; i <= columnCount; i++) {
        System.out.print(resultSet.getString(i) + '\t');
    }
    System.out.println();
}
}
finally {
    if (null != resultSet) {
        resultSet.close();
    }

    if (null != statement) {
        statement.close();
    }
}
}
```

HCatalog Secondary Development Code Example

The following code example demonstrates how to use the HCatInputFormat and HCatOutputFormat interfaces provided by HCatalog to submit MapReduce jobs.

```
public class HCatalogExample extends Configured implements Tool {

    public static class Map extends
        Mapper<LongWritable, HCatRecord, IntWritable, IntWritable> {
        int age;
        @Override
        protected void map(
            LongWritable key,
            HCatRecord value,
            org.apache.hadoop.mapreduce.Mapper<LongWritable, HCatRecord,
                IntWritable, IntWritable>.Context context)
            throws IOException, InterruptedException {
            age = (Integer) value.get(0);
            context.write(new IntWritable(age), new IntWritable(1));
        }
    }

    public static class Reduce extends Reducer<IntWritable, IntWritable,
        IntWritable, HCatRecord> {
        @Override
        protected void reduce(
            IntWritable key,
            java.lang.Iterable<IntWritable> values,
            org.apache.hadoop.mapreduce.Reducer<IntWritable, IntWritable,
                IntWritable, HCatRecord>.Context context)
            throws IOException, InterruptedException {
            int sum = 0;
            Iterator<IntWritable> iter = values.iterator();
            while (iter.hasNext()) {
                sum++;
                iter.next();
            }
        }
    }
}
```

```
        HCatRecord record = new DefaultHCatRecord(2);
        record.set(0, key.get());
        record.set(1, sum);

        context.write(null, record);
    }
}

public int run(String[] args) throws Exception {
    Configuration conf = getConf();
    String[] otherArgs = args;

    String inputTableName = otherArgs[0];
    String outputTableName = otherArgs[1];
    String dbName = "default";

    @SuppressWarnings("deprecation")
    Job job = new Job(conf, "GroupByDemo");

    HCatInputFormat.setInput(job, dbName, inputTableName);
    job.setInputFormatClass(HCatInputFormat.class);
    job.setJarByClass(HCatalogExample.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(WritableComparable.class);
    job.setOutputValueClass(DefaultHCatRecord.class);

    OutputJobInfo outputjobInfo = OutputJobInfo.create(dbName,outputTableName, null);
    HCatOutputFormat.setOutput(job, outputjobInfo);
    HCatSchema schema = outputjobInfo.getOutputSchema();
    HCatOutputFormat.setSchema(job, schema);
    job.setOutputFormatClass(HCatOutputFormat.class);

    return (job.waitForCompletion(true) ? 0 : 1);
}

public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new HCatalogExample(), args);
    System.exit(exitCode);
}
}
```

36.9 Kafka

36.9.1 Application Scenarios

Kafka is a distributed message releasing and subscription system. It provides features such as message persistence, high-throughput, multi-client support, and real-time message processing. Kafka is applicable to online and offline message consumption as well as Internet service massive data collection scenarios, such as conventional data collection, active website tracking, aggregation of operation data in statistics systems (monitoring data), and log collection.

Reasons for using the message system

- **Decoupling:** The message system inserts a hidden, data-based interface layer during data processing.
- **Redundancy:** Message queues are persistent, preventing data loss.
- **Scalability:** Message queues are decoupled from data processing, facilitating the expansion of data processing.

- Recoverability: Data processing can be recovered in case of failures.
- Sequence guarantee: Message queues help ensure the message sequence and keep the messages in a partition in order.
- Asynchronous communication: Messages can join the queue for further processing when necessary.

36.9.2 Rules

Create Topics by calling Kafka APIs (AdminZkClient.createTopic)

- For Java programming languages, correct examples are as follows:

```
import kafka.zk.AdminZkClient;
import kafka.zk.KafkaZkClient;
import kafka.admin.RackAwareMode;
...
KafkaZkClient kafkaZkClient = KafkaZkClient.apply(zkUrl, JaasUtils.isZkSecurityEnabled(),
zkSessionTimeoutMs, zkConnectionTimeoutMs, Int.MaxValue(), Time.SYSTEM, "", "", null);
AdminZkClient adminZkClient = new AdminZkClient(kafkaZkClient);
adminZkClient.createTopic(topic, partitions, replicas, new Properties(), RackAwareMode.Enforced
$.MODULE$);
...
```
- For Scala programming languages, correct examples are as follows:

```
import kafka.zk.AdminZkClient;
import kafka.zk.KafkaZkClient;
...
val kafkaZkClient: KafkaZkClient = KafkaZkClient.apply(zkUrl, JaasUtils.isZkSecurityEnabled(),
zkSessionTimeoutMs, zkConnectionTimeoutMs, Int.MaxValue, Time.SYSTEM, "", "")
val adminZkClient: AdminZkClient = new AdminZkClient(kafkaZkClient)
adminZkClient.createTopic(topic, partitions, replicas)
```

The number of Partition copies must be less than or equal to the number of nodes

Copies of Topic Partitions in Kafka are used for improving data reliability. Copies of the same Partition are distributed on different nodes. Therefore, the number of Partition copies must be less than or equal to the number of nodes.

Set the fetch.message.max.bytes parameter of the Consumer client

The value of fetch.message.max.bytes must be equal to or greater than the maximum number of bytes of messages that the Producer client generates each time. If the value is too small, the messages generated by the Producer client cannot be consumed successfully by the Consumer client.

36.9.3 Suggestions

In the same group, the number of consumers and that of Topic Partitions to be consumed should be the same

If the number of consumers is greater than that of Topic Partitions, some consumers cannot consume Topics. If the number of consumers is smaller than that of Topic Partitions, concurrent consumption cannot be fully represented. Therefore, the number of consumers and that of Topic Partitions to be consumed should be the same.

Avoid writing data with single ultra-large log

Data with single ultra-large log can affect efficiency and writing. Under such circumstance, modify the values of the **max.request.size** and **max.partition.fetch.bytes** configuration items when initializing Kafka producer instances and consumer instances, respectively.

For example, set **max.request.size** and **max.partition.fetch.bytes** to **5252880**.

```
// Protocol type:configuration SASL_PLAINTEXT or PLAINTEXT
props.put(securityProtocol, kafkaProc.getValues(securityProtocol, "SASL_PLAINTEXT"));
// service name
props.put(saslKerberosServiceName, "kafka");
props.put("max.request.size", "5252880");
// Security protocol type
props.put(securityProtocol, kafkaProc.getValues(securityProtocol, "SASL_PLAINTEXT"));
// service name
props.put(saslKerberosServiceName, "kafka");
props.put("max.partition.fetch.bytes", "5252880");
```

36.10 Mapreduce

36.10.1 Application Scenarios

Files of Mapreduce are stored in the HDFS. MapReduce is a programming model used for parallel computation of large data sets (larger than 1 TB). It is advised to use MapReduce when the file being processed cannot be loaded to memory.

It is advised to use Spark if MapReduce is not required.

36.10.2 Rules

Inherit the Mapper abstract class.

The map() and setup() methods are called during the Map procedure of a MapReduce task.

Example:

```
public static class MapperClass extends
Mapper<Object, Text, Text, IntWritable> {
/**
 * map input. The key indicates the offset of the original file, and the value is a row of characters in the
 * original file.
 * The map input key and value are provided by InputFormat. You do not need to set them. By default,
 * TextInputFormat is used.
 */
public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {
//Custom implementation
}
/**
 * The setup() method is called only once before the map() method of a map task or the reduce() method
 * of a reduce task is called.*/
public void setup(Context context) throws IOException,
InterruptedException {
// Custom implementation
}
}
```

Inherit the Reducer abstract class.

The `reduce()` and `setup()` methods are called during the Reduce procedure of a MapReduce task.

Example:

```
public static class ReducerClass extends
    Reducer<Text, IntWritable, Text, IntWritable> {
    /**
     * @param The input is a collection iterator consisting of (key, value) pairs.
     * Each map puts together all the pairs with the same key. The reduce method sums the number of the
     * same keys.
     * Call context.write(key, value) to write the output to the specified directory.
     * Outputformat writes the (key, value) pairs output by reduce to the file system.
     * By default, TextOutputFormat is used to write the reduce output to the HDFS.
     */
    public void reduce(Text key, Iterable<IntWritable> values,
        Context context) throws IOException, InterruptedException {
        // Custom implementation
    }
    /**
     * The setup() method is called only once before the map() method of a map task or the reduce() method
     * of a reduce task is called.
     */
    public void setup(Context context) throws IOException,
        InterruptedException {
        // Custom implementation. Context obtains the configuration information.
    }
}
```

Submit a MapReduce task.

Use the `main()` method to create a job, set parameters, and submit the job to the Hadoop cluster.

Example:

```
public static void main(String[] args) throws Exception {
    Configuration conf = getConfiguration();
    // Input parameters for the main method: args[0] indicates the input path of the MR job. args[1] indicates
    the output path of the MR job.
    String[] otherArgs = new GenericOptionsParser(conf, args)
        .getRemainingArgs();
    if (otherArgs.length != 2) {
        System.err.println("Usage: <in> <out>");
        System.exit(2);
    }
    Job job = new Job(conf, "job name");
    // Locate the jar package of the major task.
    job.setJar("D:\\job-examples.jar");
    // job.setJarByClass(TestWordCount.class);
    // Set the map and reduce classes to be executed. You can also specify them in the configuration file.
    job.setMapperClass(TokenizerMapperV1.class);
    job.setReducerClass(IntSumReducerV1.class);
    // Set the combiner class. By default, it is not used. If it is used, it runs the same classes as reduce. Exercise
    care when using the Combiner class. You can also specify the combiner class in the configuration file.
    job.setCombinerClass(IntSumReducerV1.class);
    // Set the output type of the job. You can also specify it in the configuration file.
    job.setOutputKeyClass(Text.class);
}
```

```
job.setOutputValueClass(IntWritable.class);  
// Set the input and output paths for the job. You can also specify them in the configuration file.  
Path outputPath = new Path(otherArgs[1]);  
FileSystem fs = outputPath.getFileSystem(conf);  
// If the output path already exists, delete it.  
if (fs.exists(outputPath)) {  
    fs.delete(outputPath, true);  
}  
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
System.exit(job.waitForCompletion(true) ? 0 : 1);  
}
```

36.10.3 Suggestions

Specify the global configuration items in mapred-site.xml

The mapred-site.xml file contains the following configuration items:

```
setMapperClass(Class<extends Mapper> cls) ->"mapreduce.job.map.class"  
setReducerClass(Class<extends Reducer> cls) ->"mapreduce.job.reduce.class"  
setCombinerClass(Class<extends Reducer> cls) ->"mapreduce.job.combine.class"  
setInputFormatClass(Class<extends InputFormat> cls) ->"mapreduce.job.inputformat.class"  
setJar(String jar) ->"mapreduce.job.jar"  
setOutputFormat(Class< extends OutputFormat> theClass) ->"mapred.output.format.class"  
setOutputKeyClass(Class<> theClass) ->"mapreduce.job.output.key.class"  
setOutputValueClass(Class<> theClass) ->"mapreduce.job.output.value.class"  
setPartitionerClass(Class<extends Partitioner> theClass) ->"mapred.partitioner.class"  
setMapOutputCompressorClass(Class<extends CompressionCodec> codecClass)  
->"mapreduce.map.output.compress"&"mapreduce.map.output.compress.codec"  
setJobPriority(JobPriority prio) ->"mapreduce.job.priority"  
setQueueName(String queueName) ->"mapreduce.job.queueName"  
setNumMapTasks(int n) ->"mapreduce.job.maps"  
setNumReduceTasks(int n) ->"mapreduce.job.reduces"
```

36.10.4 Examples

Count the number of female netizens who dwell on online shopping for more than 2 hours at a weekend.

The operation involves three steps:

1. Filter the online time of female netizens in log files using the MapperClass inherited from the Mapper abstract class.
2. Calculate the online time of each female netizen and output information about the female netizens who dwell online for more than 2 hours using the ReducerClass inherited from the Reducer abstract class.
3. Use the main method to create a MapReduce job and then submit the MapReduce job to the Hadoop cluster.

Step 1: Use MapperClass to define the map() and setup() methods of the Mapper abstract class.

```
public static class MapperClass extends  
  
Mapper<Object, Text, Text, IntWritable> {  
    // Separator  
    String delim;  
    // Filter the sex.  
    String sexFilter;  
    private final static IntWritable timeInfo = new IntWritable(1);
```

```

private Text nameInfo = new Text();
/**
 * map input. The key indicates the offset of the original file, and the value is a row of characters in the
 * original file.
 * The map input key and value are provided by InputFormat. You do not need to set them. By default,
 * TextInputFormat is used.
 */
public void map(Object key, Text value, Context context)
    throws IOException, InterruptedException {
    // A row of characters read.
    String line = value.toString();
    if (line.contains(sexFilter)) {
        // Obtain the names.
        String name = line.substring(0, line.indexOf(delim));
        nameInfo.set(name);
        // Obtain information about the online time.
        String time = line.substring(line.lastIndexOf(delim),
            line.length());
        timeInfo.set(Integer.parseInt(time));
        // map outputs (key, value) pairs.
        context.write(nameInfo, timeInfo);
    }
}
/**
 * The setup() method is called only once before the map() method of a map task or the reduce() method
 * of a reduce task is called.
 */
public void setup(Context context) throws IOException,
    InterruptedException {
    // Obtain configuration information using Context.
    sexFilter = delim + context.getConfiguration().get("log.sex.filter", "female") + delim;
}
}

```

Step 2: Use CReducerClass to define the reduce() method of the Reducer abstract class.

```

public static class ReducerClass extends
    Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();
    // Total time limit.
    private int timeThreshold;
    /**
     * @param The input is a collection iterator consisting of (key, value) pairs.
     * Each map puts together all the pairs with the same key. The reduce method sums the number of the
     * same keys.
     * Call context.write(key, value) to write the output to the specified directory.
     * Outputformat writes the (key, value) pairs output by reduce to the file system.
     * By default, TextOutputFormat is used to write the reduce output to the HDFS.
     */
    public void reduce(Text key, Iterable<IntWritable> values,
        Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        // If the time is smaller than the time limit, no information will be returned.
        if (sum < timeThreshold) {
            return;
        }
        result.set(sum);
        // reduce output: key: female netizen information, value: online time
        context.write(key, result);
    }
    /**
     * The setup() method is called only once before the map() method of a map task or the reduce() method
     * of a reduce task is called.
     */
    public void setup(Context context) throws IOException,

```

```
InterruptedException {  
    // Obtain configuration information using Context.  
    timeThreshold = context.getConfiguration().getInt(  
        "log.time.threshold", 120);  
    }  
}
```

Step 3: Use the main() method to create a job, set parameters, and submit the job to the Hadoop cluster.

```
public static void main(String[] args) throws Exception {  
    Configuration conf = getConfiguration();  
    // Input parameters for the main method: args[0] indicates the input path of the MR job. args[1] indicates  
    // the output path of the MR job.  
    String[] otherArgs = new GenericOptionsParser(conf, args)  
        .getRemainingArgs();  
    if (otherArgs.length != 2) {  
        System.err.println("Usage: <in> <out>");  
        System.exit(2);  
    }  
    Job job = new Job(conf, "Collect Female Info");  
    // Locate the jar package of the major task.  
    job.setJar("D:\\mapreduce-examples\\hadoop-mapreduce-examples\\mapreduce-examples.jar");  
    // job.setJarByClass(TestWordCount.class);  
    // Set the map and reduce classes to be executed. You can also specify them in the configuration file.  
    job.setMapperClass(TokenizerMapperV1.class);  
    job.setReducerClass(IntSumReducerV1.class);  
    // Set the combiner class. By default, it is not used. If it is used, it runs the same classes as reduce. Exercise  
    // care when using the Combiner class. You can also specify the combiner class in the configuration file.  
    job.setCombinerClass(IntSumReducerV1.class);  
    // Set the output type of the job. You can also specify it in the configuration file.  
    job.setOutputKeyClass(Text.class);  
    job.setOutputValueClass(IntWritable.class);  
    // Set the input and output paths for the job. You can also specify them in the configuration file.  
    Path outputPath = new Path(otherArgs[1]);  
    FileSystem fs = outputPath.getFileSystem(conf);  
    // If the output path already exists, delete it.  
    if (fs.exists(outputPath)) {  
        fs.delete(outputPath, true);  
    }  
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
    System.exit(job.waitForCompletion(true) ? 0 : 1);  
}
```

36.11 Oozie

36.11.1 Application Scenarios

Oozie is a workflow engine used to manage Hadoop jobs. Oozie workflows are defined and described based on the directed acyclic graph (DAG). Oozie supports multiple workflow modes and workflow scheduled triggering mechanisms. Oozie provides features such as easy extensibility, convenient maintenance, and high reliability and works closely with each component in the Hadoop ecosystem.

Oozie workflows are classified into three types:

- **Workflow:** describes a complete basic service flow.
- **Coordinator:** is built on workflows, triggers workflows on a scheduled basis or based on the specified conditions.
- **Bundle:** is built on coordinators. It centrally schedules, controls, and manages coordinators.

Oozie provides the following features:

- Supports distribution, aggregation, and selection of workflow modes.
- Works closely with each component in the Hadoop ecosystem.
- Supports parameterized workflow variables.
- Supports scheduled workflow triggering.
- Supports high availability (HA).
- Provides a built-in web console that allows users to view and monitor workflows and view logs.

36.11.2 Rules

Modified configuration files except `job.properties` must be uploaded to the HDFS again

During workflow running, configuration files except `job.properties` are read from specified HDFS directories. If the file content in the HDFS is not synchronized, the modification does not take effect.

Each workflow has only one start node and one end node

Based on the syntax requirements, each workflow has only one start node and one end node. To handle the exceptions that may occur, multiple kill nodes can be configured.

In the same workflow, each node can be traversed only once

A workflow does not support the ring structure, so each node can be traversed only once. The execution on the next node starts only when the execution on the current node ends.

A default processing branch must be reserved during process branch selecting

A default processing branch must be reserved during node decision selecting. This prevents a workflow from entering uncontrollable error status and avoids uncontrollable process execution.

Correct:

```
<decision name="[NODE-NAME]">
  <switch>
    <case to="[NODE_NAME]">[PREDICATE]</case>
    ...
    <case to="[NODE_NAME]">[PREDICATE]</case>
    <default to="[NODE_NAME]"/>
  </switch>
</decision>
```

Fork and Join nodes must appear in pairs

Correct:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.2">
  ...
```

```
<fork name="[FORK-NODE-NAME]">
  <path start="[NODE-NAME]" />
  ...
  <path start="[NODE-NAME]" />
</fork>
...
<join name="[JOIN-NODE-NAME]" to="[NODE-NAME]" />
...
</workflow-app>
```

Tag sequence cannot be changed

The locations of tags in schemas are strictly restricted. Unnecessary tags can be omitted but the location sequence cannot be changed. The location sequence of tags in different schemas may be different.

Correct:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.2">
  ...
  <action name="[NODE-NAME]">
    <map-reduce>
      <resource-manager>[RESOURCE-MANAGER]</resource-manager>
      <name-node>[NAME-NODE]</name-node>
      <prepare>
        <delete path="[PATH]" />
        ...
        <mkdir path="[PATH]" />
        ...
      </prepare>
      ...
      <job-xml>[JOB-XML-FILE]</job-xml>
      <configuration>
        <property>
          <name>[PROPERTY-NAME]</name>
          <value>[PROPERTY-VALUE]</value>
        </property>
        ...
      </configuration>
      <file>[FILE-PATH]</file>
      ...
      <archive>[FILE-PATH]</archive>
      ...
    </map-reduce>
    <ok to="[NODE-NAME]" />
    <error to="[NODE-NAME]" />
  </action>
  ...
</workflow-app>
```

A Hive SQL statement ends with a semicolon (;) and supports single-row comment starting with double hyphens (--)

Correct:

```
create table A(id int, name string, dt string);
insert into A values(1, "a1", "20150625");
select * from A;
--drop table A;
```

36.11.3 Suggestions

The name of a node must reflect the functions of a node

Standard name:

```
<action name="copyData">
  <fs>
    <delete path='${nameNode}/user/result'/>
    <mkdir path='${nameNode}/user/result'/>
    <move source='${nameNode}/output-data/map-reduce'
      target='${nameNode}/user/result'/>
    <chmod path='${nameNode}/user/result'
      permissions='-rwxrw-rw-' dir-files='true'></chmod>
  </fs>
  <ok to="end"/>
  <error to="fail"/>
</action>
```

Non-standard name:

```
<action name="fsNode">
  <fs>
    <delete path='${nameNode}/user/result'/>
    <mkdir path='${nameNode}/user/result'/>
    <move source='${nameNode}/output-data/map-reduce'
      target='${nameNode}/user/result'/>
    <chmod path='${nameNode}/user/result'
      permissions='-rwxrw-rw-' dir-files='true'></chmod>
  </fs>
  <ok to="end"/>
  <error to="fail"/>
</action>
```

Configure pre-cleanup operations under Prepare for a task involving data overwriting

Correct:

```
<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="myfirstHadoopJob">
    <map-reduce>
      <resource-manager>foo:8021</resource-manager>
      <name-node>bar:8020</name-node>
      <prepare>
        <delete path="hdfs://foo:8020/usr/tucu/output-data"/>
      </prepare>
      <job-xml>/myfirstjob.xml</job-xml>
      <configuration>
        <property>
          <name>mapred.input.dir</name>
          <value>/usr/tucu/input-data</value>
        </property>
        <property>
          <name>mapred.output.dir</name>
          <value>/usr/tucu/input-data</value>
        </property>
        <property>
          <name>mapred.reduce.tasks</name>
          <value>${firstJobReducers}</value>
        </property>
        <property>
          <name>oozie.action.external.stats.write</name>
          <value>true</value>
        </property>
      </configuration>
    </map-reduce>
    <ok to="myNextAction"/>
    <error to="errorCleanup"/>
  </action>
  ...
</workflow-app>
```

36.12 Spark2x

36.12.1 Application Scenarios

Spark is a distributed batch processing framework. It provides analysis and mining and iterative memory computing capabilities and supports application development in multiple programming languages, including Scala, Java, and Python. Spark is applicable to the following scenarios:

- Data processing: Spark can process data quickly and has fault tolerance and scalability.
- Iterative computation: Spark supports iterative computation to meet the requirements of multi-step data processing logic.
- Data mining: Spark supports complex mining and analysis of massive data and supports multiple data mining and machine learning algorithms.
- Streaming Processing: Spark supports streaming processing with a second-level delay and supports multiple external data sources.
- Query Analysis: Spark supports standard SQL query analysis, provides the DSL (DataFrame), and supports multiple external input types.

36.12.2 Rules

Import the Spark class in Spark applications

- Example in Java:

```
//Class imported when SparkContext is created.
import org.apache.spark.api.java.JavaSparkContext
//Class imported for the RDD operation.
import org.apache.spark.api.java.JavaRDD
//Class imported when SparkConf is created.
import org.apache.spark.SparkConf
```
- Example in Scala:

```
//Class imported when SparkContext is created.
import org.apache.spark.SparkContext
//Class imported for the RDD operatin.
import org.apache.spark.SparkContext._
//Class imported when SparkConf is created.
import org.apache.spark.SparkConf
```

Understand each Java and Scala function

The RDD cannot be changed, that is, the objects in the RDD are unchangeable. Therefore, you must understand each Java and Scala function.

For example, sort user position information by time. If Scala is used, the code for sorting information by time is as follows:

```
/* The function is used to obtain the position track of a user.
 * The parameter trajectory: consists of - user name and position (time, longitude, latitude)
private def getTimesOfOneUser(trajectory: (String, Seq[(String, Float, Float)]), zone: Zone, arrive: Boolean): Int =
{
  // Sort the user position by time.
  val sorted: Seq[(String, Float, Float)] = trajectory._2.sortBy(x => x._1);
```

```
}  
}
```

If Java is used, you must create an object for trajectory_2 instead of sorting trajectory_2. The reason is that Java does not support the **Collections.sort(trajectory_2)** operation. This operation changes the **trajectory_2** object, which violates the rule that the RDD object cannot be changed. The Scala code can be executed successfully because the **sortBy()** function creates an object. The following gives examples of the code in Java:

Correct:

```
//Create an object for user position.  
List<Tuple3< String, Float, Float >> list = new ArrayList<Tuple3< String, Float, Float >>( trajectory_2);  
//Sort new objects.  
Collections.sort(list);
```

Incorrect:

```
//Sort user positions.  
Collections.sort(trajectory_2);
```

Pay attention to the parameter transfer between the Driver and Executor nodes in distributed cluster

When Spark is used for programming, certain code logic needs to be determined based on the parameter entered. Generally, the parameter is specified as a global variable and assigned a null value. The actual value is assigned before the SparkContext object is instantiated using the main function. However, in the distributed cluster mode, the jar package of the executable program will be sent to each Executor. If the global variable values are changed only for the nodes in the main function and are not sent to the functions executing tasks, an error of null pointer will be reported.

Correct:

```
object Test  
{  
  private var testArg: String = null;  
  def main(args: Array[String])  
  {  
    testArg = i;  
    val sc: SparkContext = new SparkContext(i);  
  
    sc.textFile(i)  
    .map(x => testFun(x, testArg));  
  }  
  
  private def testFun(line: String, testArg: String): String =  
  {  
    testArg.split(i);  
    return i;  
  }  
}
```

Incorrect:

```
//Define an object.  
object Test  
{  
  // Define a global variable and set it to null. Assign a value to this variable before the SparkContext object  
  is instantiated using the main function.  
  private var testArg: String = null;
```

```
//main function
def main(args: Array[String])
{
  pair
  testArg = j;
  val sc: SparkContext = new SparkContext(j);

  sc.textFile(j)
  .map(x => testFun(x));
}

private def testFun(line: String): String =
{
  testArg.split(...);
  return j;
}
}
```

No error will be reported in the local mode of Spark. However, in the distributed cluster mode, an error of null pointer will be reported. In the cluster mode, the jar package of the executable program is sent to each Executor for running. When the testFun function is executed, the system queries the value of testArg from the memory. The value of testArg, however, is changed only when the nodes of the main function are started and other nodes are unaware of the change. Therefore, the value returned by the memory is null, which causes an error of null pointer.

SparkContext.stop must be added before an application program stops

When Spark is used in secondary development, SparkContext.stop() must be added before an application program stops.

NOTE

When Java is used in application development, JavaSparkContext.stop() must be added before an application program stops.

When Scala is used in application development, SparkContext.stop() must be added before an application program stops.

The following use Scala as an example to describe correct and incorrect examples.

Correct:

```
//Submit a spark job.
val sc = new SparkContext(conf)

//Specific task
...

//The application program stops.
sc.stop()
```

Incorrect:

```
//Submit a spark job.
val sc = new SparkContext(conf)

//Specific task
...
```

If you do not add SparkContext.stop, the YARN page displays the failure information. In the same task, as shown in [Figure 36-1](#), the first program does not add SparkContext.stop(), while the second program adds SparkContext.stop.

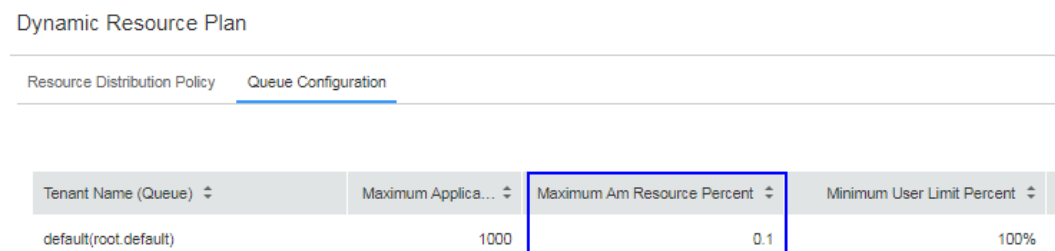
Figure 36-1 Difference when SparkContext.stop() is added

Application ID	User	Client	Engine	Queue	Start Time	End Time	Status	Result	History
application_1417593322234_0019	root	YarnClientWithoutStop	SPARK	default	Wed, 3 Dec 2014 08:49:42 UTC	Wed, 3 Dec 2014 08:49:51 UTC	FINISHED	FAILED	History
application_1417593322234_0018	root	YarnClientNormalStop	SPARK	default	Wed, 3 Dec 2014 08:48:59 UTC	Wed, 3 Dec 2014 08:49:12 UTC	FINISHED	SUCCEEDED	History

Appropriately plan the proportion of resources for AM

When there are many tasks and each task occupies few resources, the tasks may fail to start even if the cluster resources are sufficient and the tasks are submitted successfully. To address this issue, you can increase the value of **Max AM Resource Percent**.

Figure 36-2 Modify Max AM Resource Percent



36.12.3 Suggestions

Persist the RDD that will be frequently used

The default RDD storage level is StorageLevel.NONE, which means that the RDD is not stored on disks or in memory. If an RDD is frequently used, persist the RDD as follows:

Call `cache()`, `persist()`, or `persist(newLevel: StorageLevel)` of `spark.RDD` to persist the RDD. The `cache()` and `persist()` functions set the RDD storage level to `StorageLevel.MEMORY_ONLY`. The `persist(newLevel: StorageLevel)` function allows you to set other storage level for the RDD. However, before calling this function, ensure that the RDD storage level is `StorageLevel.NONE` or the same as the `newLevel`. That is, once the RDD storage level is set to a value other than `StorageLevel.NONE`, the storage level cannot be changed.

To unpersist an RDD, call `unpersist(blocking: Boolean = true)`. The function can:

1. Remove the RDD from the persistence list. The corresponding RDD data becomes recyclable.
2. Set the storage level of the RDD to `StorageLevel.NONE`.

Carefully select the the shuffle operator

This type of operator features wide dependency. That is, a partition of the parent RDD affects multiple partitions of the child RDD. The elements in an RDD are `<key, value>` pairs. During the execution process, the partitions of the RDD will be sequenced again. This operation is called shuffle.

Network transmission between nodes is involved in the shuffle operators. Therefore, for an RDD with large data volume, you are advised to extract

information as much as possible to minimize the size of each piece of data and then call the shuffle operators.

The following methods are often used:

- `combineByKey() : RDD[(K, V)] => RDD[(K, C)]`
This method is used to convert all the keys that have the same value in `RDD[(K, V)]` to a value with type of C.
- `groupByKey()` and `reduceByKey()` are two types of implementation of `combineByKey`. If `groupByKey` and `reduceByKey` cannot meet requirements in complex data aggregation, you can use customized aggregation functions as the parameters of `combineByKey`.
- `distinct(): RDD[T] => RDD[T]`
This method is used to remove repeated elements. The code is as follows:

```
map(x => (x, null)).reduceByKey((x, y) => x, numPartitions).map(_._1)
```


This process is time-consuming, especially when the data volume is high. Therefore, it is not recommended for the RDD generated from large files.
- `join() : (RDD[(K, V)], RDD[(K, W)]) => RDD[(K, (V, W))]`
This method is used to combine two RDDs through key.
If a key in `RDD[(K, V)]` has X values and the same key in `RDD[(K, W)]` has Y values, a total of (X * Y) data records will be generated in `RDD[(K, (V, W))]`.

Use high-performance operators if the service permits

1. Using `reduceByKey/aggregateByKey` to replace `groupByKey`
The map-side pre-aggregation refers to that each local node performs the aggregation operation on the same key, which is similar to the local combiner in MapReduce. The map-side pre-aggregation ensures that each key on a node is unique. When a node is collecting the data of the same key in the processing results of the previous nodes, data that needs to be obtained will be significantly reduced, decreasing disk I/O and Internet transmission cost. Generally speaking, it is advised to replace `groupByKey` operator with `reduceByKey` or `aggregateByKey` operator if possible because they will pre-aggregate the local same key on each node by using user-defined functions. However, the `groupByKey` operator does not support pre-aggregation and delivers lower performance than `reduceByKey` or `aggregateByKey` because all data are distributed and transmitted on all the nodes.
2. Using `mapPartitions` to replace ordinary map operators
During a function invocation, `mapPartitions` operators will process all the data in a partition instead of only one piece of data, and therefore delivers higher performance than the ordinary map operators. However, `mapPartitions` may occasionally result in Out of Memory (OOM). If memory is insufficient, some objects cannot be recycled during memory recycling. Therefore, exercise caution when using `mapPartitions`.
3. Performing the coalesce operation after filtering
After filtering a large portion of data (for example, above 30%) by using the filter operator in an RDD, you are advised to manually decrease the number of partitions by using `coalesce` in order to compress the data in RDD to fewer partitions. This is because after filtering, much data in each partition is filtered out, leaving little data to be processed. If the computing is continued,

resources can be wasted. The task handling speed decreases as the number of tasks increases. Therefore, decreasing the number of partitions by using `coalesce` to compress the RDD data to fewer partitions can ensure that all the partitions are handled with fewer tasks. The performance can also be enhanced in some scenarios.

4. Using `repartitionAndSortWithinPartitions` to replace `repartition` and `sort`
`repartitionAndSortWithinPartitions` is recommended by Spark official website. It is advised to use `repartitionAndSortWithinPartitions` for sorting after repartitioning. This operator can sort and shuffle repartitions at the same time, delivering higher performance.
5. Using `foreachPartitions` to replace `foreach`
Similar to "Using `mapPartitions` to replace ordinary map operators", this mechanism handles all the data in a partition during a function invocation instead of one piece of data. In practice, `foreachPartitions` is proved to be helpful in improving performance. For example, the `foreach` function can be used to write all the data in RDD into MySQL. Ordinary `foreach` operators, write data piece by piece, and a database connection is established for each function invocation. Frequent connection establishments and destructions cause low performance. `foreachPartitions`, however, processes all the data in a partition at a time. Only one database connection is required for each partition. Batch insertion delivers higher performance.

RDD Shared Variables

In application development, when a function is transferred to a Spark operation (such as `map` and `reduce`) and runs on a remote cluster, the operation is actually performed on the independent copies of all the variables involved in the function. These variables will be copied to each machine. In general, reading and writing shared variables across tasks is apparently inefficient. Spark provides two shared variables that are commonly used: broadcast variable and accumulator.

Kryo can be used to optimize serialization performance in performance-demanding scenarios.

Spark offers two serializers:

`org.apache.spark.serializer.KryoSerializer`: high-performance but low compatibility

`org.apache.spark.serializer.JavaSerializer`: average performance and high compatibility

Method: `conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")`

NOTE

The following are reasons why Spark does not use Kryo-based serialization by default:

Spark uses Java serialization by default, that is, uses the `ObjectOutputStream` and `ObjectInputStream` API to perform serialization and deserialization. Spark can also use Kryo serialization library, which delivers higher performance than Java serialization library. According to official statistics, Kryo-based serialization is 10 times more efficient than Java-based serialization. Kryo-based serialization requires the registration of all the user-defined types to be serialized, which is a burden for developers.

Suggestions on Optimizing Spark Streaming Performance

1. Set an appropriate batch processing duration (batchDuration).
2. Set concurrent data receiving appropriately.
 - Set multiple receivers to receive data.
 - Set an appropriate receiver congestion duration.
3. Set concurrent data processing appropriately.
4. Use Kryo-based serialization.
5. Optimize memory.
 - Set the persistence level to reduce GC costs.
 - Use concurrent Mark Sweep GC algorithms to shorten GC pauses.

36.13 Yarn

36.13.1 Application Scenarios

YARN is a distributed resource management system that is used to improve resource usage in the distributed cluster environment. Resources include memory, I/O, network resources, and disk resources. YARN is developed to address the shortage of the original MapReduce framework. Initially, MapReduce's committers were able to modify the existing codes periodically. As codes increase and due to the problems in the design of the original MapReduce framework, the modification becomes increasingly difficult. Therefore, MapReduce's committers decides to redesign the framework with enhancements to scalability, availability, reliability, and compatibility, increasing the resource usage of next-generation MapReduce(MRv2/Yarn) and supporting more computing frameworks apart from MapReduce.

36.13.2 Rules

Use YarnClient. createYarnClient() to create a client

Do not directly use the protocol interface
`ClientRMProxy.createRMProxy(config,ApplicationClientProtocol.class)` to create a client.

The Application Master uses the asynchronous interface `AMRMClientAsync.createAMRMClientAsync()` to interact with the ResourceManager

Do not directly use the protocol interface
`ClientRMProxy.createRMProxy(config,ApplicationMasterProtocol.class)` to create a client used by the Application Master to interact with the ResourceManager.

The Application Master uses the asynchronous interface `AMRMClientAsync.createAMRMClientAsync()` to interact with the `NodeMabager`

Do not directly use the `ContainerManagementProtocolProxy` interface to create a client used by the Application Master to interact with the `NodeManager`.

Multithread security login mode

If multiple threads are performing login operations, the relogin mode must be used for the subsequent logins of all threads after the first successful login of an application.

login example code:

```
private Boolean login(Configuration conf){
    boolean flag = false;
    UserGroupInformation.setConfiguration(conf);
    try {
        UserGroupInformation.loginUserFromKeytab(conf.get(PRINCIPAL), conf.get(KEYTAB));
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "
+UserGroupInformation.isLoginKeytabBased());
        flag = true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return flag;
}
```

relogin example code:

```
public Boolean relogin(){
    boolean flag = false;
    try {
        UserGroupInformation.getLoginUser().reloginFromKeytab();
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "
+UserGroupInformation.isLoginKeytabBased());
        flag = true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return flag;
}
```

37 Manager Management Development Guide

37.1 Overview

37.1.1 Application Development Overview

Intended Audience

This document is intended for users who need to access the FusionInsight Manager Rest APIs through HTTP basic authentication.

This document is intended for development personnel who are experienced in Java development.

Introduction to FusionInsight Manager

FusionInsight Manager, functioning as the O&M management system of cluster, provides services deployed in the cluster with a unified cluster management capability.

Manager provides functions such as installation and deployment, performance monitoring, alarms, user management, permission management, auditing, service management, health check and log collection.

37.1.2 Common Concepts

REST API

REST API is a set of APIs used to log in to the web server. REST APIs are executed through HTTP requests. Specifically, REST APIs receive the GET, PUT, POST, and DELETE requests and respond to the requests with JSON data.

The format of an HTTP request is as follows: **https://<Process IP>:<Process Port>/<Path>**

The *Process_IP* indicates the IP address of the server node where the process resides, and *Process_Port* indicates the process listening port.

For example, **https://10.162.181.57:32261/config**.

Basic Authentication

In HTTP, basic authentication is designed to allow a web browser or other client programs to provide credentials in the form of a username and password when making a request.

Before a request is sent, a space is added by using Basic to identify basic authentication, and the username is appended with a colon and concatenated with the password. Then, the character string is encoded using the Base64 algorithm.

For example:

If the username is **admin** and the password is **Asd#smSisn\$123**, the combined character string is **admin:Asd#smSisn\$123**. After the character string is encoded using the Base64 algorithm, **YWRtaW46QWRtaW5AMTlz** is generated. Then the basic authentication flag is added to obtain **Basic YWRtaW46QWRtaW5AMTlz**. Finally, the encoded character string is sent out, and the receiver decodes it to obtain a character string of the username and password separated by a colon.

37.1.3 Development Process

This document describes FusionInsight Manager application development based on the Java API.

Figure 37-1 shows and **Table 37-1** describes each phase of the development process.

Figure 37-1 FusionInsight Manager application development process

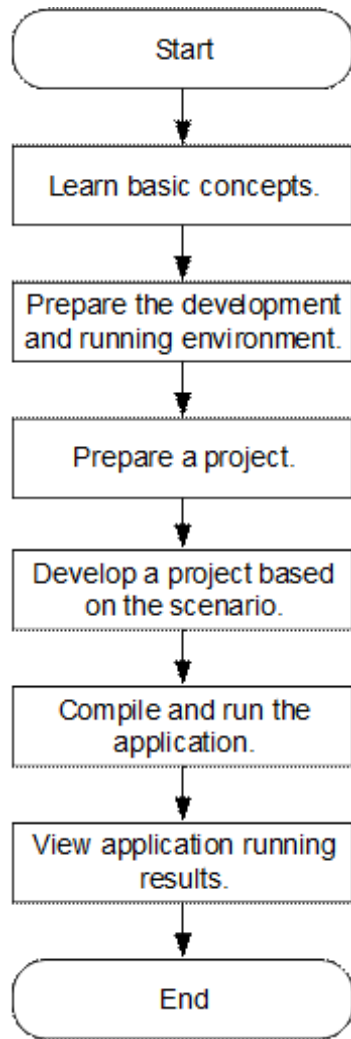


Table 37-1 Description of the FusionInsight Manager development process

Phase	Description	Reference
Learning basic concepts	Before application development, learn basic concepts of basic authentication, understand the scenario requirements, and design tables.	Common Concepts
Preparing the development and running environment	The Java language is recommended for FusionInsight Manager REST API application development. The IntelliJ IDEA tool can be used.	Preparing Development and Running Environments

Phase	Description	Reference
Preparing a project	FusionInsight Manager REST API provides example projects for different scenarios. You can import an example project to learn the application.	Configuring and Importing a Sample Project
Developing a project based on the scenario	Provide a sample project based on the Java language, including typical scenarios such as adding users, searching for users, modifying users, deleting users, and exporting user lists.	Developing an Application
Compiling and running the application	Compile the developed application and submit it for running.	Compiling and Running an Application
Viewing application running results	Application running results are exported to a specified path. Users can also view the application running status on the UI.	Viewing Windows Commissioning Results

37.2 Environment Preparation

37.2.1 Preparing Development and Running Environments

[Table 37-2](#) describes the development and running environment required for development.

Table 37-2 Development and running environment

Item	Description
Operating system (OS)	Development environment: Windows OS. Windows 7 or later version is supported. The local development environment must interconnect with the cluster service-plane network.

Item	Description
JDK installation	<p>Basic configuration for the development and running environment. The JDK must meet the following requirements:</p> <p>The JDK version number must be the same as that used by the FusionInsight Manager to be accessed. For details about the version number, see the corresponding version document or contact the system administrator.</p> <p>For example, JDK of versions later than 1.8.x must be used if FusionInsight Manager 8.1.0.1 is used.</p>
IntelliJ IDEA installation and configuration	Tool used for developing HBase applications. The version must be 2019.1 or other compatible version.

37.2.2 Configuring and Importing a Sample Project

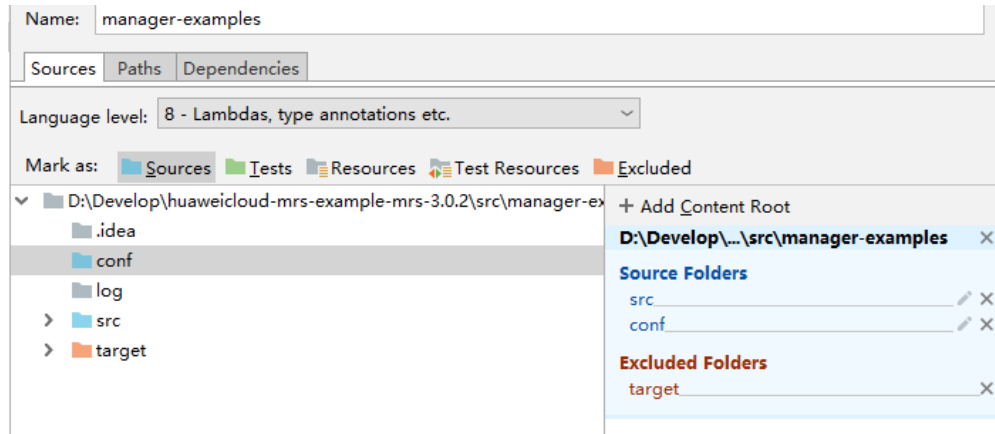
Procedure

- Step 1** Obtain the sample project folder **manager-examples** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.
1. Start IntelliJ IDEA, and then choose **File > New > Project from Existing Sources**. In the **Select File or Directory to Import** dialog box, select the sample code folder.
 2. Select the **pom.xml** file from the sample project folder, choose **Import project from external model > Maven**, and click **Next** and then **Finish**.

NOTE

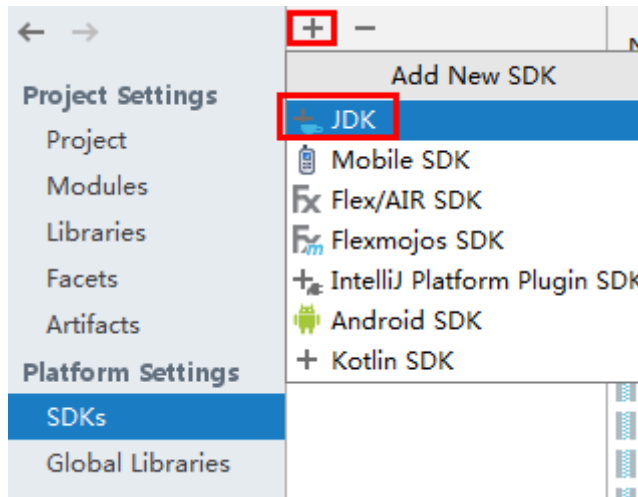
The sample code is a Maven project. You can adjust the project configuration based on the site requirements. The operations vary according to the IntelliJ IDEA version. The actual information displayed on the software interface prevails.

3. Add the **src** and **conf** directories in the project to the source file path.
After the project is imported, choose **File > Project Structure** on the menu bar of IntelliJ IDEA. In the displayed window, choose **Project Settings > Modules**.
Select the current project, click the **src** folder, click **Sources** on the right of **Mark as**, click the **conf** folder, and click **Sources** on the right of **Mark as**. Click **Apply** and then **OK**.

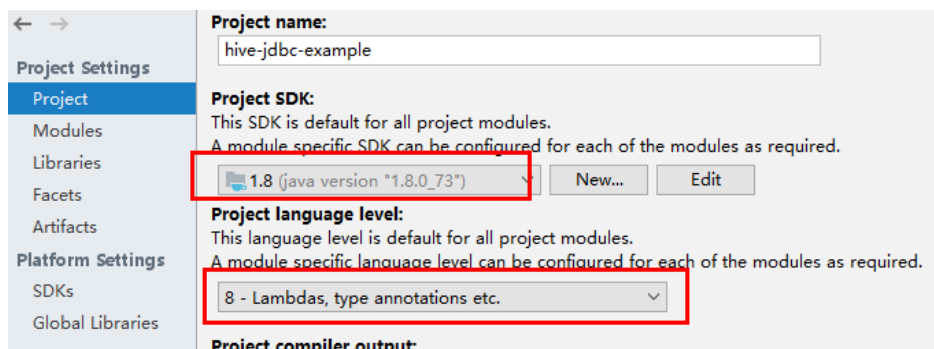


Step 3 Set the JDK of the project.

1. On the IntelliJ IDEA menu bar, choose **File > Project Structure....** The **Project Structure** window is displayed.
2. Select **SDKs**, click the plus sign (+), and select **JDK**.

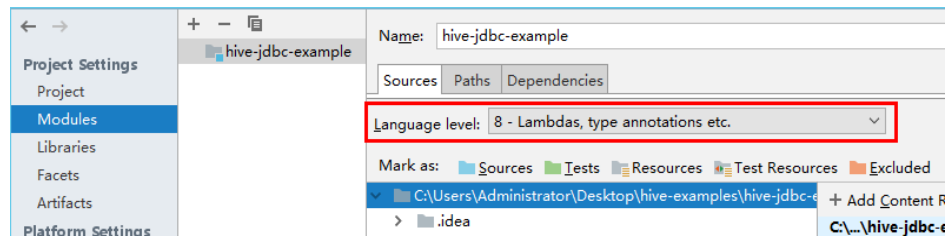


3. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
4. After selecting the JDK, click **Apply**.
5. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.

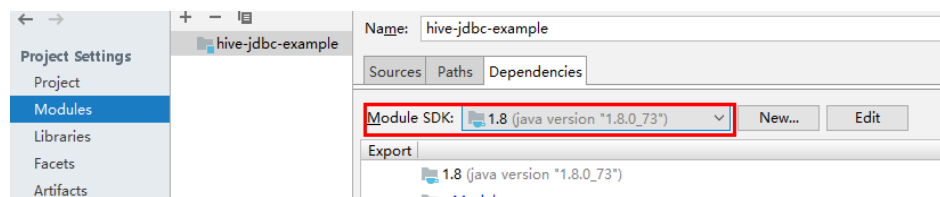


6. Click **Apply**.

7. Choose **Modules**. On the **Source** page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



On the **Dependencies** page, change the value of **Module SDK** to the JDK added in SDKs.

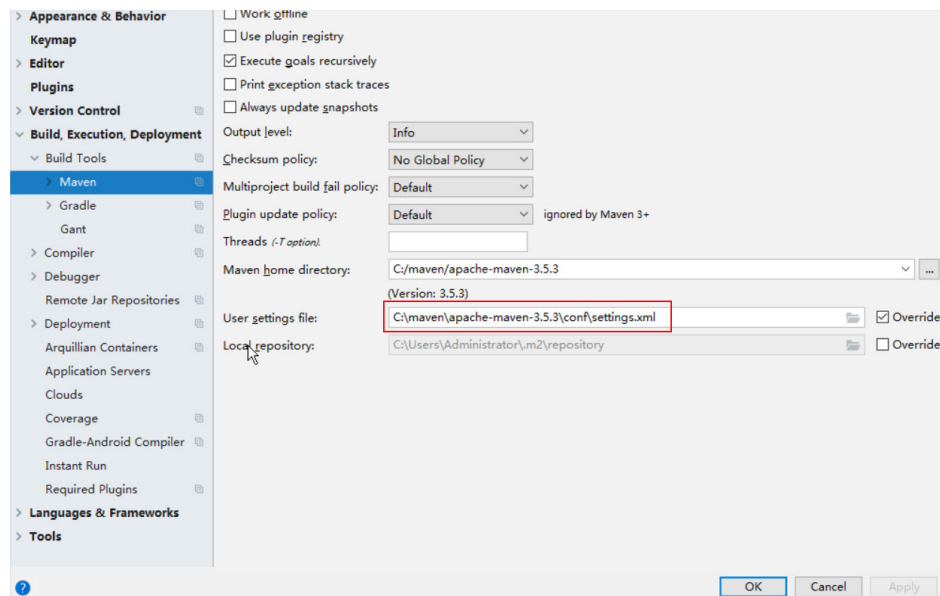


8. Click **Apply** and **OK**.

Step 4 Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is `<Local Maven installation directory>\conf\settings.xml`.

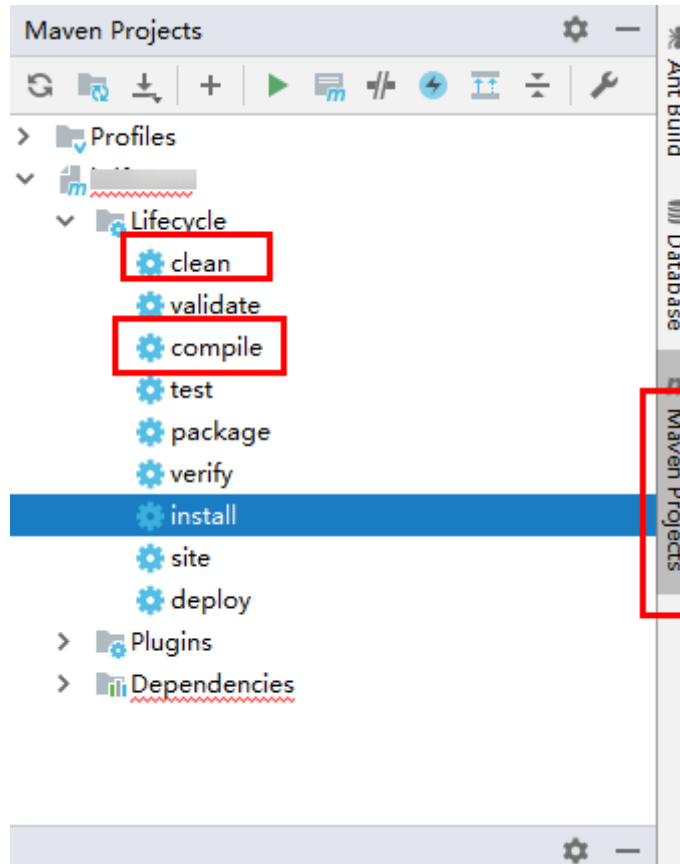
Figure 37-2 Directory for storing the **settings.xml** file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.

4. Click **Apply**, and then click **OK**.
5. On the right of the IntelliJ IDEA home page, click **Maven Projects**. On the **Maven Projects** page, choose *Project name* > **Lifecycle** and run the **clean** and **compile** scripts.

Figure 37-3 Maven Projects page



----End

37.3 Developing an Application

37.3.1 Typical Scenario Description

This section describes the application development in a typical scenario, enabling users to quickly learn and master the FusionInsight Manager REST API development process and know key functions.

Typical Scenario

Users need to work with FusionInsight Manager in non-GUI mode. This way, an HTTP basic authentication-based application is required to provide the following functions:

- Login to FusionInsight Manager.

- Login to FusionInsight Manager for data querying, adding, or deleting.

37.3.2 Development Guideline

Function Decomposition

Decompose the function based on the scenario, as listed in [Table 37-3](#).

Table 37-3 Functions to be developed in FusionInsight Manager

No.	Procedure	Code Implementation
1	Add users.	For details, see Adding Users .
2	Search for users.	For details, see Searching for Users .
3	Modify users.	For details, see Modifying Users .
4	Delete users.	For details, see Deleting Users .
5	Export a user list.	For details, see Exporting a User List .

37.3.3 Example Code Description

37.3.3.1 Login Authentication

Function

Implement basic authentication for login and return the HTTP client after login.

Prerequisites

User cluster information and login account have been configured.

- Configuration file: *Sample project folder*\conf\UserInfo.properties
- Parameters:
 - **userName**: username for logging in to FusionInsight Manager.
 - **password**: password of the username.
 - **webUrl**: address of the Manager home page.

In the following example, **IP_Address** indicates the floating IP address of the cluster.

If you want to use another user to perform operations, you need log in to FusionInsight Manager to create a user.

```
userName= admin
password= adminPassWord
webUrl= https://IP_Address28443/web/
```

Example code

The following provides an example code of invoking the `firstAccess` interface to perform login authentication, which is included in the `main` method of the `rest.UserManager` class.

```
BasicAuthAccess authAccess = new BasicAuthAccess();
HttpClient httpClient = authAccess.loginAndAccess(webUrl, userName, password, userTLSVersion);
LOG.info("Start to access REST API.");
HttpManager httpManager = new HttpManager();
String operationName = "";
String operationUrl = "";
String jsonFilePath = "";
```

37.3.3.2 Adding Users

Function

Access the FusionInsight Manager interface to add users.

Example code

The following provides an example code of adding users, which uses the `main` method of the `rest.UserManager` class.

```
//Access the FusionInsight Manager interface to add users.
operationName = "AddUser";
operationUrl = webUrl + ADD_USER_URL;
jsonFilePath = "./conf/addUser.json";
httpManager.sendHttpRequest(httpClient, operationUrl, jsonFilePath, operationName)
```

37.3.3.3 Searching for Users

Function

Access the FusionInsight Manager interface to search for users.

Example code

The following provides an example code of searching for users, which uses the `main` method of the `rest.UserManager` class.

```
//Access the FusionInsight Manager interface to search the user list.
operationName = "QueryUserList";
operationUrl = webUrl + QUERY_USER_LIST_URL;
String responseLineContent = httpManager.sendHttpRequest(httpClient, operationUrl,
operationName);
LOG.info("The {} response is {}.", operationName, responseLineContent);
```

37.3.3.4 Modifying Users

Function

Access the FusionInsight Manager interface to modify users.

Example code

The following provides an example code of modifying users, which uses the **main** method of the **rest.UserManager** class.

```
//Access the FusionInsight Manager interface to modify users.
operationName = "ModifyUser";
String modifyUserName = "user888";
operationUrl = webUrl + MODIFY_USER_URL + modifyUserName;
jsonFilePath = "./conf/modifyUser.json";
httpManager.sendHttpPutRequest(httpClient, operationUrl, jsonFilePath, operationName);
```

37.3.3.5 Deleting Users

Function

Access the FusionInsight Manager interface to delete users.

Example code

The following provides an example code of deleting users, which uses the **main** method of the **rest.UserManager** class.

```
//Access the FusionInsight Manager interface to delete users.
operationName = "DeleteUser";
String deleteJsonStr = "{\"userNames\": [\"user888\"]}";
operationUrl = webUrl + DELETE_USER_URL;
httpManager.sendHttpDeleteRequest(httpClient, operationUrl, deleteJsonStr, operationName);
LOG.info("Exit main.");
```

37.3.3.6 Exporting a User List

Function

Invoke the export and download interfaces in sequence to export the user list. The output of the export interface is the input of the download interface.

Example code

The following provides an example code of exporting a user list, which uses the **main** method of the **rest.ExportUsers** class.

```
String operationName = "ExportUsers";
String exportOperationUrl = webUrl + EXPORT_URL;
HttpManager httpManager = new HttpManager();
//Invoke the export interface.
String responseLineContent = httpManager
    .sendHttpPostRequestWithString(httpClient, exportOperationUrl, StringUtils.EMPTY, operationName);
//Invoke the download interface.
operationName = "DownloadUsers";
JSONObject jsonObj = JSON.parseObject(responseLineContent);
String downloadOperationUrl = webUrl + DOWNLOAD_URL + jsonObj.getString("fileName");
httpManager.sendHttpGetRequest(httpClient, downloadOperationUrl, operationName);
```

37.4 Application Commissioning

37.4.1 Commissioning an Application in the Windows OS

37.4.1.1 Compiling and Running an Application

Scenario

Run an application in the Windows OS after code development is complete.

NOTE

If the IBM JDK is used in the windows OS, applications cannot be directly run in the windows OS.

Procedure

In a development environment (such as IntelliJ IDEA), choose the following two projects separately and run the projects:

- Choose **UserManager.java** and right-click the project and choose **Run 'userManager.main()'** from the shortcut menu to run the project.
- Choose **ExportUser.java** and right-click the project and choose **Run 'ExportUser.main()'** from the shortcut menu to run the project.

37.4.1.2 Viewing Windows Commissioning Results

Scenario

After a FusionInsight Manager application is run, you can check the running result using either of the following methods:

- View the IntelliJ IDEA running result. Configure log printing information in the **log4j.properties** file in the **/conf** directory.
- Log in to the management node and view the system log **web.log** in the **/var/log/Bigdata/tomcat** directory.

Procedure

- Run the **UserManager** class. If the following log information is displayed, the operation is successful.

```
2020-10-19 14:22:52,111 INFO [main] Enter main. rest.UserManager.main(UserManager.java:43)
2020-10-19 14:22:52,113 INFO [main] Get the web info and user info from file .\conf
\UserInfo.properties rest.UserManager.main(UserManager.java:56)
2020-10-19 14:22:52,113 INFO [main] The user name is : admin.
rest.UserManager.main(UserManager.java:63)
2020-10-19 14:22:52,113 INFO [main] The webUrl is : https://10.112.16.93:28443/web/.
rest.UserManager.main(UserManager.java:75)
2020-10-19 14:22:52,113 INFO [main] Begin to get httpClient and first access.
rest.UserManager.main(UserManager.java:84)
2020-10-19 14:22:52,117 INFO [main] Enter loginAndAccess.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:56)
2020-10-19 14:22:52,120 INFO [main] 1.Get http client for sending https request, username is admin,
webUrl is https://10.112.16.93:28443/web/.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:66)
2020-10-19 14:22:52,121 INFO [main] Enter getHttpClient.
basicAuth.BasicAuthAccess.getHttpClient(BasicAuthAccess.java:98)
2020-10-19 14:22:52,693 INFO [main] Exit getHttpClient.
basicAuth.BasicAuthAccess.getHttpClient(BasicAuthAccess.java:104)
```

```

2020-10-19 14:22:52,693 INFO [main] The new http client is:
org.apache.http.impl.client.DefaultHttpClient@66d2e7d9.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:70)
2020-10-19 14:22:52,693 INFO [main] 2. Construct basic authentication, username is admin.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:76)
2020-10-19 14:22:52,694 INFO [main] the authentication is Basic YWRtaW46QmInZGF0YV8yMDEz .
basicAuth.BasicAuthAccess.constructAuthentication(BasicAuthAccess.java:122)
2020-10-19 14:22:52,695 INFO [main] 3. Send first access request, username is admin.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:86)
2020-10-19 14:22:53,555 INFO [main] First access status is HTTP/1.1 200
basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:162)
2020-10-19 14:22:53,556 INFO [main] Response content is [{"infoType":
1,"infoContent":"","alarmStat":[{"level":"Critical","num":0}, {"level":"Major","num":6},
{"level":"Minor","num":0}, {"level":"Warning","num":0}], "timestamp":
1603088492362, "timezoneOffset":-480}]
basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:168)
2020-10-19 14:22:53,556 INFO [main] User admin first access success
basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:174)
2020-10-19 14:22:53,556 INFO [main] Start to access REST API.
rest.UserManager.main(UserManager.java:88)
2020-10-19 14:22:53,557 INFO [main] Enter sendHttpPostRequest for userOperation AddUser.
basicAuth.HttpManager.sendHttpPostRequest(HttpManager.java:93)
2020-10-19 14:22:53,558 INFO [main] The json content =
{"userName":"user888","userType":"HM","password":"XXX","confirmPassword":"XXX","userGroups":
["supergroup"],"userRoles":[],"primaryGroup":"supergroup","description":"Add user"}.
basicAuth.HttpManager.sendHttpPostRequest(HttpManager.java:148)
2020-10-19 14:22:55,437 INFO [main] The AddUser status is HTTP/1.1 204 .
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:425)
2020-10-19 14:22:55,437 INFO [main] sendHttpPostRequest completely.
basicAuth.HttpManager.sendHttpPostRequest(HttpManager.java:178)
2020-10-19 14:22:55,437 INFO [main] Enter sendHttpGetRequest for userOperation QueryUserList.
basicAuth.HttpManager.sendHttpGetRequest(HttpManager.java:48)
2020-10-19 14:22:55,437 INFO [main] The operationUrl is:https://10.112.16.93:28443/web/api/v2/
permission/users?limit=10&offset=0&filter=&order=ASC&order_by=userName
basicAuth.HttpManager.sendHttpGetRequest(HttpManager.java:60)
2020-10-19 14:22:55,565 INFO [main] The QueryUserList status is HTTP/1.1 200 .
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:425)
2020-10-19 14:22:55,565 INFO [main] The response lineContent is {"users":
[{"userName":"admin","userType":"HM","description":"Administrator of FusionInsight
Manager.,"password":"","createTime":"2020-09-30T10:31:44+08:00","defaultUser":true,"primaryGroup
":"compcommon","locked":false,"userRoles":["Manager_administrator"],"userGroups":
[],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":"MRS_MANAGER
ER_USER","iamCustomPolicyUser":false},
{"userName":"developuser","userType":"MM","description":"","password":"","createTime":"2020-10-15
T19:16:37+08:00","defaultUser":false,"primaryGroup":"elasticsearch","locked":false,"userRoles":
["System_administrator"],"userGroups":
["elasticsearch"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":
"MRS_MANAGER_USER","iamCustomPolicyUser":false},
{"userName":"hue1","userType":"HM","description":"","password":"","createTime":"2020-10-09T17:39:5
7+08:00","defaultUser":false,"primaryGroup":"hive","locked":false,"userRoles":
["System_administrator"],"userGroups":
["hive","hadoop","supergroup"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO
","userSource":"MRS_MANAGER_USER","iamCustomPolicyUser":false},
{"userName":"user888","userType":"HM","description":"Add
user","password":"","createTime":"2020-10-19T14:21:32+08:00","defaultUser":false,"primaryGroup":"su
pergroup","locked":false,"userRoles":[],"userGroups":
["supergroup"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":
"MRS_MANAGER_USER","iamCustomPolicyUser":false},
{"userName":"yangtong","userType":"MM","description":"","password":"","createTime":"2020-10-19T10
:50:52+08:00","defaultUser":false,"primaryGroup":"supergroup","locked":false,"userRoles":
["System_administrator"],"userGroups":
["supergroup"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":
"MRS_MANAGER_USER","iamCustomPolicyUser":false}], "totalCount":5}.
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:430)
2020-10-19 14:22:55,565 INFO [main] SendHttpGetRequest completely.
basicAuth.HttpManager.sendHttpGetRequest(HttpManager.java:69)
2020-10-19 14:22:55,565 INFO [main] The QueryUserList response is {"users":
[{"userName":"admin","userType":"HM","description":"Administrator of FusionInsight
Manager.,"password":"","createTime":"2020-09-30T10:31:44+08:00","defaultUser":true,"primaryGroup

```



```

:"compcommon","locked":false,"userRoles":["Manager_administrator"],"userGroups":
[],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":"MRS_MANAG
ER_USER","iamCustomPolicyUser":false},
{"userName":"developuser","userType":"MM","description":"","password":"","createTime":"2020-10-15
T19:16:37+08:00","defaultUser":false,"primaryGroup":"elasticsearch","locked":false,"userRoles":
["System_administrator"],"userGroups":
["elasticsearch"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":
"MRS_MANAGER_USER","iamCustomPolicyUser":false},
{"userName":"hue1","userType":"HM","description":"","password":"","createTime":"2020-10-09T17:39:5
7+08:00","defaultUser":false,"primaryGroup":"hive","locked":false,"userRoles":
["System_administrator"],"userGroups":
["hive","hadoop","supergroup"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO
","userSource":"MRS_MANAGER_USER","iamCustomPolicyUser":false},
{"userName":"user888","userType":"HM","description":"Add
user","password":"","createTime":"2020-10-19T14:21:32+08:00","defaultUser":false,"primaryGroup":"su
pergroup","locked":false,"userRoles":[],"userGroups":
["supergroup"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":"
MRS_MANAGER_USER","iamCustomPolicyUser":false},
{"userName":"yangtong","userType":"MM","description":"","password":"","createTime":"2020-10-19T10
:50:52+08:00","defaultUser":false,"primaryGroup":"supergroup","locked":false,"userRoles":
["System_administrator"],"userGroups":
["supergroup"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":"
MRS_MANAGER_USER","iamCustomPolicyUser":false}],totalCount":5}.
rest.UserManager.main(UserManager.java:105)
2020-10-19 14:22:55,565 INFO [main] Enter sendHttpRequest for userOperation ModifyUser.
basicAuth.HttpManager.sendHttpRequest(HttpManager.java:239)
2020-10-19 14:22:55,566 INFO [main] The json content =
{"userName":"user888","userType":"HM","password":"XXX","confirmPassword":"XXX","userGroups":
["supergroup"],"primaryGroup":"supergroup","userRoles":
["Manager_administrator"],"description":"Modify user"}.
basicAuth.HttpManager.sendHttpRequest(HttpManager.java:293)
2020-10-19 14:22:56,299 INFO [main] The ModifyUser status is HTTP/1.1 204 .
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:425)
2020-10-19 14:22:56,299 INFO [main] sendHttpRequest completely.
basicAuth.HttpManager.sendHttpRequest(HttpManager.java:304)
2020-10-19 14:22:56,299 INFO [main] The operationUrl is:https://10.112.16.93:28443/web/api/v2/
permission/users basicAuth.HttpManager.sendHttpRequest(HttpManager.java:389)
2020-10-19 14:22:56,299 INFO [main] Enter sendHttpRequest for operation DeleteUser.
basicAuth.HttpManager.sendHttpRequest(HttpManager.java:390)
2020-10-19 14:22:57,463 INFO [main] The DeleteUser status is HTTP/1.1 204 .
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:425)
2020-10-19 14:22:57,463 INFO [main] sendHttpRequest for DeleteUser completely.
basicAuth.HttpManager.sendHttpRequest(HttpManager.java:406)
2020-10-19 14:22:57,463 INFO [main] Exit main. rest.UserManager.main(UserManager.java:120)

```

NOTE

As indicated in the log, when the main method of the UserManager class is executed, the loginAndAccess, sendHttpRequest, sendHttpRequest, sendHttpRequest, and sendHttpRequest methods are invoked in sequence to send the POST, GET, PUT, and DELETE requests for login authentication, adding users, searching for users, modifying users, and deleting users.

- Run the **ExportUsers** class. If the following log information is displayed, the operation is successful.

```

2020-10-19 14:25:02,845 INFO [main] Enter main. rest.ExportUsers.main(ExportUsers.java:41)
2020-10-19 14:25:02,847 INFO [main] Get the web info and user info from file .\conf
\UserInfo.properties rest.ExportUsers.main(ExportUsers.java:54)
2020-10-19 14:25:02,847 INFO [main] The user name is : admin.
rest.ExportUsers.main(ExportUsers.java:61)
2020-10-19 14:25:02,847 INFO [main] The webUrl is : https://10.112.16.93:28443/web/.
rest.ExportUsers.main(ExportUsers.java:73)
2020-10-19 14:25:02,847 INFO [main] Begin to get httpClient and first access.
rest.ExportUsers.main(ExportUsers.java:82)
2020-10-19 14:25:02,851 INFO [main] Enter loginAndAccess.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:56)
2020-10-19 14:25:02,854 INFO [main] 1.Get http client for sending https request, username is admin,
webUrl is https://10.112.16.93:28443/web/.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:66)

```

```

2020-10-19 14:25:02,855 INFO [main] Enter getHttpClient.
basicAuth.BasicAuthAccess.getHttpClient(BasicAuthAccess.java:98)
2020-10-19 14:25:03,414 INFO [main] Exit getHttpClient.
basicAuth.BasicAuthAccess.getHttpClient(BasicAuthAccess.java:104)
2020-10-19 14:25:03,414 INFO [main] The new http client is:
org.apache.http.impl.client.DefaultHttpClient@66d2e7d9.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:70)
2020-10-19 14:25:03,414 INFO [main] 2. Construct basic authentication, username is admin.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:76)
2020-10-19 14:25:03,415 INFO [main] the authentication is Basic YWRtaW46QmlnZGF0YV8yMDEz .
basicAuth.BasicAuthAccess.constructAuthentication(BasicAuthAccess.java:122)
2020-10-19 14:25:03,415 INFO [main] 3. Send first access request, username is admin.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:86)
2020-10-19 14:25:04,205 INFO [main] First access status is HTTP/1.1 200
basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:162)
2020-10-19 14:25:04,206 INFO [main] Response content is [{"infoType":
1,"infoContent":"","alarmStat":{"level":"Critical","num":0},"level":"Major","num":6},
{"level":"Minor","num":0},"level":"Warning","num":0},"timestamp":
1603088623013,"timezoneOffset":-480}]
basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:168)
2020-10-19 14:25:04,206 INFO [main] User admin first access success
basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:174)
2020-10-19 14:25:04,207 INFO [main] Start to access REST API.
rest.ExportUsers.main(ExportUsers.java:86)
2020-10-19 14:25:04,208 INFO [main] Enter sendHttpPostRequest for userOperation ExportUsers.
basicAuth.HttpManager.sendHttpPostRequestWithString(HttpManager.java:193)
2020-10-19 14:25:04,418 INFO [main] The ExportUsers status is HTTP/1.1 200 .
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:425)
2020-10-19 14:25:04,418 INFO [main] The response lineContent is
{"fileName":"userInfo_2020-10-19-14-23-43.zip"}.
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:430)
2020-10-19 14:25:04,418 INFO [main] SendHttpPostRequest completely.
basicAuth.HttpManager.sendHttpPostRequestWithString(HttpManager.java:216)
2020-10-19 14:25:04,464 INFO [main] Enter sendHttpGetRequest for userOperation DownloadUsers.
basicAuth.HttpManager.sendHttpGetRequest(HttpManager.java:48)
2020-10-19 14:25:04,464 INFO [main] The operationUrl is:https://10.112.16.93:28443/web/api/v2/
permission/users/download?file_name=userInfo_2020-10-19-14-23-43.zip
basicAuth.HttpManager.sendHttpGetRequest(HttpManager.java:60)
2020-10-19 14:25:04,508 INFO [main] The DownloadUsers status is HTTP/1.1 200 .
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:425)

```

NOTE

As indicated in the log, when the main method of the ExportUsers class is executed, the loginAndAccess, sendHttpPostRequestWithString, and sendHttpGetRequest methods are invoked in sequence to send the POST and GET requests for login authentication, exporting users, and downloading users.

- You can configure the log printing information in the `conf\log4j.properties` file to view the application running process and result.

The printing information is configured by default. The following provides an example.

```

##set log4j DEBUG < INFO < WARN < ERROR < FATAL
log4j.logger.rest=INFO,A1,A2
log4j.logger.basicAuth=INFO,A1,A2
log4j.logger.org.apache.http=INFO,A1,A2

#print to the console?A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%t] %m %l%n

#log file
log4j.appender.A2=org.apache.log4j.DailyRollingFileAppender
log4j.appender.A2.File=./log/rest.log
log4j.appender.A2.Append = true
log4j.appender.A2.layout=org.apache.log4j.PatternLayout
log4j.appender.A2.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%t] %m %l%n

```

37.5 More Information

37.5.1 External Interfaces

37.5.1.1 Java API

Typical APIs

Table 37-4 describes the typical methods of developing FusionInsight Manager REST APIs.

Table 37-4 restApiDemo.src.rest.BasicAuthAccess

Method	Description
loginAndAccess (String webUrl,String userName,String password,String userTLSVersion)	<ul style="list-style-type: none">• webUrl URL of the cluster home page, which is obtained from the UserInfo.properties configuration file.• userName Username for logging in to the FusionInsight system, which is obtained from the UserInfo.properties configuration file.• password Password of the account userName, which is obtained from the UserInfo.properties configuration file.• userTLSVersion TSL version. Return type: HttpClient Return: HttpClient

NOTE

- The Java API implements basic authentication for login and returns the HttpClient after login. This way, only one API is invoked during the login, simplifying the usage process.
- The API input parameters are obtained from the configuration file **UserInfo.properties**. Users need to set the parameters in the file. The Java API also invokes multiple methods of the BasicAuthAccess class.

Table 37-5 restApiDemo.src.rest.BasicAuthAcces.HttpManager

Method	Description
sendHttpGetRequest(HttpClient httpClient, String operationUrl, String operationName)	<ul style="list-style-type: none"> Parameter type: HttpClient httpClient: result returned after the login authentication is complete. operationUrl URL of the httpGet operation. operationName The operation name.
sendHttpPostRequest(HttpClient httpClient, String operationUrl, String jsonFilePath, String operationName)	<ul style="list-style-type: none"> Parameter type: HttpClient httpClient: result returned after the login authentication is complete. operationUrl URL of the httpPost operation. jsonFilePath JSON file corresponding to the httpPost operation. operationName The operation name.
sendHttpPostRequestWithString(HttpClient httpClient, String operationUrl, String jsonString, String operationName)	<ul style="list-style-type: none"> Parameter type: HttpClient httpClient: result returned after the login authentication is complete. operationUrl URL of the httpPost operation. jsonString JSON string format corresponding to the httpPost operation. operationName The operation name.
sendHttpPutRequest(HttpClient httpClient, String operationUrl, String jsonFilePath, String operationName)	<ul style="list-style-type: none"> Parameter type: HttpClient httpClient: result returned after the login authentication is complete. operationUrl URL of the httpPut operation. jsonFilePath JSON file corresponding to the httpPut operation. operationName The operation name.

Method	Description
sendHttpPutRequestWithString(HttpClient httpClient, String operationUrl, String jsonString, String operationName)	<ul style="list-style-type: none"> Parameter type: HttpClient httpClient: result returned after the login authentication is complete. operationUrl URL of the httpPut operation. jsonString JSON string format corresponding to the httpPut operation. operationName The operation name.
sendHttpDeleteRequest(HttpClient httpClient, String operationUrl, String jsonString, String operationName)	<ul style="list-style-type: none"> Parameter type: HttpClient httpClient: result returned after the login authentication is complete. operationUrl URL of the httpDelete operation. jsonString JSON string format corresponding to the httpDelete operation. operationName The operation name.

 **NOTE**

- APIs in the table are used to send HTTP requests. To invoke the APIs, users only need to provide the URLs corresponding to each type of operations, and JSON files corresponding to the operations or JSON string formats. In this way, no intermediate execution code is required, reducing the code compilation workload and simplifying operation procedures.
- The APIs return the command ID corresponding to the requests. Based on the command IDs, users can query the command execution progress.

37.5.2 FAQ

37.5.2.1 JDK1.6 Fails to Connect to the FusionInsight System Using JDK1.8

Question

The MRS system uses JDK1.8 and supports TLSv1.2 and TLSv1.2. Some users can only use JDK1.6 to connect to the MRS system, and the connection fails.

Answer

Solution: Add an SSL of an earlier version.

- Step 1** Add an SSL of an earlier version, for example, TLSv1, to the **server.xml** file in the **\$ {BIGDATA_HOME}/om-server/tomcat/conf** directory. In the file, there are two contexts to be modified.

First context:

```
<Connector port="20009" address="10.52.0.35"
protocol="com.omm.huawei.tomcat.http11.Http11Protocol" SSLEnabled="true"
maxHttpHeaderSize="8192" maxPostSize="10240"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
enableLookups="false" disableUploadTimeout="true"
acceptCount="100" scheme="https" secure="true"
clientAuth="false" sslProtocol="TLS" sslEnabledProtocols = "TLSv1.1,TLSv1.2,TLSv1"
```

Second context:

```
<Connector port="28443" address="10.52.0.35"
protocol="com.omm.huawei.tomcat.http11.Http11Protocol" SSLEnabled="true"
scheme="https"
secure="true"
maxThreads="500"
acceptCount="500"
connectionTimeout="20000"
maxPostSize="10240"
clientAuth="false"
allowTrace="false"
sslProtocol="TLS" sslEnabledProtocols = "TLSv1.1,TLSv1.2,TLSv1"
```

- Step 2** Set the **userTLSVersion** parameter in the main method of the commissioning source file to be the same as that added in the **server.xml** file, for example, TLSv1.

```
String webUrl = resourceBundle .getString("webUrl");
LOG .info("The webUrl is:{},webUrl);
if(password == null || password .isEmpty())
{
    LOG .error("The password is empty.");
}
//userTLSVersion is mandatory, which is an important parameter used for connecting the JDK1.6
server and the JDK1.8 server. If the JDK1.8 server is used, set this parameter to a null character string.
String userTLSVersion = "TLSv1" ;
//Invoke the firstAccess interface to perform the login authentication.
LOG .info("Begin to get httpclient and access.");
BasicAuthAccess authAccess = new BasicAuthAccess();
HttpClient httpclient = authAccess .firstAccess(webUrl , userName , password , userTLSVersion);
```

- Step 3** The other steps are the same as those for the JDK1.8 server. For details, see [Example Code Description](#).

 **NOTE**

This solution can resolve the connection problem. However, the SSL of earlier versions may cause security risks to the system. Remind users to use JDK1.8.

----End

37.5.2.2 An Operation Fails and "authorize failed" Is Displayed in Logs

Question

An application execution fails. [An Operation Fails and "authorize failed" Is Displayed in Logs](#) shows the log information.

Figure 37-4 Log indicating operation failure

```
2018-10-25 16:08:10,907 INFO [main] 3. Send first access request, username is admin. basicAuth.BasicAuthAccess.loginAndAccess:
2018-10-25 16:08:11,720 INFO [main] First access status is HTTP/1.1 401 basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:130)
2018-10-25 16:08:11,721 INFO [main] Response content is {"state":"FAILED","errorCode":401,"errorDescription":"Login failed.
admin.", "totalProgress":0.0,"id":-1} basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:130)
basicAuth.exception.AuthenticationException: Authorize failed!
    at basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:134)
    at basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:64)
    at rest.UserManager.main(UserManager.java:88)
2018-10-25 16:08:11,723 ERROR [main] Authorize failed! rest.UserManager.main(UserManager.java:137)
```

Answer

The username or password may be incorrect.

Solution: Check whether the values of **username** and **password** in the **UserInfo.properties** file are correct.

37.5.2.3 An Operation Fails and "log4j:WARN No appenders could be found for logger(basicAuth.Main)" Is Displayed in Logs

Question

An application execution fails. Figure 1-6 shows the log information.

Figure 37-5 Log indicating operation failure

```
log4j:WARN No appenders could be found for logger (rest.UserManager).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

Answer

Check whether the compiled **log4j.properties** file exists in the **bin** directory of the project. If it does not, add the compilation path.

Step 1 In IntelliJ IDEA, choose **File > Project Structure > Modules**, and then add the **conf** folder that contains **log4j.properties** and **UserInfo.properties** as **Source Folders**.

Step 2 Compile the file again.

----End

37.5.2.4 An Operation Fails and "illegal character in path at index 57" Is Displayed in Logs

Question

An application execution fails. **Figure 37-6** shows the log information.

Figure 37-6 Log indicating operation failure

```
java.lang.IllegalArgumentException: Illegal character in path at index 57: https://189.120.205.139:28443/web/api/v2/permission/users
    at java.net.URI.create(Unknown Source)
    at org.apache.http.client.methods.HttpPost.<init>(HttpPost.java:76)
    at basicAuth.HttpManager.sendHttpPostRequest(HttpManager.java:156)
    at rest.UserManager.main(UserManager.java:101)
Caused by: java.net.URISyntaxException: Illegal character in path at index 57: https://189.120.205.139:28443/web/api/v2/permission/users
    at java.net.URI$Parser.fail(Unknown Source)
    at java.net.URI$Parser.checkChars(Unknown Source)
    at java.net.URI$Parser.parseHierarchical(Unknown Source)
    at java.net.URI$Parser.parse(Unknown Source)
    at java.net.URI.<init>(Unknown Source)
    ... 4 more
```

Answer

The URL may contain spaces. As a result, the server cannot identify the URL.

Delete the spaces in the URL.

37.5.2.5 Run the curl Command to Access REST APIs

Description

You can use the curl tool to access the Manager REST API. Before running the curl command, run the **openssl version** command to check the OpenSSL version of the current operating system. If the version is earlier than OpenSSL 1.0.1, install the OpenSSL of a later version for the operating system. to support interaction with the cluster using TLSv1.1 and TLSv1.2.

For details about how to install and use curl, see related official documents.

Operation Example

1. Send the GET method to access the user list query interface and save cookie to the `/tmp/jsessionid.txt` file.

```
curl -k -i --basic -u <user name>:<password> -c /tmp/jsessionid.txt 'https://  
Manager floating IP address:28443/web/api/v2/permission/users?  
limit=10&offset=0&filter=&order=ASC&order_by=username'
```

NOTE

`<user name>: <password>` indicates the user name and password that have the Manager operation rights.

2. Use cookie to access the user group interface to add, modify, and delete user groups.

- Send the POST method to add a user.

```
curl -k -i -b /tmp/jsessionid.txt -X POST -HContent-type:application/  
json -d  
'{"userName":"user888","userType":"HM","password":"xxx","confirmP  
assword":"xxx","userGroups":["supergroup"],"userRoles":  
[],"primaryGroup":"supergroup","description":"Add user'}' 'https://  
Manager floating IP address:28443/web/api/v2/permission/users'
```

- Send the PUT method to modify the user information.

```
curl -k -i -b /tmp/jsessionid.txt -X PUT -HContent-type:application/json  
-d  
'{"userName":"user888","userType":"HM","password":"","confirmPass  
word":"","userGroups":  
["supergroup","hadoopmanager"],"primaryGroup":"supergroup","userR  
oles":[],"description":"Modify user'}' 'https://Manager floating IP  
address:28443/web/api/v2/permission/users/user888'
```

- Send a DELETE method to delete a user.

```
curl -k -i -b /tmp/jsessionid.txt -X DELETE -HContent-type:application/  
json -d '{"userNames":["user888']}' 'https://Manager floating IP  
address:28443/web/api/v2/permission/users'
```