MapReduce Service

Application Development Guide

Issue 01

Date 2024-04-02

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property 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 Cloud 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

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).
- You have a basic understanding about how to use Maven.

Methods for Obtaining Sample Projects

Obtaining from Huawei Mirrors

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.

2 Obtaining Sample Projects from Huawei Mirrors

Procedure

Building a sample project includes the following operations:

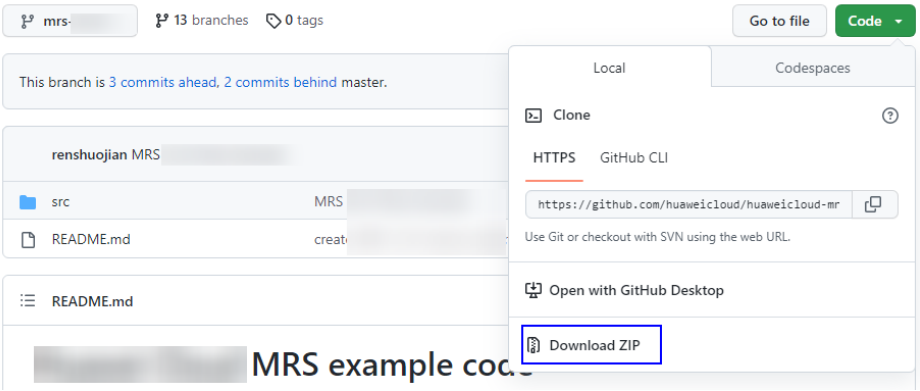
1. Download the Maven project source code and configuration files of the sample project. For details, see **Obtaining Sample Projects**.
2. Configure the Maven mirror repository of the SDK in Huawei Mirrors. For details, see **Configuring Huawei Open Source Mirrors**.
3. Build a complete Maven project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

Obtaining Sample Projects

For MRS 3.1.2-LTS, you can download sample projects at **https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-3.1.2**.

For MRS 3.2.0-LTS.1, you can download sample projects at **https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-3.2.0.1**.

Figure 2-1 Downloading sample code



Switch the branch to the version that matches the MRS cluster,for example mrs-3.2.0.1 Download the package to a local directory, and decompress the package to obtain the sample code project of each component.

NOTE

For details about how to obtain a sample project of the common MRS version, see **Obtaining Sample Projects from Huawei Mirrors**.

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.

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

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

Step 2 Download the **settings.xml** file provided by Huawei Mirrors, and overwrite the *<Maven installation directory>***/conf/settings.xml** file with the downloaded file.

If the file cannot be downloaded, search for **HuaweiCloud SDK** at Huawei Mirrors, click **HuaweiCloud SDK**, and perform operations as prompted.

Step 3 If you do not want to overwrite the Maven configuration file, you can 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**

  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**

  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 4 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 Sample Projects of MRS Components

To obtain the sample projects from **Obtaining Sample Projects from Huawei Mirrors**, **switch the branch to the version that matches the MRS cluster**, download the package to a local directory, and decompress the package to obtain the sample code project of each component.

The MRS sample code library provides sample projects of basic functions of each component. **Table 3-1** lists the projects of the current version.

Table 3-1 Sample projects of each component

Component	Sample Project Location	Description
ClickHouse	clickhouse-examples	Java program that creates and deletes ClickHouse data tables, and inserts and queries data in MRS clusters This program establishes server connections, creates databases and data tables, inserts data, queries data, and deletes data tables.
	ClickHouseJDBC-Transaction-JavaExample	Example code for ClickHouse transactions, which is available for MRS 3.3.0 and later versions.
Doris	doris-examples/doris-jdbc-example	Application development example for Doris data reads/writes, which is available for MRS 3.3.0 and later versions This example calls Doris APIs to create user tables, insert data, query data, and delete tables.

Component	Sample Project Location		Description
Flink	• If Kerberos authentication is enabled for the cluster, the sample project directory is **flink-examples/flink-examples-security**. • If Kerberos authentication is disabled for the cluster, the sample project directory is **flink-examples/flink-examples-normal**.	FlinkCheckpointJavaExample	Java/Scala program for Flink asynchronous checkpointing In this project, the program uses custom operators to continuously generate data. The generated data is a quadruple of long, string, string, and integer values. The program collects statistic results and displays them on the terminal. A checkpoint is triggered every other 6 seconds and the checkpoint result is stored in HDFS.
		FlinkCheckpointScalaExample	
		FlinkHBaseJavaExample	Java sample program that calls Flink APIs in a job to read and write HBase data. This is only supported by MRS 3.2.0 and later versions.
		FlinkKafkaJavaExample	Java/Scala program that uses a Flink job to produce and consume data from Kafka. In this project, assume that a Flink service receives one message per second. The Producer application sends data to Kafka, the Consumer application receives data from Kafka, and the program processes and prints the data.
		FlinkKafkaScalaExample	
		FlinkPipelineJavaExample	Java/Scala program for Flink job pipeline In this example, a publisher job generates 10,000 data records per second, and the other two jobs subscribe to the data, respectively. After receiving the data, the subscriber jobs convert data formats, sample the data, and output the samples.
		FlinkPipelineScalaExample	
		FlinkSqlJavaExample	SQL job submission through Jar jobs on the client

Component	Sample Project Location		Description
		FlinkStreamJavaExample	Java/Scale program for constructing DataStream with Flink This program analyzes user log data based on service requirements, reads text data, generates DataStreams, filters data that meets specified conditions, and obtains results.
		FlinkStreamScalaExample	
		FlinkStreamSqlJoinExample	Flink SQL Join program This program calls APIs of the flink-connector-kafka module to produce and consume data. It generates Table1 and Table2, uses Flink SQL to perform joint query on the tables, and displays results.
		FlinkRESTAPIJavaExample	Java program that calls FlinkServer restful APIs to create tenants
	flink-examples/flink-sql		Sample program that uses Flink Jar to submit a SQL job
	flink-examples/pyflink-example	pyflink-kafka	Python program that submits a regular job to read and write Kafka data
		pyflink-sql	Python program that submits a SQL job
HBase	hbase-examples	hbase-example	Application development example for HBase reads/writes and global secondary indexes. HBase APIs can be called to: • Create user tables, import user data, add and query user information, and create secondary indexes for user tables. • In MRS 3.3.0 and later versions, create and delete global secondary indexes, modify the status of global secondary indexes, and query global secondary indexes.

Component	Sample Project Location		Description
		hbase-rest-example	A development example for using HBase REST interfaces. This program uses REST APIs to query HBase cluster information, obtain tables, use NameSpaces, and manipulate tables.
		hbase-thrift-example	A development example for accessing HBase ThriftServer. This program accesses ThriftServer to manipulate tables, and write data to and read data from tables.
		hbase-zk-example	A development example for HBase to access ZooKeeper. You can use the same client process to access MRS ZooKeeper and third-party ZooKeeper at the same time. The HBase client accesses MRS ZooKeeper, and the customer application accesses third-party ZooKeeper.
HDFS	• If Kerberos authentication is enabled for the cluster, the sample project directory is **hdfs-example-security**. • If Kerberos authentication is disabled for the cluster, the sample project directory is **hdfs-example-normal**.		Java program for HDFS file operations. This program creates HDFS folders, writs files, appends file content, reads files, and deletes files or folders.
	hdfs-c-example		A C language development example for using HDFS. This program connects the HDFS file system and implements file operation functions, such as creating, reading, writing, appending, and deleting files.

Component	Sample Project Location		Description
HetuEngine	• If Kerberos authentication is enabled for the cluster, the sample project directory is **hetu-examples/hetu-examples-security**. • If Kerberos authentication is disabled for the cluster, the sample project directory is **hetu-examples/hetu-examples-normal**.		Java/Python program for connecting to HetuEngine in different ways In this example project, you can use the username and password to connect to HetuEngine through ZooKeeper or HSBroker, or use the KeyTab authentication file to connect to HetuEngine, and send SQL statements to HetuEngine to add, delete, modify, and query Hive data.
Hive	hive-examples	hive-jdbc-example	Java program for Hive JDBC to process data In this project, JDBC APIs are used to connect Hive and perform data operations. You can use JDBC APIs to create tables, load data, and query data. You can access FusionInsight ZooKeeper and third-party ZooKeeper in the same client process at the same time.
		hive-jdbc-example-multizk	
		hcatalog-example	Java program for Hive HCatalog to process data HCatalog APIs are used to define and query MRS Hive metadata with Hive CLI.
		python-examples	Python program to connect to Hive and execute SQL examples. This program uses Python to connect Hive and submits data analysis tasks.
		python3-examples	Python 3 program to connect Hive and execute SQL statements. This program uses Python 3 to connect Hive and submits data analysis tasks.

Component	Sample Project Location		Description
IoTDB	iotdb-examples	iotdb-flink-example	Program for using Flink to access IoTDB data, including FlinkIoTDBSink and FlinkIoTDBSource data. FlinkIoTDBSink can use Flink jobs to write time series data to IoTDB. FlinkIoTDBSource reads time series data from IoTDB through Flink jobs and prints the data.
		iotdb-jdbc-example	Java sample program for IoTDB JDBC to process data. This program demonstrates how to use JDBC APIs to connect IoTDB, and executes IoTDB SQL statements.
		iotdb-kafka-example	Sample program for accessing IoTDB data through Kafka. This program demonstrates how to send time series data to Kafka and then use multiple threads to write the data to IoTDB.
		iotdb-session-example	Java sample program for IoTDB Session to process data. This program demonstrates how to use Session to connect IoTDB, and executes IoTDB SQL statements.
		iotdb-udf-exmaple	This program demonstrates how to implement a simple IoTDB user-defined function (UDF).
Kafka	kafka-examples		Java program for processing Kafka streaming data The program is developed based on Kafka Streams to count words in each message by reading messages in the input topic and to output the result in key-value pairs by consuming data in the output topic.
Manager	manager-examples		Program for calling FusionInsight Manager APIs This program calls Manager APIs to create, modify, and delete cluster users.

Component	Sample Project Location		Description
MapReduce	• If Kerberos authentication is enabled for the cluster, the sample project directory is **mapreduce-example-security**. • If Kerberos authentication is disabled for the cluster, the sample project directory is **mapreduce-example-normal**.		Java program for submitting MapReduce jobs This program runs a MapReduce statistics data job to analyze and process data and output data required by users. It illustrates how to write MapReduce jobs to access multiple service components in HDFS, HBase, and Hive, helping you to develop for key operations such as authentication and configuration loading.
Oozie	• If Kerberos authentication is enabled for the cluster, the sample project directory is **oozie-examples/ooziesecurity-examples**. • If Kerberos authentication is disabled for the cluster, the sample project directory is **oozie-examples/oozienormal-examples**.	OozieMapReduceExample	Program for submitting MapReduce jobs with Oozie. This program demonstrates how to use Java APIs to submit MapReduce jobs, query job status, and perform offline analysis on website log files.
		OozieSparkHBaseExample	Program for using Oozie to schedule Spark jobs to access HBase.
		OozieSparkHiveExample	Program for using Oozie to schedule Spark jobs to access Hive.

Component	Sample Project Location		Description
Spark	• If Kerberos authentication is enabled for the cluster, the sample project directory is **spark-examples/sparksecurity-examples**. • If Kerberos authentication is disabled for the cluster, the sample project directory is **spark-examples/sparknormal-examples**.	SparkHbasetoCarbonJavaExample	Java program for Spark to synchronize HBase data to CarbonData. In this project, the program writes data to HBase in real time for point queries. Data is synchronized to CarbonData tables in batches at a specified interval for analytical queries.
		SparkHbasetoHbaseJavaExample	Java/Scala/Python program that uses Spark to read data from and then write data to HBase The program uses Spark jobs to analyze and summarize data of two HBase tables.
		SparkHbasetoHbasePythonExample	
		SparkHbasetoHbaseScalaExample	
		SparkHivetoHbaseJavaExample	Java/Scala/Python program that uses Spark to read data from Hive and then write data to HBase The program uses Spark jobs to analyze and summarize data of a Hive table and write result to an HBase table.
		SparkHivetoHbasePythonExample	
		SparkHivetoHbaseScalaExample	
		SparkJavaExample	Java/Python/Scala/R program of Spark Core tasks The program reads text data from HDFS and then calculates and analyzes the data. SparkRExample is only available for clusters with Kerberos authentication enabled.
		SparkPythonExample	
		SparkScalaExample	
		SparkRExample	
		SparkLauncherJavaExample	Java/Scala program that uses Spark Launcher to submit jobs The program uses the **org.apache.spark.launcher.SparkLauncher** class through Java/Scala commands to submit Spark jobs.
		SparkLauncherScalaExample	

Component	Sample Project Location		Description
		SparkOnHbaseJavaExample	Java/Scala/Python program in the Spark on HBase scenario The program uses HBase as data sources. In this project, data is stored in HBase in Avro format. Data is read from HBase, and the read data is filtered.
		SparkOnHbasePythonExample	
		SparkOnHbaseScalaExample	
		SparkOnHudiJavaExample	Java/Scala/Python program in the Spark on Hudi scenario The program uses Spark jobs to perform operations such as insertion, query, update, incremental query, query at a specific time, and data deletion on Hudi.
		SparkOnHudiPythonExample	
		SparkOnHudiScalaExample	
		SparkOnMultiHbaseScalaExample	Scala program that uses Spark to access HBase in two clusters at the same time This program is only available for clusters with Kerberos authentication enabled.
		SparkSQLJavaExample	Java/Python/Scala program of Spark SQL tasks The program reads text data from HDFS and then calculates and analyzes the data.
		SparkSQLPythonExample	
		SparkSQLScalaExample	
		SparkStreamingKafka010JavaExample	Java/Scala program used by Spark Streaming to receive data from Kafka and perform statistical analysis The program accumulates and calculates the stream data in Kafka in real time and calculates the total number of records of each word.
		SparkStreamingKafka010ScalaExample	
		SparkStreamingtoHbaseJavaExample010	Java/Scala/Python sample project used by Spark Streaming to read Kafka data and write the data into HBase The program starts a task every 5 seconds to read data from Kafka and updates the data to a specified HBase table.
		SparkStreamingtoHbasePythonExample010	

Component	Sample Project Location		Description
		SparkStreamingtoHbaseScalaExample010	
		SparkStructuredStreamingJavaExample	The program uses Structured Streaming in Spark jobs to call Kafka APIs to obtain word records. Word records are classified to obtain the number of records of each word.
		SparkStructuredStreamingPythonExample	
		SparkStructuredStreamingScalaExample	
		SparkThriftServerJavaExample	Java/Scala program for Spark SQL access through JDBC. In this sample, a custom JDBCServer client and JDBC connections are used to create, load data to, query, and delete tables.
		SparkThriftServerScalaExample	
		StructuredStreamingADScalaExample	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.
		StructuredStreamingStateScalaExample	This Spark structured streaming program collects statistics on the number of events in each session and the start and end timestamp of the sessions in different batches, and outputs the sessions that the state is updated in this batch.
Spring Boot (This component is available only in MRS 3.3.0 or later.)	clickhouse-examples	clickhouse-rest-client-example	An application development example for connecting SpringBoot to ClickHouse. This program establishes server connections, creates databases and data tables, inserts data, queries data, and deletes data tables
	doris-examples	doris-rest-client-example	SpringBoot development example for Doris data read and write This example shows you how to connect SpringBoot to Doris.

Component	Sample Project Location		Description
	flink-examples	flink-dws-read-example	Application development example for connecting GaussDB(DWS) to Flink using SpringBoot.
		flink-dws-sink-example	
	hbase-examples		Application development example for connecting SpringBoot to Phoenix. This example shows you how to connect SpringBoot to HBase and Phoenix.
	hive-examples	hive-rest-client-example	Application development example for connecting SpringBoot to Hive. This example uses SpringBoot to connect Hive to create tables, load data, query data, and delete tables in Hive.
	kafka-examples		Application development example for connecting SpringBoot to Kafka for topic production and consumption.

4 Quick Start for Component Application Development

MRS provides component-based application development sample projects. After creating an MRS cluster, you can obtain and import the sample projects, and compile and commission the sample projects locally.

- **HBase Application Development** allows you to create HBase tables, insert data, create indexes, and delete tables.
- **HDFS Application Development** allows you to create HDFS directories, and write, read, and delete files.
- **Hive JDBC Application Development** enables you to create tables, insert data, and read data in Hive after connecting to Hive using JDBC.
- **Hive HCatalog Application Development** enables you to create tables, insert data, and read data in Hive after connecting to Hive using HCatalog.
- **Kafka Application Development** enables you to implement processing of streaming data.
- **Flink Application Development** enables you to implement Flink DataStream to process data.
- **ClickHouse Application Development** enables you to create and delete ClickHouse tables, and insert and query data.
- **Spark Application Development** enables you to read data from Hive tables and re-write the data to HBase tables.

5 Using Open-source JAR File Conflict Lists

5.1 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.

5.2 HDFS

HDFS JAR File Conflict List

JAR File	Description	Solution
hadoop-plugins-*.jar	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**. The default HDFS LZC compression format class is **io.compression.codec.lzc.class=com.huawei.hadoop.datasight.io.compress.lzc.ZCodec**.	• Method 1: Add the following configuration to the **pom.xml** file in the sample code: <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> • Method 2: 1. 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. 2. Do not use the LZC compression format.

5.3 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.

5.4 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 **/opt/Bigdata/client/Spark2x/spark/jars**.

6 Mapping Between Maven Repository JAR Versions and MRS Component Versions

Mapping Between Maven Repository JAR Versions and MRS Component Versions in an MRS 3.1.2-LTS.3 Cluster

Table 6-1 Mapping between Maven repository JAR versions and MRS component versions in an MRS 3.1.2 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
Spark	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	-
ClickHouse	21.3.4.25	0.3.0
scala	2.11	-

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

Table 6-2 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
Solr	8.4.0	8.4.0-hw-ei-302002
Phoenix	5.0.0	5.0.0-HBase-2.0-hw-ei-302002
ES	7.6.0	7.6.0-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 6-3 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 6-4 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 6-5 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 6-6 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	-

7 Security Authentication

7.1 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 7-1** and module description in **Table 7-1**.

Figure 7-1 Kerberos architecture

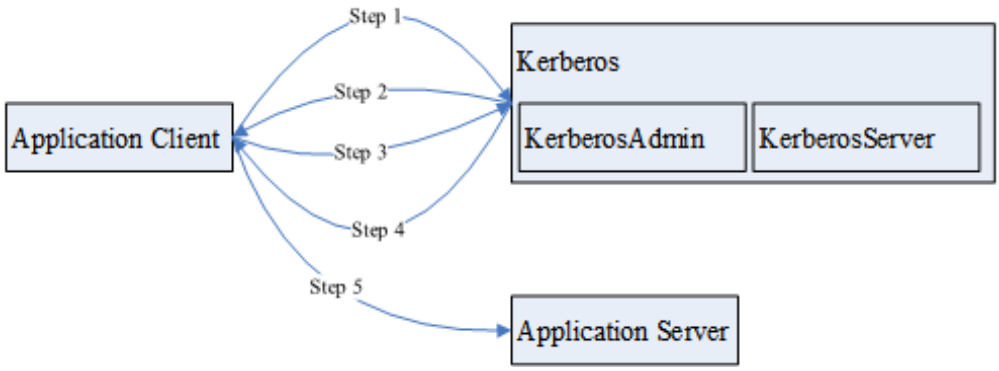


Table 7-1 Module description

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(PRNCIPAL_NAME, PATH_TO_KEYTAB);
}
```

2. Run shell commands on the client in kinit mode.
   a. Log in to the client node and go to the client installation directory.
      cd *{Client installation directory}*
   b. Run the following command to configure environment variables:
      source bigdata_env
   c. Run the following command to authenticate the current user:
      kinit *MRS cluster user*

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.huawei.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 PRNCIPAL_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(PRNCIPAL_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)**.

7.2 Preparing the 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 7-2**.

Table 7-2 List of permissions

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

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

Service	Permissions to Be Granted
Spark2x	1. (Configure this parameter if HBase is installed.) In **Configure Resource Permission**, choose *Name of the desired cluster* > **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 *Name of the desired cluster* > **HBase** > **HBase Scope > global > hbase**, select **execute** for **hbase:meta**, and click **OK**. 3. Edit the role. In **Configure Resource Permission**, choose *Name of the desired cluster* > **HDFS** > **File System > hdfs://hacluster/ > user**, select **Execute** for hive, and click **OK**. 4. Edit the role. In **Configure Resource Permission**, choose *Name of the desired cluster* > **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 *Name of the desired cluster* > **Hive** > **Hive Read Write Privileges**, select the default option **Create**, and click **OK**. 6. Edit the role. In **Configure Resource Permission**, choose *Name of the desired cluster* > **Yarn** > **Scheduler Queue** > **root**, select the default option **Submit**, and click **OK**.
Hive	In **Configure Resource Permission**, choose *Name of the desired cluster* > **Yarn** > **Scheduler Queue** > **root**, select the default option **Submit** and **Admin**, and click **OK**. **NOTE** Extra operation permissions required for Hive application development must be obtained from the system administrator.
ClickHouse	In **Configure Resource Permission**, choose *Name of the desired cluster* > **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**.

Service	Permissions to Be Granted
Flink	1. In the **Configure Resource Permission** table, choose *Name of the desired cluster* > **HDFS > File System > hdfs://hacluster/ > flink**, select **Read**, **Write**, and **Execute**, and click **Service** in the **Configure Resource Permission** table to return. 2. In **Configure Resource Permission**, choose ***Name of the desired cluster* > Yarn > Scheduler Queue > root**, select the default option **Submit**, and click **OK**. **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.
Kafka	-
Impala	-
Oozie	1. In **Configure Resource Permission**, choose *Name of the desired cluster* > **Oozie** > **Common User Privileges**, and click **OK**. 2. Edit the role. In **Configure Resource Permission**, choose *Name of the desired cluster* > **HDFS** > **File System**, select **Read**, **Write**, and **Execute** for hdfs://hacluster, and click **OK**. 3. Edit the role. In **Configure Resource Permission**, choose *Name of the desired cluster* > **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 7-3**, bind the role **developrole** obtain permissions, and click **OK**.

Table 7-3 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-Machine**	Join the **kafkaadmin** group.
Impala	**Machine-Machine**	Join the **impala** and **supergroup** group. Set the primary group to **supergroup**.
Storm/CQL	**Human-Machine**	Join the **storm** group.
ClickHouse	**Human-Machine**	Join the **developgroup** and **supergroup** groups. Set the primary group to **supergroup**.
Oozie	**Human-Machine**	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**.
Flink	**Human-Machine**	Join the **developgroup** and **hadoop** groups. Set the primary group to **developgroup**. **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.

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

7.3 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

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 shows the application development process and **Table 8-1** describes each phase in the process.

Figure 8-1 ClickHouse application development process

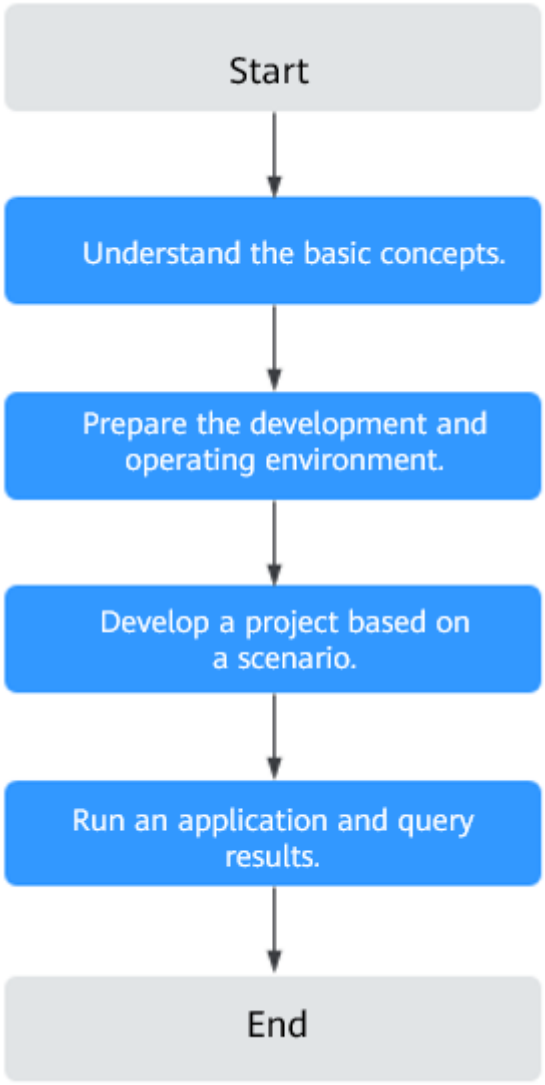


Table 8-1 Description of the ClickHouse application development process

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)	• Development environment: Windows 7 or later. • Operating environment: Linux If the program needs to be commissioned locally, the operating environment must be able to communicate with network on the cluster service plane.

Item	Description
JDK installation	Basic configurations of the development and operating environments. The version requirements are as follows: The server and client support only built-in OpenJDK 1.8.0_272. Customers' applications that need to reference the JAR files of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK. ● x86 client: Oracle JDK 1.8; IBM JDK 1.8.5.11 ● TaiShan client: OpenJDK 1.8.0_272 **NOTE** For security purposes, the server supports only TLS V1.2 or later. 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**.
Installing and configuring IntelliJ IDEA	Basic configuration of the development environment. You are advised to use version 2019.1 or other compatible versions. **NOTE** ● 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	Basic configuration of the development environment. This operation is used for project management throughout the lifecycle of software development.
Developer account preparation	Prepare a cluster user for application development and grant permissions to the user. For details, see **Preparing the Developer Account**.
7-zip	This tool is used to decompress ***.zip** and ***.rar** files. **7-Zip 16.04** is supported.

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. **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**. 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 on the node. For example, install the client in the **/opt/client** directory.

     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

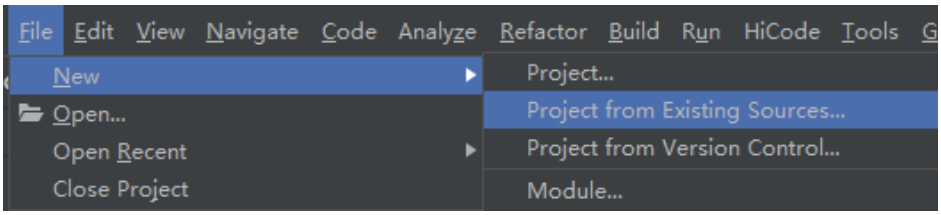
ClickHouse provides sample projects for multiple scenarios to help you quickly learn ClickHouse projects.

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=
# Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or environment variables.
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, choose **Cluster** > **Services** > **ClickHouse** > **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. Log in to FusionInsight Manager, choose **Cluster** > **Services** > **ClickHouse** > **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, choose **Cluster** > **Services** > **ClickHouse** > **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	No default value	Developer user created in **Table 8-2**.

Parameter	Default Value	Definition
password	No default value	Password of the development user. Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or environment variables. **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

Although ClickHouse has the cluster capability, it does not have a unified access entry. The client needs to directly detect all nodes in the cluster. This is not easy to use. 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.

8.3.2 Development Guideline

Development Guideline

As an independent DBMS system, ClickHouse allows you to use the SQL language to perform common operations. The clickhouse-jdbc API is used in the following development program examples:

- Setting attributes
- Establishing a connection
- Creating a database
- Creating a table
- Inserting data
- Querying data
- Deleting a table

8.4 Sample Code

8.4.1 Setting Attributes

Set connection properties in the examples of creating connections in the **ClickhouseJDBCHaDemo**, **Demo**, **NativeJDBCHaDemo**, and **Util** files. The following code sets the socket timeout interval to 60 seconds:

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

If **sslUsed** in the **clickhouse-example.properties** configuration file in **Configuring and Importing a Sample Project** is set to **true**, set the following connection properties in the examples for creating connections in the **ClickhouseJDBCHaDemo**, **Demo**, **NativeJDBCHaDemo**, and **Util** files:

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

8.4.2 Establishing a Connection

The following code snippets are provided in the **initConnection** method of the **ClickhouseJDBCHaDemo** class. 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.

```
clickHouseProperties.setPassword(userPass);
clickHouseProperties.setUser(userName);
BalancedClickhouseDataSource balancedClickhouseDataSource = new
BalancedClickhouseDataSource(JDBC_PREFIX + UriList, clickHouseProperties);
```

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**.

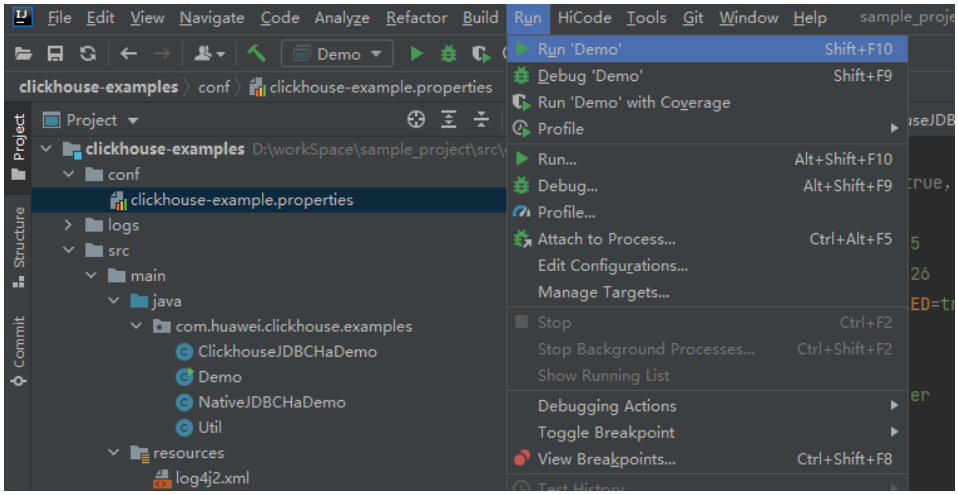
```
private void dropTable(String databaseName, String tableName, String clusterName) throws Exception {
    String dropLocalTableSql = "drop table if exists " + databaseName + "." + tableName + " on cluster " + clusterName;
    String dropDisTableSql = "drop table if exists " + databaseName + "." + 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 100.120.147.36, loadBalancerHttpPort is
21425, user is luxx, clusterName is default_cluster, isSec is true, password is xxxx. |
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
100.120.147.36:21425 | 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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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.

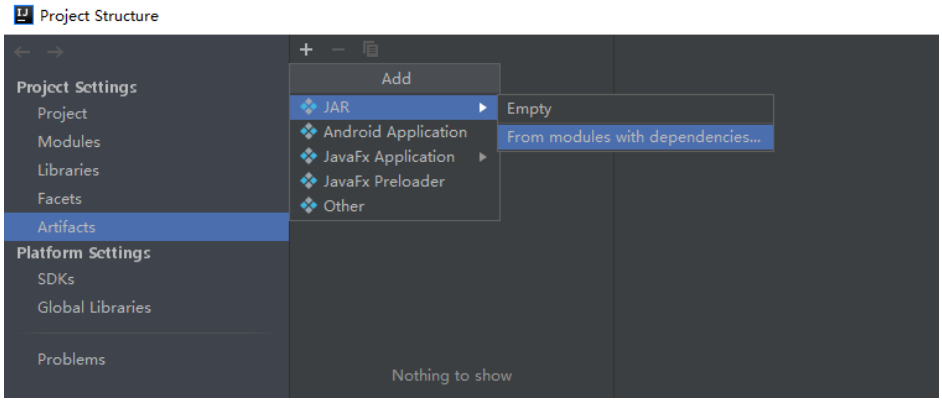
Prerequisite

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.

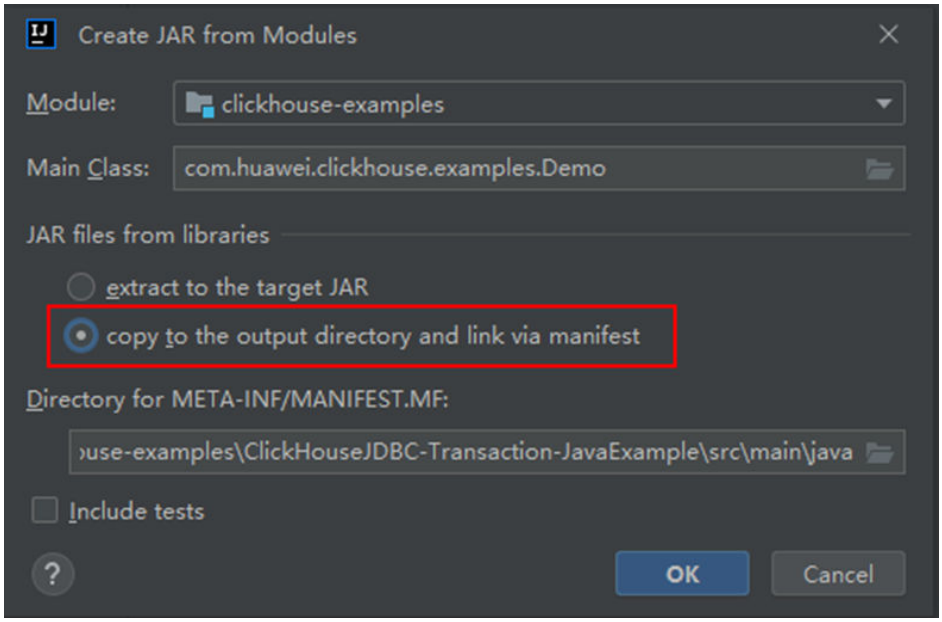
Compile and run applications.

Step 1 Export the JAR file.

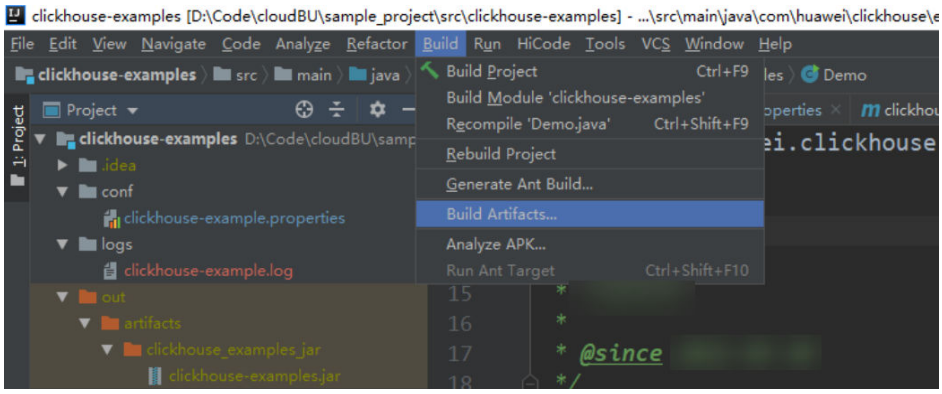
1. Log in to IntelliJ IDEA and choose **File > Project Structure > Artifacts**.
2. Click the plus sign (+) and choose **JAR > From modules with dependencies**.



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 all JAR files in the **clickhouse-examples\out\artifacts\clickhouse_examples_jar** directory.



Step 2 Copy all JAR files in the **clickhouse-examples\out\artifacts\clickhouse_examples.jar** directory and the **conf** folder in the **clickhouse-examples** directory to the ClickHouse client installation directory, for example, *Client installation directory*/**JDBC**.

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/Bigdata/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 *Client installation directory*/**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, *Client installation directory*/**JDBC/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 shows the application development process and **Table 9-1** describes each phase in the process.

Figure 9-1 ClickHouse application development process

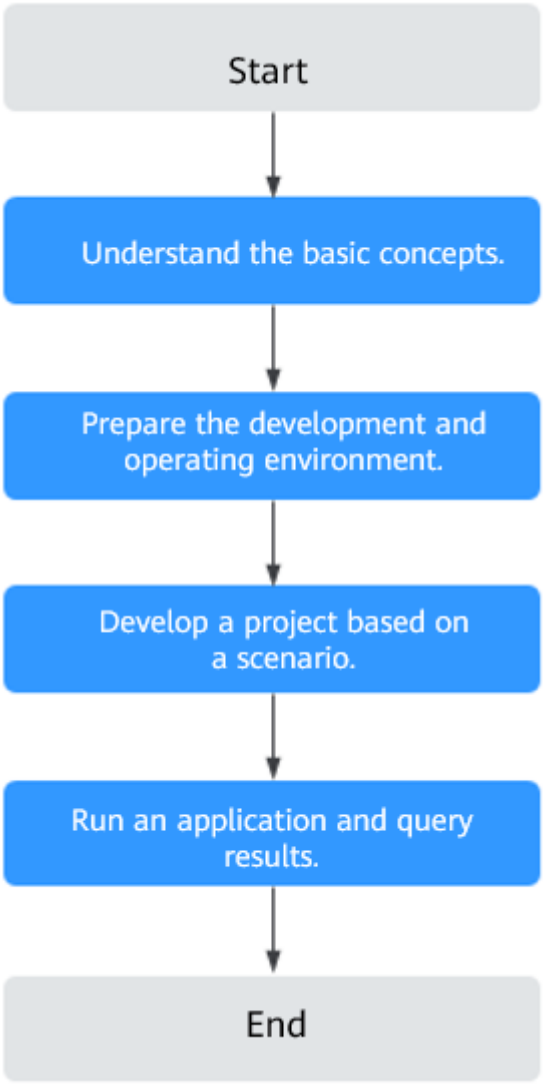


Table 9-1 Description of the ClickHouse application development process

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)	• Development environment: Windows 7 or later. • Operating environment: Linux If the program needs to be commissioned locally, the operating environment must be able to communicate with network on the cluster service plane.

Item	Description
JDK installation	Basic configurations of the development and operating environments. The version requirements are as follows: The server and client support only built-in OpenJDK 1.8.0_272. Customers' applications that need to reference the JAR files of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK. ● x86 client: Oracle JDK 1.8; IBM JDK 1.8.5.11 ● TaiShan client: OpenJDK 1.8.0_272 **NOTE** For security purposes, the server supports only TLS V1.2 or later. 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-sslcontextgetinstancetls-oracle#matchsslcontext_tls**.
Installing and configuring IntelliJ IDEA	Basic configuration of the development environment. You are advised to use version 2019.1 or other compatible versions. **NOTE** ● 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	Basic configuration of the development environment. This operation is used for project management throughout the lifecycle of software development.
Developer account preparation	Prepare the ClickHouse cluster user for application development and grant permissions to the user.
7-zip	This tool is used to decompress ***.zip** and ***.rar** files. **7-Zip 16.04** is supported.

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. **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**. 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 on the node. For example, install the client in the **/opt/client** directory.

     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

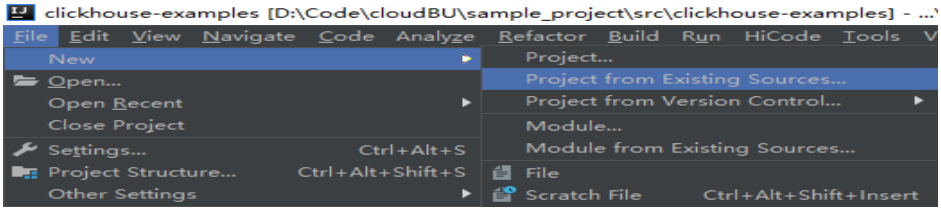
ClickHouse provides sample projects for multiple scenarios to help you quickly learn ClickHouse projects.

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=
# Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or environment variables.
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, choose **Cluster** > **Services** > **ClickHouse** > **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. Log in to FusionInsight Manager, choose **Cluster** > **Services** > **ClickHouse** > **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, choose **Cluster** > **Services** > **ClickHouse** > **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. Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or environment variables. **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

Although ClickHouse has the cluster capability, it does not have a unified access entry. The client needs to directly detect all nodes in the cluster. This is not easy to use. 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.

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.

- Setting attributes
- Establishing a connection
- Creating a database
- Creating a table
- Inserting data
- Querying data
- Deleting a table

9.4 Sample Code

9.4.1 Setting Attributes

Set connection properties in the examples of creating connections in the **ClickhouseJDBCHaDemo**, **Demo**, **NativeJDBCHaDemo**, and **Util** files. The following code sets the socket timeout interval to 60 seconds:

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

If **sslUsed** in the **clickhouse-example.properties** configuration file in **Configuring and Importing a Sample Project** is set to **true**, set the following connection properties in the examples for creating connections in the **ClickhouseJDBCHaDemo**, **Demo**, **NativeJDBCHaDemo**, and **Util** files:

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

9.4.2 Establishing a Connection

The following code snippets are provided in the **initConnection** method of the **ClickhouseJDBCHaDemo** class. 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.

```
clickHouseProperties.setPassword(userPass);
clickHouseProperties.setUser(userName);
BalancedClickhouseDataSource balancedClickhouseDataSource = new
BalancedClickhouseDataSource(JDBC_PREFIX + UriList, clickHouseProperties);
```

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**.

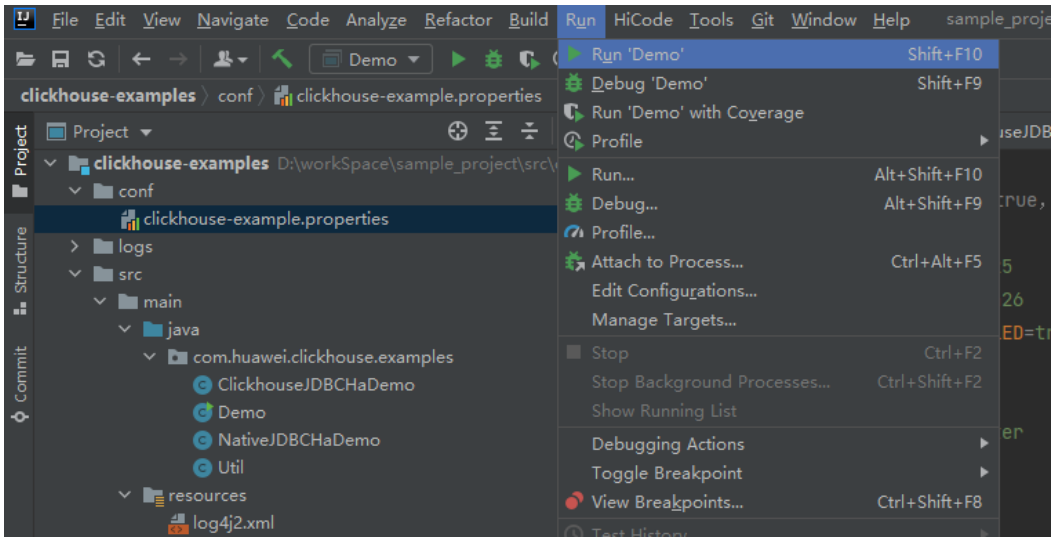
```
private void dropTable(String databaseName, String tableName, String clusterName) throws Exception {
    String dropLocalTableSql = "drop table if exists " + databaseName + "." + tableName + " on cluster " + clusterName;
    String dropDisTableSql = "drop table if exists " + databaseName + "." + 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 21:02:16,900 | INFO  | main | loadBalancerIPList is 100.120.147.36, loadBalancerHttpPort is
21425, user is luxx, clusterName is default_cluster, isSec is true, password is xxxx. |
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
100.120.147.36:21425 | 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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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 100.120.147.36:21425 |
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 package to the prepared Linux environment.

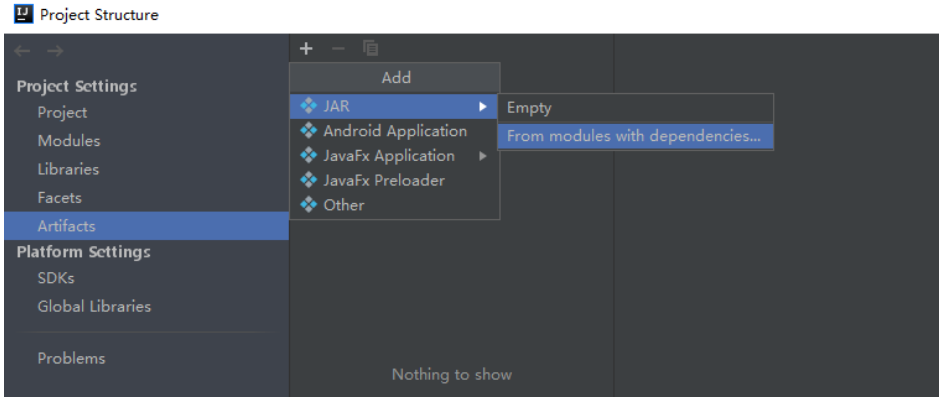
Prerequisite

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.

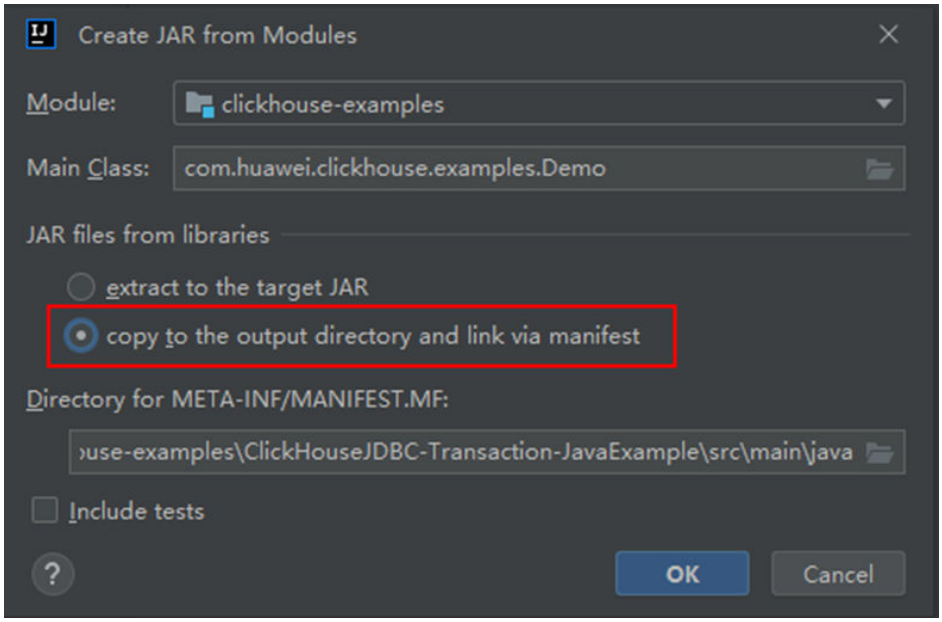
Compile and run applications.

Step 1 Export the JAR package.

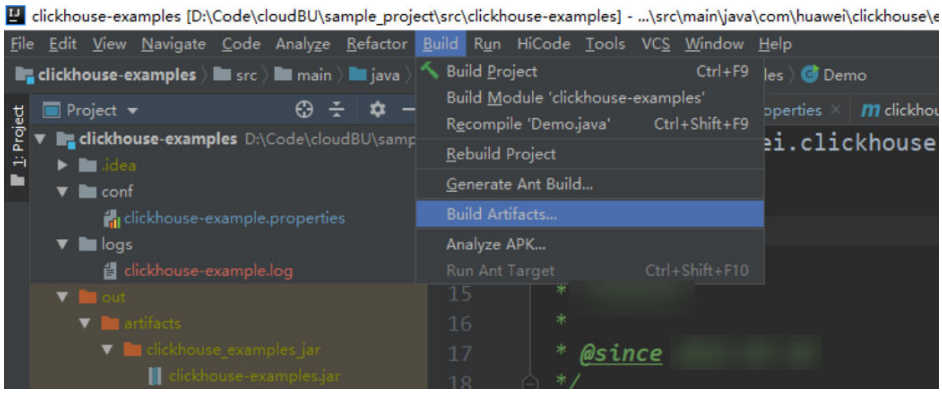
1. Log in to IntelliJ IDEA and choose **File > Project Structure > Artifacts**.
2. Click the plus sign (+) and choose **JAR > From modules with dependencies**.



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 all JAR files in the **clickhouse-examples\out\artifacts** directory.



Step 2 Copy all JAR files in the **clickhouse-examples\out\artifacts\clickhouse_examples.jar** directory and the **conf** folder in the **clickhouse-examples** directory to the ClickHouse client installation directory, for example, *Client installation directory*/**JDBC**.

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/Bigdata/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 package:

source *Client installation directory*/**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, *Client installation directory*/**JDBC/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

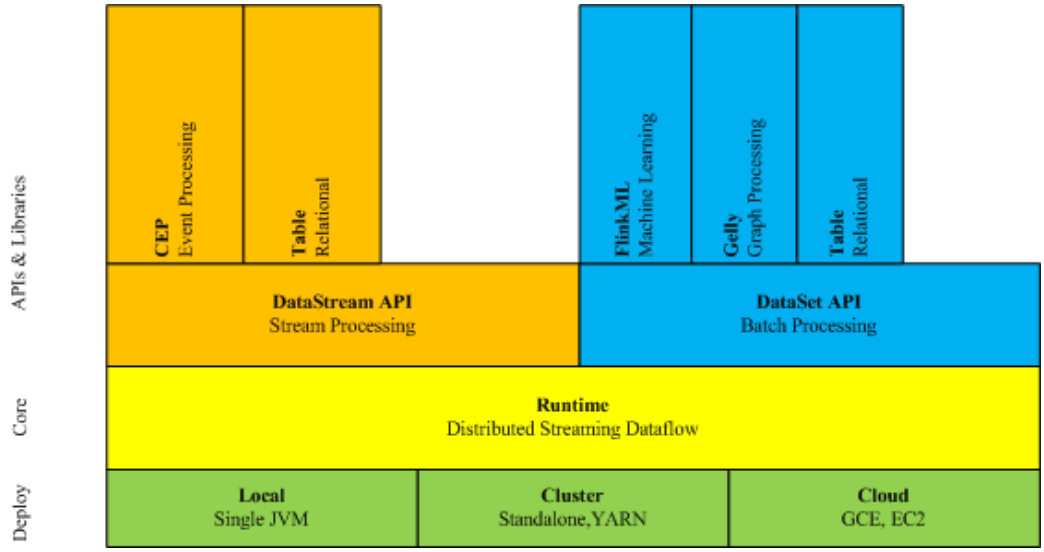
Overview

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



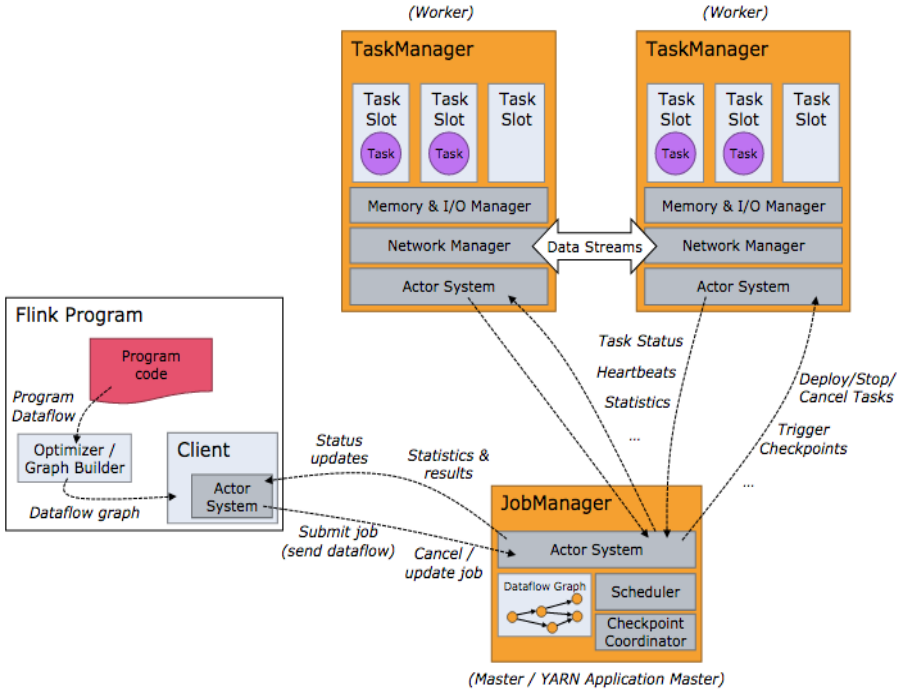
The following lists the key features of Flink in the current version:

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

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

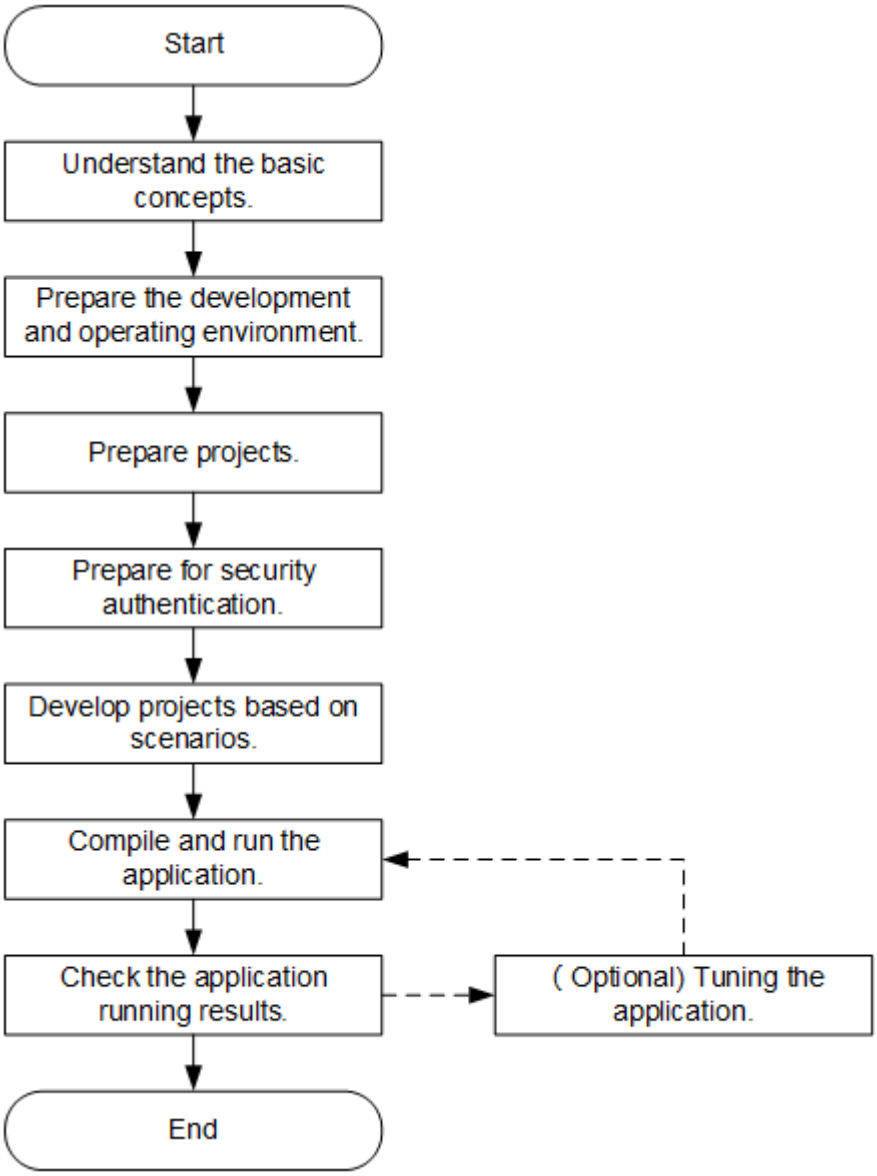


Table 10-2 Description of the development process

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	• 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	Basic configuration of the development and running environment. The version requirements are as follows: The server and client support only the built-in OpenJDK. Other JDKs cannot be used. If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows: • For x86 nodes that run clients, use the following JDKs: – Oracle JDK 1.8 – IBM JDK 1.8.0.7.20 and 1.8.0.6.15 • For Arm nodes that run clients, use the following JDKs: – OpenJDK 1.8.0_272 (built-in JDK, which can be obtained from the **JDK** folder in the cluster client installation directory.) – BiSheng JDK 1.8.0_272 **NOTE** • For security purposes, the server supports only TLS V1.2 or later. • 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, V1.1, and V1.2. For details, see **https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls**. • For details about the BiSheng JDK, see **https://www.hikunpeng.com/en/developer/devkit/compiler/jdk**.
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.

Item	Description
Developer account preparation	See **Preparing the Developer Account** for configuration.
7-zip	It is a tool used to decompress **.zip** and **.rar** packages. The 7-Zip 16.04 is supported.
Python3	Used to run Flink Python jobs. Python 3.6 or later is required.

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 the 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.
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.

d. (Optional) To run a Python job, perform the following additional configurations:

i. Log in to the node where the Flink client is installed as the **root** user and run the following command to check whether Python 3.6 or a later has been installed:

python3 -V

ii. Go to the python 3 installation path, for example, **/srv/pyflink-example**, and install the **virtualenv**:

cd /srv/pyflink-example

virtualenv venv --python=python3.x

source venv/bin/activate

iii. Copy the **Flink/flink/opt/python/apache-flink-*.tar.gz** file from the client installation directory to **/srv/pyflink-example**:

cp /*Client installation directory*/Flink/flink/opt/python/apache-flink-*.tar.gz /srv/pyflink-example

iv. Install the dependency package. If the following command output is displayed, the installation is successful:

python -m pip install apache-flink-libraries-*.tar.gz

python -m pip install apache-flink-*Version number***.tar.gz**


```
...
Successfully built apache-flink
 Installing collected packages: apache-flink
  Attempting uninstall: apache-flink
   Found existing installation: apache-flink x.xx.x
   Uninstalling apache- flink-x.xx.x:
    Successfully uninstalled apache-flink-x.xx.x
Successfully installed apache-flink-x.xx.x
```

10.2.2 Configuring and Importing a Sample Project

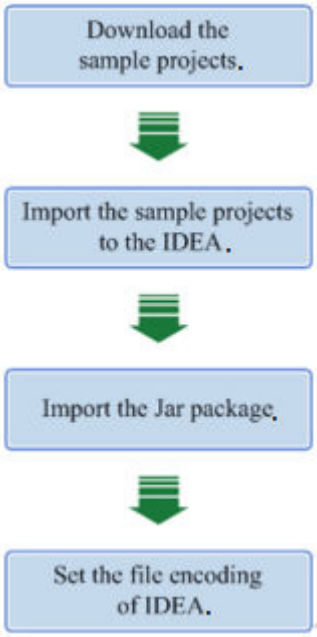
Scenarios

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**.

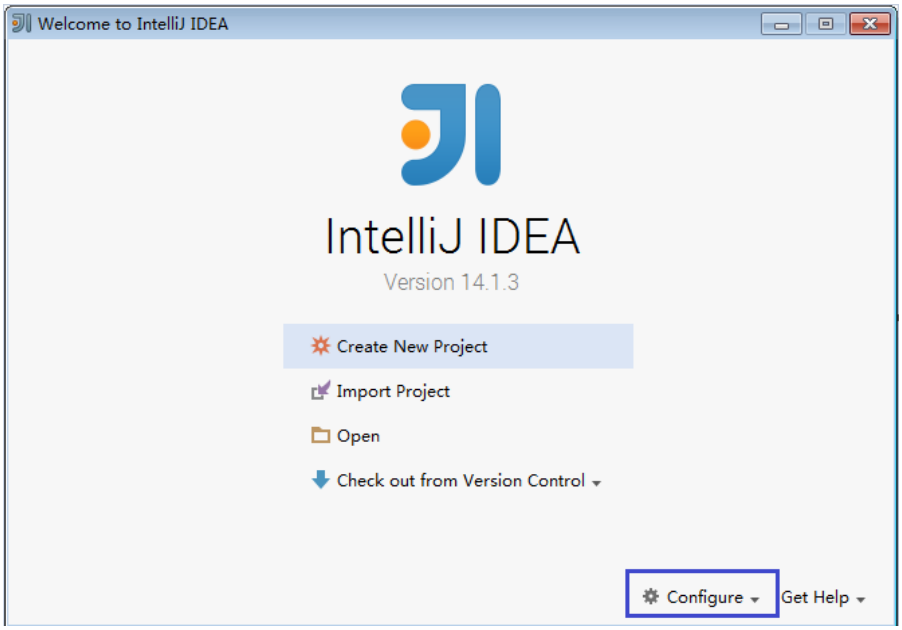
NOTE

- 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.

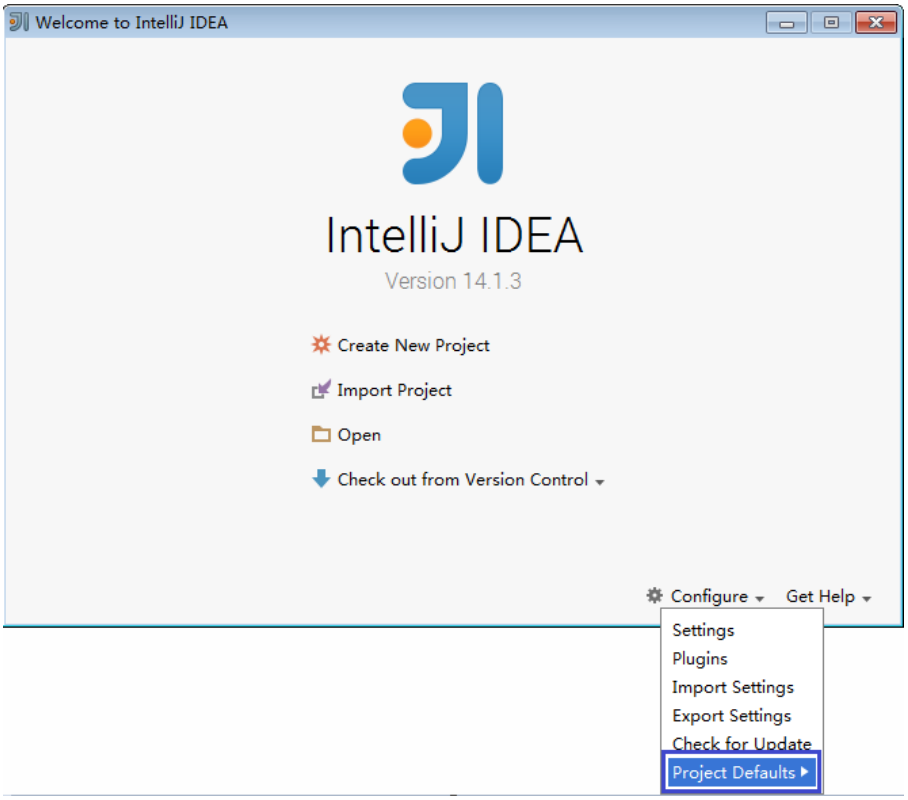
1. Start IntelliJ IDEA and click **Configure**.

Figure 10-5 Choosing Configure



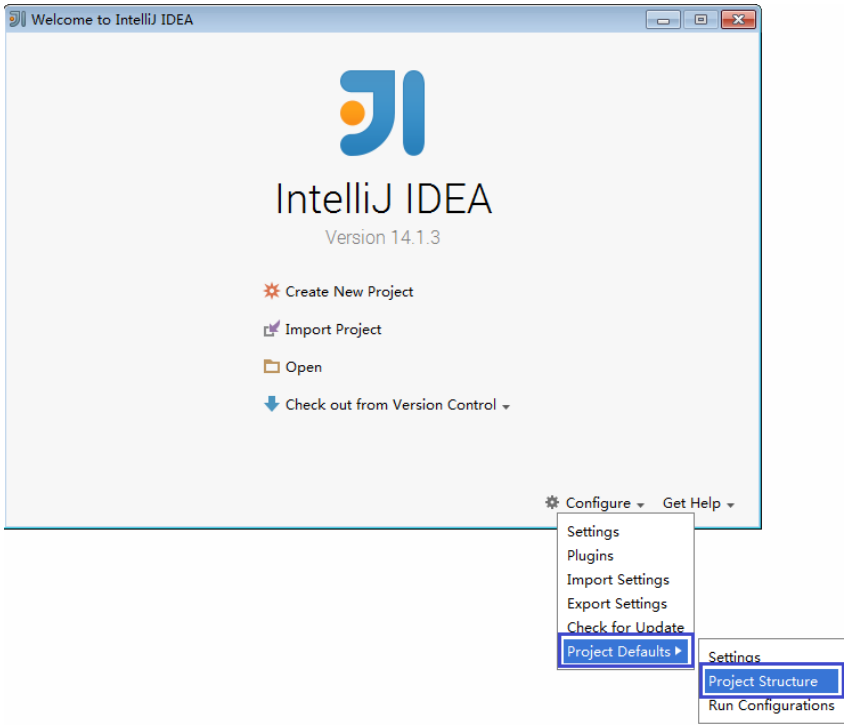
2. Choose **Project Defaults** from the **Configure** drop-down list.

Figure 10-6 Choosing Project Defaults



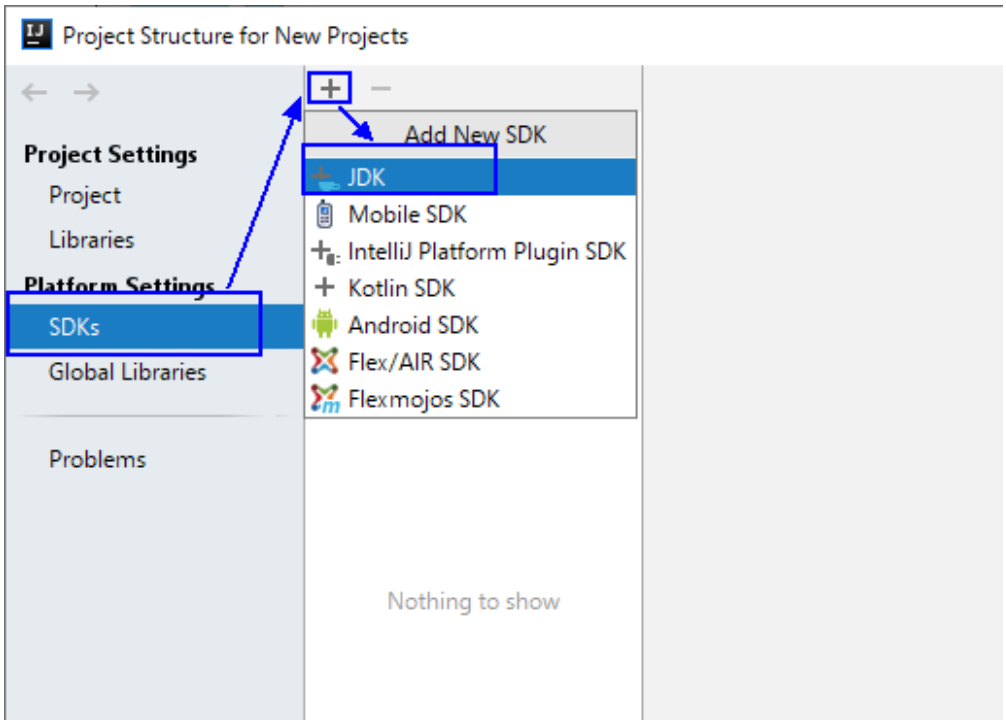
3. Choose **Project Structure** from the **Project Defaults** submenu.

Figure 10-7 Project Defaults



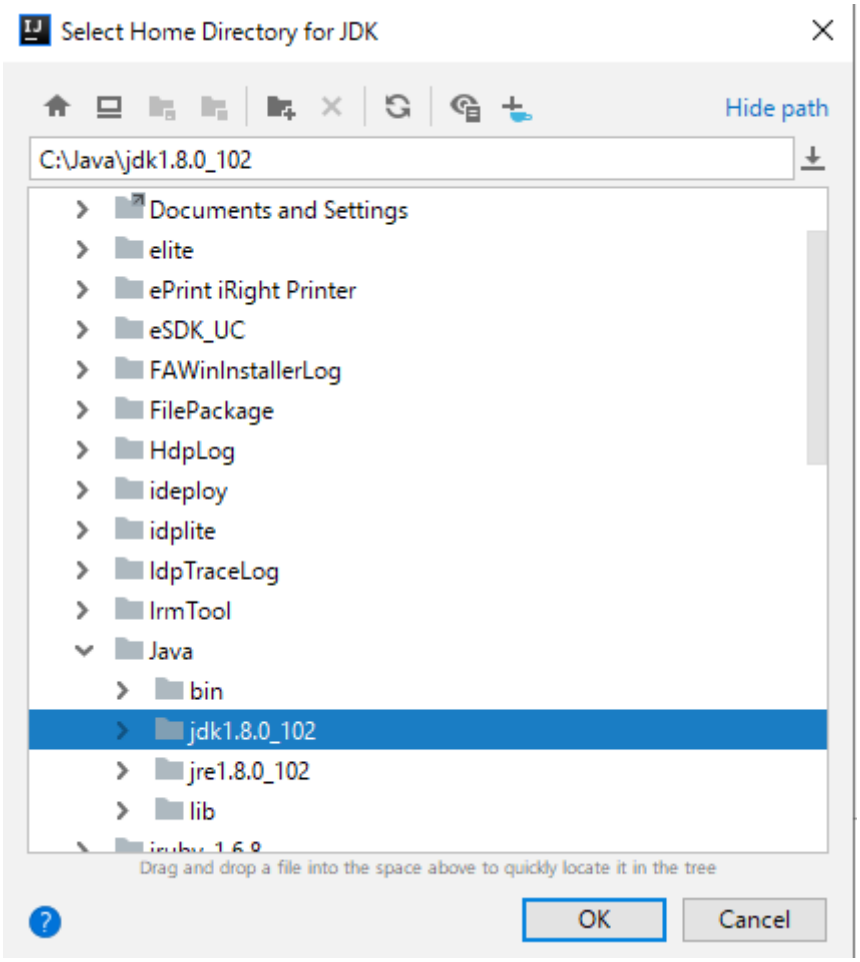
4. On the **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 10-8 Adding the JDK



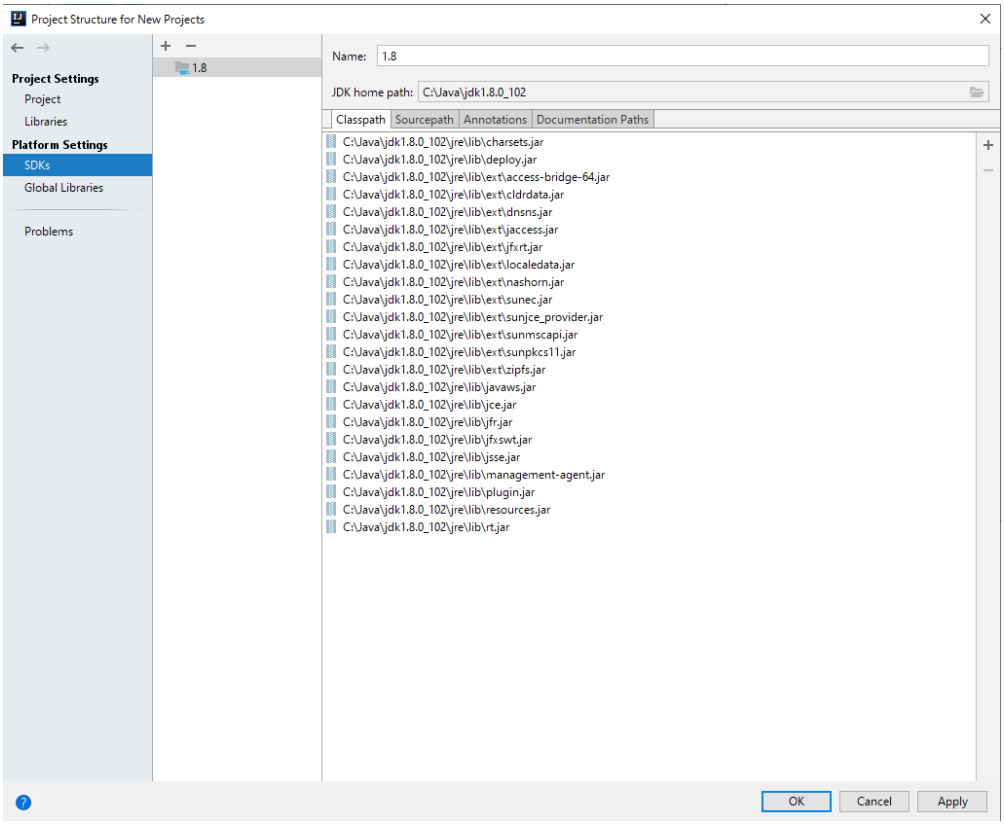
5. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 10-9 Selecting the JDK directory



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

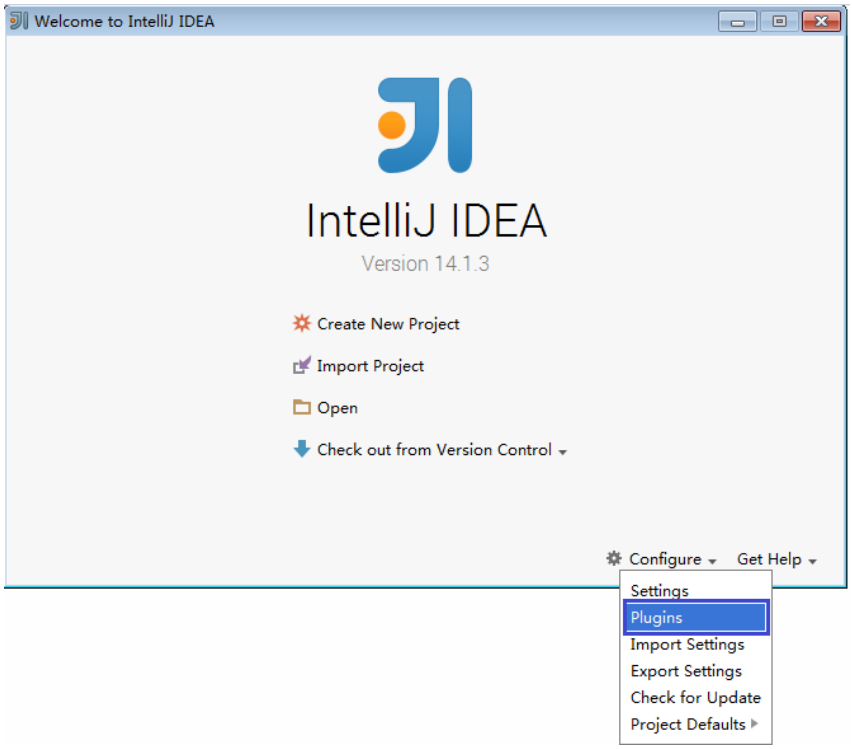
Figure 10-10 Completing the JDK configuration



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

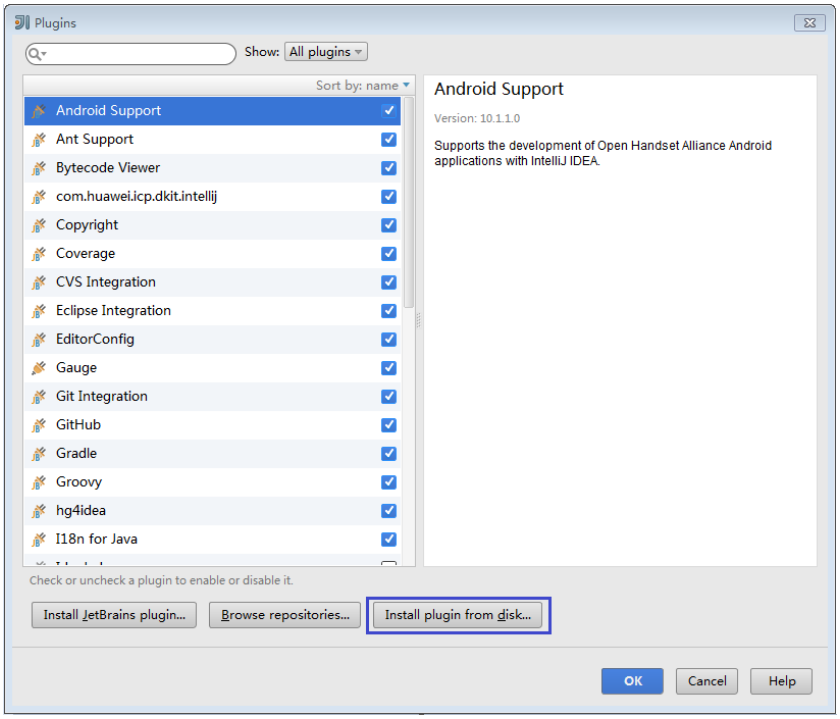
1. Choose **Plugins** from the **Configure** drop-down list.

Figure 10-11 Plugins



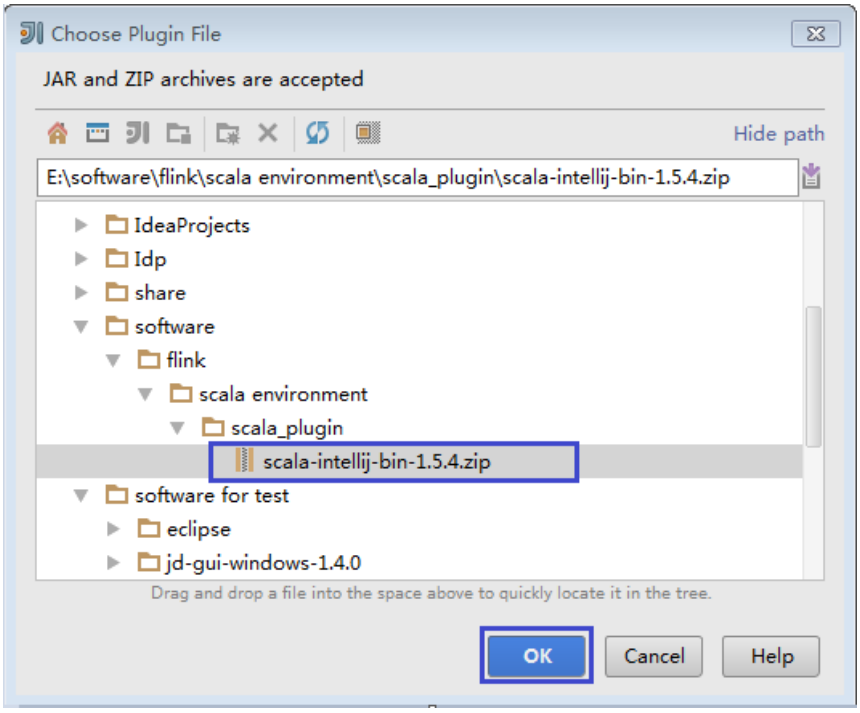
2. On the **Plugins** page, click **Install plugin from disk**.

Figure 10-12 Install plugin from disk



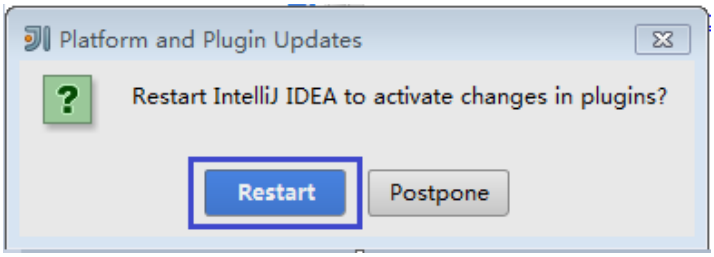
3. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.

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

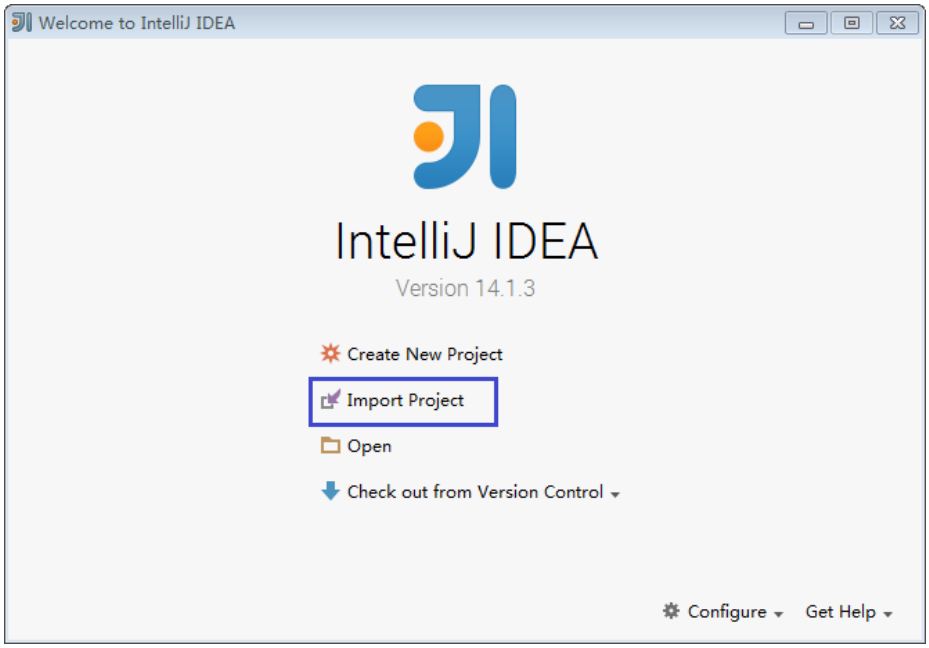


Step 4 Import the Java sample project to IDEA.

1. Start IntelliJ IDEA. On the **Quick Start** page, select **Import Project**.

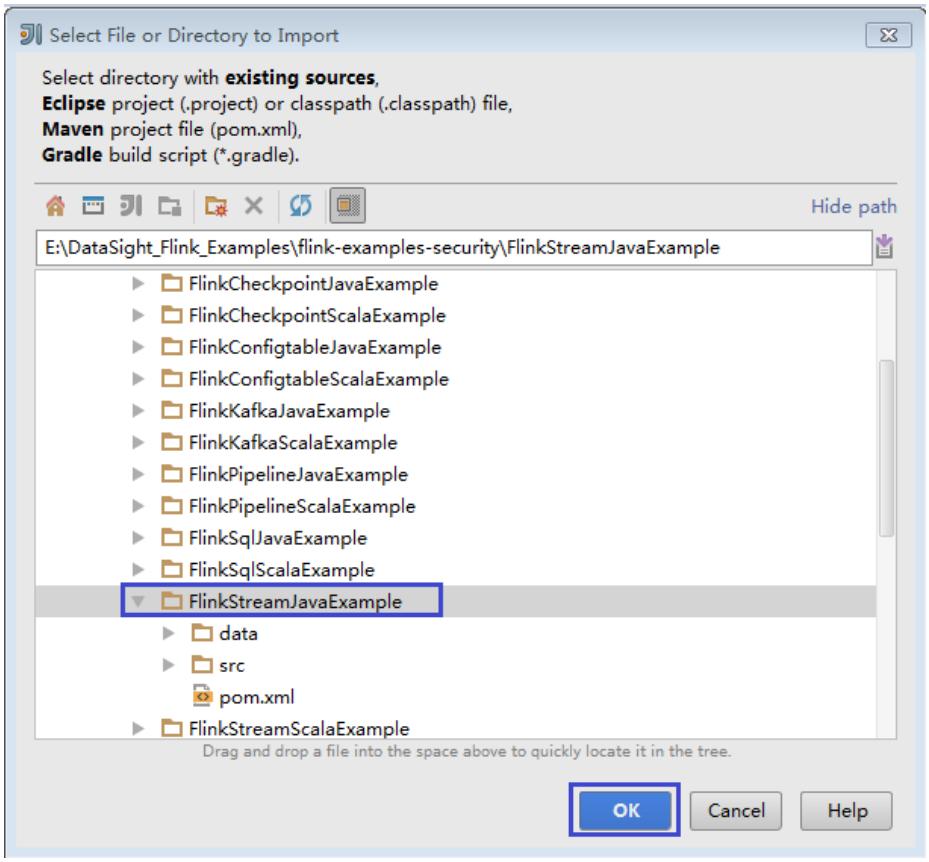
   Alternatively, for the used IDEA tool, add the project directly from the IDEA home page. Choose **File** > **Import project...** to import a project.

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



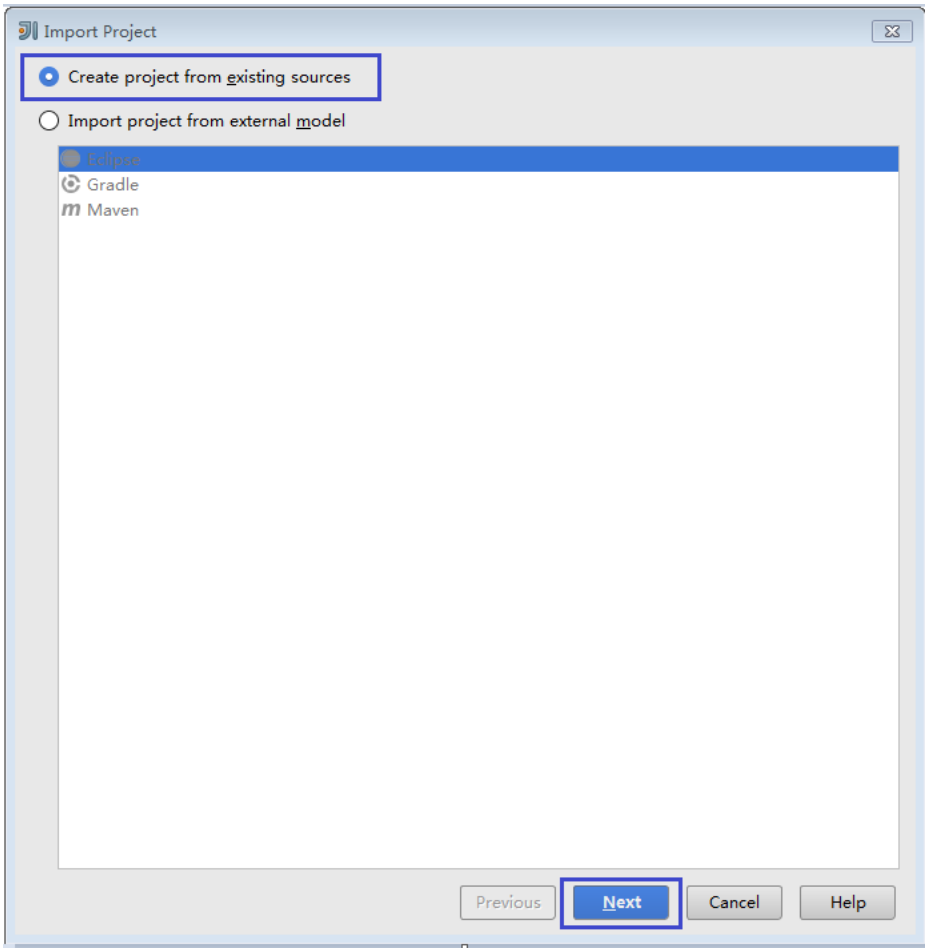
2. Select the directory for storing the imported projects and click **OK**.

Figure 10-16 Select File or Directory to Import



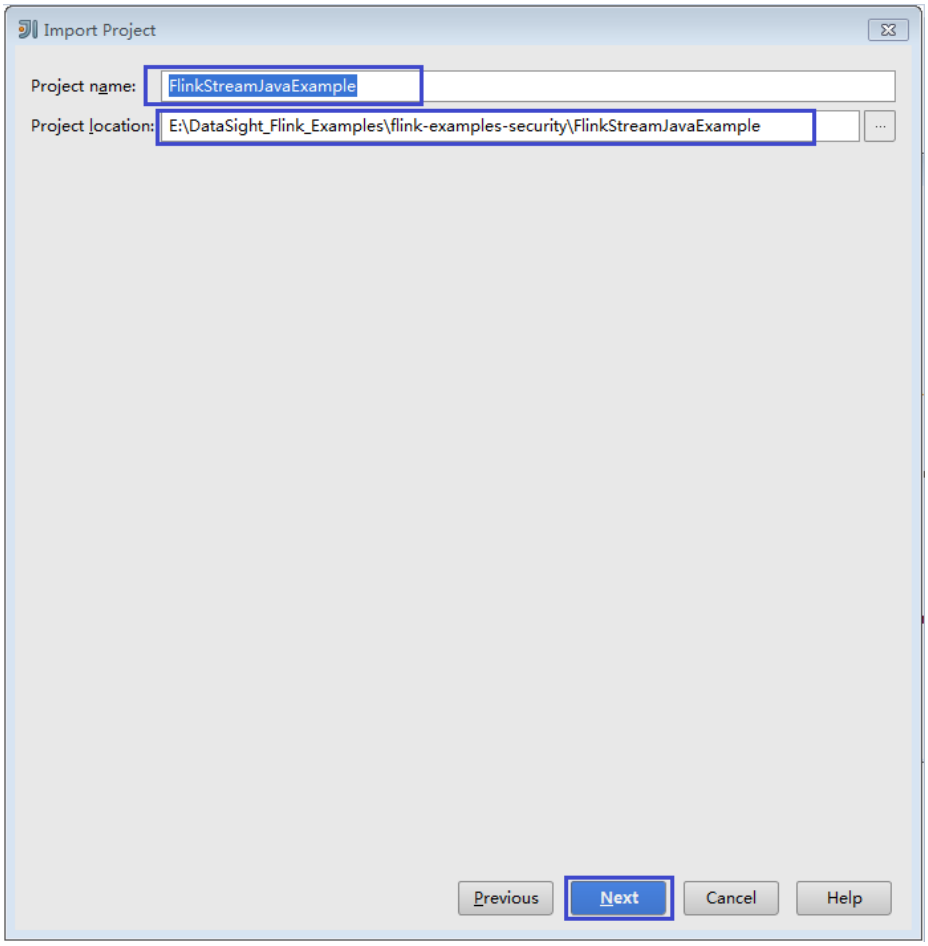
3. Select **Create project from existing sources** and click **Next**.

Figure 10-17 Create project from existing sources



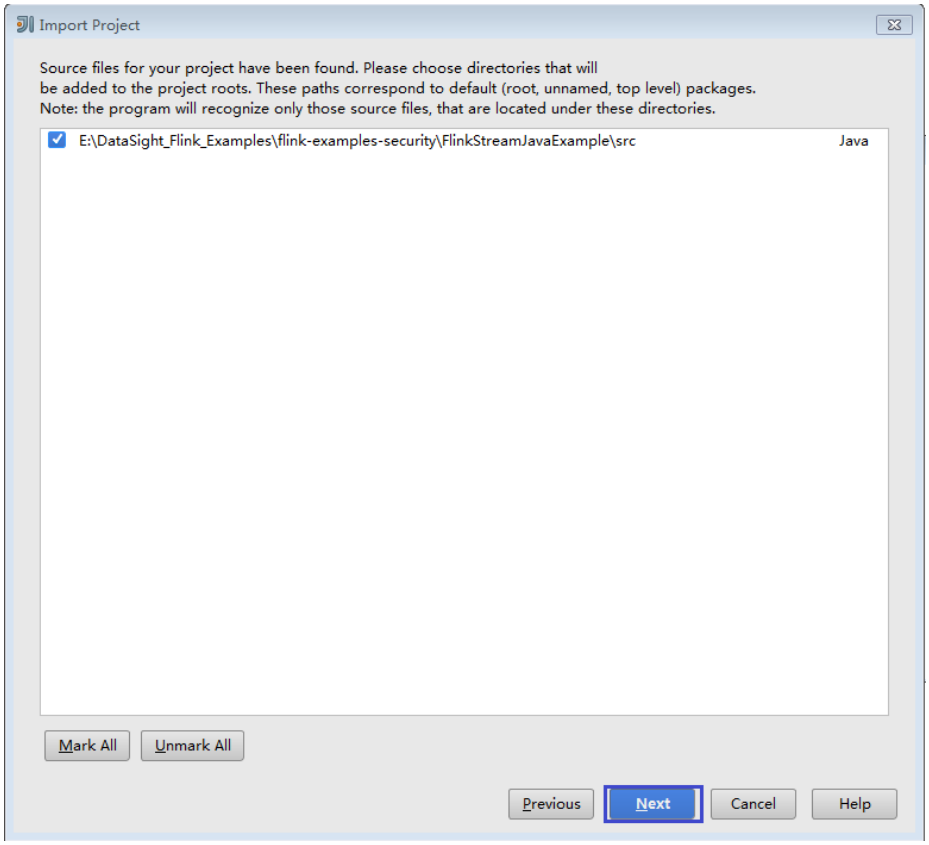
4. Confirm the project location and project name, and click **Next**.

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

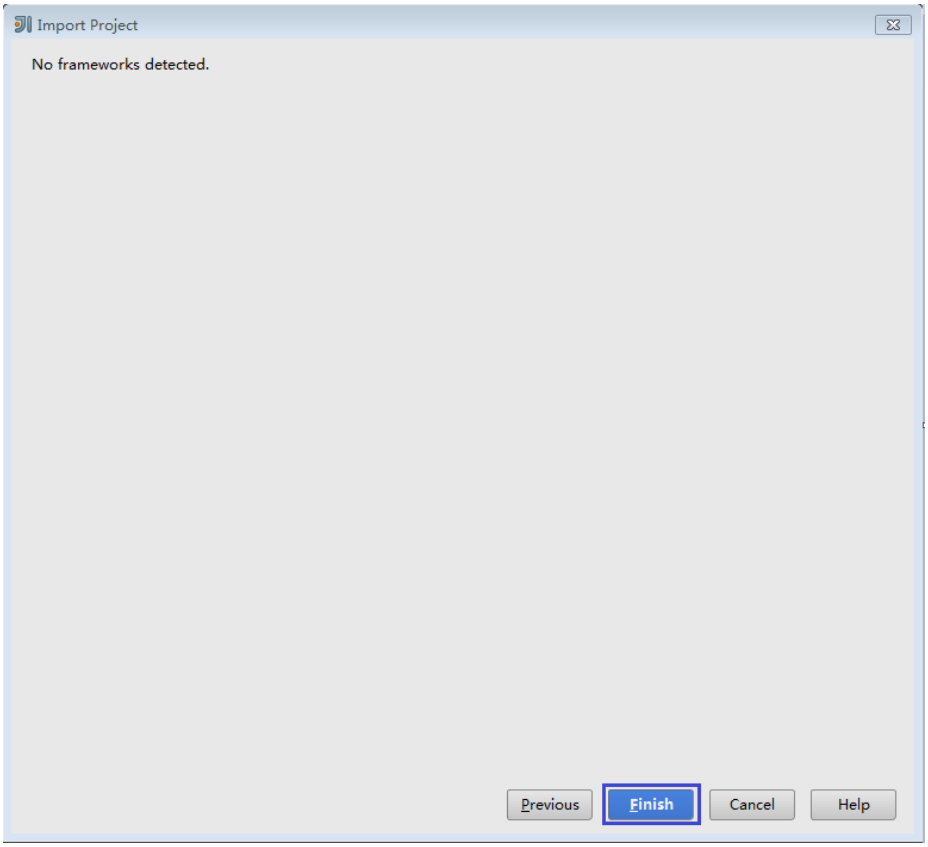
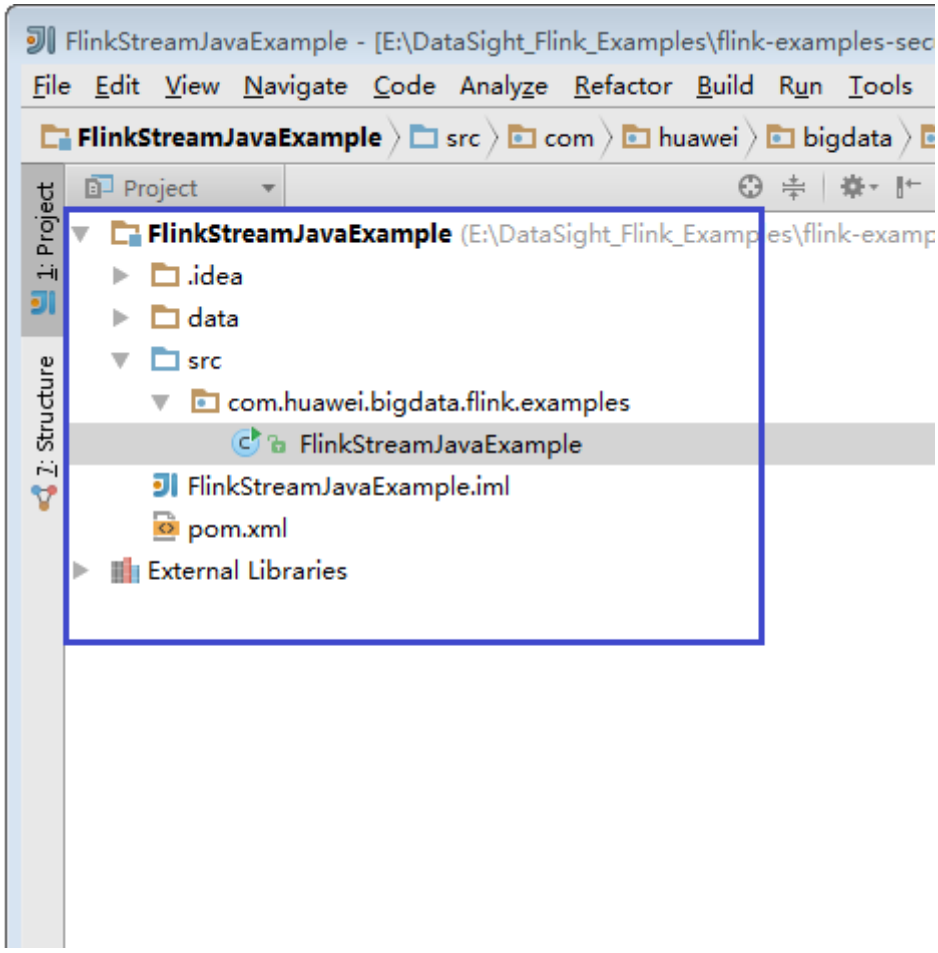


Figure 10-21 Imported project



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.

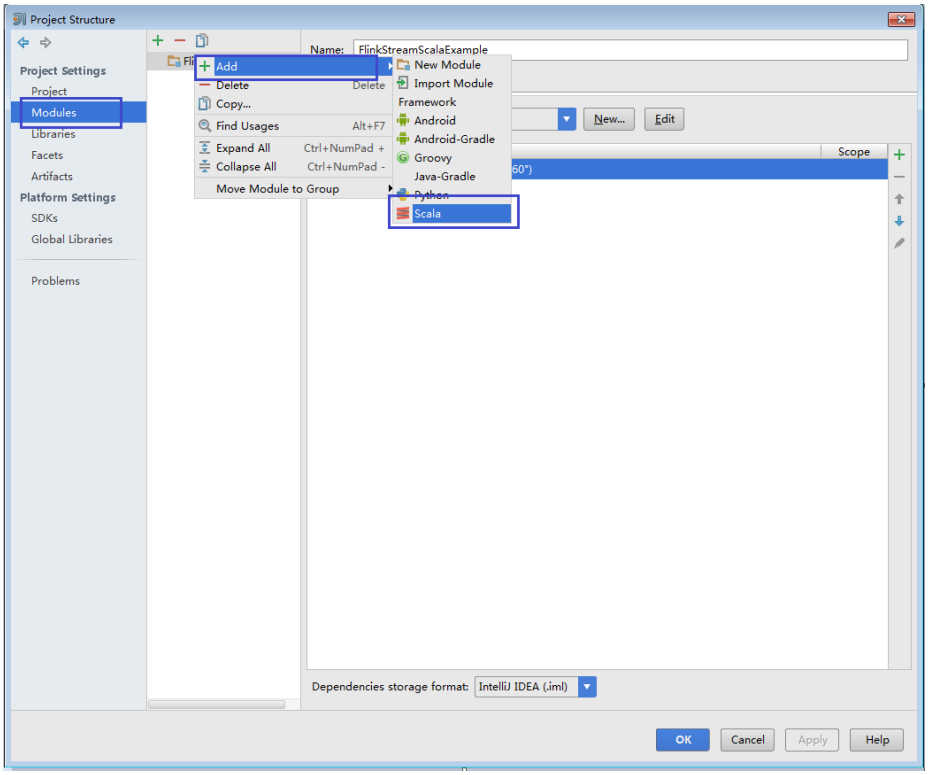
NOTE

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 about the dependency package for running the sample project**.

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

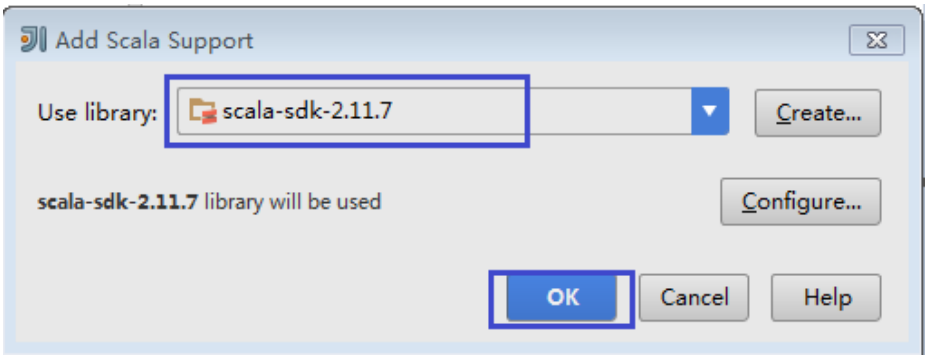
1. On the IDEA home page, choose **File** > **Project Structures...** to go to the **Project Structure** page.
2. Choose **Modules**, right-click a project name, and choose **Add** > **Scala**.

Figure 10-22 Selecting Scala



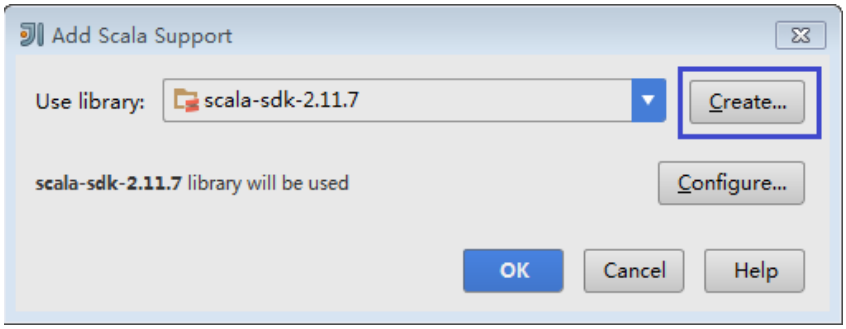
3. Wait until IDEA identifies Scala SDK, select the dependency JAR files in the **Add Scala Support** dialog box, and then click **OK**.

Figure 10-23 Add Scala Support



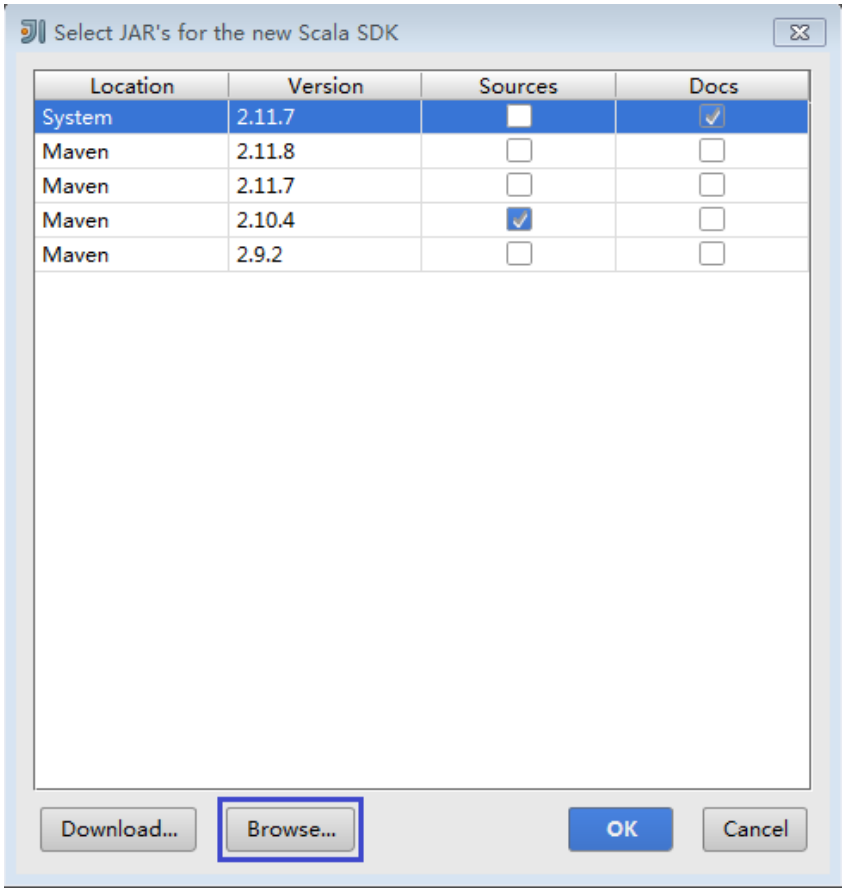
4. If IDEA fails to identify Scala SDK, create a Scala SDK.
   a. Click **Create...**.

Figure 10-24 Create...



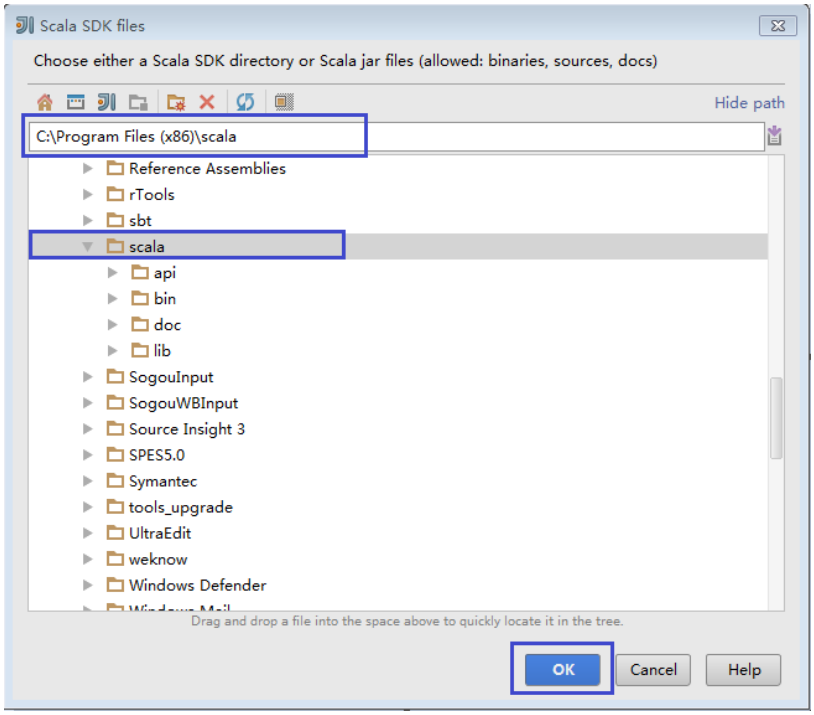
b. On the **Select JAR's for the new Scala SDK** page, click **Browse...**.

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



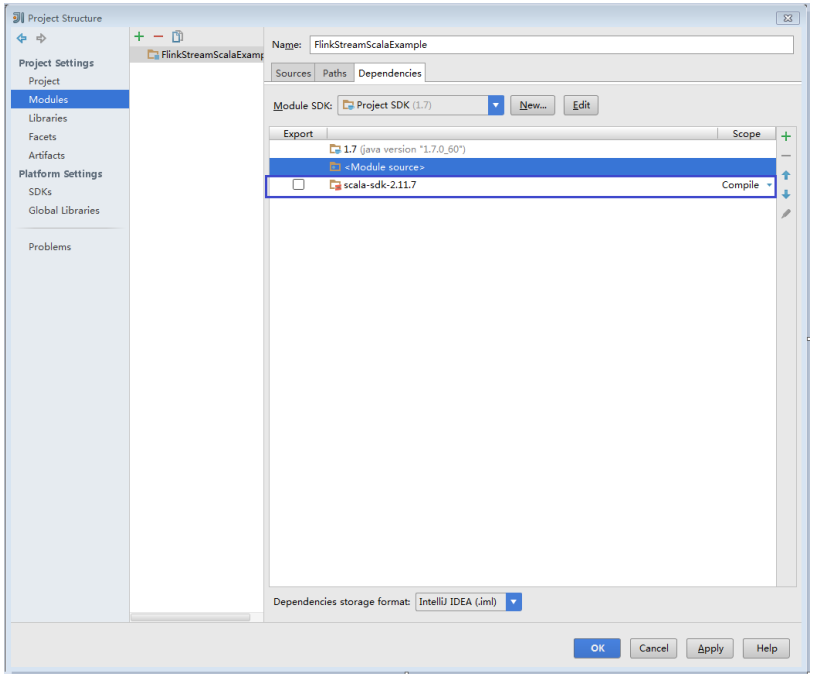
c. On the **Scala SDK files** page, select the **scala sdk** directory and click **OK**.

Figure 10-26 Scala SDK files



5. Click **OK**.

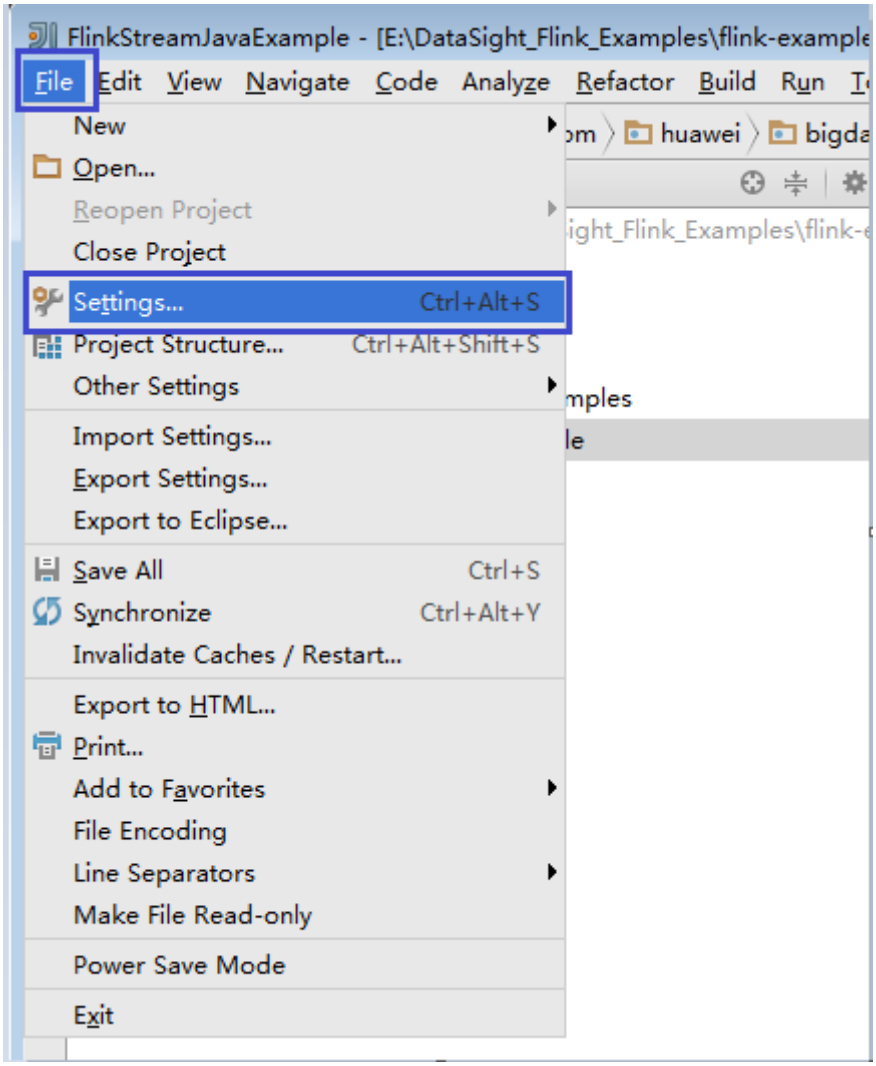
Figure 10-27 The configuration is successful.



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

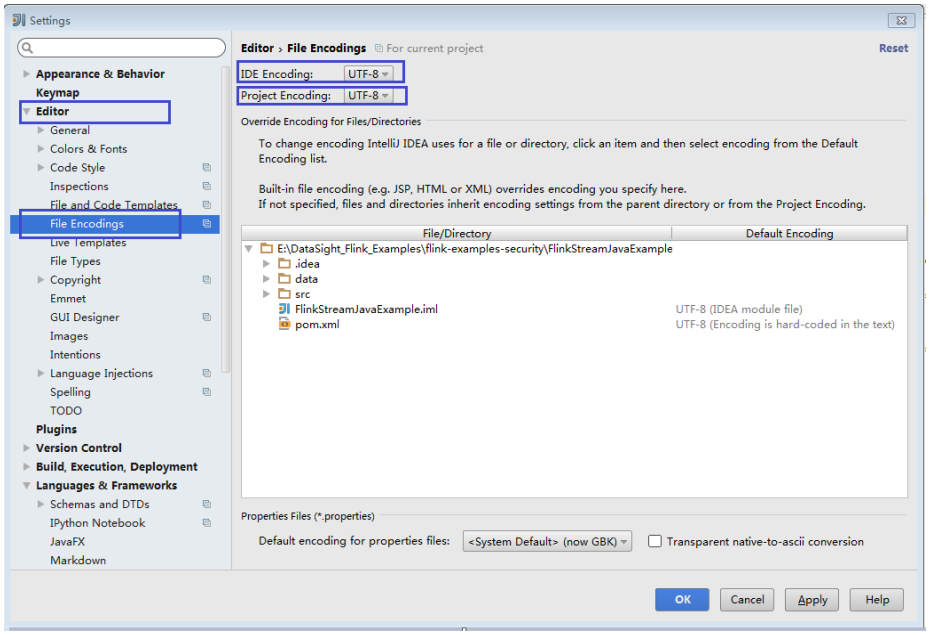
1. On the IDEA home page, choose **File** > **Settings...**.

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 about the dependency package for running the sample project**.

----End

Reference information about the dependency package for running the sample project

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

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

Table 10-5 Dependency package for running the sample project

Sample project	Dependency package	Obtaining Dependency Packages
• Sample project of the DataStream application • Sample project of the asynchronous checkpoint mechanism application	flink-dist_*.jar	Can be obtained from **lib** in the installation directory of the Flink client or server.
• Sample projects of Submitting SQL Jobs Using Flink Jar • Sample projects of FlinkServer REST API	• flink-dist_*.jar • flink-table_*.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	• kafka-clients-*.jar • flink-connector-kafka_*.jar	• **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. • flink-connector-kafka_*.jar can be obtained from **opt** in the installation directory of the Flink client or server.
Sample projects of pipeline	• flink-connector-netty_*.jar • flink-dist_*.jar	• flink-connector-netty_*.jar can be obtained from the **lib** folder generated after the secondary development sample code is compiled. • flink-dist_*.jar can be obtained from **lib** in the installation directory of the Flink client or server.

Sample project	Dependency package	Obtaining Dependency Packages
Sample projects of Stream SQL JOIN	• kafka-clients-*.jar • flink-connector-kafka_*.jar • flink-dist_*.jar • flink-table_*.jar	• **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. • flink-connector-kafka_*.jar can be obtained from **opt** in the installation directory of the Flink client or server. • flink-dist_*.jar, flink-table_*.jar can be obtained from **lib** in the installation directory of the Flink client or server.
Sample projects of HBase	• flink-connector-hbase*.jar • flink-dist_*.jar • flink-table_*.jar • hbase-clients-*.jar	• flink-connector-hbase_*.jar can be obtained from **opt** in the installation directory of the Flink client or server. • flink-dist_*.jar, flink-table_*.jar can be obtained from **lib** in the installation directory of the Flink client or server. • **hbase-clients-*.jar** is provided by HBase and can be obtained from the **lib** directory in the installation directory of the HBase client or server.
Sample projects of Hudi	hbase-unsafe-*.jar	Can be obtained from the **lib** folder generated after the secondary development sample code is compiled.

10.2.3 Creating a Project (Optional)

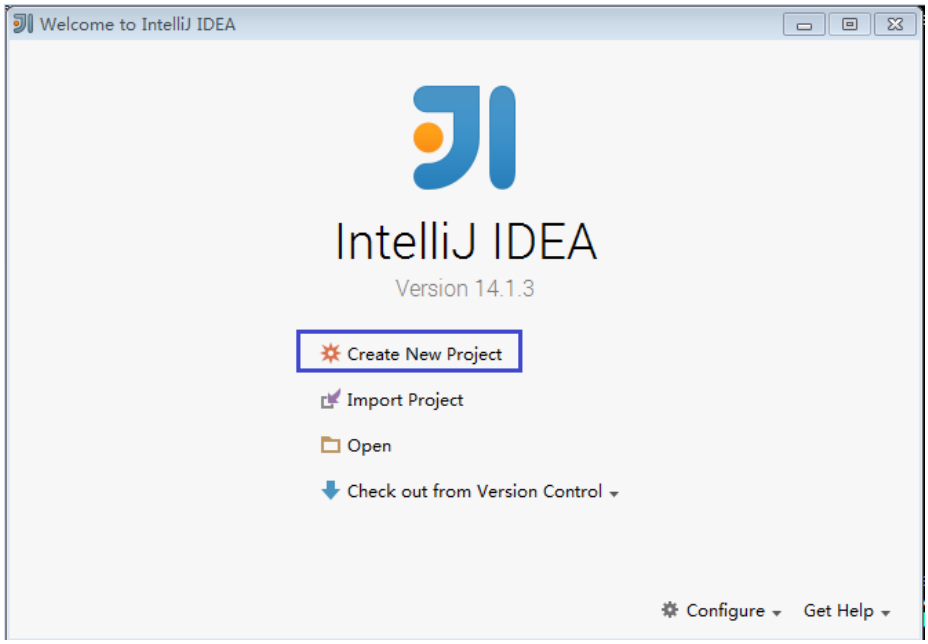
Scenarios

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

Procedure

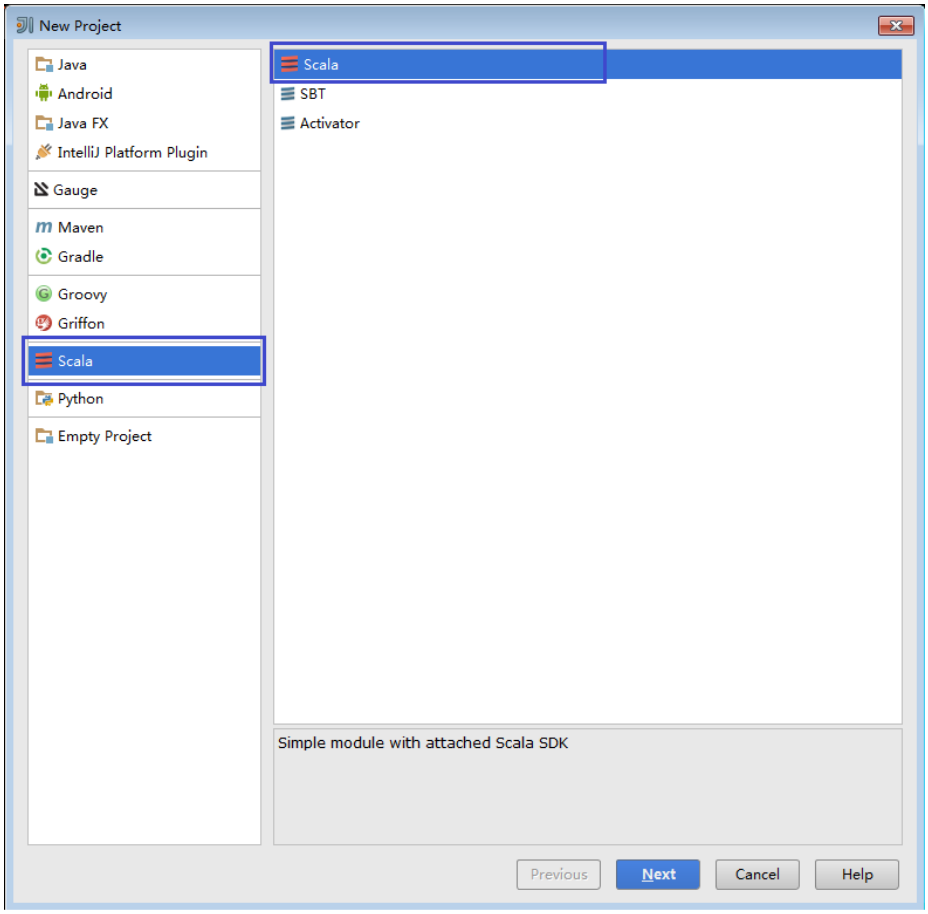
Step 1 Start the IDEA and choose **Create New Project**.

Figure 10-30 Creating a project



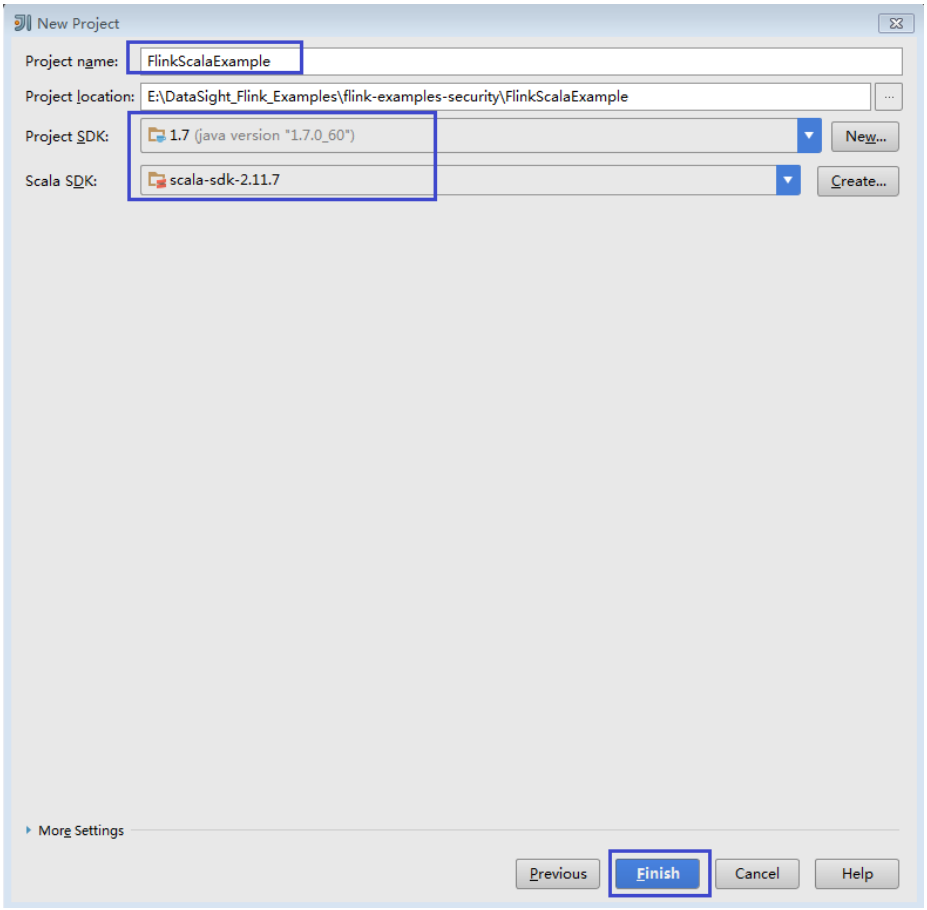
Step 2 On the **New Project** page, choose a Scala development environment, choose **Scala Module**, and then click **Next**. If you want to create a Java project, choose corresponding parameters of Java.

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

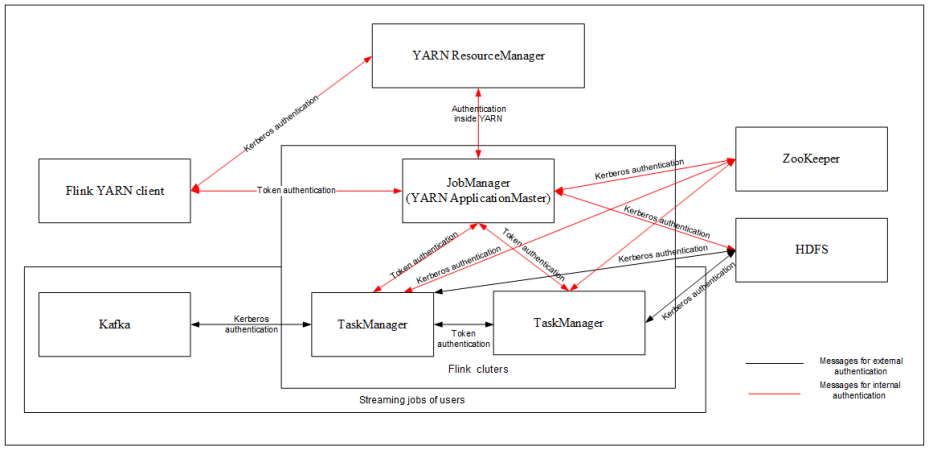
In secure cluster mode, all clusters must authenticate each other before communicating.

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 10-6 Authentication modes

Authentication Mode	Configuration Method
Kerberos authentication Currently, only keytab authentication mode is supported.	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: a. Add the service IP address of the node where the client is installed and IP address of the master node to the **jobmanager.web.access-control-allow-origin** and **jobmanager.web.allow-access-address** configuration items in the **/opt/hadoopclient/Flink/flink/conf/flink-conf.yaml** file. Use commas (,) to separate the IP addresses. jobmanager.web.access-control-allow-origin: *xx.xx.xxx.xxx,xx.xx.xxx.xxx,xx.xx.xxx.xxx* jobmanager.web.allow-access-address: *xx.xx.xxx.xxx,xx.xx.xxx.xxx,xx.xx.xxx.xxx* **NOTE** Node outside the cluster: IP address of the ECS where the client is installed.Node inside the cluster: In the navigation tree of the MRS management console, choose **Clusters > Active Clusters**, select a cluster, and click its name to switch to the cluster details page.On the **Nodes** tab page, view the IP address of the node where the client is installed. b. Keytab path. security.kerberos.login.keytab: /home/flinkuser/keytab/flinkuser.keytab **NOTE** **/home/flinkuser/keytab/** indicates the directory for storing keytab. c. Principal name. security.kerberos.login.principal. flinkuser d. 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 e. If you want to perform Kerberos authentication between Kafka client and Kafka broker, set the value as follows: security.kerberos.login.contexts: Client,KafkaClient

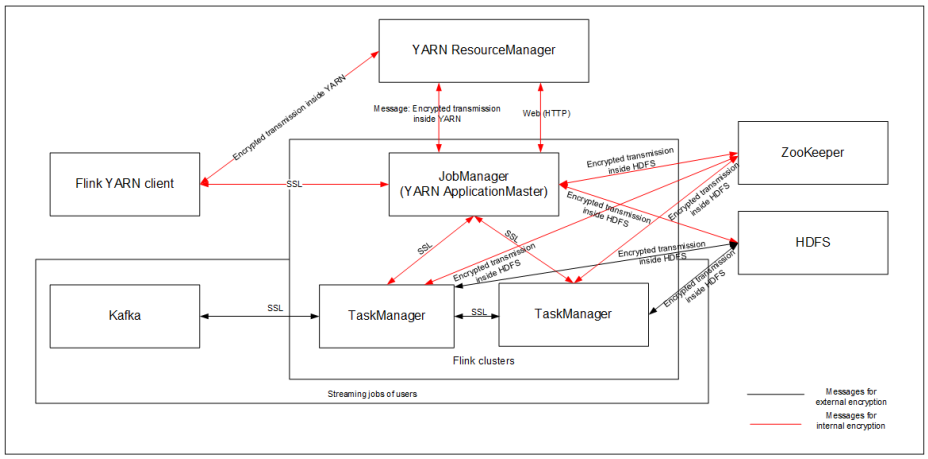
Authentication Mode	Configuration Method
Security cookie authentication	1. Obtain the SSL certificate and save it to the Flink client. For details, see **Authentication and Encryption**. 2. Assign values to the following configuration items in the **flink-conf.yaml** file in the **conf** directory of the Flink client: • Set the **security.ssl.key-password**, **security.ssl.keystore-password**, and **security.ssl.truststore-password** to *<password>*. • Set **security.ssl.keystore** to the relative path of the keystore file, that is, ssl/flink.keystore. • Set **security.ssl.truststore** to the relative path of the truststore file, that is, ssl/flink.truststore. • Set **security.cookie** to a random password string. **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**. 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, *Username*:*Password* 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. *x.x.x.x* indicates the floating IP address of Manager. 3. Set **security.enable: true** to **true** and configure **security cookie**. Example: `security.cookie: ae70acc9-9795-4c48-ad35-8b5adc8071744f605d1d-2726-432e-88ae-dd39bfec40a9`
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 10-7** lists the parameters. Modify the parameter value as required.

Table 10-7 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 10-8** are set by default. You can also configure them manually.

Table 10-8 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	123456	Password of keystore. The **123456** indicates the user-defined password.
security.ssl.key-password	123456	Password of SSL key. The **123456** indicates the user-defined password.

Parameter	Example Parameter Value	Description
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.
security.ssl.truststore-password	123456	Password of truststore, The **123456** 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**.

3. 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.
4. 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

10.2.5 Configuring a Spring Boot Sample Project

This section applies to MRS 3.3.0 or later.

Scenarios

Run the sample code of the Spring Boot interface in FusionInsight MRS Hive. Currently, GaussDB(DWS) SpringBoot sample projects are supported.

In the following example, an application is developed in Linux to connect GaussDB(DWS) to Flink using Spring Boot.

NOTE

Before run the GaussDB(DWS) sample, log in to the node where GaussDB(DWS) is deployed to create an empty table **test_lzh1** for receiving data. The creation command is as follows:

create table test_lzh1 (id integer not null);

Procedure

Step 1 Install a client that contains the Flink service.

The following example shows how to install the Flink client on a node in the cluster:

1. Log in to FusionInsight Manager. On the home page, click **Download Client**. The **Download Cluster Client** dialog box is displayed.
2. In the **Download Cluster Client** dialog box that is displayed, select **Complete Client** for **Select Client Type**, select the platform type that matches the architecture of the node where the client is to install, select **Save to Path**, and click **OK**.

   The generated file is stored in the **/tmp/FusionInsight-Client** directory on the active management node by default.

   The name of the client software package is in the following format: **FusionInsight_Cluster_*<Cluster ID>*_Services_Client.tar**. The following content uses the cluster ID **1** as an example.
3. Log in to the server where the client is to be installed as the client installation user.
4. Go to the directory where the installation package is stored and run the following commands to decompress the package:

   cd /tmp/FusionInsight-Client

   tar -xvf FusionInsight_Cluster_1_Services_Client.tar
5. Run the following command to verify the decompressed file and check whether the command output is consistent with the information in the **sha256** file:

   sha256sum -c FusionInsight_Cluster_1_Services_ClientConfig.tar.sha256
6. Decompress the obtained installation file.

   tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar

7. Go to the directory where the installation package is stored, and run the following command to install the client to a specified directory (absolute path), for example, **/opt/hadoopclient**:

   cd /tmp/FusionInsight-Client/FusionInsight_Cluster_1_Services_ClientConfig

   ./install.sh /opt/hadoopclient

8. Configure security authentication and encryption. For details, see "Using Flink from Scratch" in the *Component Operation Guide*.

Step 2 Obtain the sample project folder **flink-dws-sink-example** from **src\springboot\flink-examples** in the directory where the sample code is decompressed. For details, see **Obtaining Sample Projects from Huawei Mirrors**.

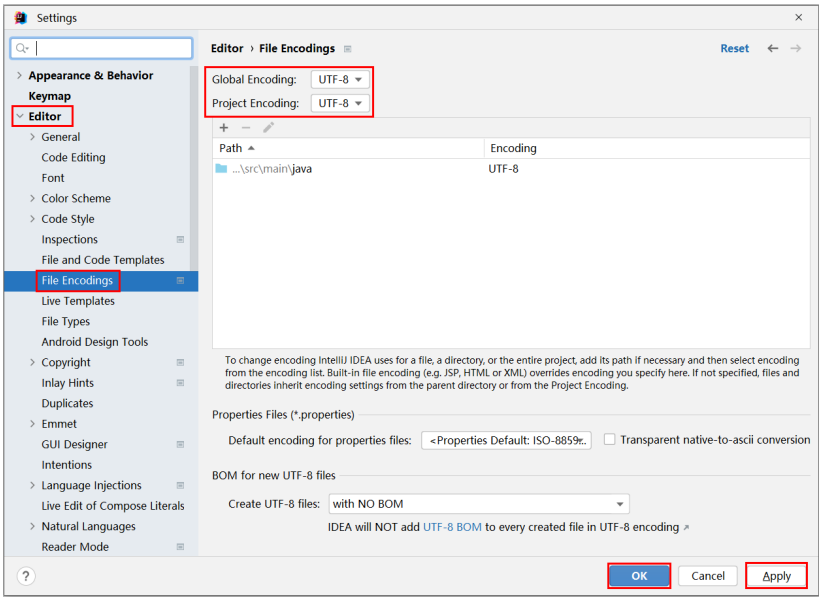
Step 3 Import the sample project to the IntelliJ IDEA development environment.

1. On the menu bar of IntelliJ IDEA, choose **File** > **Open...**.
2. In the **Open File or Project** dialog box that is displayed, select the **flink-dws-sink-example** folder and click **OK**.

Step 4 Set the IntelliJ IDEA text file coding format to prevent garbled characters.

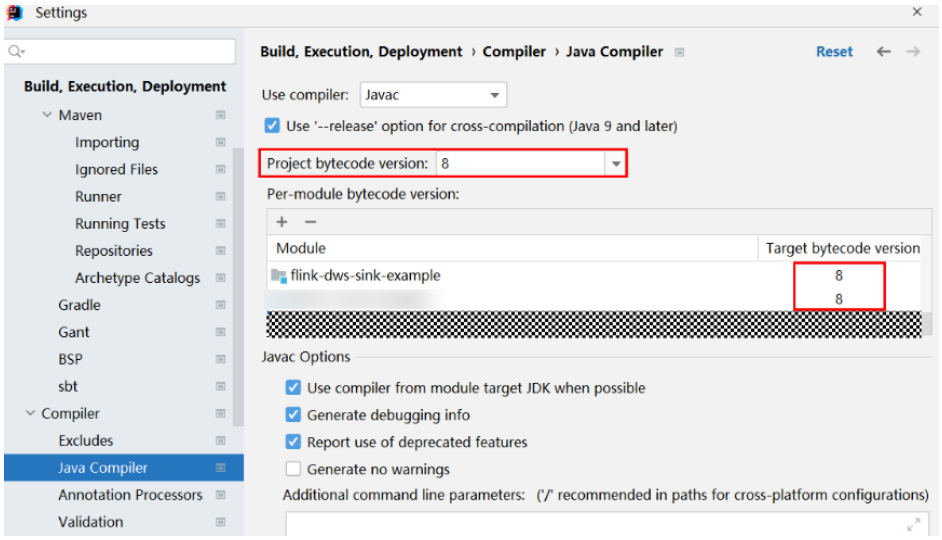
1. On the menu bar of IntelliJ IDEA, choose **File** > **Settings**. The **Settings** window is displayed.
2. In the left navigation pane, choose **Editor** > **File Encodings**. In the **Project Encoding** and **Global Encoding** areas, set the parameter to **UTF-8**. Click **Apply** and **OK**.

Figure 10-35 Setting the IntelliJ IDEA coding format

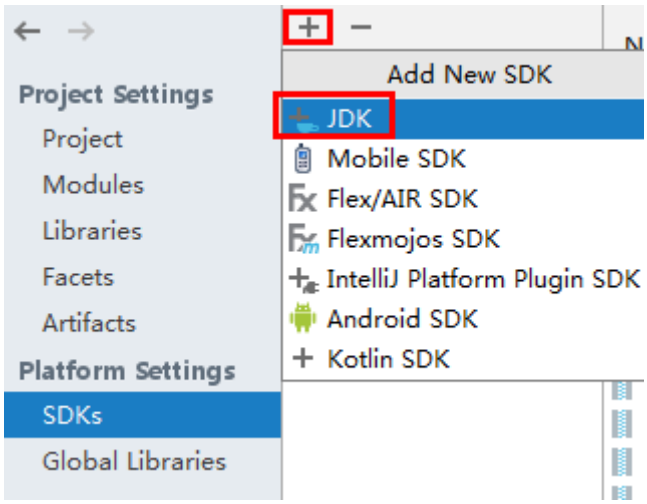


Step 5 Set the JDK of the project.

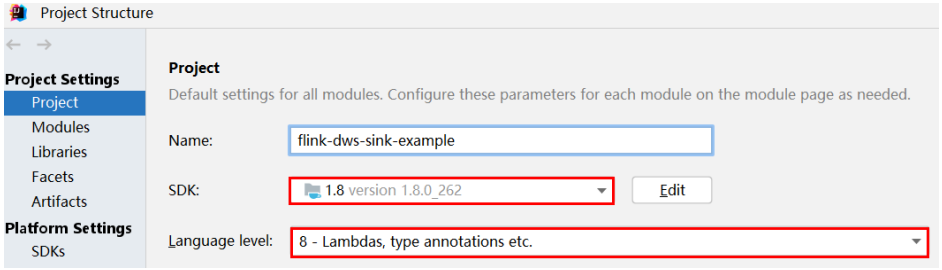
1. On the menu bar of IntelliJ IDEA, choose **File** > **Settings**. The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment** > **Compiler** > **Java Compiler** and select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** in **flink-dws-sink-example** to **8**.



3. Click **Apply** then **OK**.
4. On the menu bar of IntelliJ IDEA, choose **File** > **Project Structure...**. The **Project Structure** window is displayed.
5. Choose **SDKs**, click the plus sign (+), and select **JDK**.

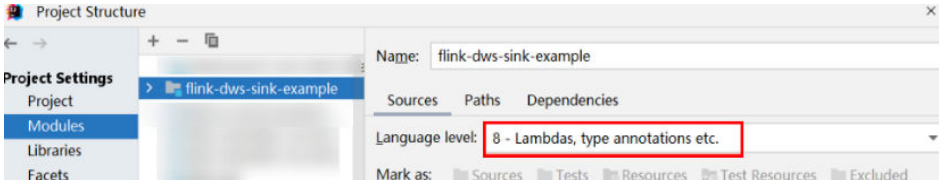


6. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.
7. Click **Apply**.
8. Select **Project**, select the JDK added in **SDKs** from the **SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Language level** drop-down list box.

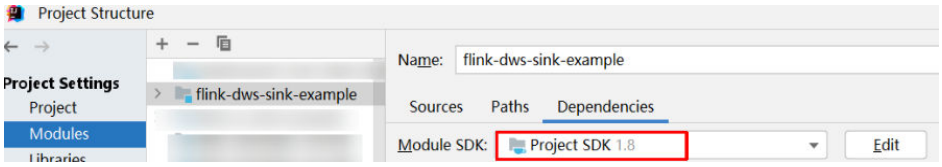


9. Click **Apply**.

10. Choose **Modules**. On the **Sources** tab page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**.



On the **Dependencies** tab page, change the value of **Module SDK** to the JDK added in **SDKs**.

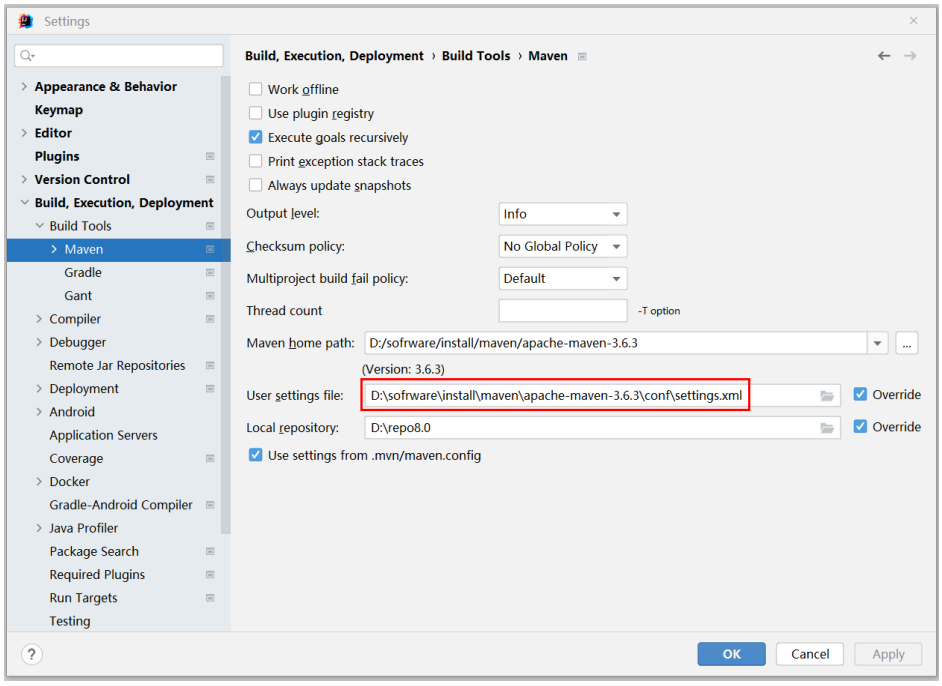


11. Click **Apply** and then **OK**.

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, choose **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 10-36 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 **OK**.

Step 7 Find the **application.properties** file in **src\main\resources** and add the following content to the file:

- Run the GaussDB(DWS) sample

```
spring.datasource.dws.url=jdbc:postgresql://IP address of the GaussDB(DWS) node:8000/postgres
spring.datasource.dws.username=dbadmin
spring.datasource.dws.password=Password of dbadmin
spring.datasource.dws.driver=org.postgresql.Driver
```

----End

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
```

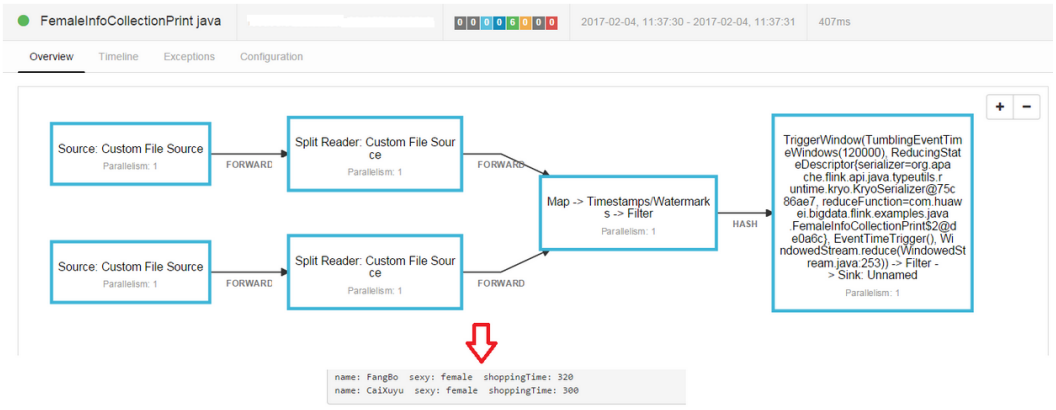
```
        public Tuple2<String, String> getKey(UserRecord value) throws Exception {
            return Tuple2.of(value.name, value.sexy);
        }
    }

    //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 10-37 shows the process of execution.

Figure 10-37 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 {
```

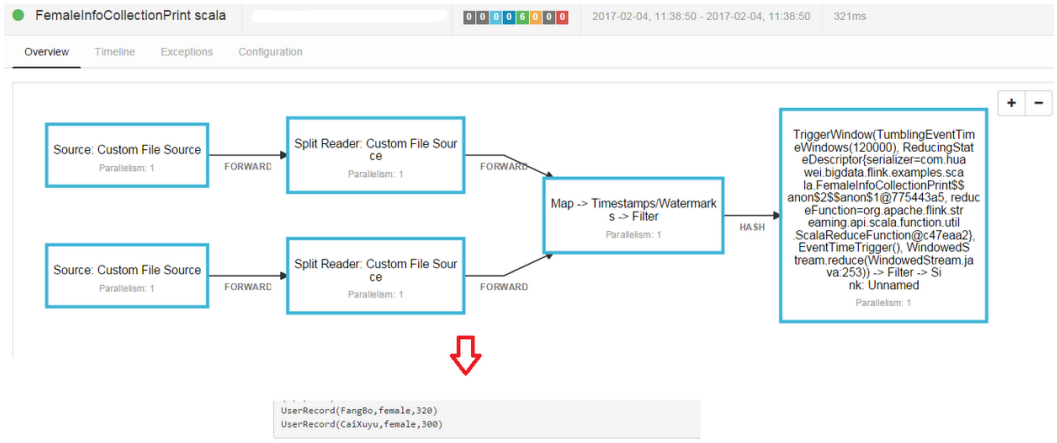
```
    env.readTextFile(filePaths.apply(0))
  }

  //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 10-38 shows the process of execution.

Figure 10-38 Process of execution



10.3.2 Interconnecting with Kafka

10.3.2.1 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.

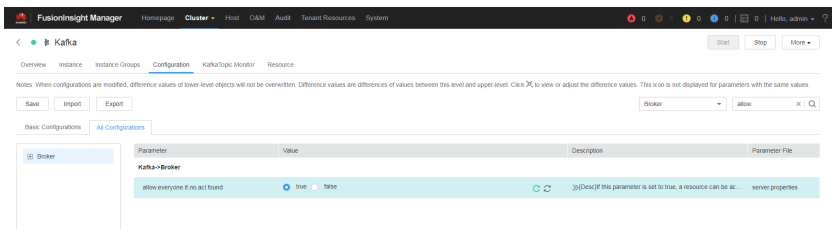
Data Planning

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.
   a. Configure permissions of the user to create topic on the server.

      Change the value of **allow.everyone.if.no.acl.found**, the Broker configuration value of Kafka, to true, as shown in **Figure 10-39**. Then restart the Kafka service.

      Figure 10-39 Configuring permissions of topics on the server

   b. Run Linux command line to create a topic. Before running commands, ensure that the kinit command, for example, **kinit flinkuser**, is run for authentication.

NOTE

Flinkuser requires the user has permission to create Kafka's topic to create it. For details, see **Preparing the Developer Account**.

The format of the command is following:

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

Table 10-9

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.
{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
```

3. Security authentication

   The Kerberos authentication, SSL encryption authentication, or Kerberos + SSL authentication mode can be used.

   - **Configurations about Kerberos authentication**

     i. Configuration on the client.

     In the configuration file **flink-conf.yaml**, add configurations about Kerberos authentication as follows:

     ```
     security.kerberos.login.keytab: /home/demo/flink/release/flink-1.12.2/keytab/user.keytab
     security.kerberos.login.principal: flinkuser
     security.kerberos.login.contexts: Client,KafkaClient
     security.kerberos.login.use-ticket-cache: false
     ```

     ii. Running parameters

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

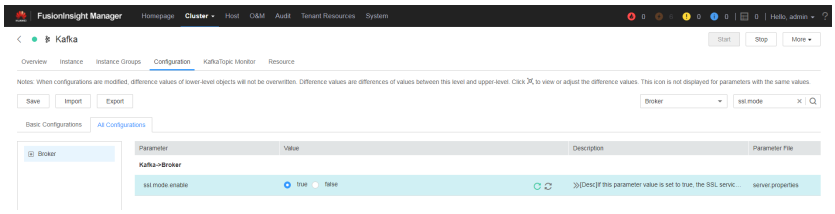
     ```
     --topic topic1 --bootstrap.servers 10.96.101.32:21007 --security.protocol SASL_PLAINTEXT --sasl.kerberos.service.name kafka --kerberos.domain.name hadoop.System domain name.com //10.96.101.32:21007 indicates the IP address:porter of Kafka server.
     ```

   - **SSL encryption**

     - **Configuration about SSL**

       Set **ssl.mode.enable** to **true**, as shown in **Figure 10-40**.

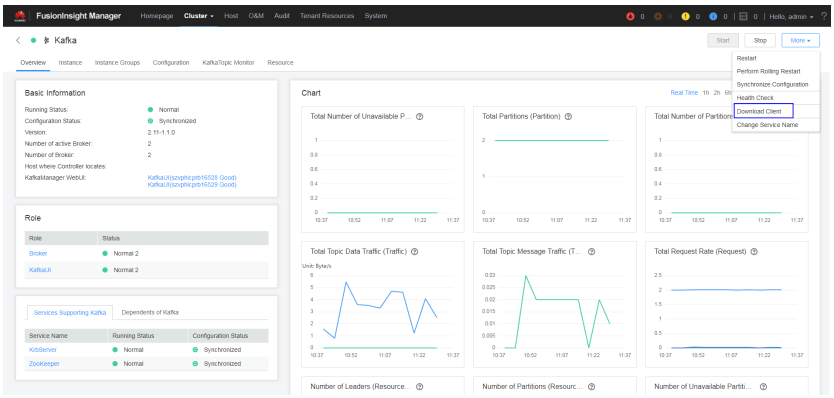
Figure 10-40 Configuration on the server



- Configuration on the client.

  1) Log in to FusionInsight Manager, choose **Cluster** >*Name of the desired cluster* **> Services > Kafka > More**, and click **Download Client** on the displayed **Service Status** page to download Kafka client, as shown in **Figure 10-41**.

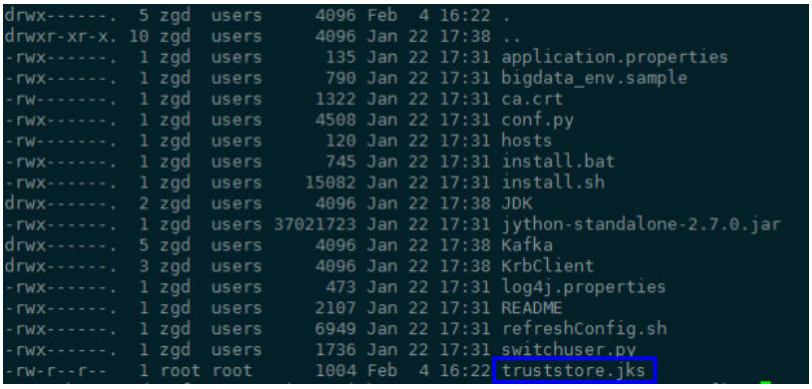
     Figure 10-41 Configuration on the client

  2) Use the ca.crt certificate on the client root directory to generate the truststore for the client. Run the following command:

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

     The command execution result is similar to the following:

     ```
     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) Running parameters.

     Run the following command to execute the running parameters (Ensure that the content of the**ssl.truststore.password** parameter must be the same as the password entered needs to be confirmed by the same as the password entered when you create a **truststore**.):

     ```
     --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 xxx //10.96.101.32:9093 indicates the IP
     ```


address:porter of Kafka server, and XXX indicates the version of FunsionInsight, xxx indicates the password.

- Configuration about Kerberos + SSL mode

  After completing preceding configurations about clients and servers of Kerberos and SSL, modify the port numbers and protocol types in running parameters to start the Kerberos + SSL 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 *xxx* //10.96.101.32:21009 indicates the IP address:porter of Kafka server, and XXX indicates the version of FunsionInsight, xxx indicates the password.

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.

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 --sasl.kerberos.service.name kafka");

      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
("*************************************************************************************");

        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.Seriablizale;

//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 =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuv
wxyzABCDEFGHIJKLMNOPQRSTUVWZYX0987654321";

    //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, Stirng, 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 processing time 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();
    }
}
```

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 =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuv
wxyzABCDEFGHIJKLMNOPQRSTUVWZYX0987654321"

    //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) throw 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 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**.

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 c**om.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("******************************************************************************************");
       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 xxxx.xxx.xxx.xxx:21005 --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 xxxx.xxx.xxx.xxx:21007 --security.protocol
SASL_PLAINTEXT --sasl.kerberos.service.name kafka"
            + "--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();
    }
}
```

10.3.5.3 Scala Sample Code

This section applies to MRS 3.3.0 or later.

Function

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

Sample Code

If you need to use Kafka in security mode before development, you need to import kafka-clients-*.jar of FusionInsight. You can obtain the JAR package from the Kafka client directory.

The following example shows the Producer, Consumer, and the main logic code used by Flink Stream SQL Join.

For the complete codes, see **com.huawei.bigdata.flink.examples.WriteIntoKafka** and **com.huawei.bigdata.flink.examples.SqlJoinWithSocket**.

1. Produce a piece of user information in Kafka every second. The user information includes the name, age, and gender.

```
// Kafka Producer code

object WriteIntoKafka {
  def main(args: Array[String]): Unit = {
    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 xxx.xxx.xxx.xxx:21005")
    System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/
test.jar --topic" + " topic-test -bootstrap.servers xxx.xxx.xxx.xxx:21007 --security.protocol
SASL_PLAINTEXT" + " --sasl.kerberos.service.name kafka")
    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("************************************************************************************************")
    val env = StreamExecutionEnvironment.getExecutionEnvironment
env.setParallelism(1)

    val paraTool = ParameterTool.fromArgs(args)

    val messageStream = env.addSource(new WriteIntoKafka.SimpleStringGenerator)
    val producer = new FlinkKafkaProducer[String](paraTool.get("topic"), new SimpleStringSchema,
paraTool.getProperties)

    producer.setWriteTimestampToKafka(true)

    messageStream.addSink(producer)
    env.execute
  }

  /**
* String source class
   *
   */
  object SimpleStringGenerator {
    private[examples] val NAME = Array("Carry", "Alen", "Mike", "Ian", "John", "Kobe", "James")
    private[examples] val SEX = Array("MALE", "FEMALE")
    private[examples] val COUNT = NAME.length
  }

  class SimpleStringGenerator extends SourceFunction[String] {
    private[examples] var running = true
    private[examples] val rand = new Random(47)

    @throws[Exception]
    override def run(ctx: SourceFunction.SourceContext[String]): Unit = {
      while (running) {
        val i = rand.nextInt(SimpleStringGenerator.COUNT)
        val age = rand.nextInt(70)
        val sexy = SimpleStringGenerator.SEX(rand.nextInt(2))
        ctx.collect(SimpleStringGenerator.NAME(i) + "," + age + "," + sexy)
        Thread.sleep(1000)
      }
    }

    override def cancel(): Unit = {
      running = false
    }
  }

}
```

2. Generate Table1 and Table2, use **Join** to jointly query Table1 and Table2, and print the output result.

```
object SqlJoinWithSocket {
  def main(args: Array[String]): Unit = {
    var hostname: String = null
    var port = 0
    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 xxxx.xxx.xxx.xxx:21005 --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 xxxx.xxx.xxx.xxx:21007 --security.protocol
SASL_PLAINTEXT" + " --sasl.kerberos.service.name kafka--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 {
      val params = ParameterTool.fromArgs(args)
      hostname = if (params.has("hostname")) params.get("hostname")
```

```
      else "localhost"
      port = params.getInt("port")
    } catch {
      case e: Exception =>
        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
    }

    val fsSettings = EnvironmentSettings.newInstance.inStreamingMode.build
    val env = StreamExecutionEnvironment.getExecutionEnvironment
val tableEnv = StreamTableEnvironment.create(env, fsSettings)

    env.getConfig.setAutoWatermarkInterval(200)
    env.setParallelism(1)
    val paraTool = ParameterTool.fromArgs(args)

    val kafkaStream = env.addSource(new FlinkKafkaConsumer[String](paraTool.get("topic"), new
SimpleStringSchema, paraTool.getProperties)).map(new MapFunction[String, Tuple3[String, String,
String]]() {
      @throws[Exception]
      override def map(str: String): Tuple3[String, String, String] = {
        val word = str.split(",")
        new Tuple3[String, String, String](word(0), word(1), word(2))
      }
    })

    tableEnv.createTemporaryView("Table1", kafkaStream, $("name"), $("age"), $("sexy"), $
("proctime").proctime)

    val socketStream = env.socketTextStream(hostname, port, "\n").map(new MapFunction[String,
Tuple2[String, String]]() {
      @throws[Exception]
      override def map(str: String): Tuple2[String, String] = {
        val words = str.split("\\s")
        if (words.length < 2) return new Tuple2[String, String]
        new Tuple2[String, String](words(0), words(1))
      }
    })

    tableEnv.createTemporaryView("Table2", socketStream, $("name"), $("job"), $
("proctime").proctime)

    val 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")

    tableEnv.toAppendStream(result, classOf[Row]).print
    env.execute
  }

}
```

10.3.6 Submitting a SQL Job Using Flink Jar

10.3.6.1 Scenario Description

This scenario applies to MRS 3.2.1 or later.

Description

If SQL statements of a job are frequently modified, submit Flink SQL statements in Flink Jar mode to reduce your workload.

Development Guideline

Use the current sample to submit and execute specified SQL statements. Use semicolons (;) to separate multiple statements.

10.3.6.2 Java Sample Code

The core logic for submitting SQL statements is as follows. Currently, only **CREATE** and **INSERT** statements can be submitted. For details about the complete code, see com.huawei.bigdata.flink.examples.FlinkSQLExecutor.

```
public class FlinkSQLExecutor {
    public static void main(String[] args) throws IOException {
        System.out.println("----------------------  begin init ----------------------");
        final String sqlPath = ParameterTool.fromArgs(args).get("sql", "config/redisSink.sql");
        final StreamExecutionEnvironment streamEnv =
StreamExecutionEnvironment.getExecutionEnvironment();
        EnvironmentSettings bsSettings = EnvironmentSettings.newInstance().inStreamingMode().build();
        StreamTableEnvironment tableEnv = StreamTableEnvironment.create(streamEnv, bsSettings);
        StatementSet statementSet = tableEnv.createStatementSet();
        String sqlStr = FileUtils.readFileToString(FileUtils.getFile(sqlPath), "utf-8");
        String[] sqlArr = sqlStr.split(";");
        for (String sql : sqlArr) {
            sql = sql.trim();
            if (sql.toLowerCase(Locale.ROOT).startsWith("create")) {
                System.out.println("------------------------------------------------\nexecuteSql=\n" + sql);
                tableEnv.executeSql(sql);
            } else if (sql.toLowerCase(Locale.ROOT).startsWith("insert")) {
                System.out.println("------------------------------------------------\ninsert=\n" + sql);
                statementSet.addInsertSql(sql);
            }
        }
        System.out.println("---------------------- begin exec sql ----------------------------");
        statementSet.execute();
    }
}
```

NOTE

Copy the dependency package required by the current sample, that is, the JAR package in the **lib** file after compilation, to the **lib** folder on the client.

The following uses Kafka in a normal cluster as an example to describe how to submit SQL statements:

```
create table kafka_sink
(
    uuid varchar(20),
    name varchar(10),
    age  int,
    ts   timestamp(3),
    p    varchar(20)
) with (
      'connector' = 'kafka',
      'topic' = 'input2',
      'properties.bootstrap.servers' = 'IP address of the Kafka broker instance',
      'properties.group.id' = 'testGroup2',
      'scan.startup.mode' = 'latest-offset',
      'format' = 'json'
      );
create TABLE datagen_source
(
    uuid varchar(20),
    name varchar(10),
    age  int,
    ts   timestamp(3),
```



```
    p    varchar(20)
) WITH (
     'connector' = 'datagen',
     'rows-per-second' = '1'
     );
INSERT INTO kafka_sink
SELECT *
FROM datagen_source;
```

10.3.7 FlinkServer REST API JavaExample

10.3.7.1 Scenario Description

Scenario

Call the FlinkServer RESTful API to create tenants.

Data Preparation

- Prepare the files required for user authentication. Specifically, log in to FusionInsight Manager and download the **user.keytab** and **krb5.conf** files.
- Prepare information required for creating a tenant, for example, **tenantId** is **92**, **tenantName** is **test92**, and **remark** is **test tenant remark1**.
- If you run this sample program in Windows, add the host names and IP addresses of all nodes where FlinkServer resides to the **C:\Windows\System32\drivers\etc\hosts** file.

Development Guideline

1. Configure user authentication information.
2. Log in as a user.
3. Send requests.

10.3.7.2 Java Sample Code

Description

Call the FlinkServer RESTful API to create tenants.

Sample Code

For the complete code, see com.huawei.bigdata.flink.examples.TestCreateTenants.

```
public class TestCreateTenants {
    public static void main(String[] args) {
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        final String hostName = paraTool.get("hostName");    // Replace hostName in the hosts file with the actual host name.
        final String keytab = paraTool.get("keytab file path");        // user.keytab file path
        final String krb5 = paraTool.get("krb5 file path");            // krb5.conf file path
        final String principal = paraTool.get("Authentication username");  // Authentication user

        System.setProperty("java.security.krb5.conf", krb5);
        String url = "https://"+hostName+":28943/flink/v1/tenants";
        String jsonstr = "{" +
                "\n\t \"tenantId\":\"92\"," +
```

```
                "\n\t \"tenantName\":\"test92\"," +
                "\n\t \"remark\":\"test tenant remark1\"," +
                "\n\t \"updateUser\":\"test_updateUser1\"," +
                "\n\t \"createUser\":\"test_createUser1\"" +
                "\n}";

        try {
            LoginClient.getInstance().setConfigure(url, principal, keytab, "");
            LoginClient.getInstance().login();
            System.out.println(HttpClientUtil.doPost(url, jsonstr, "utf-8", true));
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

10.3.7.3 Accessing Flinkserver RESTful API as a Proxy User

This section applies to MRS 3.3.0 or later.

Function

Call the FlinkServer RESTful API as a proxy user. Use a proxy to access the API as a FlinkServer administrator to obtain common user permissions.

Sample Code

Assume that the tenant user is **test92**, the tenant ID is **92**, and the user name is **flinkserveradmin** with FlinkServer administrator permissions. The following code is a complete example.

```
public class TestCreateTenants {
    public static void main(String[] args) {
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        final String hostName = paraTool.get("hostName");    // Replace hostName in the hosts file with the actual host name.
        final String keytab = paraTool.get("keytab");        // user.keytab file path
        final String krb5 = paraTool.get("krb5");            // krb5.conf file path
        final String principal = paraTool.get("principal");  // Authentication user

        System.setProperty("java.security.krb5.conf", krb5);
        String url = "https://"+hostName+":28943/flink/v1/tenants";
        String jsonstr = "{" +
                "\n\t \"tenantId\":\"92\"," +
                "\n\t \"tenantName\":\"test92\"," +
                "\n\t \"remark\":\"test tenant remark1\"," +
                "\n\t \"updateUser\":\"test_updateUser1\"," +
                "\n\t \"createUser\":\"test_createUser1\"" +
                "\n}";

        try {
            LoginClient.getInstance().setConfigure(url, principal, keytab, "");
            LoginClient.getInstance().login(); // Log in as the FlinkServer administrator.

            String proxyUrl =  "https://"+hostName+":28943/flink/v1/proxyUserLogin"; // Call the proxy user API to obtain the common user token.
            String result = HttpClientUtil.doPost(proxyUrl, "{\n" +
                    "\t\"realUser\": \"flinkserveradmin\"\n" +
                    "}", "utf-8", true);
            Gson gson = new Gson();
            JsonObject jsonObject = gson.fromJson(result, JsonObject.class);
            String token = jsonObject.get("result").toString();
            token = "hadoop_auth=" + token;

            System.out.println(HttpClientUtil.doPost(url, jsonstr, "utf-8", true , token));
```



```
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

10.3.8 Flink Reading Data from and Writing Data to HBase

10.3.8.1 Scenario Description

This section applies to MRS 3.2.0 or later.

Typical Scenario Description

Use Flink API jobs to read data from and write data to HBase.

Data Preparation

Prepare the HBase configuration file and download the cluster configuration on FusionInsight Manager to obtain the **hbase-site.xml** file.

Development Guideline

1. Writes data to HBase:
   a. Specify the parent directory of the **hbase-site.xml** file. Flink Sink can obtain the HBase connection.
   b. Use the connection to determine whether a table exists. If it does not, create one.
   c. Convert received data into Put objects and writes the Put objects to HBase.
2. Reads data from HBase:
   a. Specify the parent directory of the **hbase-site.xml** file. Flink Source can obtain the HBase connection.
   b. Use the connection to determine whether a table exists. If it does not, the job fails. In this case, you need to create a table in HBase shell or an upstream job.
   c. Read data from HBase, converts result data into Row objects, and sends the Row objects to downstream operators.

10.3.8.2 Java Sample Code

Description

Call Flink APIs to read data from and write data to HBase.

Sample Code

The following example shows the main logic code of WriteHBase and ReadHBase.

For details about the complete code, see **com.huawei.bigdata.flink.examples.WriteHBase** and **com.huawei.bigdata.flink.examples.ReadHBase**.

- Main logic code of WriteHBase

```
public static void main(String[] args) throws Exception {
        System.out.println("use command as: ");
        System.out.println(
                "./bin/flink run --class com.huawei.bigdata.flink.examples.WriteHBase"
                        + " /opt/test.jar --tableName t1 --confDir /tmp/hbaseConf");
        System.out.println(
                "******************************************************************************************");
        System.out.println("<tableName> hbase tableName");
        System.out.println("<confDir> hbase conf dir");
        System.out.println(
                "******************************************************************************************");
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        env.setParallelism(1);
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        DataStream<Row> messageStream = env.addSource(new SimpleStringGenerator());
        messageStream.addSink(
                new HBaseWriteSink(paraTool.get("tableName"), createConf(paraTool.get("confDir"))));
        env.execute("WriteHBase");
    }

    private static org.apache.hadoop.conf.Configuration createConf(String confDir) {
        LOG.info("Create HBase configuration.");
        org.apache.hadoop.conf.Configuration hbaseConf =
HBaseConfigurationUtil.getHBaseConfiguration();
        if (confDir != null) {
            File hbaseSite = new File(confDir + File.separator + "hbase-site.xml");
            if (hbaseSite.exists()) {
                LOG.info("Add hbase-site.xml");
                hbaseConf.addResource(new Path(hbaseSite.getPath()));
            }
            File coreSite = new File(confDir + File.separator + "core-site.xml");
            if (coreSite.exists()) {
                LOG.info("Add core-site.xml");
                hbaseConf.addResource(new Path(coreSite.getPath()));
            }
            File hdfsSite = new File(confDir + File.separator + "hdfs-site.xml");
            if (hdfsSite.exists()) {
                LOG.info("Add hdfs-site.xml");
                hbaseConf.addResource(new Path(hdfsSite.getPath()));
            }
        }
        LOG.info("HBase configuration created successfully.");
        return hbaseConf;
    }

    /**
     * @since 8.2.0
     */
    private static class HBaseWriteSink extends RichSinkFunction<Row> {
        private Connection conn;
        private BufferedMutator bufferedMutator;
        private String tableName;
        private final byte[] serializedConfig;
        private Admin admin;
        private org.apache.hadoop.conf.Configuration hbaseConf;
        private long flushTimeIntervalMillis = 5000; //5s
        private long preFlushTime;

        public HBaseWriteSink(String sourceTable, org.apache.hadoop.conf.Configuration conf) {
            this.tableName = sourceTable;
            this.serializedConfig = HBaseConfigurationUtil.serializeConfiguration(conf);
        }

        private void deserializeConfiguration() {
```

```
            LOG.info("Deserialize HBase configuration.");
            hbaseConf = HBaseConfigurationUtil.deserializeConfiguration(
                serializedConfig, HBaseConfigurationUtil.getHBaseConfiguration());
            LOG.info("Deserialization successfully.");
        }

        private void createTable() throws IOException {
            LOG.info("Create HBase Table.");
            if (admin.tableExists(TableName.valueOf(tableName))) {
                LOG.info("Table already exists.");
                return;
            }
            // Specify the table descriptor.
            TableDescriptorBuilder htd =
TableDescriptorBuilder.newBuilder(TableName.valueOf(tableName));
            // Set the column family name to f1.
            ColumnFamilyDescriptorBuilder hcd =
ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("f1"));
            // 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
            hcd.setCompressionType(Compression.Algorithm.SNAPPY);
            htd.setColumnFamily(hcd.build());
            try {
                admin.createTable(htd.build());
            } catch (IOException e) {
                if (!(e instanceof TableExistsException)
                || !admin.tableExists(TableName.valueOf(tableName))) {
                    throw e;
                }
                LOG.info("Table already exists, ignore.");
            }
            LOG.info("Table created successfully.");
        }

        @Override
        public void open(Configuration parameters) throws Exception {
            LOG.info("Write sink open");
            super.open(parameters);
            deserializeConfiguration();
            conn = ConnectionFactory.createConnection(hbaseConf);
            admin = conn.getAdmin();
            createTable();
            bufferedMutator = conn.getBufferedMutator(TableName.valueOf(tableName));
            preFlushTime = System.currentTimeMillis();
        }

        @Override
        public void close() throws Exception {
            LOG.info("Close HBase Connection.");
            try {
                if (admin != null) {
                    admin.close();
                    admin = null;
                }
                if (bufferedMutator != null) {
                    bufferedMutator.close();
                    bufferedMutator = null;
                }
                if (conn != null) {
                    conn.close();
                    conn = null;
                }
            } catch (IOException e) {
                LOG.error("Close HBase Exception:", e);
                throw new RuntimeException(e);
            }
            LOG.info("Close successfully.");
```



```
    }

    @Override
    public void invoke(Row value, Context context) throws Exception {
        LOG.info("Write data to HBase.");
        Put put = new Put(Bytes.toBytes(value.getField(0).toString()));
        put.addColumn(Bytes.toBytes("f1"), Bytes.toBytes("q1"),
(Bytes.toBytes(value.getField(1).toString())));
        bufferedMutator.mutate(put);

        if (preFlushTime + flushTimeIntervalMillis >= System.currentTimeMillis()) {
            LOG.info("Flush data to HBase.");
            bufferedMutator.flush();
            preFlushTime = System.currentTimeMillis();
            LOG.info("Flush successfully.");
        } else {
            LOG.info("Skip Flush.");
        }

        LOG.info("Write successfully.");
    }
}

/**
 * @since 8.2.0
 */
public static class SimpleStringGenerator implements SourceFunction<Row> {
    private static final long serialVersionUID = 2174904787118597072L;
    boolean running = true;
    long i = 0;
    Random random = new Random();

    @Override
    public void run(SourceContext<Row> ctx) throws Exception {
        while (running) {
            Row row = new Row(2);
            row.setField(0, "rk" + random.nextLong());
            row.setField(1, "v" + random.nextLong());
            ctx.collect(row);
            Thread.sleep(1000);
        }
    }

    @Override
    public void cancel() {
        running = false;
    }
}
```

- Main logic code of ReadHBase

```
public static void main(String[] args) throws Exception {
    System.out.println("use command as: ");
    System.out.println(
            "./bin/flink run --class com.huawei.bigdata.flink.examples.ReadHBase"
                    + " /opt/test.jar --tableName t1 --confDir /tmp/hbaseConf");

    System.out.println(
            "*******************************************************************************************");
    System.out.println("<tableName> hbase tableName");
    System.out.println("<confDir> hbase conf dir");
    System.out.println(
            "*******************************************************************************************");
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    env.setParallelism(1);
    ParameterTool paraTool = ParameterTool.fromArgs(args);
    DataStream<Row> messageStream =
            env.addSource(
                    new HBaseReaderSource(
                            paraTool.get("tableName"), createConf(paraTool.get("confDir"))));
    messageStream
```

```
            .rebalance()
            .map(
                    new MapFunction<Row, String>() {
                        @Override
                        public String map(Row s) throws Exception {
                            return "Flink says " + s + System.getProperty("line.separator");
                        }
                    })
            .print();
        env.execute("ReadHBase");
    }

    private static org.apache.hadoop.conf.Configuration createConf(String confDir) {
        LOG.info("Create HBase configuration.");
        org.apache.hadoop.conf.Configuration hbaseConf =
HBaseConfigurationUtil.getHBaseConfiguration();
        if (confDir != null) {
            File hbaseSite = new File(confDir + File.separator + "hbase-site.xml");
            if (hbaseSite.exists()) {
                LOG.info("Add hbase-site.xml");
                hbaseConf.addResource(new Path(hbaseSite.getPath()));
            }
            File coreSite = new File(confDir + File.separator + "core-site.xml");
            if (coreSite.exists()) {
                LOG.info("Add core-site.xml");
                hbaseConf.addResource(new Path(coreSite.getPath()));
            }
            File hdfsSite = new File(confDir + File.separator + "hdfs-site.xml");
            if (hdfsSite.exists()) {
                LOG.info("Add hdfs-site.xml");
                hbaseConf.addResource(new Path(hdfsSite.getPath()));
            }
        }
        LOG.info("HBase configuration created successfully.");
        return hbaseConf;
    }

    private static class HBaseReaderSource extends RichSourceFunction<Row> {

        private Connection conn;
        private Table table;
        private Scan scan;
        private String tableName;
        private final byte[] serializedConfig;
        private Admin admin;
        private org.apache.hadoop.conf.Configuration hbaseConf;

        public HBaseReaderSource(String sourceTable, org.apache.hadoop.conf.Configuration conf) {
            this.tableName = sourceTable;
            this.serializedConfig = HBaseConfigurationUtil.serializeConfiguration(conf);
        }

        @Override
        public void open(Configuration parameters) throws Exception {
            LOG.info("Read source open");
            super.open(parameters);
            deserializeConfiguration();
            conn = ConnectionFactory.createConnection(hbaseConf);
            admin = conn.getAdmin();
            if (!admin.tableExists(TableName.valueOf(tableName))) {
                throw new IOException("table does not exist.");
            }
            table = conn.getTable(TableName.valueOf(tableName));
            scan = new Scan();
        }

        private void deserializeConfiguration() {
            LOG.info("Deserialize HBase configuration.");
            hbaseConf = HBaseConfigurationUtil.deserializeConfiguration(
```

```
            serializedConfig, HBaseConfigurationUtil.getHBaseConfiguration());
        LOG.info("Deserialization successfully.");
    }

    @Override
    public void run(SourceContext<Row> sourceContext) throws Exception {
        LOG.info("Read source run");
        try (ResultScanner scanner = table.getScanner(scan)) {
            Iterator<Result> iterator = scanner.iterator();
            while (iterator.hasNext()) {
                Result result = iterator.next();
                String rowKey = Bytes.toString(result.getRow());
                byte[] value = result.getValue(Bytes.toBytes("f1"), Bytes.toBytes("q1"));
                Row row = new Row(2);
                row.setField(0, rowKey);
                row.setField(1, Bytes.toString(value));
                sourceContext.collect(row);
                LOG.info("Send data successfully.");
            }
        }

        LOG.info("Read successfully.");
    }

    @Override
    public void close() throws Exception {
        closeHBase();
    }

    private void closeHBase() {
        LOG.info("Close HBase Connection.");
        try {
            if (admin != null) {
                admin.close();
                admin = null;
            }
            if (table != null) {
                table.close();
                table = null;
            }
            if (conn != null) {
                conn.close();
                conn = null;
            }
        } catch (IOException e) {
            LOG.error("Close HBase Exception:", e);
            throw new RuntimeException(e);
        }
        LOG.info("Close successfully.");
    }

    @Override
    public void cancel() {
        closeHBase();
    }
}
```

10.3.9 Flink Reading Data from and Writing Data to Hudi

10.3.9.1 Scenario Description

This section applies to MRS 3.3.0 or later.

Typical Scenario Description

In this example, the job generates one data record per second, writes the data to the Hudi table, and reads and prints the data in the Hudi table.

Development Guideline

1. Write data to Hudi:
   a. Generate data through a random data generation class.
   b. Convert the generated data into **DataStream<RowData>**.
   c. Write data to the Hudi table.
2. Read data from Hudi:
   a. Read data from the Hudi table.
   b. Combine the read data into the JSON format and print the data.

10.3.9.2 Java Sample Code

Description

Call Flink APIs to read data from and write data to Hudi.

Sample Code

The following example shows the main logic code of WriteIntoHudi and ReadFromHudi.

For details about the complete code, see **com.huawei.bigdata.flink.examples.WriteIntoHudi** and **com.huawei.bigdata.flink.examples.ReadFromHudi**.

- Main logic code of WriteIntoHudi

```
public class WriteIntoHudi {
    public static void main(String[] args) throws Exception {
        System.out.println("use command as: ");
        System.out.println(
                "./bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.WriteIntoHudi"
                        + " /opt/test.jar --hudiTableName hudiSinkTable --hudiPath hdfs://hacluster/tmp/
flinkHudi/hudiTable");
        System.out.println(
                "********************************************************************************");
        System.out.println("<hudiTableName> is the hudi table name. (Default value is hudiSinkTable)");
        System.out.println("<hudiPath> Base path for the target hoodie table. (Default value is hdfs://
hacluster/tmp/flinkHudi/hudiTable)");
        System.out.println(
                "********************************************************************************");

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        env.setParallelism(1);
        env.getCheckpointConfig().setCheckpointInterval(10000);
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        DataStream<RowData> stringDataStreamSource = env.addSource(new SimpleStringGenerator())
                .map(new MapFunction<Tuple5<String, String, Integer, String, String>, RowData>() {
                    @Override
                    public RowData map(Tuple5<String, String, Integer, String, String> tuple5) throws
Exception {
                        GenericRowData rowData = new GenericRowData(5);
                        rowData.setField(0, StringData.fromString(tuple5.f0));
                        rowData.setField(1, StringData.fromString(tuple5.f1));
```



```
                    rowData.setField(2, tuple5.f2);
                    rowData.setField(3, TimestampData.fromTimestamp(Timestamp.valueOf(tuple5.f3)));
                    rowData.setField(4, StringData.fromString(tuple5.f4));
                    return rowData;
                }
            });
        String basePath = paraTool.get("hudiPath", "hdfs://hacluster/tmp/flinkHudi/hudiTable");
        String targetTable = paraTool.get("hudiTableName", "hudiSinkTable");
        Map<String, String> options = new HashMap<>();
        options.put(FlinkOptions.PATH.key(), basePath);
        options.put(FlinkOptions.TABLE_TYPE.key(), HoodieTableType.MERGE_ON_READ.name());
        options.put(FlinkOptions.PRECOMBINE_FIELD.key(), "ts");
        options.put(FlinkOptions.INDEX_BOOTSTRAP_ENABLED.key(), "true");
        HoodiePipeline.Builder builder = HoodiePipeline.builder(targetTable)
                .column("uuid VARCHAR(20)")
                .column("name VARCHAR(10)")
                .column("age INT")
                .column("ts TIMESTAMP(3)")
                .column("p VARCHAR(20)")
                .pk("uuid")
                .partition("p")
                .options(options);
        builder.sink(stringDataStreamSource, false); // The second parameter indicating whether the
input data stream is bounded
        env.execute("Hudi_Sink");
    }
    public static class SimpleStringGenerator implements SourceFunction<Tuple5<String, String,
Integer, String, String>> {
        private static final long serialVersionUID = 2174904787118597072L;
        boolean running = true;
        Integer i = 0;

        @Override
        public void run(SourceContext<Tuple5<String, String, Integer, String, String>> ctx) throws
Exception {
            while (running) {
                i++;
                String uuid = "uuid" + i;
                String name = "name" + i;
                Integer age = new Integer(i);
                String ts = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss"));
                String p = "par" + i % 5;
                Tuple5<String, String, Integer, String, String> tuple5 = Tuple5.of(uuid, name, age, ts, p);
                ctx.collect(tuple5);
                Thread.sleep(1000);
            }
        }
        @Override
        public void cancel() {
            running = false;
        }
    }
}
```

- Main logic code of ReadFromHudi

```
public class ReadFromHudi {
    public static void main(String[] args) throws Exception {
        System.out.println("use command as: ");
        System.out.println(
                "./bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.ReadFromHudi"
                        + " /opt/test.jar --hudiTableName hudiSourceTable --hudiPath hdfs://hacluster/tmp/
flinkHudi/hudiTable"
                        + " --read.start-commit 20221206111532"
        );
        System.out.println(
                "*****************************************************************************************");
        System.out.println("<hudiTableName> is the hoodie table name. (Default value is
hudiSourceTable)");
        System.out.println("<hudiPath> Base path for the target hoodie table. (Default value is hdfs://
```

```
hacluster/tmp/flinkHudi/hudiTable)");
        System.out.println("<read.start-commit> Start commit instant for reading, the commit time
format should be 'yyyyMMddHHmmss'. (Default value is earliest)");
        System.out.println(
                "''************************************************************************************");

        ParameterTool paraTool = ParameterTool.fromArgs(args);
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        String basePath = paraTool.get("hudiPath", "hdfs://hacluster/tmp/flinkHudi/hudiTable");
        String targetTable = paraTool.get("hudiTableName", "hudiSourceTable");
        String startCommit = paraTool.get(FlinkOptions.READ_START_COMMIT.key(),
FlinkOptions.START_COMMIT_EARLIEST);
        Map<String, String> options = new HashMap();
        options.put(FlinkOptions.PATH.key(), basePath);
        options.put(FlinkOptions.TABLE_TYPE.key(), HoodieTableType.MERGE_ON_READ.name());
        options.put(FlinkOptions.READ_AS_STREAMING.key(), "true"); // this option enable the
streaming read
        options.put(FlinkOptions.READ_START_COMMIT.key(), startCommit); // specifies the start
commit instant time
        HoodiePipeline.Builder builder = HoodiePipeline.builder(targetTable)
                .column("uuid VARCHAR(20)")
                .column("name VARCHAR(10)")
                .column("age INT")
                .column("ts TIMESTAMP(3)")
                .column("p VARCHAR(20)")
                .pk("uuid")
                .partition("p")
                .options(options);

        DataStream<RowData> rowDataDataStream = builder.source(env);
        rowDataDataStream.map(new MapFunction<RowData, String>() {
            @Override
            public String map(RowData rowData) throws Exception {
                StringBuilder sb = new StringBuilder();
                sb.append("{");
                sb.append("\"uuid\":\"").append(rowData.getString(0)).append("\",");
                sb.append("\"name\":\"").append(rowData.getString(1)).append("\",");
                sb.append("\"age\":").append(rowData.getInt(2)).append(",");
                sb.append("\"ts\":\"").append(rowData.getTimestamp(3, 0)).append("\",");
                sb.append("\"p\":\"").append(rowData.getString(4)).append("\"");
                sb.append("}");
                return sb.toString();
            }
        }).print();
        env.execute("Hudi_Source");
    }
}
```

10.3.10 Python Development Examples

10.3.10.1 Submitting a Regular Job Using Python

10.3.10.1.1 Description

This scenario applies to MRS 3.3.0 or later.

Assume that you need to submit a Flink task to an MRS cluster. The main language used by the service platform is Python. The following content uses an example Python program to read and writ Kafka jobs.

10.3.10.1.2 Python Sample Code

Function

Submit Flink Kafka read and write jobs to Yarn through Python APIs.

Sample Code

The main logic code in **pyflink-kafka.py** is provided. Before submitting the code, ensure that **file_path** is the path where the SQL statement to be executed. You are advised to use a full path.

For details about the complete code, see **pyflink-kafka.py** in **flink-examples/pyflink-example/pyflink-kafka**.

```
import os
import logging
import sys
from pyflink.common import JsonRowDeserializationSchema, JsonRowSerializationSchema
from pyflink.common.typeinfo import Types
from pyflink.datastream.connectors import FlinkKafkaProducer, FlinkKafkaConsumer
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import TableEnvironment, EnvironmentSettings
def read_sql(file_path):
    if not os.path.isfile(file_path):
        raise TypeError(file_path + " does not exist")
    all_the_text = open(file_path).read()
    return all_the_text
def exec_sql():
    # Change the SQL path before job submission.
    # file_path = "/opt/client/Flink/flink/insertData2kafka.sql"
    # file_path = os.getcwd() + "/../../../../yarnship/insertData2kafka.sql"
    # file_path = "/opt/client/Flink/flink/conf/ssl/insertData2kafka.sql"
    file_path = "insertData2kafka.sql"
    sql = read_sql(file_path)
    t_env = TableEnvironment.create(EnvironmentSettings.in_streaming_mode())
    statement_set = t_env.create_statement_set()
    sqlArr = sql.split(";")
    for sqlStr in sqlArr:
        sqlStr = sqlStr.strip()
        if sqlStr.lower().startswith("create"):
            print("---------create----------")
            print(sqlStr)
            t_env.execute_sql(sqlStr)
        if sqlStr.lower().startswith("insert"):
            print("---------insert----------")
            print(sqlStr)
            statement_set.add_insert_sql(sqlStr)
    statement_set.execute()
def read_write_kafka():
    # find kafka connector jars
    env = StreamExecutionEnvironment.get_execution_environment()
    env.set_parallelism(1)
    specific_jars = "file:///opt/client/Flink/flink/lib/flink-connector-kafka-xxx.jar"
    # specific_jars = "file://" + os.getcwd() + "/../../../../yarnship/flink-connector-kafka-xxx.jar"
    # specific_jars = "file:///opt/client/Flink/flink/conf/ssl/flink-connector-kafka-xxx.jar"
    # the sql connector for kafka is used here as it's a fat jar and could avoid dependency issues
    env.add_jars(specific_jars)
    kafka_properties = {'bootstrap.servers': '192.168.20.162:21005', 'group.id': 'test_group'}
    deserialization_schema = JsonRowDeserializationSchema.builder() \
        .type_info(type_info=Types.ROW([Types.INT(), Types.STRING()])).build()
    kafka_consumer = FlinkKafkaConsumer(
        topics='test_source_topic',
        deserialization_schema=deserialization_schema,
        properties=kafka_properties)
    print("---------read ----------")
    ds = env.add_source(kafka_consumer)
```



```
    serialization_schema = JsonRowSerializationSchema.builder().with_type_info(
        type_info=Types.ROW([Types.INT(), Types.STRING()])).build()
    kafka_producer = FlinkKafkaProducer(
        topic='test_sink_topic',
        serialization_schema=serialization_schema,
        producer_config=kafka_properties)
    print("--------write------------------")
    ds.add_sink(kafka_producer)
    env.execute("pyflink kafka test")
if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    print("------------------insert data to kafka----------------")
    exec_sql()
    print("------------------read_write_kafka----------------")
    read_write_kafka()
```

Table 10-10 Parameters for submitting a regular job using Python

Parameter	Description	Example
bootstrap.servers	Service IP address and port number of the Broker instance of Kafka	192.168.12.25:21005
specific_jars	Package path: *Client installation directory*/**Flink/flink/lib/flink-connector-kafka-*.jar**. You are advised to use a full path. **NOTE** If a job needs to be submitted as yarn-application, replace the following path. Replace the JAR package version with the actual one. specific_jars="file://"+os.getcwd()+"/../../../../yarnship/flink-connector-kafka-1.15.0-h0.cbu.mrs.330.r13.jar"	specific_jars = file:///*Client installation directory*/Flink/flink/lib/flink-connector-kafka-1.15.0-h0.cbu.mrs.330.r13.jar
file_path	Path of the **insertData2kafka.sql** file. You are advised to use a full path. Obtain the package from the secondary development sample code and upload it to the specified directory on the client. **NOTE** If a job needs to be submitted as yarn-application, replace the following path: file_path = os.getcwd() + "/../../../../yarnship/insertData2kafka.sql"	file_path = /*Client installation directory*/Flink/flink/insertData2kafka.sql

The following is an example SQL statement:

```
create table kafka_sink_table (
  age int,
  name varchar(10)
) with (
  'connector' = 'kafka',
'topic' = 'test_source_topic', --Name of the topic written to Kafka. Ensure that the topic name is the same as that in the Python file.
  'properties.bootstrap.servers' = 'IP address of the Kafka broker instance:Kafka port number',
  'properties.group.id' = 'test_group',
  'format' = 'json'
```



```
);
create TABLE datagen_source_table (
   age int,
  name varchar(10)
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
);
INSERT INTO
  kafka_sink_table
SELECT
  *
FROM
  datagen_source_table;
```

10.3.10.1.3 Running the Program

Step 1 Obtain **pyflink-kafka.py** and **insertData2kafka.sql** from the sample project **flink-examples/pyflink-example/pyflink-kafka**.

Step 2 Package the prepared Python virtual environment by referring to **Preparing for Development and Operating Environment** and obtain the **venv.zip** file.

```
zip -q -r venv.zip venv/
```

Step 3 Log in to the active management node as the **root** user and upload **venv.zip**, **pyflink-kafka.py**, and **insertData2kafka.sql** files obtained in **Step 1**and **Step 2** to the client environment.

- Per-job: Upload the preceding files to *Client installation directory*/**Flink/flink**.
- yarn-application: Upload the preceding files and the **flink-connector-kafka-***Actual version number*.**jar** package to *Client installation directory*/**Flink/flink/yarnship**.

Step 4 Change the **specific_jars** path in **pyflink-kafka.py**.

- per-job: Change the path to the actual path of the SQL file, for example, **file:///***Client installation directory*/**Flink/flink/lib/flink-connector-kafka-***Actual version number*.**jar**.
- yarn-application: Change to **file://"+os.getcwd()+"/../../../../yarnship/flink-connector-kafka-***Actual version number*.**jar**.

Step 5 Change **file_path** in **pyflink-kafka.py**.

- per-job: Change the path to the actual path of the SQL file. For example: *Client installation directory*/**Flink/flink/insertData2kafka.sql**
- yarn-application: Change the path to **os.getcwd () + "/../../../../yarnship/insertData2kafka.sql"**

Step 6 Run the following command to specify the running environment:

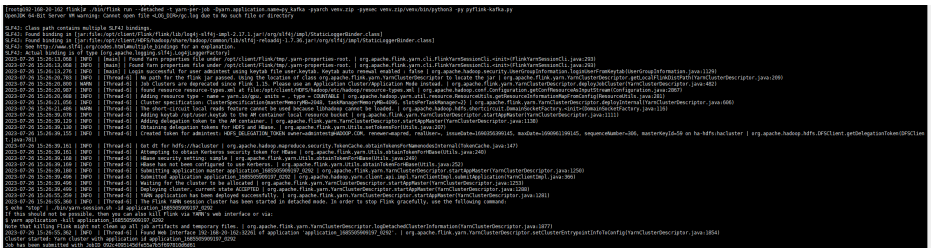
```
export PYFLINK_CLIENT_EXECUTABLE=venv.zip/venv/bin/python3
```

Step 7 Run the following command to run the program:

- Per-job:

  ```
  ./bin/flink run --detached -t yarn-per-job -Dyarn.application.name=py_kafka -pyarch venv.zip -pyexec venv.zip/venv/bin/python3 -py pyflink-kafka.py
  ```

  Execution result:



- yarn-application
  ./bin/flink run-application --detached -t yarn-application -Dyarn.application.name=py_kafka -Dyarn.ship-files=/opt/client/Flink/flink/yarnship/ -pyarch yarnship/venv.zip -pyexec venv.zip/venv/bin/python3 -pyclientexec venv.zip/venv/bin/python3 -pyfs yarnship -pym pyflink-kafka

  Execution result:

----End

10.3.10.2 Submitting a SQL Job Using Python

10.3.10.2.1 Description

Assume that you need to submit Flink tasks to the MRS cluster, the main language used by the service platform is Python, and core service processing requires SQL. The following content provides an example to describe how to submit a SQL job using Python.

10.3.10.2.2 Python Sample Code

Function

Submit a Flink SQL job to Yarn through Python APIs.

Sample Code

The main logic code in **pyflink-sql.py** is provided. Before submitting the code, ensure that **file_path** is the path where the SQL statement to be executed. You are advised to use a full path.

For details about the complete code, see **pyflink-sql.py** in **flink-examples/pyflink-example/pyflink-sql**.

```
import logging
import sys
import os
from pyflink.table import (EnvironmentSettings, TableEnvironment)
def read_sql(file_path):
    if not os.path.isfile(file_path):
        raise TypeError(file_path + " does not exist")
    all_the_text = open(file_path).read()
    return all_the_text
def exec_sql():
```

```
    # Change the SQL path before job submission.
    file_path = "datagen2kafka.sql"
    sql = read_sql(file_path)
    t_env = TableEnvironment.create(EnvironmentSettings.in_streaming_mode())
    statement_set = t_env.create_statement_set()
    sqlArr = sql.split(";")
    for sqlStr in sqlArr:
        sqlStr = sqlStr.strip()
        if sqlStr.lower().startswith("create"):
            print("---------create----------")
            print(sqlStr)
            t_env.execute_sql(sqlStr)
        if sqlStr.lower().startswith("insert"):
            print("---------insert----------")
            print(sqlStr)
            statement_set.add_insert_sql(sqlStr)
    statement_set.execute()
if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    exec_sql()
```

Table 10-11 Parameters for submitting a SQL job using Python

Parameter	Description	Example
file_path	Path of the **datagen2kafka.sql** file. You are advised to use a full path. Obtain the package from the secondary development sample code and upload it to the specified directory on the client. **NOTE** If a job needs to be submitted as yarn-application, replace the following path: file_path = os.getcwd() + "/../../../../yarnship/datagen2kafka.sql"	file_path = /*Client installation directory*/Flink/flink/datagen2kafka.sql

The following is an example SQL statement:

```
create table kafka_sink (
  uuid varchar(20),
  name varchar(10),
  age int,
  ts timestamp(3),
  p varchar(20)
) with (
  'connector' = 'kafka',
  'topic' = 'input2',
  'properties.bootstrap.servers' = 'IP address of the Kafka broker instance:Kafka port number',
  'properties.group.id' = 'testGroup2',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);
create TABLE datagen_source (
  uuid varchar(20),
  name varchar(10),
  age int,
  ts timestamp(3),
  p varchar(20)
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
```



```
);
INSERT INTO
  kafka_sink
SELECT
  *
FROM
  datagen_source;
```

10.3.10.2.3 Running the Program

Step 1 Obtain **pyflink-sql.py** and **datagen2kafka.sql** from the sample project **flink-examples/pyflink-example/pyflink-sql**.

Step 2 Package the prepared Python virtual environment by referring to **Preparing for Development and Operating Environment** and obtain the **venv.zip** file.

```
zip -q -r venv.zip venv/
```

Step 3 Log in to the active management node as the **root** user and upload **venv.zip**, **pyflink-sql.py**, and **datagen2kafka.sql** files obtained in **Step 1**and **Step 2** to the client environment.

- Per-job: Upload the preceding files to *Client installation directory*/**Flink/flink**.
- yarn-application: Upload the preceding files to *Client installation directory*/**Flink/flink/yarnship**.
- yarn-session: Upload the preceding files to *Client installation directory*/**Flink/flink/conf/ssl**.

Step 4 Change **file_path** in **pyflink-sql.py**.

- per-job: Change the path to the actual path of the SQL file. For example: *Client installation directory*/**Flink/flink/datagen2kafka.sql**
- yarn-application: Change the path to **os.getcwd() + "/../../../../yarnship/datagen2kafka.sql"**
- yarn-session: Change the path to the actual path of the SQL file. For example: *Client installation directory*/**Flink/flink/conf/ssl//datagen2kafka.sql**

Step 5 Run the following command to specify the running environment:

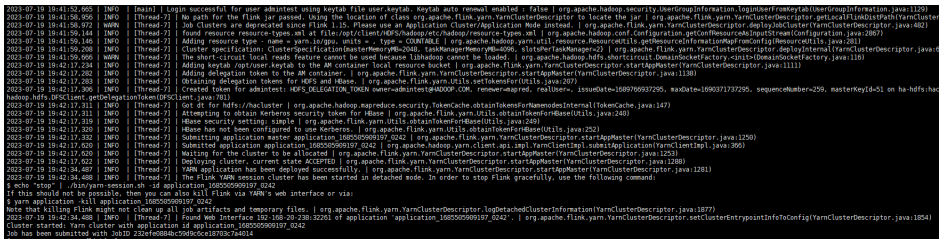
```
export PYFLINK_CLIENT_EXECUTABLE=venv.zip/venv/bin/python3
```

Step 6 Run the following command to run the program:

- Per-job:

  ```
  ./bin/flink run --detached -t yarn-per-job -Dyarn.application.name=py_sql -pyarch venv.zip -pyexec venv.zip/venv/bin/python3 -py pyflink-sql.py
  ```

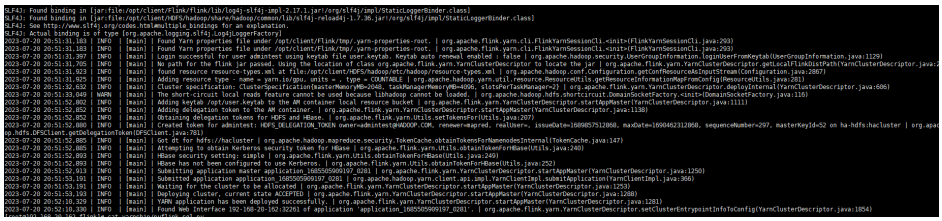
  Execution result:

- yarn-application

  ```
  ./bin/flink run-application --detached -t yarn-application -Dyarn.application.name=py_sql -Dyarn.ship-files=/opt/client/Flink/flink/yarnship/ -pyarch yarnship/venv.zip -pyexec venv.zip/venv/bin/python3 -pyclientexec venv.zip/venv/bin/python3 -pyfs yarnship -pym pyflink-sql
  ```

  Execution result:



- yarn-session

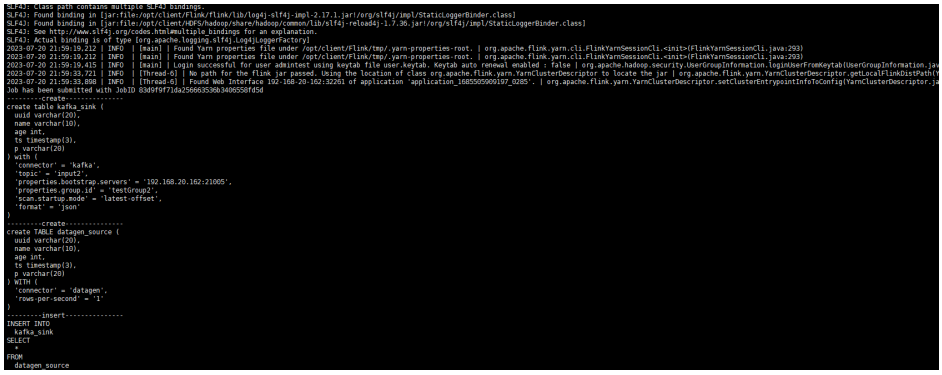
  Before starting the Yarn session, prepare the running environment by referring to **Preparing for Development and Operating Environment**. Run the following command to start yarn-session:

  ```
  bin/yarn-session.sh -jm 1024 -tm 4096 -t conf/ssl/ -d
  ```

  Run the following command to submit the job:

  ```
  ./bin/flink run --detached -t yarn-session -Dyarn.application.name=py_sql -Dyarn.application.id=application_1685505909197_0285 -pyarch conf/ssl/venv.zip -pyexec conf/ssl/venv.zip/venv/bin/python3 -py conf/ssl/pyflink-sql.py
  ```

  Execution result:



----End

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.

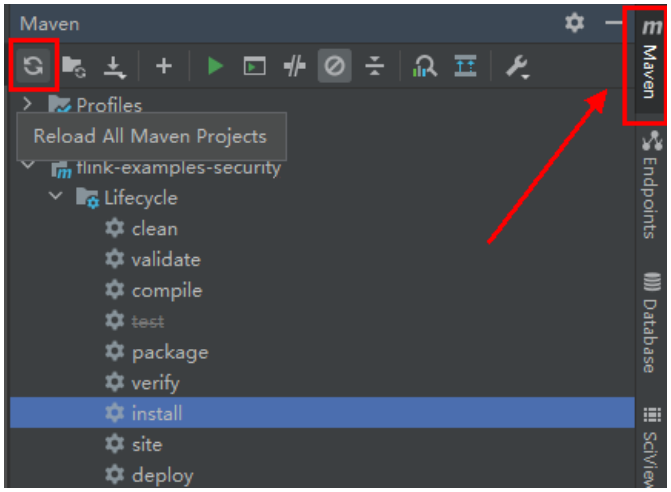
NOTE

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, click **Reload All Maven Projects** in the Maven window on the right of IDEA to import Maven project dependencies.

Figure 10-42 Reload projects

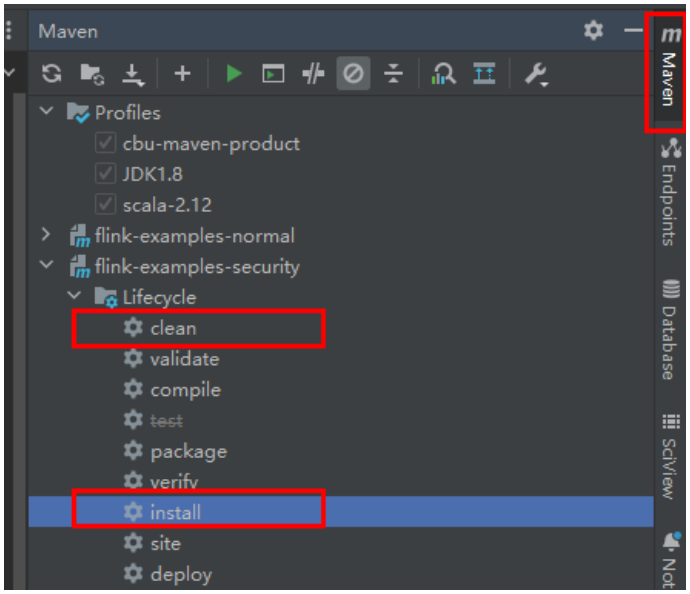


Step 2 Compile and run the application.

Use either of the following two methods:

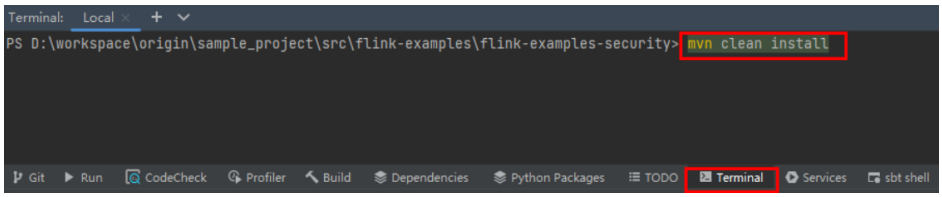
- Method 1:
  a. Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.
  b. Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

  Figure 10-43 Maven clean and install



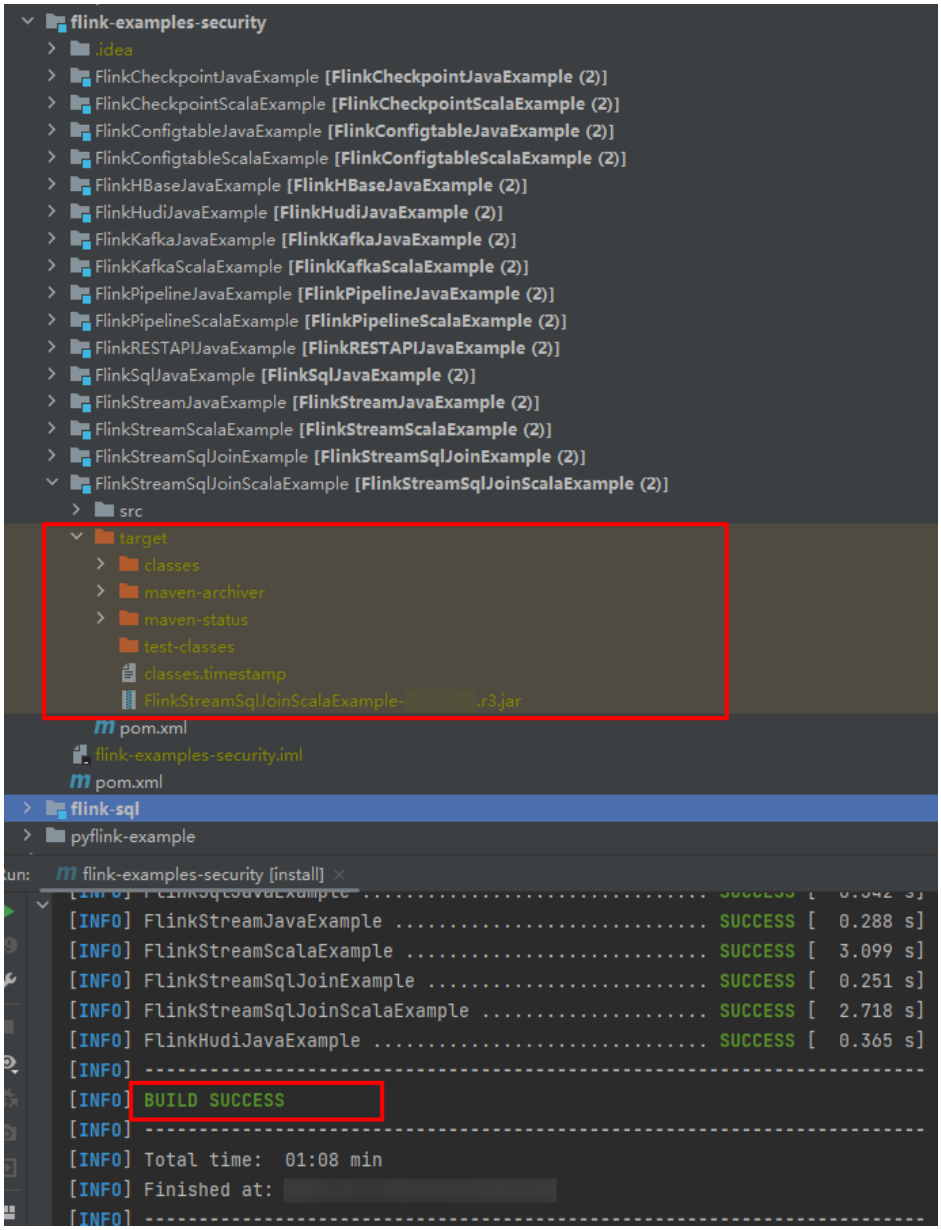
- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

Figure 10-44 Entering **mvn clean install** in the IDEA Terminal text box



Step 3 After the compilation is complete, the message "BUILD SUCCESS" is printed and the **target** directory is generated. Obtain the JAR package in the directory.

Figure 10-45 Compilation completed



Step 4 Copy the **.jar** package (for example **FlinkStreamJavaExample.jar**) created in **Step 3** 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 4096 -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 about the dependency package for running the sample project**.
- 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.
- The following applies to MRS 3.2.1 or later. The memory size of TaskManagers specified by using the **-tm** command must be at least 4,096 MB.

- 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 10-12 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] [sasl.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] [sasl.kerberos.service.name]
  [ssl.truststore.location] [ssl.truststore.password] [kerberos.domain.name]
  ```

  Table 10-13 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

Parameter	Description	Mandatory (Yes/No)
security.protocol	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. – 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 **hadoop**. *System domain name* and the **security.kerberos.login** parameter in **conf/flink-conf.yaml** are mandatory. **NOTE** You can log in to Manager, choose **System** > **Permission** > **Domain and Mutual Trust** > **Local Domain**, can view the system domain name. All letters in the system domain name must be converted to lowercase letters. – If the SSL is configured, **ssl.truststore.location** (path of **truststore**) and **ssl.truststore.password** (password of **truststore**) must be set.	No **NOTE** – 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, System domain name is **HADOOP.COM**:

- 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 xxx
  ```
- 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 xxx --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
  - Java
    i. 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
    ```
    ii. Run the **netcat** command on any node in the cluster to wait for the connection of the application.
    ```
    netcat -l -p 9000
    ```
    iii. 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
    ```
  - Scala (for MRS 3.3.0 or later)

i. Start the application to produce data in Kafka. For details about how to configure Kafka, see **• Run the following code to....**.

```
bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/client/FlinkStreamSqlJoinScalaExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.xxx:9092
```

ii. Run the **netcat** command on any node in the cluster to wait for an application connection.

```
netcat -l -p 9000
```

iii. Start the application to receive socket data and perform a joint query.

```
bin/flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket /opt/client/FlinkStreamSqlJoinScalaExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.xxx:9092 --hostname xxx.xxx.xxx.xxx --port 9000
```

- Running the Flink HBase sample program(MRS 3.2.0 or later clusters.)
  - yarn-session
    i. Start the flink cluster.

    ```
    ./bin/yarn-session.sh -t config -jm 1024 -tm 1024
    ```

    ii. Run the Flink program and set parameters.

    ```
    bin/flink run --class com.huawei.bigdata.flink.examples.WriteHBase /opt/client1/Flink/flink/FlinkHBaseJavaExample-xxx.jar  --tableName xxx --confDir xxx

    bin/flink run --class com.huawei.bigdata.flink.examples.ReadHBase /opt/client1/Flink/flink/FlinkHBaseJavaExample-xxx.jar  --tableName xxx --confDir xxx
    ```

  - yarn-cluster

    ```
    bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.WriteHBase /opt/client1/Flink/flink/FlinkHBaseJavaExample-xxx.jar  --tableName xxx --confDir xxx

    bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.ReadHBase /opt/client1/Flink/flink/FlinkHBaseJavaExample-xxx.jar  --tableName xxx --confDir xxx
    ```

- Running the Flink Hudi sample application (MRS 3.2.1 or later).
  - yarn-session mode
    i. Start the Flink cluster.

    ```
    ./bin/yarn-session.sh -t config -jm 1024 -tm 4096
    ```

    ii. Run the Flink application and enter parameters.

    ```
    ./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoHudi /opt/test.jar --hudiTableName hudiSinkTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable

    ./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromHudi /opt/test.jar --hudiTableName hudiSourceTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable --read.start-commit xxx
    ```

  - yarn-cluster mode

    ```
    ./bin/flink run -m yarn-cluster -yt config --class com.huawei.bigdata.flink.examples.WriteIntoHudi /opt/test.jar --hudiTableName hudiSinkTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable

    ./bin/flink run -m yarn-cluster -yt config --class com.huawei.bigdata.flink.examples.ReadFromHudi /opt/test.jar --hudiTableName hudiSourceTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable --read.start-commit xxx
    ```

- Running the REST APIs for tenant sample program creation (The TestCreateTenants program is used as an example.)
  - yarn-session mode
    i. Start the Flink cluster.

    ```
    ./bin/yarn-session.sh -t config -jm 1024 -tm 4096
    ```

    ii. Run the Flink program and enter parameters.

    ```
    ./bin/flink run --class com.huawei.bigdata.flink.examples.TestCreateTenants /opt/client/FlinkRESTAPIJavaExample-xxx.jar --hostName xx-xx-xx-xx --keytab xx/xx/user.keytab --krb5 xx/xx/krb5.conf --principal username
    ```

  - yarn-cluster mode
    ```
    ./bin/flink run -m yarn-cluster -yt config -yjm 1024 -ytm 4096 --class com.huawei.bigdata.flink.examples.TestCreateTenants /opt//opt/client/FlinkRESTAPIJavaExample-xxx.jar --hostName xx-xx-xx-xx --keytab xx/xx/user.keytab --krb5 xx/xx/krb5.conf --principal username
    ```

- Run a SQL task to submit Flink JAR jobs (applicable to MRS 3.2.1 or later).
  - yarn-session mode
    i. Start the Flink cluster.
    ```
    ./bin/yarn-session.sh -t config -jm 1024 -tm 4096
    ```
    ii. Run the Flink application and enter parameters.
    ```
    bin/flink run -d --class com.huawei.mrs.FlinkSQLExecutor /opt/flink-sql-xxx.jar --sql ./sql/datagen2kafka.sql
    ```
  - yarn-cluster mode
    ```
    bin/flink run -m yarn-cluster -yt config -yjm 1024 -ytm 4096 -d --class com.huawei.mrs.FlinkSQLExecutor /opt/flink-sql-xxx.jar --sql ./sql/datagen2kafka.sql
    ```

----End

NOTE

For details about sample projects of dependency packages provided by Flink, see **Reference information about the dependency package for running the sample project**.

If HA is enabled for Flink (MRS 3.2.0 or later), the value of **--hostName** in the RESTAPI tenant creation example is the floating IP address of FlinkServer.

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 Task Manager 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 10-46**. Choose **Task Managers** > **Stdout** to view the running result, as shown in**Figure 10-47**.

      Figure 10-46 submitted jobs

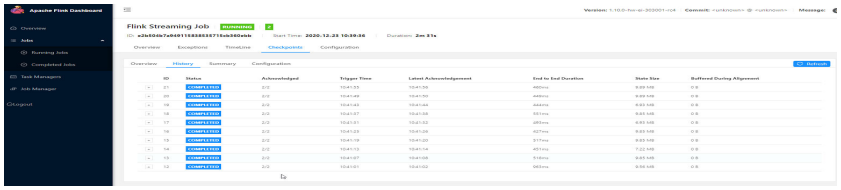
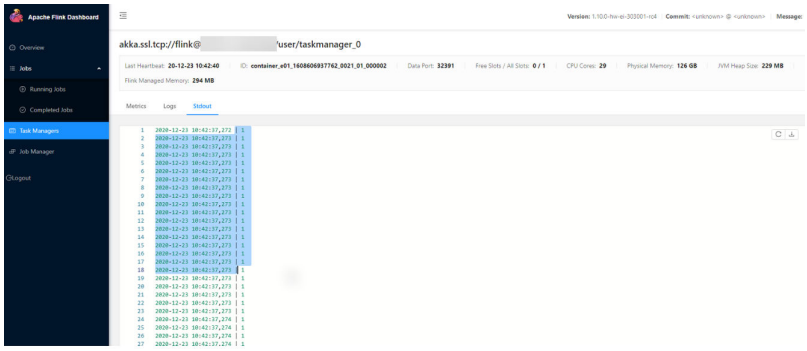
      

Figure 10-47 execution result



- 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 10-48**. Choose **Task Managers**. You can see two tasks, as shown in **Figure 10-49**. Click any task, choose **Stdout** to view the output of the task, as shown in **Figure 10-50** and **Figure 10-51**.

Figure 10-48 running jobs

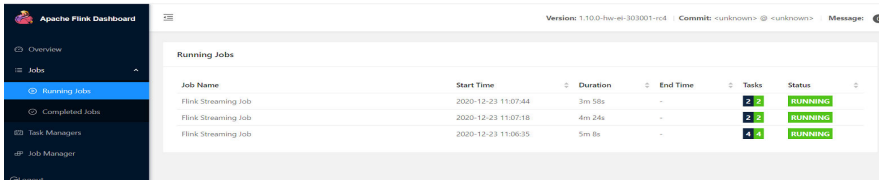


Figure 10-49 submitted Tasks

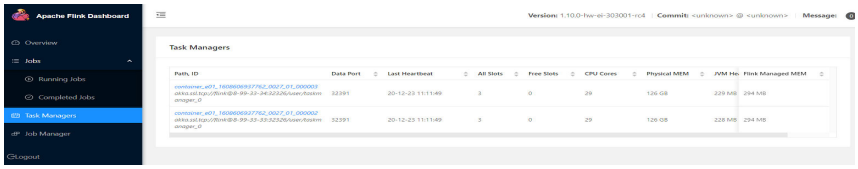


Figure 10-50 output of task1

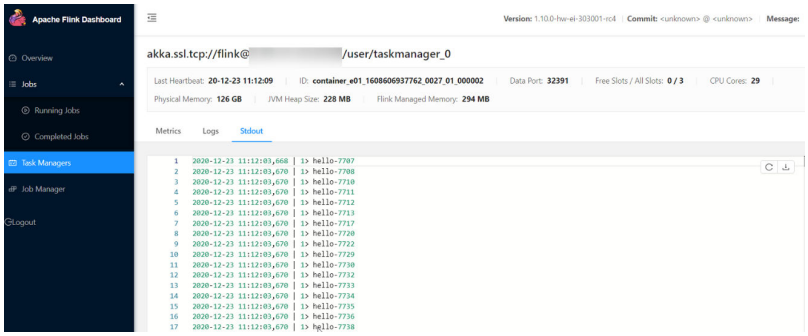
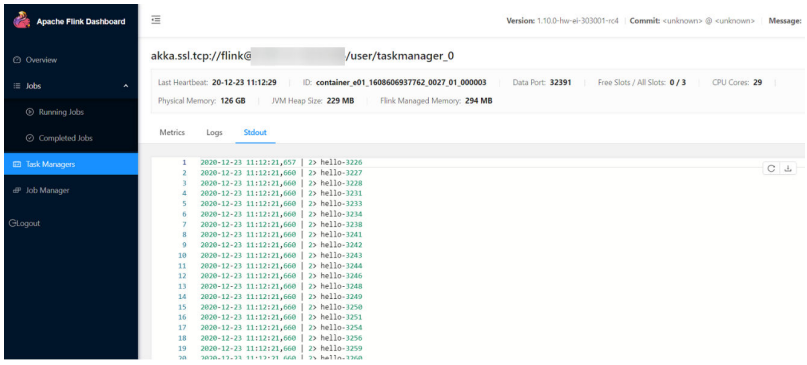


Figure 10-51 output of task2



– **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 10-52**. Choose **Task Managers**, you can see submitted task, as shown in **Figure 10-53**. Choose **Stdout** to view the running result, as shown in **Figure 10-54**.

Figure 10-52 completed job

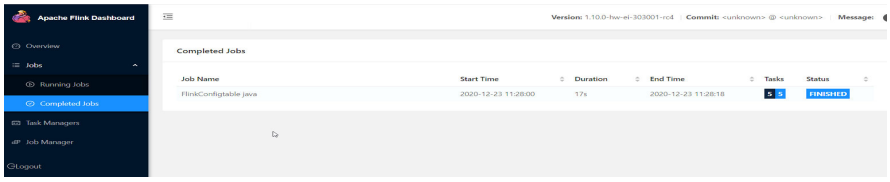


Figure 10-53 submitted task

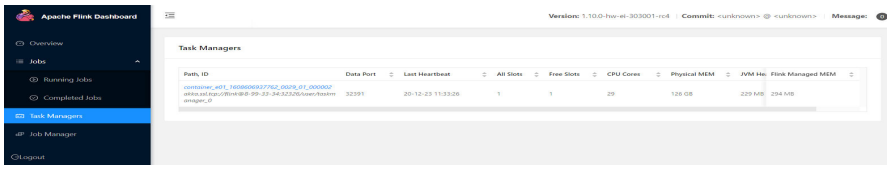
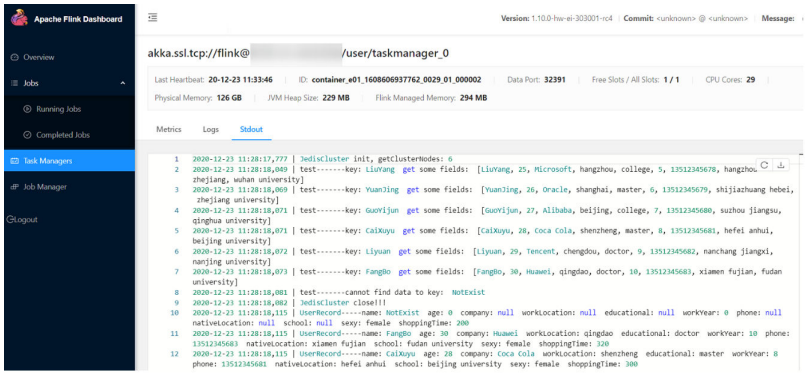


Figure 10-54 execution result



- **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 10-55**. Choose **Task Managers**, you can see submitted task, as shown in **Figure 10-56**. Choose **Stdout** to view the running result, as shown in **Figure 10-57**.

    Figure 10-55 completed job

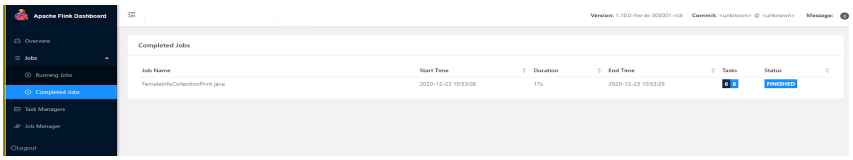
    

    Figure 10-56 submitted task

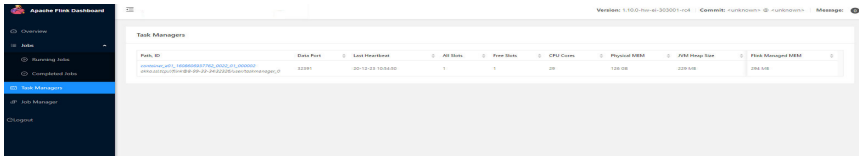
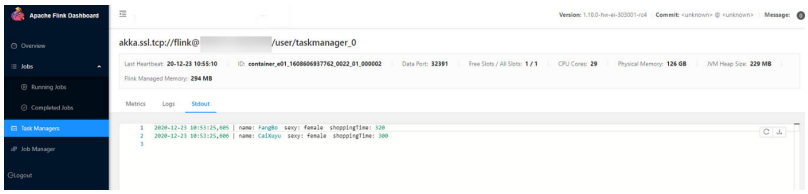
    

    Figure 10-57 execution result

- **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 10-58**. Choose **Task Managers**, you can see submitted task, as shown in **Figure 10-59**. Choose **Stdout** to view the running result, as shown in **Figure 10-60**.

Figure 10-58 running jobs

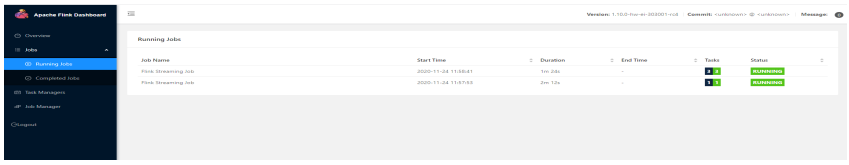


Figure 10-59 submitted task

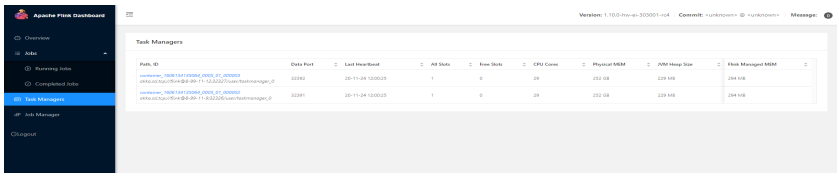
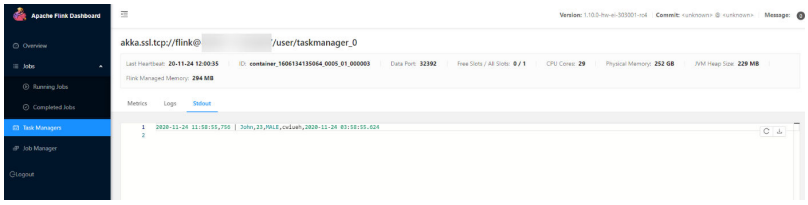


Figure 10-60 execution result



– **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 10-61**. Choose **Task Managers**, you can see submitted task, as shown in **Figure 10-62**. Choose **Stdout** to view the running result, as shown in **Figure 10-63**.

Figure 10-61 runnning jobs

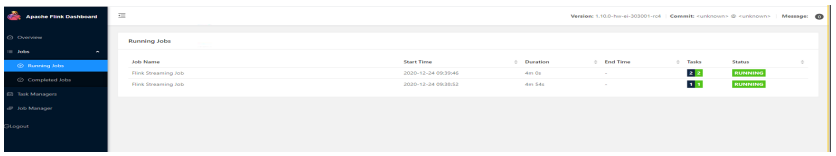


Figure 10-62 submitted task

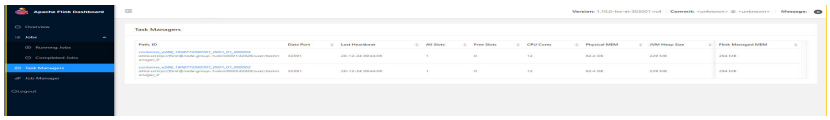
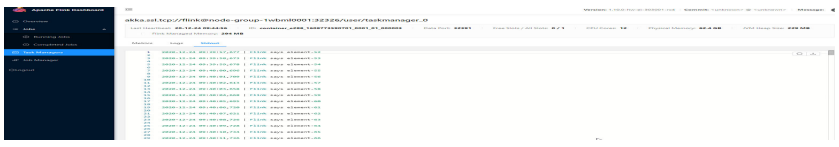


Figure 10-63 execution result



- **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. 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:

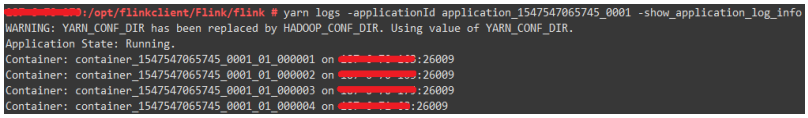
    source bigdata_env

    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*

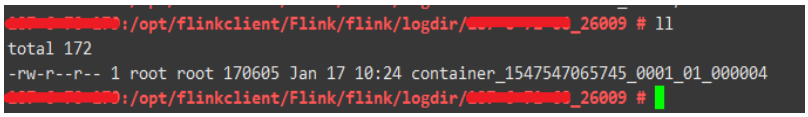
    vi. Run the following commands to obtain container information of the Flink cluster:

    yarn logs -applicationId application_* -show_application_log_info

    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/

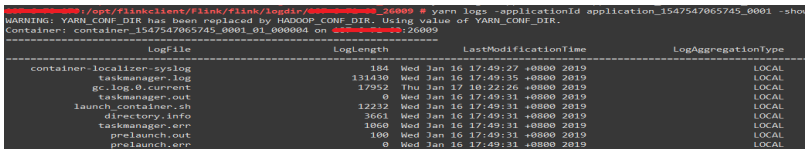
    

    After the command is executed, container run logs, including run logs of the Task Manager and Job Manager and GC logs, are downloaded to the local host.

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



Download **taskmanager.log** to the local host.

yarn logs -applicationId application_* --containerId container_1547547065745_0001_01_000004 -log_files taskmanager.log -out localpath

10.4.3 Running a Spring Boot Sample Project and Viewing Results

This section applies to MRS 3.3.0 or later.

Running the Spring Boot Sample in CLI

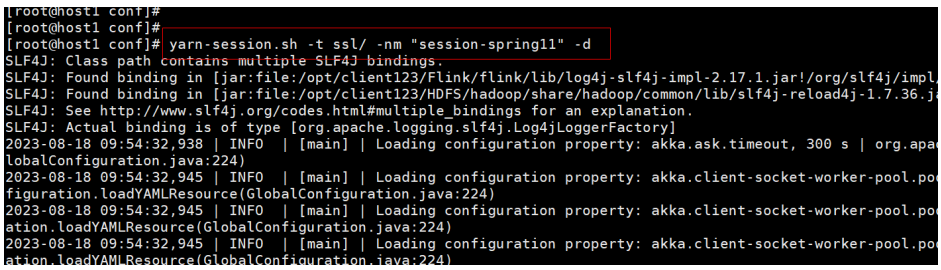
Step 1 Use Maven to run **install** on the IDEA.

If "BUILD SUCCESS" is displayed, the compilation is successful. A JAR file containing the **flink-dws-sink-example-1.0.0-SNAPSHOT** field is generated in the **target** directory of the sample project.

Step 2 On Linux, go to the client installation directory, for example, **/opt/client/Flink/flink/conf**, and save the JAR packages whose names contain **flink-dws-sink-example-1.0.0-SNAPSHOT** in the **target** directory generated in to this directory.

Step 3 Run the following command to create a Yarn session:

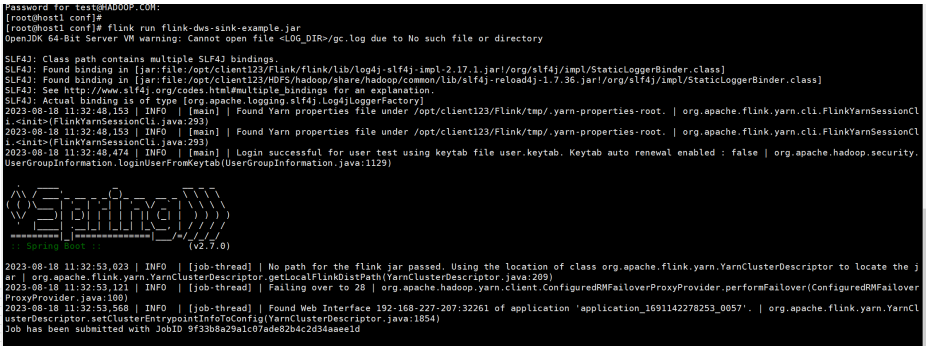
yarn-session.sh -t ssl/ -nm "*session-spring11***" -d**



Step 4 Run the following command to start the SpringBoot service:

- Run the GaussDB(DWS) sample

  flink run flink-dws-sink-example.jar



----End

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.

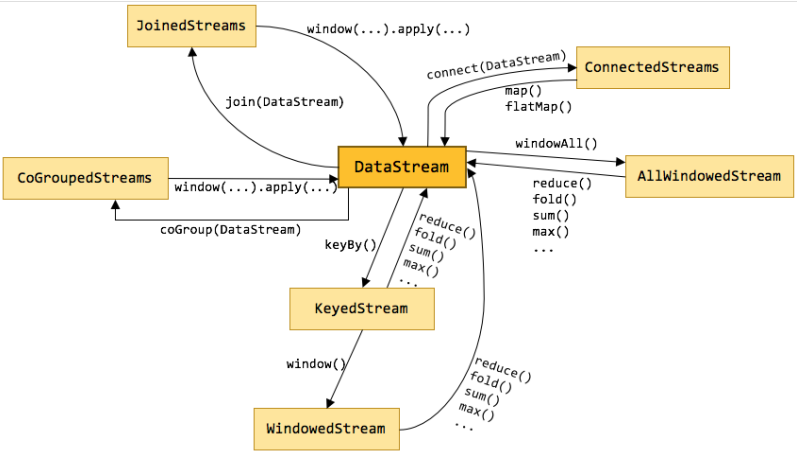
Common APIs of Flink

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 10-64 Conversion of Flink stream types



Data Stream Source

Table 10-14 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. • **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. • **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
public <OUT> DataStreamSource<OUT> fromParallelCollection(SplittableIterator<OUT> iterator, Class<OUT> type)	Obtain the user-defined data collection as parallel data stream source. • **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)	
public DataStreamSource<Long> generate Sequence(long from, long to)	Obtain a sequence of user-defined data as the data stream source. • **from** indicates the starting point of numbers. • **to** indicates the end point of numbers.
public DataStreamSource<String> readTextFile(String filePath)	Obtain the user-defined text file from a specific path as the data stream source. • **filePath** indicates the path of the text file. • **charsetName** indicates the encoding format.
public DataStreamSource<String> readTextFile(String filePath, String charsetName)	
public <OUT> DataStreamSource<OUT> readFile(FileInputformat<OUT> inputformat, String filePath)	Obtain the user-defined file from a specific path as the data stream source. • **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> DataStreamSource<OUT> readFile(FileInputformat<OUT> inputformat, String filePath, FileProcessingMode watchType, long interval)	

API	Description
public <OUT> DataStreamSource<OUT> readFile(FileInputformat<OUT> inputformat, String filePath, FileProcessingMode watchType, long interval, TypeInformation<OUT> typeInformation)	
public DataStreamSource<String> socketTextStream(String hostname, int port, String delimiter, long maxRetry)	Obtain user-defined socket data as the data stream source. • **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 DataStreamSource<String> socketTextStream(String hostname, int port, String delimiter)	
public DataStreamSource<String> socketTextStream(String hostname, int port)	
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. • **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 10-15 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. • **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)	Writhe data to a specific **.csv** file. • **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.
public DataStreamSink<T> writeAsCsv(String path, WriteMode writeMode)	
public <X extends Tuple> DataStreamSink<T> writeAsCsv(String path, WriteMode writeMode, String rowDelimiter, String fieldDelimiter)	
public DataStreamSink<T> writeToSocket(String hostName, int port, SerializationSchema<T> schema)	Write data to the socket connection. • **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 10-16 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 10-17 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. Return the KeyedStream after the KeyBy operation, and call the KeyedStream function, such as reduce, fold, min, minby, max, maxby, sum, and sumby. • **fields** indicates the numbers of columns or names of member variables. • **key** indicates the user-defined basis for partitioning.
public KeyedStream<T, Tuple> keyBy(String... fields)	
public <K> KeyedStream<T, K> keyBy(KeySelector<T, K> key)	
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. • **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)	

API	Description
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. • **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 10-18 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. • **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.

API	Description
public DataStream<T> broadcast()	Broadcast each element to all partitions.

Project Capabilities

Table 10-19 APIs about projecting

API	Description
public <R extends Tuple> SingleOutputStreamOperator<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 10-20 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 10-21 lists differences of AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks APIs.

Table 10-21 Difference of AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks APIs

Parameter	Description
AssignerWithPeriodic-Watermarks	Generate Watermark based on the getConfig().setAutoWatermarkInterval(200L) timestamp of StreamExecutionEnvironment class.
AssignerWithPunc-tuatedWatermarks	Generate a Watermark each time an element is received. Watermarks can be different based on received elements.

Iteration

Table 10-22 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. **NOTE** • This API is helpful to algorithms that require constant update of models. • long maxWaitTimeMillis: The timeout period of each round of iteration.
public IterativeStream<T> iterate(long maxWaitTimeMillis)	

Stream Splitting

Table 10-23 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 10-24 lists APIs for generating windows.

Table 10-24 APIs for generating windows

API	Description
public <W extends Window> WindowedStream<T, KEY, W> window(WindowAssigner<? super T, W> assigner)	Define windows in partitioned KeyedStreams. A window groups each key according to some characteristics (for example, data received within the latest 5 seconds.
public <W extends Window> AllWindowedStream<T, W> windowAll(WindowAssigner<? super T, W> assigner)	Define windows in DataStreams.
public WindowedStream<T, KEY, TimeWindow> timeWindow(Time size)	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. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window. **NOTE** • 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, TimeWindow> timeWindow(Time size, Time slide)	

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. • **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. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window. **NOTE** • 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. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window.
public AllWindowedStream<T, GlobalWindow> countWindowAll(Time size, Time slide)	

Table 10-25 lists APIs for operating windows.

Table 10-25 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. • **function** indicates the window function to be executed. • **resultType** indicates the type of returned data.

Method	API	Description
	public <R> SingleOutputStreamOperator<R> apply(WindowFunction<T, R, K, W> function, TypeInformation<R> resultType)	
	public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> function)	Apply a reduce function to the window and return the result. • **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)	
	public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, WindowFunction<T, R, K, W> function, TypeInformation<R> resultType)	
	public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function)	Apply a fold function to the window and return the result. • **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.
	public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function, TypeInformation<R> resultType)	
	public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, WindowFunction<ACC, R, K, W> function)	

Method	API	Description
	public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, WindowFunction<ACC, R, K, W> function, TypeInformation<ACC> foldAccumulatorType, TypeInformation<R> resultType)	
Window All	public <R> SingleOutputStreamOperator<R> apply(AllWindowFunction<T, R, W> function)	Apply a general function to a window. The data in the window is calculated as a whole.
	public <R> SingleOutputStreamOperator<R> apply(AllWindowFunction<T, R, W> function, TypeInformation<R> resultType)	
	public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> function)	Apply a reduce function to the window and return the result. • **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.
	public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, AllWindowFunction<T, R, W> function)	
	public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, AllWindowFunction<T, R, W> function, TypeInformation<R> resultType)	

Method	API	Description
	public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function)	Apply a fold function to the window and return the result. • **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.
	public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function, TypeInformation<R> resultType)	
	public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, AllWindowFunction<ACC, R, W> function)	
	public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, AllWindowFunction<ACC, R, W> function, TypeInformation<ACC> foldAccumulatorType, TypeInformation<R> resultType)	
Window and Window All	public SingleOutputStreamOperator<T> sum(int positionToSum)	Sum a specified column of the window data. **field** and **positionToSum** indicate a specific column of the data.
	public SingleOutputStreamOperator<T> sum(String field)	
	public SingleOutputStreamOperator<T> min(int positionToMin)	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.
	public SingleOutputStreamOperator<T> min(String field)	

Method	API	Description
	public SingleOutputStreamOperator<T> minBy(int positionToMinBy)	Obtain the row where the minimum value of a column locates in the window data. minBy returns all elements of that row. • **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> 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. • **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)	

Combining Multiple DataStreams

Table 10-26 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<IN1, 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.
public <R> SingleOutputStreamOperator<R> flatMap(CoFlatMapFunction<IN1, IN2, R> coFlatMapper)	Perform mapping operation, which is similar to flatMap operation in DataStream, on elements.

Join Operation

Table 10-27 APIs about join operation

API	Description
public <T2> JoinedStreams<T, T2> join(DataStream<T2> otherStream)	Join two DataStreams using a given key in a specified window.
public <T2> CoGroupedStreams<T, T2> coGroup(DataStream<T2> otherStream)	Co-group two DataStreams using a given key in a specified window.

10.5.1.2 Scala

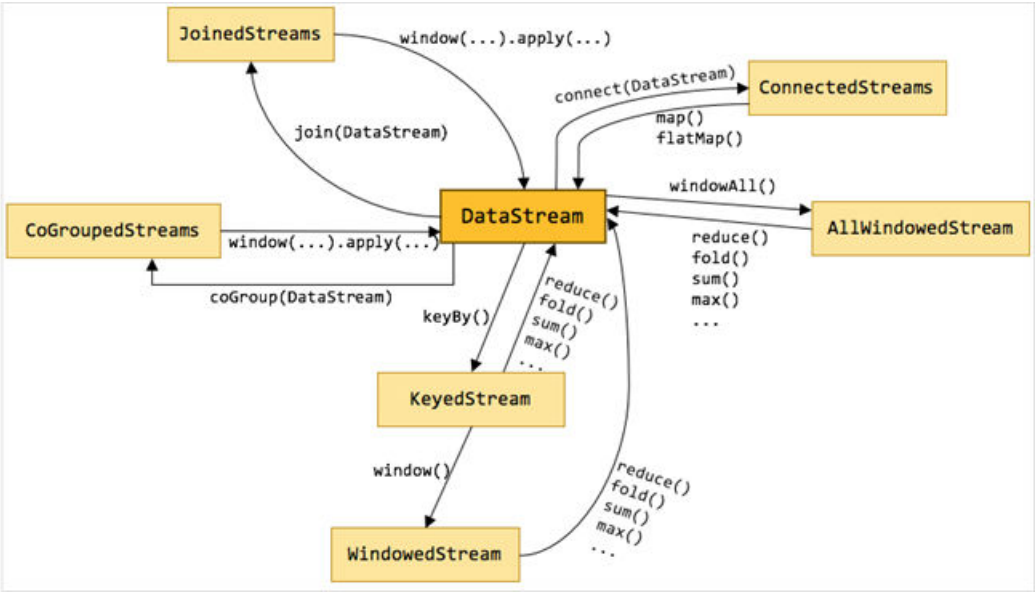
To avoid API compatibility or reliability issues after updates to the open-source Flink, recommended that APIs of the matching version are recommended.

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 10-65 Conversion of Flink stream types



Data Stream Source

Table 10-28 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.
def generateSequence(from: Long, to: Long): DataStream[Long]	Obtain a sequence of user-defined data as the data stream source. • **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. • **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. • **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]	

API	Description
def socketTextStream(hostname: String, port: Int, delimiter: Char = '\n', maxRetry: Long = 0): DataStream[String]	Obtain user-defined socket data as the data stream source. • **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.
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. • **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 10-29 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. • **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]	

API	Description
def writeAsCsv(path: String): DataStreamSink[T]	Writhe data to a specific **.csv** file. • **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. • **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 10-30 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](flatMapper: 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]	

API	Description
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 10-31 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. • **fields** indicates the IDs of certain columns • **firstField** and **otherFields** are names of member variables. • **key** indicates the user-defined basis for partitioning.
def keyBy(firstField: String, otherFields: String*): KeyedStream[T, JavaTuple]	
def keyBy[K: TypeInformation](fun: T => K): KeyedStream[T, K]	
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]	

API	Description
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 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 10-32 APIs about DataStream distribution

API	Description
def partitionCustom[K: TypeInformation](partitioner: Partitioner[K], field: Int) : DataStream[T]	Use a user-defined partitioner to select target task for each element. • **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: TypeInformation](partitioner: Partitioner[K], field: String):DataStream[T]	
def partitionCustom[K: TypeInformation](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 10-33 APIs about configuring the eventtime attribute

API	Description
def assignTimestamp-sAndWatermarks(assigner: AssignerWithPerio-dicWatermarks[T]): DataStream[T]	Extract timestamp from records, enabling event time window to trigger computing.

API	Description
def assignTimestamp-sAndWatermarks(assigner: AssignerWithPunc-tuatedWatermarks[T]): DataStream[T]	

Iteration

Table 10-34 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** • 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 10-35 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 10-36 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.
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. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window. **NOTE** • 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 countWindow(size: Long, slide: Long): WindowedStream[T, K, GlobalWindow]	

API	Description
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. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window.
def timeWindowAll(size: Time, slide: time AllWindowedStream[T, TimeWindow]	
def countWindow(size: Long, slide: Long): WindowedStream[T, K, GlobalWindow]	Divide windows according to the number of elements and define windows in partitioned KeyedStreams. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window. **NOTE** • 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 countWindow(size: Long): WindowedStream[T, K, GlobalWindow]	
def countWindowAll(size: Long, slide: Long): AllWindowedStream[T, GlobalWindow]	Divide windows according to the number of elements and define windows in DataStreams. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window.
def countWindowAll(size: Long): AllWindowedStream[T, GlobalWindow]	

Table 10-37 lists APIs for operating windows.

Table 10-37 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. **function** indicates the window function to be executed.
	def apply[R: TypeInformation] (function: (K, W, Iterable[T], Collector[R]) => Unit): DataStream[R]	

Method	API	Description
	def reduce(function: ReduceFunction[T]): DataStream[T]	Apply a reduce function to the window and return the result. • **reduceFunction** indicates the reduce function to be executed. • **function** of **WindowFunction** indicates triggering an operation to the window after a reduce operation.
	def reduce(function: (T, T) => T): DataStream[T]	
	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. • **initialValue** indicates the initial value. • **foldFunction** indicates the fold function. • **function** of **WindowFunction** indicates triggering an operation to the window after a fold operation.
	def fold[R: TypeInformation] (initialValue: R)(function: (R, T) => R): DataStream[R]	
	def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, foldFunction: FoldFunction[T, ACC], function: WindowFunction[ACC, R, K, W]): DataStream[R]	
	def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, foldFunction: (ACC, T) => ACC, windowFunction: (K, W, Iterable[ACC], Collector[R]) => Unit): DataStream[R]	
Window All	def apply[R: TypeInformation] (function: AllWindowFunction[T, R, W]): DataStream[R]	Apply a general function to a window. The data in the window is calculated as a whole.

Method	API	Description
	def apply[R: TypeInformation](function: (W, Iterable[T], Collector[R]) => Unit): DataStream[R]	
	def reduce(function: ReduceFunction[T]): DataStream[T]	Apply a reduce function to the window and return the result. • **reduceFunction** indicates the reduce function to be executed. • **AllWindowFunction** indicates triggering an operation to the window after a reduce operation.
	def reduce(function: (T, T) => T): DataStream[T]	
	def reduce[R: TypeInformation](preAggregator: ReduceFunction[T], windowFunction: AllWindowFunction[T, R, W]): DataStream[R]	
	def reduce[R: TypeInformation](preAggregator: (T, T) => T, windowFunction: (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. • **initialValue** indicates the initial value. • **foldFunction** indicates the fold function. • **function** of **WindowFunction** indicates triggering an operation to the window after a fold operation.
	def fold[R: TypeInformation](initialValue: R)(function: (R, T) => R): DataStream[R]	
	def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, preAggregator: FoldFunction[T, ACC], windowFunction: AllWindowFunction[ACC, R, W]): DataStream[R]	
	def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, preAggregator: (ACC, T) => ACC, windowFunction: (W, Iterable[ACC], Collector[R]) => Unit): DataStream[R]	

Method	API	Description
Window and Window All	def sum(position: Int): DataStream[T]	Sum a specified column of the window data. **field** and **position** indicate a specific column of the data.
	def sum(field: String): DataStream[T]	
	def min(position: Int): DataStream[T]	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.
	def min(field: String): DataStream[T]	
	def max(position: Int): DataStream[T]	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.
	def max(field: String): DataStream[T]	
	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 10-38 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]	
def flatMap[R: TypeInformation] (coFlatMapper: CoFlatMapFunction[IN1, IN2, R]): DataStream[R]	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 flatMap[R: TypeInformation](fun1: (IN1, Collector[R]) => Unit, fun2: (IN2, Collector[R]) => Unit): DataStream[R]	
def flatMap[R: TypeInformation] (fun1·IN1 => TraversableOnce[R], fun2: IN2 => TraversableOnce[R]): DataStream[R]	

Join Operation

Table 10-39 APIs about join operation

API	Description
def join[T2] (otherStream: DataStream[T2]): JoinedStreams[T, T2]	Join two DataStreams using a given key in a specified window. The key value of the join operation is specified by the **where** and **eaualTo** method, indicating filtering data with equivalent conditions from two DataStreams.
def coGroup[T2] (otherStream: DataStream[T2]): CoGroupedStreams[T, T2]	Co-group two DataStreams using a given key in a specified window. The key value of the coGroup operation is specified by the where and eaualTo method, indicating partitioning two DataStreams using equivalent conditions.

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 10-40**. 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 10-40 lists all RESTful API paths supported by Flink.

Table 10-40 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 *running*, *finished*, *failed*, *canceled*.
/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.
/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.

Path	Description
/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.
/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.

Path	Description
/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 10-41 describes variables listed in **Table 10-40**.

Table 10-41 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 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 10-66**. **Figure 10-67** shows the normal situation when the title is displayed.

Figure 10-66 Title not displayed on the page

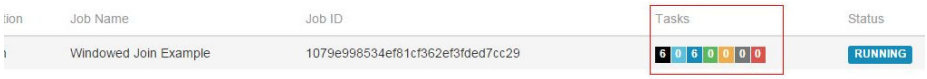
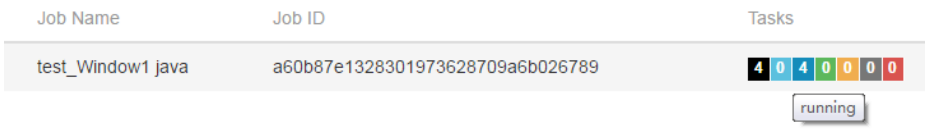


Figure 10-67 Title displayed normally



Answer

If the Chrome browser does not display the title, perform the following procedure:

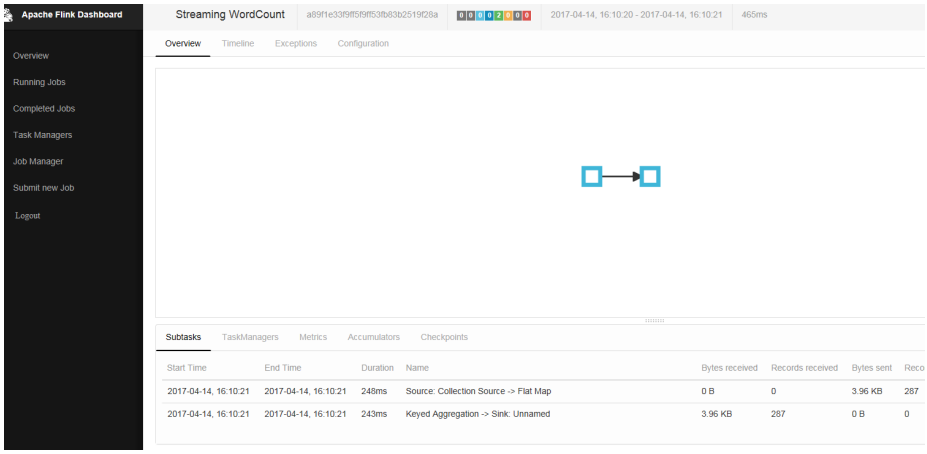
Check whether a tool that affects the tooltip display by the Chrome browser is running. If so, shut down the tool.

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 10-68**?

Figure 10-68 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 10-69**.
- 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 10-70**.

Figure 10-69 Time monitoring

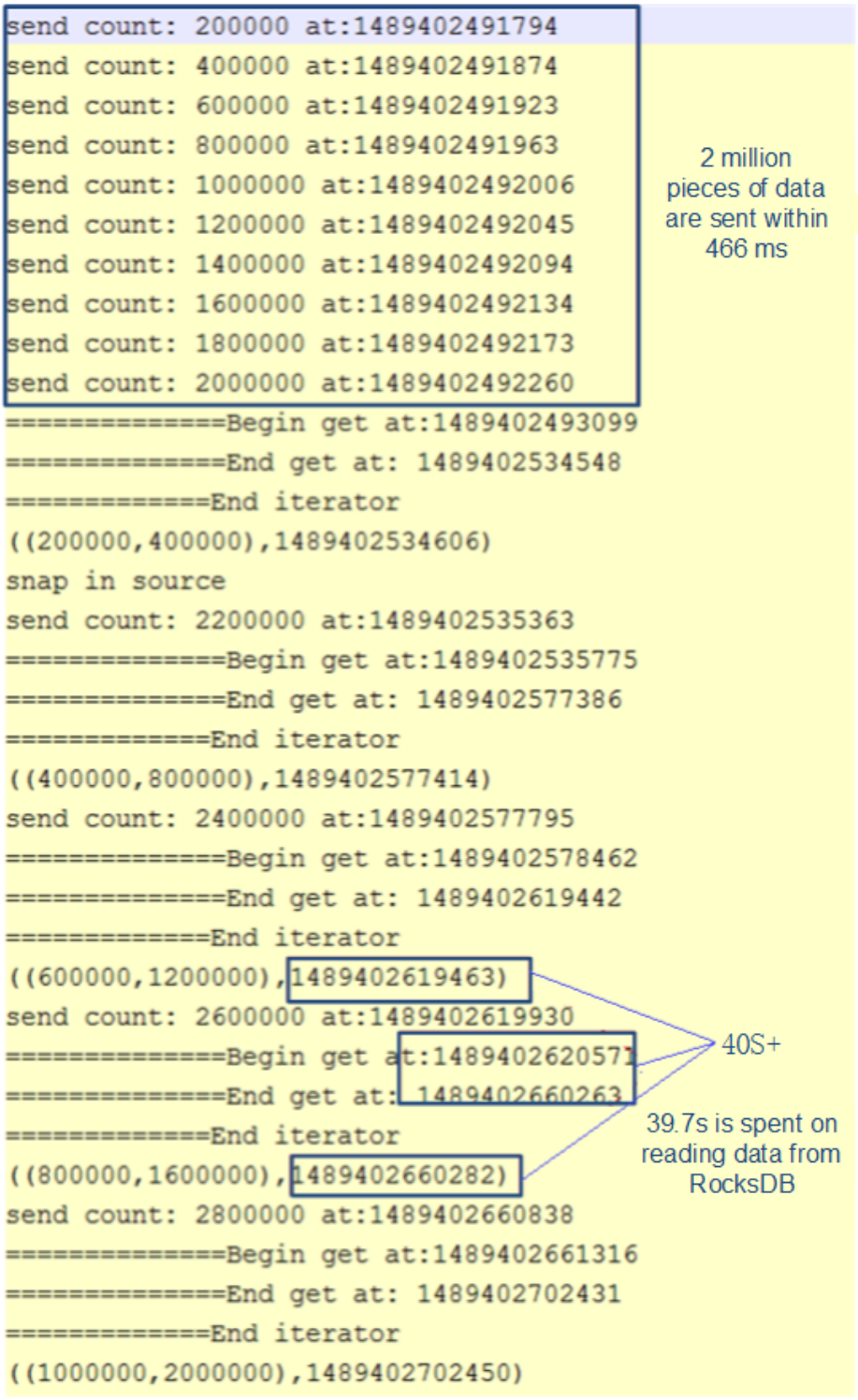
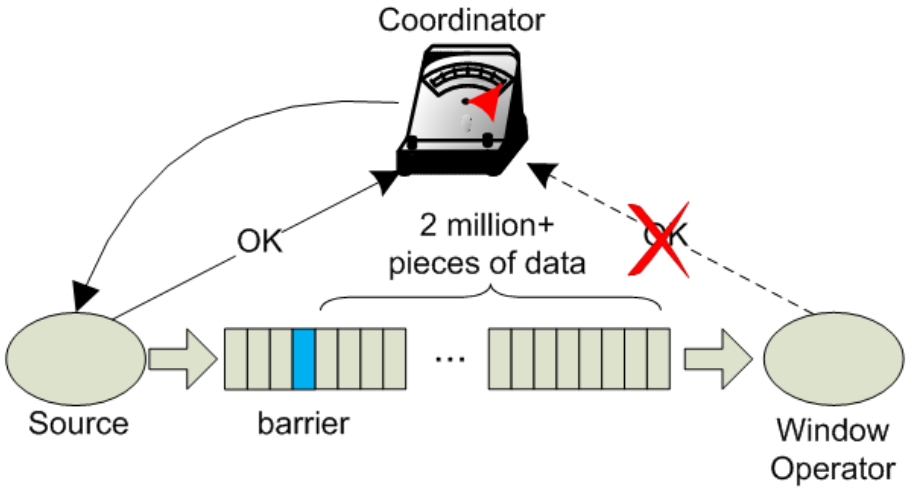


Figure 10-70 Relationship



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?

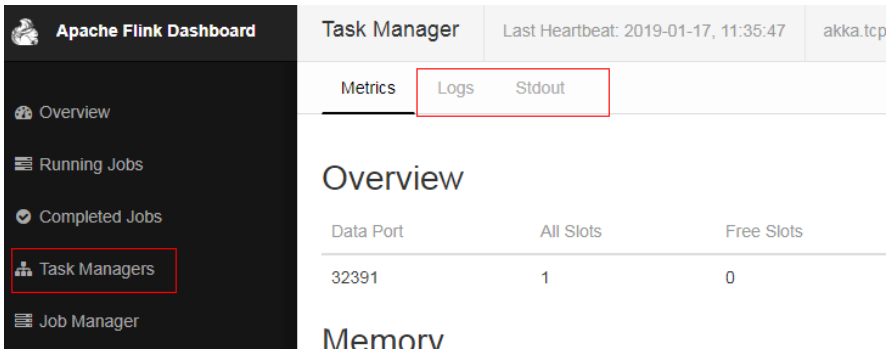
Question

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** > **Stdouts** on the left to view log information.



Configure the function of printing service logs and Task Manager run logs separately:

NOTICE

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(RocksDBStateBacken
d.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

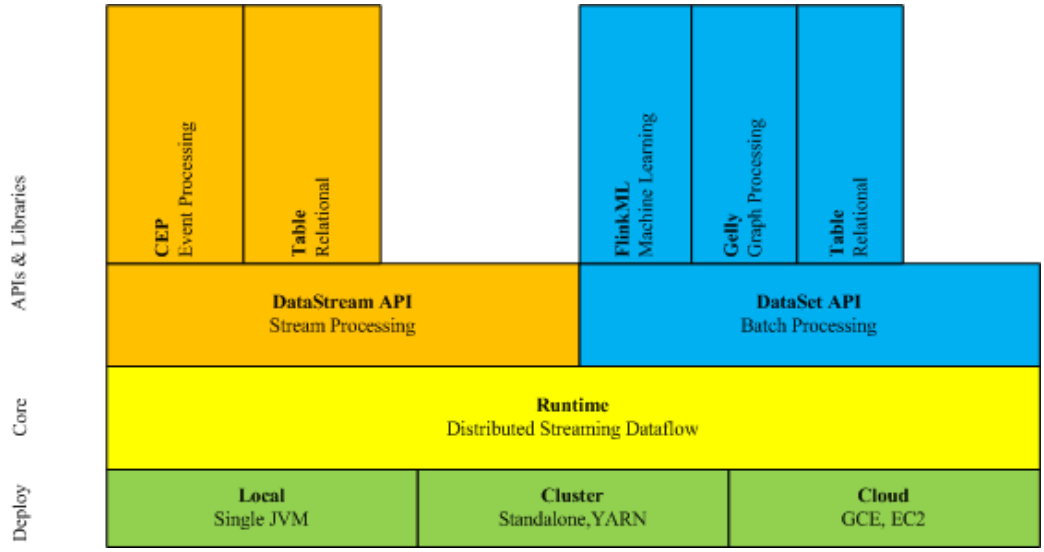
Overview

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



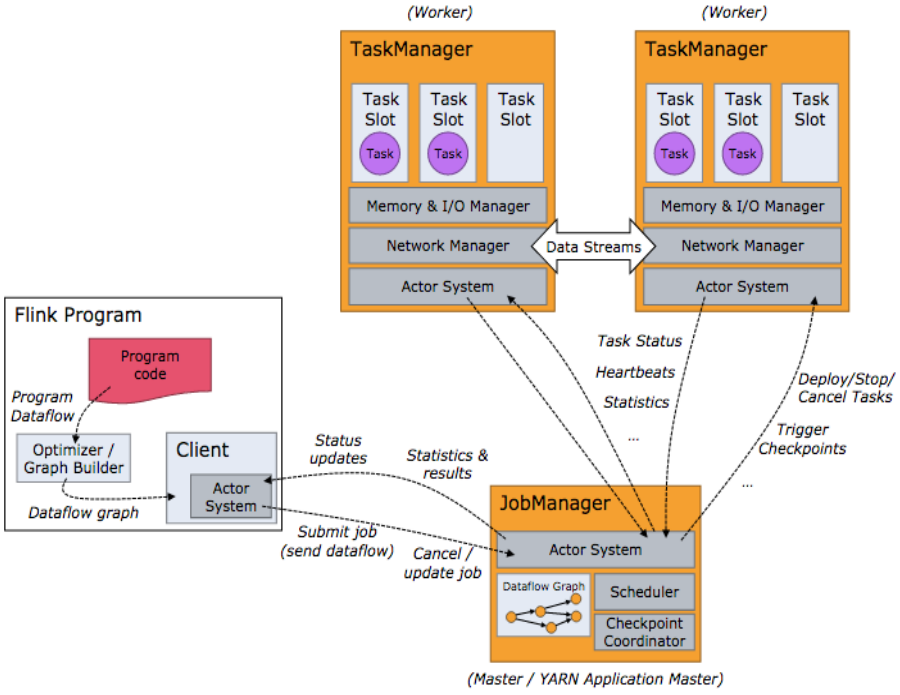
The following lists the key features of Flink in the current version:

- 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

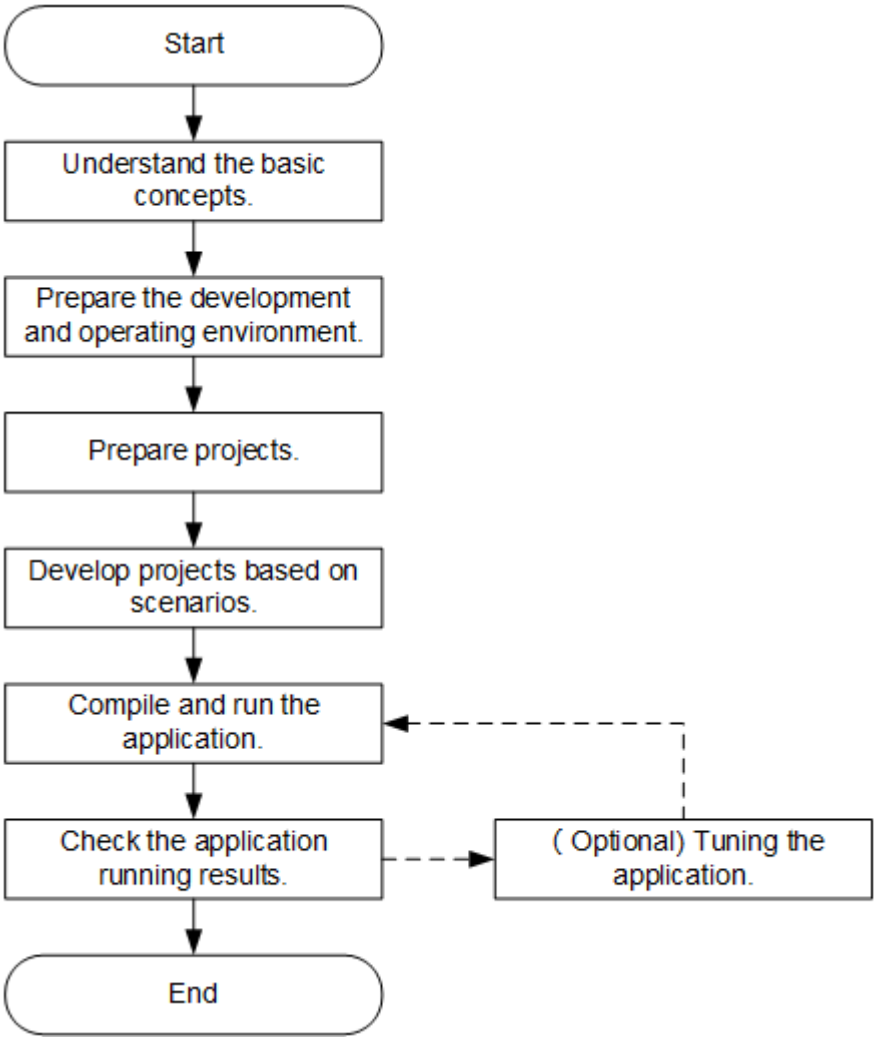


Table 11-2 Description of the development process

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	• 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	Basic configuration of the development and running environment. The version requirements are as follows: The server and client support only the built-in OpenJDK. Other JDKs cannot be used. If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows: • For x86 nodes that run clients, use the following JDKs: – Oracle JDK 1.8 – IBM JDK 1.8.0.7.20 and 1.8.0.6.15 • For Arm nodes that run clients, use the following JDKs: – OpenJDK 1.8.0_272 (built-in JDK, which can be obtained from the **JDK** folder in the cluster client installation directory.) – BiSheng JDK 1.8.0_272 **NOTE** • For security purposes, the server supports only TLS V1.2 or later. • 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, V1.1, and V1.2. For details, see **https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls**. • For details about the BiSheng JDK, see **https://www.hikunpeng.com/en/developer/devkit/compiler/jdk**.
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.

Item	Description
7-zip	It is a tool used to decompress **.zip** and **.rar** packages. The 7-Zip 16.04 is supported.
Python3	Used to run Flink Python jobs. Python 3.6 or later is required.

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.

Document Name	Function
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.

d. (Optional) To run a Python job, perform the following additional configurations:

   i. Log in to the node where the Flink client is installed as the **root** user and run the following command to check whether Python 3.6 or a later has been installed:

      python3 -V

   ii. Go to the python 3 installation path, for example, **/srv/pyflink-example**, and install the **virtualenv**:

      cd /srv/pyflink-example

      virtualenv venv --python=python3.x

      source venv/bin/activate

   iii. Copy the **Flink/flink/opt/python/apache-flink-*.tar.gz** file from the client installation directory to **/srv/pyflink-example**:

      cp /*Client installation directory*/Flink/flink/opt/python/apache-flink-*.tar.gz /srv/pyflink-example

   iv. Install the dependency package. If the following command output is displayed, the installation is successful:

      python -m pip install apache-flink-libraries-*.tar.gz

      python -m pip install apache-flink-*Version number***.tar.gz**

```
...
Successfully built apache-flink
 Installing collected packages: apache-flink
  Attempting uninstall: apache-flink
   Found existing installation: apache-flink x.xx.x
   Uninstalling apache- flink-x.xx.x:
    Successfully uninstalled apache-flink-x.xx.x
Successfully installed apache-flink-x.xx.x
```

11.2.2 Configuring and Importing a Sample Project

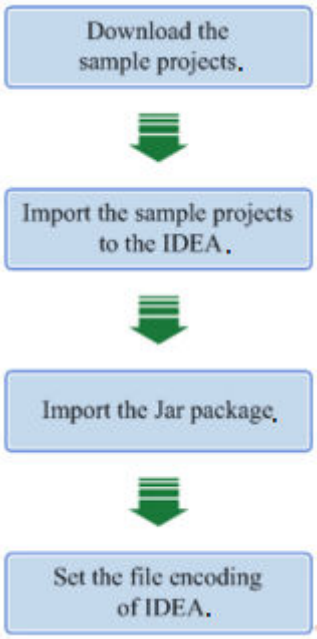
Scenarios

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 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**.

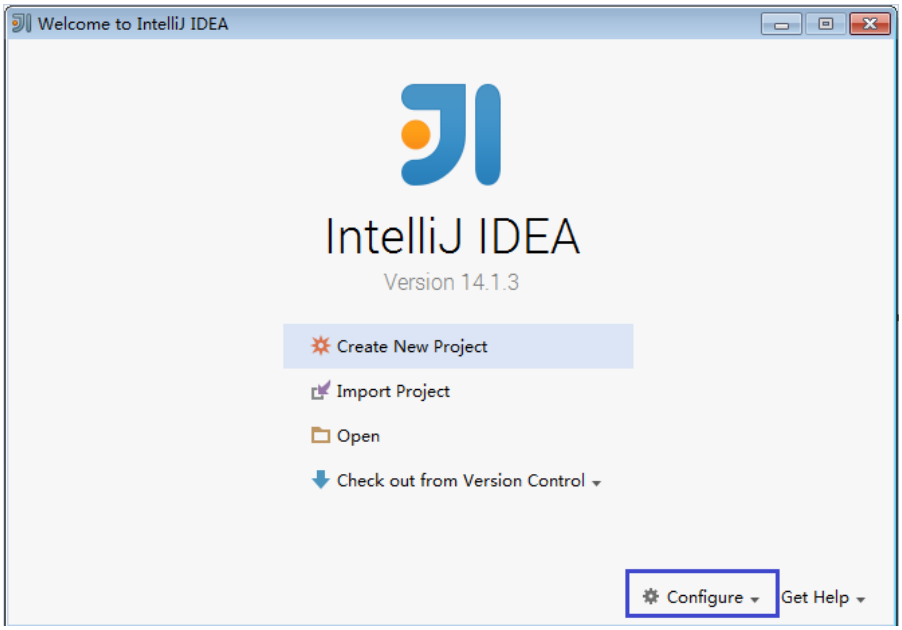
NOTE

- 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.

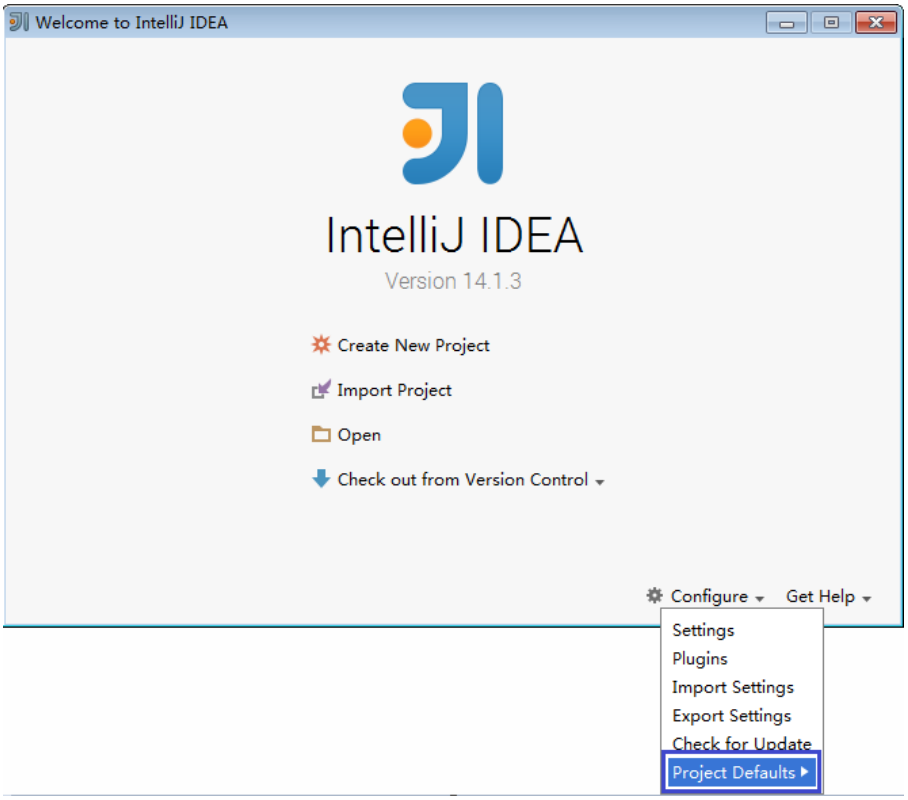
1. Start IntelliJ IDEA and click **Configure**.

Figure 11-5 Choosing Configure



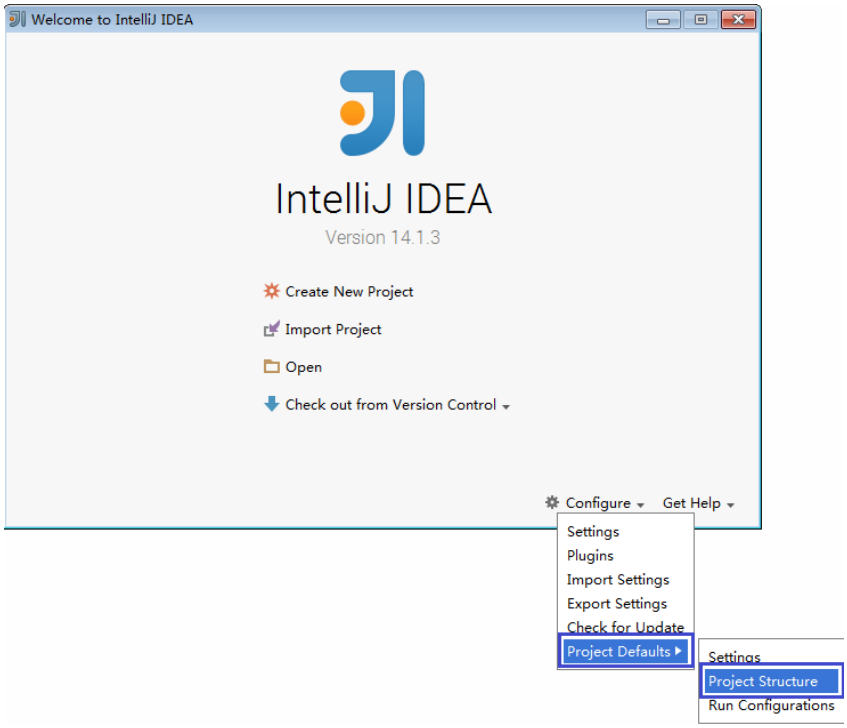
2. Choose **Project Defaults** from the **Configure** drop-down list.

Figure 11-6 Choosing Project Defaults



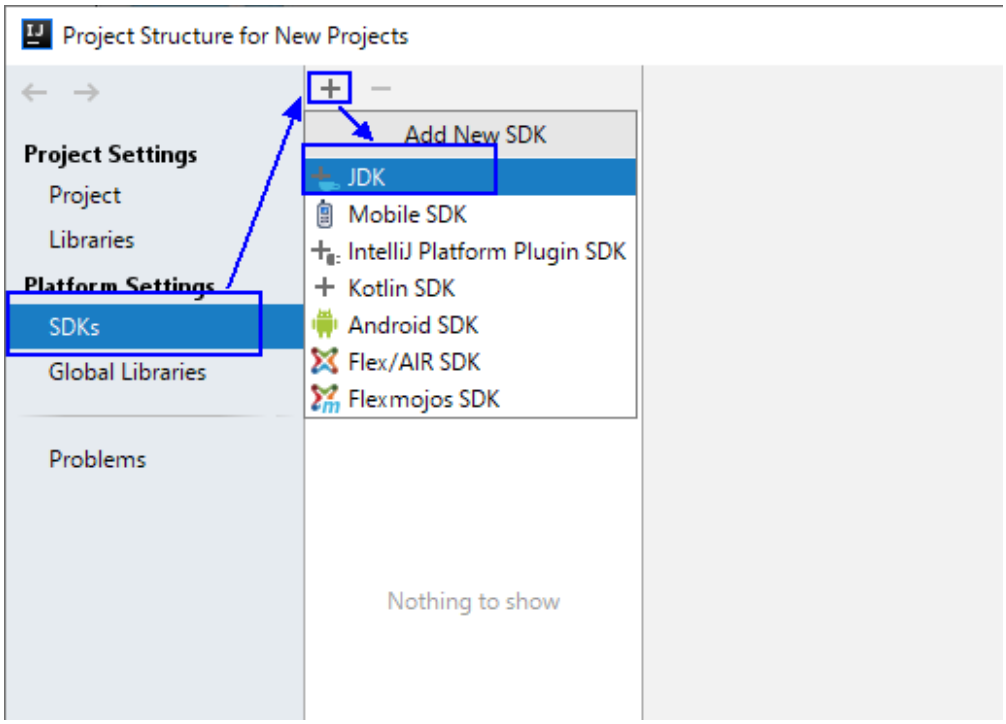
3. Choose **Project Structure** from the **Project Defaults** submenu.

Figure 11-7 Project Defaults



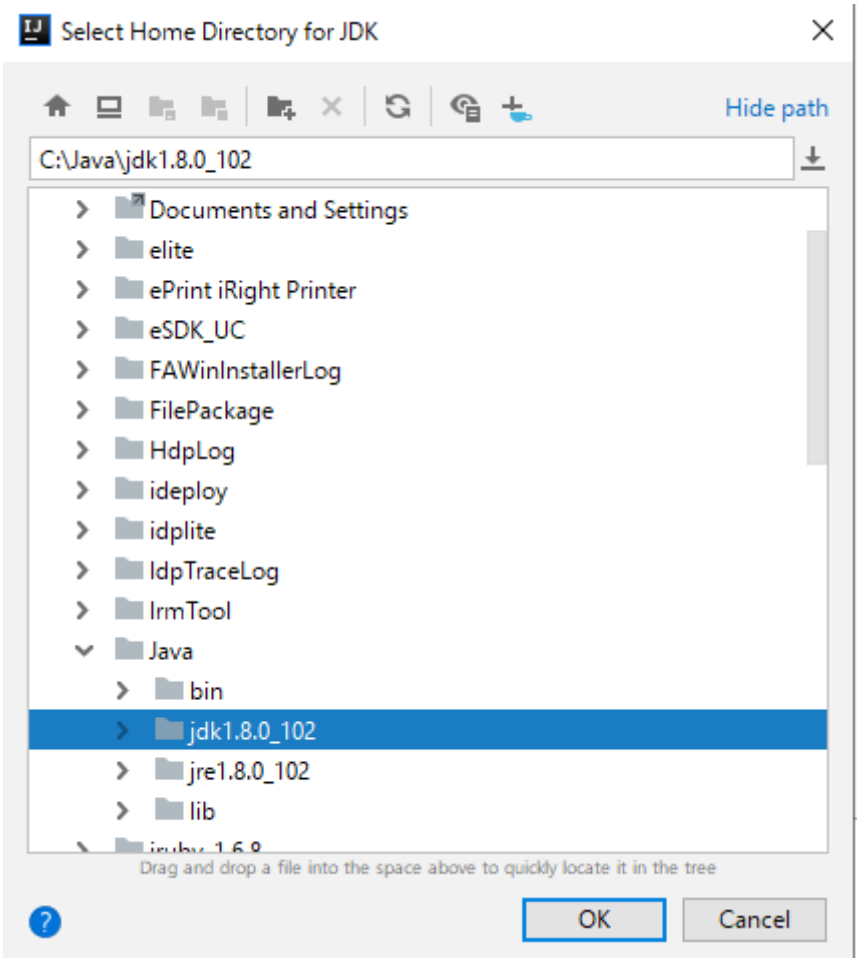
4. On the **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 11-8 Adding the JDK



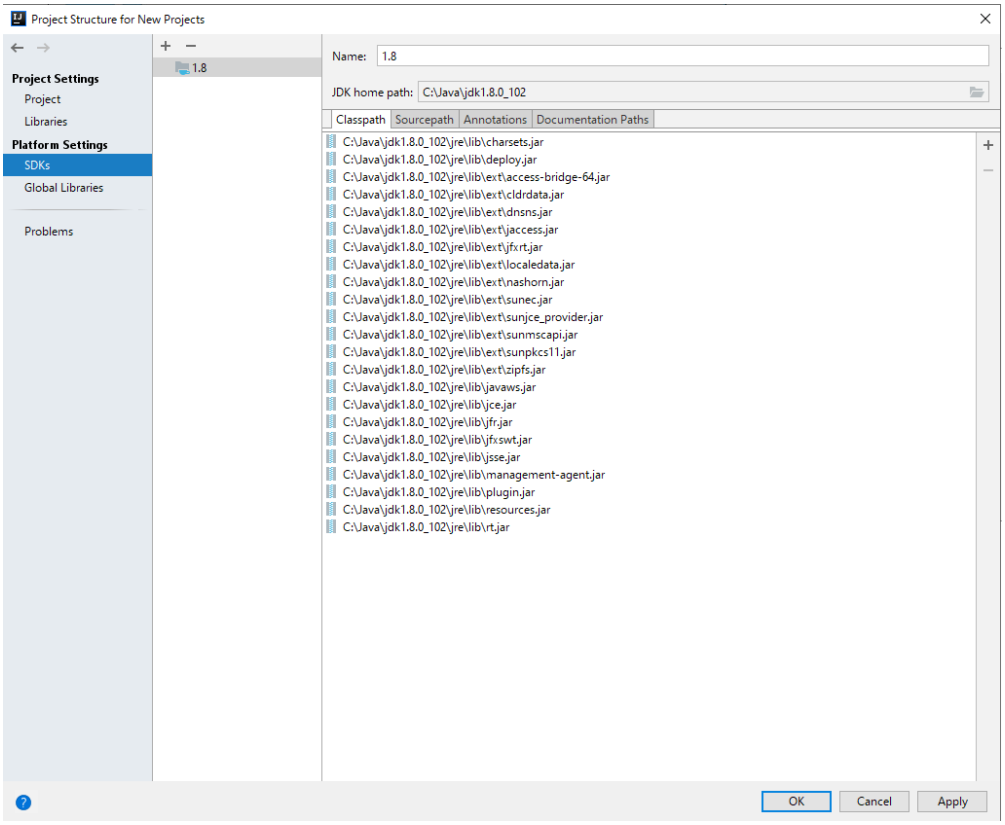
5. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 11-9 Selecting the JDK directory



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

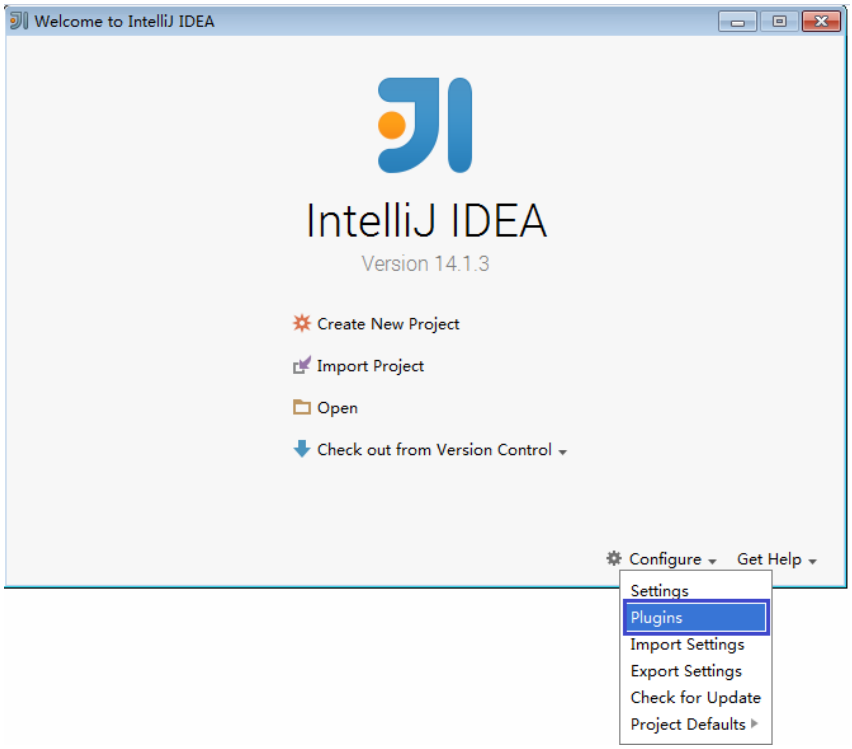
Figure 11-10 Completing the JDK configuration



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

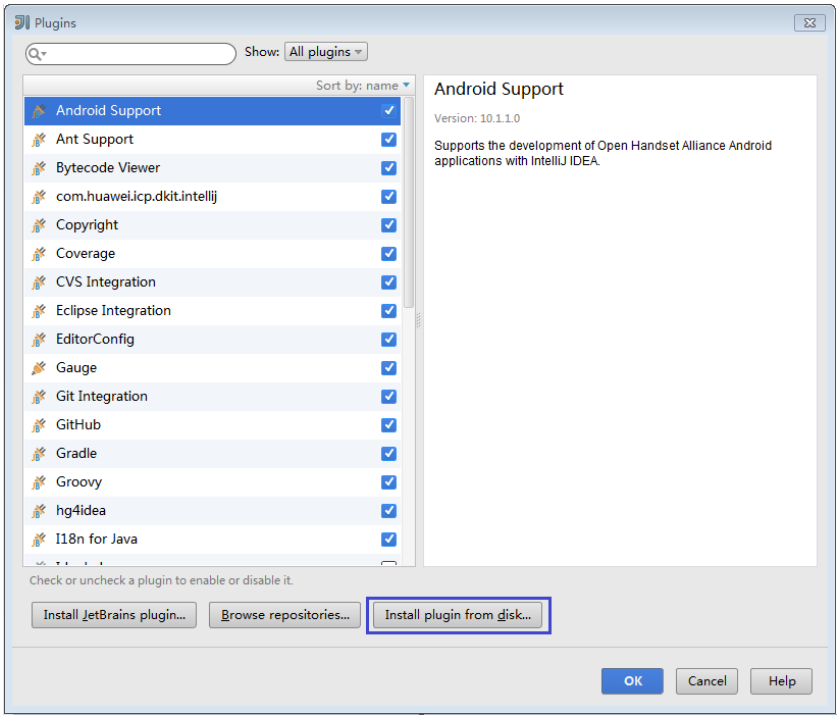
1. Choose **Plugins** from the **Configure** drop-down list.

Figure 11-11 Plugins



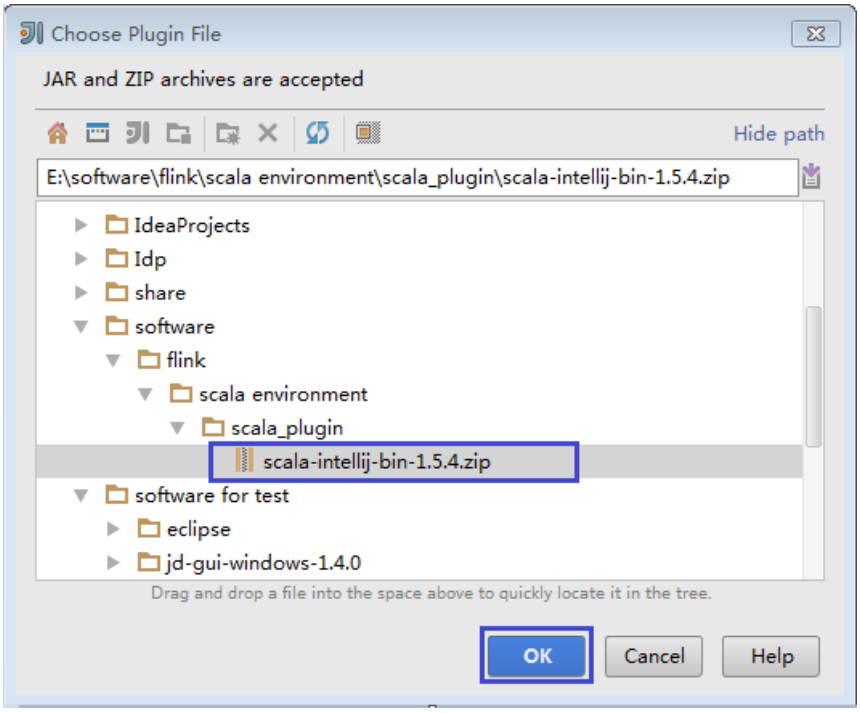
2. On the **Plugins** page, click **Install plugin from disk**.

Figure 11-12 Installing plugins from disk



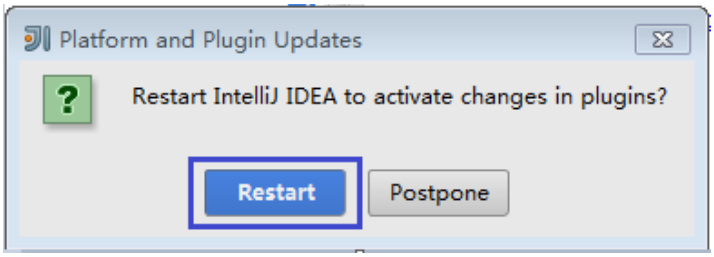
3. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.

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

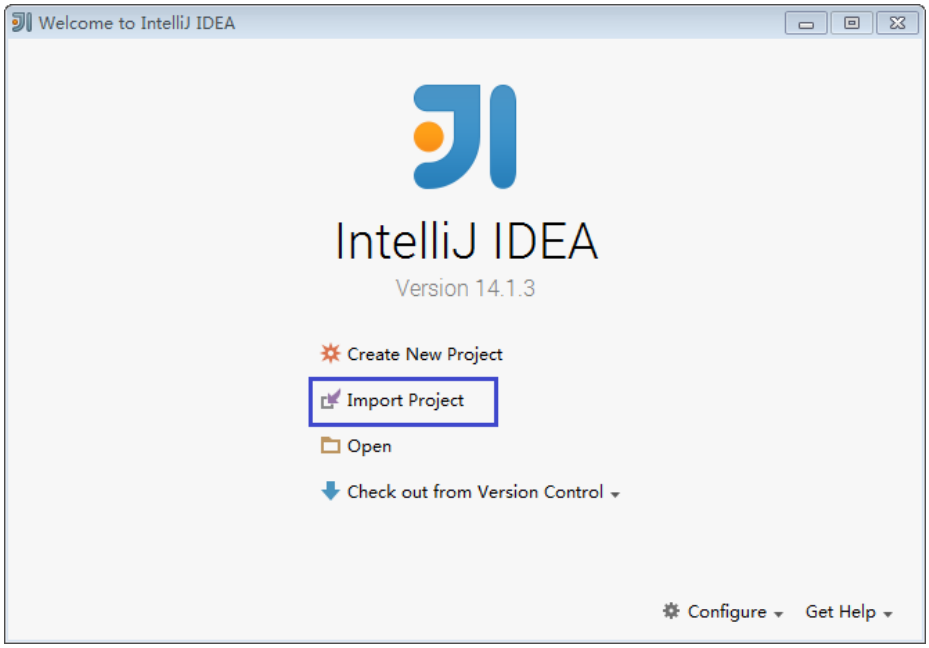


Step 4 Import the Java sample project to IDEA.

1. Start IntelliJ IDEA. On the **Quick Start** page, select **Import Project**.

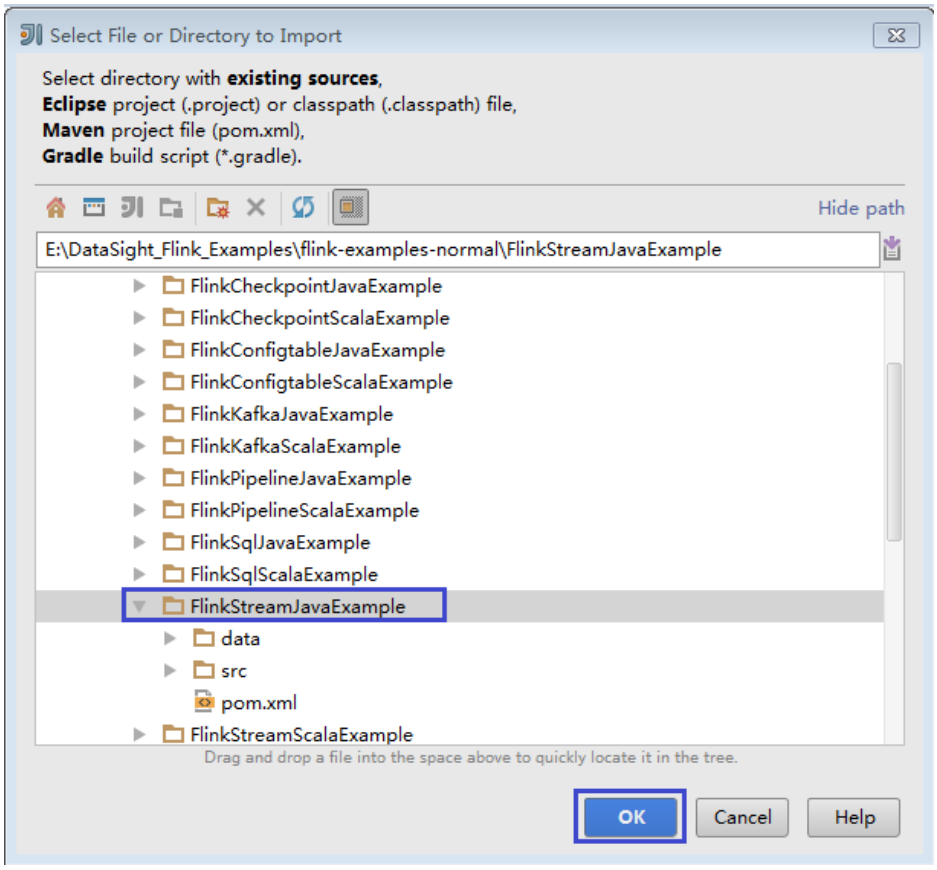
   Alternatively, for the used IDEA tool, add the project directly from the IDEA home page. Choose **File** > **Import project...** to import a project.

Figure 11-15 Importing a project (on the Quick Start page)



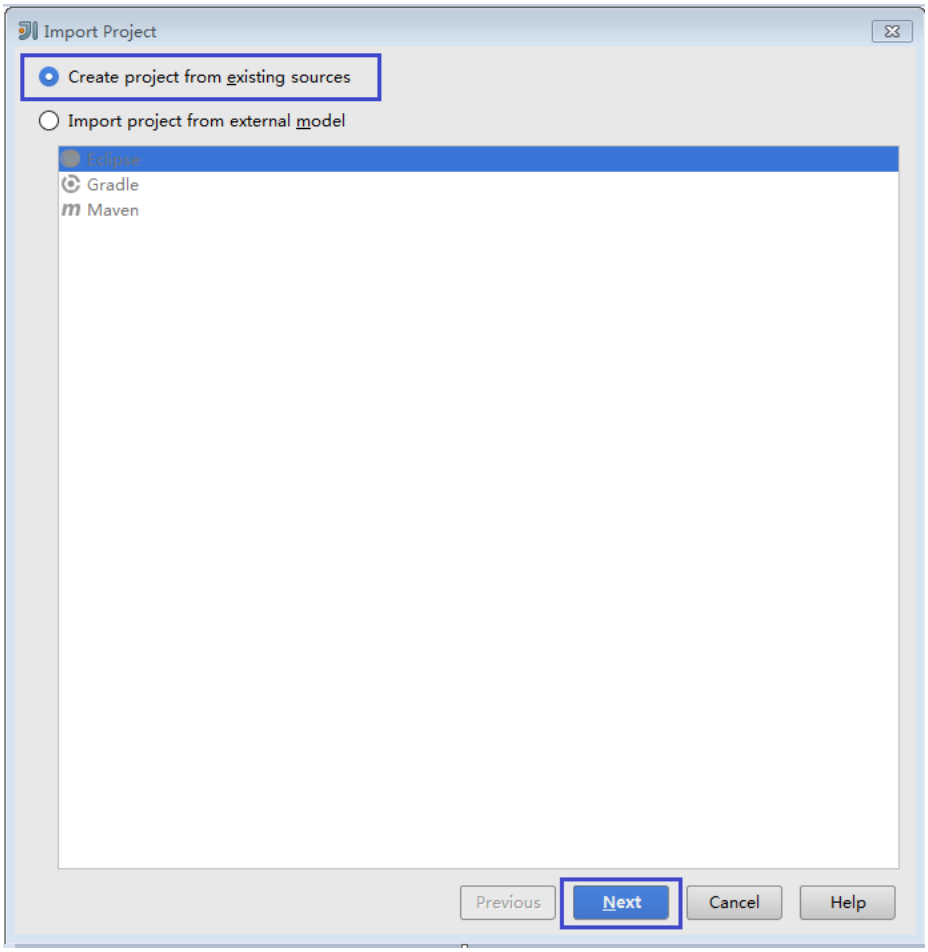
2. Select the directory for storing the imported projects and click **OK**.

Figure 11-16 Select File or Directory to Import



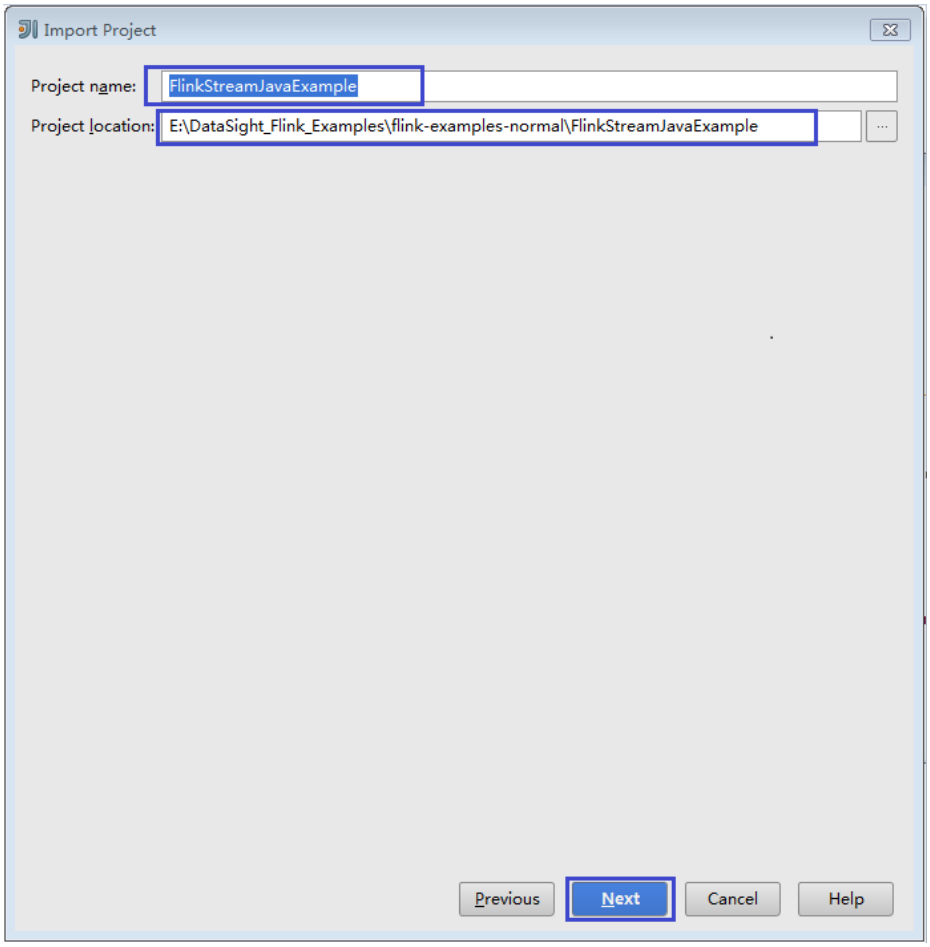
3. Select **Create project from existing sources** and click **Next**.

Figure 11-17 Create project from existing sources



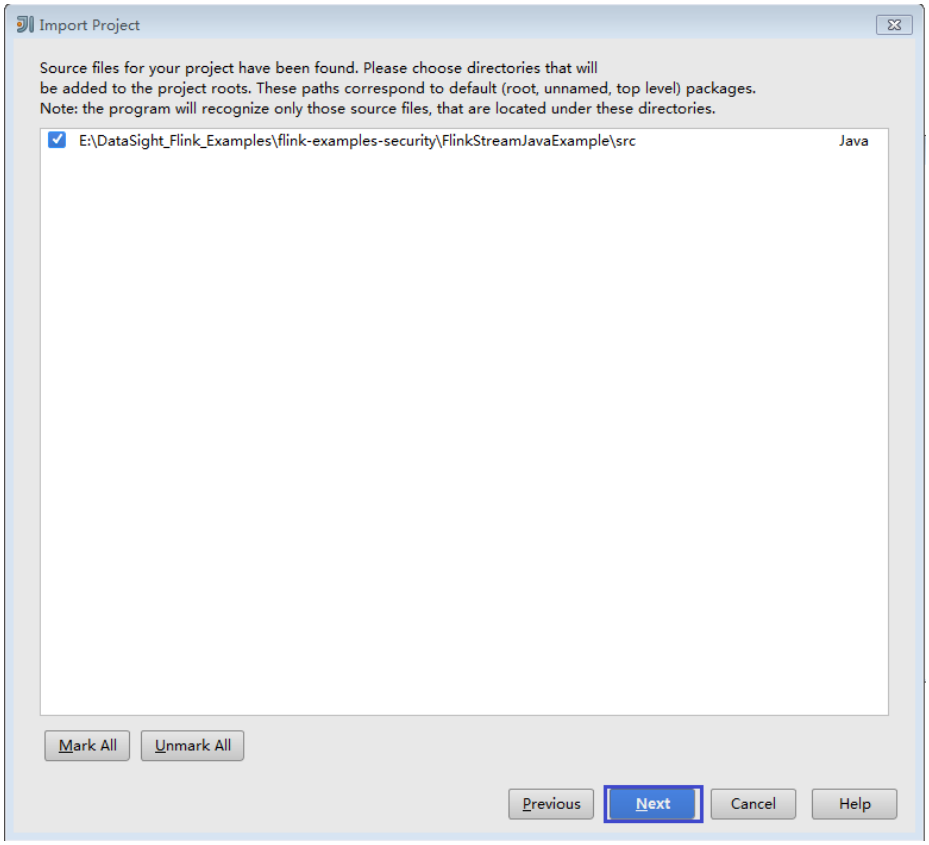
4. Confirm the project location and project name, and click **Next**.

Figure 11-18 Import Project



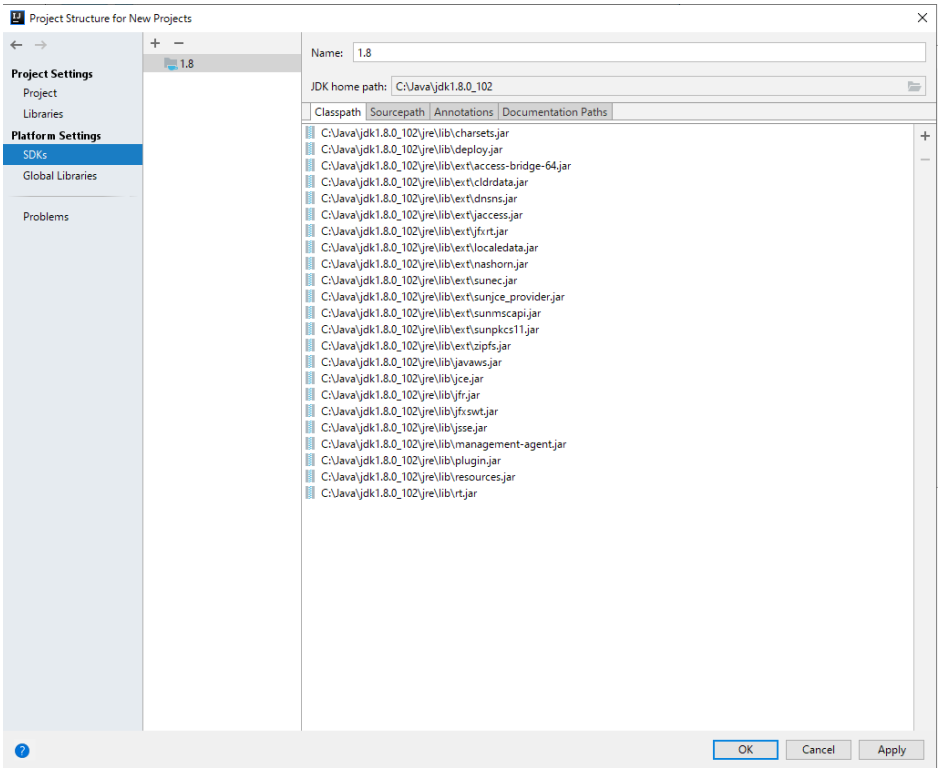
5. Retain the default value of the **root** directory for the project to be imported and click **Next**.

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

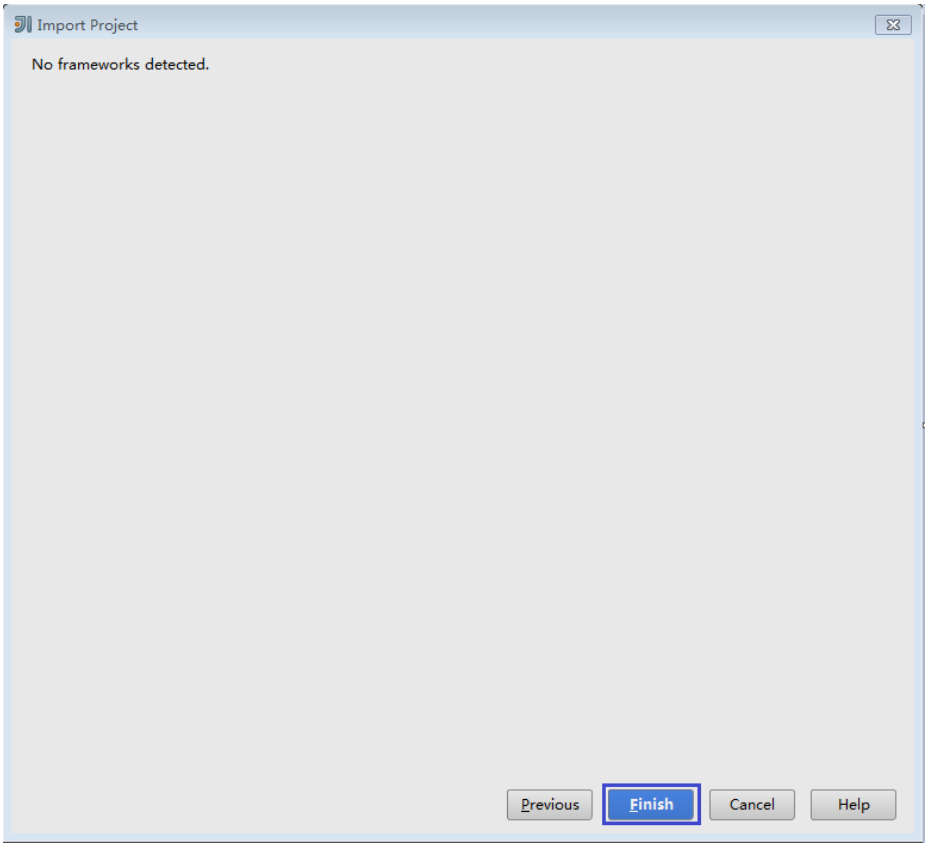
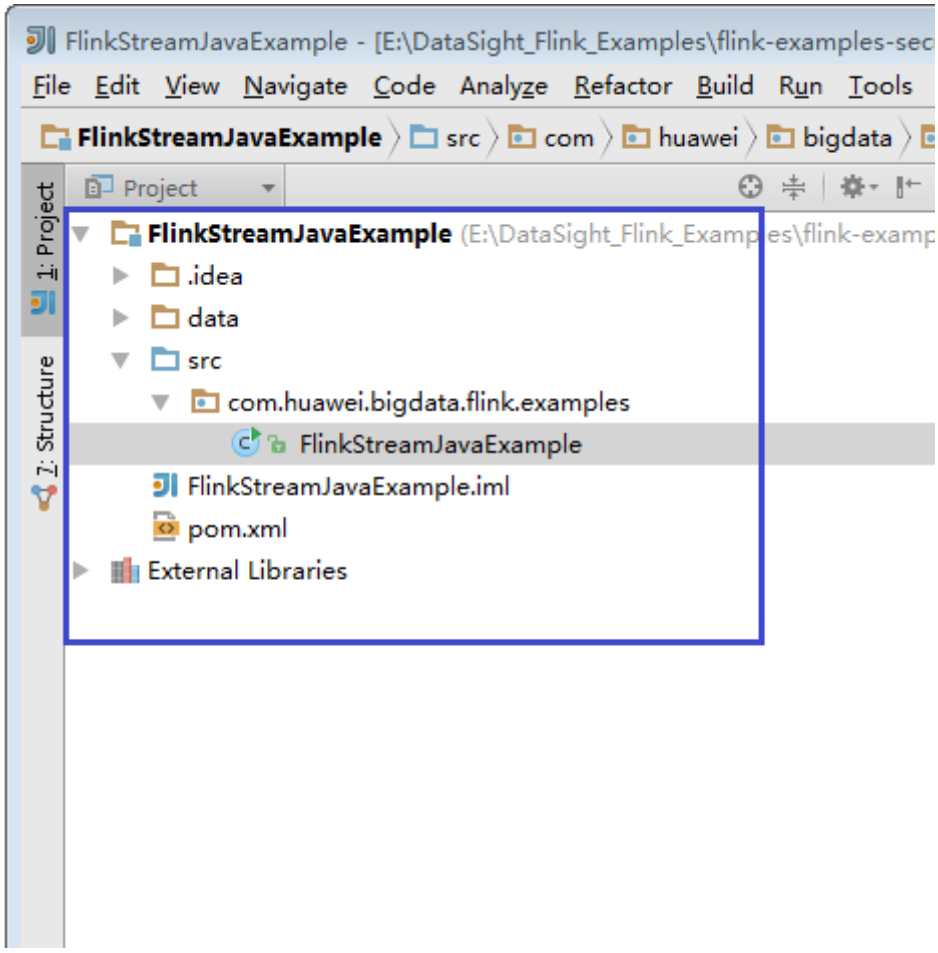


Figure 11-22 Imported project



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.

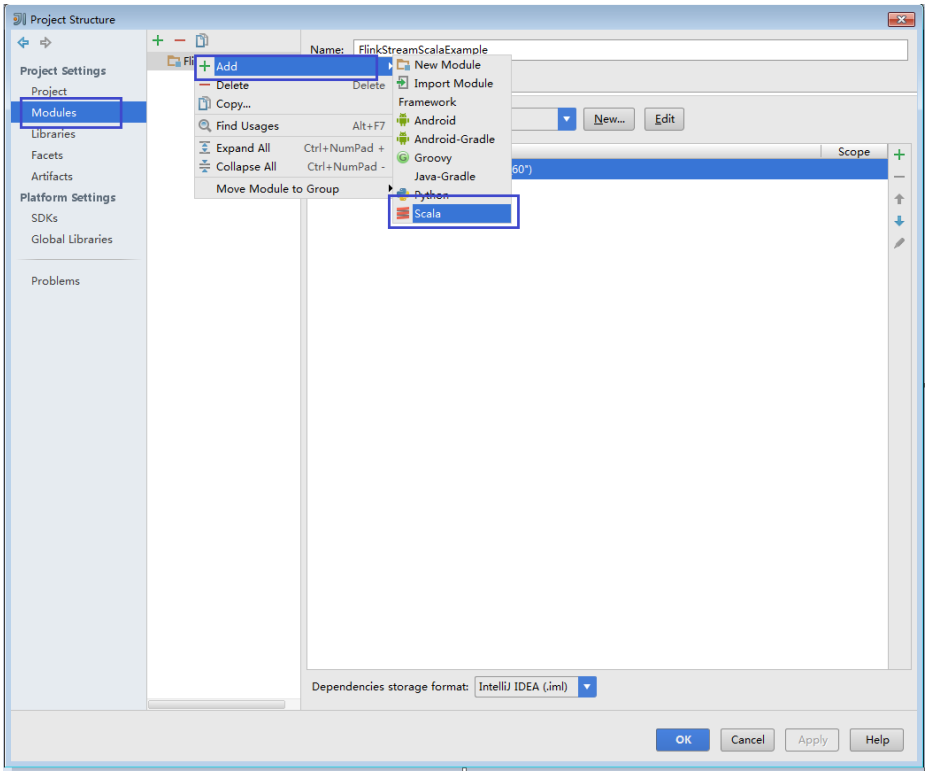
NOTE

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 about the dependency package for running the sample project**.

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

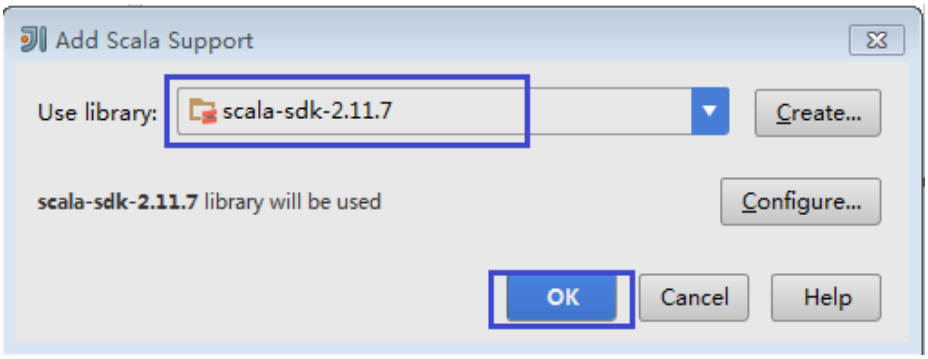
1. On the IDEA home page, choose **File** > **Project Structures...** to go to the **Project Structure** page.
2. Choose **Modules**, right-click a project name, and choose **Add** > **Scala**.

Figure 11-23 Selecting Scala



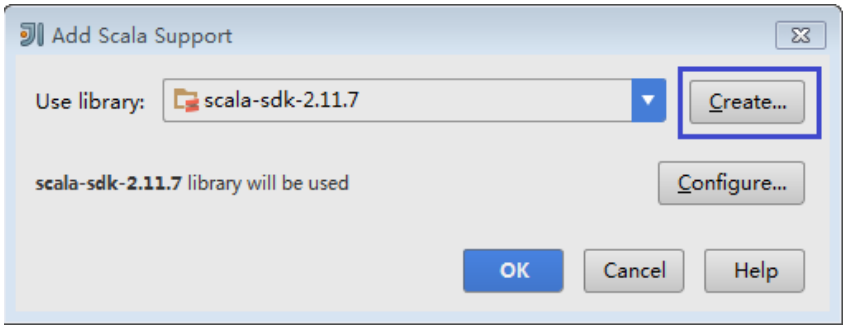
3. Wait until IDEA identifies Scala SDK, select the dependency JAR files in the **Add Scala Support** dialog box, and then click **OK**.

Figure 11-24 Add Scala Support



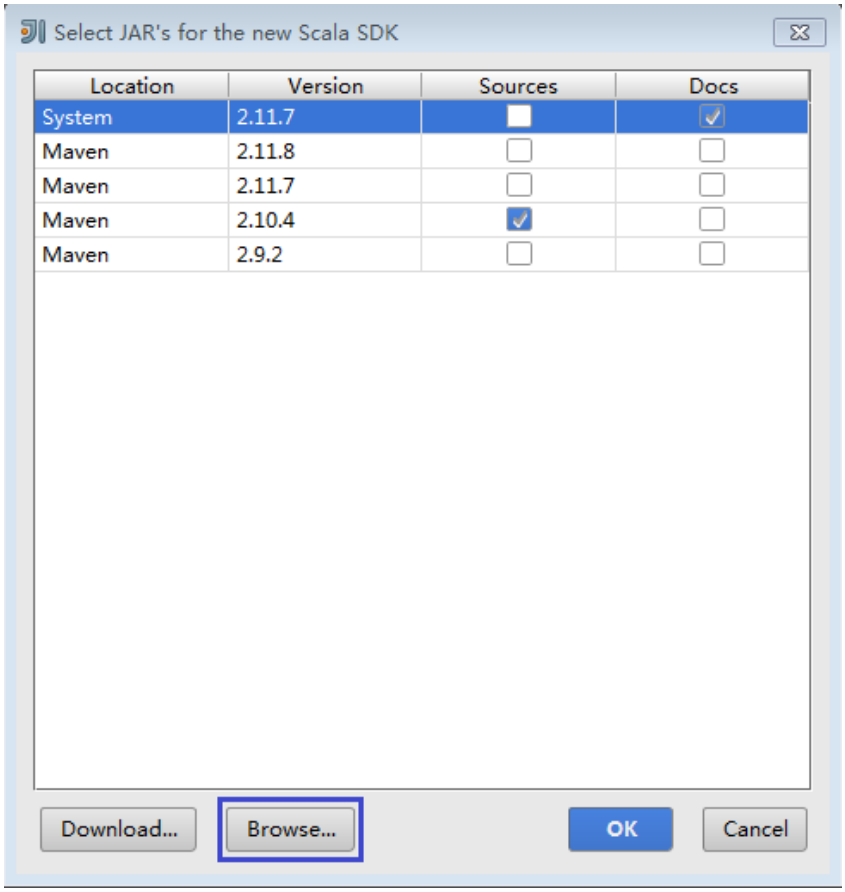
4. If IDEA fails to identify Scala SDK, create a Scala SDK.
   a. Click **Create...**.

Figure 11-25 Create...



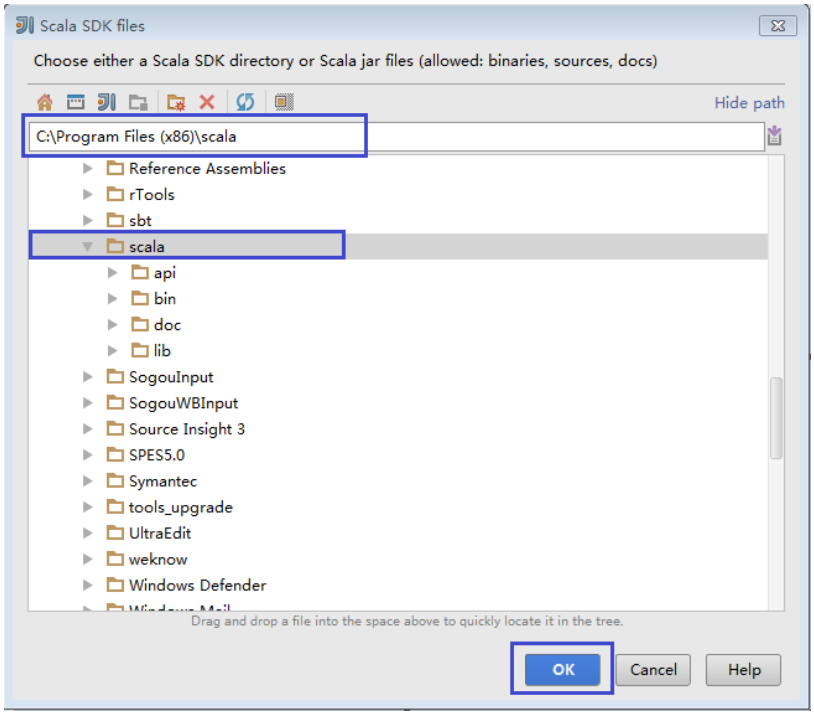
b. On the **Select JAR's for the new Scala SDK** page, click **Browse...**.

Figure 11-26 Select JAR's for the new Scala SDK



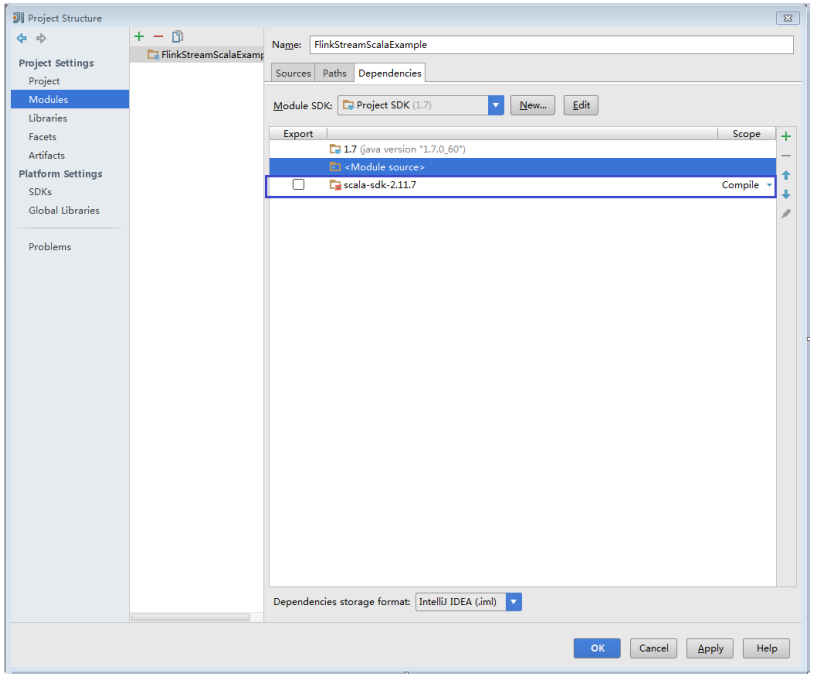
c. On the **Scala SDK files** page, select the **scala sdk** directory and click **OK**.

Figure 11-27 Scala SDK files



5. Click **OK**.

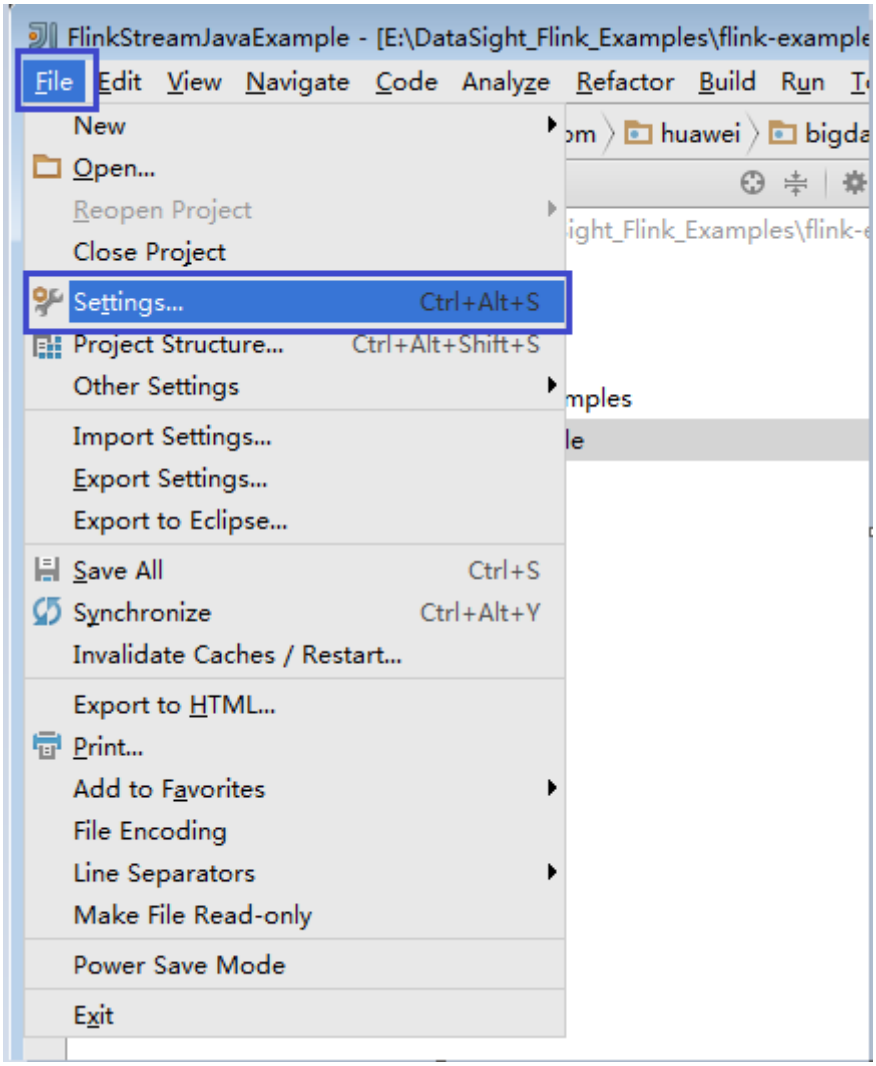
Figure 11-28 Configuration successful



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

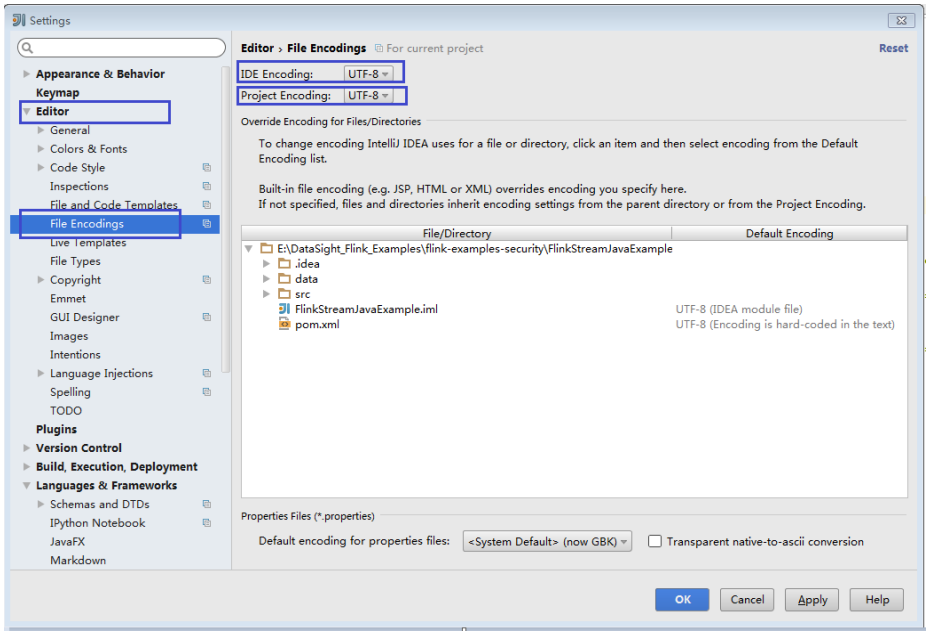
1. On the IDEA home page, choose **File** > **Settings...**.

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



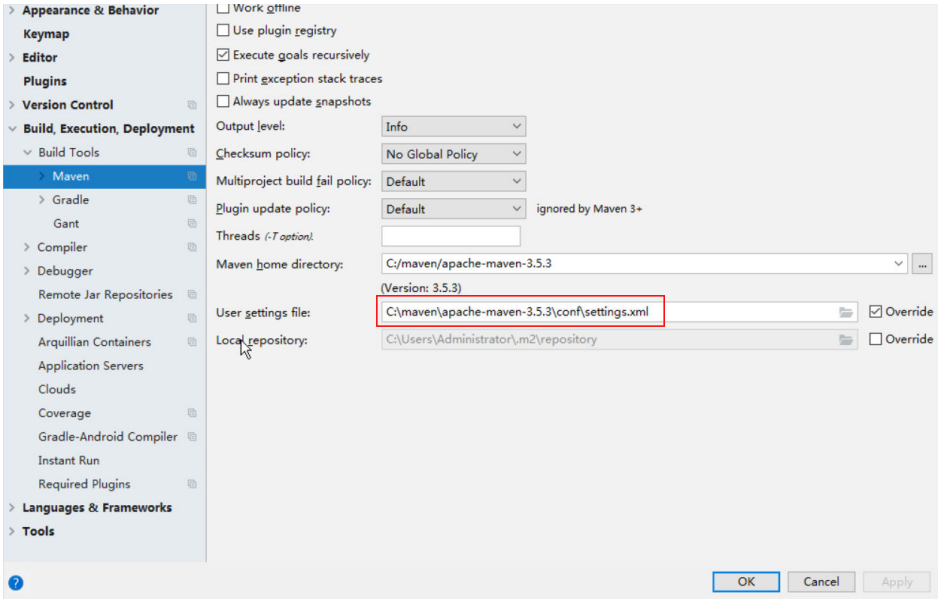
NOTE

- 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 about the dependency package for running the sample project**.

Step 8 Configure Maven.

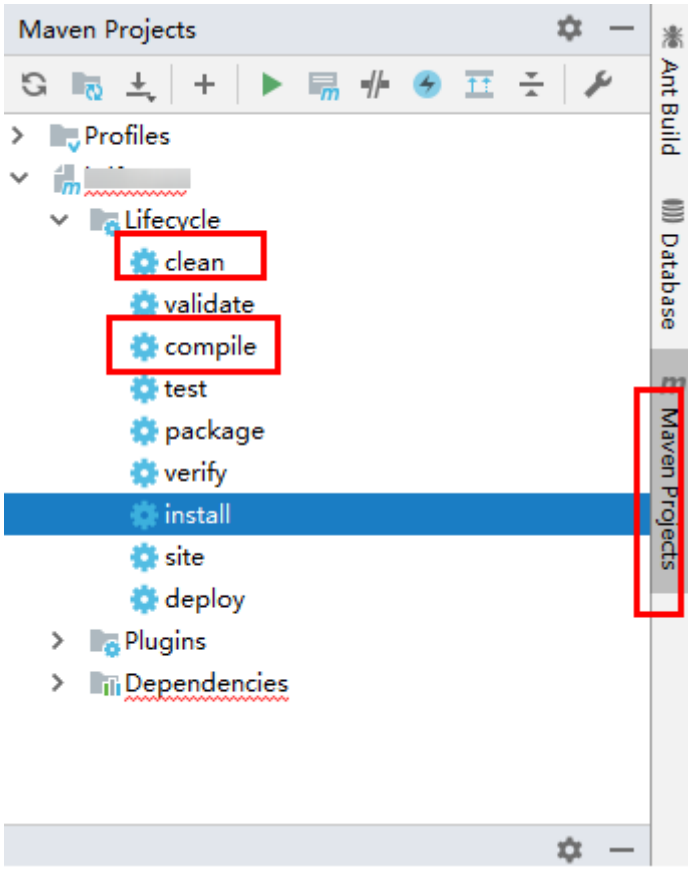
1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to **Configuring Huawei Open Source Mirrors**.
2. On the IntelliJ IDEA page, select **File** > **Settings** > **Build**, **Execution**, **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 about the dependency package for running the sample project

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:

Table 11-5 Dependency package for running the sample project

Sample project	Dependency package	Obtaining Dependency Packages
• Sample project of the DataStream application • Sample project of the asynchronous checkpoint mechanism application	flink-dist_*.jar	Can be obtained from **lib** in the installation directory of the Flink client or server.
• Sample projects of Submitting SQL Jobs Using Flink Jar • Sample projects of FlinkServer REST API	• flink-dist_*.jar • flink-table_*.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	• kafka-clients-*.jar • flink-connector-kafka_*.jar	• **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. • flink-connector-kafka_*.jar can be obtained from **opt** in the installation directory of the Flink client or server.
Sample projects of pipeline	• flink-connector-netty_*.jar • flink-dist_*.jar	• flink-connector-netty_*.jar can be obtained from the **lib** folder generated after the secondary development sample code is compiled. • flink-dist_*.jar can be obtained from **lib** in the installation directory of the Flink client or server.

Sample project	Dependency package	Obtaining Dependency Packages
Sample projects of Stream SQL JOIN	• kafka-clients-*.jar • flink-connector-kafka_*.jar • flink-dist_*.jar • flink-table_*.jar	• **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. • flink-connector-kafka_*.jar can be obtained from **opt** in the installation directory of the Flink client or server. • flink-dist_*.jar, flink-table_*.jar can be obtained from **lib** in the installation directory of the Flink client or server.
Sample projects of HBase	• flink-connector-hbase*.jar • flink-dist_*.jar • flink-table_*.jar • hbase-clients-*.jar	• flink-connector-hbase_*.jar can be obtained from **opt** in the installation directory of the Flink client or server. • flink-dist_*.jar, flink-table_*.jar can be obtained from **lib** in the installation directory of the Flink client or server. • **hbase-clients-*.jar** is provided by HBase and can be obtained from the **lib** directory in the installation directory of the HBase client or server.
Sample projects of Hudi	hbase-unsafe-*.jar	Can be obtained from the **lib** folder generated after the secondary development sample code is compiled.

11.2.3 Creating a Project (Optional)

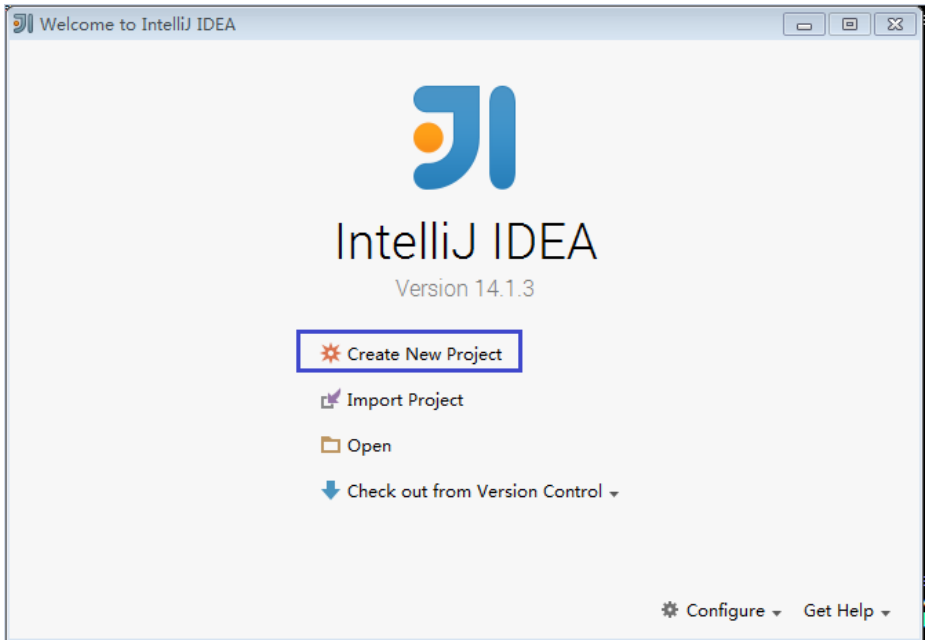
Scenarios

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

Procedure

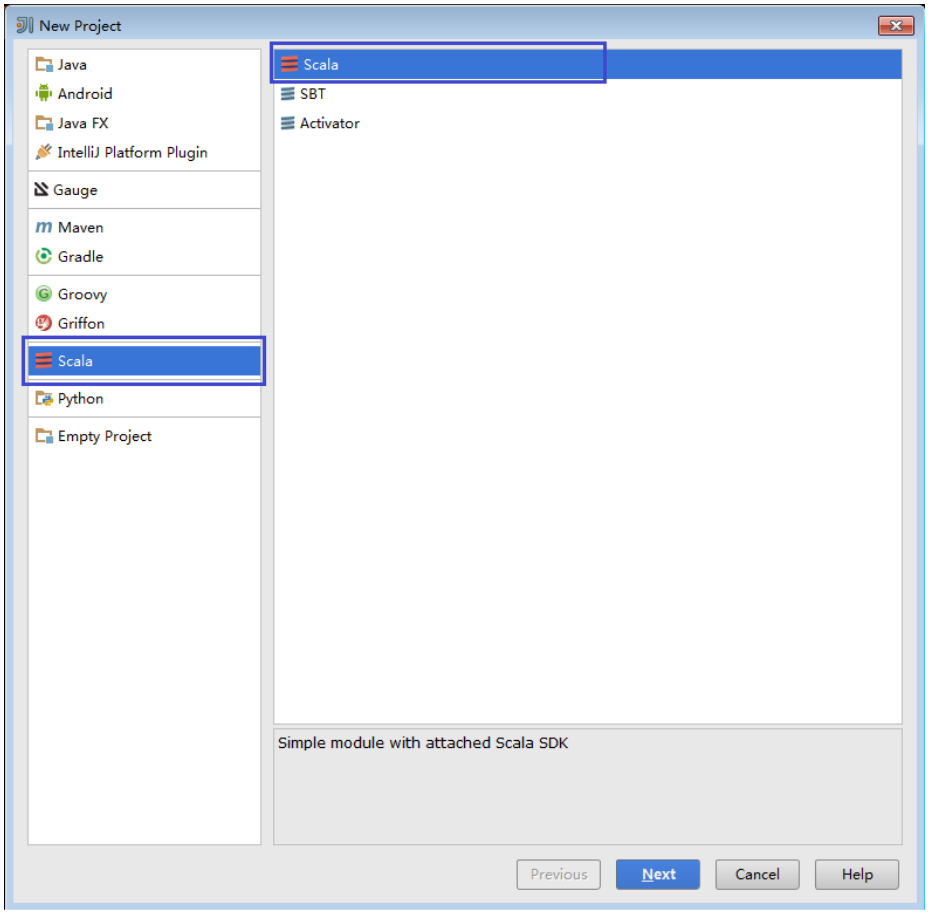
Step 1 Start the IDEA and choose **Create New Project**.

Figure 11-33 Creating a project



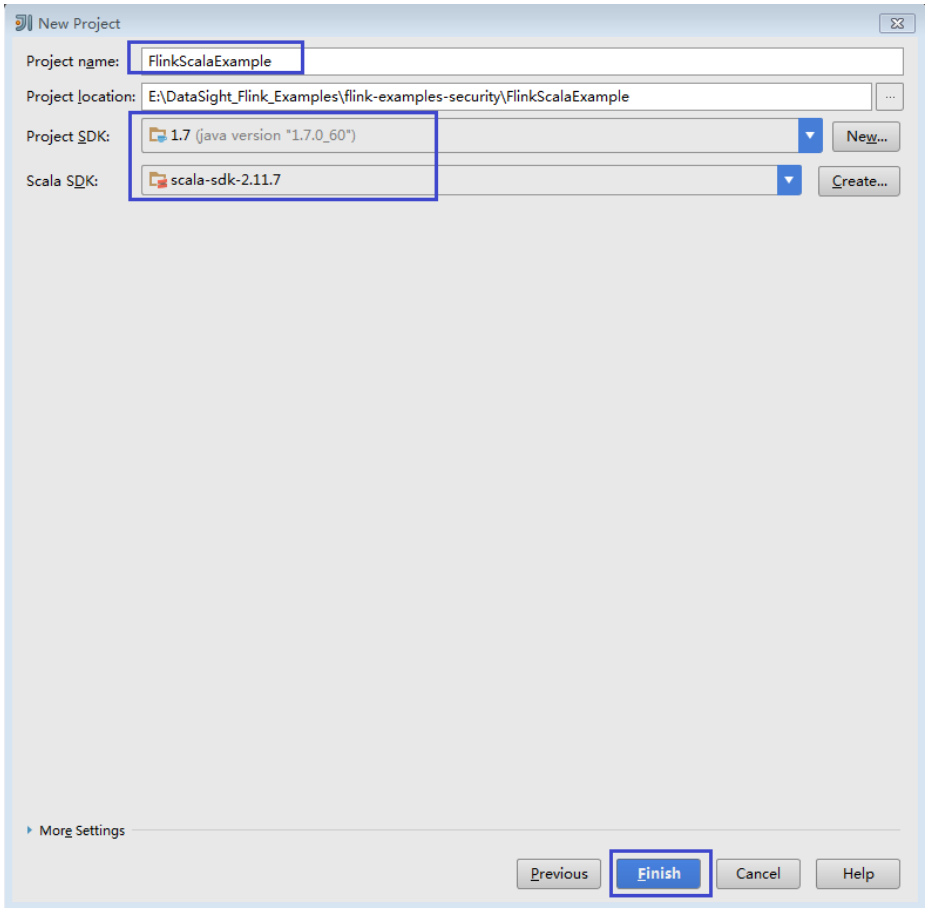
Step 2 On the **New Project** page, choose a Scala development environment, choose **Scala Module**, and then click **Next**. If you want to create a Java project, choose corresponding parameters of Java.

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

11.2.4 Configuring a Spring Boot Sample Project

This section applies to MRS 3.3.0 or later.

Scenarios

Run the sample code of the Spring Boot interface in FusionInsight MRS Hive. Currently, GaussDB(DWS) SpringBoot sample projects are supported.

In the following example, an application is developed in Linux to connect GaussDB(DWS) to Flink using Spring Boot.

NOTE

Before run the GaussDB(DWS) sample, log in to the node where GaussDB(DWS) is deployed to create an empty table **test_lzh1** for receiving data. The creation command is as follows:

create table test_lzh1 (id integer not null);

Procedure

Step 1 Install a client that contains the Flink service.

The following example shows how to install the Flink client on a node in the cluster:

1. Log in to FusionInsight Manager. On the home page, click **Download Client**. The **Download Cluster Client** dialog box is displayed.
2. In the **Download Cluster Client** dialog box that is displayed, select **Complete Client** for **Select Client Type**, select the platform type that matches the architecture of the node where the client is to install, select **Save to Path**, and click **OK**.

   The generated file is stored in the **/tmp/FusionInsight-Client** directory on the active management node by default.

   The name of the client software package is in the following format: **FusionInsight_Cluster_***<Cluster ID>***_Services_Client.tar**. The following content uses the cluster ID **1** as an example.
3. Log in to the server where the client is to be installed as the client installation user.
4. Go to the directory where the installation package is stored and run the following commands to decompress the package:

   cd /tmp/FusionInsight-Client

   tar -xvf FusionInsight_Cluster_1_Services_Client.tar
5. Run the following command to verify the decompressed file and check whether the command output is consistent with the information in the **sha256** file:

   sha256sum -c FusionInsight_Cluster_1_Services_ClientConfig.tar.sha256
6. Decompress the obtained installation file.

   tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
7. Go to the directory where the installation package is stored, and run the following command to install the client to a specified directory (absolute path), for example, **/opt/hadoopclient**:

   cd /tmp/FusionInsight-Client/FusionInsight_Cluster_1_Services_ClientConfig

   ./install.sh /opt/hadoopclient
8. Configure security authentication and encryption. For details, see "Using Flink from Scratch" in the *Component Operation Guide*.

Step 2 Obtain the sample project folder **flink-dws-sink-example** from **src\springboot\flink-examples** in the directory where the sample code is decompressed. For details, see **Obtaining Sample Projects from Huawei Mirrors**.

Step 3 Import the sample project to the IntelliJ IDEA development environment.

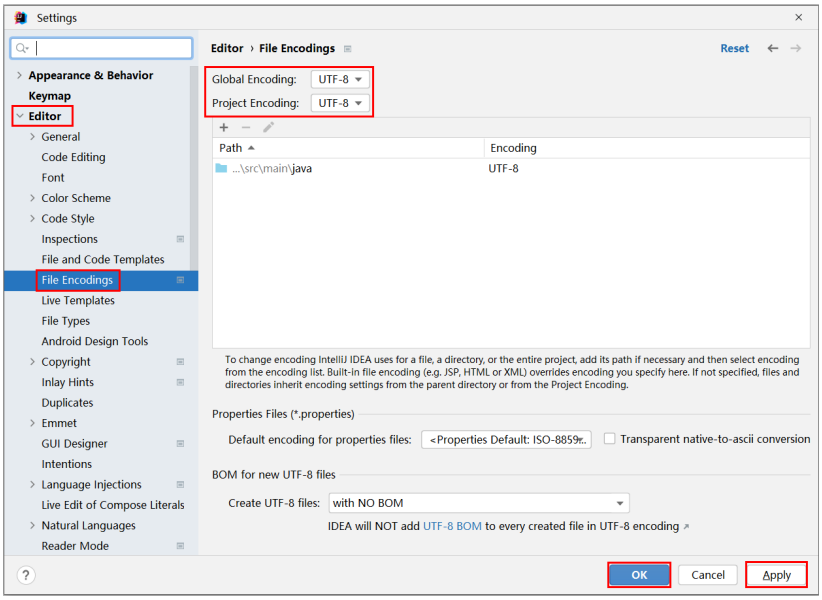
1. On the menu bar of IntelliJ IDEA, choose **File** > **Open...**.
2. In the **Open File or Project** dialog box that is displayed, select the **flink-dws-sink-example** folder and click **OK**.

Step 4 Set the IntelliJ IDEA text file coding format to prevent garbled characters.

1. On the menu bar of IntelliJ IDEA, choose **File** > **Settings**. The **Settings** window is displayed.

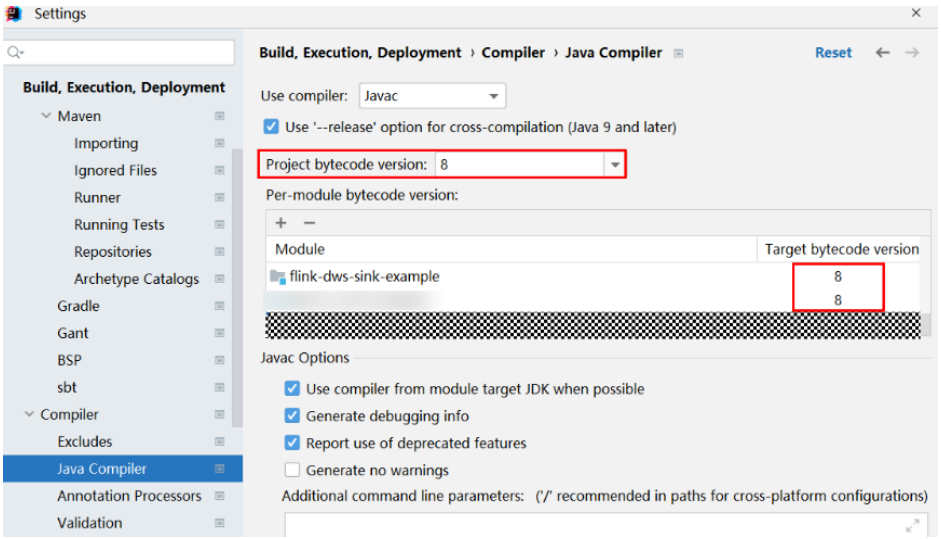
2. In the left navigation pane, choose **Editor** > **File Encodings**. In the **Project Encoding** and **Global Encoding** areas, set the parameter to **UTF-8**. Click **Apply** and **OK**.

Figure 11-36 Setting the IntelliJ IDEA coding format

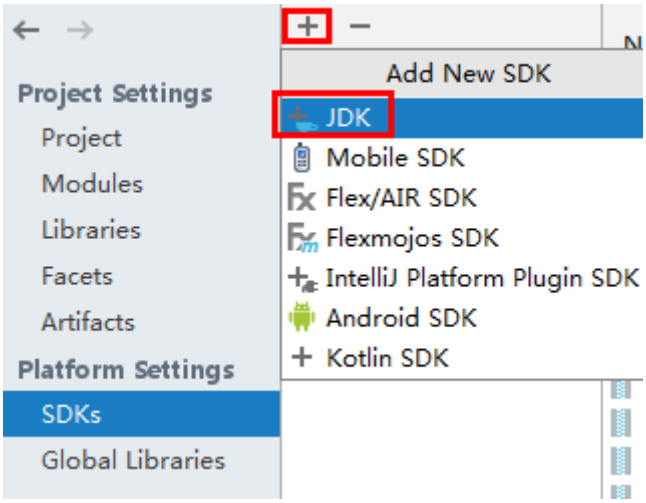


Step 5 Set the JDK of the project.

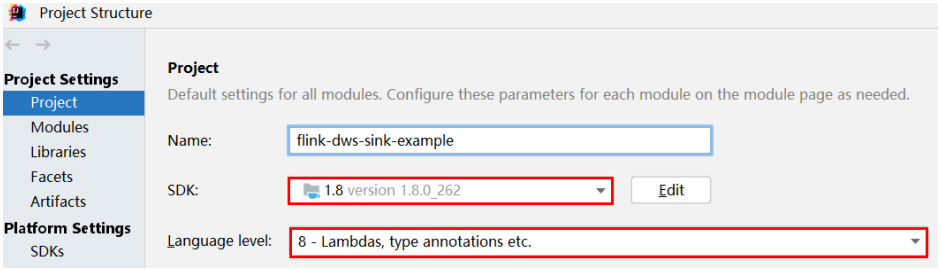
1. On the menu bar of IntelliJ IDEA, choose **File** > **Settings**. The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment** > **Compiler** > **Java Compiler** and select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** in **flink-dws-sink-example** to **8**.



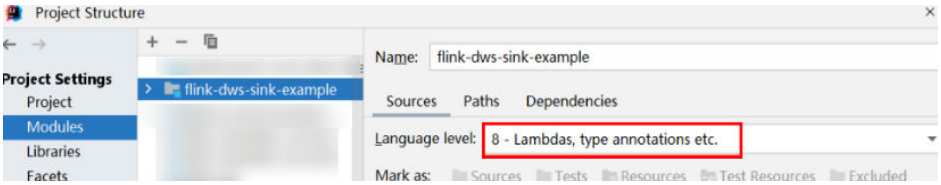
3. Click **Apply** then **OK**.
4. On the menu bar of IntelliJ IDEA, choose **File** > **Project Structure...**. The **Project Structure** window is displayed.
5. Choose **SDKs**, click the plus sign (+), and select **JDK**.



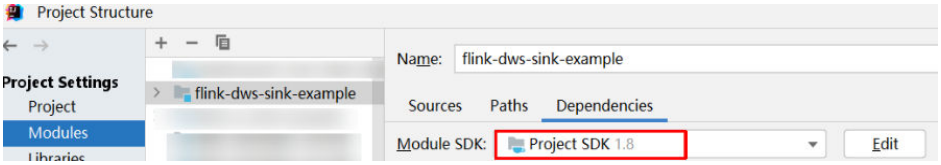
6. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.
7. Click **Apply**.
8. Select **Project**, select the JDK added in **SDKs** from the **SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Sources** tab page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**.



On the **Dependencies** tab page, change the value of **Module SDK** to the JDK added in **SDKs**.



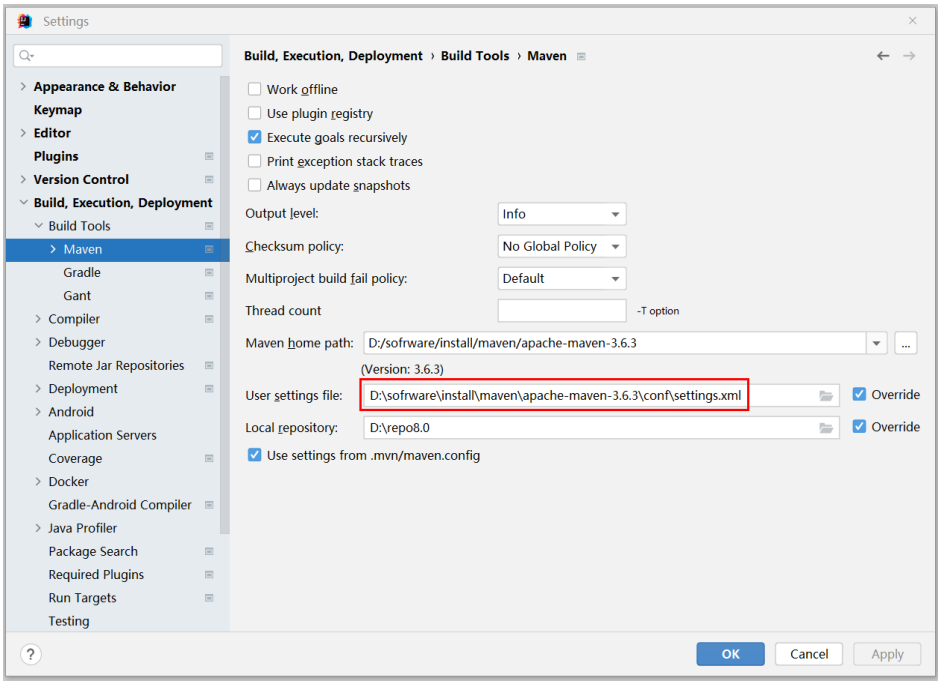
11. Click **Apply** and then **OK**.

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, choose **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 11-37 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 **OK**.

Step 7 Find the **application.properties** file in **src\main\resources** and add the following content to the file:

- Run the GaussDB(DWS) sample

```
spring.datasource.dws.url=jdbc:postgresql://IP address of the GaussDB(DWS) node:8000/postgres
spring.datasource.dws.username=dbadmin
spring.datasource.dws.password=Password of dbadmin
spring.datasource.dws.driver=org.postgresql.Driver
```

----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;
```

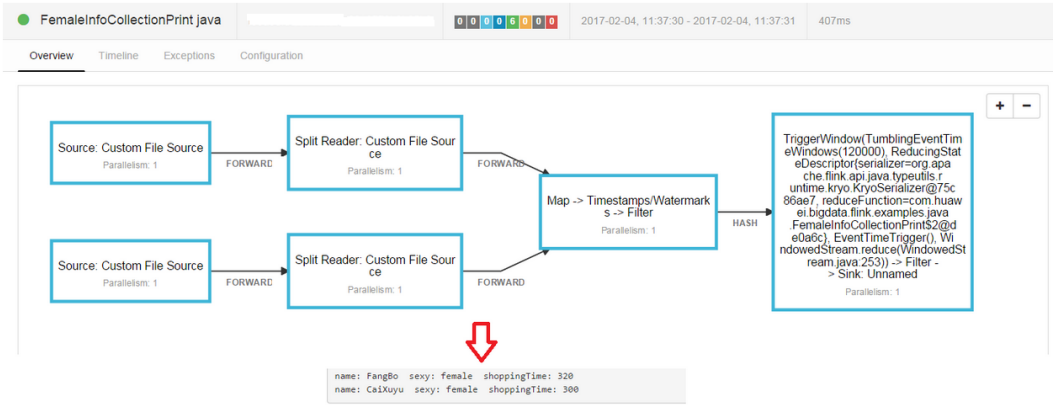
```
        public UserRecord(String n, String s, int t) {
            name = n;
            sexy = s;
            shoppingTime = t;
        }

        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 11-38 shows the process of execution.

Figure 11-38 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.
```

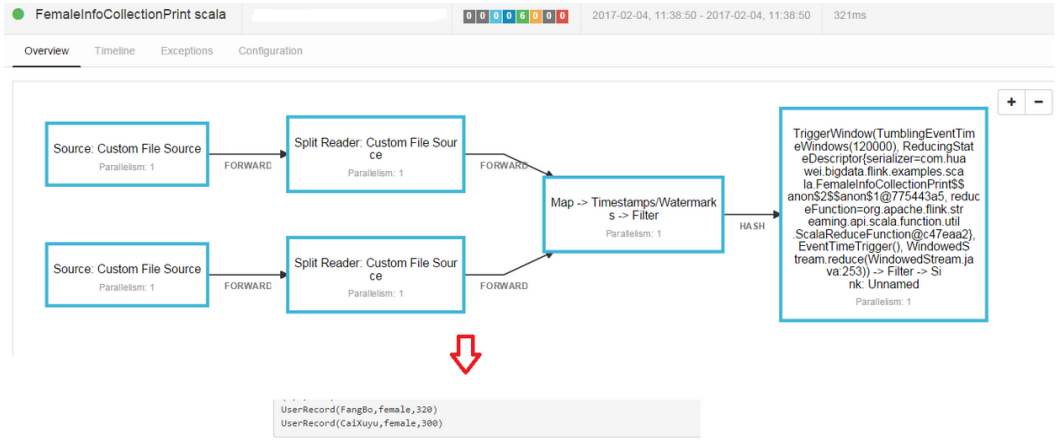
```
 private class Record2TimestampExtractor extends AssignerWithPunctuatedWatermarks[UserRecord] {

   //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 11-39 shows the process of execution.

Figure 11-39 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 11-6

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.
{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 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()
  }
}
```

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.Seriablizale;

//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 =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuv
wxyzABCDEFGHIJKLMNOPQRSTUVWZYX0987654321";

    //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, Stirng, 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 processing time 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 =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuv
wxyzABCDEFGHIJKLMNOPQRSTUVWZYX0987654321"

    //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) throw 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.utils.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.utils.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.utils.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 c**om.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("******************************************************************************************");
        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 xxxx.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 xxxx.xxx.xxx.xxx:21007 --security.protocol
SASL_PLAINTEXT --sasl.kerberos.service.name kafka"
            + "--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();
```



```
    }
}
```

11.3.5.3 Scala Sample Code

This section applies to MRS 3.3.0 or later.

Function

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

Sample Code

If you need to use Kafka in security mode before development, you need to import kafka-clients-*.jar of FusionInsight. You can obtain the JAR package from the Kafka client directory.

The following example shows the Producer, Consumer, and the main logic code used by Flink Stream SQL Join.

For the complete codes, see **com.huawei.bigdata.flink.examples.WriteIntoKafka** and **com.huawei.bigdata.flink.examples.SqlJoinWithSocket**.

1. Produce a piece of user information in Kafka every second. The user information includes the name, age, and gender.

```
// Kafka Producer code

object WriteIntoKafka {
  def main(args: Array[String]): Unit = {
    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 xxx.xxx.xxx.xxx:21005")
    System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/
test.jar --topic" + " topic-test -bootstrap.servers xxx.xxx.xxx.xxx:21007 --security.protocol
SASL_PLAINTEXT" + " --sasl.kerberos.service.name kafka")
    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("******************************************************************************************")
    val env = StreamExecutionEnvironment.getExecutionEnvironment
env.setParallelism(1)

    val paraTool = ParameterTool.fromArgs(args)

    val messageStream = env.addSource(new WriteIntoKafka.SimpleStringGenerator)
    val producer = new FlinkKafkaProducer[String](paraTool.get("topic"), new SimpleStringSchema,
paraTool.getProperties)

    producer.setWriteTimestampToKafka(true)

    messageStream.addSink(producer)
    env.execute
  }

  /**
* String source class
   *
   */
  object SimpleStringGenerator {
    private[examples] val NAME = Array("Carry", "Alen", "Mike", "Ian", "John", "Kobe", "James")
    private[examples] val SEX = Array("MALE", "FEMALE")
    private[examples] val COUNT = NAME.length
```

```
  }

  class SimpleStringGenerator extends SourceFunction[String] {
    private[examples] var running = true
    private[examples] val rand = new Random(47)

    @throws[Exception]
    override def run(ctx: SourceFunction.SourceContext[String]): Unit = {
      while (running) {
        val i = rand.nextInt(SimpleStringGenerator.COUNT)
        val age = rand.nextInt(70)
        val sexy = SimpleStringGenerator.SEX(rand.nextInt(2))
        ctx.collect(SimpleStringGenerator.NAME(i) + "," + age + "," + sexy)
        Thread.sleep(1000)
      }
    }

    override def cancel(): Unit = {
      running = false
    }
  }

}
```

2. Generate Table1 and Table2, use **Join** to jointly query Table1 and Table2, and print the output result.

```
object SqlJoinWithSocket {
  def main(args: Array[String]): Unit = {
    var hostname: String = null
    var port = 0
    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 xxxx.xxx.xxx.xxx:21005 --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 xxxx.xxx.xxx.xxx:21007 --security.protocol
SASL_PLAINTEXT" + " --sasl.kerberos.service.name kafka--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 {
      val params = ParameterTool.fromArgs(args)
      hostname = if (params.has("hostname")) params.get("hostname")
      else "localhost"
      port = params.getInt("port")
    } catch {
      case e: Exception =>
        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
    }

    val fsSettings = EnvironmentSettings.newInstance.inStreamingMode.build
    val env = StreamExecutionEnvironment.getExecutionEnvironment
val tableEnv = StreamTableEnvironment.create(env, fsSettings)

    env.getConfig.setAutoWatermarkInterval(200)
    env.setParallelism(1)
    val paraTool = ParameterTool.fromArgs(args)

    val kafkaStream = env.addSource(new FlinkKafkaConsumer[String](paraTool.get("topic"), new
SimpleStringSchema, paraTool.getProperties)).map(new MapFunction[String, Tuple3[String, String,
String]]() {
      @throws[Exception]
      override def map(str: String): Tuple3[String, String, String] = {
        val word = str.split(",")
```

```
        new Tuple3[String, String, String](word(0), word(1), word(2))
      }
    })

    tableEnv.createTemporaryView("Table1", kafkaStream, $("name"), $("age"), $("sexy"), $
("proctime").proctime)

    val socketStream = env.socketTextStream(hostname, port, "\n").map(new MapFunction[String,
Tuple2[String, String]]() {
      @throws[Exception]
      override def map(str: String): Tuple2[String, String] = {
        val words = str.split("\\s")
        if (words.length < 2) return new Tuple2[String, String]
        new Tuple2[String, String](words(0), words(1))
      }
    })

    tableEnv.createTemporaryView("Table2", socketStream, $("name"), $("job"), $
("proctime").proctime)

    val 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")

    tableEnv.toAppendStream(result, classOf[Row]).print
    env.execute
  }

}
```

11.3.6 Submitting a SQL Job Using Flink Jar

11.3.6.1 Scenario Description

This scenario applies to MRS 3.2.1 or later.

Description

If SQL statements of a job are frequently modified, submit Flink SQL statements in Flink Jar mode to reduce your workload.

Development Guideline

Use the current sample to submit and execute specified SQL statements. Use semicolons (;) to separate multiple statements.

11.3.6.2 Java Sample Code

The core logic for submitting SQL statements is as follows. Currently, only **CREATE** and **INSERT** statements can be submitted. For details about the complete code, see com.huawei.bigdata.flink.examples.FlinkSQLExecutor.

```
public class FlinkSQLExecutor {
    public static void main(String[] args) throws IOException {
        System.out.println("----------------------  begin init ----------------------");
        final String sqlPath = ParameterTool.fromArgs(args).get("sql", "config/redisSink.sql");
        final StreamExecutionEnvironment streamEnv =
StreamExecutionEnvironment.getExecutionEnvironment();
        EnvironmentSettings bsSettings = EnvironmentSettings.newInstance().inStreamingMode().build();
        StreamTableEnvironment tableEnv = StreamTableEnvironment.create(streamEnv, bsSettings);
        StatementSet statementSet = tableEnv.createStatementSet();
        String sqlStr = FileUtils.readFileToString(FileUtils.getFile(sqlPath), "utf-8");
```



```
        String[] sqlArr = sqlStr.split(";");
        for (String sql : sqlArr) {
            sql = sql.trim();
            if (sql.toLowerCase(Locale.ROOT).startsWith("create")) {
                System.out.println("----------------------------------------------\nexecuteSql=\n" + sql);
                tableEnv.executeSql(sql);
            } else if (sql.toLowerCase(Locale.ROOT).startsWith("insert")) {
                System.out.println("----------------------------------------------\ninsert=\n" + sql);
                statementSet.addInsertSql(sql);
            }
        }
        System.out.println("---------------------- begin exec sql ---------------------------");
        statementSet.execute();
    }
}
```

NOTE

Copy the dependency package required by the current sample, that is, the JAR package in the **lib** file after compilation, to the **lib** folder on the client.

The following uses Kafka in a normal cluster as an example to describe how to submit SQL statements:

```
create table kafka_sink
(
    uuid varchar(20),
    name varchar(10),
    age  int,
    ts   timestamp(3),
    p    varchar(20)
) with (
      'connector' = 'kafka',
      'topic' = 'input2',
      'properties.bootstrap.servers' = 'IP address of the Kafka broker instance',
      'properties.group.id' = 'testGroup2',
      'scan.startup.mode' = 'latest-offset',
      'format' = 'json'
      );
create TABLE datagen_source
(
    uuid varchar(20),
    name varchar(10),
    age  int,
    ts   timestamp(3),
    p    varchar(20)
) WITH (
      'connector' = 'datagen',
      'rows-per-second' = '1'
      );
INSERT INTO kafka_sink
SELECT *
FROM datagen_source;
```

11.3.7 FlinkServer REST API JavaExample

11.3.7.1 Accessing Flinkserver RESTful API as a Proxy User

This section applies to MRS 3.3.0 or later.

Function

Call the FlinkServer RESTful API as a proxy user. Use a proxy to access the API as a FlinkServer administrator to obtain common user permissions.

Sample Code

Assume that the tenant user is **test92**, the tenant ID is **92**, and the user name is **flinkserveradmin** with FlinkServer administrator permissions. The following code is a complete example.

```
public class TestCreateTenants {
    public static void main(String[] args) {
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        final String hostName = paraTool.get("hostName");    // Replace hostName in the hosts file with the actual host name.
        final String keytab = paraTool.get("keytab");        // user.keytab file path
        final String krb5 = paraTool.get("krb5");            // krb5.conf file path
        final String principal = paraTool.get("principal");  // Authentication user

        System.setProperty("java.security.krb5.conf", krb5);
        String url = "https://"+hostName+":28943/flink/v1/tenants";
        String jsonstr = "{" +
                "\n\t \"tenantId\":\"92\"," +
                "\n\t \"tenantName\":\"test92\"," +
                "\n\t \"remark\":\"test tenant remark1\"," +
                "\n\t \"updateUser\":\"test_updateUser1\"," +
                "\n\t \"createUser\":\"test_createUser1\"" +
                "\n}";

        try {
            LoginClient.getInstance().setConfigure(url, principal, keytab, "");
            LoginClient.getInstance().login(); // Log in as the FlinkServer administrator.

            String proxyUrl =  "https://"+hostName+":28943/flink/v1/proxyUserLogin"; // Call the proxy user API to obtain the common user token.
            String result = HttpClientUtil.doPost(proxyUrl, "{\n" +
                    "\t\"realUser\": \"flinkserveradmin\"\n" +
                    "}", "utf-8", true);
            Gson gson = new Gson();
            JsonObject jsonObject = gson.fromJson(result, JsonObject.class);
            String token = jsonObject.get("result").toString();
            token = "hadoop_auth=" + token;

            System.out.println(HttpClientUtil.doPost(url, jsonstr, "utf-8", true , token));
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

11.3.8 Flink Reading Data from and Writing Data to HBase

11.3.8.1 Scenario Description

This section applies to MRS 3.2.0 or later.

Typical Scenario Description

Use Flink API jobs to read data from and write data to HBase.

Data Preparation

Prepare the HBase configuration file and download the cluster configuration on FusionInsight Manager to obtain the **hbase-site.xml** file.

Development Guideline

1. Writes data to HBase:
   a. Specify the parent directory of the **hbase-site.xml** file. Flink Sink can obtain the HBase connection.
   b. Use the connection to determine whether a table exists. If it does not, create one.
   c. Convert received data into Put objects and writes the Put objects to HBase.
2. Reads data from HBase:
   a. Specify the parent directory of the **hbase-site.xml** file. Flink Source can obtain the HBase connection.
   b. Use the connection to determine whether a table exists. If it does not, the job fails. In this case, you need to create a table in HBase shell or an upstream job.
   c. Read data from HBase, converts result data into Row objects, and sends the Row objects to downstream operators.

11.3.8.2 Java Sample Code

Description

Call Flink APIs to read data from and write data to HBase.

Sample Code

The following example shows the main logic code of WriteHBase and ReadHBase.

For details about the complete code, see **com.huawei.bigdata.flink.examples.WriteHBase** and **com.huawei.bigdata.flink.examples.ReadHBase**.

- Main logic code of WriteHBase

```
public static void main(String[] args) throws Exception {
        System.out.println("use command as: ");
        System.out.println(
                "./bin/flink run --class com.huawei.bigdata.flink.examples.WriteHBase"
                        + " /opt/test.jar --tableName t1 --confDir /tmp/hbaseConf");
        System.out.println(
                "********************************************************************************************");
        System.out.println("<tableName> hbase tableName");
        System.out.println("<confDir> hbase conf dir");
        System.out.println(
                "********************************************************************************************");
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        env.setParallelism(1);
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        DataStream<Row> messageStream = env.addSource(new SimpleStringGenerator());
        messageStream.addSink(
                new HBaseWriteSink(paraTool.get("tableName"), createConf(paraTool.get("confDir"))));
        env.execute("WriteHBase");
    }

    private static org.apache.hadoop.conf.Configuration createConf(String confDir) {
        LOG.info("Create HBase configuration.");
        org.apache.hadoop.conf.Configuration hbaseConf =
HBaseConfigurationUtil.getHBaseConfiguration();
```



```
        if (confDir != null) {
            File hbaseSite = new File(confDir + File.separator + "hbase-site.xml");
            if (hbaseSite.exists()) {
                LOG.info("Add hbase-site.xml");
                hbaseConf.addResource(new Path(hbaseSite.getPath()));
            }
            File coreSite = new File(confDir + File.separator + "core-site.xml");
            if (coreSite.exists()) {
                LOG.info("Add core-site.xml");
                hbaseConf.addResource(new Path(coreSite.getPath()));
            }
            File hdfsSite = new File(confDir + File.separator + "hdfs-site.xml");
            if (hdfsSite.exists()) {
                LOG.info("Add hdfs-site.xml");
                hbaseConf.addResource(new Path(hdfsSite.getPath()));
            }
        }
        LOG.info("HBase configuration created successfully.");
        return hbaseConf;
    }

    /**
     * @since 8.2.0
     */
    private static class HBaseWriteSink extends RichSinkFunction<Row> {
        private Connection conn;
        private BufferedMutator bufferedMutator;
        private String tableName;
        private final byte[] serializedConfig;
        private Admin admin;
        private org.apache.hadoop.conf.Configuration hbaseConf;
        private long flushTimeIntervalMillis = 5000; //5s
        private long preFlushTime;

        public HBaseWriteSink(String sourceTable, org.apache.hadoop.conf.Configuration conf) {
            this.tableName = sourceTable;
            this.serializedConfig = HBaseConfigurationUtil.serializeConfiguration(conf);
        }

        private void deserializeConfiguration() {
            LOG.info("Deserialize HBase configuration.");
            hbaseConf = HBaseConfigurationUtil.deserializeConfiguration(
                serializedConfig, HBaseConfigurationUtil.getHBaseConfiguration());
            LOG.info("Deserialization successfully.");
        }

        private void createTable() throws IOException {
            LOG.info("Create HBase Table.");
            if (admin.tableExists(TableName.valueOf(tableName))) {
                LOG.info("Table already exists.");
                return;
            }
            // Specify the table descriptor.
            TableDescriptorBuilder htd =
TableDescriptorBuilder.newBuilder(TableName.valueOf(tableName));
            // Set the column family name to f1.
            ColumnFamilyDescriptorBuilder hcd =
ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("f1"));
            // 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
            hcd.setCompressionType(Compression.Algorithm.SNAPPY);
            htd.setColumnFamily(hcd.build());
            try {
                admin.createTable(htd.build());
            } catch (IOException e) {
                if (!(e instanceof TableExistsException)
                || !admin.tableExists(TableName.valueOf(tableName))) {
```



```
                throw e;
            }
            LOG.info("Table already exists, ignore.");
        }
        LOG.info("Table created successfully.");
    }

    @Override
    public void open(Configuration parameters) throws Exception {
        LOG.info("Write sink open");
        super.open(parameters);
        deserializeConfiguration();
        conn = ConnectionFactory.createConnection(hbaseConf);
        admin = conn.getAdmin();
        createTable();
        bufferedMutator = conn.getBufferedMutator(TableName.valueOf(tableName));
        preFlushTime = System.currentTimeMillis();
    }

    @Override
    public void close() throws Exception {
        LOG.info("Close HBase Connection.");
        try {
            if (admin != null) {
                admin.close();
                admin = null;
            }
            if (bufferedMutator != null) {
                bufferedMutator.close();
                bufferedMutator = null;
            }
            if (conn != null) {
                conn.close();
                conn = null;
            }
        } catch (IOException e) {
            LOG.error("Close HBase Exception:", e);
            throw new RuntimeException(e);
        }
        LOG.info("Close successfully.");
    }

    @Override
    public void invoke(Row value, Context context) throws Exception {
        LOG.info("Write data to HBase.");
        Put put = new Put(Bytes.toBytes(value.getField(0).toString()));
        put.addColumn(Bytes.toBytes("f1"), Bytes.toBytes("q1"),
(Bytes.toBytes(value.getField(1).toString())));
        bufferedMutator.mutate(put);

        if (preFlushTime + flushTimeIntervalMillis >= System.currentTimeMillis()) {
            LOG.info("Flush data to HBase.");
            bufferedMutator.flush();
            preFlushTime = System.currentTimeMillis();
            LOG.info("Flush successfully.");
        } else {
            LOG.info("Skip Flush.");
        }

        LOG.info("Write successfully.");
    }
}

/**
 * @since 8.2.0
 */
public static class SimpleStringGenerator implements SourceFunction<Row> {
    private static final long serialVersionUID = 2174904787118597072L;
    boolean running = true;
```

```
    long i = 0;
    Random random = new Random();

    @Override
    public void run(SourceContext<Row> ctx) throws Exception {
        while (running) {
            Row row = new Row(2);
            row.setField(0, "rk" + random.nextLong());
            row.setField(1, "v" + random.nextLong());
            ctx.collect(row);
            Thread.sleep(1000);
        }
    }

    @Override
    public void cancel() {
        running = false;
    }
}
```

- Main logic code of ReadHBase

```
public static void main(String[] args) throws Exception {
    System.out.println("use command as: ");
    System.out.println(
            "./bin/flink run --class com.huawei.bigdata.flink.examples.ReadHBase"
                    + " /opt/test.jar --tableName t1 --confDir /tmp/hbaseConf");

    System.out.println(
            "********************************************************************************************");
    System.out.println("<tableName> hbase tableName");
    System.out.println("<confDir> hbase conf dir");
    System.out.println(
            "********************************************************************************************");
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    env.setParallelism(1);
    ParameterTool paraTool = ParameterTool.fromArgs(args);
    DataStream<Row> messageStream =
            env.addSource(
                    new HBaseReaderSource(
                            paraTool.get("tableName"), createConf(paraTool.get("confDir"))));
    messageStream
            .rebalance()
            .map(
                    new MapFunction<Row, String>() {
                        @Override
                        public String map(Row s) throws Exception {
                            return "Flink says " + s + System.getProperty("line.separator");
                        }
                    })
            .print();
    env.execute("ReadHBase");
}

private static org.apache.hadoop.conf.Configuration createConf(String confDir) {
    LOG.info("Create HBase configuration.");
    org.apache.hadoop.conf.Configuration hbaseConf =
HBaseConfigurationUtil.getHBaseConfiguration();
    if (confDir != null) {
        File hbaseSite = new File(confDir + File.separator + "hbase-site.xml");
        if (hbaseSite.exists()) {
            LOG.info("Add hbase-site.xml");
            hbaseConf.addResource(new Path(hbaseSite.getPath()));
        }
        File coreSite = new File(confDir + File.separator + "core-site.xml");
        if (coreSite.exists()) {
            LOG.info("Add core-site.xml");
            hbaseConf.addResource(new Path(coreSite.getPath()));
        }
        File hdfsSite = new File(confDir + File.separator + "hdfs-site.xml");
        if (hdfsSite.exists()) {
```



```
            LOG.info("Add hdfs-site.xml");
            hbaseConf.addResource(new Path(hdfsSite.getPath()));
        }
    }
    LOG.info("HBase configuration created successfully.");
    return hbaseConf;
}

private static class HBaseReaderSource extends RichSourceFunction<Row> {

    private Connection conn;
    private Table table;
    private Scan scan;
    private String tableName;
    private final byte[] serializedConfig;
    private Admin admin;
    private org.apache.hadoop.conf.Configuration hbaseConf;

    public HBaseReaderSource(String sourceTable, org.apache.hadoop.conf.Configuration conf) {
        this.tableName = sourceTable;
        this.serializedConfig = HBaseConfigurationUtil.serializeConfiguration(conf);
    }

    @Override
    public void open(Configuration parameters) throws Exception {
        LOG.info("Read source open");
        super.open(parameters);
        deserializeConfiguration();
        conn = ConnectionFactory.createConnection(hbaseConf);
        admin = conn.getAdmin();
        if (!admin.tableExists(TableName.valueOf(tableName))) {
            throw new IOException("table does not exist.");
        }
        table = conn.getTable(TableName.valueOf(tableName));
        scan = new Scan();
    }

    private void deserializeConfiguration() {
        LOG.info("Deserialize HBase configuration.");
        hbaseConf = HBaseConfigurationUtil.deserializeConfiguration(
            serializedConfig, HBaseConfigurationUtil.getHBaseConfiguration());
        LOG.info("Deserialization successfully.");
    }

    @Override
    public void run(SourceContext<Row> sourceContext) throws Exception {
        LOG.info("Read source run");
        try (ResultScanner scanner = table.getScanner(scan)) {
            Iterator<Result> iterator = scanner.iterator();
            while (iterator.hasNext()) {
                Result result = iterator.next();
                String rowKey = Bytes.toString(result.getRow());
                byte[] value = result.getValue(Bytes.toBytes("f1"), Bytes.toBytes("q1"));
                Row row = new Row(2);
                row.setField(0, rowKey);
                row.setField(1, Bytes.toString(value));
                sourceContext.collect(row);
                LOG.info("Send data successfully.");
            }
        }

        LOG.info("Read successfully.");
    }

    @Override
    public void close() throws Exception {
        closeHBase();
    }
```



```
    private void closeHBase() {
        LOG.info("Close HBase Connection.");
        try {
            if (admin != null) {
                admin.close();
                admin = null;
            }
            if (table != null) {
                table.close();
                table = null;
            }
            if (conn != null) {
                conn.close();
                conn = null;
            }
        } catch (IOException e) {
            LOG.error("Close HBase Exception:", e);
            throw new RuntimeException(e);
        }
        LOG.info("Close successfully.");
    }

    @Override
    public void cancel() {
        closeHBase();
    }
}
```

11.3.9 Flink Reading Data from and Writing Data to Hudi

11.3.9.1 Scenario Description

This section applies to MRS 3.3.0 or later.

Typical Scenario Description

In this example, the job generates one data record per second, writes the data to the Hudi table, and reads and prints the data in the Hudi table.

Development Guideline

1. Write data to Hudi:
   a. Generate data through a random data generation class.
   b. Convert the generated data into **DataStream<RowData>**.
   c. Write data to the Hudi table.
2. Read data from Hudi:
   a. Read data from the Hudi table.
   b. Combine the read data into the JSON format and print the data.

11.3.9.2 Java Sample Code

Description

Call Flink APIs to read data from and write data to Hudi.

Sample Code

The following example shows the main logic code of WriteIntoHudi and ReadFromHudi.

For details about the complete code, see **com.huawei.bigdata.flink.examples.WriteIntoHudi** and **com.huawei.bigdata.flink.examples.ReadFromHudi**.

- Main logic code of WriteIntoHudi

```
public class WriteIntoHudi {
    public static void main(String[] args) throws Exception {
        System.out.println("use command as: ");
        System.out.println(
                "./bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.WriteIntoHudi"
                        + " /opt/test.jar --hudiTableName hudiSinkTable --hudiPath hdfs://hacluster/tmp/
flinkHudi/hudiTable");
        System.out.println(
                "******************************************************************************************");
        System.out.println("<hudiTableName> is the hudi table name. (Default value is hudiSinkTable)");
        System.out.println("<hudiPath> Base path for the target hoodie table. (Default value is hdfs://
hacluster/tmp/flinkHudi/hudiTable)");
        System.out.println(
                "******************************************************************************************");

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        env.setParallelism(1);
        env.getCheckpointConfig().setCheckpointInterval(10000);
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        DataStream<RowData> stringDataStreamSource = env.addSource(new SimpleStringGenerator())
                .map(new MapFunction<Tuple5<String, String, Integer, String, String>, RowData>() {
                    @Override
                    public RowData map(Tuple5<String, String, Integer, String, String> tuple5) throws
Exception {
                        GenericRowData rowData = new GenericRowData(5);
                        rowData.setField(0, StringData.fromString(tuple5.f0));
                        rowData.setField(1, StringData.fromString(tuple5.f1));
                        rowData.setField(2, tuple5.f2);
                        rowData.setField(3, TimestampData.fromTimestamp(Timestamp.valueOf(tuple5.f3)));
                        rowData.setField(4, StringData.fromString(tuple5.f4));
                        return rowData;
                    }
                });
        String basePath = paraTool.get("hudiPath", "hdfs://hacluster/tmp/flinkHudi/hudiTable");
        String targetTable = paraTool.get("hudiTableName", "hudiSinkTable");
        Map<String, String> options = new HashMap<>();
        options.put(FlinkOptions.PATH.key(), basePath);
        options.put(FlinkOptions.TABLE_TYPE.key(), HoodieTableType.MERGE_ON_READ.name());
        options.put(FlinkOptions.PRECOMBINE_FIELD.key(), "ts");
        options.put(FlinkOptions.INDEX_BOOTSTRAP_ENABLED.key(), "true");
        HoodiePipeline.Builder builder = HoodiePipeline.builder(targetTable)
                .column("uuid VARCHAR(20)")
                .column("name VARCHAR(10)")
                .column("age INT")
                .column("ts TIMESTAMP(3)")
                .column("p VARCHAR(20)")
                .pk("uuid")
                .partition("p")
                .options(options);
        builder.sink(stringDataStreamSource, false); // The second parameter indicating whether the
input data stream is bounded
        env.execute("Hudi_Sink");
    }
    public static class SimpleStringGenerator implements SourceFunction<Tuple5<String, String,
Integer, String, String>> {
        private static final long serialVersionUID = 2174904787118597072L;
        boolean running = true;
        Integer i = 0;
```

```
    @Override
    public void run(SourceContext<Tuple5<String, String, Integer, String, String>> ctx) throws
Exception {
        while (running) {
            i++;
            String uuid = "uuid" + i;
            String name = "name" + i;
            Integer age = new Integer(i);
            String ts = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss"));
            String p = "par" + i % 5;
            Tuple5<String, String, Integer, String, String> tuple5 = Tuple5.of(uuid, name, age, ts, p);
            ctx.collect(tuple5);
            Thread.sleep(1000);
        }
    }
    @Override
    public void cancel() {
        running = false;
    }
  }
}
```

- Main logic code of ReadFromHudi

```
public class ReadFromHudi {
  public static void main(String[] args) throws Exception {
    System.out.println("use command as: ");
    System.out.println(
        "./bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.ReadFromHudi"
            + " /opt/test.jar --hudiTableName hudiSourceTable --hudiPath hdfs://hacluster/tmp/
flinkHudi/hudiTable"
            + " --read.start-commit 20221206111532"
    );
    System.out.println(
        "********************************************************************************");
    System.out.println("<hudiTableName> is the hoodie table name. (Default value is
hudiSourceTable)");
    System.out.println("<hudiPath> Base path for the target hoodie table. (Default value is hdfs://
hacluster/tmp/flinkHudi/hudiTable)");
    System.out.println("<read.start-commit> Start commit instant for reading, the commit time
format should be 'yyyyMMddHHmmss'. (Default value is earliest)");
    System.out.println(
        "********************************************************************************");

    ParameterTool paraTool = ParameterTool.fromArgs(args);
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    String basePath = paraTool.get("hudiPath", "hdfs://hacluster/tmp/flinkHudi/hudiTable");
    String targetTable = paraTool.get("hudiTableName", "hudiSourceTable");
    String startCommit = paraTool.get(FlinkOptions.READ_START_COMMIT.key(),
FlinkOptions.START_COMMIT_EARLIEST);
    Map<String, String> options = new HashMap();
    options.put(FlinkOptions.PATH.key(), basePath);
    options.put(FlinkOptions.TABLE_TYPE.key(), HoodieTableType.MERGE_ON_READ.name());
    options.put(FlinkOptions.READ_AS_STREAMING.key(), "true"); // this option enable the
streaming read
    options.put(FlinkOptions.READ_START_COMMIT.key(), startCommit); // specifies the start
commit instant time
    HoodiePipeline.Builder builder = HoodiePipeline.builder(targetTable)
        .column("uuid VARCHAR(20)")
        .column("name VARCHAR(10)")
        .column("age INT")
        .column("ts TIMESTAMP(3)")
        .column("p VARCHAR(20)")
        .pk("uuid")
        .partition("p")
        .options(options);

    DataStream<RowData> rowDataDataStream = builder.source(env);
    rowDataDataStream.map(new MapFunction<RowData, String>() {
```

```
            @Override
            public String map(RowData rowData) throws Exception {
                StringBuilder sb = new StringBuilder();
                sb.append("{");
                sb.append("\"uuid\":\"").append(rowData.getString(0)).append("\",");
                sb.append("\"name\":\"").append(rowData.getString(1)).append("\",");
                sb.append("\"age\":").append(rowData.getInt(2)).append(",");
                sb.append("\"ts\":\"").append(rowData.getTimestamp(3, 0)).append("\",");
                sb.append("\"p\":\"").append(rowData.getString(4)).append("\"");
                sb.append("}");
                return sb.toString();
            }
        }).print();
        env.execute("Hudi_Source");
    }
}
```

11.3.10 Python Development Examples

11.3.10.1 Submitting a Regular Job Using Python

11.3.10.1.1 Description

This scenario applies to MRS 3.3.0 or later.

Assume that you need to submit a Flink task to an MRS cluster. The main language used by the service platform is Python. The following content uses an example Python program to read and writ Kafka jobs.

11.3.10.1.2 Python Sample Code

Function

Submit Flink Kafka read and write jobs to Yarn through Python APIs.

Sample Code

The main logic code in **pyflink-kafka.py** is provided. Before submitting the code, ensure that **file_path** is the path where the SQL statement to be executed. You are advised to use a full path.

For details about the complete code, see **pyflink-kafka.py** in **flink-examples/pyflink-example/pyflink-kafka**.

```
import os
import logging
import sys
from pyflink.common import JsonRowDeserializationSchema, JsonRowSerializationSchema
from pyflink.common.typeinfo import Types
from pyflink.datastream.connectors import FlinkKafkaProducer, FlinkKafkaConsumer
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import TableEnvironment, EnvironmentSettings
def read_sql(file_path):
    if not os.path.isfile(file_path):
        raise TypeError(file_path + " does not exist")
    all_the_text = open(file_path).read()
    return all_the_text
def exec_sql():
    # Change the SQL path before job submission.
    # file_path = "/opt/client/Flink/flink/insertData2kafka.sql"
    # file_path = os.getcwd() + "/../../../../yarnship/insertData2kafka.sql"
```

```
    # file_path = "/opt/client/Flink/flink/conf/ssl/insertData2kafka.sql"
    file_path = "insertData2kafka.sql"
    sql = read_sql(file_path)
    t_env = TableEnvironment.create(EnvironmentSettings.in_streaming_mode())
    statement_set = t_env.create_statement_set()
    sqlArr = sql.split(";")
    for sqlStr in sqlArr:
        sqlStr = sqlStr.strip()
        if sqlStr.lower().startswith("create"):
            print("---------create----------")
            print(sqlStr)
            t_env.execute_sql(sqlStr)
        if sqlStr.lower().startswith("insert"):
            print("---------insert----------")
            print(sqlStr)
            statement_set.add_insert_sql(sqlStr)
    statement_set.execute()
def read_write_kafka():
    # find kafka connector jars
    env = StreamExecutionEnvironment.get_execution_environment()
    env.set_parallelism(1)
    specific_jars = "file:///opt/client/Flink/flink/lib/flink-connector-kafka-xxx.jar"
    # specific_jars = "file://" + os.getcwd() + "/../../../../yarnship/flink-connector-kafka-xxx.jar"
    # specific_jars = "file:///opt/client/Flink/flink/conf/ssl/flink-connector-kafka-xxx.jar"
    # the sql connector for kafka is used here as it's a fat jar and could avoid dependency issues
    env.add_jars(specific_jars)
    kafka_properties = {'bootstrap.servers': '192.168.20.162:21005', 'group.id': 'test_group'}
    deserialization_schema = JsonRowDeserializationSchema.builder() \
        .type_info(type_info=Types.ROW([Types.INT(), Types.STRING()])).build()
    kafka_consumer = FlinkKafkaConsumer(
        topics='test_source_topic',
        deserialization_schema=deserialization_schema,
        properties=kafka_properties)
    print("---------read ----------")
    ds = env.add_source(kafka_consumer)
    serialization_schema = JsonRowSerializationSchema.builder().with_type_info(
        type_info=Types.ROW([Types.INT(), Types.STRING()])).build()
    kafka_producer = FlinkKafkaProducer(
        topic='test_sink_topic',
        serialization_schema=serialization_schema,
        producer_config=kafka_properties)
    print("--------write----------")
    ds.add_sink(kafka_producer)
    env.execute("pyflink kafka test")
if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    print("----------insert data to kafka----------")
    exec_sql()
    print("----------read_write_kafka----------")
    read_write_kafka()
```

Table 11-7 Parameters for submitting a regular job using Python

Parameter	Description	Example
bootstrap.servers	Service IP address and port number of the Broker instance of Kafka	192.168.12.25:21005

Parameter	Description	Example
specific_jars	Package path: *Client installation directory*/**Flink/flink/lib/flink-connector-kafka-*.jar**. You are advised to use a full path. **NOTE** If a job needs to be submitted as yarn-application, replace the following path. Replace the JAR package version with the actual one. specific_jars="file://"+os.getcwd()+"/../../../../yarnship/flink-connector-kafka-1.15.0-h0.cbu.mrs.330.r13.jar"	specific_jars = file:///*Client installation directory*/Flink/flink/lib/flink-connector-kafka-1.15.0-h0.cbu.mrs.330.r13.jar
file_path	Path of the **insertData2kafka.sql** file. You are advised to use a full path. Obtain the package from the secondary development sample code and upload it to the specified directory on the client. **NOTE** If a job needs to be submitted as yarn-application, replace the following path: file_path = os.getcwd() + "/../../../../yarnship/insertData2kafka.sql"	file_path = /*Client installation directory*/Flink/flink/insertData2kafka.sql

The following is an example SQL statement:

```
create table kafka_sink_table (
  age int,
  name varchar(10)
) with (
  'connector' = 'kafka',
'topic' = 'test_source_topic', --Name of the topic written to Kafka. Ensure that the topic name is the same as that in the Python file.
  'properties.bootstrap.servers' = 'IP address of the Kafka broker instance:Kafka port number',
  'properties.group.id' = 'test_group',
  'format' = 'json'
);
create TABLE datagen_source_table (
   age int,
  name varchar(10)
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
);
INSERT INTO
  kafka_sink_table
SELECT
  *
FROM
  datagen_source_table;
```

11.3.10.1.3 Running the Program

Step 1 Obtain **pyflink-kafka.py** and **insertData2kafka.sql** from the sample project **flink-examples/pyflink-example/pyflink-kafka**.

Step 2 Package the prepared Python virtual environment by referring to **Preparing for Development and Operating Environment** and obtain the **venv.zip** file.

```
zip -q -r venv.zip venv/
```

Step 3 Log in to the active management node as the **root** user and upload **venv.zip**, **pyflink-kafka.py**, and **insertData2kafka.sql** files obtained in **Step 1**and **Step 2** to the client environment.

- Per-job: Upload the preceding files to *Client installation directory*/**Flink/flink**.
- yarn-application: Upload the preceding files and the **flink-connector-kafka-***Actual version number*.**jar** package to *Client installation directory*/**Flink/flink/yarnship**.

Step 4 Change the **specific_jars** path in **pyflink-kafka.py**.

- per-job: Change the path to the actual path of the SQL file, for example, **file:///***Client installation directory*/**Flink/flink/lib/flink-connector-kafka-***Actual version number*.**jar**.
- yarn-application: Change to **file://"+os.getcwd()+"/../../../../yarnship/flink-connector-kafka-***Actual version number*.**jar**.

Step 5 Change **file_path** in **pyflink-kafka.py**.

- per-job: Change the path to the actual path of the SQL file. For example: *Client installation directory*/**Flink/flink/insertData2kafka.sql**
- yarn-application: Change the path to **os.getcwd () + "/../../../../yarnship/insertData2kafka.sql"**

Step 6 Run the following command to specify the running environment:

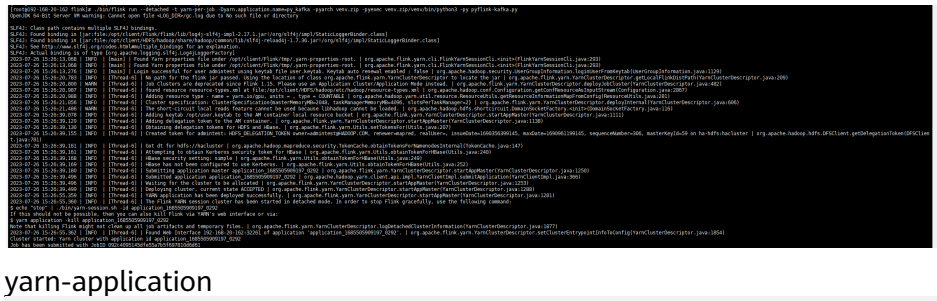
```
export PYFLINK_CLIENT_EXECUTABLE=venv.zip/venv/bin/python3
```

Step 7 Run the following command to run the program:

- Per-job:

  ```
  ./bin/flink run --detached -t yarn-per-job -Dyarn.application.name=py_kafka -pyarch venv.zip -pyexec venv.zip/venv/bin/python3 -py pyflink-kafka.py
  ```

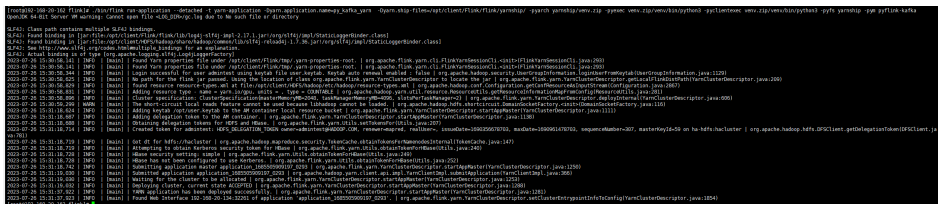
  Execution result:

- yarn-application

  ```
  ./bin/flink run-application --detached -t yarn-application -Dyarn.application.name=py_kafka -Dyarn.ship-files=/opt/client/Flink/flink/yarnship/ -pyarch yarnship/venv.zip -pyexec venv.zip/venv/bin/python3 -pyclientexec venv.zip/venv/bin/python3 -pyfs yarnship -pym pyflink-kafka
  ```

  Execution result:

----End

11.3.10.2 Submitting a SQL Job Using Python

11.3.10.2.1 Description

Assume that you need to submit Flink tasks to the MRS cluster, the main language used by the service platform is Python, and core service processing requires SQL. The following content provides an example to describe how to submit a SQL job using Python.

11.3.10.2.2 Python Sample Code

Function

Submit a Flink SQL job to Yarn through Python APIs.

Sample Code

The main logic code in **pyflink-sql.py** is provided. Before submitting the code, ensure that **file_path** is the path where the SQL statement to be executed. You are advised to use a full path.

For details about the complete code, see **pyflink-sql.py** in **flink-examples/pyflink-example/pyflink-sql**.

```
import logging
import sys
import os
from pyflink.table import (EnvironmentSettings, TableEnvironment)
def read_sql(file_path):
    if not os.path.isfile(file_path):
        raise TypeError(file_path + " does not exist")
    all_the_text = open(file_path).read()
    return all_the_text
def exec_sql():
    # Change the SQL path before job submission.
    file_path = "datagen2kafka.sql"
    sql = read_sql(file_path)
    t_env = TableEnvironment.create(EnvironmentSettings.in_streaming_mode())
    statement_set = t_env.create_statement_set()
    sqlArr = sql.split(";")
    for sqlStr in sqlArr:
        sqlStr = sqlStr.strip()
        if sqlStr.lower().startswith("create"):
            print("---------create----------")
            print(sqlStr)
            t_env.execute_sql(sqlStr)
        if sqlStr.lower().startswith("insert"):
            print("---------insert----------")
            print(sqlStr)
            statement_set.add_insert_sql(sqlStr)
    statement_set.execute()
if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    exec_sql()
```


Table 11-8 Parameters for submitting a SQL job using Python

Parameter	Description	Example
file_path	Path of the **datagen2kafka.sql** file. You are advised to use a full path. Obtain the package from the secondary development sample code and upload it to the specified directory on the client. **NOTE** If a job needs to be submitted as yarn-application, replace the following path: file_path = os.getcwd() + "/../../../../yarnship/datagen2kafka.sql"	file_path = /*Client installation directory*/Flink/flink/datagen2kafka.sql

The following is an example SQL statement:

```
create table kafka_sink (
  uuid varchar(20),
  name varchar(10),
  age int,
  ts timestamp(3),
  p varchar(20)
) with (
  'connector' = 'kafka',
  'topic' = 'input2',
  'properties.bootstrap.servers' = 'IP address of the Kafka broker instance:Kafka port number',
  'properties.group.id' = 'testGroup2',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);
create TABLE datagen_source (
  uuid varchar(20),
  name varchar(10),
  age int,
  ts timestamp(3),
  p varchar(20)
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
);
INSERT INTO
  kafka_sink
SELECT
  *
FROM
  datagen_source;
```

11.3.10.2.3 Running the Program

Step 1 Obtain **pyflink-sql.py** and **datagen2kafka.sql** from the sample project **flink-examples/pyflink-example/pyflink-sql**.

Step 2 Package the prepared Python virtual environment by referring to **Preparing for Development and Operating Environment** and obtain the **venv.zip** file.

```
zip -q -r venv.zip venv/
```

Step 3 Log in to the active management node as the **root** user and upload **venv.zip**, **pyflink-sql.py**, and **datagen2kafka.sql** files obtained in **Step 1**and **Step 2** to the client environment.

- Per-job: Upload the preceding files to *Client installation directory*/**Flink/flink**.
- yarn-application: Upload the preceding files to *Client installation directory*/**Flink/flink/yarnship**.
- yarn-session: Upload the preceding files to *Client installation directory*/**Flink/flink/conf/ssl**.

Step 4 Change **file_path** in **pyflink-sql.py**.

- per-job: Change the path to the actual path of the SQL file. For example: *Client installation directory*/**Flink/flink/datagen2kafka.sql**
- yarn-application: Change the path to **os.getcwd() + "/../../../../yarnship/datagen2kafka.sql"**
- yarn-session: Change the path to the actual path of the SQL file. For example: *Client installation directory*/**Flink/flink/conf/ssl//datagen2kafka.sql**

Step 5 Run the following command to specify the running environment:

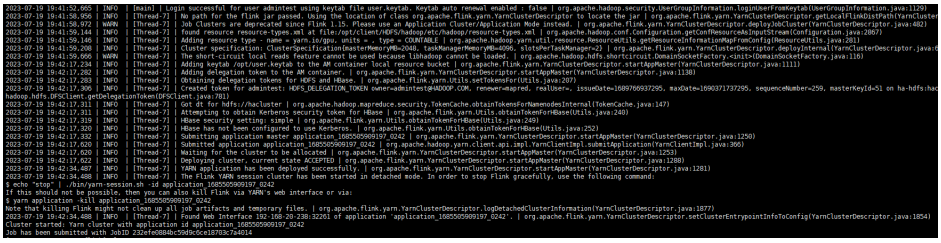
```
export PYFLINK_CLIENT_EXECUTABLE=venv.zip/venv/bin/python3
```

Step 6 Run the following command to run the program:

- Per-job:

  ```
  ./bin/flink run --detached -t yarn-per-job -Dyarn.application.name=py_sql -pyarch venv.zip -pyexec venv.zip/venv/bin/python3 -py pyflink-sql.py
  ```

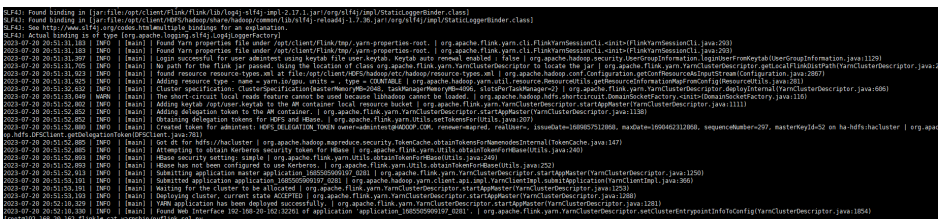
  Execution result:

- yarn-application

  ```
  ./bin/flink run-application --detached -t yarn-application -Dyarn.application.name=py_sql -Dyarn.ship-files=/opt/client/Flink/flink/yarnship/ -pyarch yarnship/venv.zip -pyexec venv.zip/venv/bin/python3 -pyclientexec venv.zip/venv/bin/python3 -pyfs yarnship -pym pyflink-sql
  ```

  Execution result:

- yarn-session

  Before starting the Yarn session, prepare the running environment by referring to **Preparing for Development and Operating Environment**. Run the following command to start yarn-session:

  ```
  bin/yarn-session.sh -jm 1024 -tm 4096 -t conf/ssl/ -d
  ```

  Run the following command to submit the job:

  ```
  ./bin/flink run --detached -t yarn-session -Dyarn.application.name=py_sql -Dyarn.application.id=application_1685505909197_0285 -pyarch conf/ssl/venv.zip -pyexec conf/ssl/venv.zip/venv/bin/python3 -py conf/ssl/pyflink-sql.py
  ```

  Execution result:

----End

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.

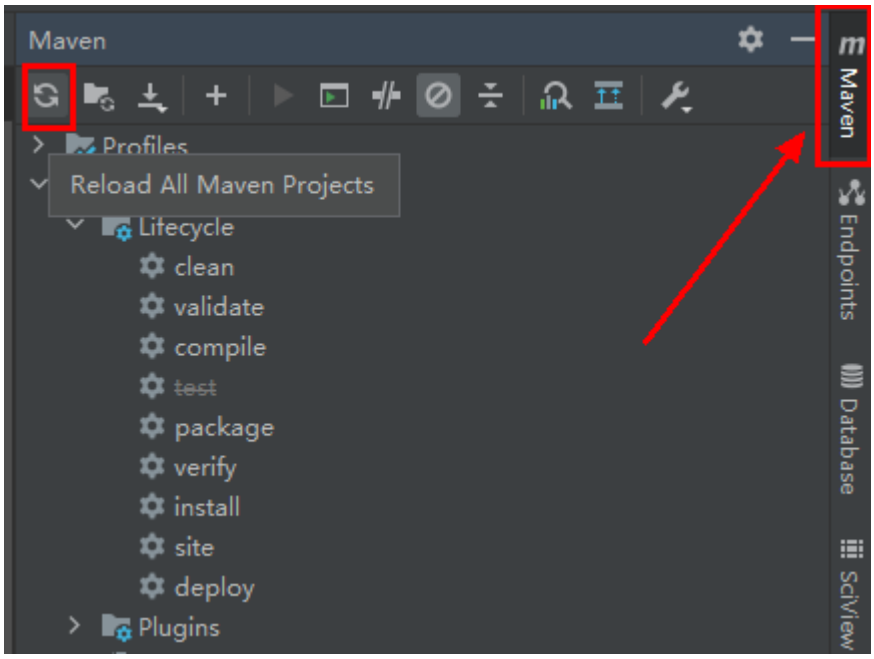
NOTE

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, click **Reload All Maven Projects** in the Maven window on the right of IDEA to import Maven project dependencies.

Figure 11-40 Reload projects

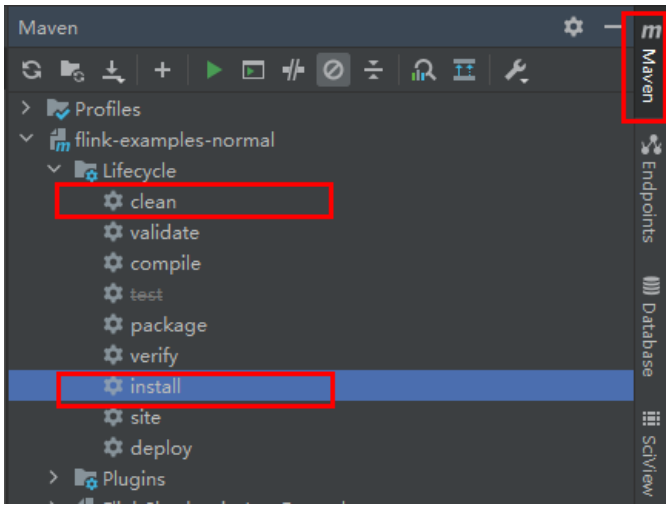


Step 2 Compile and run the application.

Use either of the following two methods:

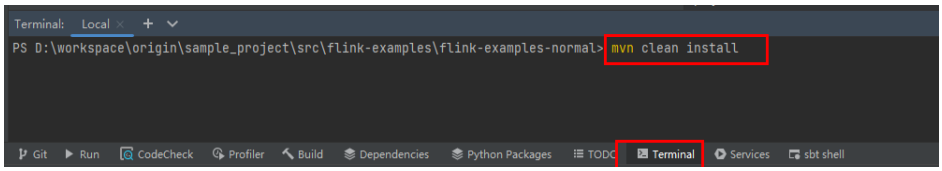
- Method 1:
  a. Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.
  b. Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

  Figure 11-41 Maven clean and install

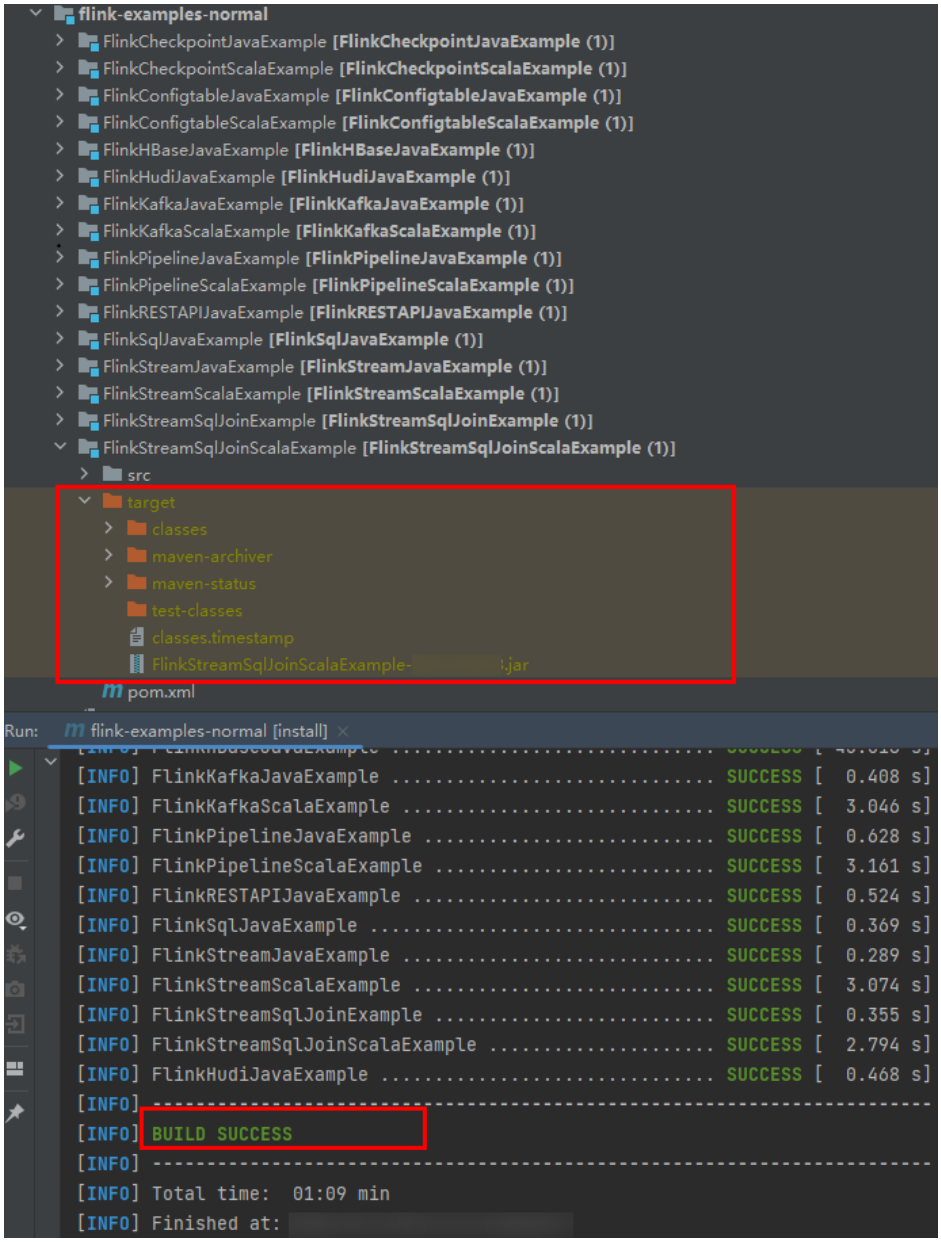
- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

Figure 11-42 Entering **mvn clean install** in the IDEA Terminal text box



Step 3 After the compilation is complete, the message "BUILD SUCCESS" is printed and the **target** directory is generated. Obtain the JAR package in the directory.

Figure 11-43 Compilation completed



Step 4 Copy the **.jar** package (for example **FlinkStreamJavaExample.jar**) created in **Step 3** 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 4096
```

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 about the dependency package for running the sample project**.
- 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**.
- The following applies to MRS 3.2.1 or later. The memory size of TaskManagers specified by using the **-tm** command must be at least 4,096 MB.

- 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 11-9 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] [sasl.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] [sasl.kerberos.service.name] [ssl.truststore.location] [ssl.truststore.password]
  ```

Table 11-10 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	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. – 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 the SSL is configured, **ssl.truststore.location** (path of **truststore**) and **ssl.truststore.password** (password of **truststore**) must be set.	No **NOTE** – 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, System domain name is **HADOOP.COM**:

- 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 *xxx*
- 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 --sasl.kerberos.service.name kafka --ssl.truststore.location /config/truststore.jks --ssl.truststore.password *xxx* --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
  - Java

    i. Start the program to generate data for Kafka. For details about Kafka configuration, see **Run the following code to interconnecting with Kafka**
    ```
    bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/client/FlinkStreamSqlJoinExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.xxx:9092
    ```

    ii. 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.

    iii. 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
    ```

  - Scala (for MRS 3.3.0 or later)

    i. Start the application to produce data in Kafka. For details about how to configure Kafka, see **Run the following code to interconnecting with Kafka**.
    ```
    bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/client/FlinkStreamSqlJoinScalaExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.xxx:9092
    ```

    ii. Run the **netcat** command on any node in the cluster to wait for an application connection.
    ```
    netcat -l -p 9000
    ```

    iii. Start the application to receive socket data and perform a joint query.
    ```
    bin/flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket /opt/client/FlinkStreamSqlJoinScalaExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.xxx:9092 --hostname xxx.xxx.xxx.xxx --port 9000
    ```

- Running the Flink HBase sample program(MRS 3.2.0 or later clusters.)
  - yarn-session

    i. Start the flink cluster.
    ```
    ./bin/yarn-session.sh -t config -jm 1024 -tm 1024
    ```

    ii. Run the Flink program and set parameters.
    ```
    bin/flink run --class com.huawei.bigdata.flink.examples.WriteHBase /opt/client1/Flink/flink/FlinkHBaseJavaExample-xxx.jar  --tableName xxx --confDir xxx

    bin/flink run --class com.huawei.bigdata.flink.examples.ReadHBase /opt/client1/Flink/flink/FlinkHBaseJavaExample-xxx.jar  --tableName xxx --confDir xxx
    ```

  - yarn-cluster

    ```
    bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.WriteHBase /opt/client1/Flink/flink/FlinkHBaseJavaExample-xxx.jar  --tableName xxx --confDir xxx

    bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.ReadHBase /opt/client1/Flink/flink/FlinkHBaseJavaExample-xxx.jar  --tableName xxx --confDir xxx
    ```

- Running the Flink Hudi sample application (MRS 3.2.1 or later)
  - yarn-session mode
    i. Start the Flink cluster.

    ```
    ./bin/yarn-session.sh -jm 1024 -tm 4096
    ```

    ii. Run the Flink application and enter parameters.

    ```
    ./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoHudi /opt/test.jar --hudiTableName hudiSinkTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable

    ./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromHudi /opt/test.jar --hudiTableName hudiSourceTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable --read.start-commit xxx
    ```

  - yarn-cluster mode

    ```
    ./bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.WriteIntoHudi /opt/test.jar --hudiTableName hudiSinkTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable

    ./bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.ReadFromHudi /opt/test.jar --hudiTableName hudiSourceTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable --read.start-commit xxx
    ```

- Run the REST APIs to create a tenant sample application. The TestCreateTenants application is used as an example.
  - yarn-session mode
    i. Start the Flink cluster.

    ```
    ./bin/yarn-session.sh -t config -jm 1024 -tm 4096
    ```

    ii. Run the Flink application and enter parameters.

    ```
    ./bin/flink run --class com.huawei.bigdata.flink.examples.TestCreateTenants /opt/client/FlinkRESTAPIJavaExample-xxx.jar --hostName xx-xx-xx-xx
    ```

  - yarn-cluster mode

    ```
    ./bin/flink run -m yarn-cluster -yt config -yjm 1024 -ytm 4096 --class com.huawei.bigdata.flink.examples.TestCreateTenants /opt/client/FlinkRESTAPIJavaExample-xxx.jar --hostName xx-xx-xx-xx
    ```

- Run a SQL task to submit Flink JAR jobs (applicable to MRS 3.2.1 or later).
  - yarn-session mode
    i. Start the Flink cluster.

    ```
    ./bin/yarn-session.sh -t config -jm 1024 -tm 4096
    ```

    ii. Run the Flink application and enter parameters.

    ```
    bin/flink run -d --class com.huawei.mrs.FlinkSQLExecutor /opt/flink-sql-xxx.jar --sql ./sql/datagen2kafka.sql
    ```

  - yarn-cluster mode

    ```
    bin/flink run -m yarn-cluster -yt config -yjm 1024 -ytm 4096 -d --class com.huawei.mrs.FlinkSQLExecutor /opt/flink-sql-xxx.jar --sql ./sql/datagen2kafka.sql
    ```

----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 Task Manager 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 11-44**. Choose **Task Managers** > **Stdout** to view the running result, as shown in **Figure 11-45**.

    Figure 11-44 Submitted jobs

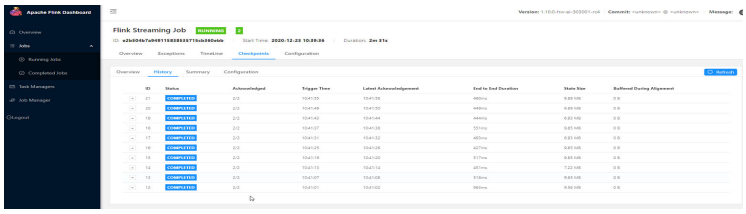
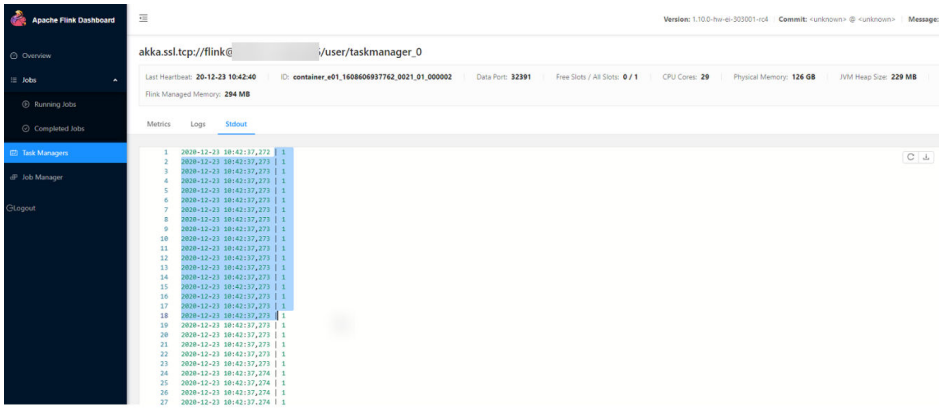
    

    Figure 11-45 Execution result

  - 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 11-46**. Choose **Task Managers**. You can see two tasks, as shown in **Figure 11-47**. Click any task, choose **Stdout** to view the output of the task, as shown in **Figure 11-48** and **Figure 11-49**.

    Figure 11-46 Running jobs

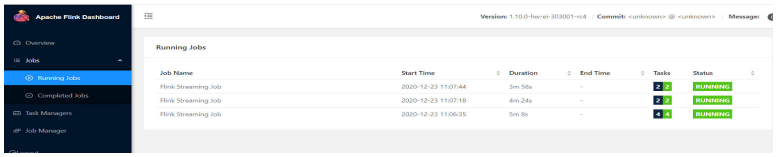
    

Figure 11-47 Submitted tasks

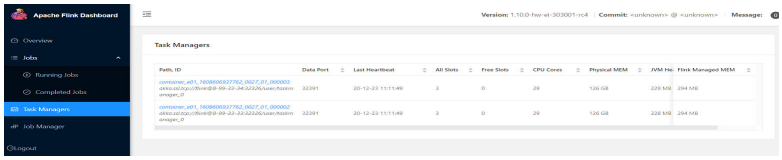


Figure 11-48 Output of task1

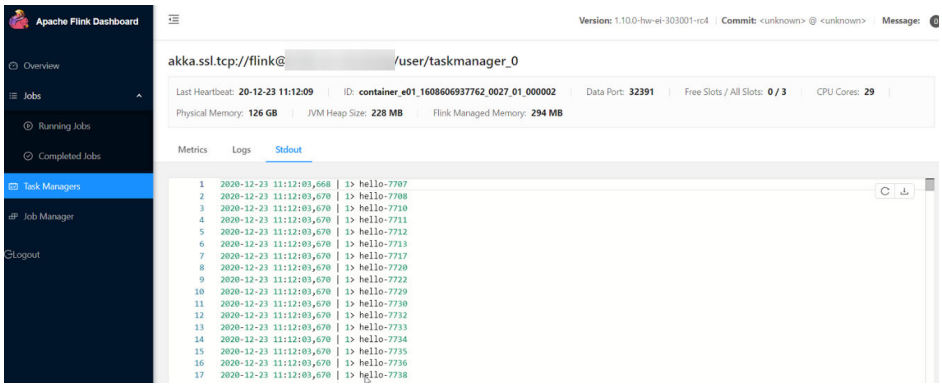
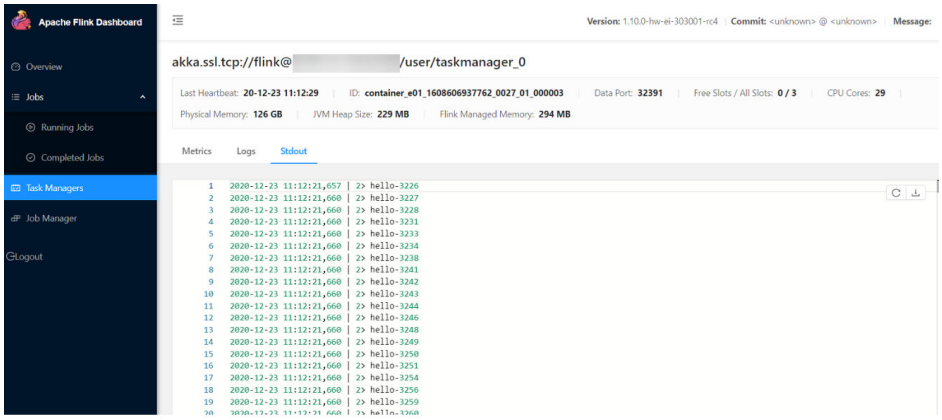


Figure 11-49 Output of task2



- **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 11-50**. Choose **Task Managers**, you can see submitted task, as shown in **Figure 11-51**. Choose **Stdout** to view the running result, as shown in **Figure 11-52**.

Figure 11-50 Completed job

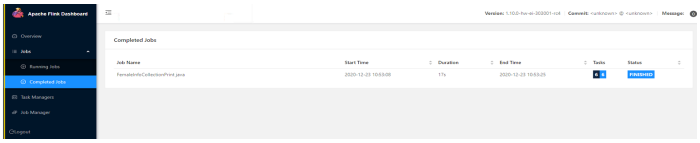


Figure 11-51 Submitted task

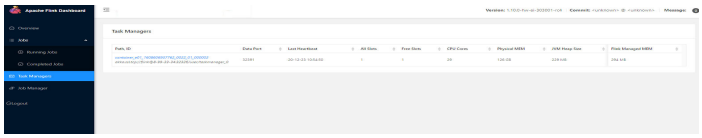
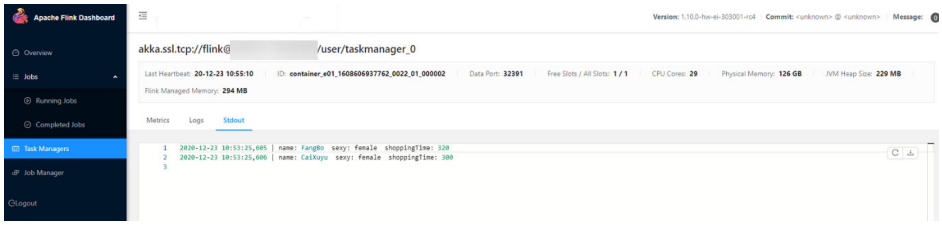


Figure 11-52 Execution result



- **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 11-53**. Choose **Task Managers**, you can see submitted task, as shown in **Figure 11-54**. Choose **Stdout** to view the running result, as shown in **Figure 11-55**.

Figure 11-53 Running jobs

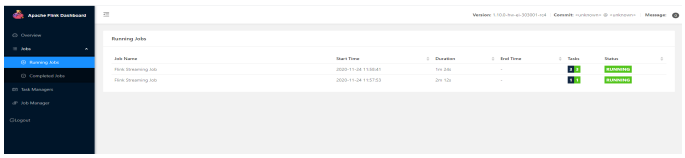


Figure 11-54 Submitted task

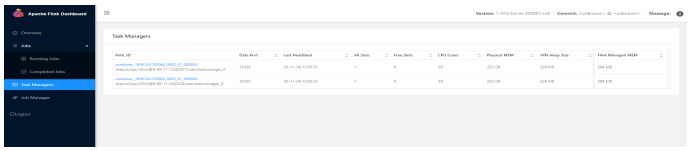
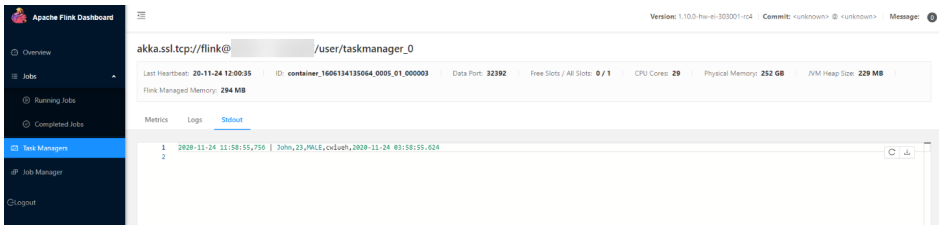


Figure 11-55 Execution result



- **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 11-56**. Choose **Task Managers**, you can see submitted task, as shown in **Figure 11-57**. Choose **Stdout** to view the running result, as shown in **Figure 11-58**.

Figure 11-56 Runnning jobs

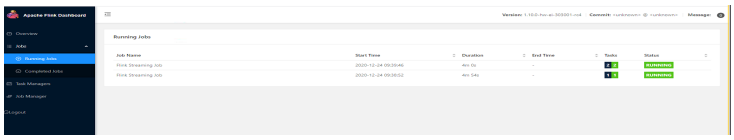


Figure 11-57 Submitted task

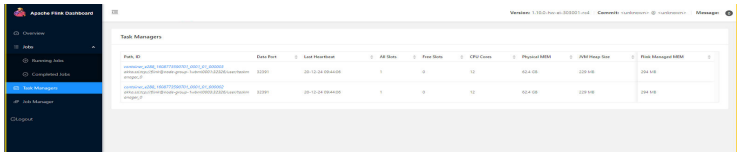
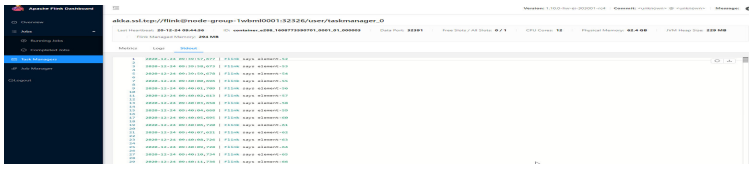


Figure 11-58 Execution result



- **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

```
[redacted]:/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 [redacted]:26009
Container: container_1547547065745_0001_01_000002 on [redacted]:26009
Container: container_1547547065745_0001_01_000003 on [redacted]:26009
Container: container_1547547065745_0001_01_000004 on [redacted]: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/

```
[redacted]:/opt/flinkclient/Flink/flink/logdir/[redacted]_26009 # ll
total 172
-rw-r--r-- 1 root root 170605 Jan 17 10:24 container_1547547065745_0001_01_000004
[redacted]:/opt/flinkclient/Flink/flink/logdir/[redacted]_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

```
[redacted]:/opt/flinkclient/Flink/flink/logdir/[redacted]_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 [redacted]: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.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                    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

11.4.3 Running a Spring Boot Sample Project and Viewing Results

This section applies to MRS 3.3.0 or later.

Running the Spring Boot Sample in CLI

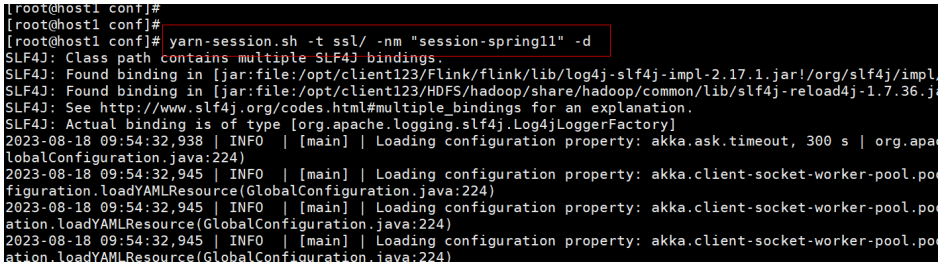
Step 1 Use Maven to run **install** on the IDEA.

If "BUILD SUCCESS" is displayed, the compilation is successful. A JAR file containing the **flink-dws-sink-example-1.0.0-SNAPSHOT** field is generated in the **target** directory of the sample project.

Step 2 On Linux, go to the client installation directory, for example, **/opt/client/Flink/flink/conf**, and save the JAR packages whose names contain **flink-dws-sink-example-1.0.0-SNAPSHOT** in the **target** directory generated in to this directory.

Step 3 Run the following command to create a Yarn session:

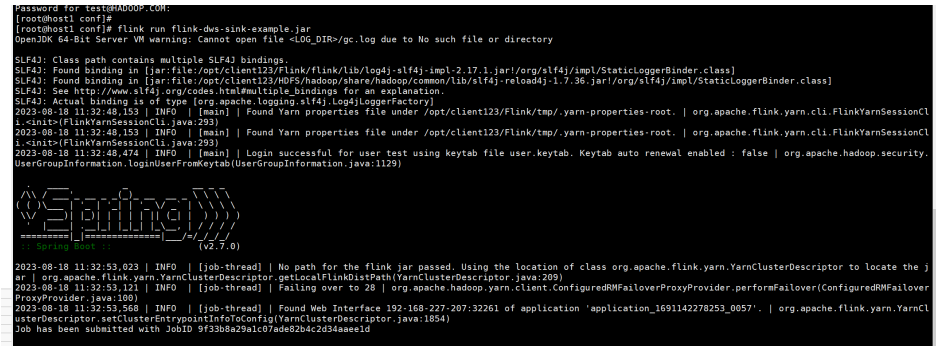
yarn-session.sh -t ssl/ -nm "*session-spring11***" -d**



Step 4 Run the following command to start the SpringBoot service:

- Run the GaussDB(DWS) sample

  flink run flink-dws-sink-example.jar



----End

11.5 More Information

11.5.1 Introduction to Common APIs

11.5.1.1 Java

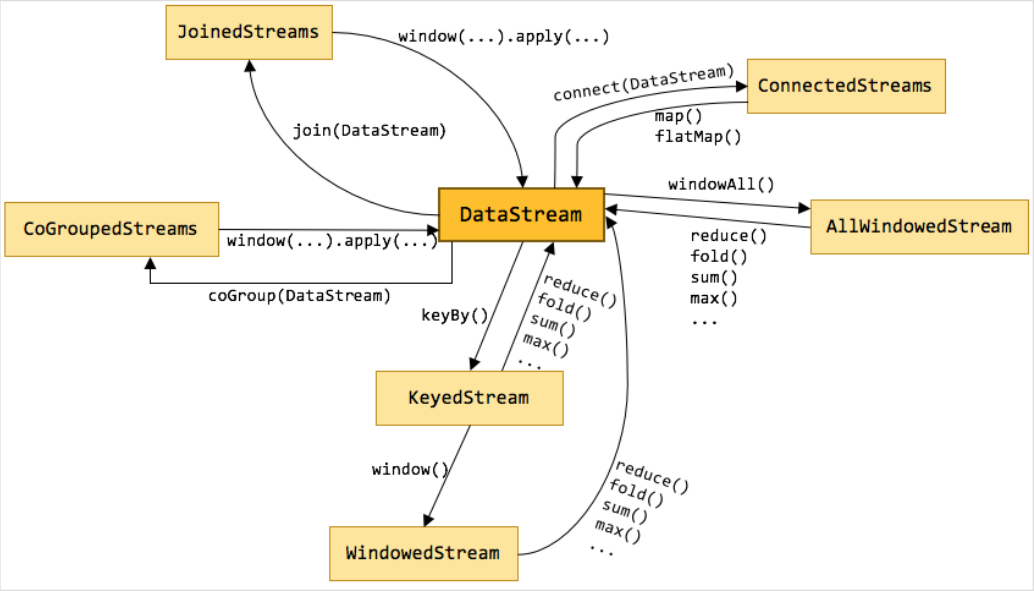
To avoid API compatibility or reliability issues after updates to the open-source Flink, recommended that APIs of the matching version are recommended.

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 11-59 Conversion of Flink stream types



Data Stream Source

Table 11-11 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. • **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. • **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. • **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 DataStreamSource<Long> generate Sequence(long from, long to)	Obtain a sequence of user-defined data as the data stream source. • **from** indicates the starting point of numbers. • **to** indicates the end point of numbers.
public DataStreamSource<String> readTextFile(String filePath)	Obtain the user-defined text file from a specific path as the data stream source. • **filePath** indicates the path of the text file. • **charsetName** indicates the encoding format.
public DataStreamSource<String> readTextFile(String filePath, String charsetName)	
public <OUT> DataStreamSource<OUT> readFile(FileInputformat<OUT> inputformat, String filePath)	Obtain the user-defined file from a specific path as the data stream source. • **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> DataStreamSource<OUT> readFile(FileInputformat<OUT> inputformat, String filePath, FileProcessingMode watchType, long interval)	
public <OUT> DataStreamSource<OUT> readFile(FileInputformat<OUT> inputformat, String filePath, FileProcessingMode watchType, long interval, TypeInformation<OUT> typeInformation)	
public DataStreamSource<String> socketTextStream(String hostname, int port, String delimiter, long maxRetry)	Obtain user-defined socket data as the data stream source. • **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 DataStreamSource<String> socketTextStream(String hostname, int port, String delimiter)	
public DataStreamSource<String> socketTextStream(String hostname, int port)	

API	Description
public <OUT> DataStreamSource<OUT> addSource(SourceFunction<OUT> function)	Customize the **SourceFunction** and a**ddSource** methods to add data sources such as Kafka. The implementation method is the run method of SourceFunction. • **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 11-12 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. • **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)	

API	Description
public DataStreamSink<T> writeAsCsv(String path)	Writhe data to a specific **.csv** file. • **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.
public DataStreamSink<T> writeAsCsv(String path, WriteMode writeMode)	
public <X extends Tuple> DataStreamSink<T> writeAsCsv(String path, WriteMode writeMode, String rowDelimiter, String fieldDelimiter)	
public DataStreamSink<T> writeToSocket(String hostName, int port, SerializationSchema<T> schema)	Write data to the socket connection. • **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 11-13 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 11-14 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. Return the KeyedStream after the KeyBy operation, and call the KeyedStream function, such as reduce, fold, min, minby, max, maxby, sum, and sumby. • **fields** indicates the numbers of columns or names of member variables. • **key** indicates the user-defined basis for partitioning.
public KeyedStream<T, Tuple> keyBy(String... fields)	
public <K> KeyedStream<T, K> keyBy(KeySelector<T, K> key)	
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. **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)	

API	Description
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. • **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. • **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 11-15 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. • **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 11-16 APIs about projecting

API	Description
public <R extends Tuple> SingleOutputStreamOperator<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 11-17 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 11-18 lists differences of AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks APIs.

Table 11-18 Difference of AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks APIs

Parameter	Description
AssignerWithPeriodic-Watermarks	Generate Watermark based on the getConfig().setAutoWatermarkInterval(200L) timestamp of StreamExecutionEnvironment class.
AssignerWithPunc-tuatedWatermarks	Generate a Watermark each time an element is received. Watermarks can be different based on received elements.

Iteration

Table 11-19 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. **NOTE** • This API is helpful to algorithms that require constant update of models. • long maxWaitTimeMillis: The timeout period of each round of iteration.
public IterativeStream<T> iterate(long maxWaitTimeMillis)	

Stream Splitting

Table 11-20 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 11-21 lists APIs for generating windows.

Table 11-21 APIs for generating windows

API	Description
public <W extends Window> WindowedStream<T, KEY, W> window(WindowAssigner<? super T, W> assigner)	Define windows in partitioned KeyedStreams. A window groups each key according to some characteristics (for example, data received within the latest 5 seconds.
public <W extends Window> AllWindowedStream<T, W> windowAll(WindowAssigner<? super T, W> assigner)	Define windows in DataStreams.
public WindowedStream<T, KEY, TimeWindow> timeWindow(Time size)	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. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window. **NOTE** • 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, TimeWindow> timeWindow(Time size, Time slide)	

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. • **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. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window. **NOTE** • 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. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window.
public AllWindowedStream<T, GlobalWindow> countWindowAll(Time size, Time slide)	

Table 11-22 lists APIs for operating windows.

Table 11-22 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. • **function** indicates the window function to be executed. • **resultType** indicates the type of returned data.

Method	API	Description
	public <R> SingleOutputStreamOperator<R> apply(WindowFunction<T, R, K, W> function, TypeInformation<R> resultType)	
	public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> function)	Apply a reduce function to the window and return the result. • **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)	
	public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, WindowFunction<T, R, K, W> function, TypeInformation<R> resultType)	
	public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function)	Apply a fold function to the window and return the result. • **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.
	public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function, TypeInformation<R> resultType)	
	public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, WindowFunction<ACC, R, K, W> function)	

Method	API	Description
	public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, WindowFunction<ACC, R, K, W> function, TypeInformation<ACC> foldAccumulatorType, TypeInformation<R> resultType)	
Window All	public <R> SingleOutputStreamOperator<R> apply(AllWindowFunction<T, R, W> function)	Apply a general function to a window. The data in the window is calculated as a whole.
	public <R> SingleOutputStreamOperator<R> apply(AllWindowFunction<T, R, W> function, TypeInformation<R> resultType)	
	public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> function)	Apply a reduce function to the window and return the result. • **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.
	public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, AllWindowFunction<T, R, W> function)	
	public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, AllWindowFunction<T, R, W> function, TypeInformation<R> resultType)	

Method	API	Description
	public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function)	Apply a fold function to the window and return the result. • **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.
	public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function, TypeInformation<R> resultType)	
	public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, AllWindowFunction<ACC, R, W> function)	
	public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, AllWindowFunction<ACC, R, W> function, TypeInformation<ACC> foldAccumulatorType, TypeInformation<R> resultType)	
Window and Window All	public SingleOutputStreamOperator<T> sum(int positionToSum)	Sum a specified column of the window data. **field** and **positionToSum** indicate a specific column of the data.
	public SingleOutputStreamOperator<T> sum(String field)	
	public SingleOutputStreamOperator<T> min(int positionToMin)	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.
	public SingleOutputStreamOperator<T> min(String field)	

Method	API	Description
	public SingleOutputStreamOperator<T> minBy(int positionToMinBy)	Obtain the row where the minimum value of a column locates in the window data. minBy returns all elements of that row. • **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> 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. • **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)	

Combining Multiple DataStreams

Table 11-23 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.
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<IN1, 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.
public <R> SingleOutputStreamOperator<R> flatMap(CoFlatMapFunction<IN1, IN2, R> coFlatMapper)	Perform mapping operation, which is similar to flatMap operation in DataStream, on elements.

Join Operation

Table 11-24 APIs about join operation

API	Description
public <T2> JoinedStreams<T, T2> join(DataStream<T2> otherStream)	Join two DataStreams using a given key in a specified window.
public <T2> CoGroupedStreams<T, T2> coGroup(DataStream<T2> otherStream)	Co-group two DataStreams using a given key in a specified window.

11.5.1.2 Scala

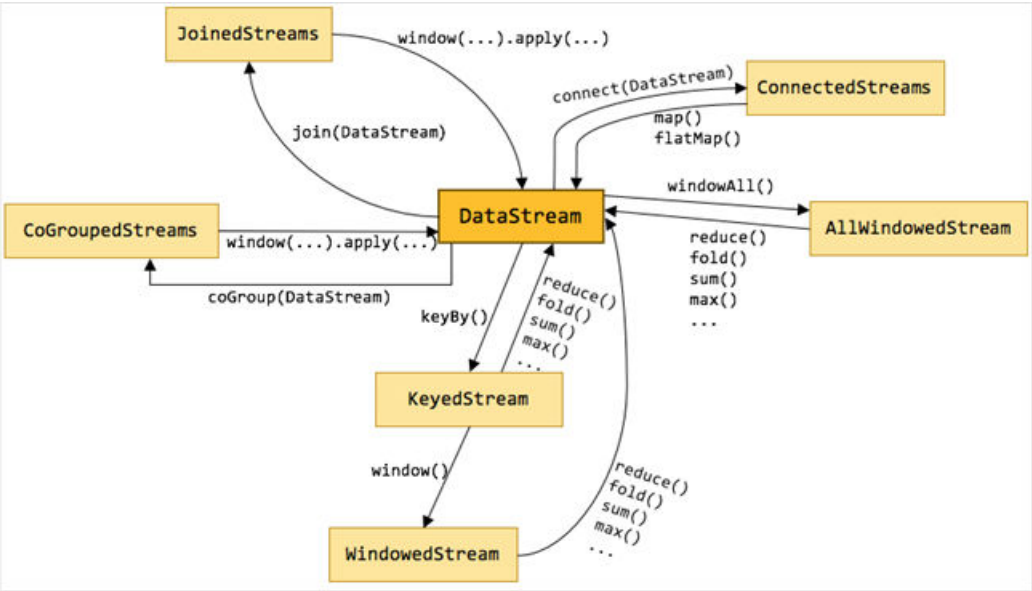
To avoid API compatibility or reliability issues after updates to the open-source Flink, recommended that APIs of the matching version are recommended.

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 11-60 Conversion of Flink stream types



Data Stream Source

Table 11-25 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.
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. • **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. • **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. • **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]	

API	Description
def socketTextStream(hostname: String, port: Int, delimiter: Char = '\n', maxRetry: Long = 0): DataStream[String]	Obtain user-defined socket data as the data stream source. • **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.
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. • **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 11-26 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. • **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]	

API	Description
def writeAsCsv(path: String): DataStreamSink[T]	Writhe data to a specific **.csv** file. • **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. • **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 11-27 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](flatMapper: 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]	

API	Description
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 11-28 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. Return the KeyedStream after the KeyBy operation, and call the KeyedStream function, such as reduce, fold, min, minby, max, maxby, sum, and sumby. • **fields** indicates the IDs of certain columns • **firstField** and **otherFields** are names of member variables. • **key** indicates the user-defined basis for partitioning.
def keyBy(firstField: String, otherFields: String*): KeyedStream[T, JavaTuple]	
def keyBy[K: TypeInformation](fun: T => K): KeyedStream[T, K]	
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]	

API	Description
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 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 11-29 APIs about DataStream distribution

API	Description
def partitionCustom[K: TypeInformation](partitioner: Partitioner[K], field: Int) : DataStream[T]	Use a user-defined partitioner to select target task for each element. • **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: TypeInformation](partitioner: Partitioner[K], field: String):DataStream[T]	
def partitionCustom[K: TypeInformation](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 11-30 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.

API	Description
def assignTimestamp-sAndWatermarks(assigner: AssignerWithPunc-tuatedWatermarks[T]): DataStream[T]	

Iteration

Table 11-31 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** • 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 11-32 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 11-33 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.
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. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window. **NOTE** • 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 countWindow(size: Long, slide: Long): WindowedStream[T, K, GlobalWindow]	

API	Description
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. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window.
def timeWindowAll(size: Time, slide: time AllWindowedStream[T, TimeWindow]	
def countWindow(size: Long, slide: Long): WindowedStream[T, K, GlobalWindow]	Divide windows according to the number of elements and define windows in partitioned KeyedStreams. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window. **NOTE** • 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 countWindow(size: Long): WindowedStream[T, K, GlobalWindow]	
def countWindowAll(size: Long, slide: Long): AllWindowedStream[T, GlobalWindow]	Divide windows according to the number of elements and define windows in DataStreams. • **size** indicates the duration of the window. • **slide** indicates the sliding time of window.
def countWindowAll(size: Long): AllWindowedStream[T, GlobalWindow]	

Table 11-34 lists APIs for operating windows.

Table 11-34 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. **function** indicates the window function to be executed.
	def apply[R: TypeInformation] (function: (K, W, Iterable[T], Collector[R]) => Unit): DataStream[R]	

Method	API	Description
	def reduce(function: ReduceFunction[T]): DataStream[T]	Apply a reduce function to the window and return the result. • **reduceFunction** indicates the reduce function to be executed. • **function** of **WindowFunction** indicates triggering an operation to the window after a reduce operation.
	def reduce(function: (T, T) => T): DataStream[T]	
	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. • **initialValue** indicates the initial value. • **foldFunction** indicates the fold function. • **function** of **WindowFunction** indicates triggering an operation to the window after a fold operation.
	def fold[R: TypeInformation] (initialValue: R)(function: (R, T) => R): DataStream[R]	
	def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, foldFunction: FoldFunction[T, ACC], function: WindowFunction[ACC, R, K, W]): DataStream[R]	
	def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, foldFunction: (ACC, T) => ACC, windowFunction: (K, W, Iterable[ACC], Collector[R]) => Unit): DataStream[R]	
Window All	def apply[R: TypeInformation] (function: AllWindowFunction[T, R, W]): DataStream[R]	Apply a general function to a window. The data in the window is calculated as a whole.

Method	API	Description
	def apply[R: TypeInformation](function: (W, Iterable[T], Collector[R]) => Unit): DataStream[R]	
	def reduce(function: ReduceFunction[T]): DataStream[T]	Apply a reduce function to the window and return the result. • **reduceFunction** indicates the reduce function to be executed. • **AllWindowFunction** indicates triggering an operation to the window after a reduce operation.
	def reduce(function: (T, T) => T): DataStream[T]	
	def reduce[R: TypeInformation](preAggregator: ReduceFunction[T], windowFunction: AllWindowFunction[T, R, W]): DataStream[R]	
	def reduce[R: TypeInformation](preAggregator: (T, T) => T, windowFunction: (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. • **initialValue** indicates the initial value. • **foldFunction** indicates the fold function. • **function** of **WindowFunction** indicates triggering an operation to the window after a fold operation.
	def fold[R: TypeInformation](initialValue: R)(function: (R, T) => R): DataStream[R]	
	def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, preAggregator: FoldFunction[T, ACC], windowFunction: AllWindowFunction[ACC, R, W]): DataStream[R]	
	def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, preAggregator: (ACC, T) => ACC, windowFunction: (W, Iterable[ACC], Collector[R]) => Unit): DataStream[R]	

Method	API	Description
Window and Window All	def sum(position: Int): DataStream[T]	Sum a specified column of the window data. **field** and **position** indicate a specific column of the data.
	def sum(field: String): DataStream[T]	
	def min(position: Int): DataStream[T]	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.
	def min(field: String): DataStream[T]	
	def max(position: Int): DataStream[T]	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.
	def max(field: String): DataStream[T]	
	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 11-35 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]	
def flatMap[R: TypeInformation](coFlatMapper: CoFlatMapFunction[IN1, IN2, R]): DataStream[R]	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 flatMap[R: TypeInformation](fun1: (IN1, Collector[R]) => Unit, fun2: (IN2, Collector[R]) => Unit): DataStream[R]	
def flatMap[R: TypeInformation](fun1·IN1 => TraversableOnce[R], fun2: IN2 => TraversableOnce[R]): DataStream[R]	

Join Operation

Table 11-36 APIs about join operation

API	Description
def join[T2] (otherStream: DataStream[T2]): JoinedStreams[T, T2]	Join two DataStreams using a given key in a specified window. The key value of the join operation is specified by the **where** and **eaualTo** method, indicating filtering data with equivalent conditions from two DataStreams.
def coGroup[T2] (otherStream: DataStream[T2]): CoGroupedStreams[T, T2]	Co-group two DataStreams using a given key in a specified window. The key value of the coGroup operation is specified by the where and eaualTo method, indicating partitioning two DataStreams using equivalent conditions.

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 11-37**. 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 11-37 lists all RESTful API paths supported by Flink.

Table 11-37 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 *running*, *finished*, *failed*, *canceled*.
/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.
/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.

Path	Description
/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.
/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.

Path	Description
/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 11-38 describes variables listed in **Table 11-37**.

Table 11-38 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

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 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 11-61**. **Figure 11-62** shows the normal situation when the title is displayed.

Figure 11-61 Title not displayed on the page

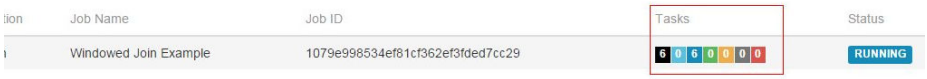
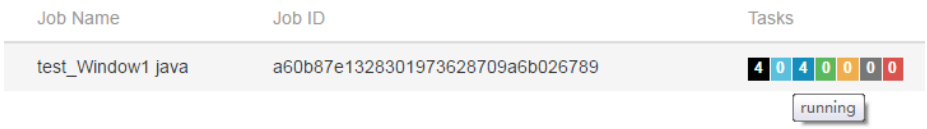


Figure 11-62 Title displayed normally



Answer

If the Chrome browser does not display the title, perform the following procedure:

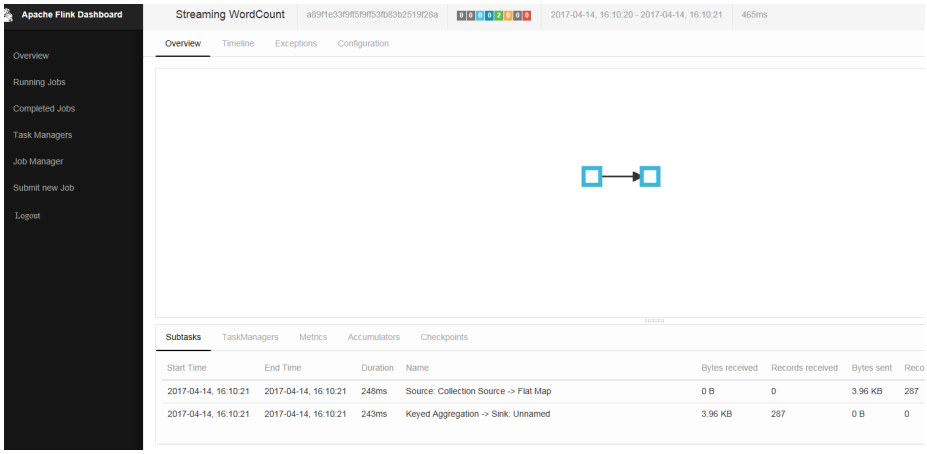
Check whether a tool that affects the tooltip display by the Chrome browser is running. If so, shut down the tool.

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 11-63?

Figure 11-63 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 11-64**.
- 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 11-65**.

Figure 11-64 Time monitoring

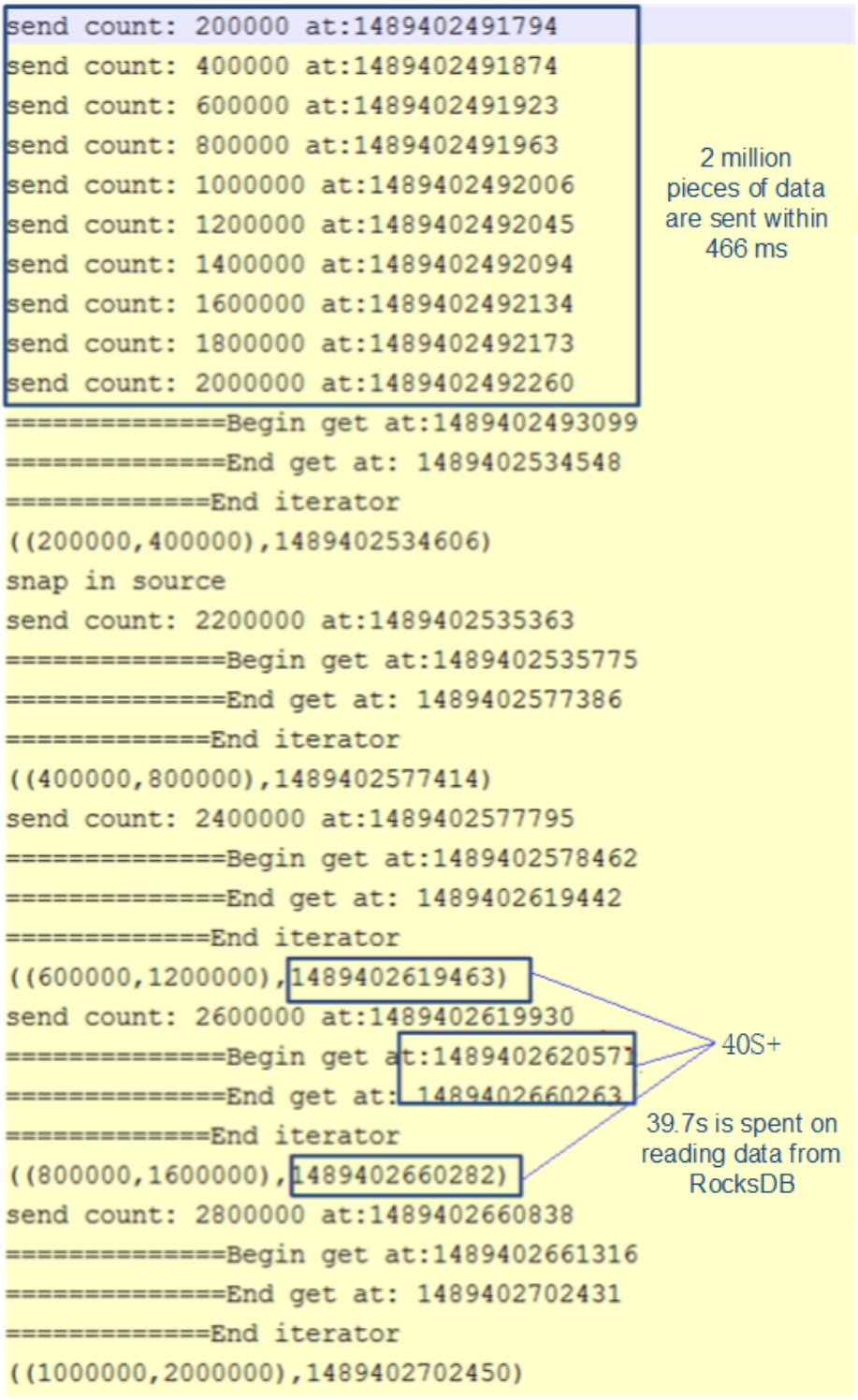
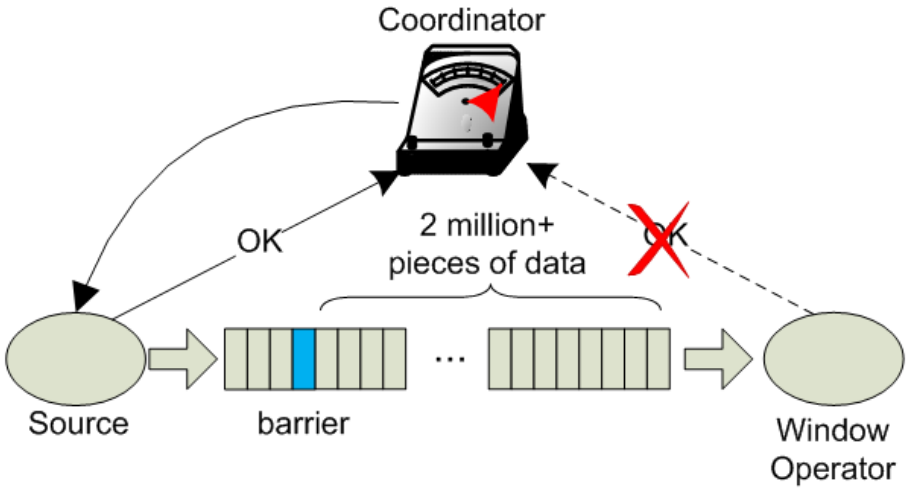


Figure 11-65 Relationship



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 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

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 **2**. If yes, go to **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?

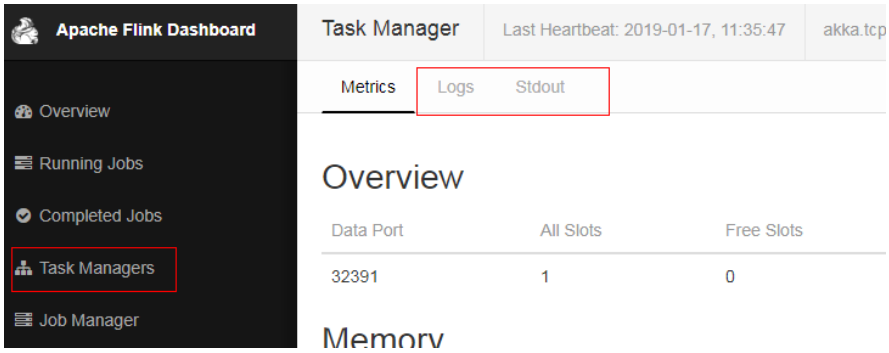
Question

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** > **Stdouts** 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(RocksDBStateBacken
d.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

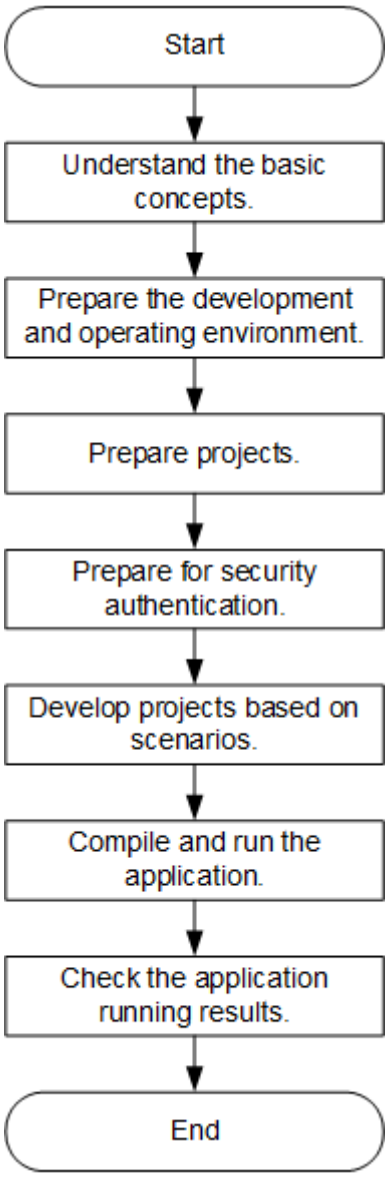


Table 12-2 HBase application development process description

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	• 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	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. • For x86 nodes that run clients, the following JDKs can be used: – 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: – OpenJDK: 1.8.0_272 **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 com.ibm.jsse2.overrideDefaultTLS 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.**
IntelliJ IDEA installation and configuration	Tool used for developing HBase applications. The version must be 2019.1 or other compatible version. **NOTE** • 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 the 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. Download and decompress the client.
     - **Log in to the FusionInsight Manager portal** and choose **Cluster** > **Dashboard** > **More** > **Download Client**. Set **Select Client Type** to **Configuration Files Only** 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 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 the 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 the 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

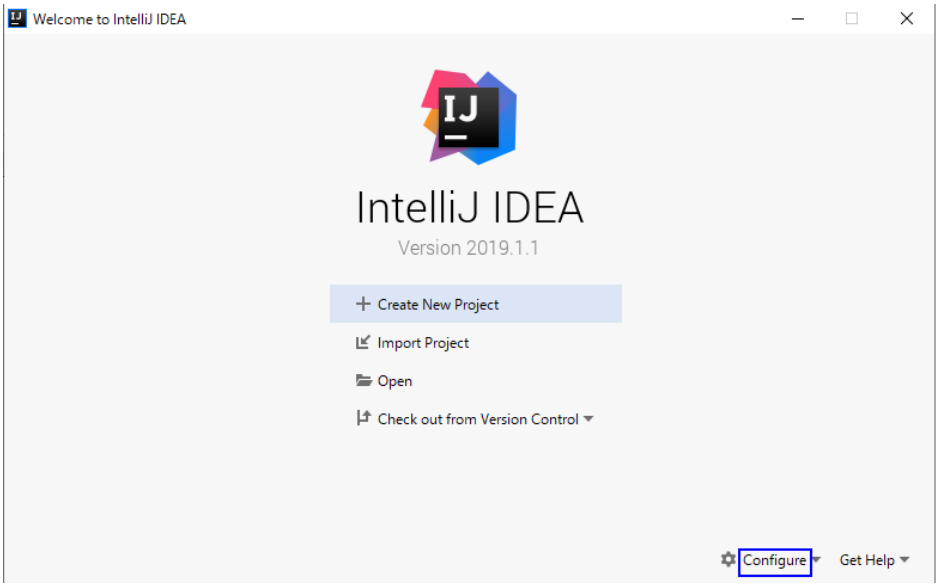
Step 1 Obtain the sample project folder **hbase-example** in the **src/hbase-examples** directory where the sample code is decompressed. For details, see **Obtaining Sample Projects from Huawei Mirrors**.

Step 2 Copy the keytab authentication files **user.keytab** and **krb5.conf** files obtained in **Preparing the 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 **READEME.md** file of the corresponding sample project.

Step 3 After the IntelliJ IDEA and JDK are installed, configure the JDK in IntelliJ IDEA.

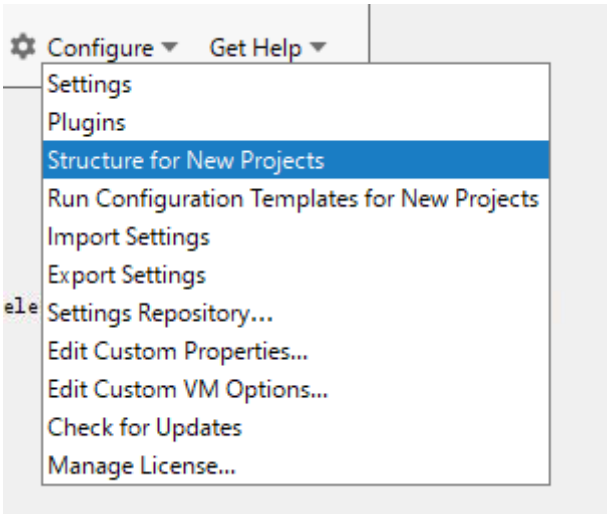
1. Start the IntelliJ IDEA and click **Configure**.

Figure 12-2 Quick Start



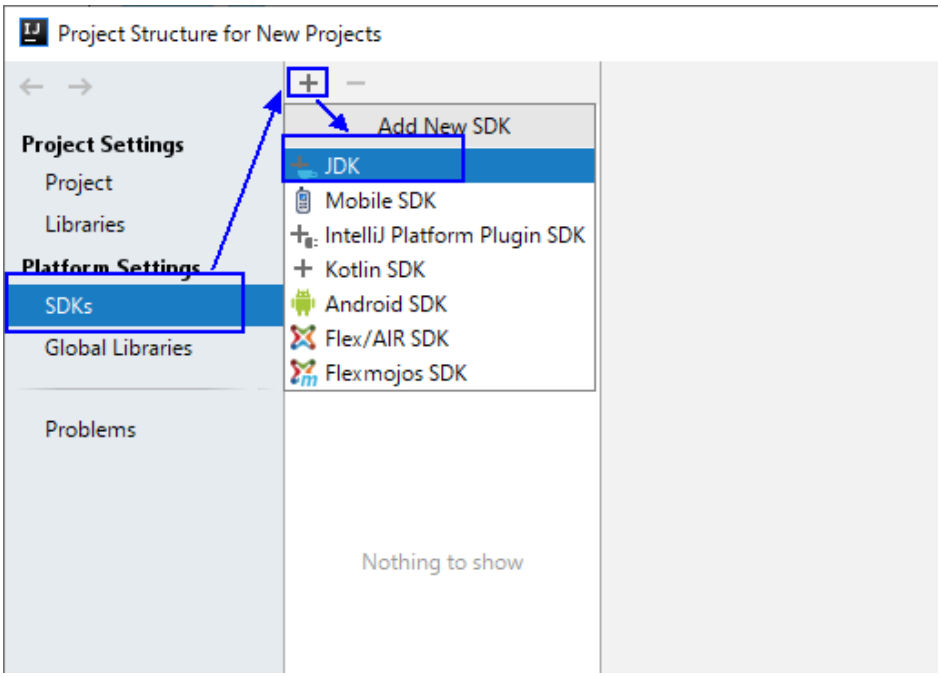
2. Click **Structure for New Projects** from the drop-down list.

Figure 12-3 Configure



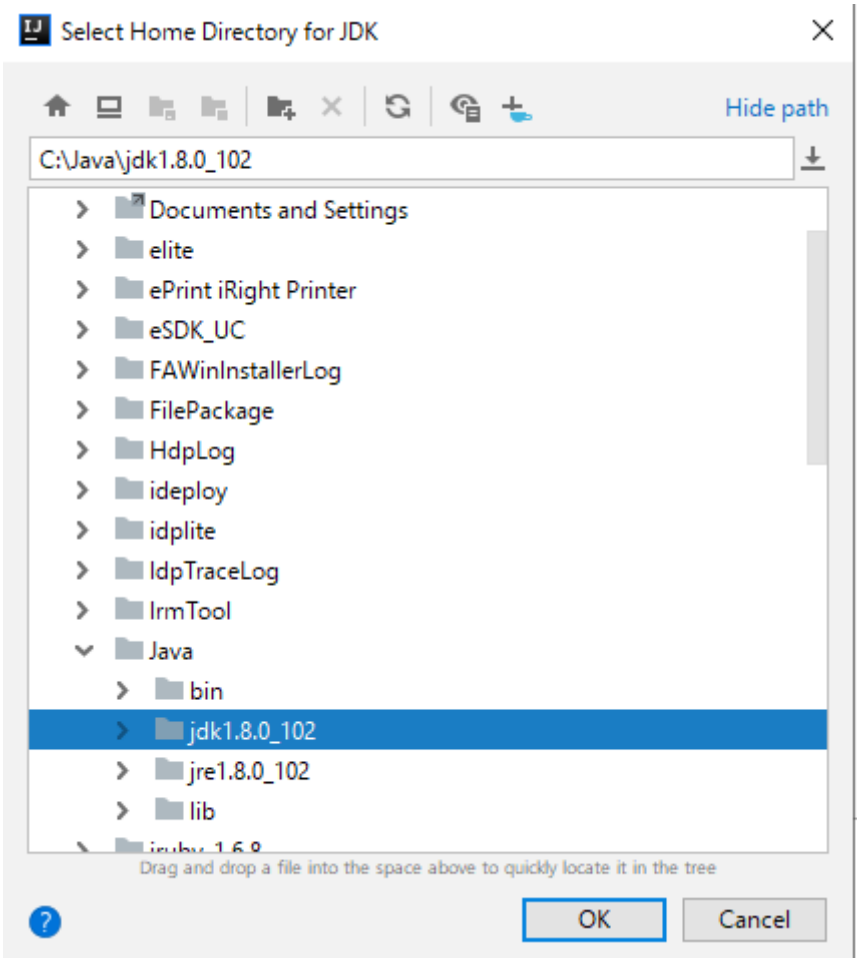
3. On the displayed **Project Structure for New Projects** page, select **SDKs** and click the plus sign (+) to add the **JDK**.

Figure 12-4 Project Structure for New Projects



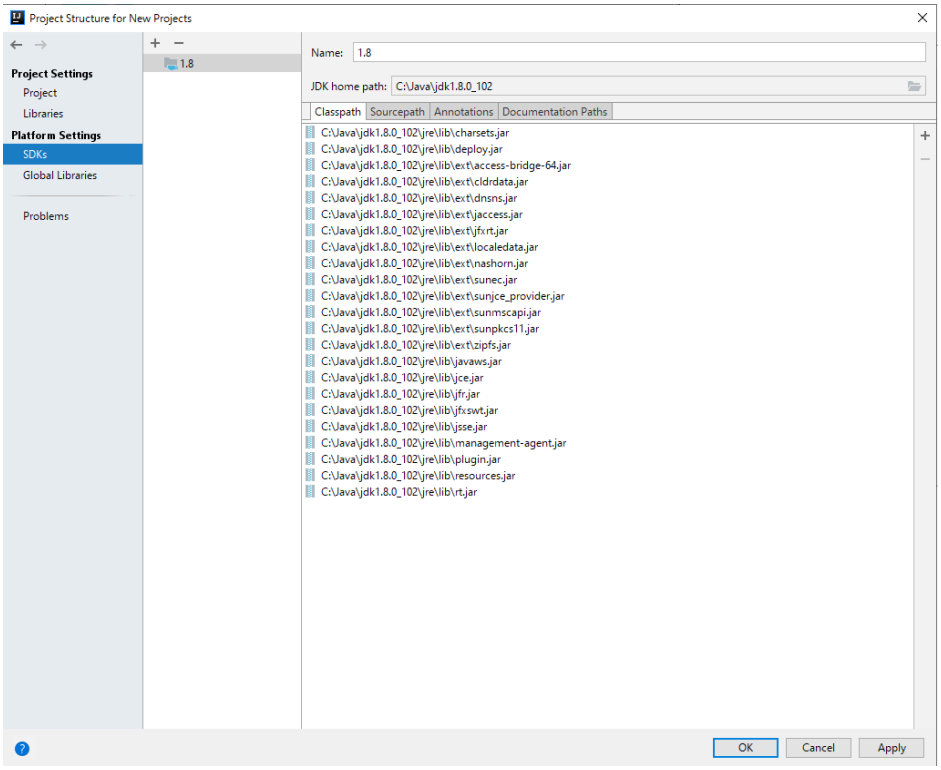
4. In the **Select Home Directory for JDK** window that is displayed, select the JDK directory, and click **OK**.

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

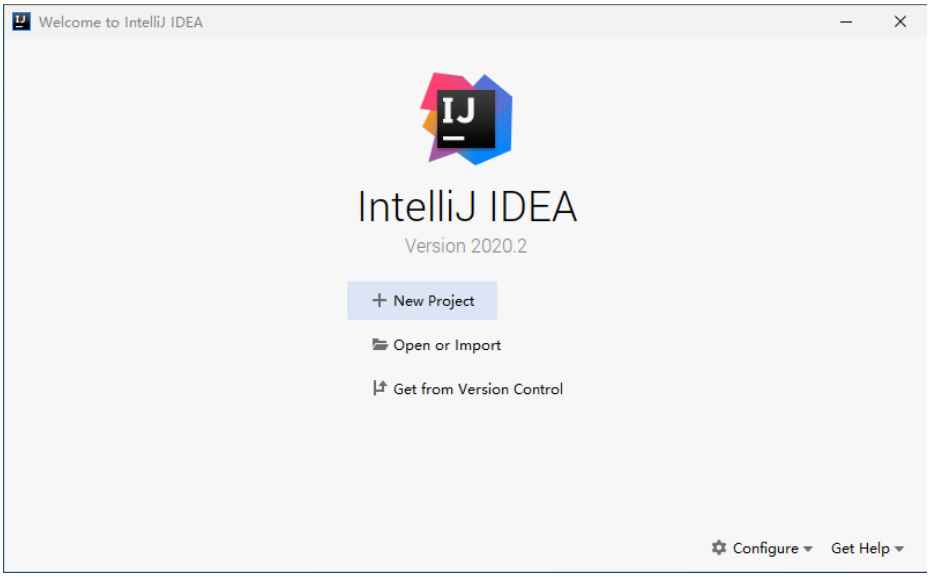
The operations vary by the IDEA version. The operation procedure varies according to the actual version.

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.

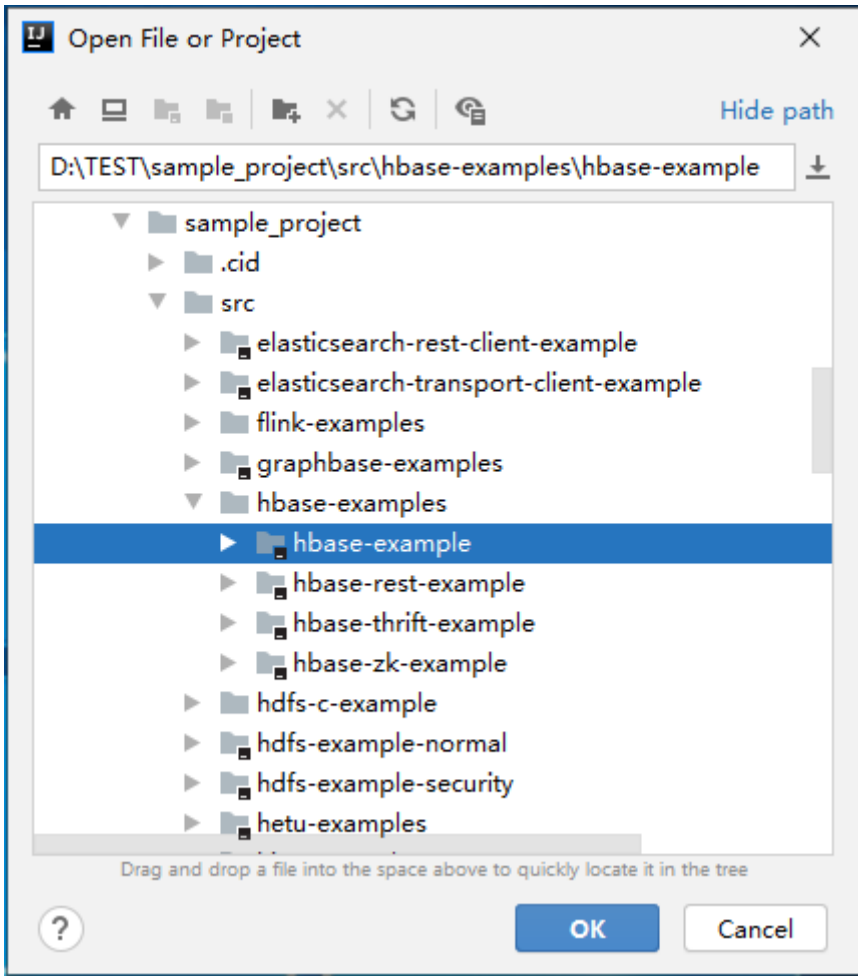
   For the IDEA tool that has been used, you can choose **File** > **Import project**... on the main interface to import the sample project.

Figure 12-7 Open or Import (**Quick Start** page)



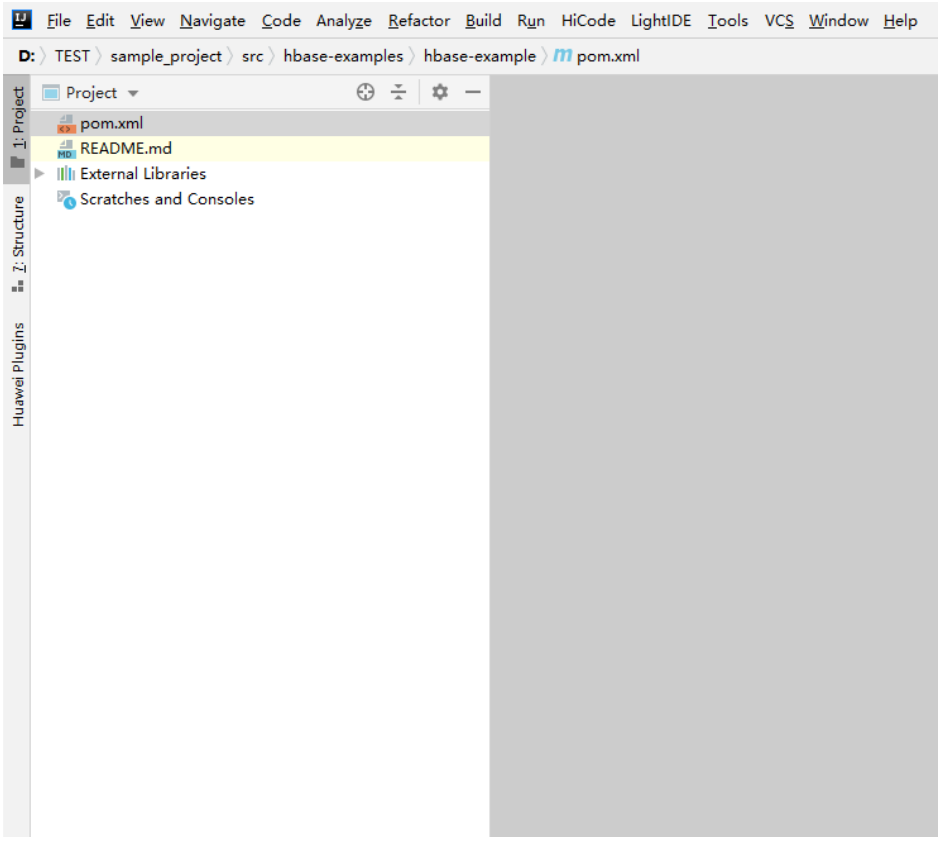
2. Select the example project folder **hbase-example**, and click **OK**.

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

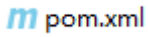
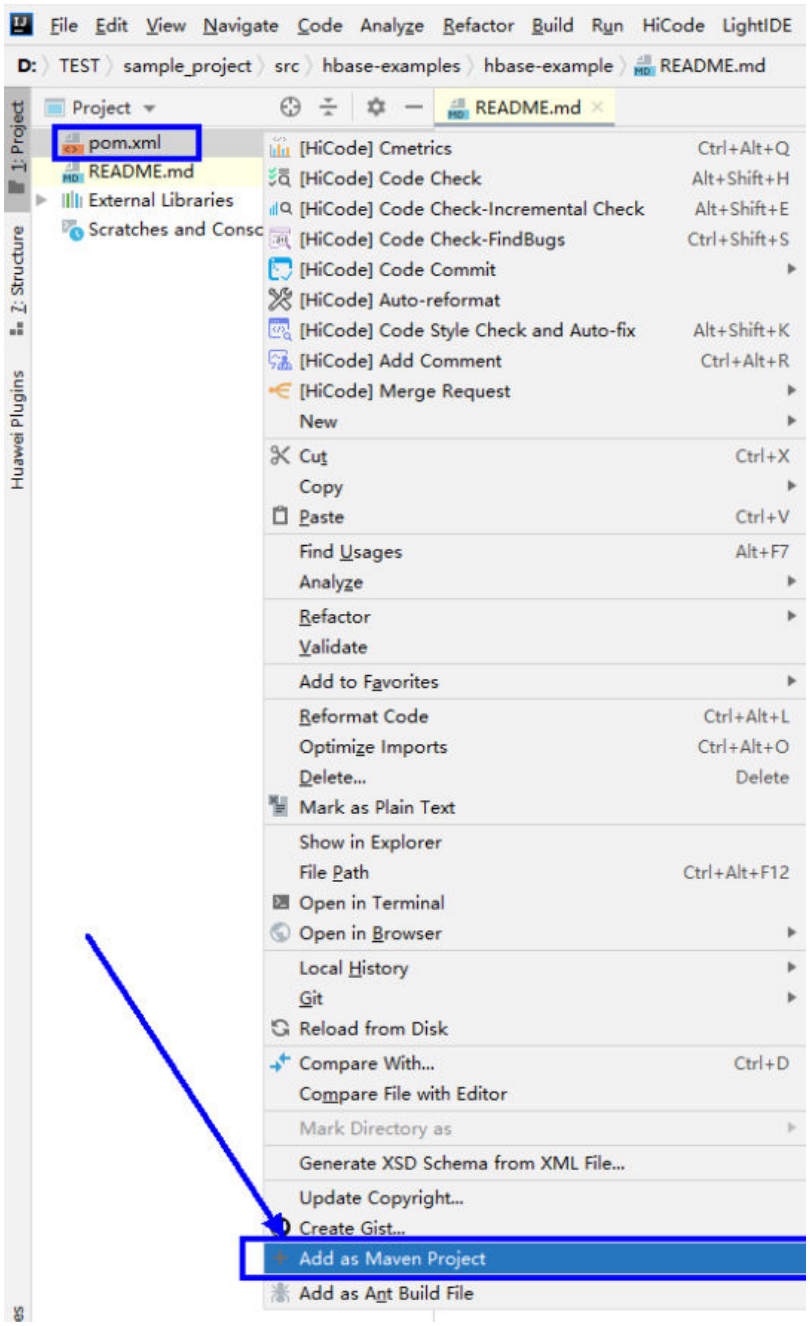
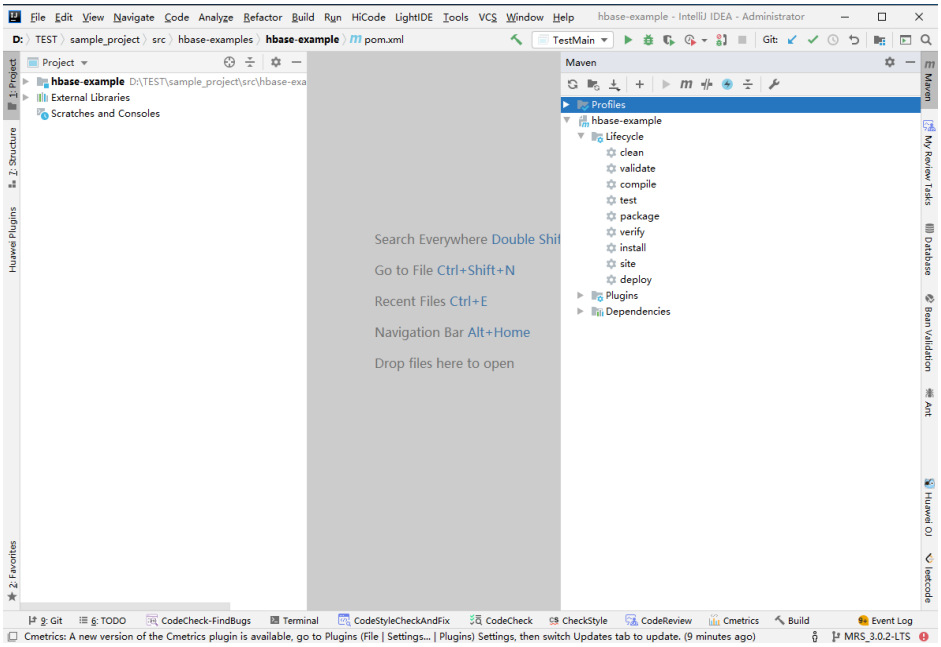


Figure 12-11 Add as Maven Project



In this case, the IDEA can identify the project as a Maven project.

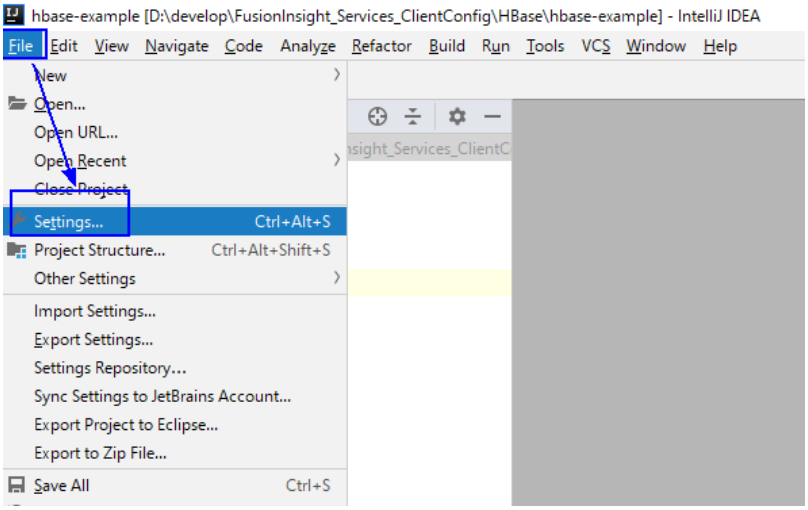
Figure 12-12 Sample project displayed in IDEA as a Maven project



Step 5 Set the Maven version used by the project.

1. Choose **File > Settings...** from the main menu of IntelliJ IDEA.

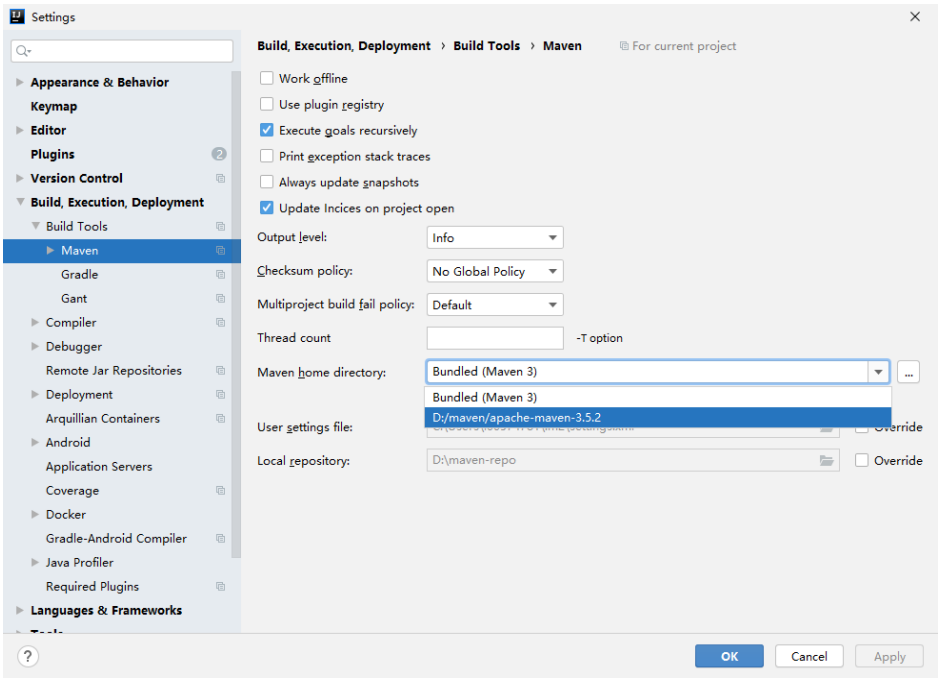
Figure 12-13 Settings



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**.

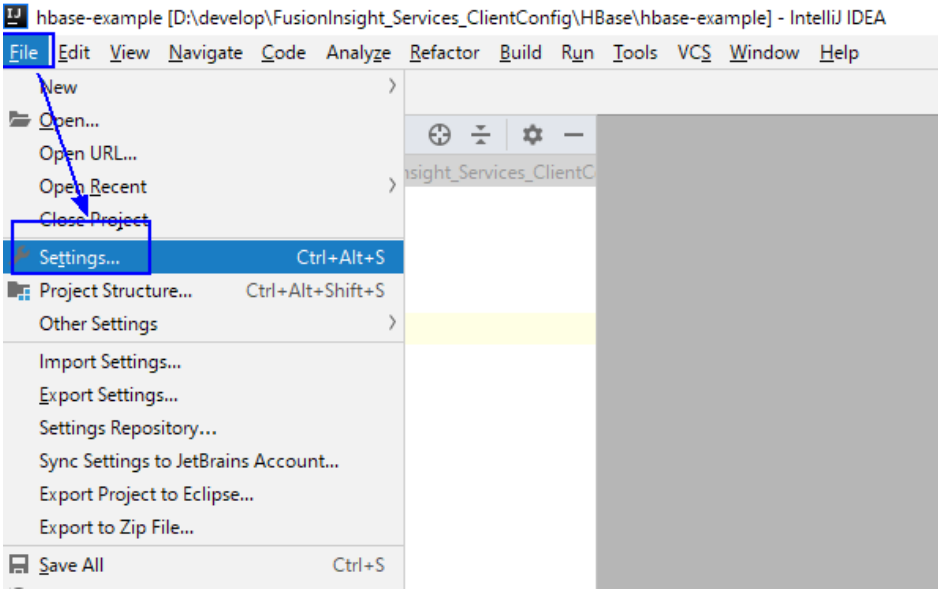
Figure 12-14 Selecting the local Maven installation directory



Step 6 Set the IntelliJ IDEA text file coding format to prevent garbled characters.

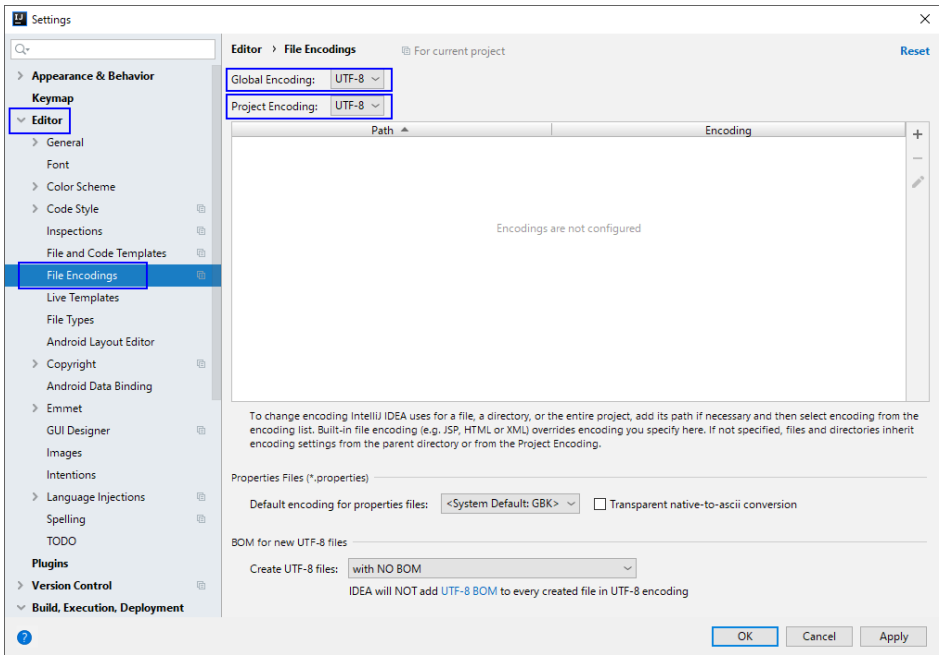
1. On the IntelliJ IDEA menu bar, choose **File >Settings...**.

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 the 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 R**EADME.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_8.1.2.2/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 nor 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"; [7]
};
```

[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.

[7] **password**: Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or environment variables.

12.3 Developing an Application

12.3.1 HBase data read/write sample program

12.3.1.1 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.

CAUTION

HBase tables are stored in *Namespace*:*Table name* format. If no namespace is specified when a table is created, the table is stored in **default**. The **hbase** namespace is the system table namespace. Do not create service tables or read or write data in the system table namespace.

12.3.1.2 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 **Table 12-4**.	For details, see **Creating a Table**.
2	Import user data.	For details, see **Inserting Data**.
3	Add the **Education** column family, and add diplomas and titles to the user information table.	For details, see **Modifying a Table**.
4	Query user names and addresses by user ID.	For details, see **Reading Data Using Get**.
5	Query information by user name.	For details, see **Filtering Data**.
6	To improve query performance, create or delete secondary indexes.	For details, see **Creating a Secondary Index** and **Secondary Index-based Query**.
7	Deregister users, and delete user data.	For details, see **Deleting Data**.
8	Delete the user information table after service A ends.	For details, see **Deleting a Table**.

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.

12.3.1.3 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);
  }
```

12.3.1.4 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.

12.3.1.5 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 effeciency,fit for cold data
        // SNAPPY has low compression rate, but high compression and
        // decompression effeciency,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);
```


12.3.1.6 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**.

12.3.1.7 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.

12.3.1.8 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.

12.3.1.9 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.

12.3.1.10 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.");
  }
```

12.3.1.11 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.

12.3.1.12 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);
```

12.3.1.13 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.

12.3.1.14 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.");
}
```

12.3.1.15 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.hindex.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)"}
```

12.3.1.16 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();

     // initilize 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.

12.3.1.17 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.examples**package.

```
/**
 * 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.");
}
```

12.3.1.18 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.");
}
```

12.3.1.19 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 PrepareStatement
    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.");
}
```

12.3.1.20 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.

Add the Active/Standby Cluster Configuration to the hbase-dual.xml File

Step 1 Save the keytab authentication files **user.keytab** and **krb5.conf** of the active cluster obtained when **Preparing the 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>
```

Step 5 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);
 }
```

Step 6 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());
}
```

Step 7 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
```

----End

12.3.1.21 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
```

12.3.2 HBase Rest API Invoking Sample Program

12.3.2.1 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);
  }
  ```

12.3.2.2 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);
}
```

12.3.2.3 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.

CAUTION

HBase tables are stored in *Namespace*:*Table name* format. If no namespace is specified when a table is created, the table is stored in **default**. The **hbase** namespace is the system table namespace. Do not create service tables or read or write data in the system table 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);
}
```

12.3.2.4 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 Accessing the HBase ThriftServer Sample Program

12.3.3.1 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);
    }
 }
```

12.3.3.2 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.");
}
```


12.3.3.3 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 write 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.");
}
```

12.3.4 Sample Program for HBase to Access Multiple ZooKeepers

12.3.4.1 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
    String confDirPath = TestZKSample.class.getClassLoader().getResource("").getPath() + File.separator;
[1]
    //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 pakcage 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.

12.4 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.

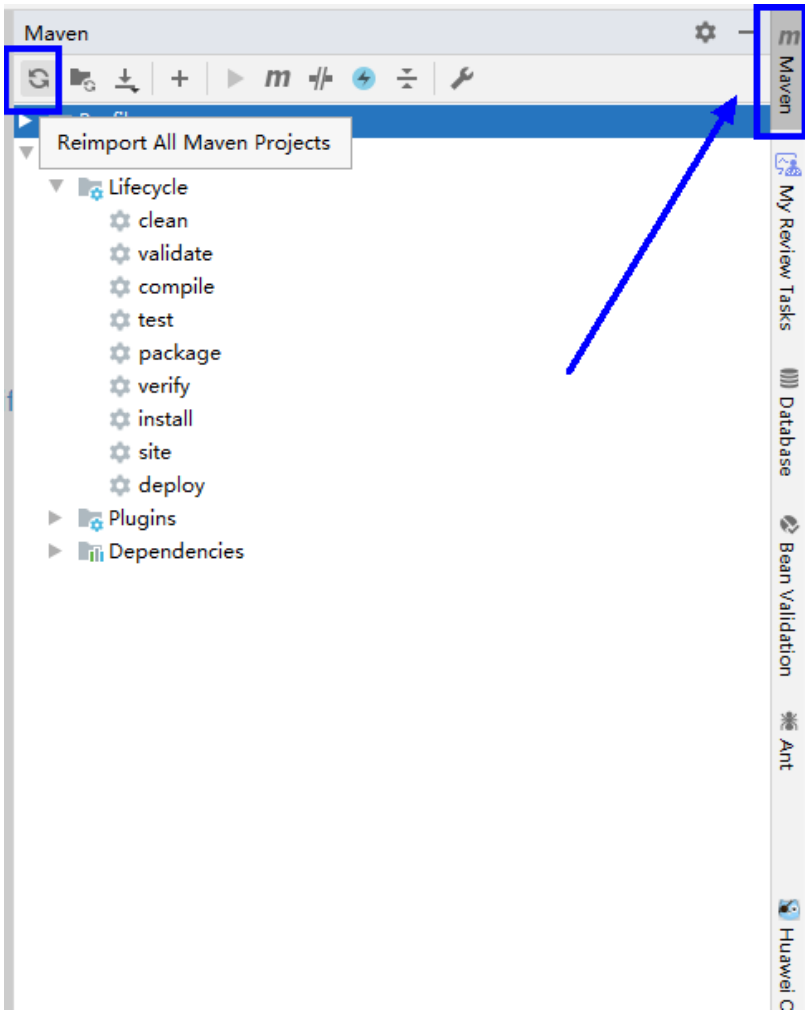
NOTE

- 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.

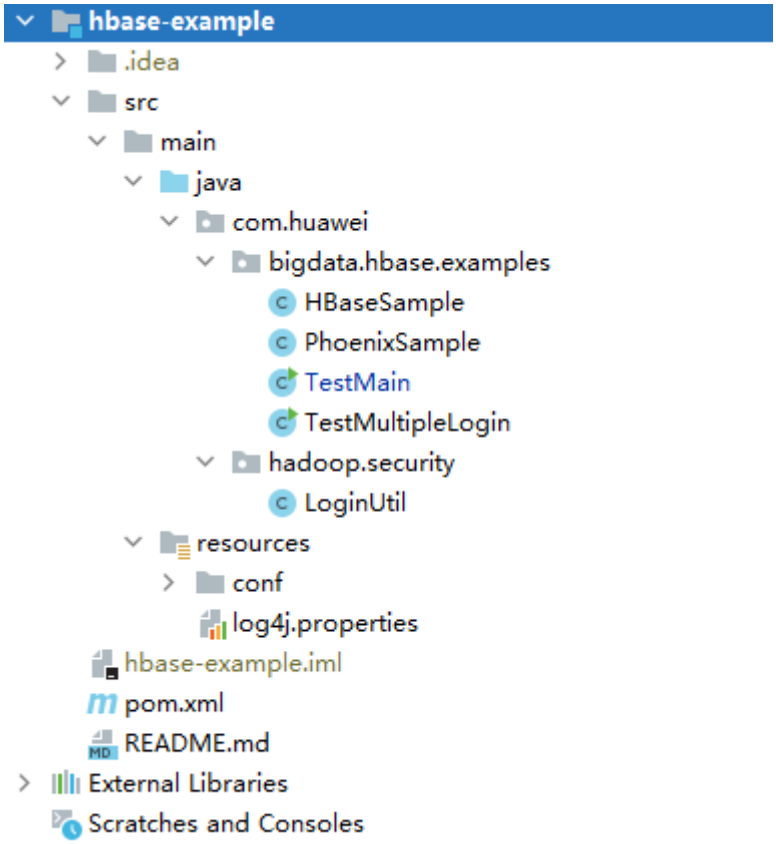
Figure 12-17 reimport projects



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

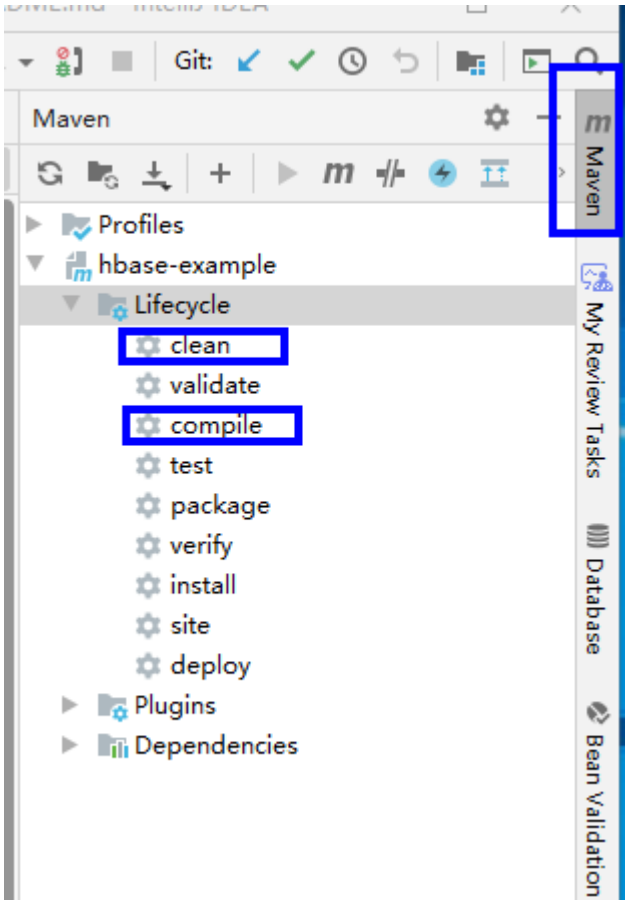


1. Compile an application.
   - Method 1:

     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.

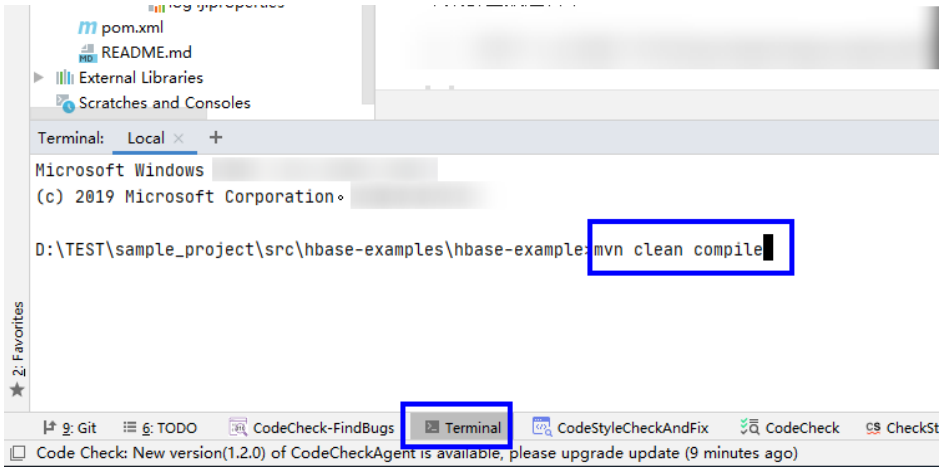
Figure 12-19 clean and **compile** tools of Maven



- Method 2:

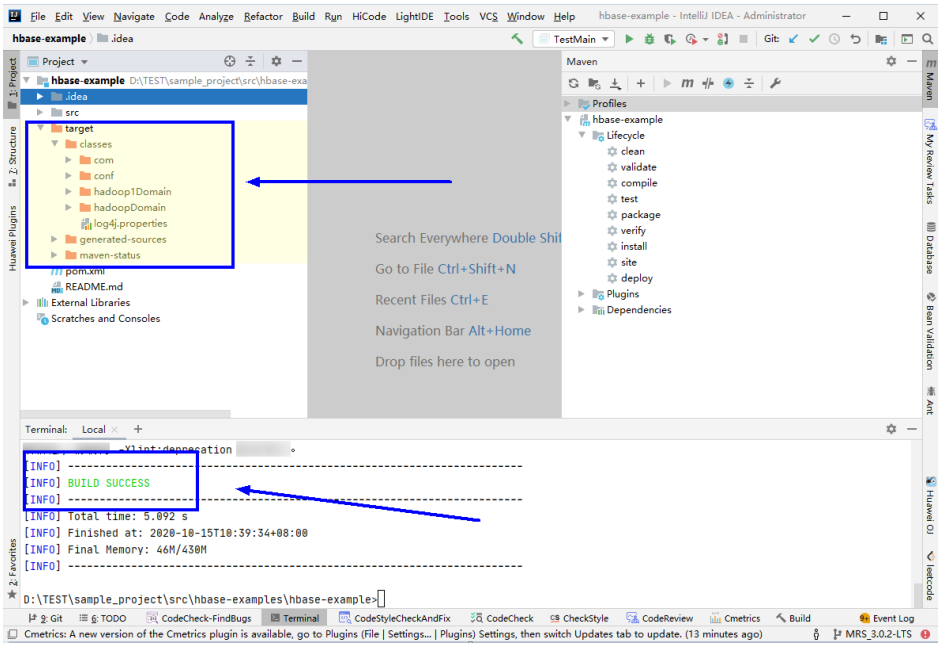
  Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

  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.

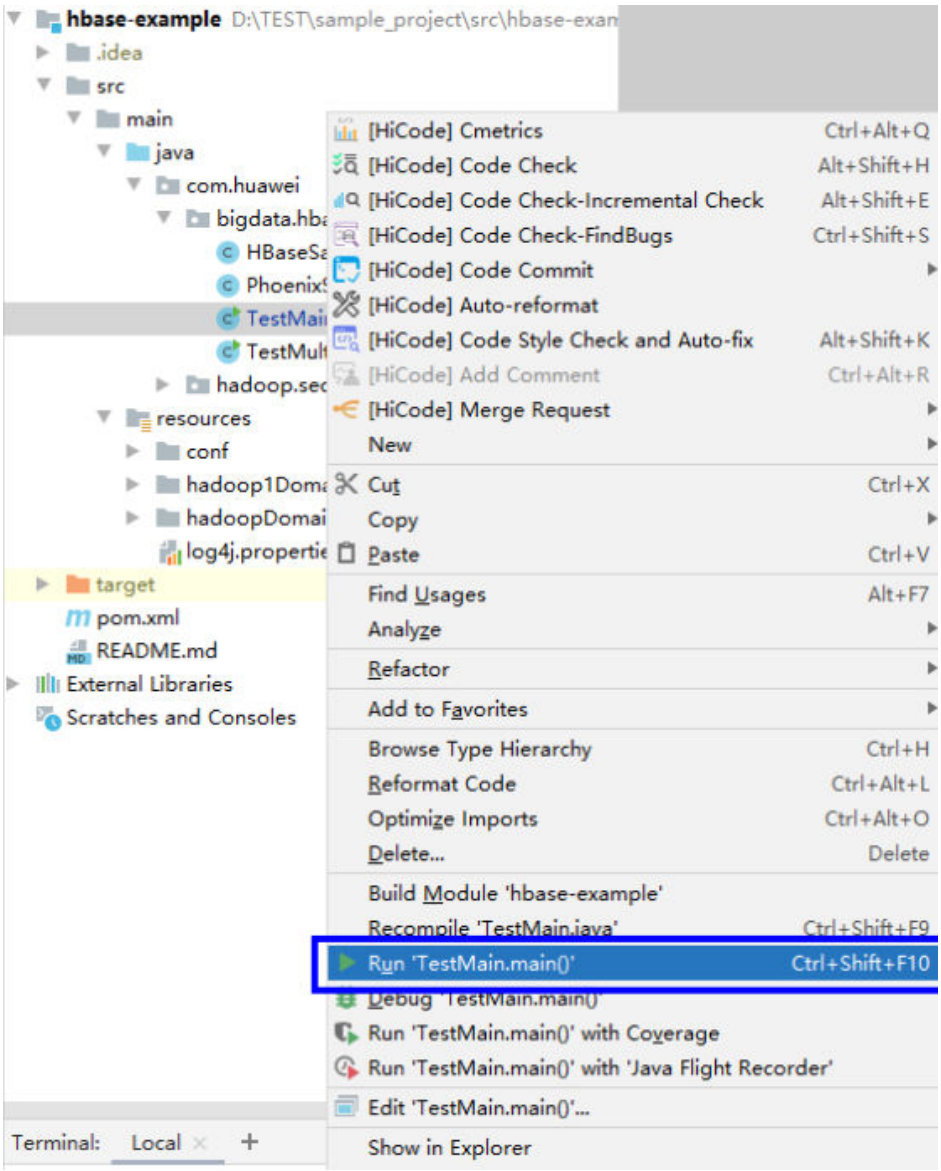
Figure 12-21 Compilation completed



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.

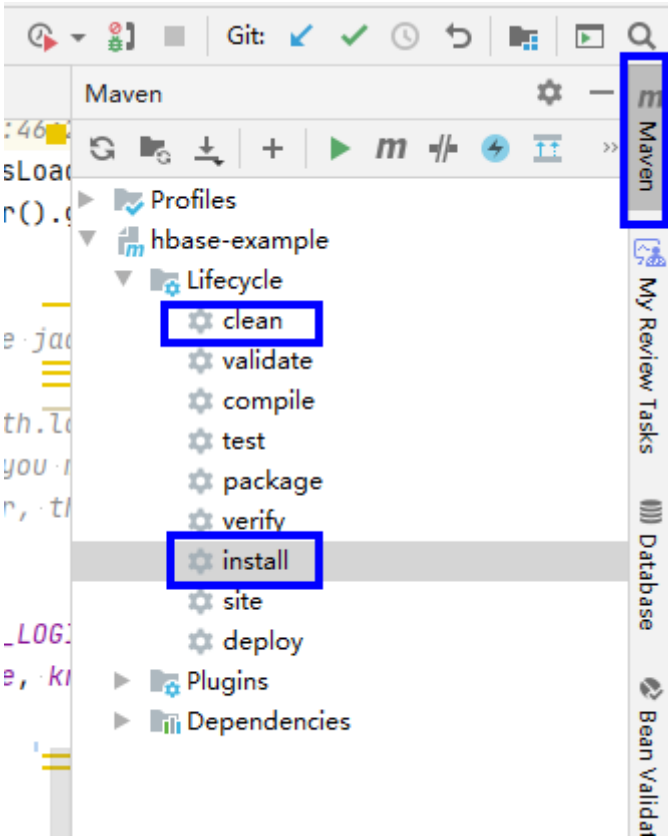
You can build a JAR file in either of the following ways:

- Method 1:

  Choose **Maven** > *Sample projectname* > **Lifecycle** > **clean**, double click **clean** to run the **clean** command of Maven.

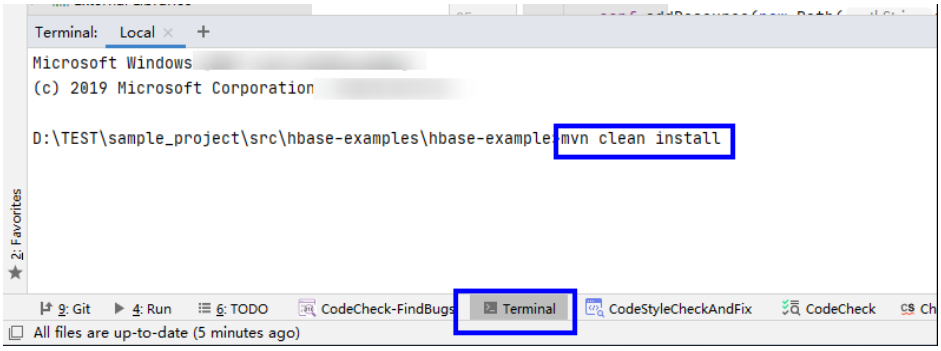
  Choose **Maven** > *Sample project name* > **Lifecycle** > **install**, double click **install** to run the **install** command of Maven.

  Figure 12-23 Maven tools: **clean** and **installl**



- 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

- You can view the execution details of the submitted application in the run logs. For example, after the **hbase-sample** is successfully executed, the following information is displayed:

```
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 PrepareStatement
              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

Configuration Item	Description	Default Value	Level
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
hbase.dualclient.standby.executor.thread.max	Maximum size of the thread pool for processing requests to the standby cluster	100	Optional

Configuration Item	Description	Default Value	Level
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 12-7 Basic specifications

Metric Name	Description	Log level
total_request_count	Total number of queries in a period	INFO

Metric Name	Description	Log level
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 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
95thPercentileLaten-cy(ms)	Maximum latency of 95% requests	INFO
99thPercentileLaten-cy(ms)	Maximum latency of 99% requests	INFO

Metric Name	Description	Log level
99.9PercentileLaten-cy(ms)	Maximum latency of 99.9% requests	INFO
99.99PercentileLaten-cy(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 **hbase:meta** table, **true** is returned. Otherwise, **false** 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 **true**, 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(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.
boolean isTableDisabled(TableName 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 **StartRow**, **StopRow**, and **caching**.

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

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 interfaces of Phoenix are the same as those of the open-source community. For details, see **http://phoenix.apache.org/**.

NOTE

This version does not include the Phoenix secondary index feature of the open-source community.

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.

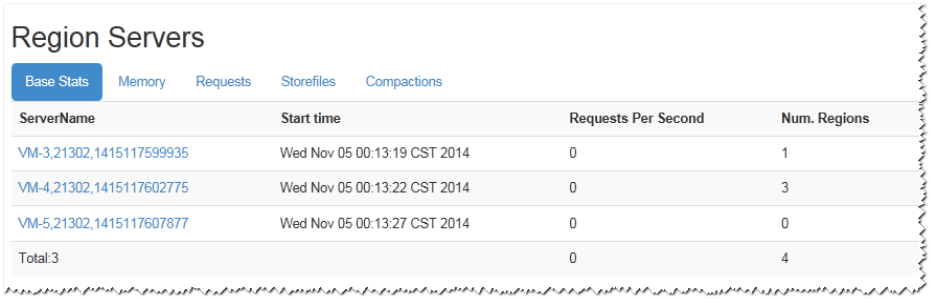
NOTE

Contact the administrator to obtain a service account that has the permission to access the WebUI and obtain its password.

Procedure

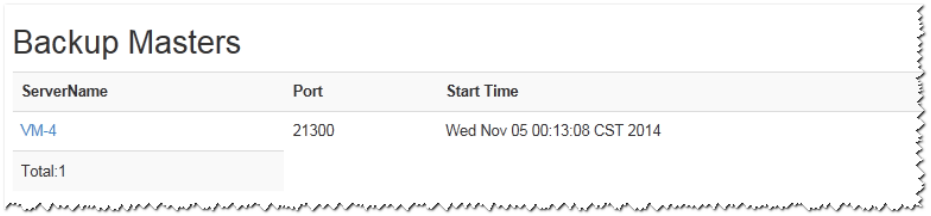
1. **Log in to the FusionInsight Manager portal**. Choose **Cluster** > *Name of the desired cluster* > **Services** > **HBase** > **HMaster(Active)** to open the HBase WebUI.
2. On the home page of the HBase WebUI, view the HBase summary. The following information is included.
   a. On the **Region Servers** page, view basic information about the RegionServer, as shown in **Figure 12-25**.

   Figure 12-25 Region Servers

   b. On the **Backup Masters** page, view information about the Backup Master, as shown in **Figure 12-26**.

   Figure 12-26 Backup Masters

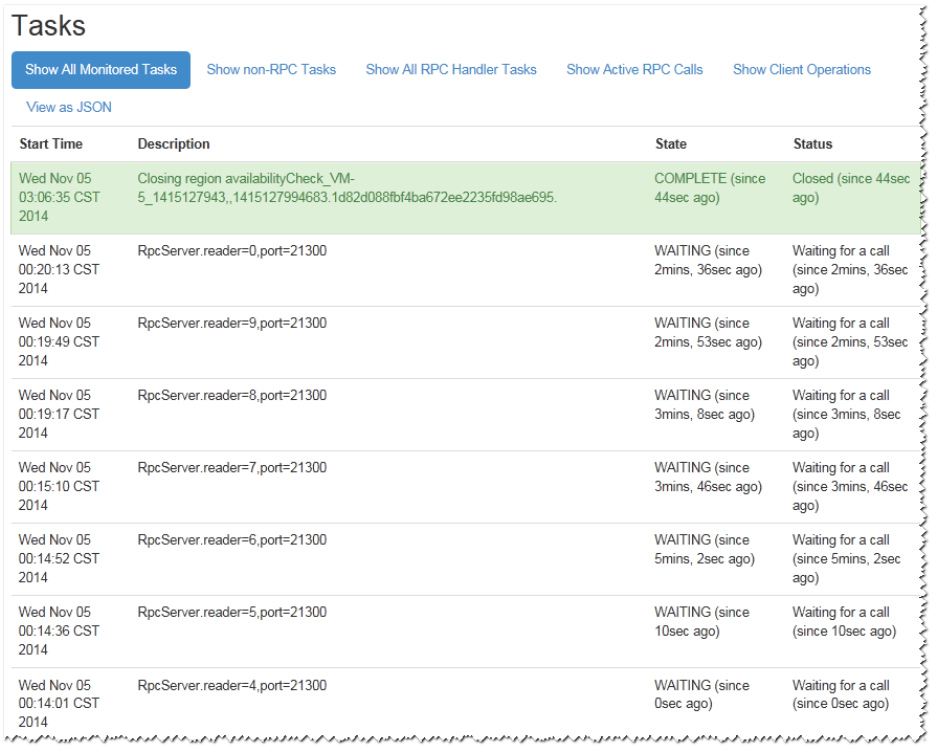
   c. On the **Tables** page, view HBase table information, including **User Tables**, **Catalog Tables**, and **Snapshots**, as shown in **Figure 12-27**.

   Figure 12-27 Tables

   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



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

4. On the **Debug dump** page of the HBase WebUI, view the HBase debug information, as shown in **Figure 12-30**.

Figure 12-30 Debug dump

5. On the **HBase Configuration** page of the HBase WebUI, view the HBase control information, as shown in **Figure 12-31**.

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>
```

12.5.4 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. For details, see **Creating a Role** to configure roles with required permissions. For details about how to bind roles with users, see **Creating a User**. 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

12.5.5 FAQs

12.5.5.1 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-5.0.0-HBase-2.0-hw-ei.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

12.5.5.2 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.

12.5.5.3 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

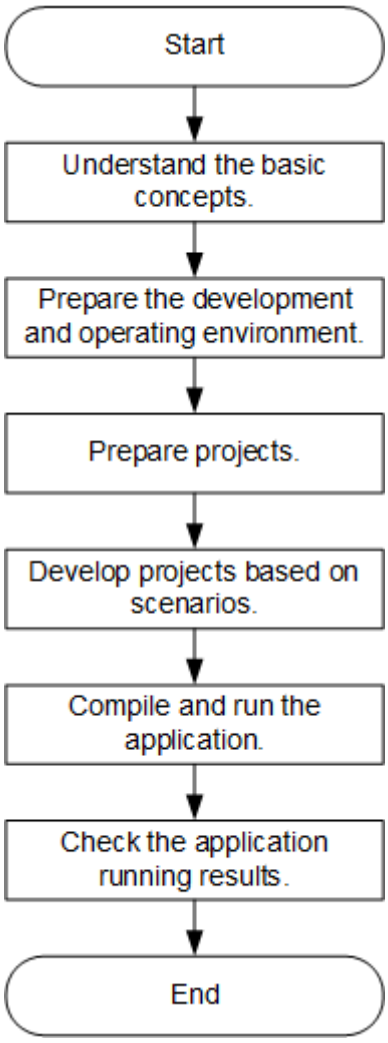


Table 13-2 HBase application development process description

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.	**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**
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**

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	• 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	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. • For x86 nodes that run clients, the following JDKs can be used: – 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: – OpenJDK: 1.8.0_272 **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 com.ibm.jsse2.overrideDefaultTLS 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.**
IntelliJ IDEA installation and configuration	Tool used for developing HBase applications. The version must be 2019.1 or other compatible version. **NOTE** • 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
7-zip	Used to decompress **.zip** and **.rar** 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. Download and decompress the client.
     - Log in to the FusionInsight Manager portal and choose **Cluster** > **Dashboard** > **More** > **Download Client**. Set **Select Client Type** to **Configuration Files Only** 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

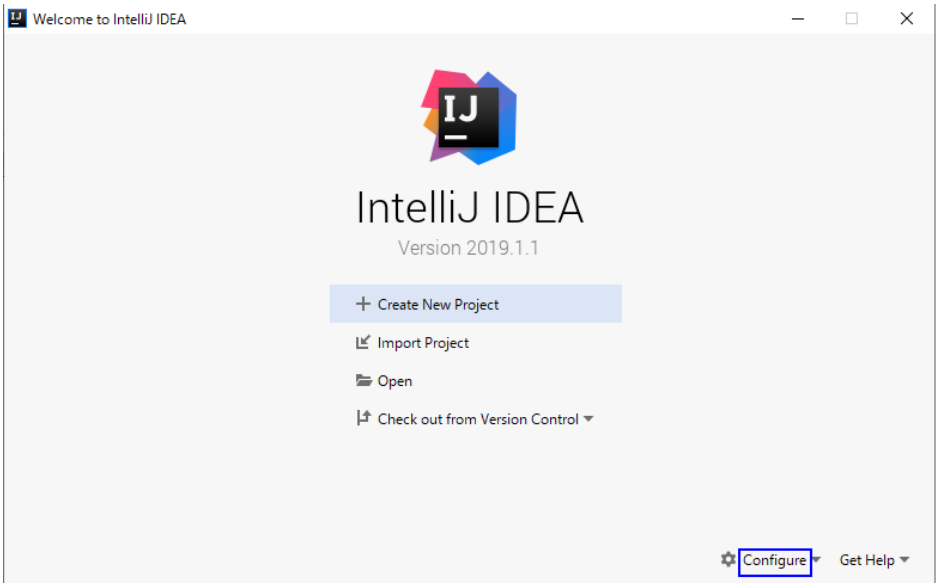
Step 1 Obtain the sample project folder **hbase-example** in the **src/hbase-examples** directory where the sample code is decompressed. For details, see **Obtaining Sample Projects from Huawei Mirrors**.

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 **READEME.md** file of the corresponding sample project.

Step 3 After the IntelliJ IDEA and JDK are installed, configure the JDK in IntelliJ IDEA.

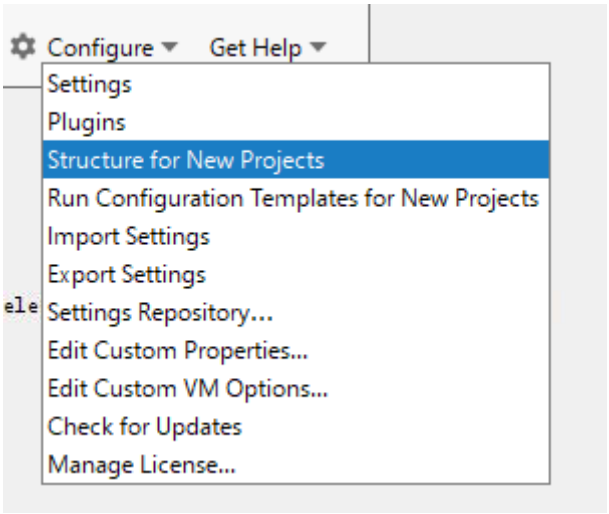
1. Start the IntelliJ IDEA and click **Configure**.

Figure 13-2 Quick Start



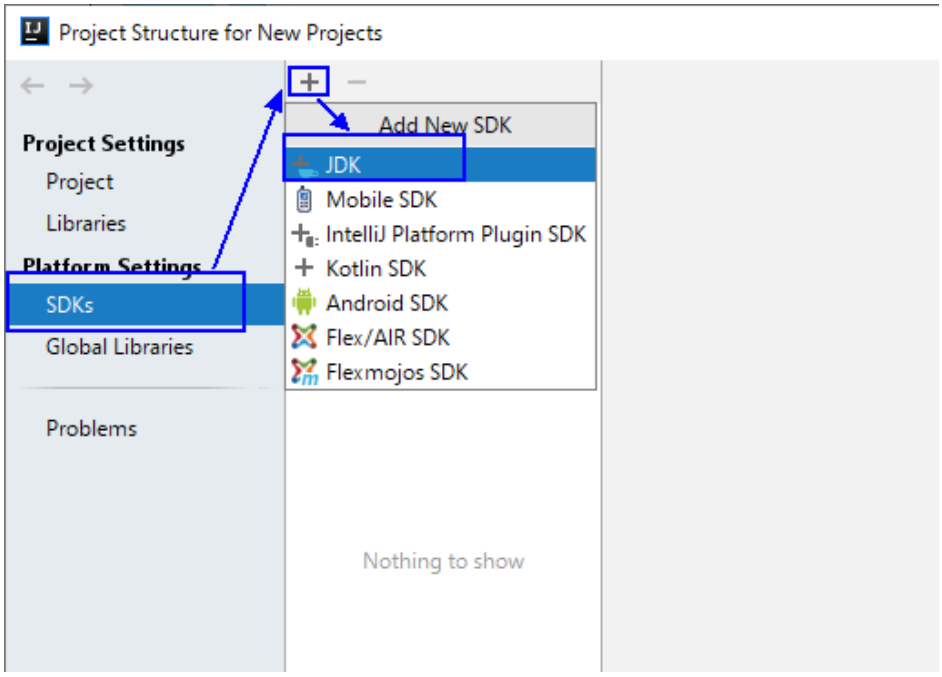
2. Click **Structure for New Projects** from the drop-down list.

Figure 13-3 Configure



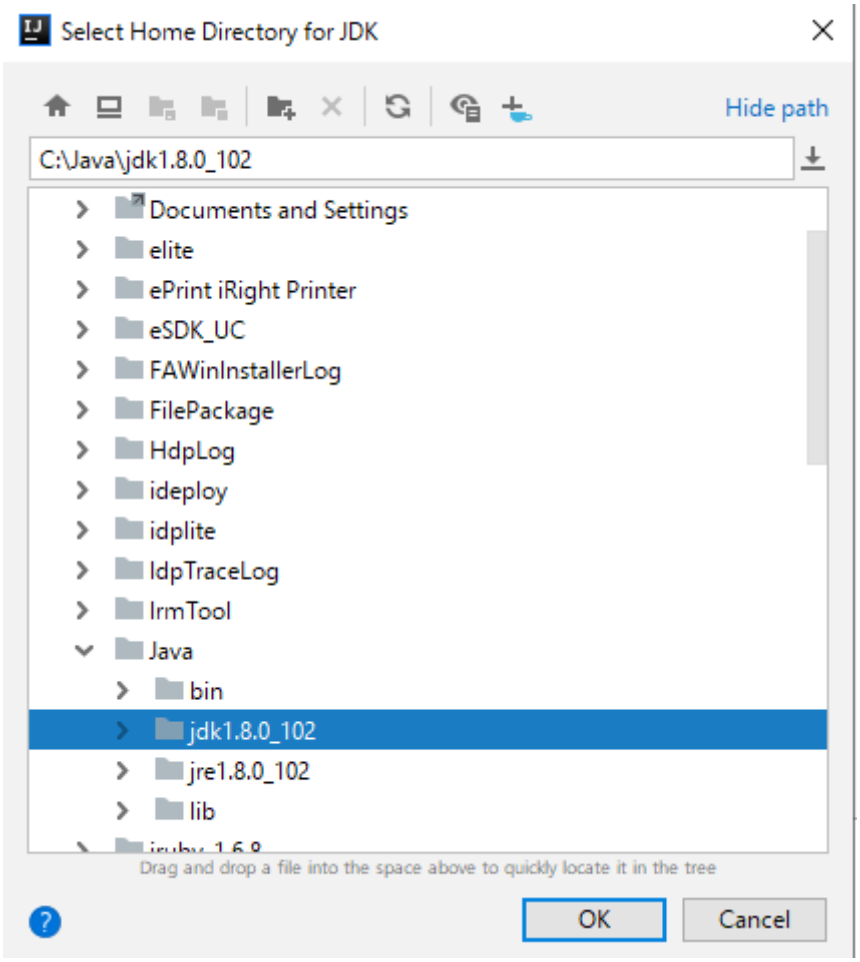
3. On the displayed **Project Structure for New Projects** page, select **SDKs** and click the plus sign (+) to add the **JDK**.

Figure 13-4 Project Structure for New Projects



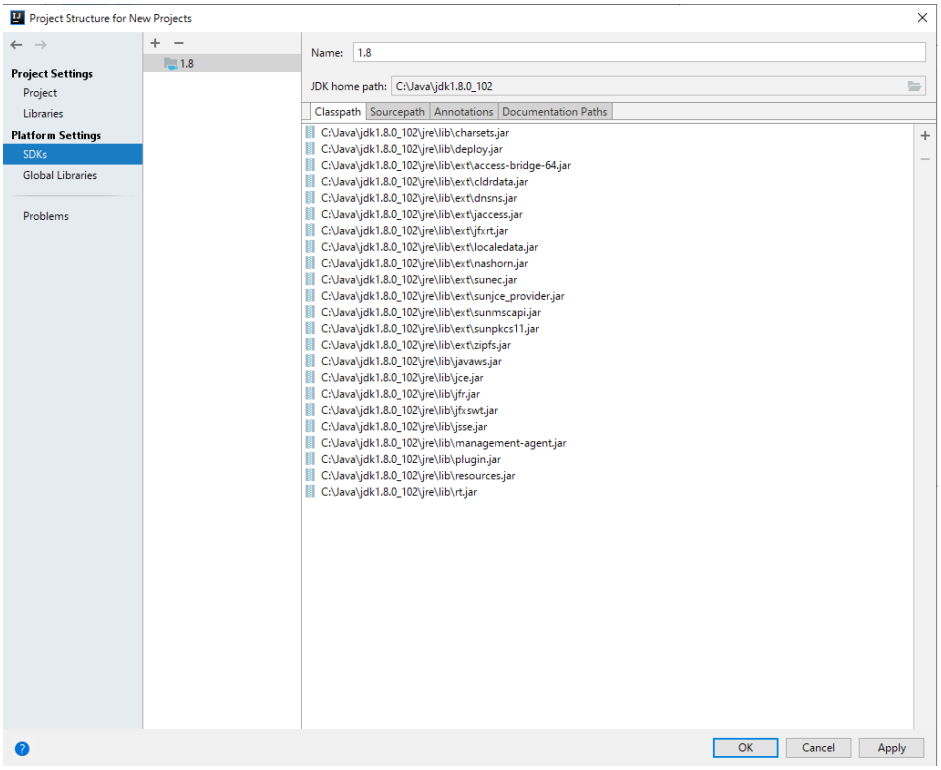
4. In the **Select Home Directory for JDK** window that is displayed, select the JDK directory, and click **OK**.

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

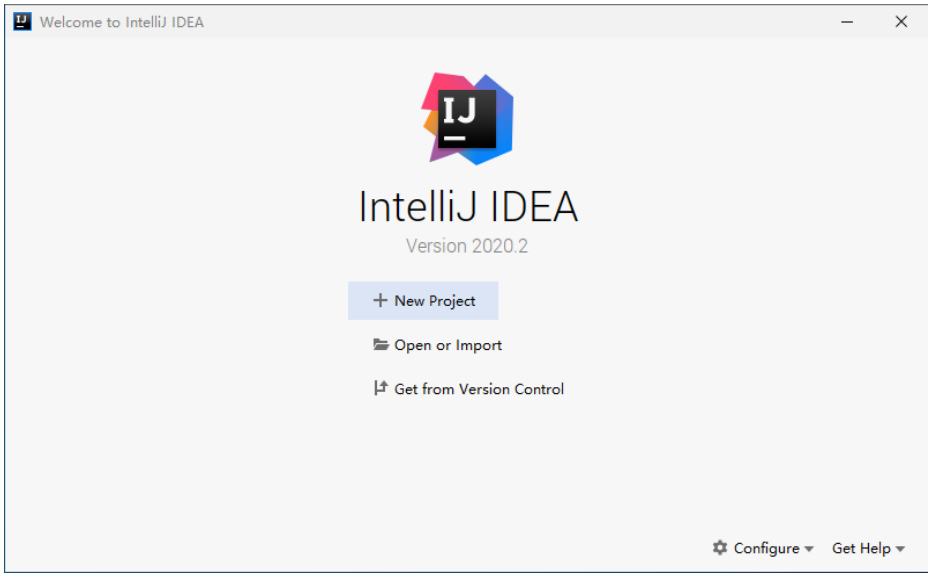
The operations vary by the IDEA version. The operation procedure varies according to the actual version.

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.

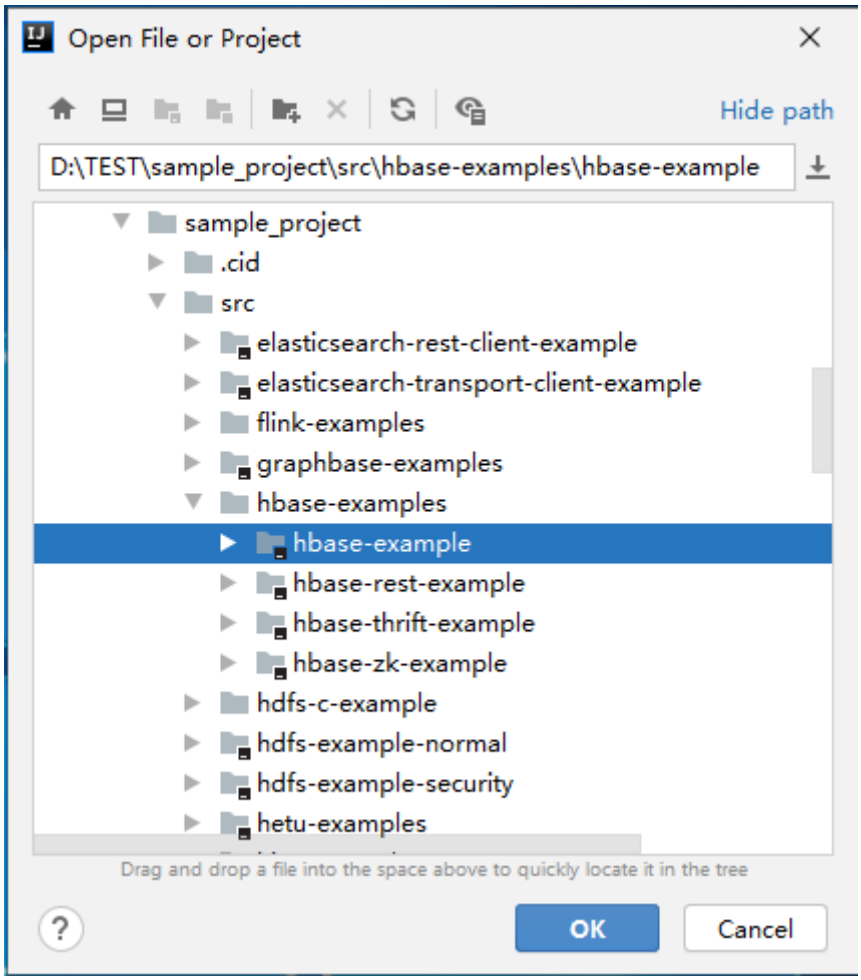
   For the IDEA tool that has been used, you can choose **File** > **Import project**... on the main interface to import the sample project.

Figure 13-7 Open or Import (**Quick Start** page)



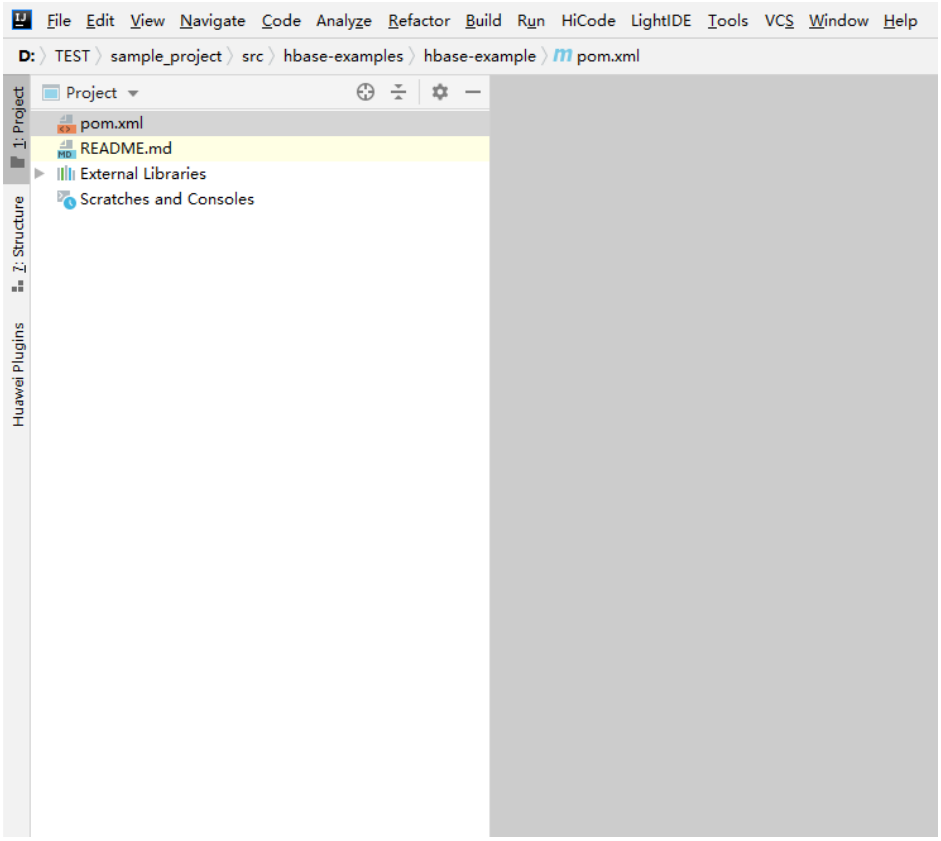
2. Select the example project folder **hbase-example**, and click **OK**.

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

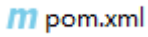
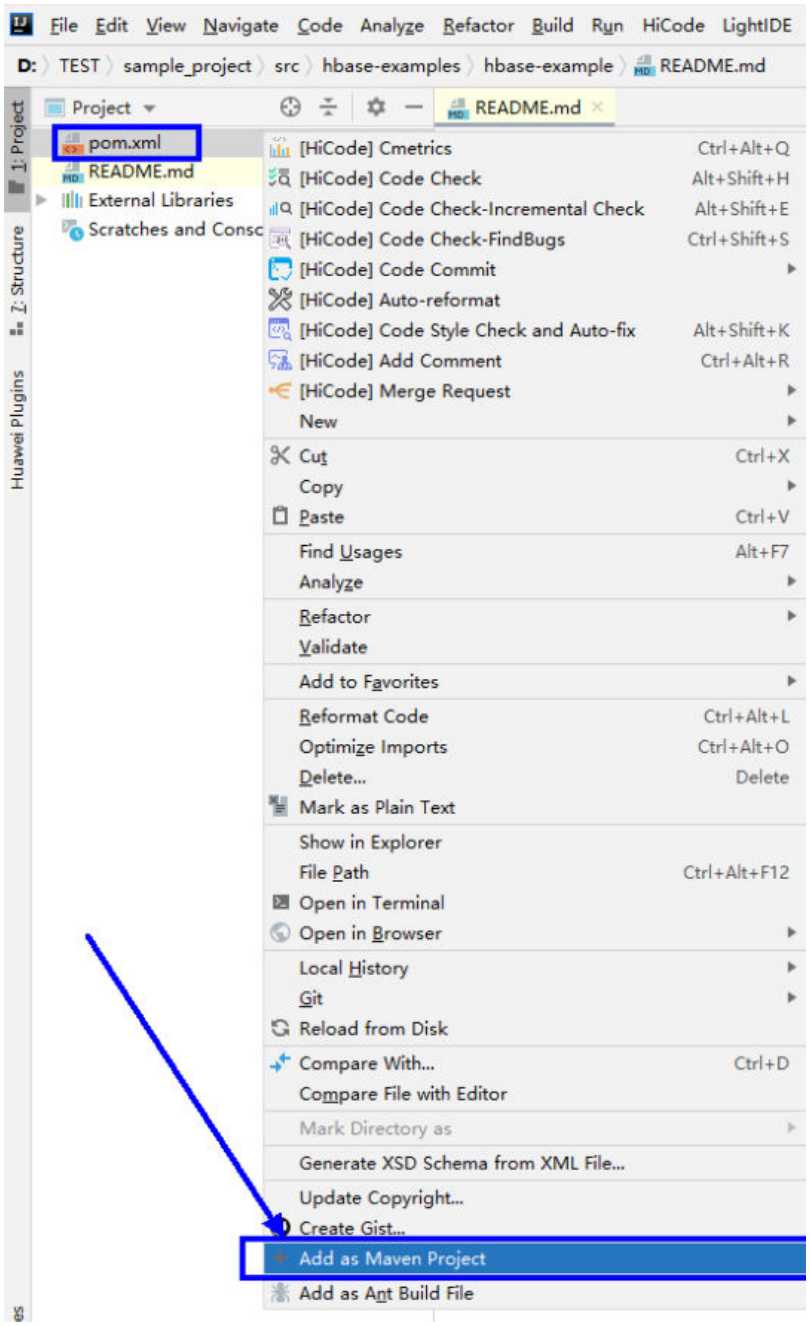
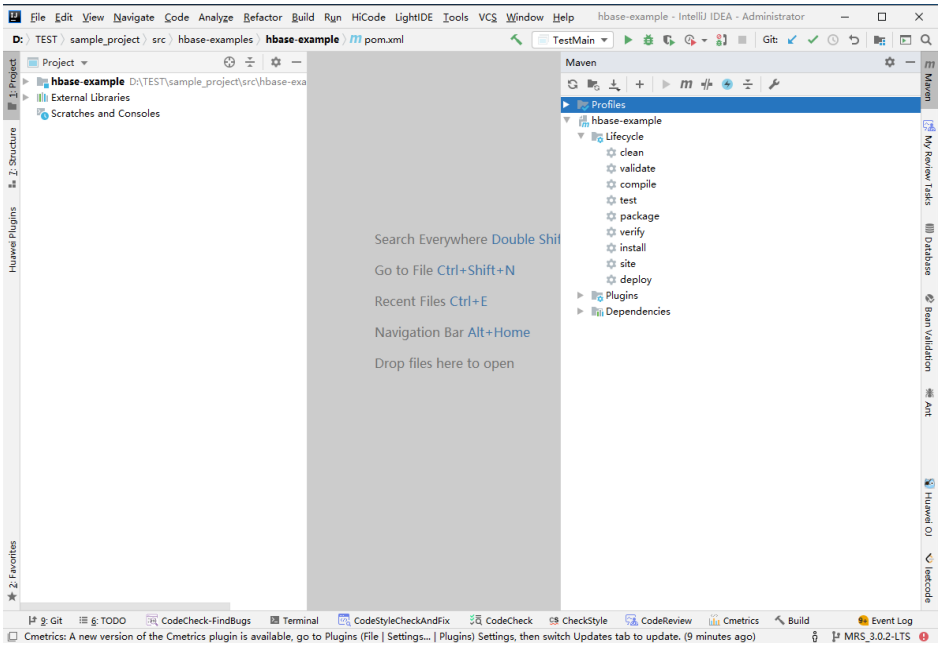


Figure 13-11 Add as Maven Project



In this case, the IDEA can identify the project as a Maven project.

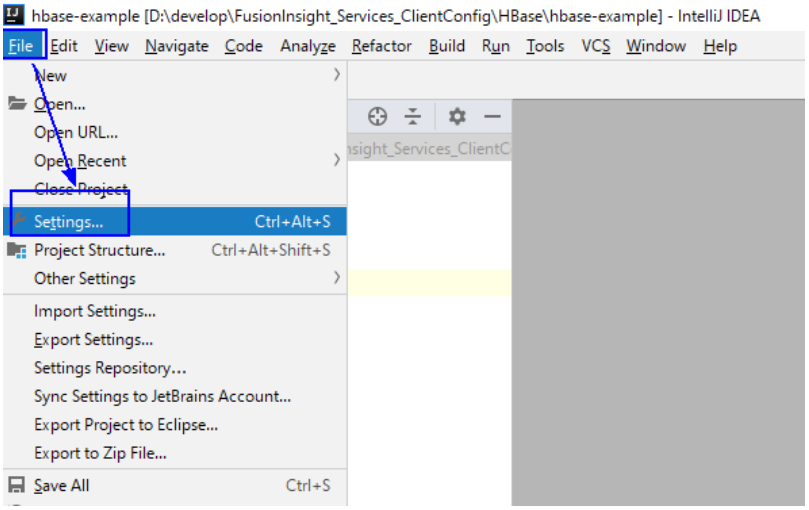
Figure 13-12 Sample project displayed in IDEA as a Maven project



Step 5 Set the Maven version used by the project.

1. Choose **File > Settings...** from the main menu of IntelliJ IDEA.

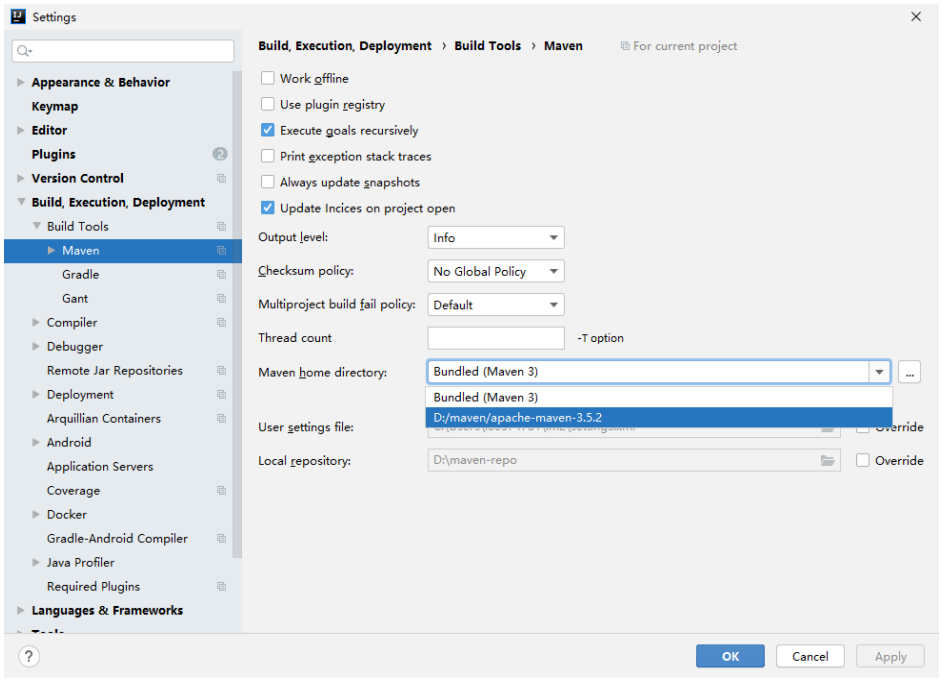
Figure 13-13 Settings



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**.

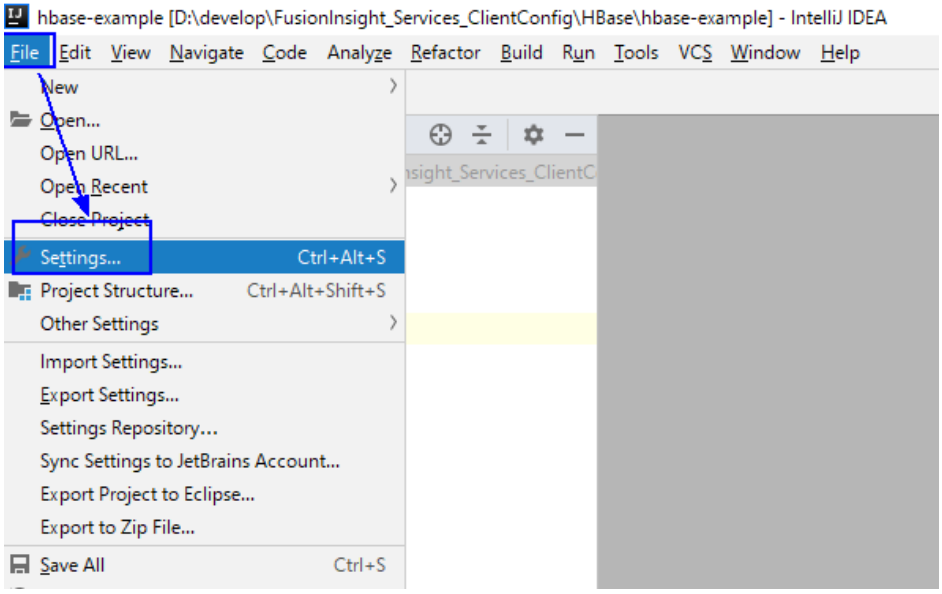
Figure 13-14 Selecting the local Maven installation directory



Step 6 Set the IntelliJ IDEA text file coding format to prevent garbled characters.

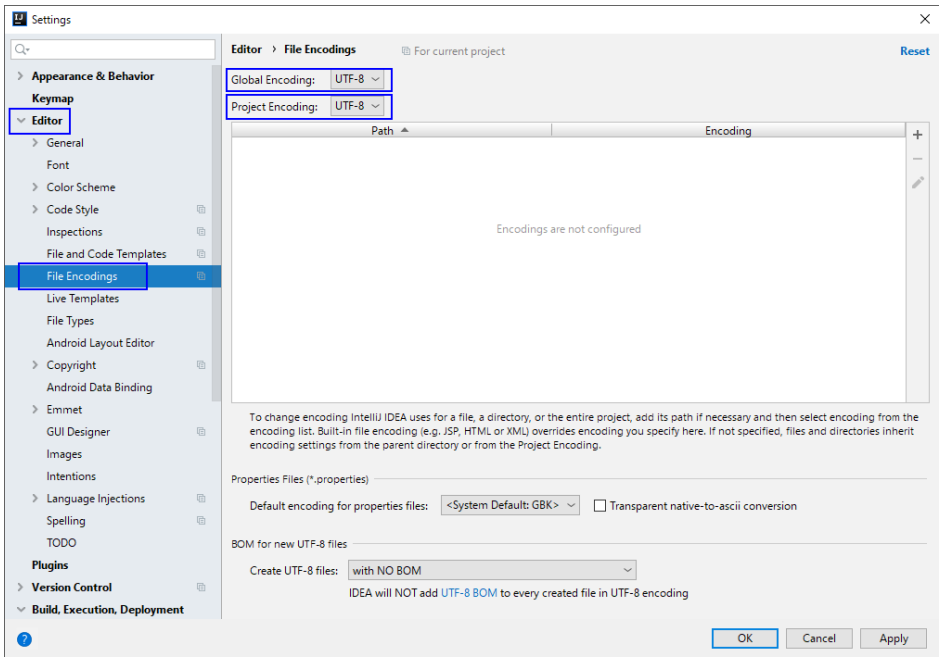
1. On the IntelliJ IDEA menu bar, choose **File** > **Settings...**.

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 HBase Data Read/Write Sample Program

13.3.1.1 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
12005000201	Zhang San	Male	19	Shenzhen, Guangdong
12005000202	Li Wanting	Female	23	Shijiazhuang, Hebei
12005000203	Wang Ming	Male	26	Ningbo, Zhejiang
12005000204	Li Gang	Male	18	Xiangyang, Hubei
12005000205	Zhao Enru	Female	21	Shangrao, Jiangxi
12005000206	Chen Long	Male	32	Zhuzhou, Hunan
12005000207	Zhou Wei	Female	29	Nanyang, Henan
12005000208	Yang Yiwen	Female	30	Kaixian, Chongqing
12005000209	Xu Bing	Male	26	Weinan, Shaanxi
12005000210	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.

CAUTION

HBase tables are stored in *Namespace*:*Table name* format. If no namespace is specified when a table is created, the table is stored in **default**. The **hbase** namespace is the system table namespace. Do not create service tables or read or write data in the system table namespace.

13.3.1.2 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 **Table 13-4**.	For details, see **Creating a Table**.
2	Import user data.	For details, see **Inserting Data**.
3	Add the **Education** column family, and add diplomas and titles to the user information table.	For details, see **Modifying a Table**.
4	Query user names and addresses by user ID.	For details, see **Reading Data Using Get**.
5	Query information by user name.	For details, see **Filtering Data**.
6	To improve query performance, create or delete secondary indexes.	For details, see **Creating a Secondary Index** and **Secondary Index-based Query**.
7	Deregister users, and delete user data.	For details, see **Deleting Data**.
8	Delete the user information table after service A ends.	For details, see **Deleting a Table**.

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.

13.3.1.3 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 s**rc/main/resources/conf** directory.

13.3.1.4 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.

13.3.1.5 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 effeciency,fit for cold data
        // SNAPPY has low compression rate, but high compression and
        // decompression effeciency,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);
```

13.3.1.6 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**.

13.3.1.7 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.

13.3.1.8 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.

13.3.1.9 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.

13.3.1.10 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.");
  }
```

13.3.1.11 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.

13.3.1.12 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);
```


13.3.1.13 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'
```


"#" 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.

13.3.1.14 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.");
  }
```


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.

13.3.1.15 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.hindex.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)"}
```


13.3.1.16 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();

     // initilize 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.

13.3.1.17 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.examples**package.

```
/**
 * 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.");
}
```

13.3.1.18 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.");
}
```

13.3.1.19 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 PrepareStatement
    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.");
}
```

13.3.1.20 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.

Add the Active/Standby Cluster Configuration to the hbase-dual.xml File

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>
```

Step 4 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);
  }
```

Step 5 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
```

----End

Configure the Active/Standby Cluster Configuration in HBaseMultiClusterConnection

Step 1 Create a dual-read Configuration and delete the comment of **testHBaseDualReadSample** from the **main** method of the **TestMain** class in the **com.huawei.bigdata.hbase.examples** package. Ensure that the value of **IS_CREATE_CONNECTION_BY_XML** in the **HBaseDualReadSample** class in the **com.huawei.bigdata.hbase.examples package** is **false**.

Step 2 Add related configurations to the **addHbaseDualXmlParam** method of the **HBaseDualReadSample** class. For details about related configuration items, see **HBase Dual-Read Configuration Items**.

```
private void addHbaseDualXmlParam(Configuration conf) {
    // We need to set the optional parameters contained in hbase-dual.xml to conf
    // when we use configuration transfer solution
    conf.set(CONNECTION_IMPL_KEY, DUAL_READ_CONNECTION);
```



```
    // conf.set("", "");
}
```

Step 3 Add configurations related to the Active cluster client to the **initActiveConf** method of the **HBaseDualReadSample** class.

```
private void initActiveConf() {
    // The hbase-dual.xml configuration scheme is used to generate the client configuration of the active cluster.
    // In actual application development, you need to generate the client configuration of the active cluster.
    String activeDir = HBaseDualReadSample.class.getClassLoader().getResource(Utils.CONF_DIRECTORY).getPath()
        + File.separator + ACTIVE_DIRECTORY + File.separator;
    Configuration activeConf = Utils.createConfByUserDir(activeDir);
    HBaseMultiClusterConnection.setActiveConf(activeConf);
}
```

Step 4 Add configurations related to the Standby cluster client to the **initStandbyConf** method of the **HBaseDualReadSample** class.

```
private void initStandbyConf() {
    // The hbase-dual.xml configuration scheme is used to generate the client configuration of the standby cluster.
    // In actual application development, you need to generate the client configuration of the standby cluster.
    String standbyDir = HBaseDualReadSample.class.getClassLoader().getResource(Utils.CONF_DIRECTORY).getPath()
        + File.separator + STANDBY_DIRECTORY + File.separator;
    Configuration standbyConf = Utils.createConfByUserDir(standbyDir);
    HBaseMultiClusterConnection.setStandbyConf(standbyConf);
}
```

Step 5 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());
  }
  ```

Step 6 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
```

----End

13.3.1.21 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
```

13.3.2 HBase Rest API Invoking Sample Program

13.3.2.1 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 = "100.120.16.170";[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);
}
```

13.3.2.2 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);
}
```

13.3.2.3 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.

CAUTION

HBase tables are stored in *Namespace*:*Table name* format. If no namespace is specified when a table is created, the table is stored in **default**. The **hbase** namespace is the system table namespace. Do not create service tables or read or write data in the system table 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);
}
```

13.3.2.4 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 Accessing the HBase ThriftServer Sample Program

13.3.3.1 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_8.1.2.2/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("100.120.16.170", 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.

- 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);
    }
}
```

13.3.3.2 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.");
}
```


13.3.3.3 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 write 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.");
}
```

13.3.4 Sample Program for HBase to Access Multiple ZooKeepers

13.3.4.1 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
    String confDirPath = TestZKSample.class.getClassLoader().getResource("").getPath() + File.separator;
[1]
    //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 pakcage 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 s**rc/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.

13.4 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.

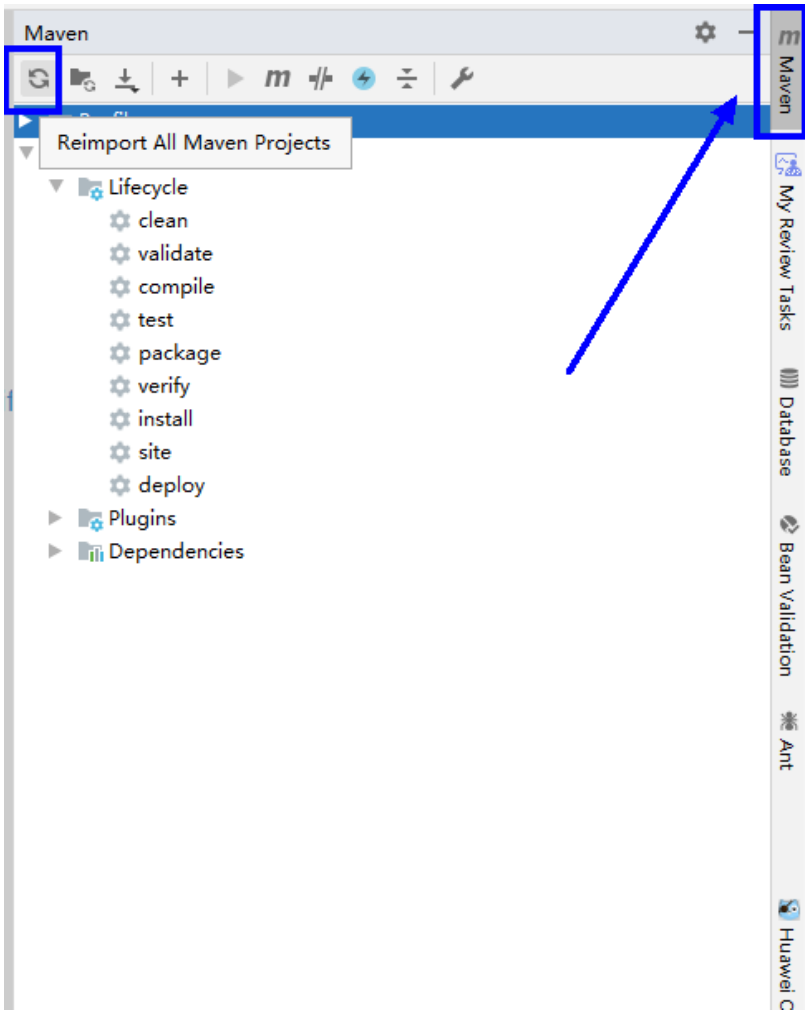
NOTE

- 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.

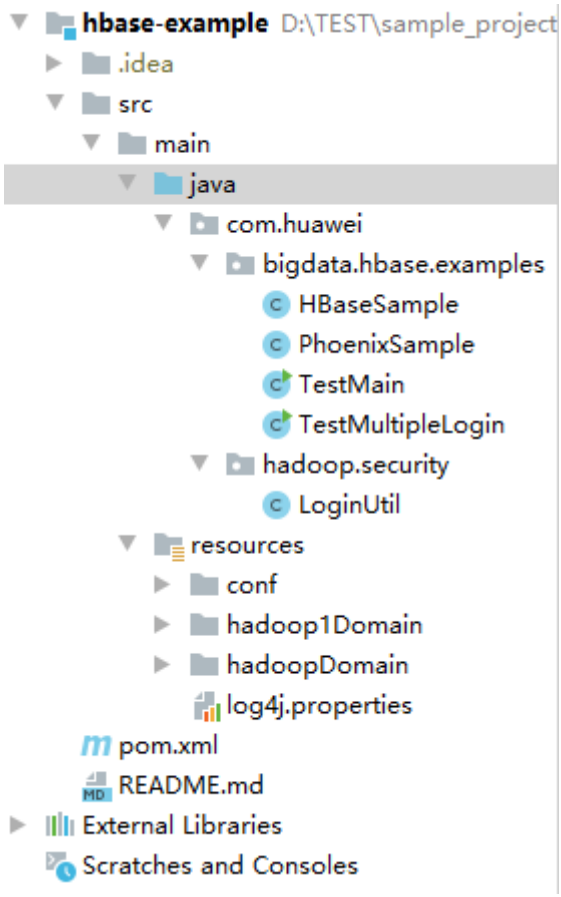
Figure 13-17 reimport projects



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

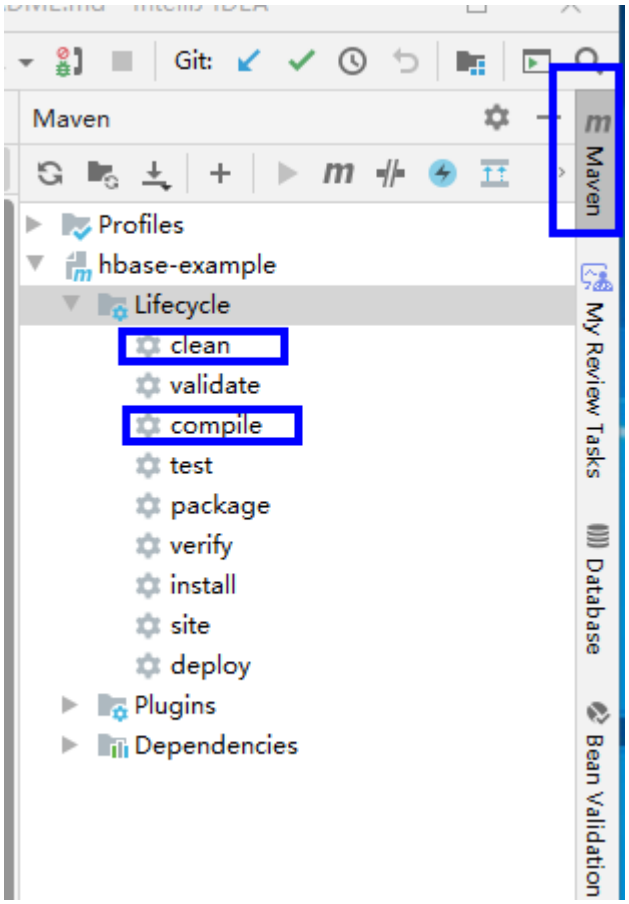


1. Compile an application.
   - Method 1:

     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.

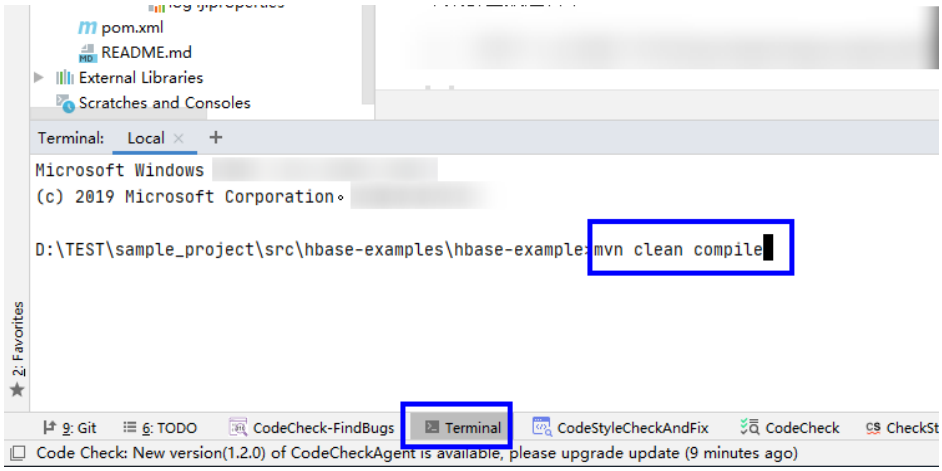
Figure 13-19 clean and **compile** tools of Maven



- Method 2:

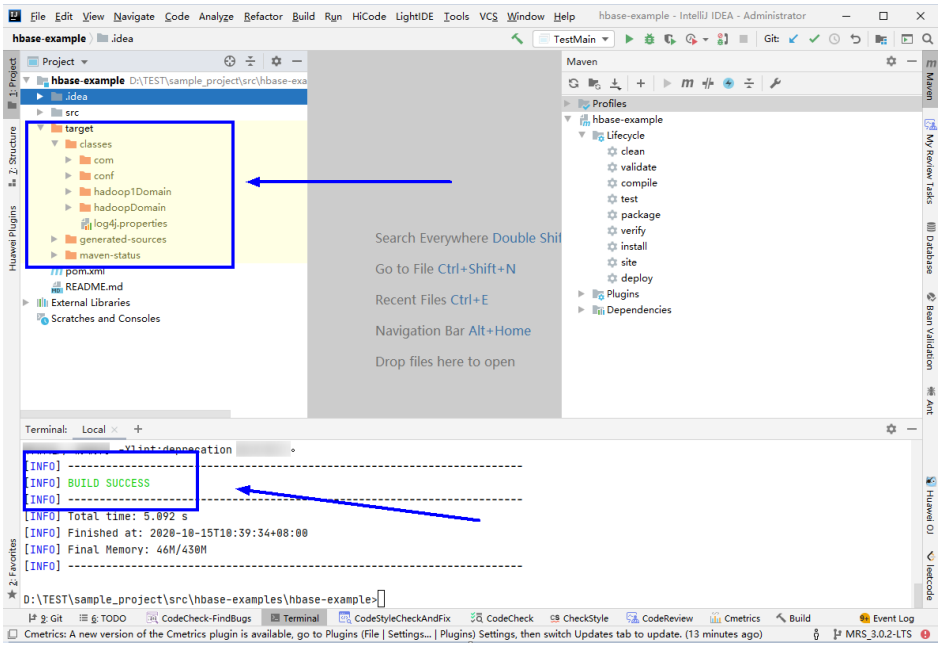
  Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

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.

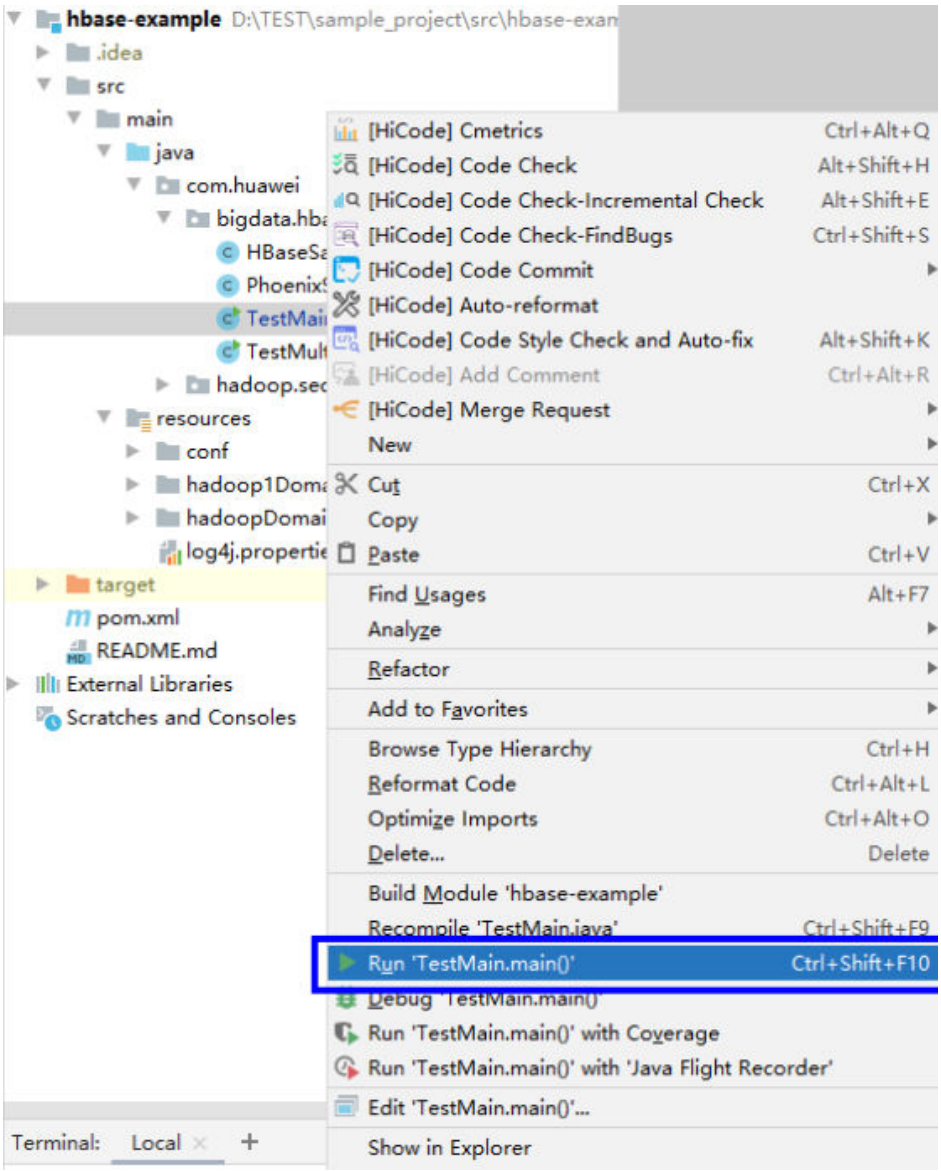
Figure 13-21 Compilation completed



2. Run the program.

   Right-click the **TestMain.java** file and choose **Run'TestMain.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 successful!
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.

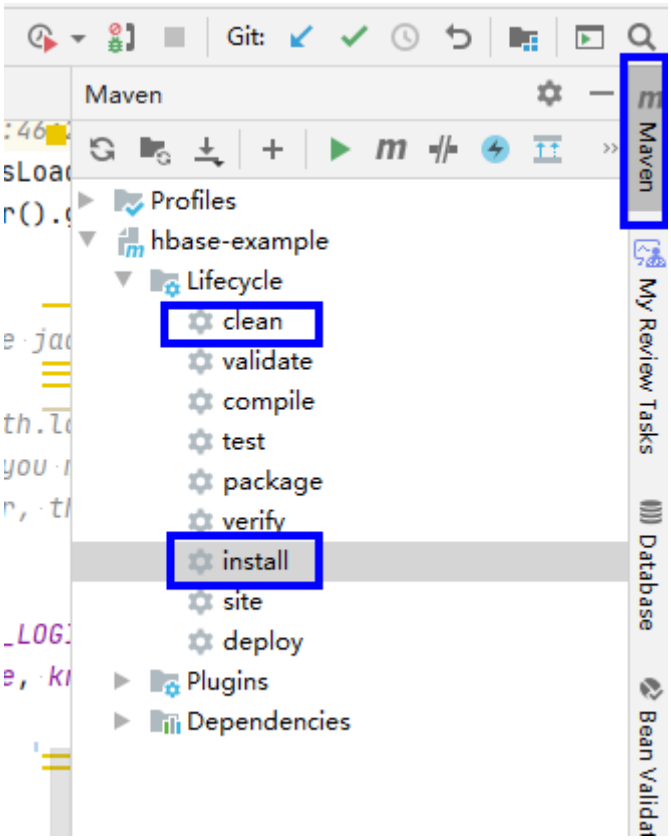
You can build a JAR file in either of the following ways:

- Method 1:

  Choose **Maven** > *Sample projectname* > **Lifecycle** > **clean**, double click **clean** to run the **clean** command of Maven.

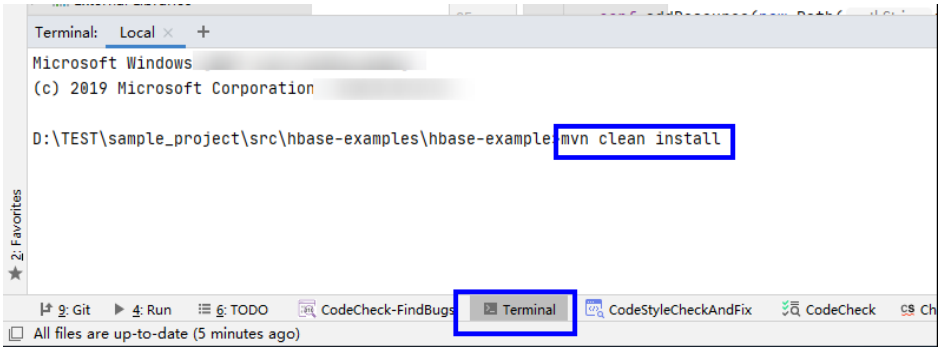
  Choose **Maven** > *Sample project name* > **Lifecycle** > **install**, double click **install** to run the **install** command of Maven.

  Figure 13-23 Maven tools: **clean** and **installl**

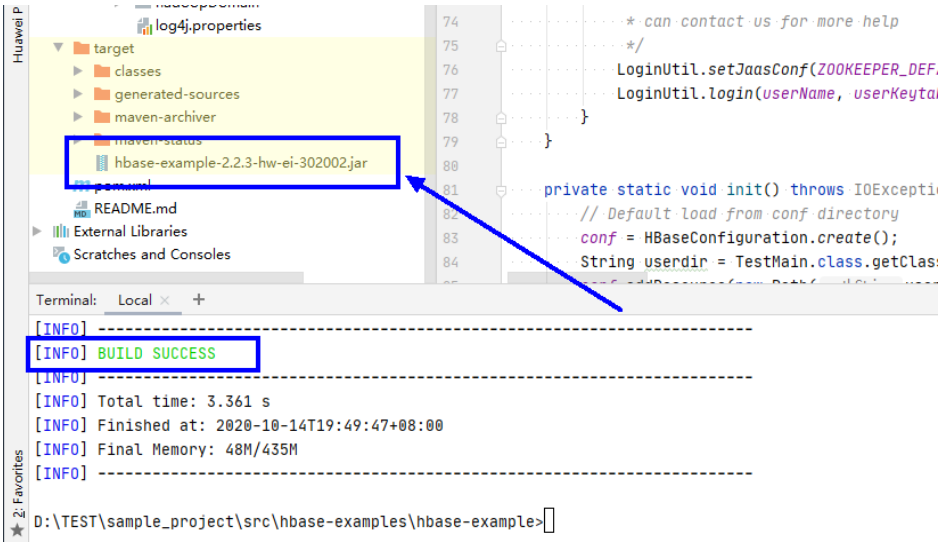
- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the **pom.xml** file.

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

- You can view the execution details of the submitted application in the run logs. For example, after the **hbase-sample** is successfully executed, the following information is displayed:

```
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 PrepareStatement
            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
95thPercentileLaten-cy(ms)	Maximum latency of 95% requests	INFO
99thPercentileLaten-cy(ms)	Maximum latency of 99% requests	INFO
99.9PercentileLaten-cy(ms)	Maximum latency of 99.9% requests	INFO
99.99PercentileLaten-cy(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 **hbase:meta** table, **true** is returned. Otherwise, **false** 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 **true**, 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(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.
boolean isTableDisabled(TableName 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 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 **StartRow**, **StopRow**, and **caching**.

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 interfaces of Phoenix are the same as those of the open-source community. For details, see **http://phoenix.apache.org/**.

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.

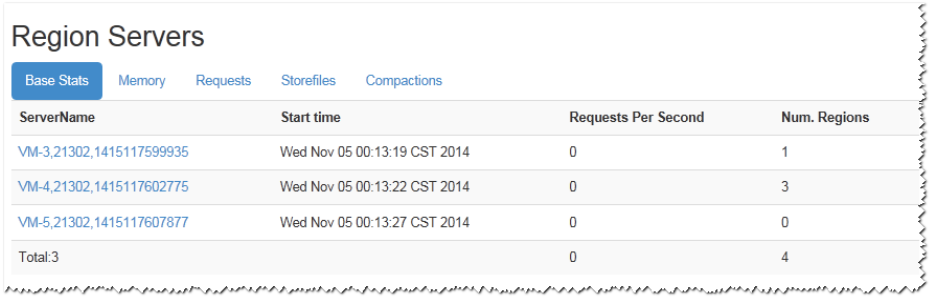
Procedure

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.

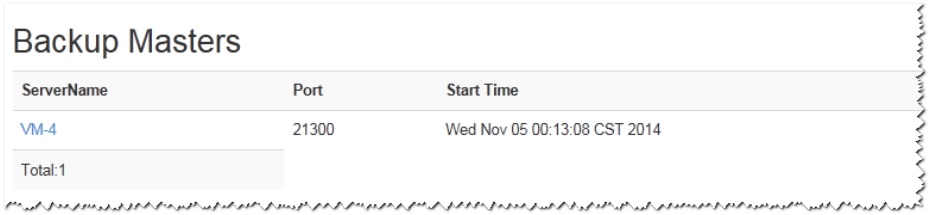
   a. On the **Region Servers** page, view basic information about the RegionServer, as shown in **Figure 13-26**.

   Figure 13-26 Region Servers



   b. On the **Backup Masters** page, view information about the Backup Master, as shown in **Figure 13-27**.

   Figure 13-27 Backup Masters

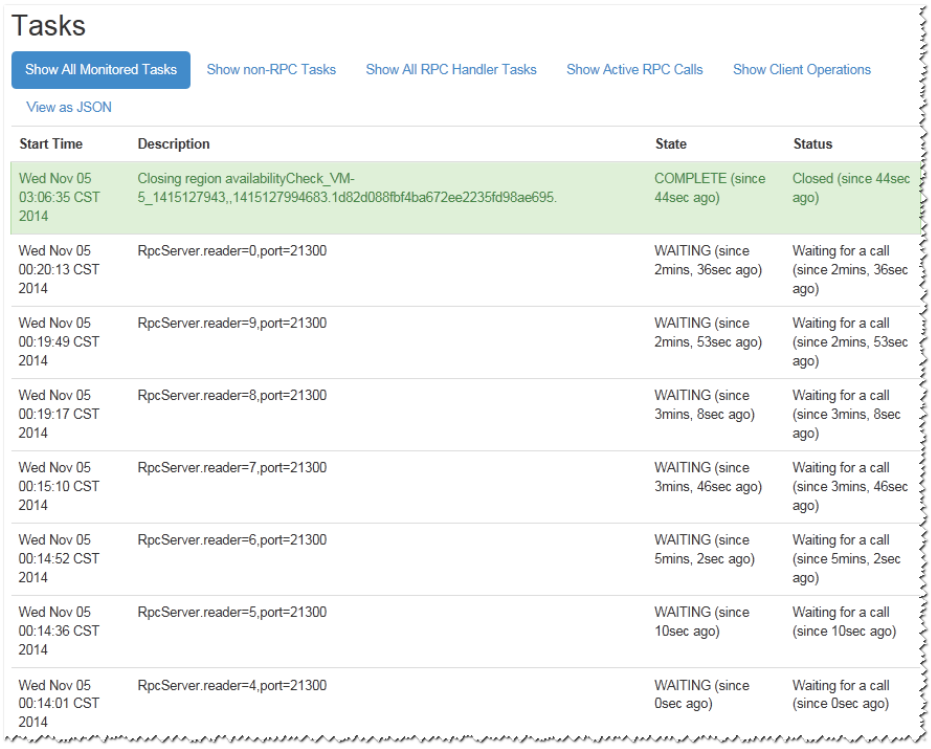


   c. On the **Tables** page, view HBase table information, including **User Tables**, **Catalog Tables**, and **Snapshots**, as shown in **Figure 13-28**.

   Figure 13-28 Tables

   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



3. On the **Table Details** page of the HBase WebUI, view the summary of HBase storage tables, as shown in **Figure 13-30**.

Figure 13-30 Table Details

4. On the **Debug dump** page of the HBase WebUI, view the HBase debug information, as shown in **Figure 13-31**.

Figure 13-31 Debug dump

5. On the **HBase Configuration** page of the HBase WebUI, view the HBase control information, as shown in **Figure 13-32**.

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.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>
```

13.5.4 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. For details, see **Creating a Role** to configure roles with required permissions. For details about how to bind roles with users, see **Creating a User**. 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

13.5.5 FAQs

13.5.5.1 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-5.0.0-HBase-2.0-hw-ei.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

13.5.5.2 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, theprocess 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.

13.5.5.3 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 interface, 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

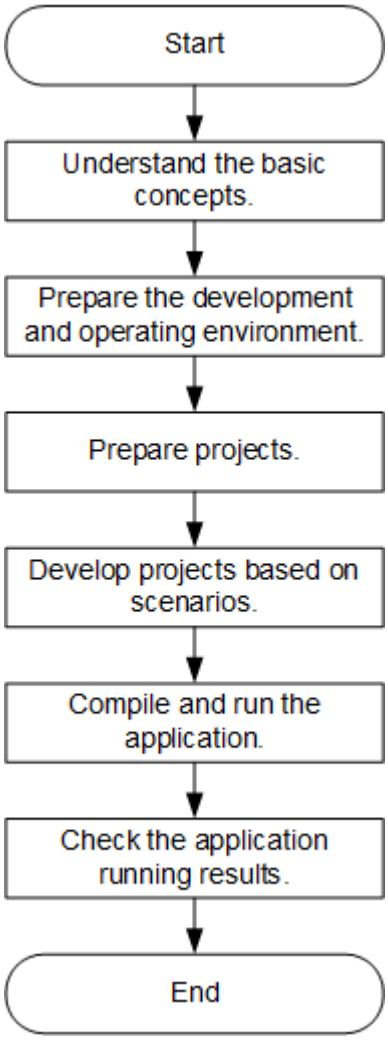


Table 14-1 Description of HDFS development process

Stage	Description	Reference
Understand the basic concepts.	Before the application development, the basic concepts of HDFS are required to be understood.	**Basic Concepts**

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.	**Preparing Development and Operating Environment**
Prepare projects.	HDFS provides sample projects in various scenarios. Sample projects can be imported for studying.	**Configuring and Importing Sample Projects**
Prepare for security authentication.	If a safe cluster is used, the safety certification must be performed.	**Preparing the Authentication Mechanism**
Develop projects based on scenarios.	Provide the sample project. This helps users to learn about the programming interfaces of all HDFS components quickly.	**Developing the Project**
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	**Commissioning the Application**
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.	**Commissioning the Application**

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	• 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	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. • For x86 nodes that run clients, the following JDKs can be used: – 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: – OpenJDK: 1.8.0_272 **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 com.ibm.jsse2.overrideDefaultTLS 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.**
IntelliJ IDEA installation and configuration	It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version. **NOTE** • 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.
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
User development preparation	See **Preparing the Developer Account** for configuration.

Preparation	Description
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** (For MRS 3.3.0 or later, click **Download Client** in the upper right corner of the **Homepage**). 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 the 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/Bigdata/client/conf**

   The keytab file obtained during the **Preparing the 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.

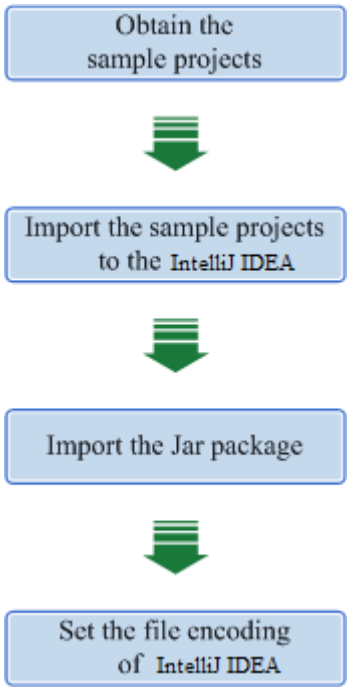
14.2.2 Configuring and Importing Sample Projects

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

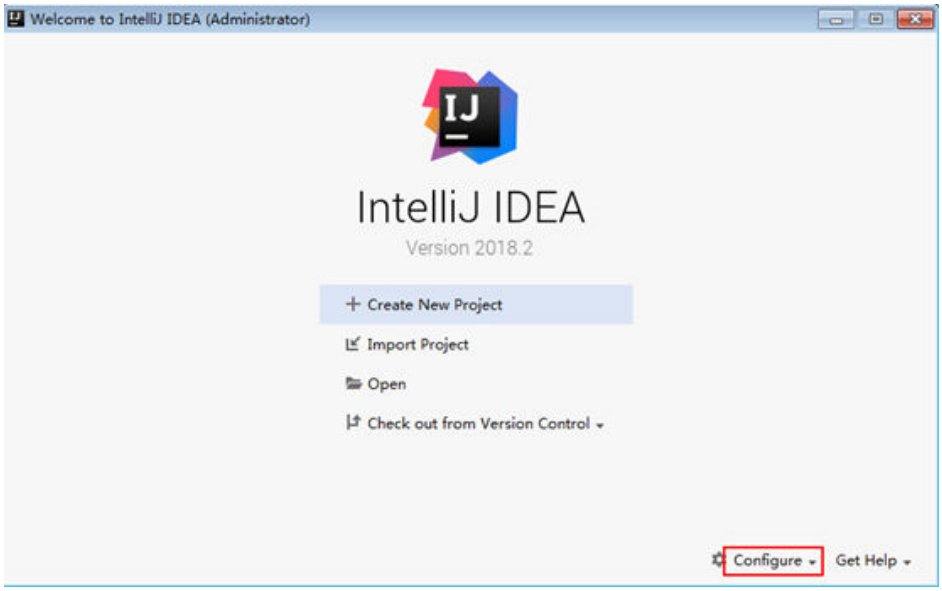
Step 1 Obtain the sample project folder **hdfs-example-security** in the **src** directory where the sample code is decompressed. For details, see **Obtaining Sample Projects from Huawei Mirrors**.

Step 2 Copy the **user.keytab** and **krb5.conf** files obtained in **Preparing the 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.

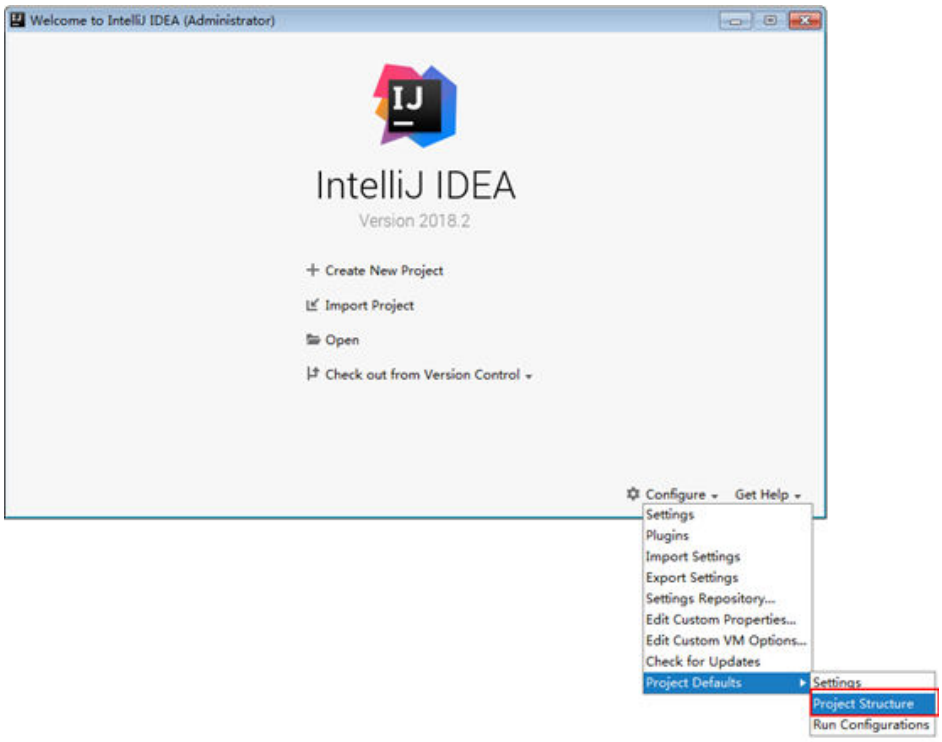
1. Open IntelliJ IDEA and click **Configure**.

Figure 14-3 Quick Start



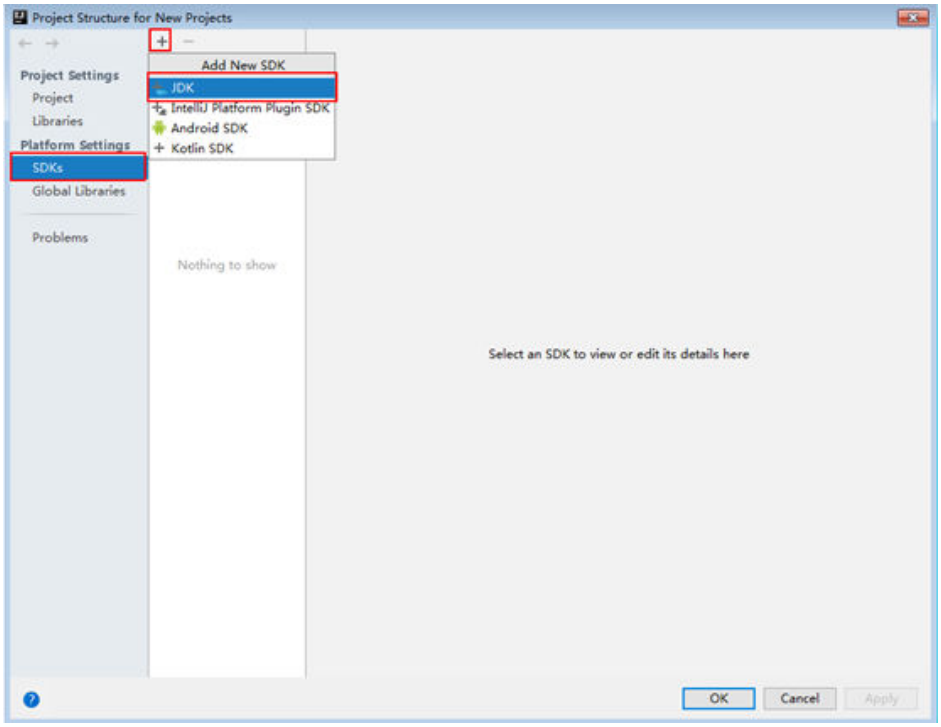
2. Choose **Project Defaults** > **Project Structure**.

Figure 14-4 Configure



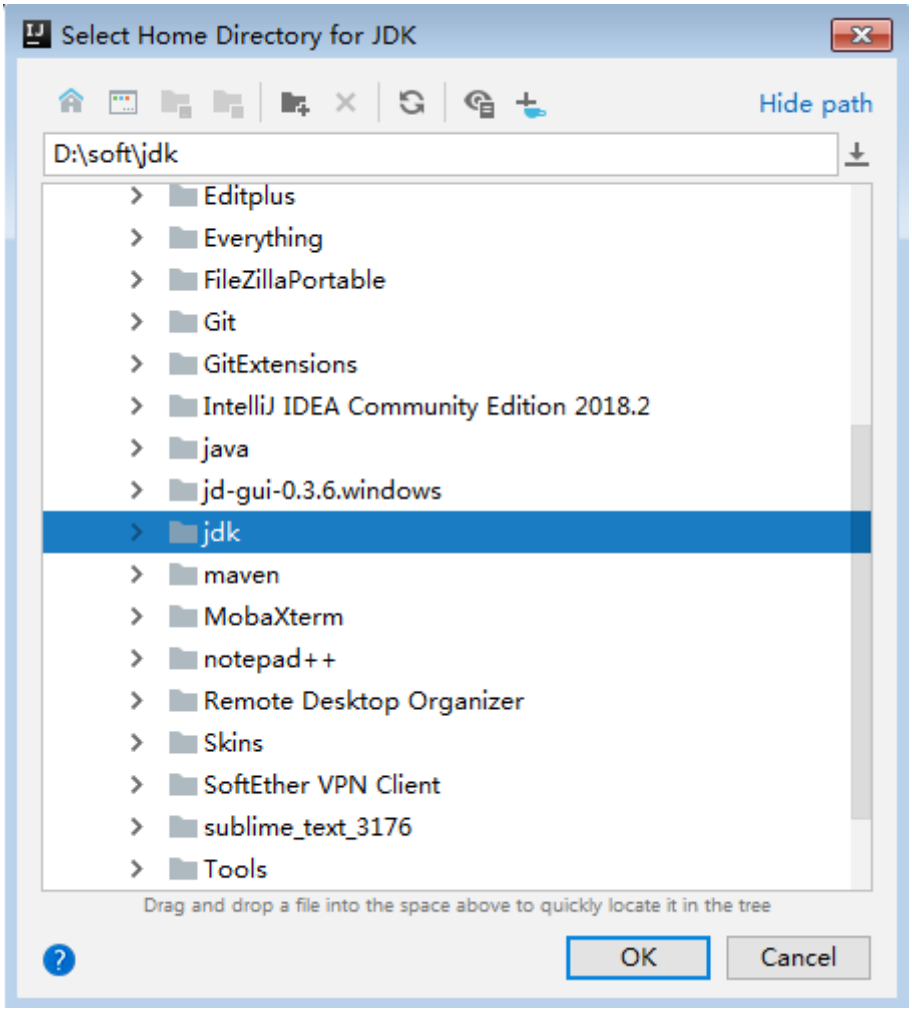
3. In the displayed **Project Structure for New Projects** interface, click **SDKs**. Then click the plus sign (+) and choose **JDK**.

Figure 14-5 Project Structure for New Projects



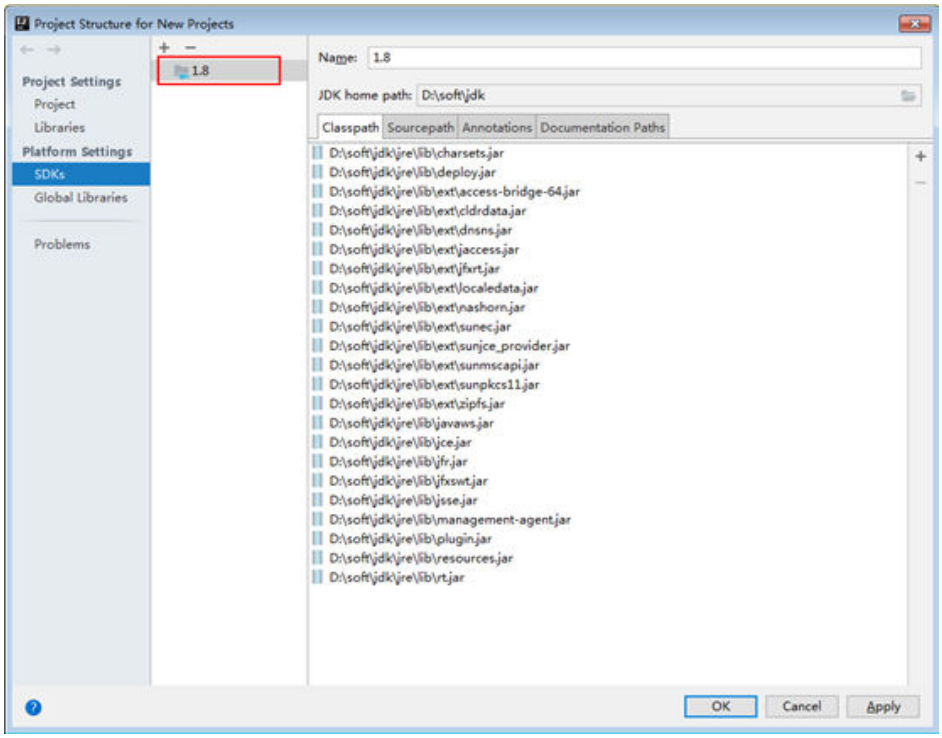
4. In the displayed **Select Home Directory for JDK** interface, select the JDK directory, and click **OK**.

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.

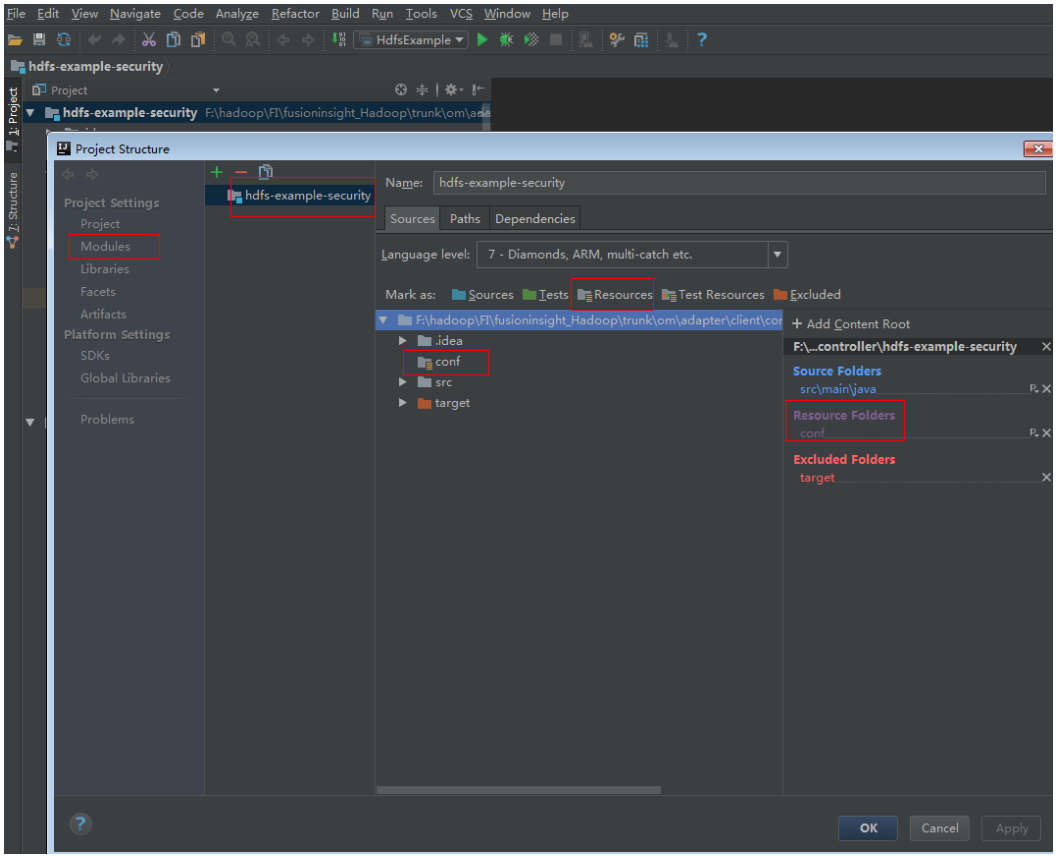
1. Open IntelliJ IDEA and choose **File > Open**.
2. Choose the directory of the example project **hdfs-example-security**. Click **OK**.

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.

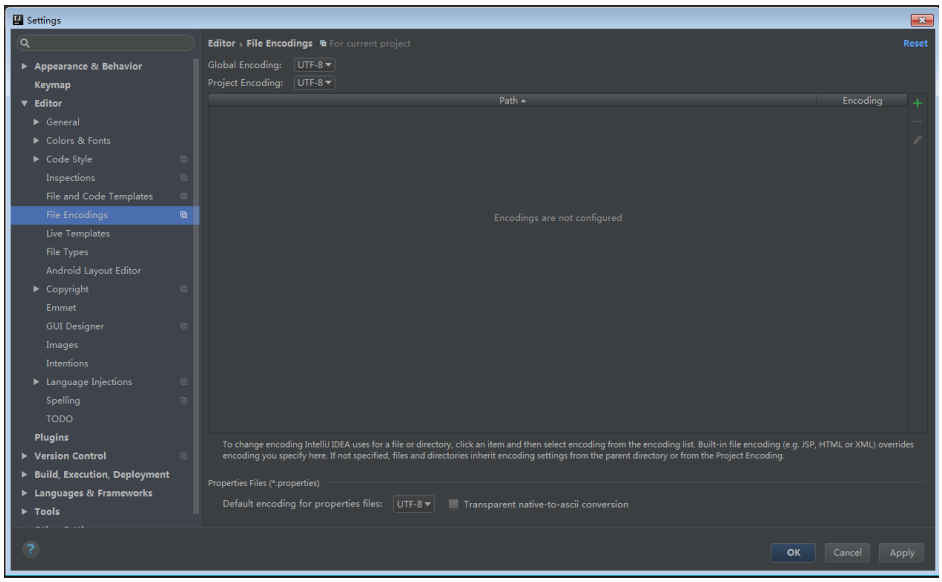
Figure 14-8 Setting the Project Resource Directory



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 PRNCIPAL_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 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
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	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 {
  fSystem = 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 exsit.");
          throw new IOException("userKeytabFile(" + userKeytabFile.getAbsolutePath() + ") does not exsit.");
      }
      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 exsit.");
          throw new IOException("krb5ConfFile(" + krb5ConfFile.getAbsolutePath() + ") does not exsit.");
      }
      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 (!fSystem.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 (!fSystem.exists(filePath)) {
        return false;
    }
    // fSystem.delete(filePath, true);
    return fSystem.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:
  * <li>HOT
  * <li>WARM
  * <li>COLD
  * <li>LAZY_PERSIST
  * <li>ALL_SSD
  * <li>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.	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. 2. Data nodes are allocated to specific locators. Therefore, after capacity expansion, Colocation needs to reallocated data nodes to locators.
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

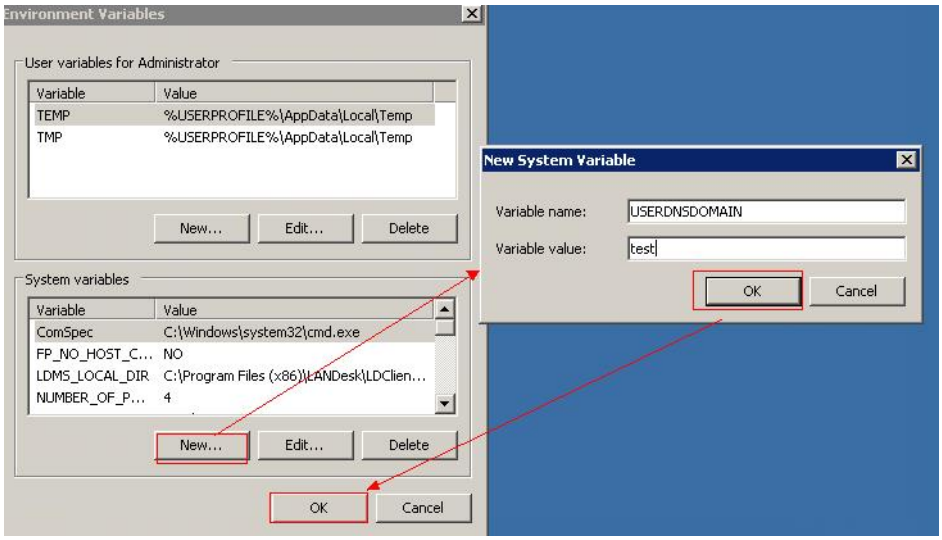
During security authentication, Hadoop needs to obtain the domain name of the host where the client is located (**Default Realm**, which is obtained from environment variable **USERDNSDOMAIN**). If the host does not have a domain name, the following error message is displayed when you run the sample 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 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.

API	Description
public FSDataOutputStream create(Path f)	This API is used to create files in the HDFS. *f* 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. *src* and *dst* indicate complete file paths.
public boolean mkdirs(Path f)	This API is used to create folders in the HDFS. *f* indicates a complete folder path.
public abstract boolean rename(Path src, Path dst)	This API is used to rename a specified HDFS file. *src* and *dst* indicate complete file paths.
public abstract boolean delete(Path f, boolean recursive)	This API is used to delete a specified HDFS file. *f* 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. *f* 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 FsStatus 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. *file* indicates a complete file path, and *start* and *len* 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. *f* 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 *f* 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.
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 *f* 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.

API	Description
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.

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 14-12 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.

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(!hdfsFileIsOpenForWrite(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 contant : %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 contant : %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 contant : %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. 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 contant : Hadoop HDFS Native file write!
hdfsSeek- SUCCESS!, Byte read : 21, File seek contant : S Native file write!
hdfsPread- SUCCESS!, Byte read : 10, File pead contant : 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 contant : 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.

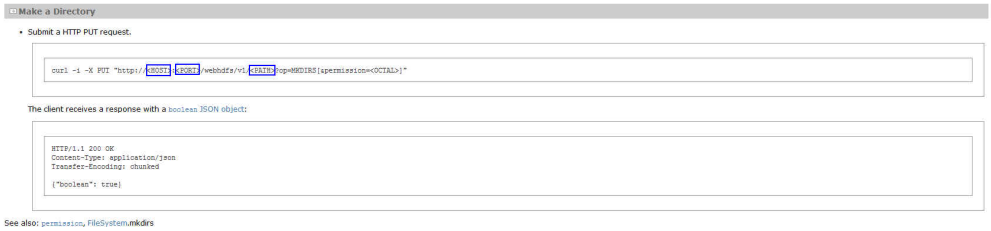
1. Click **Instance**, and in HDFS Instances page, view the host name and IP address of the active NameNode.
2. Click **Configurations**, and in HDFS Service Configuration page, find **namenode.https.port** (9870) and **namenode.https.port** (9871).

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

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 YGoGCSqGSIb3EgECAgIAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKiRARCArhuv39Ttp6lhBlG3B0JAmFjv9weLp+SGFI+t2HSEHN6p4UVWKKy/
kd9dKEgNMlyDu/o7ytzs0cqMxNsI69WbN5H
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 YGoGCSqGSIb3EgECAgIAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKiRARCugB+yT3Y+z8YCRMYJHXF84o1cyCfJq157+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 YGoGCSqGSIb3EgECAgIAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKiRARCzQ6w
+9pNzWCTJEdoU3z9xKEyg1JQNka0nYaB9TndvrL5S0neAoK2usnictTFnqIincAjwB6SnTtht8Q16WDlHJX/
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=HgAFYWRtaW4FYWRtaW4AigFUf4lZdIoBVKOV3XQOCBSyXvFAp92alcRs4j-
KNulnN6wUoBJXRUJIREZTIGRlbGVnYXRpb24UMTAuMTIwLjE3Mi4xMDk6MjUwMDA&namenoderpcadd
ress=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 YGoGCSqGSIb3EgECAgIAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKiRARCZMYR8GGUkn7pPZaoOYZD5HxzLTRZ71angUHKubW2wC/18m9/
OOZstGQ6M1wH2pGriipuCNsKIfwP93eO2Co0fQF3
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=HgAFYWRtaW4FYWRtaW4AigFUfwX3t4oBVKMSe7cCCBSFJTi9j7X64QwnSz59T
GFPKFf7GhNTV0VCSERGUyBkZWxlZ2F0aW9uFDEwLjEyMC4xNzIuMTA5OjI1MDAw&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=HgAFYWRtaW4FYWRtaW4AigFUf4lZdIoBVKOV3XQOCBSyXvFAp92alcRs4j-KNulnN6wUoBJXRUJIREZTIGRlbGVnYXRpb24UMTAuMTIwLjE3Mi4xMDk6MjUwMDA&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
```
- 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=HgAFYWRtaW4FYWRtaW4AigFUfwX3t4oBVKMSe7cCCBSFJTi9j7X64QwnSz59TGFPKFf7GhNTV0VCSERGUyBkZWxlZ2F0aW9uFDEwLjEyMC4xNzIuMTA5OjI1MDAw&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 YGoGCSqGSIb3EgECAgIAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKiRARCTYvNX/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=HgAFYWRtaW4FYWRtaW4AigFUf2mGHooBVKN2Ch4KCBRzjM3jwSMlAowXb
4dhqfKB5rT-8hJXRUJIREZTIGRlbGVnYXRpb24UMTAuMTIwLjE3Mi4xMDk6MjUwMDA&namenoderpcadd
ress=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 YGoGCSqGSIb3EgECAgIAb1swWaADAgEFoQMCAQ
+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_4oBVKNo5v8HCBSE3Fg0f_EwtFKKlODK
QSM2t32CjhNTV0VCSERGUyBkZWxlZ2F0aW9uFDEwLjEyMC4xNzIuMTA5OjI1MDAw&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=HgAFYWRtaW4FYWRtaW4AigFUf2mGHooBVKN2Ch4KCBRzjM3jwSMlAowXb
4dhqfKB5rT-8hJXRUJIREZTIGRlbGVnYXRpb24UMTAuMTIwLjE3Mi4xMDk6MjUwMDA&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_4oBVKNo5v8HCBSE3Fg0f_EwtFKKlODKQSM2t32CjhNTV0VCSERGUyBkZWxlZ2F0aW9uFDEwLjEyMC4xNzIuMTA5OjI1MDAw&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?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":"supergr
```



```
oup","length":70,"modificationTime":1462426678379,"owner":"hdfs","pathSuffix":"testHdfs","permissio
n":"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":"supergr
oup","length":70,"modificationTime":1462426678379,"owner":"hdfs","pathSuffix":"testHdfs","permissio
n":"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://linux1:9870/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 YGoGCSqGSIb3EgECAgIAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKiRARC9k0/v6Ed8VlUBy3kuT0b4RkqkNMCrDevsLGQOUQRORkzWI3Wu
+XLJUMKlmZaWpP+bPzpx8O2Od81mLBgdi8sOkLw
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1462463677153&s=Pwxe5UIqaULjFb9R6ZwlSX85GoI="; 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 YGoGCSqGSIb3EgECAgIAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKiRARCLY5vrVmgsiH2VWRypc30iZGffRUf4nXNaHCWni3TIDUOTl+S+hfjatSbo/+uayQI/
6k9jAfaJrvFIfxqppFtofpp
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 interface has done security hardening to prevent script injection attack. Through REST API interface, 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 14-13 Transparent encryption-related commands

Scenario	Operation	Command	Description
hadoop shell command management key	Create keys.	***hadoop key create*** *<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	***hadoop key roll*** *<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	***hadoop key delete*** *<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	***hadoop key list*** *[-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 14-14 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**.

14.5.3 Access HDFS of the Cluster in Security Mode 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.

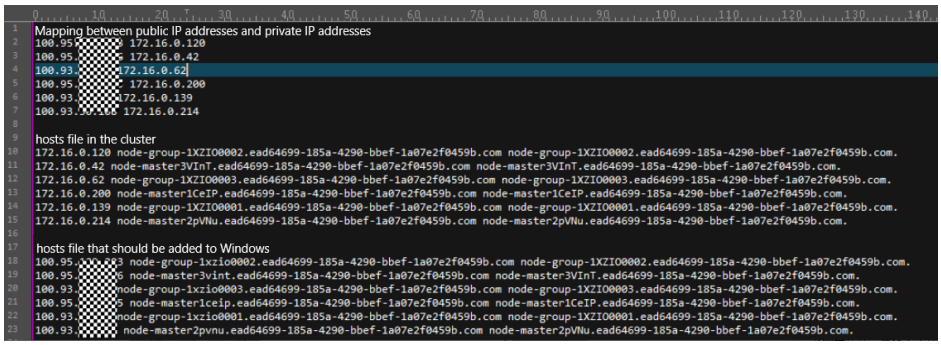
Procedure

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**.

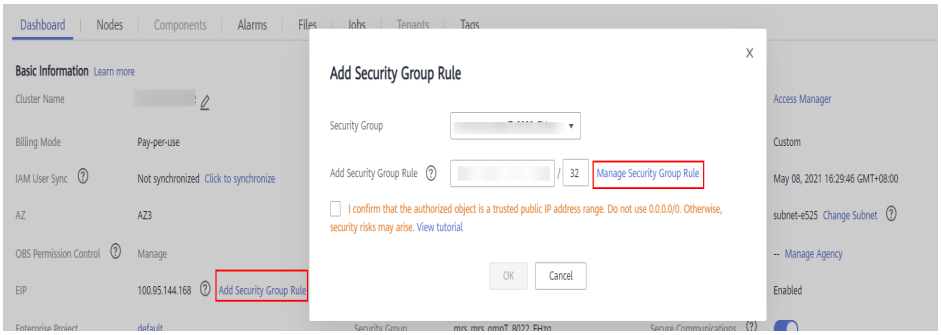
2. Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.



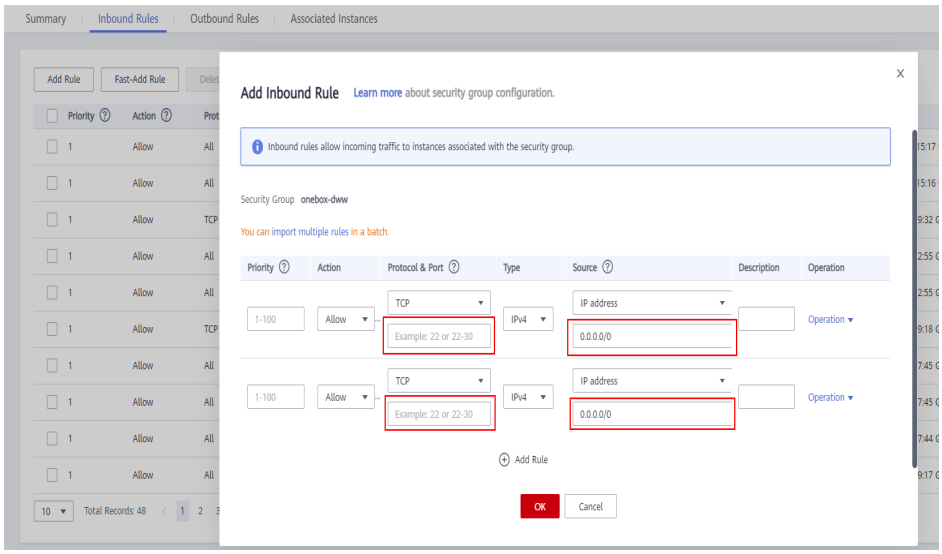
Step 2 Change the IP addresses in the **krb5.conf** file to the corresponding host names.

Step 3 Configure security group rules for the cluster.

1. On the **Dashboard** page, choose **Add Security Group Rule** > **Manage Security Group Rule**.



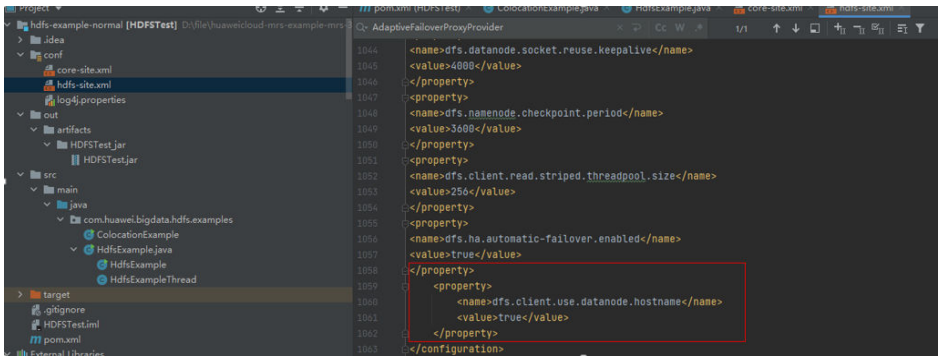
2. On the **Inbound Rules** tab page, click **Add Rule**. In the **Add Inbound Rule** dialog box, configure Windows IP addresses and ports 21730TCP, 21731TCP/UDP, and 21732TCP/UDP.



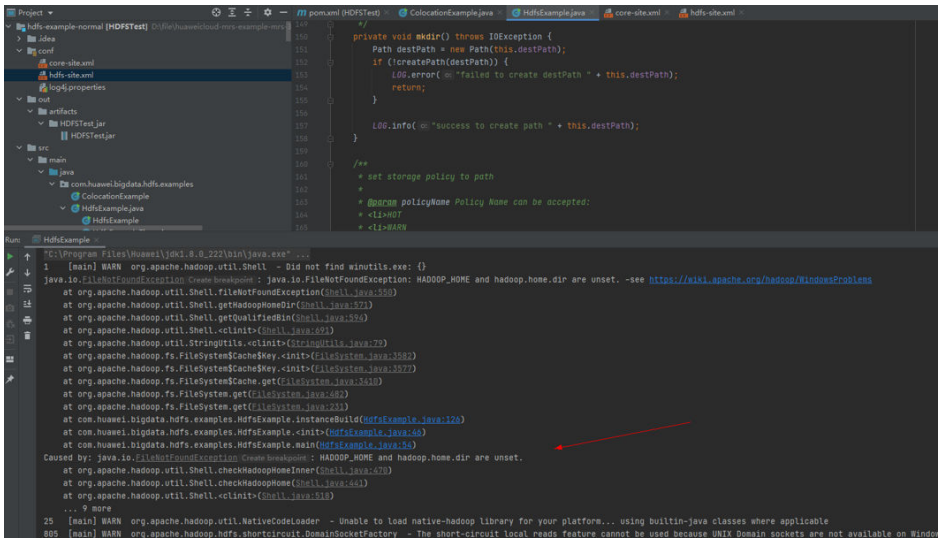
Step 4 On Manager, choose **Cluster** > **Services** > **HDFS** > **More** > **Download Client**, copy the **core-site.xml** and **hdfs-site.xml** files on the client to the **conf** directory of the sample project, and add the following content to the **hdfs-site.xml** file.

```
<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 **PRNCIPAL_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 interface, 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

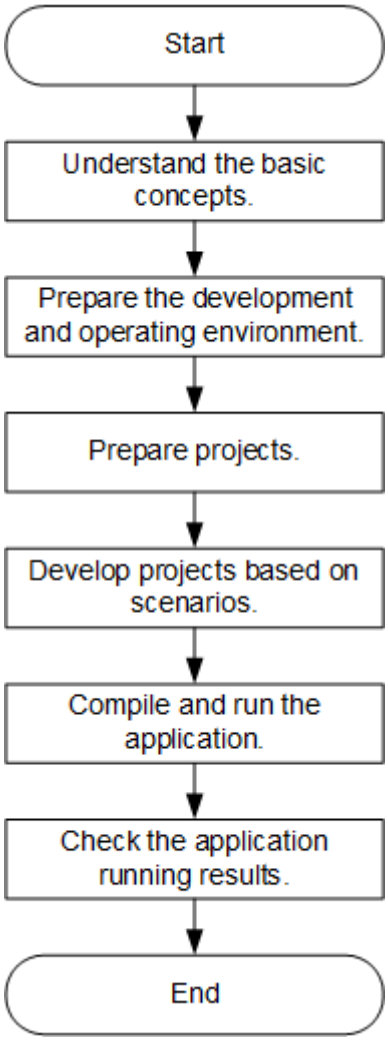


Table 15-1 Description of HDFS development process

Stage	Description	Reference
Understand the basic concepts.	Before the application development, the basic concepts of HDFS are required to be understood.	**Basic Concepts**

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.	**Development and Operating Environment**
Prepare projects.	HDFS provides sample projects in various scenarios. Sample projects can be imported for studying.	**Configuring and Importing Sample Projects**
Develop projects based on scenarios.	Provide the sample project. This helps users to learn about the programming interfaces of all HDFS components quickly.	**Developing the Project**
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	**Commissioning the Application**
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.	**Commissioning the Application**

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	• 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.

Preparation	Description
JDK installation	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. • For x86 nodes that run clients, the following JDKs can be used: – 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: – OpenJDK: 1.8.0_272 **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 com.ibm.jsse2.overrideDefaultTLS 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.**
IntelliJ IDEA installation and configuration	It is the basic configuration for the development environment. IntelliJ IDEA 2019.1 or other compatible versions are used. **NOTE** • 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.
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
7-zip	Used to decompress **.zip** and **.rar** 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** (For MRS 3.3.0 or later, click **Download Client** in the upper right corner of the **Homepage**). 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/Bigdata/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.

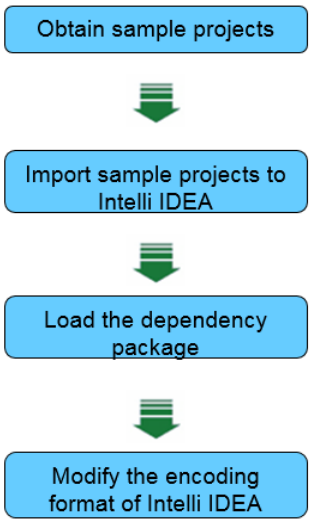
15.2.2 Configuring and Importing Sample Projects

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**.

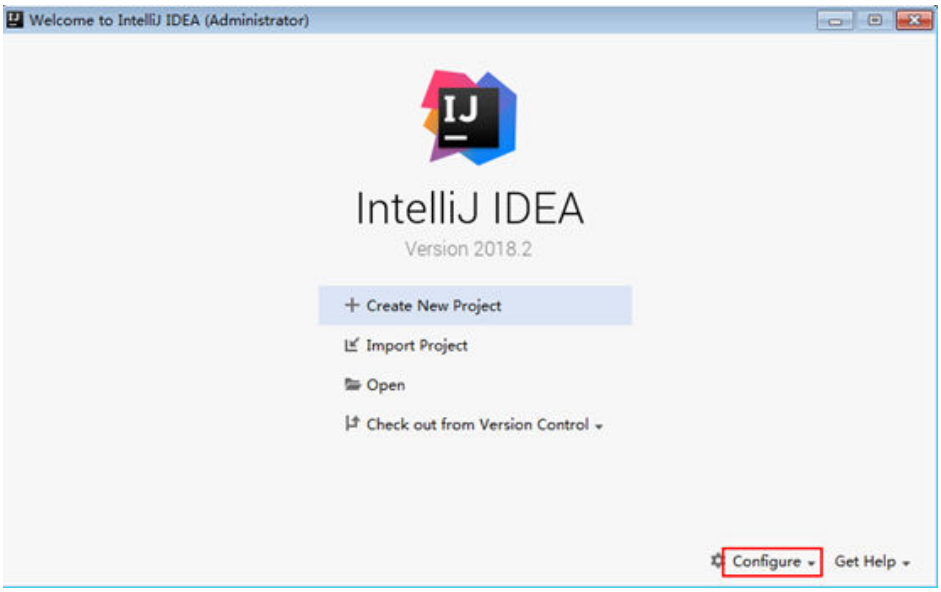
Step 2 Import the example project to the IntelliJ IDEA development environment.

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.

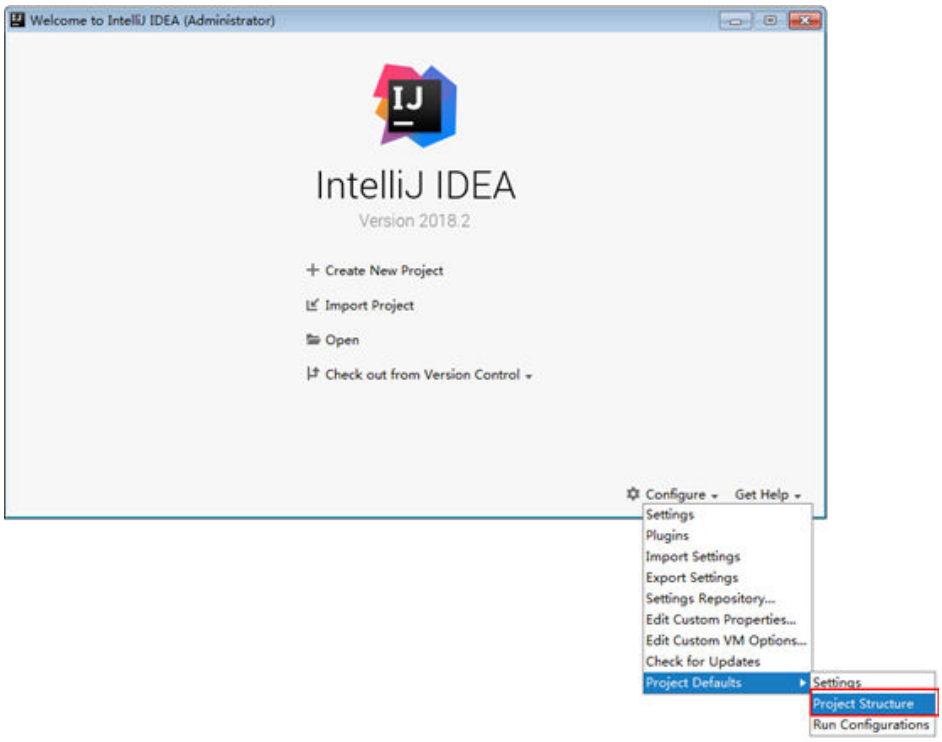
1. Open IntelliJ IDEA and click **Configure**.

Figure 15-3 Quick Start



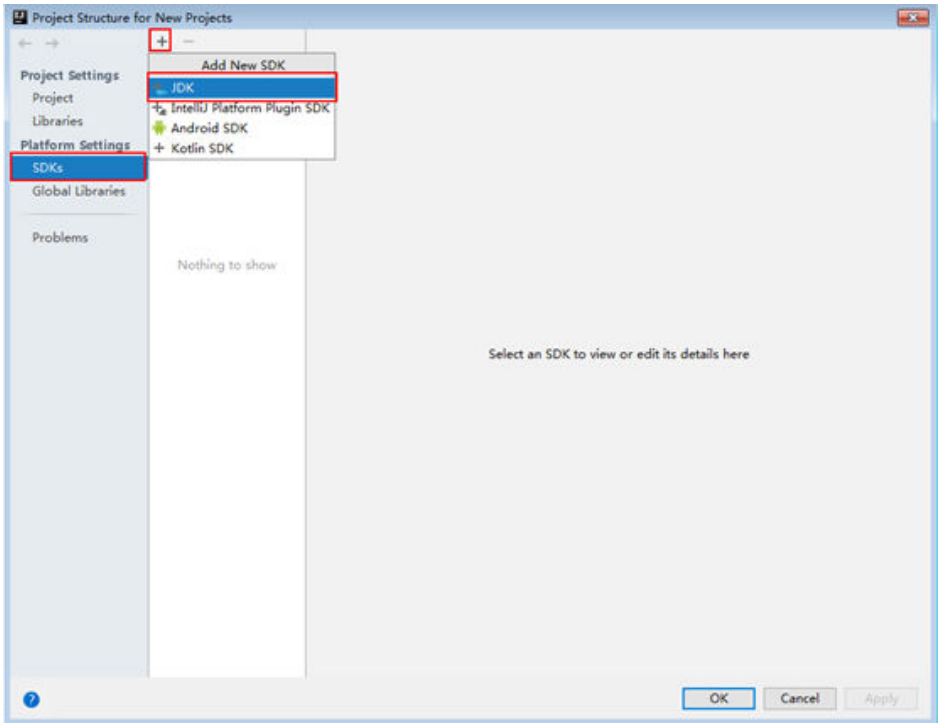
2. Choose **Project Defaults** > **Project Structure**.

Figure 15-4 Configure



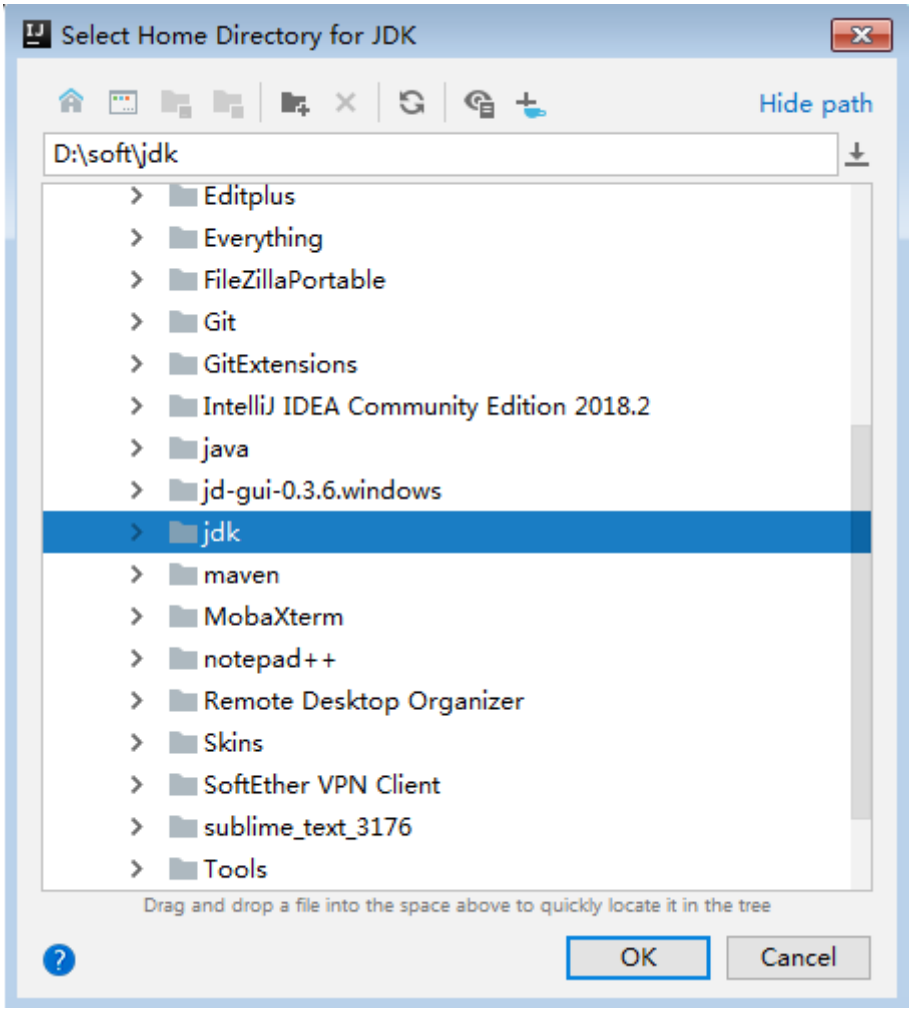
3. In the displayed **Project Structure for New Projects** interface, click **SDKs**. Then click the plus sign (+) and choose **JDK**.

Figure 15-5 Project Structure for New Projects



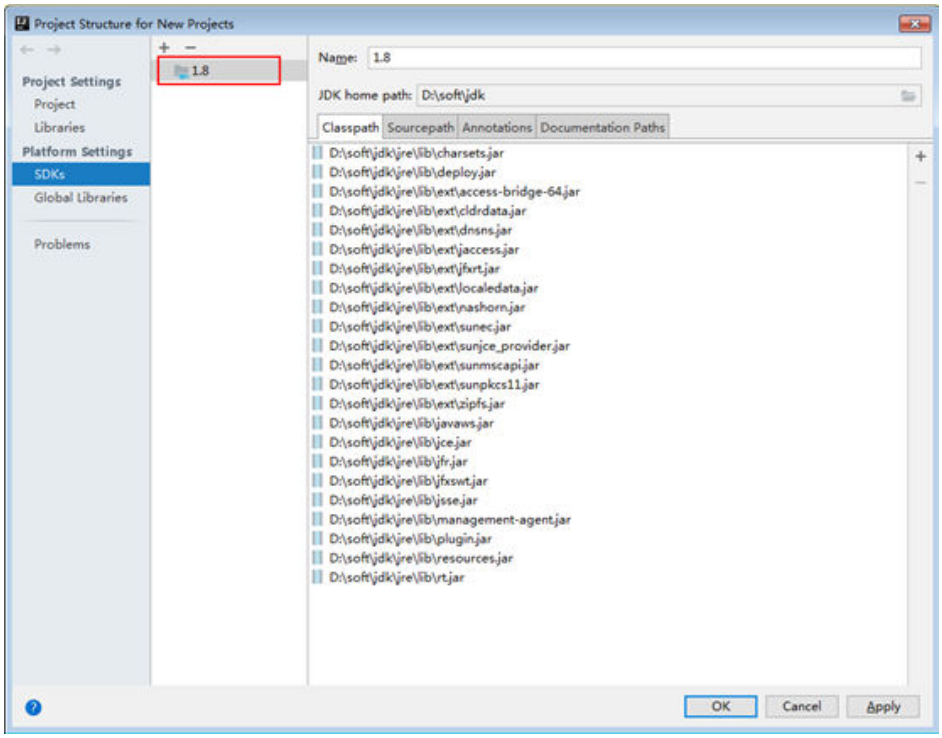
4. In the displayed **Select Home Directory for JDK** interface, select the JDK directory, and click **OK**.

Figure 15-6 Select Home Directory for JDK



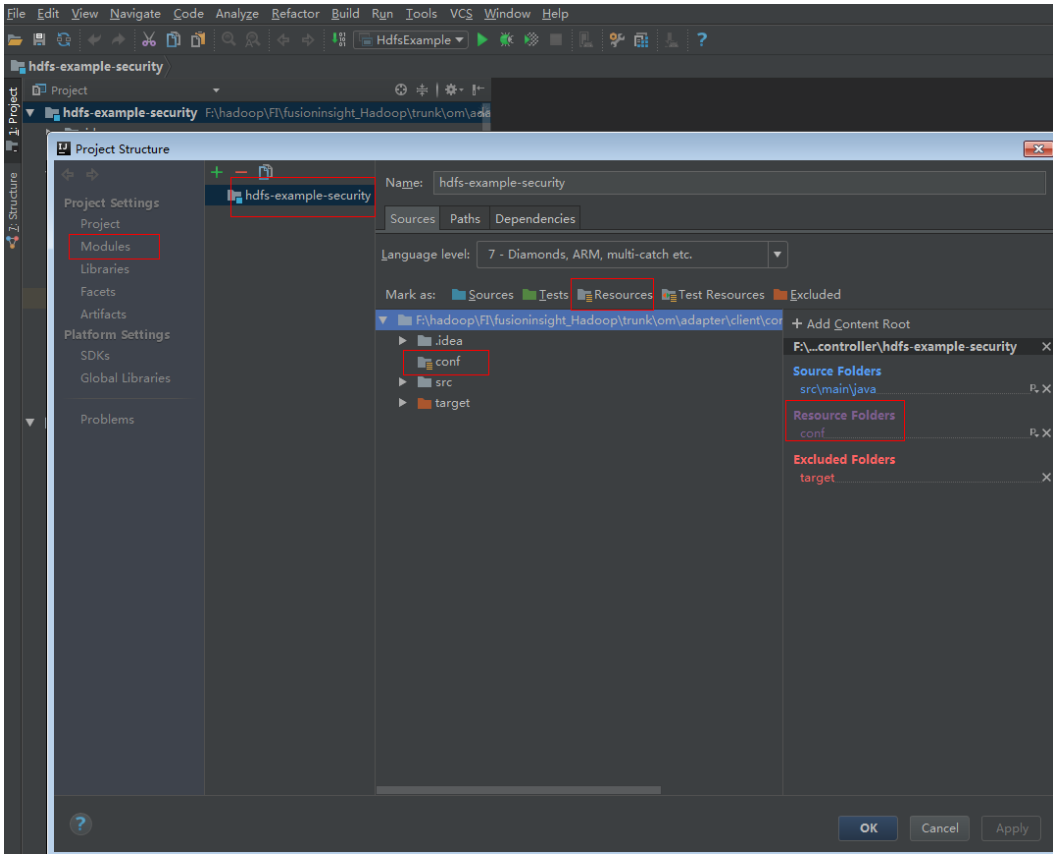
5. Click **OK**.

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



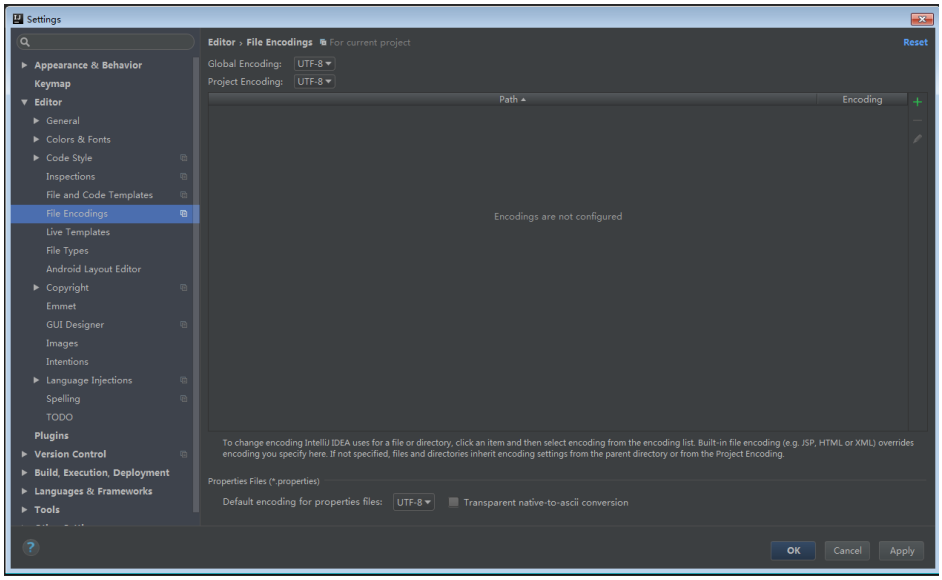
Step 5 Add the related jar files the example project depending on into classpath.

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
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.

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 (!fSystem.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 (!fSystem.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:
  * <li>HOT
  * <li>WARM
  * <li>COLD
  * <li>LAZY_PERSIST
  * <li>ALL_SSD
  * <li>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 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 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.	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. 2. Data nodes are allocated to specific locators. Therefore, after capacity expansion, Colocation needs to reallocated data nodes to locators.
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

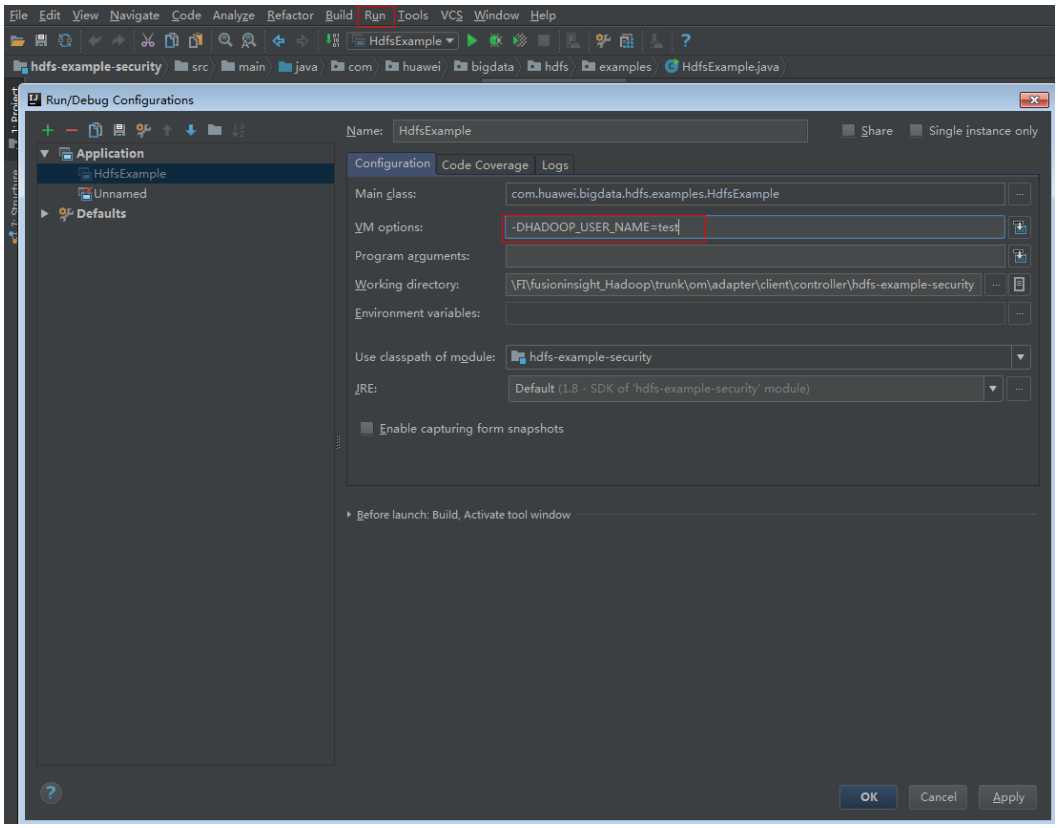
After the code development is complete, you can run an application in the Windows development environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

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=V100R002C30, 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=1.8.0_131
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
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Client
environment:java.class.path=D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\bin;D:\FIClient\nonSafety
\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib
\commons-cli-1.2.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\commons-codec-1.4.jar;D:\FIClient
\nonSafety\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\hdfs-example-normal
\lib\commons-collections-3.2.2.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\commons-
configuration-1.6.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\commons-io-2.4.jar;D:\FIClient
\nonSafety\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\hdfs-example-normal
\lib\commons-lang-2.6.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\commons-logging-1.1.3.jar;D:\FIClient
\nonSafety\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\hdfs-example-normal
\lib\dynalogger-V100R002C30.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\guava-11.0.2.jar;D:\FIClient\nonSafety
\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib
\hadoop-annotations-3.1.1.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\hadoop-auth-3.1.1.jar;D:\FIClient
\nonSafety\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\hdfs-example-normal
\lib\hadoop-common-3.1.1.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\hadoop-hdfs-3.1.1.jar;D:\FIClient
\nonSafety\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\hdfs-example-normal
\lib\hadoop-hdfs-client-3.1.1.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\hadoop-hdfs-
colocation-3.1.1.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\hadoop-hdfs-
datamovement-3.1.1.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\hadoop-hdfs-nfs-3.1.1.jar;D:\FIClient
\nonSafety\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\hdfs-example-normal
\lib\hadoop-hdfs-restore-3.1.1.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\hadoop-mapreduce-client-
core-3.1.1.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig
\HDFS\hdfs-example-normal\lib\hadoop-nfs-3.1.1.jar;D:\FIClient\nonSafety
\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib
\htrace-core-3.1.0-incubating.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\log4j-1.2.17.jar;D:\FIClient\nonSafety
\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib
\protobuf-java-2.5.0.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\slf4j-api-1.7.10.jar;D:\FIClient
\nonSafety\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\hdfs-example-normal
\lib\slf4j-log4j12-1.7.10.jar;D:\FIClient\nonSafety\FusionInsight_Cluster_<Cluster
ID>_Services_ClientConfig\HDFS\hdfs-example-normal\lib\zookeeper-3.5.1.jar;D:\FIClient
\nonSafety\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\SmallFS\FusionInsight-
SmallFS-1.0.0.tar.gz\smallfs\share\datasight\smallfs\smallfs-main-V100R002C30.jar
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Client
environment:java.library.path=D:\Program Files\Java\jre1.8.0_131\bin;C:\Windows\Sun\Java
\bin;C:\Windows\system32;C:\Windows;D:/Program Files/Java/jre1.8.0_131/bin/server;D:/Program
Files/Java/jre1.8.0_131/bin;D:/Program Files/Java/jre1.8.0_131/lib/amd64;C:\ProgramData\Oracle
\Java\javapath;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows
\System32\WindowsPowerShell\v1.0\;D:\Program Files\Java\jdk1.8.0_131\bin;D:\Program Files
\Java\jdk1.8.0_131\jre\bin;D:\Program Files (x86)\GitExtensions\;D:\Program Files\Git\cmd;D:\soft
\apache-maven-3.2.2\bin;D:\soft\gnubin;D:\soft\protoc-2.5.0-win32;;D:\soft\TMSS;D:\installation
package\eclipse-jee-mars-R-win32-x86_64\eclipse;;.
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Client
environment:java.io.tmpdir=C:\Users\L00430~1\AppData\Local\Temp\
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Client environment:java.compiler=<NA>
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Client environment:os.name=Windows 7
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Client environment:os.arch=amd64
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Client environment:os.version=6.1
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Client environment:user.name=user
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Client environment:user.home=C:\Users
\user
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Client environment:user.dir=D:\FIClient
\nonSafety\FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig\HDFS\hdfs-example-normal
```

```
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Client
environment:os.memory.free=107MB
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Client
environment:os.memory.max=1819MB
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Client
environment:os.memory.total=123MB
1670 [main] INFO  org.apache.zookeeper.ZooKeeper  - Initiating client connection,
connectString=192-168-32-144:2181,192-168-32-67:2181,192-168-33-190:2181
sessionTimeout=45000 watcher=com.huawei.hadoop.oi.colocation.ZooKeeperWatcher@5f9b2141
1794 [main] INFO  org.apache.zookeeper.ClientCnxn  - zookeeper.request.timeout is not
configured. Using default value 120000.
1794 [main] INFO  org.apache.zookeeper.ClientCnxn  - zookeeper.client.bind.port.range is not
configured.
1794 [main] INFO  org.apache.zookeeper.ClientCnxn  - zookeeper.client.bind.address is not
configured.
1794 [main-SendThread(192-168-32-67:2181)] INFO
org.apache.zookeeper.client.FourLetterWordMain  - connecting to 192-168-32-67 2181
1904 [main-SendThread(192-168-32-67:2181)] INFO  org.apache.zookeeper.ClientCnxn  - Got
server principal from the server and it is null
1904 [main-SendThread(192-168-32-67:2181)] INFO  org.apache.zookeeper.ClientCnxn  - Using
server principal zookeeper/192-168-32-67
1904 [main-SendThread(192-168-32-67:2181)] INFO  org.apache.zookeeper.ClientCnxn  -
Opening socket connection to server 192-168-32-67/192.168.32.67:2181. Will not attempt to
authenticate using SASL (unknown error)
1966 [main-SendThread(192-168-32-67:2181)] INFO  org.apache.zookeeper.ClientCnxn  - Socket
connection established, initiating session, client: /192.168.35.189:50954, server:
192-168-32-67/192.168.32.67:2181
2029 [main-SendThread(192-168-32-67:2181)] INFO  org.apache.zookeeper.ClientCnxn  -
Session establishment complete on server 192-168-32-67/192.168.32.67:2181, sessionid =
0x13000074b7e464b7, negotiated timeout = 45000
2169 [main] INFO  com.huawei.hadoop.oi.colocation.ZKUtil  - ZooKeeper colocation znode : /
hadoop/colocationDetails. Will publish colocation details under this znode hierarchy.
Create Group is running...
5212 [main] INFO  org.apache.zookeeper.ZooKeeper  - Initiating client connection,
connectString=192-168-32-144:2181,192-168-32-67:2181,192-168-33-190:2181
sessionTimeout=45000 watcher=com.huawei.hadoop.oi.colocation.ZooKeeperWatcher@2438dcd
5212 [main] INFO  org.apache.zookeeper.ClientCnxn  - zookeeper.request.timeout is not
configured. Using default value 120000.
5212 [main] INFO  org.apache.zookeeper.ClientCnxn  - zookeeper.client.bind.port.range is not
configured.
5212 [main] INFO  org.apache.zookeeper.ClientCnxn  - zookeeper.client.bind.address is not
configured.
5212 [main-SendThread(192-168-33-190:2181)] INFO
org.apache.zookeeper.client.FourLetterWordMain  - connecting to 192-168-33-190 2181
5321 [main-SendThread(192-168-33-190:2181)] INFO  org.apache.zookeeper.ClientCnxn  - Got
server principal from the server and it is null
5321 [main-SendThread(192-168-33-190:2181)] INFO  org.apache.zookeeper.ClientCnxn  -
Using server principal zookeeper/192-168-33-190
5321 [main-SendThread(192-168-33-190:2181)] INFO  org.apache.zookeeper.ClientCnxn  -
Opening socket connection to server 192-168-33-190/192.168.33.190:2181. Will not attempt to
authenticate using SASL (unknown error)
5368 [main-SendThread(192-168-33-190:2181)] INFO  org.apache.zookeeper.ClientCnxn  -
Socket connection established, initiating session, client: /192.168.35.189:50962, server:
192-168-33-190/192.168.33.190:2181
5430 [main-SendThread(192-168-33-190:2181)] INFO  org.apache.zookeeper.ClientCnxn  -
Session establishment complete on server 192-168-33-190/192.168.33.190:2181, sessionid =
0x14000073f13b657b, negotiated timeout = 45000
5540 [main] INFO  com.huawei.hadoop.oi.colocation.ZKUtil  - ZooKeeper colocation znode : /
hadoop/colocationDetails. Will publish colocation details under this znode hierarchy.
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
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:java.class.path=/opt/client/
Yarn/config:/opt/client/HDFS/hadoop/etc/hadoop:/opt/client/HDFS/hadoop/share/hadoop/
common/lib/apacheds-i18n-2.0.0-M15.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
commons-math3-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/jackson-
xc-1.9.13.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/jaxb-impl-2.2.3-1.jar:/opt/
client/HDFS/hadoop/share/hadoop/common/lib/commons-io-2.4.jar:/opt/client/HDFS/hadoop/
share/hadoop/common/lib/commons-digester-1.8.jar:/opt/client/HDFS/hadoop/share/hadoop/
common/lib/jetty-6.1.26.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/jackson-core-
asl-1.9.13.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/api-asn1-api-1.0.0-
M20.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/avro-1.7.4.jar:/opt/client/HDFS/
hadoop/share/hadoop/common/lib/jsr305-3.0.0.jar:/opt/client/HDFS/hadoop/share/hadoop/
common/lib/crypter-0.0.6.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/dynalogger-
V100R002C30.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/jersey-server-1.9.jar:/opt/
client/HDFS/hadoop/share/hadoop/common/lib/commons-codec-1.4.jar:/opt/client/HDFS/
hadoop/share/hadoop/common/lib/curator-client-2.7.1.jar:/opt/client/HDFS/hadoop/share/
hadoop/common/lib/jaxb-api-2.2.2.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
nimbus-jose-jwt-3.9.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/json-
smart-1.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/zookeeper-3.5.1.jar:/opt/
client/HDFS/hadoop/share/hadoop/common/lib/cas-client-core-hw-3.3.3.jar:/opt/client/HDFS/
hadoop/share/hadoop/common/lib/opensaml-2.6.5.jar:/opt/client/HDFS/hadoop/share/hadoop/
common/lib/htrace-core-3.1.0-incubating.jar:/opt/client/HDFS/hadoop/share/hadoop/
common/lib/wc2frm-v1r2c60-20160429.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
jettison-1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/commons-
collections-3.2.2.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/asm-3.2.jar:/opt/client/
HDFS/hadoop/share/hadoop/common/lib/jetty-util-6.1.26.jar:/opt/client/HDFS/hadoop/share/
hadoop/common/lib/servlet-api-2.5.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
hadoop-auth-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/log4j-1.2.17.jar:/opt/
client/HDFS/hadoop/share/hadoop/common/lib/curator-recipes-2.7.1.jar:/opt/client/HDFS/
hadoop/share/hadoop/common/lib/gson-2.2.4.jar:/opt/client/HDFS/hadoop/share/hadoop/
common/lib/commons-beanutils-1.7.0.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
jets3t-0.9.0.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/mockito-all-1.8.5.jar:/opt/
client/HDFS/hadoop/share/hadoop/common/lib/jersey-json-1.9.jar:/opt/client/HDFS/hadoop/
share/hadoop/common/lib/apache-log4j-extras-1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/
common/lib/jackson-mapper-asl-1.9.13.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
protobuf-java-2.5.0.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/hadoop-crypto-
adapter-0.0.1.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/api-util-1.0.0-M20.jar:/opt/
client/HDFS/hadoop/share/hadoop/common/lib/jsch-0.1.54.jar:/opt/client/HDFS/hadoop/share/
hadoop/common/lib/hamcrest-core-1.3.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
commons-logging-1.1.3.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/java-
xmlbuilder-0.4.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/apacheds-kerberos-
```


codec-2.0.0-M15.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
netty-3.6.2.Final.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/hadoop-
annotations-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/commons-
cli-1.2.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/activation-1.1.jar:/opt/client/
HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar:/opt/client/HDFS/hadoop/
share/hadoop/common/lib/junit-4.11.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
jsp-api-2.1.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/httpcore-4.4.4.jar:/opt/client/
HDFS/hadoop/share/hadoop/common/lib/httpclient-4.5.2.jar:/opt/client/HDFS/hadoop/share/
hadoop/common/lib/commons-beanutils-core-1.8.0.jar:/opt/client/HDFS/hadoop/share/hadoop/
common/lib/xz-1.0.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/jcip-
annotations-1.0.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/paranamer-2.3.jar:/opt/
client/HDFS/hadoop/share/hadoop/common/lib/slf4j-api-1.7.10.jar:/opt/client/HDFS/hadoop/
share/hadoop/common/lib/curator-framework-2.7.1.jar:/opt/client/HDFS/hadoop/share/hadoop/
common/lib/commons-httpclient-3.1.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
stax-api-1.0-2.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/commons-
compress-1.4.1.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/xmlenc-0.52.jar:/opt/
client/HDFS/hadoop/share/hadoop/common/lib/commons-lang-2.6.jar:/opt/client/HDFS/hadoop/
share/hadoop/common/lib/jersey-core-1.9.jar:/opt/client/HDFS/hadoop/share/hadoop/
common/lib/commons-net-3.1.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
xmlsec-1.5.7.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/jackson-
jaxrs-1.9.13.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/jetty-
sslengine-6.1.26.jar:/opt/client/HDFS/hadoop/share/hadoop/common/lib/guava-11.0.2.jar:/opt/
client/HDFS/hadoop/share/hadoop/common/lib/snappy-java-1.0.4.1.jar:/opt/client/HDFS/
hadoop/share/hadoop/common/lib/SSO-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/
common/lib/commons-configuration-1.6.jar:/opt/client/HDFS/hadoop/share/hadoop/common/
datasight-hadoop-trace-ping-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/common/hadoop-
common-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/common/hadoop-nfs-3.1.1.jar:/opt/
client/HDFS/hadoop/share/hadoop/hdfs:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/
commons-io-2.4.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/jetty-6.1.26.jar:/opt/client/
HDFS/hadoop/share/hadoop/hdfs/lib/jackson-core-asl-1.9.13.jar:/opt/client/HDFS/hadoop/share/
hadoop/hdfs/lib/jsr305-3.0.0.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/jersey-
server-1.9.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/commons-codec-1.4.jar:/opt/client/
HDFS/hadoop/share/hadoop/hdfs/lib/leveldbjni-all-1.8.jar:/opt/client/HDFS/hadoop/share/
hadoop/hdfs/lib/javaluator-3.0.1.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/
xercesImpl-2.9.1.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/htrace-core-3.1.0-
incubating.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/asm-3.2.jar:/opt/client/HDFS/
hadoop/share/hadoop/hdfs/lib/jetty-util-6.1.26.jar:/opt/client/HDFS/hadoop/share/hadoop/
hdfs/lib/servlet-api-2.5.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/log4j-1.2.17.jar:/opt/
client/HDFS/hadoop/share/hadoop/hdfs/lib/jackson-mapper-asl-1.9.13.jar:/opt/client/HDFS/
hadoop/share/hadoop/hdfs/lib/protobuf-java-2.5.0.jar:/opt/client/HDFS/hadoop/share/hadoop/
hdfs/lib/commons-logging-1.1.3.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/
netty-3.6.2.Final.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/commons-cli-1.2.jar:/opt/
client/HDFS/hadoop/share/hadoop/hdfs/lib/xml-apis-1.3.04.jar:/opt/client/HDFS/hadoop/share/
hadoop/hdfs/lib/hdfs-inode-provider-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/
netty-all-4.0.23.Final.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/hadoop-hdfs-
client-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/commons-
daemon-1.0.13.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/xmlenc-0.52.jar:/opt/client/
HDFS/hadoop/share/hadoop/hdfs/lib/commons-lang-2.6.jar:/opt/client/HDFS/hadoop/share/
hadoop/hdfs/lib/jersey-core-1.9.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/lib/
guava-11.0.2.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/hadoop-hdfs-
colocation-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/hdfs/hadoop-hdfs-nfs-3.1.1.jar:/opt/
client/HDFS/hadoop/share/hadoop/hdfs/hadoop-hdfs-datamovement-3.1.1.jar:/opt/client/HDFS/
hadoop/share/hadoop/hdfs/hadoop-hdfs-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/
yarn/lib/superior-client-1.0.0.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/spark-3.1.1-
yarn-shuffle.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/jackson-xc-1.9.13.jar:/opt/client/
HDFS/hadoop/share/hadoop/yarn/lib/jaxb-impl-2.2.3-1.jar:/opt/client/HDFS/hadoop/share/
hadoop/yarn/lib/commons-io-2.4.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/
aopalliance-1.0.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/jetty-6.1.26.jar:/opt/client/
HDFS/hadoop/share/hadoop/yarn/lib/jackson-core-asl-1.9.13.jar:/opt/client/HDFS/hadoop/share/
hadoop/yarn/lib/jsr305-3.0.0.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/dynalogger-
V100R002C30.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/jersey-server-1.9.jar:/opt/
client/HDFS/hadoop/share/hadoop/yarn/lib/commons-codec-1.4.jar:/opt/client/HDFS/hadoop/
share/hadoop/yarn/lib/leveldbjni-all-1.8.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/
jaxb-api-2.2.2.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/guice-servlet-3.0.jar:/opt/
client/HDFS/hadoop/share/hadoop/yarn/lib/zookeeper-3.5.1.jar:/opt/client/HDFS/hadoop/share/
hadoop/yarn/lib/javaluator-3.0.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/
xercesImpl-2.9.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/htrace-core-3.1.0-
incubating.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/jettison-1.1.jar:/opt/client/HDFS/
hadoop/share/hadoop/yarn/lib/commons-collections-3.2.2.jar:/opt/client/HDFS/hadoop/share/

hadoop/yarn/lib/asm-3.2.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/jetty-util-6.1.26.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/servlet-api-2.5.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/log4j-1.2.17.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/spark-3.1.1-yarn-shuffle.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/jersey-json-1.9.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/apache-log4j-extras-1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/jackson-mapper-asl-1.9.13.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/protobuf-java-2.5.0.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/jersey-client-1.9.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/commons-logging-1.1.3.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/netty-3.6.2.Final.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/commons-cli-1.2.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/activation-1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/javax.inject-1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/xml-apis-1.3.04.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/guice-3.0.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/jersey-guice-1.9.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/superior-yarn-scheduler-1.0.0.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/xz-1.0.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/netty-all-4.0.23.Final.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/commons-httpclient-3.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/commons-daemon-1.0.13.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/stax-api-1.0-2.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/commons-compress-1.4.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/xmlenc-0.52.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/commons-lang-2.6.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/jersey-core-1.9.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/jackson-jaxrs-1.9.13.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/lib/guava-11.0.2.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/hadoop-yarn-applications-distributedshell-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/hadoop-yarn-server-web-proxy-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/hadoop-yarn-applications-unmanaged-am-launcher-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/hadoop-datasight-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/hadoop-yarn-client-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/hadoop-yarn-server-nodemanager-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/hadoop-yarn-server-common-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/hadoop-yarn-registry-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/hadoop-yarn-common-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/hadoop-yarn-server-applicationhistoryservice-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/hadoop-yarn-server-sharedcachemanager-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/hadoop-yarn-server-resourcemanager-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/yarn/hadoop-yarn-api-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/commons-io-2.4.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/aopalliance-1.0.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/jackson-core-asl-1.9.13.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/avro-1.7.4.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/jersey-server-1.9.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/leveldbjni-all-1.8.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/guice-servlet-3.0.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/asm-3.2.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/log4j-1.2.17.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/jackson-mapper-asl-1.9.13.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/protobuf-java-2.5.0.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/hamcrest-core-1.3.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/netty-3.6.2.Final.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/hadoop-annotations-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/javax.inject-1.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/junit-4.11.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/guice-3.0.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/jersey-guice-1.9.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/xz-1.0.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/paranamer-2.3.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/commons-compress-1.4.1.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/jersey-core-1.9.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/lib/snappy-java-1.0.4.1.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-hs-plugins-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-nativetask-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-core-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-hs-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-app-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-common-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-shuffle-3.1.1.jar:/opt/client/HDFS/hadoop/contrib/capacity-scheduler/*.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/apacheds-i18n-2.0.0-M15.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/commons-math3-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/jackson-xc-1.9.13.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/jaxb-impl-2.2.3-1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/hadoop-openstack-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/commons-io-2.4.jar:/opt/client/HDFS/hadoop/share/hadoop/


```
tools/lib/commons-digester-1.8.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/
jetty-6.1.26.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/jackson-core-asl-1.9.13.jar:/opt/
client/HDFS/hadoop/share/hadoop/tools/lib/api-asn1-api-1.0.0-M20.jar:/opt/client/HDFS/
hadoop/share/hadoop/tools/lib/hadoop-datajoin-3.1.1.jar:/opt/client/HDFS/hadoop/share/
hadoop/tools/lib/avro-1.7.4.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/
jsr305-3.0.0.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/dynalogger-
V100R002C30.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/jersey-server-1.9.jar:/opt/
client/HDFS/hadoop/share/hadoop/tools/lib/commons-codec-1.4.jar:/opt/client/HDFS/hadoop/
share/hadoop/tools/lib/curator-client-2.7.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/
hadoop-gridmix-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/jaxb-api-2.2.2.jar:/opt/
client/HDFS/hadoop/share/hadoop/tools/lib/nimbus-jose-jwt-3.9.jar:/opt/client/HDFS/hadoop/
share/hadoop/tools/lib/json-smart-1.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/
zookeeper-3.5.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/hadoop-
rumen-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/htrace-core-3.1.0-
incubating.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/jettison-1.1.jar:/opt/client/HDFS/
hadoop/share/hadoop/tools/lib/commons-collections-3.2.2.jar:/opt/client/HDFS/hadoop/share/
hadoop/tools/lib/asm-3.2.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/jetty-
util-6.1.26.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/servlet-api-2.5.jar:/opt/client/
HDFS/hadoop/share/hadoop/tools/lib/hadoop-auth-3.1.1.jar:/opt/client/HDFS/hadoop/share/
hadoop/tools/lib/log4j-1.2.17.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/jackson-
databind-2.2.3.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/curator-recipes-2.7.1.jar:/opt/
client/HDFS/hadoop/share/hadoop/tools/lib/gson-2.2.4.jar:/opt/client/HDFS/hadoop/share/
hadoop/tools/lib/commons-beanutils-1.7.0.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/
jets3t-0.9.0.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/mockito-all-1.8.5.jar:/opt/client/
HDFS/hadoop/share/hadoop/tools/lib/jersey-json-1.9.jar:/opt/client/HDFS/hadoop/share/hadoop/
tools/lib/apache-log4j-extras-1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/jackson-
mapper-asl-1.9.13.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/protobuf-
java-2.5.0.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/api-util-1.0.0-M20.jar:/opt/client/
HDFS/hadoop/share/hadoop/tools/lib/jsch-0.1.54.jar:/opt/client/HDFS/hadoop/share/hadoop/
tools/lib/hadoop-archives-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/hamcrest-
core-1.3.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/commons-logging-1.1.3.jar:/opt/
client/HDFS/hadoop/share/hadoop/tools/lib/java-xmlbuilder-0.4.jar:/opt/client/HDFS/hadoop/
share/hadoop/tools/lib/joda-time-2.9.7.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/
apacheds-kerberos-codec-2.0.0-M15.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/
netty-3.6.2.Final.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/hadoop-
distcp-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/commons-cli-1.2.jar:/opt/client/
HDFS/hadoop/share/hadoop/tools/lib/activation-1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/
tools/lib/jackson-core-2.2.3.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/
junit-4.11.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/jsp-api-2.1.jar:/opt/client/HDFS/
hadoop/share/hadoop/tools/lib/httpcore-4.4.4.jar:/opt/client/HDFS/hadoop/share/hadoop/
tools/lib/hadoop-hdfs-restore-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/
httpclient-4.5.2.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/metrics-core-3.0.1.jar:/opt/
client/HDFS/hadoop/share/hadoop/tools/lib/commons-beanutils-core-1.8.0.jar:/opt/client/HDFS/
hadoop/share/hadoop/tools/lib/hadoop-azure-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/
tools/lib/jackson-annotations-2.2.3.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/hadoop-
streaming-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/xz-1.0.jar:/opt/client/HDFS/
hadoop/share/hadoop/tools/lib/jcip-annotations-1.0.jar:/opt/client/HDFS/hadoop/share/hadoop/
tools/lib/commons-lang3-3.3.2.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/
paranamer-2.3.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/curator-
framework-2.7.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/hadoop-sls-3.1.1.jar:/opt/
client/HDFS/hadoop/share/hadoop/tools/lib/commons-httpclient-3.1.jar:/opt/client/HDFS/
hadoop/share/hadoop/tools/lib/stax-api-1.0-2.jar:/opt/client/HDFS/hadoop/share/hadoop/
tools/lib/commons-compress-1.4.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/
xmlenc-0.52.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/commons-lang-2.6.jar:/opt/
client/HDFS/hadoop/share/hadoop/tools/lib/azure-storage-2.0.0.jar:/opt/client/HDFS/hadoop/
share/hadoop/tools/lib/jersey-core-1.9.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/
hadoop-extras-3.1.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/commons-
net-3.1.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/hadoop-ant-3.1.1.jar:/opt/client/
HDFS/hadoop/share/hadoop/tools/lib/jackson-jaxrs-1.9.13.jar:/opt/client/HDFS/hadoop/share/
hadoop/tools/lib/jetty-sslengine-6.1.26.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/
guava-11.0.2.jar:/opt/client/HDFS/hadoop/share/hadoop/tools/lib/snappy-java-1.0.4.1.jar:/opt/
client/HDFS/hadoop/share/hadoop/tools/lib/commons-configuration-1.6.jar
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:java.library.path=/opt/client/
HDFS/hadoop/lib/native
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:java.io.tmpdir=/tmp
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:java.compiler=<NA>
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:os.name=Linux
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:os.arch=amd64
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client
```



```
environment:os.version=2.6.32-504.el6.x86_64
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:user.name=root
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:user.home=/root
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:user.dir=/opt/client
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:os.memory.free=97MB
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:os.memory.max=123MB
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:os.memory.total=123MB
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Initiating client connection,
connectString=192-168-32-144:2181,192-168-32-67:2181,192-168-33-190:2181
sessionTimeout=20000 watcher=com.huawei.hadoop.oi.colocation.ZooKeeperWatcher@60410cd
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: zookeeper.request.timeout is not configured.
Using default value 120000.
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: zookeeper.client.bind.port.range is not configured.
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: zookeeper.client.bind.address is not configured.
17/10/26 19:12:38 INFO client.FourLetterWordMain: connecting to 192-168-32-67 2181
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: Got server principal from the server and it is null
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: Using server principal zookeeper/192-168-32-67
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: Opening socket connection to server
192-168-32-67/192.168.32.67:2181. Will not attempt to authenticate using SASL (unknown
error)
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: Socket connection established, initiating session,
client: /192.168.32.144:52882, server: 192-168-32-67/192.168.32.67:2181
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: Session establishment complete on server
192-168-32-67/192.168.32.67:2181, sessionid = 0x13000074b7e4687f, negotiated timeout =
20000
17/10/26 19:12:38 INFO colocation.ZKUtil: ZooKeeper colocation znode : /hadoop/
colocationDetails. Will publish colocation details under this znode hierarchy.
Create Group is running...
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Initiating client connection,
connectString=192-168-32-144:2181,192-168-32-67:2181,192-168-33-190:2181
sessionTimeout=20000 watcher=com.huawei.hadoop.oi.colocation.ZooKeeperWatcher@f80945f
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: zookeeper.request.timeout is not configured.
Using default value 120000.
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: zookeeper.client.bind.port.range is not configured.
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: zookeeper.client.bind.address is not configured.
17/10/26 19:12:38 INFO client.FourLetterWordMain: connecting to 192-168-32-144 2181
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: Got server principal from the server and it is null
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: Using server principal zookeeper/192-168-32-144
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: Opening socket connection to server
192-168-32-144/192.168.32.144:2181. Will not attempt to authenticate using SASL (unknown
error)
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: Socket connection established, initiating session,
client: /192.168.32.144:40383, server: 192-168-32-144/192.168.32.144:2181
17/10/26 19:12:38 INFO zookeeper.ClientCnxn: Session establishment complete on server
192-168-32-144/192.168.32.144:2181, sessionid = 0x12000059699f69e1, negotiated timeout =
20000
17/10/26 19:12:38 INFO colocation.ZKUtil: ZooKeeper colocation znode : /hadoop/
colocationDetails. Will publish colocation details under this znode hierarchy.
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. *f* indicates a complete file path.

API	Description
public void copyFromLocalFile(Path src, Path dst)	This API is used to upload local files to a specified directory in the HDFS. *src* and *dst* indicate complete file paths.
public boolean mkdirs(Path f)	This API is used to create folders in the HDFS. *f* indicates a complete folder path.
public abstract boolean rename(Path src, Path dst)	This API is used to rename a specified HDFS file. *src* and *dst* indicate complete file paths.
public abstract boolean delete(Path f, boolean recursive)	This API is used to delete a specified HDFS file. *f* 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. *f* 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 FsStatus 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. *file* indicates a complete file path, and *start* and *len* 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. *f* 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. *f* 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.
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. *f* 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.

API	Description
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.

API	Description
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(!hdfsFileIsOpenForWrite(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 contant : %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 contant : %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 contant : %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 contant : Hadoop HDFS Native file write!
hdfsSeek- SUCCESS!, Byte read : 21, File seek contant : S Native file write!
hdfsPread- SUCCESS!, Byte read : 10, File pead contant : 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 contant : 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.

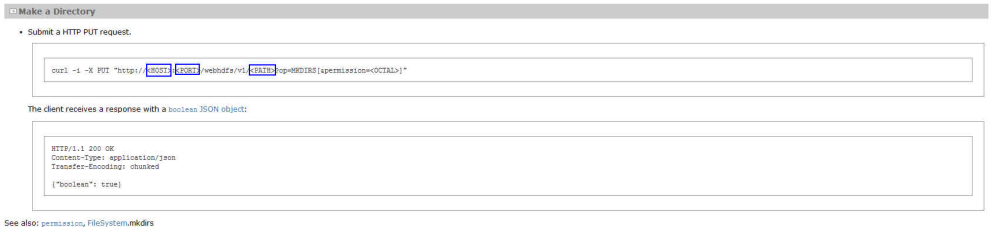
1. Click **Instances**, view the host name and IP address of the active NameNode.
2. Click **Configurations**, search **namenode.http.port** in the search box (9870).

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/J
+ZnVrN7NSz1yKnB2JVIwkj0="; 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&overwrite=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
+6M6PsVMFdAw2cgIHaKU1kBM="; 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 interface has done security hardening to prevent script injection attack. Through REST API interface, 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.	***hadoop key create*** *<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	***hadoop key roll*** *<keyname> [-provider <provider>] [-help]*	The roll subcommand creates a new version for the specified key within the provider indicated using the -provider argument.

Scenario	Operation	Command	Description
	Delete keys	***hadoop key delete*** *<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	***hadoop key list*** *[-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.

Operation	Command	Description
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.

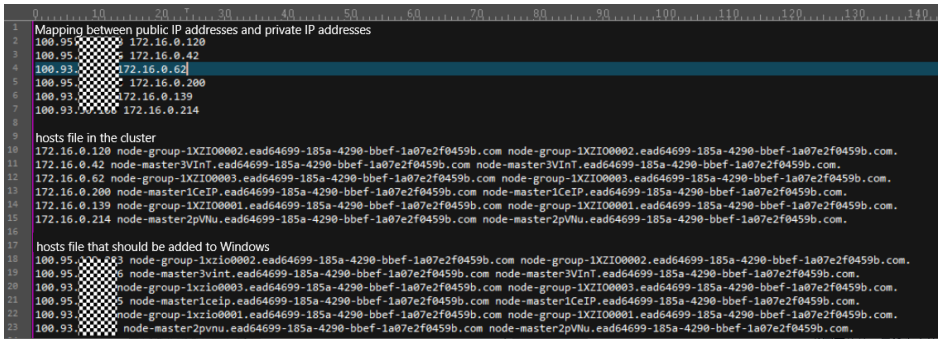
Procedure

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.

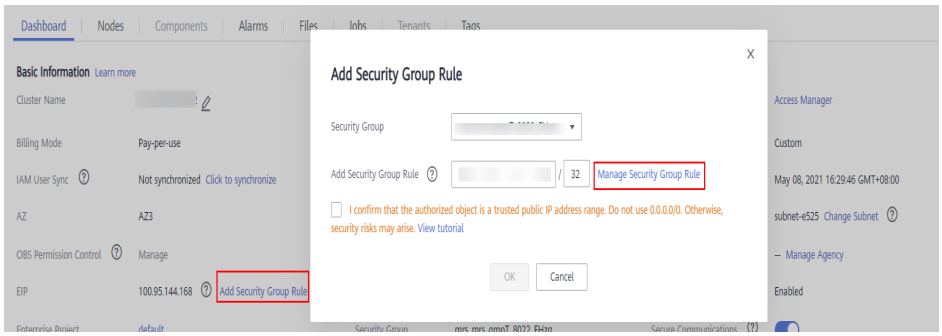
   For details, see **Virtual Private Cloud** > **User Guide** > **EIP** > **Assigning an EIP and Binding It to an ECS**.

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.

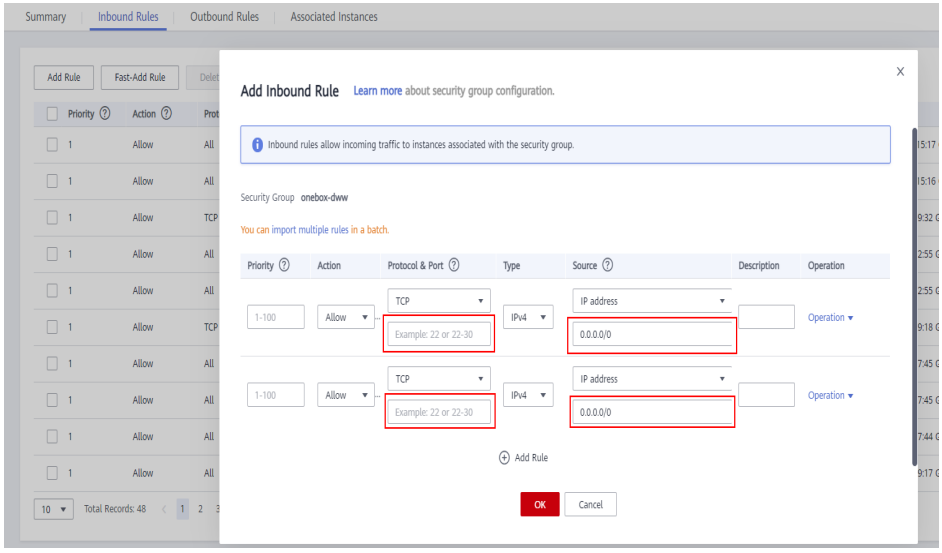


Step 2 Configure security group rules for the cluster.

1. On the **Dashboard** page, choose **Add Security Group Rule** > **Manage Security Group Rule**.



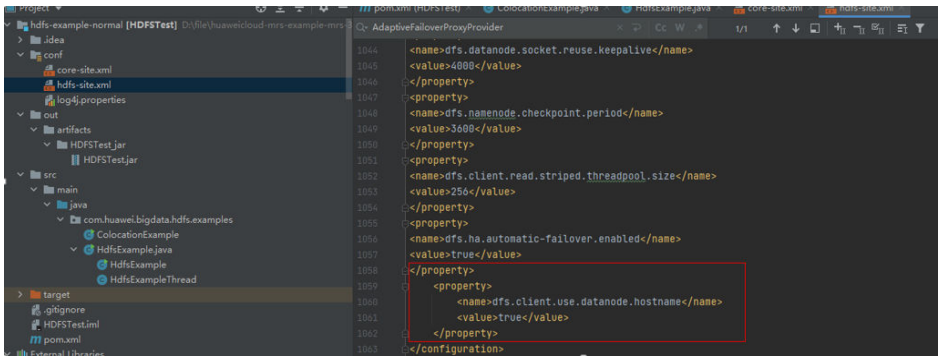
2. On the **Inbound Rules** tab page, click **Add Rule**. In the **Add Inbound Rule** dialog box, configure the Windows IP addresses and ports 8020 and 9866.



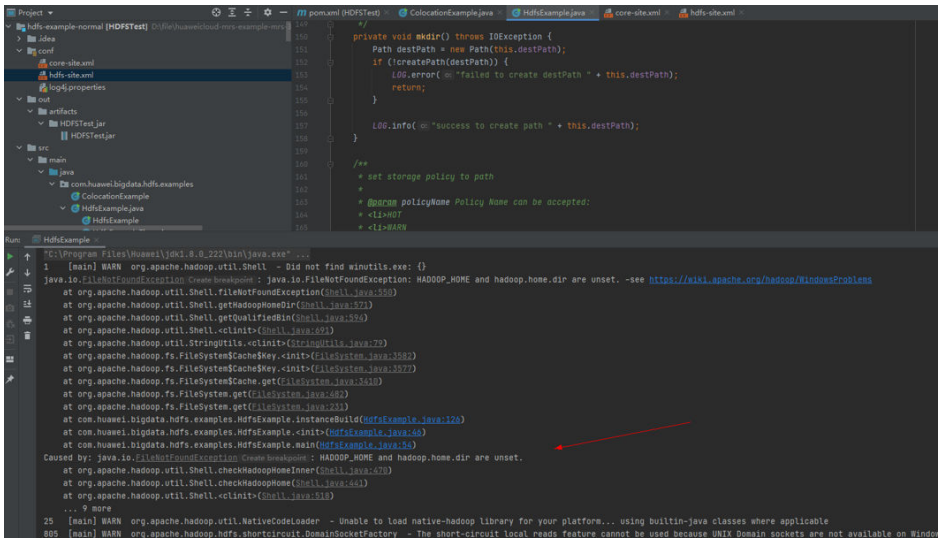
Step 3 On Manager, choose **Cluster** > **Services** > **HDFS** > **More** > **Download Client**, copy the **core-site.xml** and **hdfs-site.xml** files on the client to the **conf** directory of the sample project, and add the following content to the **hdfs-site.xml** file.

```
<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.



----End

16 HetuEngine Development Guide (Security Mode)

16.1 Overview

16.1.1 Introduction to HetuEngine

Introduction to HetuEngine

HetuEngine is a high-performance, interactive SQL analysis and data virtualization engine developed by Huawei. It seamlessly integrates with the big data ecosystem to implement interactive query of massive amounts of data within seconds, and supports cross-source and cross-domain unified data access to enable one-stop SQL convergence analysis in the data lake, between lakes, and between lakehouses.

16.1.2 Concepts

Basic Concepts

- HSBroker: the HetuEngine proxy, which is used for tenant management verification and obtaining the URL for accessing HetuEngine
- Coordinator: the coordinator of HetuEngine services, responsible for SQL parsing and optimization
- Worker: responsible for task execution and data processing
- Connector: an API for HetuEngine to access the database. Through the connector driver, HetuEngine connects to data sources, reads data source metadata, and operates data (adding, deletion, modification, and query).
- Catalog: the catalog configuration file corresponds to a data source in HetuEngine. A data source can have multiple catalog configurations, which can be configured in the **properties** file of a data source.
- Schema: corresponds a schema in a database.
- Table: corresponds to a table name is a database.

16.1.3 Connection Modes

Connection Mode	Support for Username/ Password Authentication	Support for Keytab Authentication	Support for Cross-Network-Segment Client Access	Prerequisite
HSFabric	Supported	Supported	Supported	• The node running user services can communicate with the service nodes where HSFabric resides on the HetuEngine server side. • Dual-plane network scenarios are supported. • Only fixed IP addresses and ports need to be opened for HSFabric. • Supported version: MRS 3.1.3 or later.
HSBroker	Supported	Unsupported	Unsupported	• The node running user services can communicate with the service nodes where HSBroker and Coordinator (randomly distributed in Yarn NodeManger) reside on the HetuEngine server side. • A large number of IP addresses and ports need to be opened for coordinators. • Supported version: MRS 3.1.0 or later

16.1.4 Development Process

This section describes the development process, as shown in **Figure 16-1**.

Figure 16-1 HetuEngine application development process

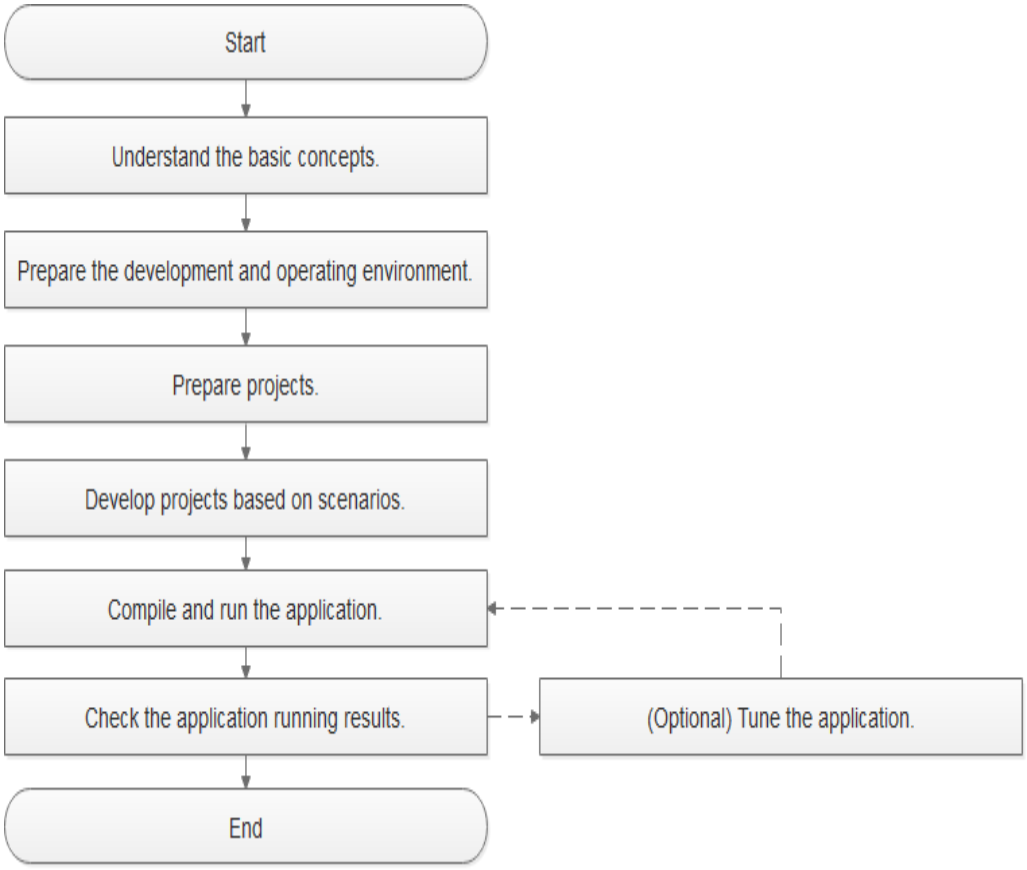


Table 16-1 Description of the HetuEngine application development process

Phase	Description	Reference Documents
Understand basic concepts.	Before application development, learn basic concepts of HetuEngine, and understand the scenario requirements.	**Concepts**
Prepare the development and running environment.	The HetuEngine application can invoke the JDBC interface in any language for development. The current example uses the Java language. You are advised to use the IDEA tool to configure development environments in different languages according to the guide. The HetuEngine running environment is the client. Install and configure the client according to the guide.	**Preparing Development and Running Environments**
Prepare a project.	HetuEngine provides sample projects for different scenarios. You can import a sample project to learn the application. You can also create a HetuEngine project according to the guide.	**Configuring and Importing a Sample Project**

Phase	Description	Reference Documents
Prepare for security authentication.	If a safe cluster is used, the safety certification must be performed.	**Preparing for Security Authentication**
Develop a project based on the scenario.	A sample project in the Java language is provided, including an example project that connects the HetuEngine, SQL statement execution, result parsing, and disconnection.	**Application Development**
Compile and run applications.	You can compile the developed application and submit it for running.	**Application Commissioning**
Check the application running results.	The program running result is displayed in the expected display according to the implementation of the result parsing part.	**Application Commissioning**

16.2 Environment Preparation

16.2.1 Preparing Development and Running Environments

Preparing Development Environment

This section describes the development and running environment to be prepared for application development, as listed in **Table 16-2**.

Table 16-2 Development environment

Item	Description
OS	• 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.

Item	Description
JDK installation	Basic configuration of the development and running environment. The version requirements are as follows: The server and client support only the built-in OpenJDK. Other JDKs cannot be used. If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows: • For x86 nodes that run clients, use the following JDKs: – Oracle JDK 1.8 – IBM JDK 1.8.0.7.20 and 1.8.0.6.15 • For Arm nodes that run clients, use the following JDKs: – OpenJDK 1.8.0_272 (built-in JDK, which can be obtained from the **JDK** folder in the cluster client installation directory.) – BiSheng JDK 1.8.0_272 **NOTE** • For security purposes, the server supports only TLS V1.2 or later. • 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, V1.1, and V1.2. For details, see **https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls**. • For details about the BiSheng JDK, see **https://www.hikunpeng.com/en/developer/devkit/compiler/jdk**.
Installation and configuration of IntelliJ IDEA	It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version. **NOTE** • 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.
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.

Item	Description
User development preparation	See **Preparing the Developer Account** for configuration.
7-zip	Used to decompress **.zip** and **.rar** packages. The 7-Zip 16.04 is supported.

Table 16-3 Python3 environment (required when the Python sample project is used) for MRS 3.3.0 and later versions

Item	Description
Python3 (for MRS 3.3.0 or later)	Tool used to develop HetuEngine Python applications. The version must range from 3.6 to 3.9.
Setuptools	Basic configuration of the Python3 development environment. Version: 47.3.1
jaydebeapi	Basic configuration of the Python3 development environment. You can use this module to connect to the database using Java JDBC.

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\HetuEngine\config,**

place the configuration file in the **resources** directory of the HetuEngine sample project.

The keytab file obtained during the **Preparing the Developer Account** is also stored in this directory.

Log in to the node where the HSBroker role is deployed as user **omm**, go to the **${BIGDATA_HOME}/FusionInsight_Hetu_*XXX*/*XXX*_HSBroker/etc/** directory, download the **hetuserver.jks** file, and place the file to the **resources** directory. For details about the configuration files, see **Table 16-4**. (Obtain the corresponding files as required.)

Table 16-4 Configuration files

File	Function
hdfs-site.xml	Configures HDFS parameters.
hetuserver-client.properties	Configures connection parameters for the HetuEngine client.
hetuserver-client-logging.properties	Configures log parameters for the HetuEngine client.
user.keytab	Provides HetuEngine user information for Kerberos security authentication.
krb5.conf	Contains Kerberos server configuration information.
hetuserver.jks	Java TrustStore 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 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/HetuEngine/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 HetuEngine/config/* root@*IP address of the client node***:/opt/client/conf**

   Log in to the node where the HSBroker role is deployed as user **omm**, go to the **${BIGDATA_HOME}/FusionInsight_Hetu_xxx/xxx_HSBroker/etc/** directory, download the **hetuserver.jks** file, and place the file to the this directory

   The keytab file obtained during the **Preparing the Developer Account** is also stored in this directory. **Table 16-5** describes the main configuration files (Obtain required files as required).

   Table 16-5 Configuration files

File	Function
hdfs-site.xml	Configures HDFS parameters.
hetuserver-client.properties	Configures connection parameters for the HetuEngine client.
hetuserver-client-logging.properties	Configures log parameters for the HetuEngine client.
user.keytab	Provides HetuEngine user information for Kerberos security authentication.
krb5.conf	Contains Kerberos server configuration information.
hetuserver.jks	Java TrustStore 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.

16.2.2 Configuring and Importing a Sample Project

Scenario

The client installation program directory contains a HetuEngine development sample project. You can start the sample project learning by importing the project. This document uses IntelliJ IDEA 2020.1.3 (Community Edition) as an example.

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 time of the MRS cluster in the upper-right corner on the FusionInsight Manager page.

Procedure

Step 1 Obtain the sample project folder **hetu-examples\hetu-examples-security** 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. Open IntelliJ IDEA and choose **File** > **New** > **General** > **Project from Existing Sources** > **Select File or Directory to Import** to go to the **Browse Folder** dialog box is displayed.
2. Select the sample project folder, choose **Import project from external model** > **Maven** during import, 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.

----End

16.2.3 Configuring the Python3 Sample Project

This section applies to MRS 3.3.0 or later.

Scenario

The following content describes the operations you need to do to run the pyhon3 sample code of HetuEngine on FusionInsight MRS.

Procedure

Step 1 Install Python 3.6 or a later version (before 3.9) on the client node.

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 must be installed on the client.

Download the software from the official website **https://pypi.org/project/setuptools/#files**.

Copy the downloaded setuptools package to the client, decompress the package, go to the setuptools project directory, and run the **python3 setup.py install** command in the CLI of the client.

Take version 47.3.1 as an example, 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 setuptools 47.3.1 fails to be installed, check whether the environment is faulty or Python is faulty.

Step 3 The JayDeBeApi module must be installed on the client. You can use the Java JDBC to connect to the database through this module.

You can install it either of the following ways:

- pip:

  Run the **pip install JayDeBeApi** command on the client node.

- Script:

  a. Download the **JayDeBeApi project** file from the official website at **https://pypi.org/project/JayDeBeApi/**

  b. Go to the **JayDeBeApi** project directory and run the **python3 setup.py install** command. During the installation, if the system displays a message indicating that the python3 module or package is missing, you need to add the module or package.

  Take JayDeBeApi-1.2.3 as an example. If **Successfully installed JayDeBeApi-1.2.3** is displayed, the installation is successful.

Step 4 Install Java on the client. For details about the supported Java versions, see "JDK Installation" in the **Table 16-2**.

Step 5 Obtain the Python3 sample code.

1. Obtain the sample project folder **python3-examples** in the **src\hetu-examples** directory where the sample code is decompressed. For details, see **Obtaining Sample Projects from Huawei Mirrors**.
2. Go to the **python3-examples** folder.
   - **normal** folder: Python3 sample code for interconnecting with HetuEngine in common mode
   - **security** folder: Python3 sample code for interconnecting with HetuEngine in security mode

Step 6 Obtain the **hetu-jdbc** JAR package.

- Download the client file to the local PC through Manager.
  a. Log in to FusionInsight Manager and choose **Cluster** > **Services** > **HetuEngine** > **More** > **Download Client**.
  b. Select **Complete Client**, select the correct platform type (**x86_64** for x86 and **aarch64** for ARM) for the node where the client is to be installed, deselect **Save to Path**, and click **OK**. Wait until the client file package is generated and download it.
  c. Decompress the downloaded software package to obtain the **hetu-jdbc** package and decompress it to obtain the JDBC package.

     For example, if the client file package is **FusionInsight_Cluster_1_Services_Client.tar**, decompress the package to obtain the **FusionInsight_Cluster_1_Services_ClientConfig.tar** file. Then decompress the package to obtain the **FusionInsight_Cluster_1_Services_ClientConfig** file. (The path cannot contain spaces.) Decompress **\FusionInsight_Cluster_1_Services_ClientConfig\HetuEngine\x86\hetu-jdbc.tar.gz** to obtain **hetu-jdbc-XXX.jar** and copy it to the user-defined path on the host where the sample code will run.
- Obtain the cluster client node.

  Log in to the node where the HetuEngine client has been installed. For example, if the client installation path is **/opt/hadoopclient**, obtain **hetu-jdbc-XXX.jar** from **/opt/hadoopclient/HetuEngine/hetuserver/jars/**, and copy the file to the user-defined path on the node where the sample code will run.

----End

16.2.4 Preparing for Security Authentication

16.2.4.1 KeyTab File Authentication Using HSFabric

The KeyTab file authentication requires the **jaas-zk.conf**, **krb5.conf**, and **user.keytab** files.

For details about how to obtain the **krb5.conf** and **user.keytab** files, see **Security Authentication**.

In the **jaas-zk.conf** file, **principal** is *the username added for authentication in section **Security Authentication**@domain name*. The domain name is the value of the **default_realm** field in the **krb5.conf** file (For example, HADOOP.COM). **keyTab** is the path of the **user.keytab** file.

```
Client {
com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true
keyTab="/opt/client/user.keytab"
principal="hivetest@System domain name"
useTicketCache=false
storeKey=true
debug=true;
};
```


NOTE

Change the value of **keyTab** in the **jaas-zk.conf** file based on the site requirements.

For example:

- Windows Path: "D:\\hetu-examples\\hetu-examples-security\\src\\main\\resources\\user.keytab".
- Linux Path: "/opt/client/user.keytab".

16.2.4.2 Username and Password Authentication Using HSFabric

Only the username and password are required to implement this authentication using HSFabric.

16.2.4.3 Username and Password Authentication Using HSBroker

Only the username and password are required to implement this authentication using HSBroker.

16.3 Application Development

16.3.1 Typical Application Scenario

This section describes the application development in a typical scenario, helping you quickly learn and master the HetuEngine development process and know key functions.

Scenario

Assume that a user develops an application and needs to perform the join operation on table A of the Hive data source and table B of the MPPDB data source. In this case, HetuEngine can be used to implement data query of the Hive data source, the process is as follows:

1. Connect to the HetuEngine JDBC server.
2. Assemble SQL statements.
3. Execute SQL statements.
4. Parse the returned result.
5. Close the HetuEngine JDBC Server connection.

16.3.2 Java Sample Code

16.3.2.1 KeyTab File Authentication Using HSFabric

Description

This section describes how to use the KeyTab file to connect to HetuEngine, assemble SQL statements, and send the SQL statements to HetuEngine for execution to add, delete, modify, and query Hive data sources.


```
public class JDBCExampleZk {
    private static Properties properties = new Properties();
    private final static String PATH_TO_JAAS_ZK_CONF = JDBCExample.class.getClassLoader()
            .getResource("jaas-zk.conf")
            .getPath();
    private final static String PATH_TO_KRB5_CONF = JDBCExample.class.getClassLoader()
            .getResource("krb5.conf")
            .getPath();
    private final static String PATH_TO_USER_KEYTAB = JDBCExample.class.getClassLoader()
            .getResource("user.keytab")
            .getPath();
    private final static String PATH_TO_HETUSERVER_JKS = JDBCExamplePasswordZK.class.getClassLoader()
            .getResource("hetuserver.jks")
            .getPath();
    private static void init() throws ClassNotFoundException {
        System.setProperty("user.timezone", "UTC");
        System.setProperty("java.security.auth.login.config", PATH_TO_JAAS_ZK_CONF);
        System.setProperty("java.security.krb5.conf", PATH_TO_KRB5_CONF);
        properties.setProperty("user", "hivetest");
        properties.setProperty("SSL", "true");
        properties.setProperty("KerberosConfigPath", PATH_TO_KRB5_CONF);
        properties.setProperty("KerberosPrincipal", "hivetest");
        properties.setProperty("KerberosKeytabPath", PATH_TO_USER_KEYTAB);
        properties.setProperty("SSLTrustStorePath", PATH_TO_HETUSERVER_JKS);
        properties.setProperty("KerberosRemoteServiceName", "HTTP");
        properties.setProperty("tenant", "default");
        properties.setProperty("deploymentMode", "on_yarn");
        properties.setProperty("ZooKeeperAuthType", "kerberos");
        properties.setProperty("ZooKeeperSaslClientConfig", "Client");
        Class.forName("io.XXXsql.jdbc.XXXDriver");
    }
    /**
     * Program entry
     *
     * @param args no need program parameter
     */
    public static void main(String[] args) {
        Connection connection = null;
        ResultSet result = null;
        PreparedStatement statement = null;
        String url = "jdbc:XXX://192.168.1.130:2181,192.168.1.131:2181,192.168.1.132:2181/hive/default?"
            + "serviceDiscoveryMode=zooKeeper&zooKeeperNamespace=hsbroker";
        try {
            init();
            String sql = "show tables";
            connection = DriverManager.getConnection(url, properties);
            statement = connection.prepareStatement(sql.trim());
            result = statement.executeQuery();
            ResultSetMetaData resultMetaData = result.getMetaData();
            Integer colNum = resultMetaData.getColumnCount();
            for (int j = 1; j <= colNum; j++) {
                System.out.print(resultMetaData.getColumnLabel(j) + "\t");
            }
            System.out.println();
            while (result.next()) {
                for (int j = 1; j <= colNum; j++) {
                    System.out.print(result.getString(j) + "\t");
                }
                System.out.println();
            }
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            if (result != null) {
                try {
                    result.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
```



```
            }
            if (statement != null) {
                try {
                    statement.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (connection != null) {
                try {
                    connection.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Table 16-6 describes the parameters in the preceding code.

Table 16-6 Parameter description

Parameter	Description
url	jdbc:*XXX*://*zkNode1_IP*:*zkNode1_Port*,*zkNode2_IP*:*zkNode2_Port*,*zkNode3_IP*:*zkNode3_Port*/*catalog*/*schema*?serviceDiscoveryMode=zooKeeper&zooKeeperNamespace=hsbroker **NOTE** • xxx: driver name, which is subjective to the real-world code you use. • *catalog* and *schema* indicate the names of the catalog and schema to be connected to the JDBC client, respectively. • *zkNode_IP*:*zkNode_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.
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
socksProxy	Indicates the SOCKS proxy server, for example, **localhost:1080**.
httpProxy	Indicates the HTTP proxy server address, for example, **localhost:8888**.
applicationNamePrefix	Indicates the prefix to be attached to any specified ApplicationName client information property that is used to set the source name for a HetuEngine query. If neither this property nor **ApplicationName** is set, the source for the query is HetuEngine JDBC.
accessToken	Indicates the token-based authentication token.

Parameter	Description
SSL	Indicates whether to use the HTTPS connection. The default value is **false**.
SSLKeyStorePath	Indicates the Java KeyStore file path.
SSLKeyStorePassword	Indicates the Java KeyStore password.
SSLTrustStorePath	Indicates the Java TrustStore file path.
SSLTrustStorePassword	Indicates the Java TrustStore password.
KerberosRemoteServiceName	Indicates the Kerberos service name, which is fixed to **HTTP**.
KerberosPrincipal	Indicates the username corresponding to **keytab** specified by **KerberosKeytabPath**.
KerberosUseCanonicalHostname	Indicates whether to use the standardized host name. The default value is **false**.
KerberosConfigPath	The **krb5** configuration file to access the data source user. For details, see **Preparing for Security Authentication**.
KerberosKeytabPath	Indicates the **keytab** configuration file of the data source user, which can be obtained by following the instructions in **Preparing for Security Authentication**.
KerberosCredentialCachePath	Indicates the Kerberos **credential** cache path.
extraCredentials	Indicates additional credentials used to connect to external systems. **extraCredentials** is the key-value pair list, for example, **foo:bar;abc: xyz** creates credentials **abc = xyz** and **foo = bar**.
java.security.auth.login.config	Indicates the path of the **jaas-zk.conf** configuration file, which is used to access ZooKeeper in security mode.
java.security.krb5.conf	Indicates the **krb5** configuration file. For details, see **Preparing for Security Authentication**.
ZooKeeperAuthType	Indicates the ZooKeeper authentication mode. The value is **kerberos** in security mode.
ZooKeeperSaslClientConfig	Indicates the item name in the **jaas-zk.conf** configuration file.
tenant	Indicates the tenant to which a user belongs.
deploymentMode	Only **on_yarn** is supported.

16.3.2.2 Username and Password Authentication Using HSFabric

Description

This section describes how to use HSFabric to connect to HetuEngine with the username and password, and assemble and send the SQL statements to HetuEngine for execution.

```
public class JDBCExampleFabric {
    private static Properties properties = new Properties();
    private static void init() throws ClassNotFoundException {
        // Hard-coded password or plaintext password in code poses significant security risks. Encrypt and store them in configuration files or environment variables and decrypt them when needed.
        // The password is stored in environment variables for identity authentication. Before running this example, set the environment variable HETUENGINE_PASSWORD.
        properties.setProperty("user", "YourUserName");
        String password = System.getenv("HETUENGINE_PASSWORD");
        properties.setProperty("password", password);
        Class.forName("io.xxx.jdbc.xxxDriver");
    }
    public static void main(String[] args){
        Connection connection = null;
        ResultSet resultSet = null;
        PreparedStatement statement = null;
        String url = "jdbc:xxx://192.168.1.130:29902,192.168.1.131:29902/hive/default?serviceDiscoveryMode=hsfabric";
        try {
            init();
            String sql = "show tables";
            connection = DriverManager.getConnection(url, properties);
            statement = connection.prepareStatement(sql.trim());
            resultSet = statement.executeQuery();
            Integer colNum = resultSet.getMetaData().getColumnCount();
            while (resultSet.next()) {
                for (int i = 1; i <= colNum; i++) {
                    System.out.println(resultSet.getString(i) + "\t");
                }
            }
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            if (resultSet != null) {
                try {
                    resultSet.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (statement != null) {
                try {
                    statement.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (connection != null) {
                try {
                    connection.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Table 16-7 describes the parameters in the preceding code.

Table 16-7 Parameter description

Parameter	Description
url	jdbc:*xxx*://*HSFabric1_IP.HSFabric1_Port*,*HSFabric2_IP.HSFabric2_Port*,*HSFabric3_IP.HSFabric3_Port*/*catalog*/*schema*?serviceDiscovery-Mode=hsfabric **NOTE** • xxx: driver name, which is subjective to the real-world code you use. • *catalog* and *schema* indicate the names of the catalog and schema to be connected to the JDBC client, respectively. • *HSFabric_IP:HSFabric_Port* indicates the HSFabric URL. Use commas (,) to separate multiple URLs, for example, 192.168.1.130:29902,192.168.1.131:29902,192.168.1.132:29902. On FusionInsight Manager, choose **Cluster** > **Services** > **HetuEngine** and click **Instance** to obtain the service IP addresses of all HSFabric instances. On the **Configurations** page, search for **gateway.port** to obtain the port number of HSFabric.
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
password	Password of the human-machine user created in the cluster.

16.3.2.3 Querying the Execution Progress and Status of an SQL Statement Using JDBC

Description

This section describes how to use JDBC to connect to HetuEngine with the username and password, and assemble and send the SQL statements to HetuEngine for execution.

```
import io.xxx.jdbc.xxxResultSet;
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 java.util.Timer;
import java.util.TimerTask;

public class JDBCExampleStatementProgressPercentage{

    private static Properties properties = new Properties();
    public static Connection connection = null;
    public static ResultSet result = null;
    public static PreparedStatement statement = null;
    private static void init() throws ClassNotFoundException {
        // Hard-coded password or plaintext password in code poses significant security risks. Encrypt and store them in configuration files or environment variables and decrypt them when needed.
        // The password is stored in environment variables for identity authentication. Before running this example, set the environment variable HETUENGINE_PASSWORD.
        properties.setProperty("user", "YourUserName");
        String password = System.getenv("HETUENGINE_PASSWORD");
```

```
        properties.setProperty("password", password);
        Class.forName("io.xxx.jdbc.xxxDriver");
    }

    /**
     * Program entry
     *
     * @param args no need program parameter
     */
    public static void main(String[] args) {
        String url = "jdbc:xxx://192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181/hive/default?
serviceDiscoveryMode=hsbroker";
        try {
            init();
            String sql = "show tables";
            connection = DriverManager.getConnection(url, properties);
            statement = connection.prepareStatement(sql.trim());
            result = statement.executeQuery();

            xxxResultSet rs = (xxxResultSet) result;
            new Thread() {
                public void run() {
                    Timer timer = new Timer();
                    //The SQL statement is executed after 3 seconds and is executed every 2 seconds
                    timer.schedule(new TimerTask() {
                        @Override
                        public void run() {
                            double statementProgressPercentage = rs.getProgressPercentage().orElse(0.0);
                            System.out.println("The Current Query Progress Percentage is " +
statementProgressPercentage*100 + "%");
                            if("FINISHED".equals(rs.getStatementStatus().orElse(""))) {
                                System.out.println("The Current Query Progress Percentage is 100%");
                                timer.cancel();
                                Thread.currentThread().interrupt();
                            }
                        }
                    }, 3000, 2000);
                }
            }.start();

            ResultSetMetaData resultMetaData = result.getMetaData();
            int colNum = resultMetaData.getColumnCount();
            for (int j = 1; j <= colNum; j++) {
                try {
                    System.out.print(resultMetaData.getColumnLabel(j) + "\t");
                } catch (SQLException throwables) {
                    throwables.printStackTrace();
                }
            }

            while (result.next()) {
                for (int j = 1; j <= colNum; j++) {
                    System.out.print(result.getString(j) + "\t");
                }
                System.out.println();
            }
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            if (result != null) {
                try {
                    result.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (statement != null) {
                try {
                    statement.close();
```

```
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
  }
}
```

Table 16-8 describes the parameters in the preceding code.

Table 16-8 Parameter description

Parameter	Description
url	jdbc:*xxx*://*HSBroker1_IP*.*HSBroker1_Port*,*HSBroker2_IP*.*HSBroker2_Port*,*HSBroker3_IP*.*HSBroker3_Port*/*catalog*/*schema*?serviceDiscoveryMode=hsbroker **NOTE** • xxx: driver name, which is subjective to the real-world code you use. • *catalog* and *schema* indicate the names of the catalog and schema to be connected to the JDBC client, respectively. • *HSBroker_IP:HSBroker_Port* indicates the HSBroker URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181.
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
password	Password of the human-machine user created in the cluster.
getStatementStatus()	Execution status of an SQL statement, which can be **RUNNING**, **FAILED**, **FINISHED**, **QUEUEED**, **WAITING_FOR_RESOURCES**, **DISPATCHING**, **PLANNING**, **STARTING**, **RESCHEDULING**, **RESUMING**, or **FINISHING**.
getProgressPercentage()	Execution progress of an SQL statement. The value range is 0 to 1.

16.3.2.4 Username and Password Authentication Using HSBroker

Description

This section describes how to use HSBroker to connect to HetuEngine with the username and password, and assemble and send the SQL statements to HetuEngine for execution.

```
public class JDBCExampleBroker {
    private static Properties properties = new Properties();
    private static void init() throws ClassNotFoundException {
```



```
        // Hard-coded password or plaintext password in code poses significant security risks. Encrypt and
store them in configuration files or environment variables and decrypt them when needed.
        // The password is stored in environment variables for identity authentication. Before running this
example, set the environment variable HETUENGINE_PASSWORD.
        properties.setProperty("user", "YourUserName");
        String password = System.getenv("HETUENGINE_PASSWORD");
        properties.setProperty("password", password);
        Class.forName("io.XXX.sql.jdbc.XXXDriver");
    }
    public static void main(String[] args){
        Connection connection = null;
        ResultSet resultSet = null;
        PreparedStatement statement = null;
        String url = "jdbc:XXX://192.168.1.130:29860,192.168.1.131:29860/hive/default?
serviceDiscoveryMode=hsbroker";
        try {
            init();
            String sql = "show tables";
            connection = DriverManager.getConnection(url, properties);
            statement = connection.prepareStatement(sql.trim());
            resultSet = statement.executeQuery();
            Integer colNum = resultSet.getMetaData().getColumnCount();
            while (resultSet.next()) {
                for (int i = 1; i <= colNum; i++) {
                    System.out.println(resultSet.getString(i) + "\t");
                }
            }
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            if (resultSet != null) {
                try {
                    resultSet.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (statement != null) {
                try {
                    statement.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (connection != null) {
                try {
                    connection.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Table 16-9 describes the parameters in the preceding code.

Table 16-9 Parameter description

Parameter	Description
url	jdbc:*XXX*://*HSBroker1_IP.HSBroker1_Port,HSBroker2_IP.HSBroker2_Port,HSBroker3_IP.HSBroker3_Port*/*catalog*/*schema*?serviceDiscovery-Mode=hsbroker **NOTE** • xxx: driver name, which is subjective to the real-world code you use. • *catalog* and *schema* indicate the names of the catalog and schema to be connected to the JDBC client, respectively. • *HSBroker_IP:HSBroker_Port* indicates the HSBroker URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181.
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
password	Password of the human-machine user created in the cluster.

16.3.3 Python3 Sample Code

16.3.3.1 Username and Password Authentication Using HSBroker

This section applies to MRS 3.3.0 or later.

This section describes how to use HSBroker to connect to HetuEngine with the username and password, and assemble and send the SQL statements to HetuEngine for execution.

```
import jaydebeapi

driver = "io.xxx.jdbc.xxxDriver"

# need to change the value based on the cluster information
url = "jdbc:xxx://192.168.43.223:29860,192.168.43.244:29860/hive/default?serviceDiscoveryMode=hsbroker"
user = "YourUserName"
// Hard-coded password or plaintext password in code poses significant security risks. Encrypt and store them in configuration files or environment variables and decrypt them when needed.
// The password is stored in environment variables for identity authentication. Before running this example, set the environment variable HETUENGINE_PASSWORD.
password = os.getenv('HETUENGINE_PASSWORD')
tenant = "YourTenant"
jdbc_location = "Your file path of the jdbc jar"

sql = "show tables"

if __name__ == '__main__':
    conn = jaydebeapi.connect(driver, url, {"user": user,
                                            "password": password,
                                            "tenant": tenant},
                              [jdbc_location])
    curs = conn.cursor()
    curs.execute(sql)
    result = curs.fetchall()
    print(result)
```



```
    curs.close()
conn.close()
```

The following table describes the parameters in the preceding code.

Table 16-10 Parameter description

Parameter	Description
url	jdbc:*xxx*://*HSBroker1_IP*:*HSBroker1_Port*,*HSBroker2_IP*:*HSBroker2_Port*,*HSBroker3_IP*:*HSBroker3_Port*/*catalog*/*schema*?serviceDiscovery-Mode=hsbroker **NOTE** • xxx: driver name, which is subjective to the real-world code you use. • *catalog* and *schema* indicate the names of the catalog and schema to be connected to the JDBC client, respectively. • *HSBroker_IP:HSBroker_Port* indicates the HSBroker URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181.
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
password	Password of the human-machine user created in the cluster.
tenant	Tenant resource queue for accessing HetuEngine compute instances
jdbc_location	Full path of the **hetu-jdbc-XXX.jar** package obtained in **Configuring the Python3 Sample Project**. • Example path in Windows: **D:\\hetu-examples-python3\\hetu-jdbc-XXX.jar** • Example path in Linux: **/opt/hetu-examples-python3/hetu-jdbc-XXX.jar**

16.3.3.2 Username and Password Authentication Using HSFabric

This section describes how to use HSFabric to connect to HetuEngine with the username and password, and assemble and send the SQL statements to HetuEngine for execution.

```
import jaydebeapi

driver = "io.xxx.jdbc.xxxDriver"

# need to change the value based on the cluster information
url = "jdbc:xxx://192.168.43.244:29902/hive/default?serviceDiscoveryMode=hsfabric"
user = "YourUserName"
// Hard-coded password or plaintext password in code poses significant security risks. Encrypt and store them in configuration files or environment variables and decrypt them when needed.
// The password is stored in environment variables for identity authentication. Before running this example, set the environment variable HETUENGINE_PASSWORD.
password = os.getenv('HETUENGINE_PASSWORD')
tenant = "YourTenant"
jdbc_location = "Your file path of the jdbc jar"
```



```
sql = "show tables"

if __name__ == '__main__':
    conn = jaydebeapi.connect(driver, url, {"user": user,
                                            "SSL": "true",
                                            "password": password,
                                            "tenant": tenant},
                              [jdbc_location])
    curs = conn.cursor()
    curs.execute(sql)
    result = curs.fetchall()
    print(result)
    curs.close()
    conn.close()
```

The following table describes the parameters in the preceding code.

Table 16-11 Parameter description

Parameter	Description
url	jdbc:*xxx*://*HSFabric1_IP*.*HSFabric1_Port*,*HSFabric2_IP*.*HSFabric2_Port*,*HSFabric3_IP*.*HSFabric3_Port*/*catalog*/*schema*?serviceDiscovery-Mode=hsfabric **NOTE** • xxx: driver name, which is subjective to the real-world code you use. • **catalog** and **schema** indicate the names of the catalog and schema to be connected to the JDBC client, respectively. • **HSFabric_IP:HSFabric_Port** indicates the HSFabric URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:29902,192.168.195.232:29902,192.168.169.84:29902. On FusionInsight Manager, choose **Cluster** > **Services** > **HetuEngine** and click **Instance** to obtain the service IP addresses of all HSFabric instances. On the **Configurations** page, search for **gateway.port** to obtain the port number of HSFabric.
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
password	Password of the human-machine user created in the cluster.
tenant	Tenant resource queue for accessing HetuEngine compute instances
jdbc_location	Full path of the **hetu-jdbc-XXX.jar** package obtained in **Configuring the Python3 Sample Project**. • Example path in Windows: **D:\\hetu-examples-python3\\hetu-jdbc-XXX.jar** • Example path in Linux: **/opt/hetu-examples-python3/hetu-jdbc-XXX.jar**

16.3.3.3 KeyTab File Authentication Using HSFabric

This section describes how to use the KeyTab file and HSFabric to connect to HetuEngine, assemble SQL statements, and send the SQL statements to HetuEngine for execution to add, delete, modify, and query Hive data sources.


```
import jaydebeapi

driver = "io.xxx.jdbc.xxxDriver"

# need to change the value based on the cluster information
url = "jdbc:xxx://192.168.43.244:29902/hive/default?serviceDiscoveryMode=hsfabric"
user = "YourUserName"
KerberosPrincipal = "YourUserName"
tenant = "YourTenant"
KerberosConfigPath = "Your file path of krb5.conf"
KerberosKeytabPath = "Your file path of user.keytab"
jdbc_location = "Your file path of the jdbc jar"

sql = "show tables"

if __name__ == '__main__':
    conn = jaydebeapi.connect(driver, url, {'user': user,
                                            'SSL': "true",
                                            'KerberosPrincipal': KerberosPrincipal,
                                            'KerberosConfigPath': KerberosConfigPath,
                                            'KerberosRemoteServiceName': "HTTP",
                                            "KerberosKeytabPath": KerberosKeytabPath,
                                            "tenant": tenant,
                                            'deploymentMode': "on_yarn",
                                            "ZooKeeperSaslClientConfig": "Client"},
                              [jdbc_location])
    curs = conn.cursor()
    curs.execute(sql)
    result = curs.fetchall()
    print(result)
    curs.close()
    conn.close()
```

The following table describes the parameters in the preceding code.

Table 16-12 Parameter description

Parameter	Description
url	jdbc:*xxx*://*HSFabric1_IP*.*HSFabric1_Port*,*HSFabric2_IP*.*HSFabric2_Port*,*HSFabric3_IP*.*HSFabric3_Port*/*catalog*/*schema*?serviceDiscovery-Mode=hsfabric **NOTE** • xxx: driver name, which is subjective to the real-world code you use. • **catalog** and **schema** indicate the names of the catalog and schema to be connected to the JDBC client, respectively. • **HSFabric_IP:HSFabric_Port** indicates the HSFabric URL. Use commas (,) to separate multiple URLs, for example, 192.168.1.130:29902,192.168.1.131:29902,192.168.1.132:29902. On FusionInsight Manager, choose **Cluster** > **Services** > **HetuEngine** and click **Instance** to obtain the service IP addresses of all HSFabric instances. On the **Configurations** page, search for **gateway.port** to obtain the port number of HSFabric.
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
KerberosPrincipal	Username corresponding to **keytab** specified by **KerberosKeytabPath**

Parameter	Description
tenant	Tenant resource queue for accessing HetuEngine compute instances
KerberosConfigPath	The **krb5** configuration file to access the data source user. For details, see **Preparing for Security Authentication**.
KerberosKeytabPath	The **keytab** configuration file of the data source user, which can be obtained by following the instructions in **Preparing for Security Authentication**.
jdbc_location	Full path of the **hetu-jdbc-XXX.jar** package obtained in **Configuring the Python3 Sample Project**. • Example path in Windows: **D:\\hetu-examples-python3\\hetu-jdbc-XXX.jar** • Example path in Linux: **/opt/hetu-examples-python3/hetu-jdbc-XXX.jar**

16.4 Application Commissioning

16.4.1 Commissioning Applications on Windows

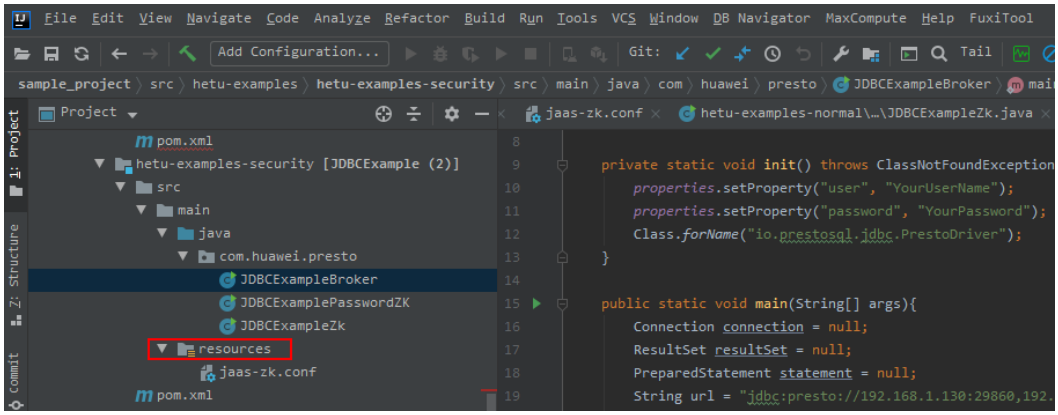
Scenario

After the program code is developed, you can compile the program in the Windows environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

Procedure

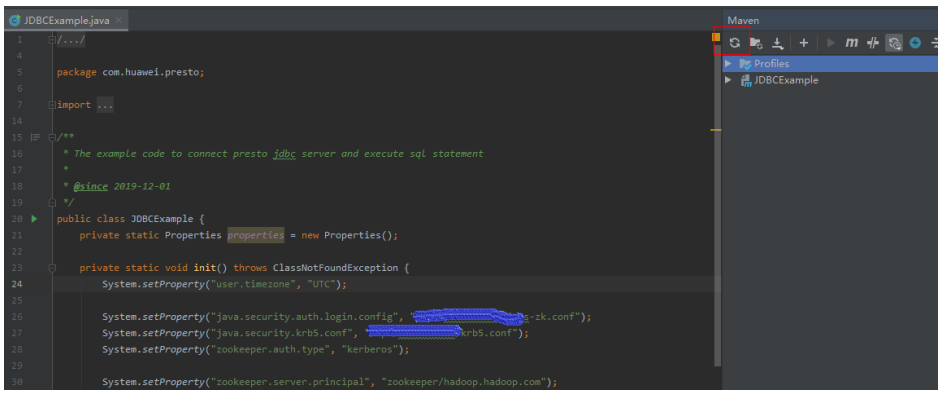
Step 1 In the IntelliJ IDEA development environment on Windows, check that the **user.keytab** and **krb5.conf** files obtained in **Preparing for Security Authentication** are saved to the **resources** directory, and modify the parameters in the **jaas-zk.conf** file based on the actual path and username.

Figure 16-2 Saving the authentication file to the resources directory



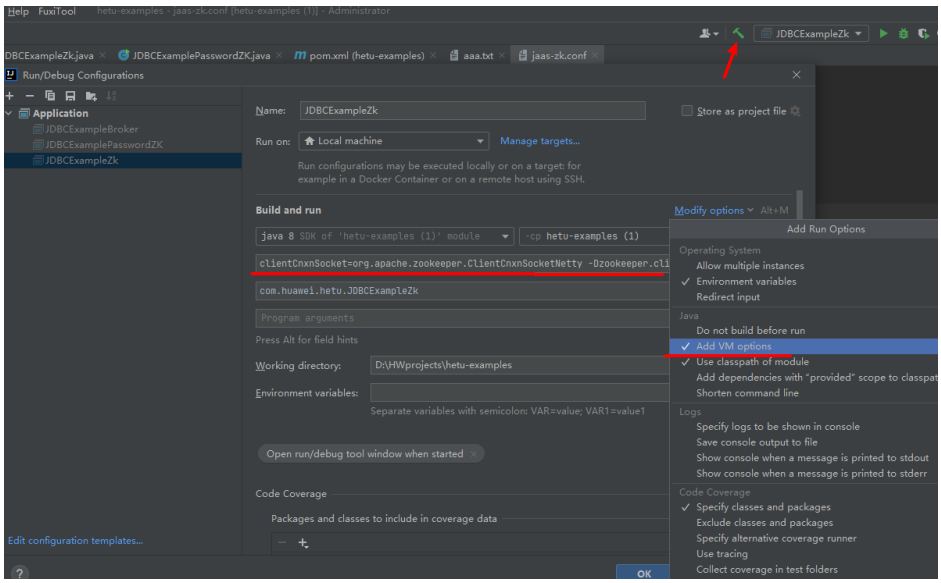
Step 2 Click **Maven** on the right of IDEA to import the dependency.

Figure 16-3 Importing the dependency



Step 3 (Optional) If the SSL authentication communication function of ZooKeeper is enabled for the interconnected cluster, add the JVM configuration

-Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true when you run JDBCExampleZk and JDBCExamplePasswordZK.



Step 4 Right-click the ***.java** file of the sample project and choose **Run'*.main()'** from the shortcut menu. Wait until the execution is successful. (The default sample is to query a Hive table.)

- The running result of the JDBCExampleZk example application is shown as follows:

```
...
principal is hivetest@HADOOP.COM
Will use keytab
Commit Succeeded
...
The final connection url is: XXX://192.168.1.189:29896/hive/default
Table
user_info
user_info2
```

- The running result of the JDBCExamplePasswordZK example application is as follows:

```
...
The final connection url is: XXX://192.168.1.189:29896/hive/default
Table
user_info
user_info2
```

- The running result of the JDBCExampleBroker example application is as follows:

```
...
The final connection url is: XXX://192.168.1.189:29896/hive/default
coordinator uri is XXX:/192.168.1.189:29896/hive/default
user_info
user_info2
```

----End

16.4.2 Commissioning Applications on Linux

Scenario

After the code is developed, you can compile the code into a JAR file and upload it to a Linux environment for application function commissioning.

NOTE

Before commissioning applications on Linux, you need to pre-install a client on the Linux node.

Procedure

Step 1 Change the path for storing the KeyTab file in the **jaas-zk.conf** file in the Linux environment as required. for example, **/opt/client/conf/user.keytab**.

Step 2 Modify the configuration file path of the sample code, as shown in the following example:

```
private final static String PATH_TO_KRB5_CONF = "/opt/client/krb5.conf"
```

Step 3 In the Windows development environment IntelliJ IDEA, choose **Maven Projects** > *Example project name* > **Lifecycle** and perform the clean and package operations. After the compilation is complete, the **hetu-examples-*XXX*.jar** file is generated in the **target** directory.

Step 4 Upload the **hetu-examples-*XXX*.jar** file to the **/opt/client** directory in the Linux environment.

Step 5 Download and decompress the client file **FusionInsight_Cluster**_*Cluster ID*_**HetuEngine_Client.tar** by referring to **Preparing Development Environment**, obtain the JDBC drive package, and upload it to the **/opt//client** directory in the Linux environment.

NOTE

The JDBC driver package **hetu-jdbc-*.jar** can be obtained from the **FusionInsight_Cluster_1_Services_ClientConfig\HetuEngine*XXX*** directory where the cluster client software package is decompressed. In the directory, *XXX* indicates ARM or x86.

Step 6 Upload the **jaas-zk.conf**, **user.keytab**, and **krb5.conf** files obtained in **Preparing for Security Authentication** to the **/opt/client** directory in the Linux environment.

Step 7 Run the following command to go to the client installation directory:

cd /opt/client

Step 8 Run the following command to configure environment variables:

source bigdata_env

Step 9 Run the following command to debug the development program:

java -classpath hetu-examples-*.jar:hetu-jdbc-*.jar com.huawei.hetu.*className*

NOTE

- Replace the JDBC driver package name and class name with the actual ones, for example, **java -classpath hetu-examples-*.jar:hetu-jdbc-*.jar com.huawei.hetu.JDBCExampleBroker**.
- If SSL authentication of ZooKeeper is enabled for the interconnected cluster, you need to add the following JVM configuration

  -Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true.

  Using JDBCExampleZK as an example, the corresponding debugging command is as follows: **java -cp hetu-examples-*.jar:hetu-jdbc-*.jar -Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true com.huawei.hetu.JDBCExampleZK**.

Step 10 Check whether the command output is normal.

```
Jul 01, 2021 8:41:23 PM io.XXXsql.jdbc.$internal.airlift.log.Logger info
INFO: hsbroker finalUri is https://192.168.1.150:29860
Jul 01, 2021 8:41:24 PM io.XXXsql.jdbc.$internal.airlift.log.Logger info
INFO: The final connection url is: XXX://192.168.1.189:29896/hive/default
Jul 01, 2021 8:41:24 PM io.XXXsql.jdbc.$internal.airlift.log.Logger info
INFO: coordinator uri is XXX://192.168.1.189:29896/hive/default
user_info
user_info2
```

----End

16.4.3 Commissioning the Python3 Sample Project

This section applies to MRS 3.3.0 or later.

Scenario

After the program code is written, you can commission the code in the Windows environment or upload the code to the Linux environment. If the local environment can communicate with the cluster service plane network, you can commission the program locally.

Procedure

1. Refer to **Configuring the Python3 Sample Project** to obtain the sample code, obtain the **hetu-jdbc-XXX.jar** file, and copy it to the user-defined directory.

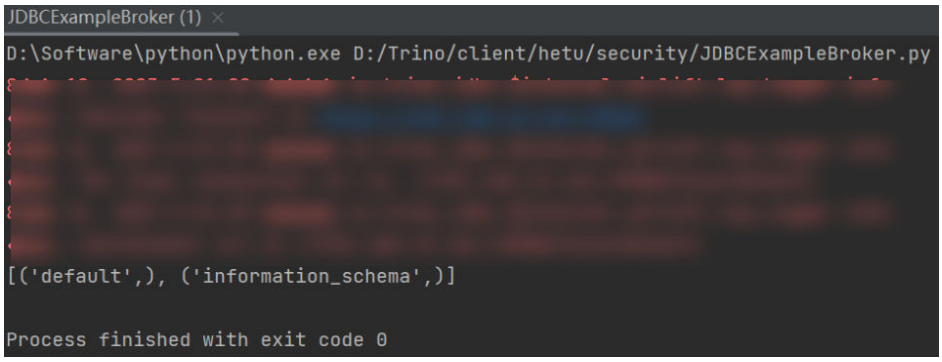
2. Refer to **KeyTab File Authentication Using HSFabric** to obtain the **user.keytab** and **krb5.conf** files and save them to the customized directory.
3. Edit the sample code, modify URLs, user information, and passwords based on the cluster requirements, and modify **jdbc_location** based on the actual path.
   - Example path in Windows: **D:\\hetu-examples-python3\\hetu-jdbc-XXX.jar**
   - Example path in Linux: **/opt/hetu-examples-python3/hetu-jdbc-XXX.jar**
4. Run the python3 sample code.
   - In Windows, run the **py** script through pycharm or Python IDLE.
   - To run the sample code on Linux, install Java first.

     Go to the sample code path and run the **py** script. An example of the sample code path is **/opt/hetu-examples-python3**.

     cd /opt/hetu-examples-python3

     python3 JDBCExampleBroker.py
5. View the command output.
   - In Windows, view the running result on the console.

     Figure 16-4 Running result

   - The following is an example of the running result in Linux:

```
Aug 12, 2023 5:19:53 PM io.xxx.jdbc.$internal.airlift.log.Logger info
INFO: hsbroker finalUri is https://192.168.43.223:29860
Aug 12, 2023 5:19:53 PM io.xxx.jdbc.$internal.airlift.log.Logger info
INFO: The final connection url is: //192.168.43.161:29888/hive/default
Aug 12, 2023 5:19:53 PM io.xxx.jdbc.$internal.airlift.log.Logger info
INFO: coordinator uri is //192.168.43.161:29888/hive/default
[('default',), ('information_schema',)]
```


17 HetuEngine Development Guide (Normal Mode)

17.1 Overview

17.1.1 Introduction to HetuEngine

Introduction to HetuEngine

HetuEngine is a high-performance, interactive SQL analysis and data virtualization engine developed by Huawei. It seamlessly integrates with the big data ecosystem to implement interactive query of massive amounts of data within seconds, and supports cross-source and cross-domain unified data access to enable one-stop SQL convergence analysis in the data lake, between lakes, and between lakehouses.

17.1.2 Concepts

Basic Concepts

- HSBroker: the HetuEngine proxy, which is used for tenant management verification and obtaining the URL for accessing HetuEngine
- Coordinator: the coordinator of HetuEngine services, responsible for SQL parsing and optimization
- Worker: responsible for task execution and data processing
- Connector: an API for HetuEngine to access the database. Through the connector driver, HetuEngine connects to data sources, reads data source metadata, and operates data (adding, deletion, modification, and query).
- Catalog: the catalog configuration file corresponds to a data source in HetuEngine. A data source can have multiple catalog configurations, which can be configured in the **properties** file of a data source.
- Schema: corresponds a schema in a database.
- Table: corresponds to a table name is a database.

17.1.3 Connection Modes

Connection Mode	Support for Username/Password Authentication	Support for Keytab Authentication	Support for Cross-Network-Segment Client Access	Prerequisite
HSFabric	Supported	Supported	Supported	• The node running user services can communicate with the service nodes where HSFabric resides on the HetuEngine server side. • Dual-plane network scenarios are supported. • Only fixed IP addresses and ports need to be opened for HSFabric. • Supported version: MRS 3.1.3 or later.
HSBroker	Supported	Unsupported	Unsupported	• The node running user services can communicate with the service nodes where HSBroker and Coordinator (randomly distributed in Yarn NodeManger) reside on the HetuEngine server side. • A large number of IP addresses and ports need to be opened for coordinators. • Supported version: MRS 3.1.0 or later

17.1.4 Development Process

This section describes the development process, as shown in **Figure 17-1**.

Figure 17-1 HetuEngine application development process

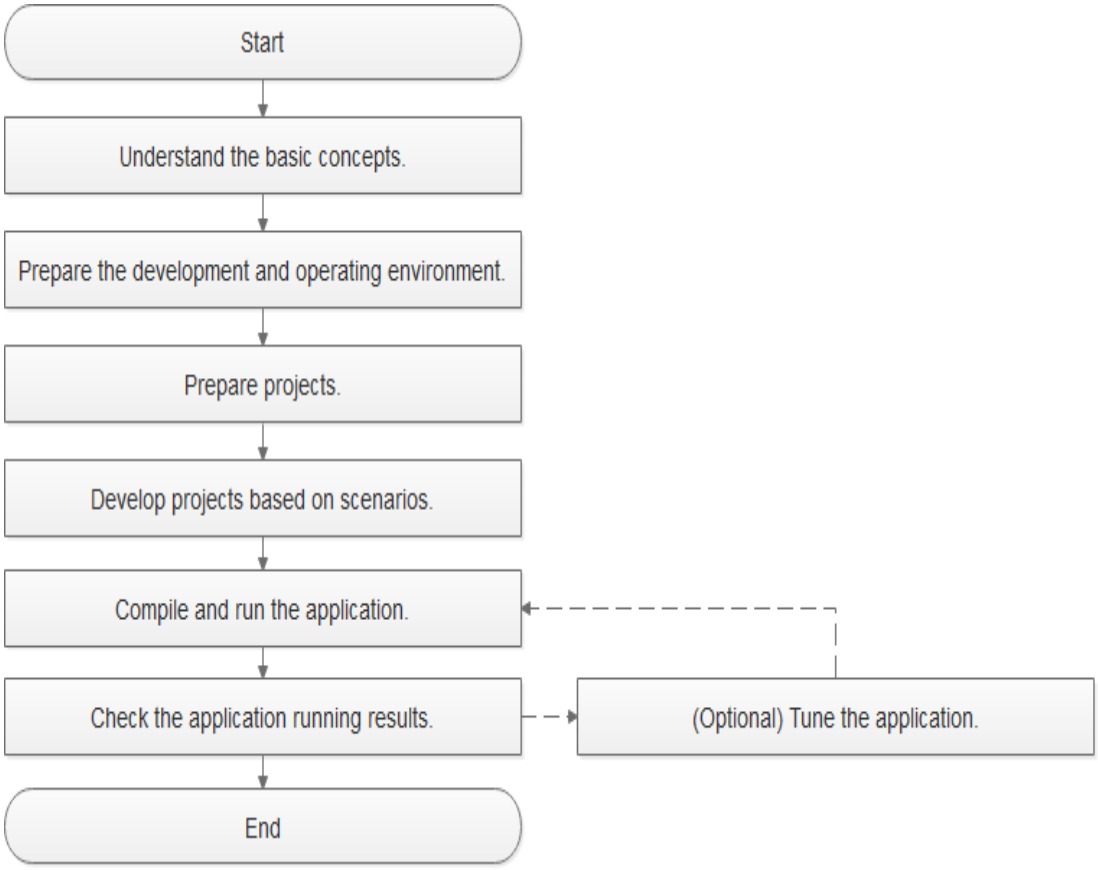


Table 17-1 Description of the HetuEngine application development process

Phase	Description	Reference Documents
Understand basic concepts.	Before application development, learn basic concepts of HetuEngine, and understand the scenario requirements.	**Concepts**
Prepare the development and running environment.	The HetuEngine application can invoke the JDBC interface in any language for development. The current example uses the Java language. You are advised to use the IDEA tool to configure development environments in different languages according to the guide. The HetuEngine running environment is a client. Install and configure the client according to the guide.	**Preparing Development and Running Environments**
Prepare a project.	HetuEngine provides sample projects for different scenarios. You can import a sample project to learn the application. You can also create a HetuEngine project according to the guide.	**Configuring and Importing a Sample Project**

Phase	Description	Reference Documents
Develop a project based on the scenario.	A sample project in the Java language is provided, including an example project that connects the HetuEngine, SQL statement execution, result parsing, and disconnection.	Application Development
Compile and run applications.	You can compile the developed application and submit it for running.	Application Commissioning
Check the application running results.	The program running result is displayed in the expected display according to the implementation of the result parsing part.	Application Commissioning

17.2 Preparing Environment

17.2.1 Preparing Development and Running Environments

Table 17-2 Python3 environment (required when the Python sample project is used) for MRS 3.3.0 and later versions

Item	Description
Python3 (for MRS 3.3.0 or later)	Tool used to develop HetuEngine Python applications. The version must range from 3.6 to 3.9.
Setuptools	Basic configuration of the Python3 development environment. Version: 47.3.1
jaydebeapi	Basic configuration of the Python3 development environment. You can use this module to connect to the database using Java JDBC.

Preparing Development Environment

This section describes the development and running environment to be prepared for application development, as listed in **Table 17-3**.

Table 17-3 Development environment

Item	Description
OS	• 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	Basic configuration of the development and running environment. The version requirements are as follows: The server and client support only the built-in OpenJDK. Other JDKs cannot be used. If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows: • For x86 nodes that run clients, use the following JDKs: – Oracle JDK 1.8 – IBM JDK 1.8.0.7.20 and 1.8.0.6.15 • For Arm nodes that run clients, use the following JDKs: – OpenJDK 1.8.0_272 (built-in JDK, which can be obtained from the **JDK** folder in the cluster client installation directory.) – BiSheng JDK 1.8.0_272 **NOTE** • For security purposes, the server supports only TLS V1.2 or later. • 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, V1.1, and V1.2. For details, see **https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls**. • For details about the BiSheng JDK, see **https://www.hikunpeng.com/en/developer/devkit/compiler/jdk**.

Item	Description
Installation and configuration of IntelliJ IDEA	It is the basic configuration for the development environment. IntelliJ IDEA 2019.1 or other compatible versions are used. **NOTE** • 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.
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
7-zip	Used to decompress **.zip** and **.rar** 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\HetuEngine\config** and manually import the configuration file to the configuration file directory of the HetuEngine sample project. For example, **D:\hetuclient\conf** is the **conf** folder.

Table 17-4 describes the main configuration files (Obtain required files as required).

Table 17-4 Configuration files

File	Function
hdfs-site.xml	Configures HDFS parameters.
hetuserver-client.properties	Configures connection parameters for the HetuEngine client.
hetuserver-client-logging.properties	Configures log parameters for the HetuEngine client.

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 FusionInsight Manager. 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/HetuEngine/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 HetuEngine/config/* root@*IP address of the client node***:/opt/client/conf**

Table 17-5 describes the main configuration files (Obtain required files as required).

Table 17-5 Configuration files

File	Function
hdfs-site.xml	Configures HDFS parameters.
hetuserver-client.properties	Configures connection parameters for the HetuEngine client.
hetuserver-client-logging.properties	Configures log parameters for the HetuEngine client.

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 and Importing a Sample Project

Scenario

The HetuEngine client installation program directory contains a HetuEngine development sample project. You can start the sample project learning by importing the project. This document uses IntelliJ IDEA 2020.1.3 (Community Edition) as an example.

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 time of the MRS cluster in the upper-right corner on the FusionInsight Manager page.

Procedure

Step 1 Obtain the sample project folder **hetu-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. Open IntelliJ IDEA and choose **File** > **New** > **General** > **Project from Existing Sources** > **Select File or Directory to Import** to go to the **Browse Folder** dialog box is displayed.
2. Select the sample project folder, choose **Import project from external model** > **Maven** during import, 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.

----End

17.2.3 Configuring the Python3 Sample Project

This section applies to MRS 3.3.0 or later.

Scenario

The following content describes the operations you need to do to run the pyhon3 sample code of HetuEngine on FusionInsight MRS.

Procedure

Step 1 Install Python 3.6 or a later version (before 3.9) on the client node.

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 must be installed on the client.

Download the software from the official website **https://pypi.org/project/setuptools/#files**.

Copy the downloaded setuptools package to the client, decompress the package, go to the setuptools project directory, and run the **python3 setup.py install** command in the CLI of the client.

Take version 47.3.1 as an example, 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 setuptools 47.3.1 fails to be installed, check whether the environment is faulty or Python is faulty.

Step 3 The JayDeBeApi module must be installed on the client. You can use the Java JDBC to connect to the database through this module.

You can install it either of the following ways:

- pip:

  Run the **pip install JayDeBeApi** command on the client node.

- Script:
  a. Download the **JayDeBeApi project** file from the official website at **https://pypi.org/project/JayDeBeApi/**
  b. Go to the **JayDeBeApi** project directory and run the **python3 setup.py install** command. During the installation, if the system displays a message indicating that the python3 module or package is missing, you need to add the module or package.

     Take JayDeBeApi-1.2.3 as an example. If **Successfully installed JayDeBeApi-1.2.3** is displayed, the installation is successful.

Step 4 Install Java on the client. For details about the supported Java versions, see "JDK Installation" in the **Table 16-2**.

Step 5 Obtain the Python3 sample code.

1. Obtain the sample project folder **python3-examples** in the **src\hetu-examples** directory where the sample code is decompressed. For details, see **Obtaining Sample Projects from Huawei Mirrors**.
2. Go to the **python3-examples** folder.
   - **normal** folder: Python3 sample code for interconnecting with HetuEngine in common mode
   - **security** folder: Python3 sample code for interconnecting with HetuEngine in security mode

Step 6 Obtain the **hetu-jdbc** JAR package.

- Download the client file to the local PC through Manager.
  a. Log in to FusionInsight Manager and choose **Cluster** > **Services** > **HetuEngine** > **More** > **Download Client**.
  b. Select **Complete Client**, select the correct platform type (**x86_64** for x86 and **aarch64** for ARM) for the node where the client is to be installed, deselect **Save to Path**, and click **OK**. Wait until the client file package is generated and download it.
  c. Decompress the downloaded software package to obtain the **hetu-jdbc** package and decompress it to obtain the JDBC package.

     For example, if the client file package is **FusionInsight_Cluster_1_Services_Client.tar**, decompress the package to obtain the **FusionInsight_Cluster_1_Services_ClientConfig.tar** file. Then decompress the package to obtain the **FusionInsight_Cluster_1_Services_ClientConfig** file. (The path cannot contain spaces.) Decompress **\FusionInsight_Cluster_1_Services_ClientConfig\HetuEngine\x86\hetu-jdbc.tar.gz** to obtain **hetu-jdbc-XXX.jar** and copy it to the user-defined path on the host where the sample code will run.
- Obtain the cluster client node.

  Log in to the node where the HetuEngine client has been installed. For example, if the client installation path is **/opt/hadoopclient**, obtain **hetu-jdbc-XXX.jar** from **/opt/hadoopclient/HetuEngine/hetuserver/jars/**, and copy the file to the user-defined path on the node where the sample code will run.

----End

17.3 Application Development

17.3.1 Typical Application Scenario

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

Scenario

Assume that a user develops an application and needs to perform the join operation on table A of the Hive data source and table B of the MPPDB data source. In this case, HetuEngine can be used to implement data query of the Hive data source, the process is as follows:

1. Connect to a HetuEngine JDBC server.
2. Assemble SQL statements.
3. Execute SQL statements.
4. Parse the returned result.
5. Disconnect HetuEngine JDBC server.

17.3.2 Java Sample Code

17.3.2.1 Accessing Hive Data Sources Using HSFabric

Description

This section describes how to use HSFabric to connect to HetuServer, assemble SQL statements, and send the SQL statements to HetuServer for execution to add, delete, modify, and query Hive data sources.

```
public class JDBCExampleFabric {
    private static Properties properties = new Properties();
    private static void init() throws ClassNotFoundException {
        properties.setProperty("user", "YourUserName");
        properties.setProperty("SSL", "false");
        Class.forName("io.XXX.jdbc.XXXDriver");
    }
    /**
     * Program entry
     *
     * @param args no need program parameter
     */
    public static void main(String[] args) {
        Connection connection = null;
        ResultSet result = null;
        PreparedStatement statement = null;
        String url = "jdbc:XXX://192.168.1.130:29903,192.168.1.131:29903/hive/default?
serviceDiscoveryMode=hsfabric";
        try {
            init();
            String sql = "show tables";
            connection = DriverManager.getConnection(url, properties);
            statement = connection.prepareStatement(sql.trim());
            result = statement.executeQuery();
            ResultSetMetaData resultMetaData = result.getMetaData();
```



```
            Integer colNum = resultMetaData.getColumnCount();
            for (int j = 1; j <= colNum; j++) {
                System.out.print(resultMetaData.getColumnLabel(j) + "\t");
            }
            System.out.println();
            while (result.next()) {
                for (int j = 1; j <= colNum; j++) {
                    System.out.print(result.getString(j) + "\t");
                }
                System.out.println();
            }
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            if (result != null) {
                try {
                    result.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (statement != null) {
                try {
                    statement.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (connection != null) {
                try {
                    connection.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Table 17-6 describes the parameters in the preceding code.

Table 17-6 Parameter description

Parameter	Description
url	jdbc:*XXX*://*HSFabric1_IP*.*HSFabric1_Port*,*HSFabric2_IP*.*HSFabric2_Port*,*HSFabric3_IP*.*HSFabric3_Port*/*catalog*/*schema*?serviceDiscoveryMode=hsfabric **NOTE** • xxx: driver name, which is subjective to the real-world code you use. • *catalog* and *schema* indicate the names of the catalog and schema to be connected to the JDBC client, respectively. • *HSFabric_IP:HSFabric_Port:* indicates the HSFabric URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:29903,192.168.195.232:29903,192.168.169.84:29903. On FusionInsight Manager, choose **Cluster** > **Services** > **HetuEngine** and click **Instance** to obtain the service IP addresses of all HSFabric instances. On the **Configurations** page, search for **gateway.port** to obtain the port number of HSFabric.

Parameter	Description
user	Username for accessing HetuServer, that is, the username of the machine-machine user created in the cluster.
SSL	Indicates whether to use the HTTPS connection. The default value is **false**.

17.3.2.2 Accessing Hive Data Sources Using HSBroker

Description

This section describes how to use HSBroker to connect to HetuEngine, assemble SQL statements, and send the SQL statements to HetuEngine for execution to add, delete, modify, and query Hive data sources.

```
public class JDBCExampleBroker {
    private static Properties properties = new Properties();
    private static void init() throws ClassNotFoundException {
        properties.setProperty("user", "YourUserName");
        properties.setProperty("SSL", "false");
        Class.forName("io.XXX.sql.jdbc.XXXDriver");
    }
    /**
     * Program entry
     *
     * @param args no need program parameter
     */
    public static void main(String[] args) {
        Connection connection = null;
        ResultSet result = null;
        PreparedStatement statement = null;
        String url = "jdbc:XXX://192.168.1.130:29861,192.168.1.131:29861/hive/default?
serviceDiscoveryMode=hsbroker";
        try {
            init();
            String sql = "show tables";
            connection = DriverManager.getConnection(url, properties);
            statement = connection.prepareStatement(sql.trim());
            result = statement.executeQuery();
            ResultSetMetaData resultMetaData = result.getMetaData();
            Integer colNum = resultMetaData.getColumnCount();
            for (int j = 1; j <= colNum; j++) {
                System.out.print(resultMetaData.getColumnLabel(j) + "\t");
            }
            System.out.println();
            while (result.next()) {
                for (int j = 1; j <= colNum; j++) {
                    System.out.print(result.getString(j) + "\t");
                }
                System.out.println();
            }
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            if (result != null) {
                try {
                    result.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (statement != null) {
                try {
```

```
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
  }
}
```

Table 17-7 describes the parameters in the preceding code.

Table 17-7 Parameter description

Parameter	Description
url	jdbc:*XXX*://*HSBroker1_IP*.*HSBroker1_Port*,*HSBroker2_IP*.*HSBroker2_Port*,*HSBroker3_IP*.*HSBroker3_Port*/*catalog*/*schema*?serviceDiscoveryMode=hsbroker **NOTE** • xxx: driver name, which is subjective to the real-world code you use. • *catalog* and *schema* indicate the names of the catalog and schema to be connected to the JDBC client, respectively. • *HSBroker_IP:HSBroker_Port:* indicates the HSBroker URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181.
user	Username for accessing HetuEngine, that is, the username of the machine-machine user created in the cluster.
SSL	Indicates whether to use the HTTPS connection. The default value is **false**.

17.3.2.3 Querying the Execution Progress and Status of an SQL Statement Using JDBC

Description

This section describes how to use JDBC to connect to HetuEngine, and assemble and send the SQL statements to HetuEngine for execution.

```
import io.XXX.jdbc.XXXResultSet;
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 java.util.Timer;
import java.util.TimerTask;
```



```
public class JDBCExampleStatementProgressPercentage{

   private static Properties properties = new Properties();
   public static Connection connection = null;
   public static ResultSet result = null;
   public static PreparedStatement statement = null;
   private static void init() throws ClassNotFoundException {
      properties.setProperty("user", "YourUserName");
      properties.setProperty("SSL", "false");
      Class.forName("io.XXX.jdbc.XXXDriver");
   }

   /**
    * Program entry
    *
    * @param args no need program parameter
    */
   public static void main(String[] args) {
      String url = "jdbc:XXX://192.168.1.130:29861,192.168.1.131:29861/hive/default?
serviceDiscoveryMode=hsbroker";
      try {
         init();
         String sql = "show tables";
         connection = DriverManager.getConnection(url, properties);
         statement = connection.prepareStatement(sql.trim());
         result = statement.executeQuery();

         XXXResultSet rs = (XXXResultSet) result;
         new Thread() {
            public void run() {
               Timer timer = new Timer();
               //The SQL statement is executed after 3 seconds and is executed every 2 seconds
               timer.schedule(new TimerTask() {
                  @Override
                  public void run() {
                     double statementProgressPercentage = rs.getProgressPercentage().orElse(0.0);
                     System.out.println("The Current Query Progress Percentage is " +
statementProgressPercentage*100 + "%");
                     if("FINISHED".equals(rs.getStatementStatus().orElse(""))) {
                        System.out.println("The Current Query Progress Percentage is 100%");
                        timer.cancel();
                        Thread.currentThread().interrupt();
                     }
                  }
               }, 3000, 2000);
            }
         }.start();

         ResultSetMetaData resultMetaData = result.getMetaData();
         int colNum = resultMetaData.getColumnCount();
         for (int j = 1; j <= colNum; j++) {
            try {
               System.out.print(resultMetaData.getColumnLabel(j) + "\t");
            } catch (SQLException throwables) {
               throwables.printStackTrace();
            }
         }

         while (result.next()) {
            for (int j = 1; j <= colNum; j++) {
               System.out.print(result.getString(j) + "\t");
            }
            System.out.println();
         }
      } catch (SQLException | ClassNotFoundException e) {
         e.printStackTrace();
      } finally {
         if (result != null) {
```



```
            try {
                result.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
  }
}
```

Table 17-8 describes the parameters in the preceding code.

Table 17-8 Parameter description

Parameter	Description
url	jdbc:*XXX*://*HSBroker1_IP*.*HSBroker1_Port*,*HSBroker2_IP*.*HSBroker2_Port*,*HSBroker3_IP*.*HSBroker3_Port*/*catalog*/*schema*?serviceDiscoveryMode=hsbroker **NOTE** • xxx: driver name, which is subjective to the real-world code you use. • *catalog* and *schema* indicate the names of the catalog and schema to be connected to the JDBC client, respectively. • *HSBroker_IP:HSBroker_Port:* indicates the HSBroker URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181.
user	Username for accessing HetuServer, that is, the username of the machine-machine user created in the cluster.
SSL	Indicates whether to use the HTTPS connection. The default value is **false**.
getStatementStatus()	Execution status of an SQL statement, which can be **RUNNING**, **FAILED**, **FINISHED**, **QUEUEED**, **WAITING_FOR_RESOURCES**, **DISPATCHING**, **PLANNING**, **STARTING**, **RESCHEDULING**, **RESUMING**, or **FINISHING.**
getProgressPercentage()	Execution progress of an SQL statement. The value range is 0 to 1.

17.3.3 Python3 Sample Code

17.3.3.1 Accessing Hive Data Sources Using HSBroker

This section applies to MRS 3.3.0 or later.

This section describes how to use HSBroker to connect to HetuEngine, assemble SQL statements, and send the them to HetuEngine for execution to add, delete, modify, and query Hive data sources.

```
import jaydebeapi

driver = "io.XXX.jdbc.XXXDriver"

# need to change the value based on the cluster information
url = "jdbc:XXX://192.168.37.61:29861,192.168.37.62:29861/hive/default?serviceDiscoveryMode=hsbroker"
user = "YourUserName"
tenant = "YourTenant"
jdbc_location = "Your file path of the jdbc jar"

sql = "show catalogs"

if __name__ == '__main__':
    conn = jaydebeapi.connect(driver, url, {"user": user,
                                            "SSL": "false",
                                            "tenant": tenant},
                              [jdbc_location])
    curs = conn.cursor()
    curs.execute(sql)
    result = curs.fetchall()
    print(result)
    curs.close()
    conn.close()
```

The following table describes the parameters in the preceding code.

Table 17-9 Parameter description

Parameter	Description
url	jdbc:*XXX*://*HSBroker1_IP*:*HSBroker1_Port*,*HSBroker2_IP*:*HSBroker2_Port*,*HSBroker3_IP*:*HSBroker3_Port*/*catalog*/*schema*?serviceDiscoveryMode=hsbroker **NOTE** • xxx: driver name, which is subjective to the real-world code you use. • *catalog* and *schema* indicate the names of the catalog and schema to be connected to the JDBC client, respectively. • *HSBroker_IP:HSBroker_Port* indicates the HSBroker URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181.
user	Username for accessing HetuServer, that is, the username of the human-machine user created in the cluster.
tenant	Tenant resource queue for accessing HetuEngine compute instances

Parameter	Description
jdbc_location	Full path of the **hetu-jdbc-XXX.jar** package obtained in **Configuring the Python3 Sample Project**. • Example path in Windows: **D:\\hetu-examples-python3\\hetu-jdbc-XXX.jar** • Example path in Linux: **/opt/hetu-examples-python3/hetu-jdbc-XXX.jar**

17.3.3.2 Accessing Hive Data Sources Using HSFabric

This section describes how to use HSFabric to connect to HetuEngine, assemble SQL statements, and send the them to HetuEngine for execution to add, delete, modify, and query Hive data sources.

```
import jaydebeapi

driver = "io.XXX.jdbc.XXXDriver"

# need to change the value based on the cluster information
url = "jdbc:XXX://192.168.37.61:29903,192.168.37.61:29903/hive/default?serviceDiscoveryMode=hsfabric"
user = "YourUserName"
tenant = "YourTenant"
jdbc_location = "Your file path of the jdbc jar"

sql = "show catalogs"

if __name__ == '__main__':
    conn = jaydebeapi.connect(driver, url, {"user": user,
                                            "SSL": "false",
                                            "tenant": tenant},
                              [jdbc_location])
    curs = conn.cursor()
    curs.execute(sql)
    result = curs.fetchall()
    print(result)
    curs.close()
conn.close()
```

The following table describes the parameters in the preceding code.

Table 17-10 Parameter description

Parameter	Description
url	jdbc:*XXX*://*HSFabric1_IP*.*HSFabric1_Port*,*HSFabric2_IP*.*HSFabric2_Port*,*HSFabric3_IP*.*HSFabric3_Port*/*catalog*/*schema*?serviceDiscovery-Mode=hsfabric **NOTE** • xxx: driver name, which is subjective to the real-world code you use. • **catalog** and **schema** indicate the names of the catalog and schema to be connected to the JDBC client, respectively. • **HSFabric_IP:HSFabric_Port** indicates the HSFabric URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:29902,192.168.195.232:29902,192.168.169.84:29902. On FusionInsight Manager, choose **Cluster** > **Services** > **HetuEngine** and click **Instance** to obtain the service IP addresses of all HSFabric instances. On the **Configurations** page, search for **gateway.port** to obtain the port number of HSFabric.
user	Username for accessing HetuServer, that is, the username of the human-machine user created in the cluster.
tenant	Tenant resource queue for accessing HetuEngine compute instances
jdbc_location	Full path of the **hetu-jdbc-XXX.jar** package obtained in **Configuring the Python3 Sample Project**. • Example path in Windows: **D:\\hetu-examples-python3\\hetu-jdbc-XXX.jar** • Example path in Linux: **/opt/hetu-examples-python3/hetu-jdbc-XXX.jar**

17.4 Application Commissioning

17.4.1 Commissioning Applications on Windows

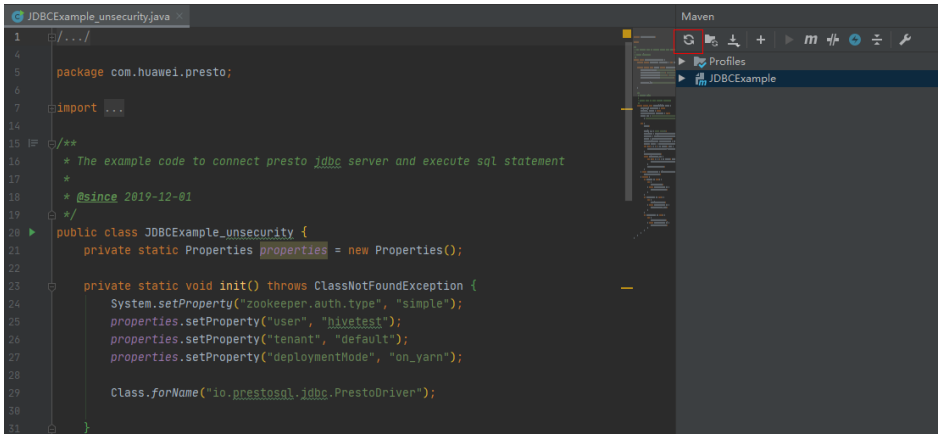
Scenario

After the program code is developed, you can compile the program in the Windows environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

Procedure

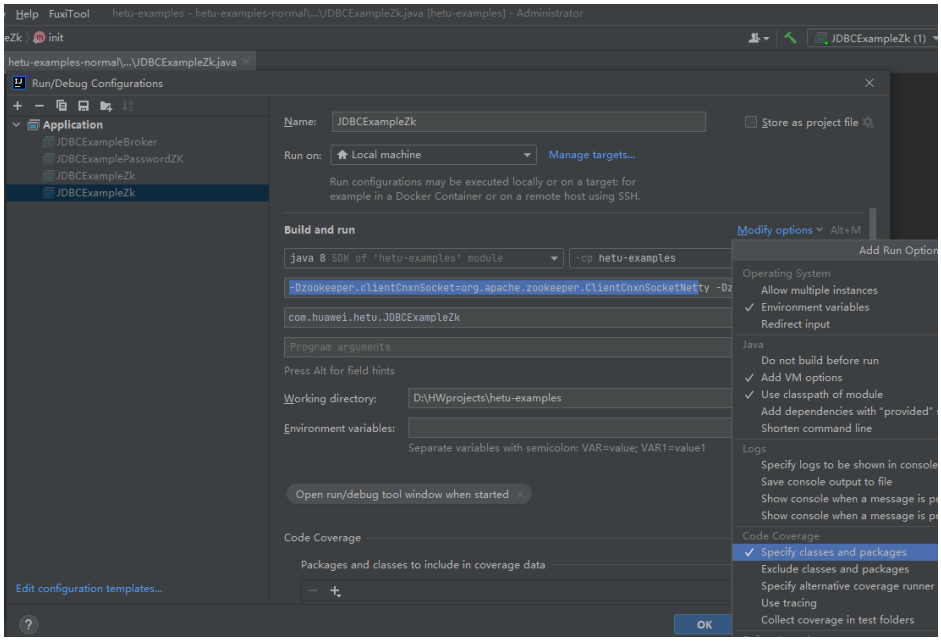
Step 1 In the IntelliJ IDEA development environment on Windows, click **Maven** on the right of IDEA to import the dependency.

Figure 17-2 Importing the dependency



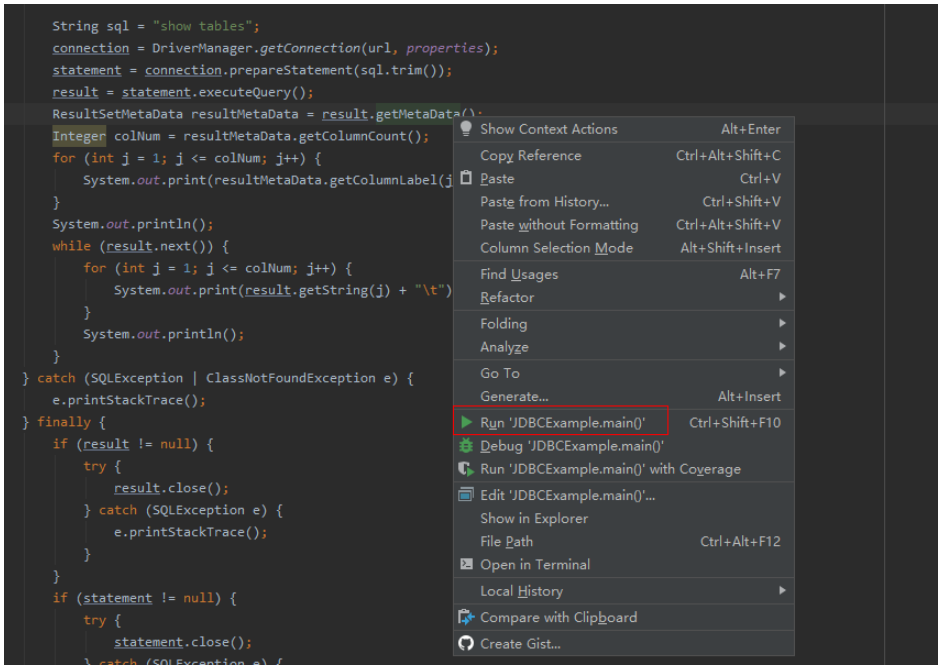
Step 2 (Optional) If the SSL authentication communication function of ZooKeeper is enabled for the interconnected cluster, add the JVM configuration

-Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true.



Step 3 Right-click the **JDBCExampleZK.java** file and choose **Run'JDBCExampleZK.main()'** from the shortcut menu.

Figure 17-3 Running the program



You can view the output result on the console of the IDEA.

Figure 17-4 Outputs

----End

17.4.2 Commissioning Applications on Linux

Scenario

After the code is developed, you can compile the code into a JAR file and upload it to a Linux environment for application function commissioning.

NOTE

Before commissioning applications on Linux, you need to pre-install a client on the Linux node.

Procedure

Step 1 In the Windows development environment IntelliJ IDEA, choose **Maven Projects** > **JDBCExample** > **Lifecycle** and perform the **clean** and **package** operations. After the compilation is complete, the **JDBCExample-1.0-SNAPSHOT.jar** file is generated in the target directory.

Step 2 Run the following command to create the running directory in Linux:

mkdir -p /opt/hetuclient;

Step 3 Upload the **JDBCExample-1.0-SNAPSHOT.jar** package to the **/opt/hetuclient** directory in Linux.

Step 4 Download and decompress the client file **FusionInsight_Cluster**_*Cluster ID*_**HetuEngine_Client.tar** by referring to **Preparing an Operating Environment**, obtain the JDBC drive package, and upload it to the **/opt/hetuclient** directory in the Linux environment.

NOTE

To obtain the JDBC driver package:

Obtain the **hetu-jdbc-*.jar** file from the **FusionInsight_Cluster**_*Cluster ID*_**HetuEngine_ClientConfig\HetuEngine\xxx** directory.

Note: *xxx* can be **arm** or **x86**.

Step 5 Run the following command to go to the client installation directory:

cd /opt/client;

Step 6 Run the following command to configure environment variables:

source bigdata_env;

Step 7 Run the following command to debug the development program:

cd /opt/hetuclient;

java -classpath JDBCExample-*.jar:hetu-jdbc-*.jar com.huawei.hetu.*className*

NOTE

- Replace the JDBC driver package name and class name with the actual ones, for example, **java -classpath JDBCExample-*.jar:hetu-jdbc-*.jar com.huawei.hetu.JDBCExampleZk**.
- If SSL authentication of ZooKeeper is enabled for the interconnected cluster, you need to add the following JVM configuration

  -Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true.

  The corresponding debugging command is as follows: **java -cp JDBCExample-*.jar:hetu-jdbc-*.jar -Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true com.huawei.hetu.JDBCExampleZK**

  .

Step 8 Check whether the command output is normal.


```
.......
Table
testtable
```

----End

17.4.3 Commissioning the Python3 Sample Project

This section applies to MRS 3.3.0 or later.

Scenario

After the program code is written, you can commission the code in the Windows environment or upload the code to the Linux environment. If the local environment can communicate with the cluster service plane network, you can commission the program locally.

Procedure

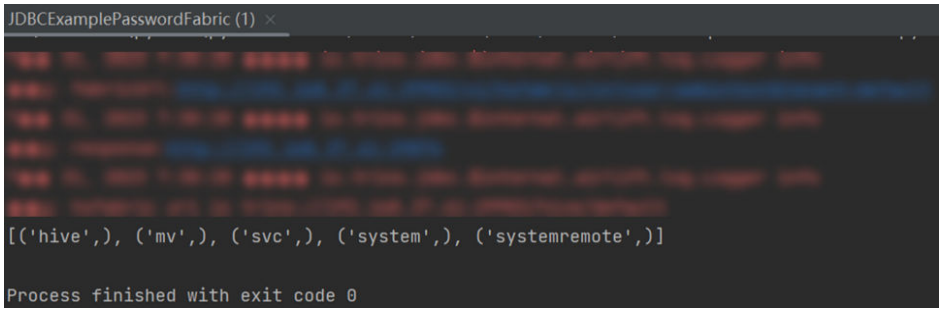
1. Refer to **Configuring the Python3 Sample Project** to obtain the sample code, obtain the **hetu-jdbc-XXX.jar** file, and copy it to the user-defined directory.
2. Edit the sample code, modify URLs and user information based on the cluster requirements, and modify **jdbc_location** based on the actual path.
   - Example path in Windows: **D:\\hetu-examples-python3\\hetu-jdbc-XXX.jar**
   - Example path in Linux: **/opt/hetu-examples-python3/hetu-jdbc-XXX.jar**
3. Run the python3 sample code.
   - In Windows, run the **py** script through pycharm or Python IDLE.
   - To run the sample code on Linux, install Java first.

     Go to the sample code path and run the **py** script. An example of the sample code path is **/opt/hetu-examples-python3**.

     cd /opt/hetu-examples-python3

     python3 JDBCExampleBroker.py
4. View the command output.
   - In Windows, view the running result on the console.

     Figure 17-5 Running result

   - The following is an example of the running result in Linux:

     ```
     ...
     [ ('hive', ('mv',), ('svc',), ('system',), ('systemremote',)]
     ```


18 Hive Development Guide (Security Mode)

18.1 Overview

18.1.1 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, ORCFILE, 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**.

18.1.2 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.

18.1.3 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 18-1** describes the permission required for each operation. To run example programs, you must have the create permission for the default database.

Table 18-1 Required permissions

Operation Type/ Functional Object	Operation	Required Permission
DATABASE	CREATE DATABASE *dbname* [LOCATION "hdfs_path"]	If the HDFS path **hdfs_path** is specified, the ownership and RWX permission of **hdfs_path** are required.
	DROP DATABASE *dbname*	The database **dbname** ownership is required.
	ALTER DATABASE *dbname* SET OWNER *user_or_role*	The admin permission is required.

Operation Type/ Functional Object	Operation	Required Permission
TABLE	CREATE TABLE *table_a*	The create permission for the database is required.
	CREATE TABLE *table_a* AS SELECT *table_b*	The create permission for the database and the select permission for **table_b** are required.
	CREATE TABLE *table_a* LIKE *table_b*	The create permission for the database is required.
	CREATE [EXTERNAL] TABLE *table_a* LOCATION "hdfs_path"	The create permission for the database, and the ownership and RWX permission of **hdfs_path** on HDFS are required.
	DROP TABLE *table_a*	The ownership of **table_a** is required.
	ALTER TABLE *table_a* SET LOCATION "hdfs_path"	The ownership of **table_a**, and the ownership and RWX permission of **hdfs_path** on HDFS are required.
	ALTER TABLE *table_a* SET *FILEFORMAT*	The ownership of **table_a** is required.
	TRUNCATE TABLE *table_a*	The ownership of **table_a** is required.
	ANALYZE TABLE *table_a* COMPUTE STATISTICS	The select and insert permission for **table_a** is required.
	SHOW TBLPROPERTIES *table_a*	The select permission for **table_a** is required.
	SHOW CREATE TABLE *table_a*	The select permission with grant option for **table_a** is required.
Alter	ALTER TABLE *table_a* ADD COLUMN	The ownership of **table_a** is required.
	ALTER TABLE *table_a* REPLACE COLUMN	The ownership of **table_a** is required.
	ALTER TABLE *table_a* RENAME	The ownership of **table_a** is required.
	ALTER TABLE *table_a* SET SERDE	The ownership of **table_a** is required.

Operation Type/ Functional Object	Operation	Required Permission
	ALTER TABLE *table_a* CLUSTER BY	The ownership of **table_a** is required.
PARTITION	ALTER TABLE *table_a* ADD PARTITION *partition_spec* LOCATION "hdfs_path"	The insert permission for **table_a**, and the ownership and RWX permission of **hdfs_path** on HDFS are required.
	ALTER TABLE *table_a* DROP PARTITION *partition_spec*	The delete permission for **table_a** is required.
	ALTER TABLE *table_a* PARTITION *partition_spec* SET LOCATION "hdfs_path"	The ownership of **table_a**, and the ownership and RWX permission of **hdfs_path** on HDFS are required.
	ALTER TABLE *table_a* PARTITION *partition_spec* SET FILEFORMAT	The ownership of **table_a** is required.
LOAD	LOAD INPATH 'hdfs_path' INTO TABLE *table_a*	The insert permission for **table_a**, and the ownership and RWX permission of **hdfs_path** on HDFS are required.
INSERT	INSERT TABLE *table_a* SELECT FROM *table_b*	The update permission for **table_a** and select permission for **table_b** are required.
SELECT	SELECT * FROM *table_a*	The select permission for **table_a** is required.
	SELECT FROM *table_a* JOIN *table_b*	The select permission for **table_a** and **table_b, t**he Submit permission of the default Yarn queue is required.
	SELECT FROM (SELECT FROM *table_a* UNION ALL SELECT FROM *table_b*)	The select permission for **table_a** and **table_b, t**he Submit permission of the default Yarn queue is required.

Operation Type/ Functional Object	Operation	Required Permission
EXPLAIN	EXPLAIN [EXTENDED| DEPENDENCY] *query*	The RX permissions for related table directory is required.
VIEW	CREATE VIEW *view_name* AS SELECT ...	The select permission with grant option for related tables is required.
	ALTER VIEW *view_name* RENAME TO *new_view_name*	The ownership of **view_name** is required.
	DROP VIEW *view_name*	The ownership of **view_name** is required.
INDEX	CREATE INDEX *index_name* ON TABLE *base_table_name* (*col_name*, ...) AS *index_type*	The ownership of **table_a** is required.
	DROP INDEX *index_name* ON *table_name*	The ownership of **index_name** is required.
	ALTER INDEX *index_name* ON *table_name* REBUILD	The ownership of **index_name** is required.
FUNCTION	CREATE [TEMPORARY] FUNCTION *function_name* AS 'class_name'	The admin permission is required.
	DROP [TEMPORARY] *function_name*	The admin permission is required.
MACRO	CREATE TEMPORARY MACRO *macro_name* ...	The admin permission is required.
	DROP TEMPORARY MACRO *macro_name*	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**.

18.1.4 Development Process

Figure 18-1 and Table 18-2 illustrates each phase of the development process.

Figure 18-1 Hive application development process

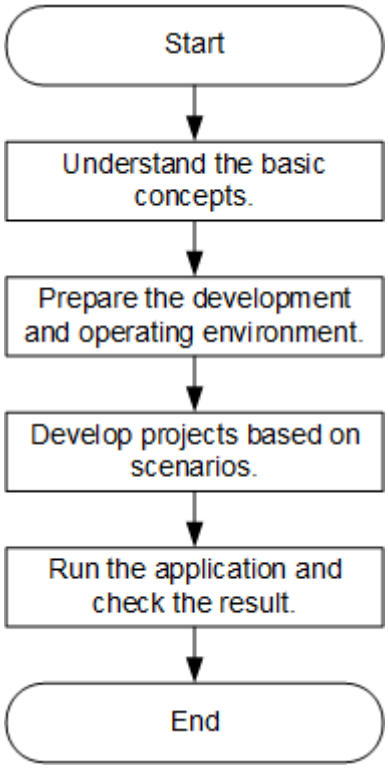


Table 18-2 Description of the Hive application development process

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**

18.2 Preparing the Environment

18.2.1 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 18-3 JDBC development environment

Item	Description
OS	• 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	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. For x86 nodes that run clients, the following JDKs can be used: • 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: • OpenJDK: 1.8.0_272 **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 com.ibm.jsse2.overrideDefaultTLS 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.**
IntelliJ IDEA installation and configuration	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. **NOTE** • 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	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.
Developer account preparation	See **Preparing the Developer Account** for configuration.
7-zip	Used to decompress **.zip** and **.rar** packages. The 7-Zip 16.04 is supported.

Table 18-4 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 the 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 18-5 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 the 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. Download and decompress the client software package.

   - **Log in to the FusionInsight Manager portal** and choose **Cluster** > **Dashboard** > **More** > **Download Client**. Set **Select Client Type** to **Configuration Files Only** 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 the Developer Account** is also stored in this directory. **Table 18-6** describes the main configuration files.

   Table 18-6 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 the Developer Account** is also stored in this directory. **Table 18-7** describes the main configuration files.

   Table 18-7 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.

18.2.2 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 the 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 the 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.

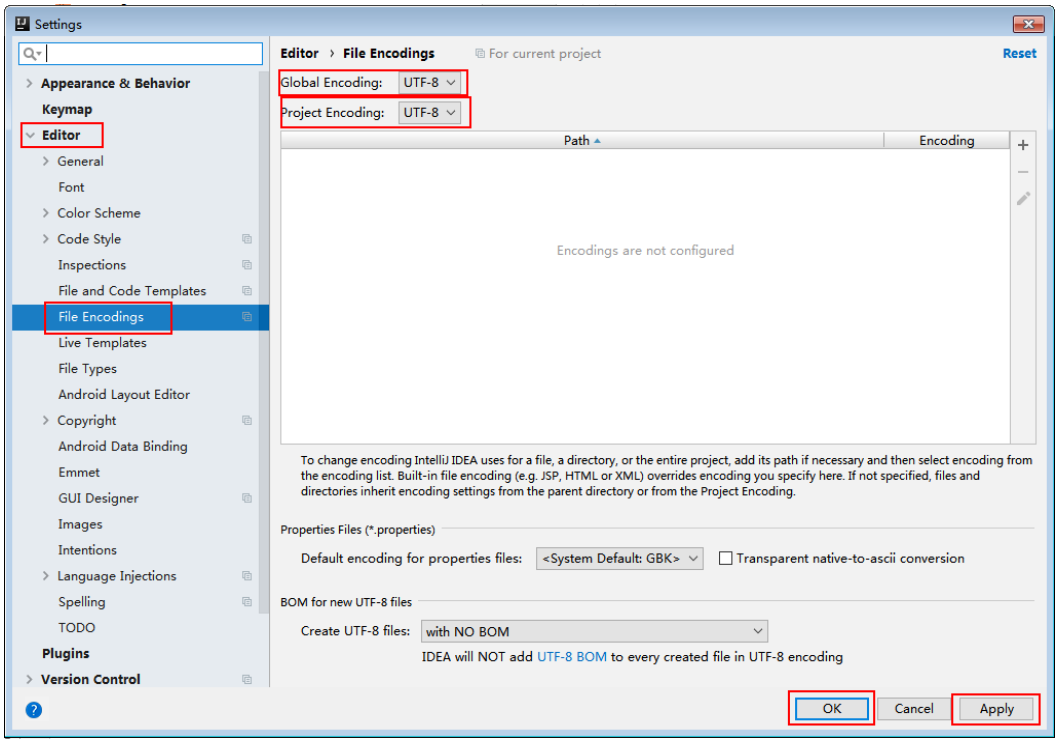
1. On the IntelliJ IDEA menu bar, choose **File** > **Open...**. The **Open File or Project** window is displayed.
2. In the displayed window, select the folder **hive-jdbc-example**, and click **OK**. On Windows, the path cannot contain any space.

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

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

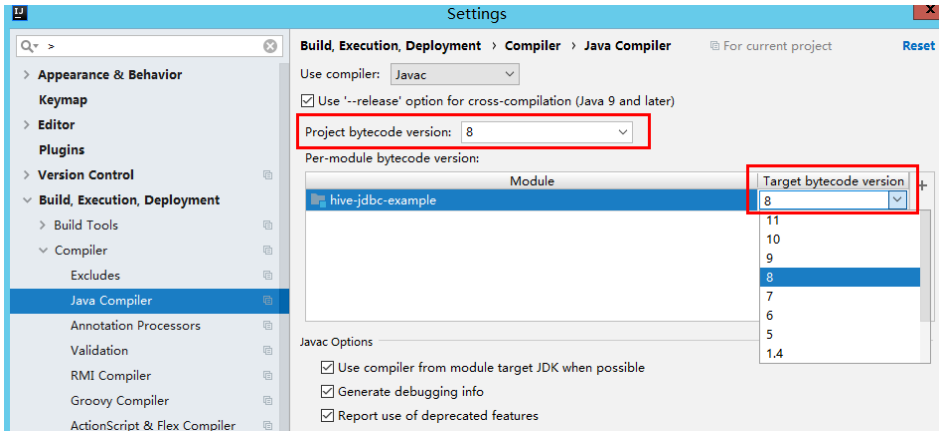
   The **Settings** window is displayed.
2. In the navigating tree on the left, choose **Editor** > **File Encodings**. In the **Project Encoding** area and **Global Encoding** area, set the value to **UTF-8**, and click **Apply** and **OK**, as shown in **Figure 18-2**.

Figure 18-2 Setting the IntelliJ IDEA coding format

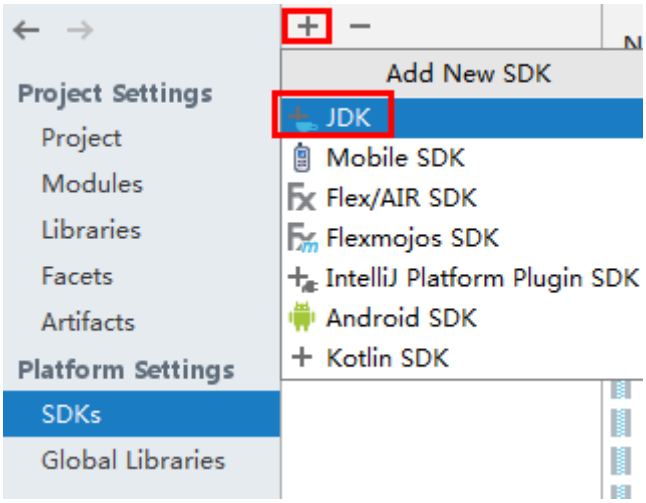


Step 6 Set the JDK of the project.

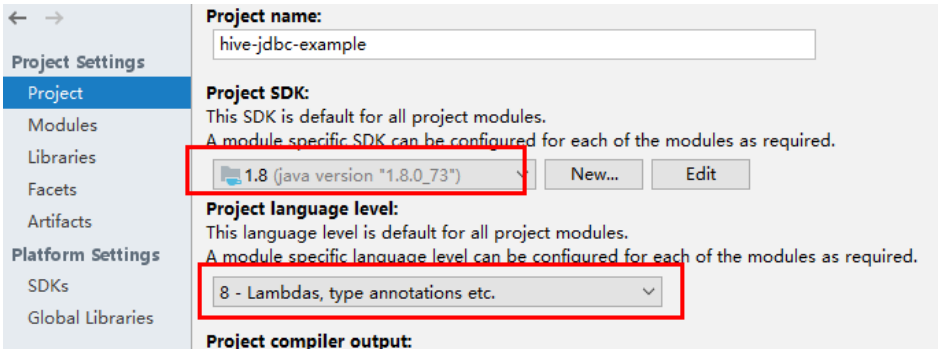
1. On the IntelliJ IDEA menu bar, choose **File** > **Settings...**. The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment** > **Compiler** > **Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **hive-jdbc-example**.



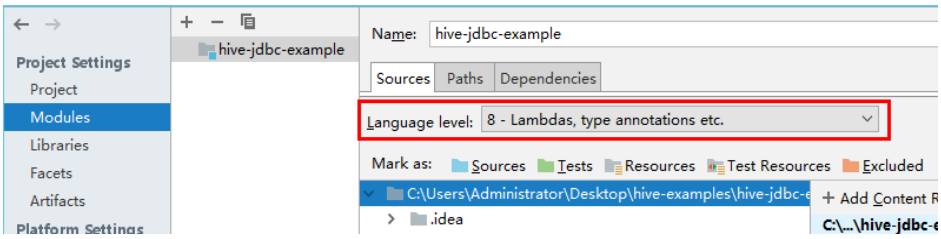
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File** > **Project Structure...**. The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



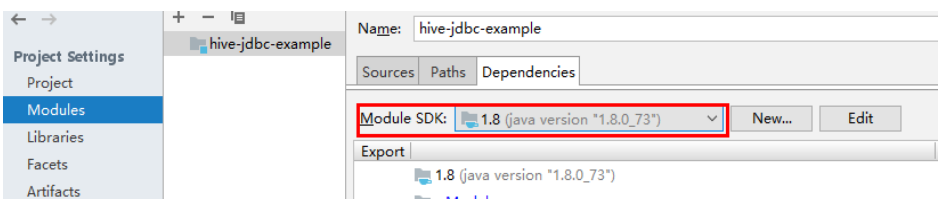
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** 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.

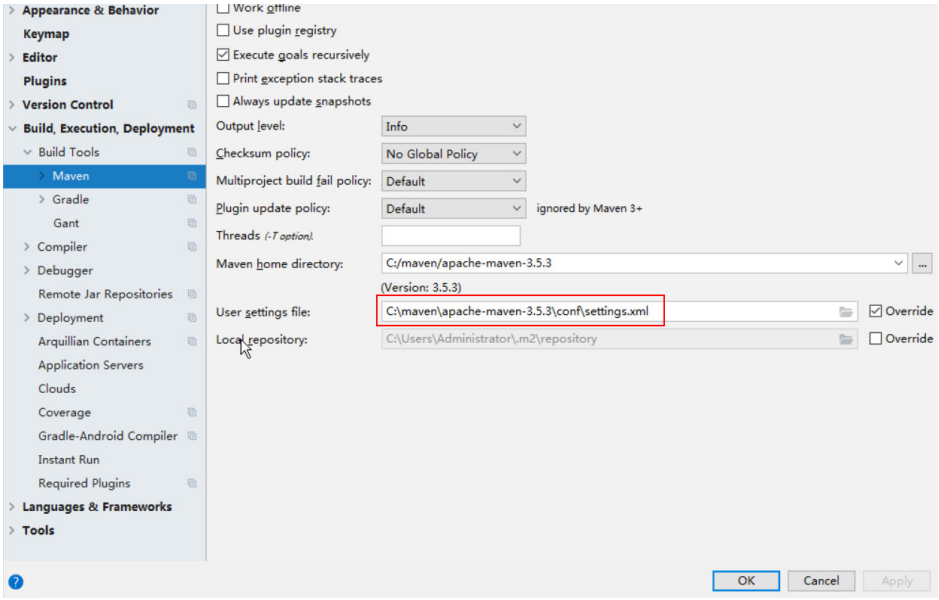


11. Click **Apply** and **OK**.

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 18-3 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

18.2.3 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**.

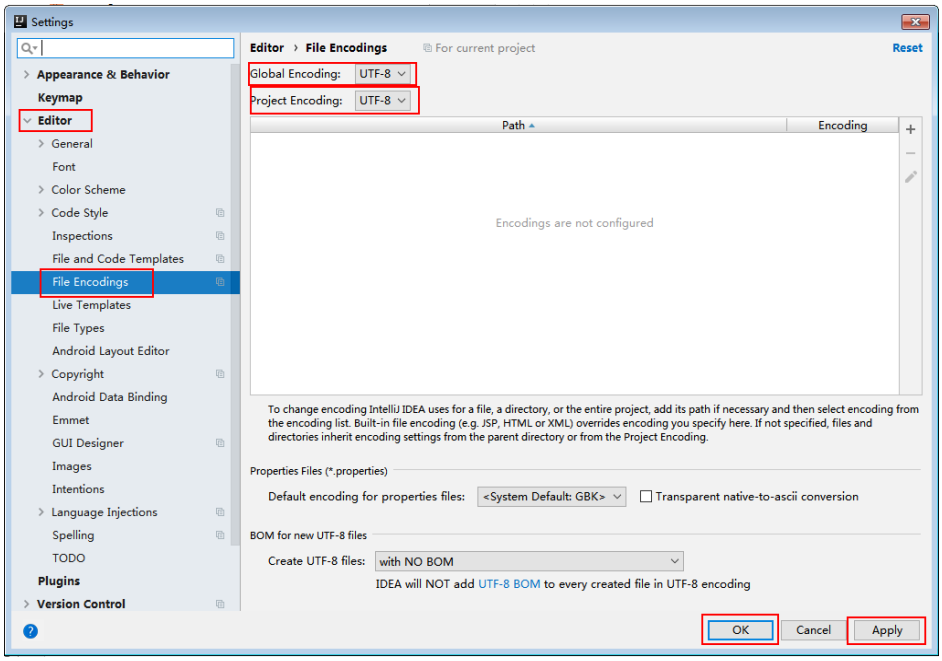
Step 2 Import the example project to the IntelliJ IDEA development environment.

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.

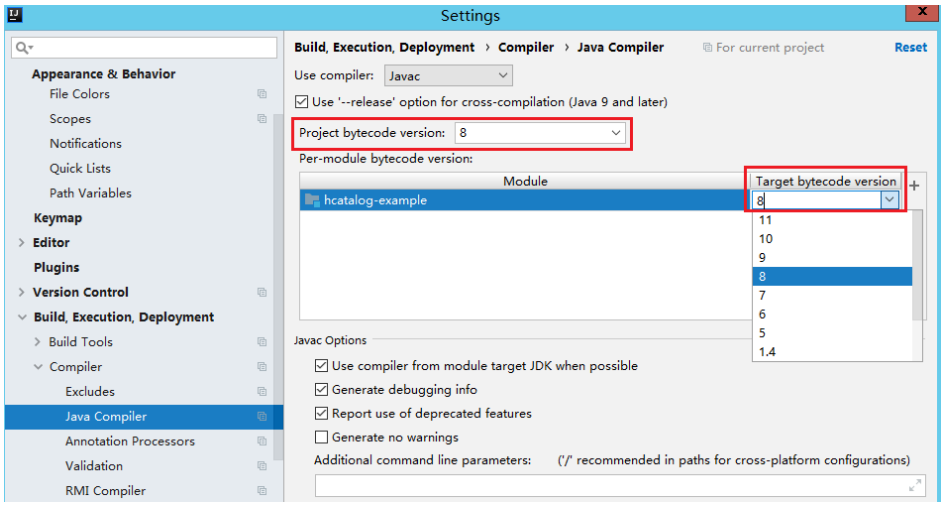
1. On the IntelliJ IDEA menu bar, choose **File** > **Settings**. The **Settings** window is displayed.
2. Choose **Editor** > **File Encodings** from the navigation tree. In the **Project Encoding** area and **Global Encoding** area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in **Figure 18-4**.

Figure 18-4 Setting the IntelliJ IDEA coding format

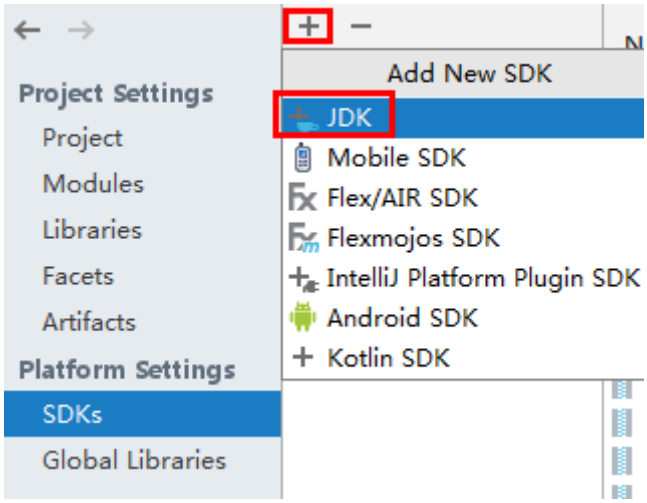


Step 4 Set the JDK of the project.

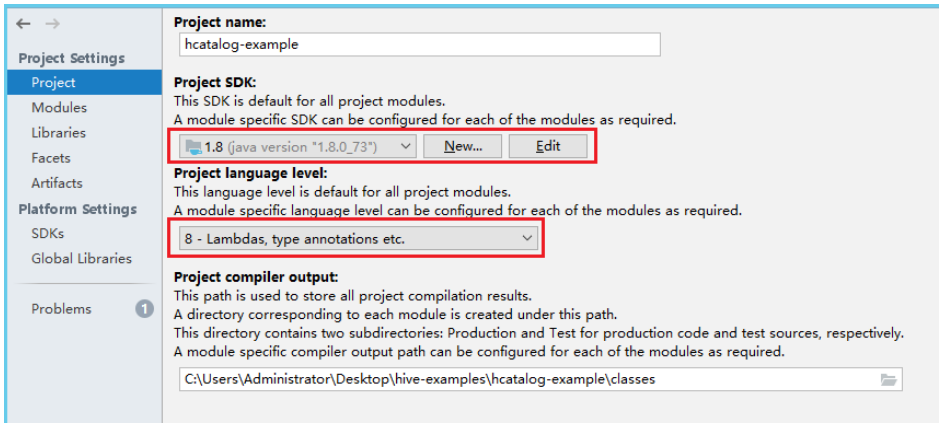
1. On the IntelliJ IDEA menu bar, choose **File** > **Settings...**. The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment** > **Compiler** > **Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **hive-jdbc-example**.



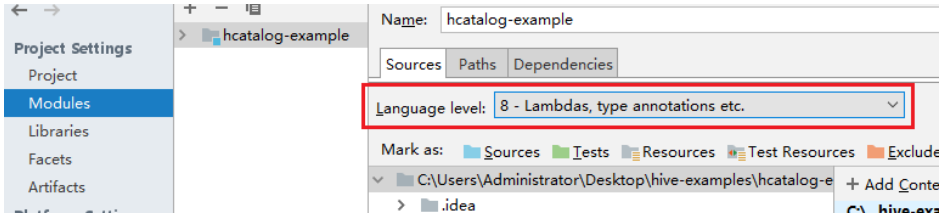
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File** > **Project Structure...**. The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



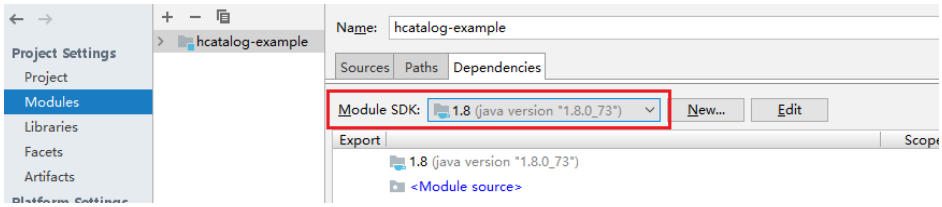
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** 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.

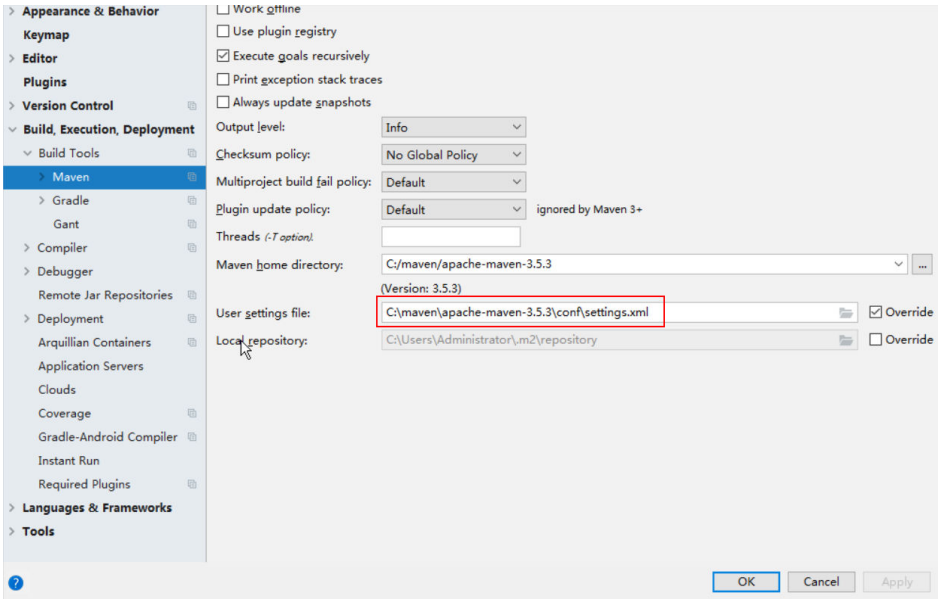


12. Click **Apply** and **OK**.

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 18-5 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

18.2.4 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, **hive_python_client 'show tables'**.

NOTE

This function applies to only simple SQL statements and depends on the ZooKeeper client.

----End

18.2.5 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. Versions earlier than MRS 3.2.0, 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

   The following information indicates that egg is installed successfully.

   ```
   Finished processing dependencies for future==0.18.2
   ```

   NOTICE

   If egg files of both **aarch64** and **x86_64** versions exist in the **dependency_python3.x** folder, you need to select one of the versions based on the operating system and run the **uname -p** command to check the current operating system architecture.

6. In MRS 3.2.0 and later versions, 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

18.3 Developing an Application

18.3.1 Typical Scenario Description

Scenarios

A user develops a Hive data analysis application for managing employee information described in **Table 18-8** and **Table 18-9**.

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 18-8 describes employee information.

Table 18-8 Employee information

ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hiring 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

ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hiring Date
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 18-9 describes employee contact information.

   Table 18-9 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

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

18.3.2 Example Codes

18.3.2.1 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

- 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 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 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;

-- Use CREATE Like to create a table.
CREATE TABLE employees_like LIKE employees_info;

-- Use DESCRIBE to query the structures of employees_info, employees_like, and employees_as_select tables.
DESCRIBE employees_info;
DESCRIBE employees_like;
```


Extensions

- Create a partition.

  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.

  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 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);
  ```

- Configure Hive data encryption when creating a table.

  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.encryption.arc4.ARC4Codec;
  create table seq_Codec (key string, value string) stored as RCFile;
  ```

18.3.2.2 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.

NOTE

To perform the following operations on a cluster enabled with the security service, you must have the update permission for databases, 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

Loading data is to copy data to a specified table in the Hadoop distributed file system (HDFS).

Extensions

None

18.3.2.3 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.encryption.arc4.ARC4Codec;
  ```

- For details about UDFs, see **UDF**.

18.3.2.4 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

18.3.2.5 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 the Hive example in the **Development Specifications**. 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.

18.3.2.6 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.

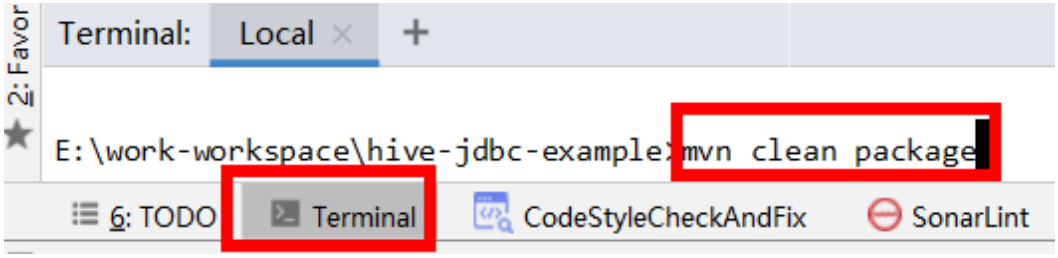
```
digestZK = new org.apache.zookeeper.ZooKeeper("xxx.xxx.xxx.xxx:2181", 60000, null);
```

18.4 Commissioning Applications

18.4.1 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.


```
Terminal:  Local  +
[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 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!
```

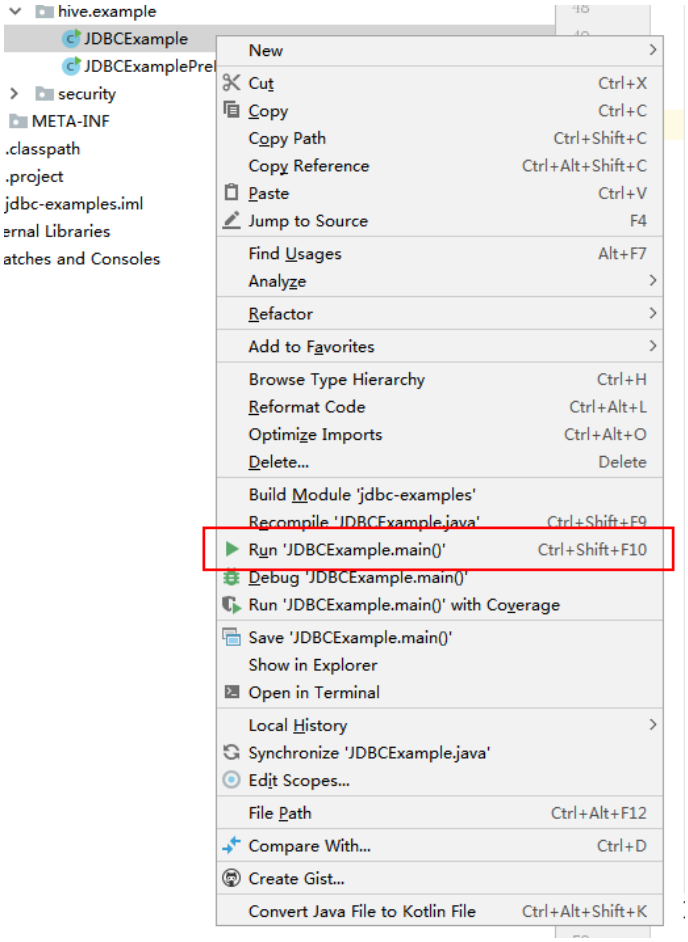
The following information is displayed if the sample project is successful in Linux:

```
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.

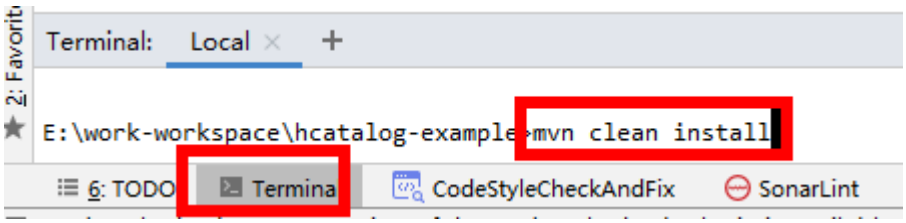
```
Create table success!
_c0
0
Delete table success!
```

----End

18.4.2 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  +
[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-exa
[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-3.1.0.jar,$HCAT_HOME/lib/hive-metastore-3.1.0.jar,$HCAT_HOME/lib/hive-standalone-metastore-3.1.0.jar,$HIVE_HOME/lib/hive-exec-3.1.0.jar,$HCAT_HOME/lib/libfb303-0.9.3.jar,$HCAT_HOME/lib/slf4j-api-1.7.30.jar,$HCAT_HOME/lib/jdo-api-3.0.1.jar,$HCAT_HOME/lib/antlr-runtime-3.5.2.jar,$HCAT_HOME/lib/datanucleus-api-jdo-4.2.4.jar,$HCAT_HOME/lib/datanucleus-core-4.1.17.jar,$HCAT_HOME/lib/datanucleus-rdbms-fi-4.1.19.jar,$HCAT_HOME/lib/log4j-api-2.10.0.jar,$HCAT_HOME/lib/log4j-core-2.10.0.jar
export HADOOP_CLASSPATH=$HCAT_HOME/lib/hive-hcatalog-core-3.1.0.jar:$HCAT_HOME/lib/hive-metastore-3.1.0.jar:$HCAT_HOME/lib/hive-standalone-metastore-3.1.0.jar:$HIVE_HOME/lib/hive-exec-3.1.0.jar:$HCAT_HOME/lib/libfb303-0.9.3.jar:$HADOOP_HOME/etc/hadoop:$HCAT_HOME/conf:$HCAT_HOME/lib/slf4j-api-1.7.30.jar:$HCAT_HOME/lib/jdo-api-3.0.1.jar:$HCAT_HOME/lib/antlr-runtime-3.5.2.jar:$HCAT_HOME/lib/datanucleus-api-jdo-4.2.4.jar:$HCAT_HOME/lib/datanucleus-core-4.1.17.jar:$HCAT_HOME/lib/datanucleus-rdbms-fi-4.1.19.jar:$HCAT_HOME/lib/log4j-api-2.10.0.jar:$HCAT_HOME/lib/log4j-core-2.10.0.jar
```

NOTE

- **Change the version numbers of the JAR files specified in LIB_JARS and HADOOP_CLASSPATH based on the actual environment.** For example, if the version number of the **hive-hcatalog-core** JAR file in **$HCAT_HOME/lib** is **3.1.0-hw-ei-302001**, change **$HCAT_HOME/lib/hive-hcatalog-core-3.1.0.jar** in **LIB_JARS** to **$HCAT_HOME/lib/hive-hcatalog-core-3.1.0-hw-ei-302001.jar**.
- 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);**

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

18.4.3 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 the 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

18.4.4 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 the 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:

```
[['default', '']]
 [{'comment': 'from deserializer', 'columnName': 'tab_name', 'type': 'STRING_TYPE'}]
 ['xx']
```

----End

18.5 More Information

18.5.1 Interface Reference

18.5.1.1 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.

18.5.1.2 Hive SQL

Hive SQL supports all features of Hive-3.1.0. For details, see **https://cwiki.apache.org/confluence/display/hive/languagemanual**.

Table 18-10 describes the extended Hive statements provided by .

Table 18-10 Extended Hive statements

Extended Syntax	Syntax Description	Syntax Example	Example Description
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 ("groupId"=" group1 ","locatorId"="locator1")] ...;	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**.	CREATE TABLE tab1 (id INT, name STRING) row format delimited fields terminated by '\t' stored as RCFILE TBLPROPERTIES(" groupId"=" group1 ","locatorId"="locator1");	The statement is used to create table **tab1** and specify locator1 on which the table data of **tab1** locates.

Extended Syntax	Syntax Description	Syntax Example	Example Description
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')]...;	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**.	create table encode_test(id INT, name STRING, phone STRING, address STRING) ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe' WITH SERDEPROPERTIES ('column.encode.indices'='2,3', 'column.encode.classname'='org.apache.hadoop.hive.serde2.SMS4Rewriter') STORED AS TEXTFILE;	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.
REMOVE TABLE *hbase_tablename* [WHERE where_condition];	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**.	remove table hbase_table1 where id = 1;	The statement is used to delete data that meets the criterion of "id = 1" from the table.

Extended Syntax	Syntax Description	Syntax Example	Example Description
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.hadoop.hive.contrib.fileformat.SpecifiedDelimiterInput-Format'** outputformat 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat';	The statement is used to create a hive table and specify that the table supports customized row delimiters. For details, see **Customizing Row Separators**.	create table blu(time string, num string, msg string) row format delimited fields terminated by ',' **stored as inputformat 'org.apache.hadoop.hive.contrib.fileformat.SpecifiedDelimiterInput-Format'** outputformat 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat';	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.

18.5.1.3 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 **kinit** 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
:version	WebHCat version number. Currently, the version number must be v1.

- Returned result

Parameter	Description
responseTypes	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
status	If the WebHCat connection is normal, **OK** is returned.
version	Character string, including the version number, for example, v1.

   - 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	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 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 the Hive rule in the **Development Specifications**.

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.

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/$attempt_id/syslog
callback	Callback address after MapReduce job execution. Use **$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 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"
```

18.5.2 Hive of the Cluster in Security Mode 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**.

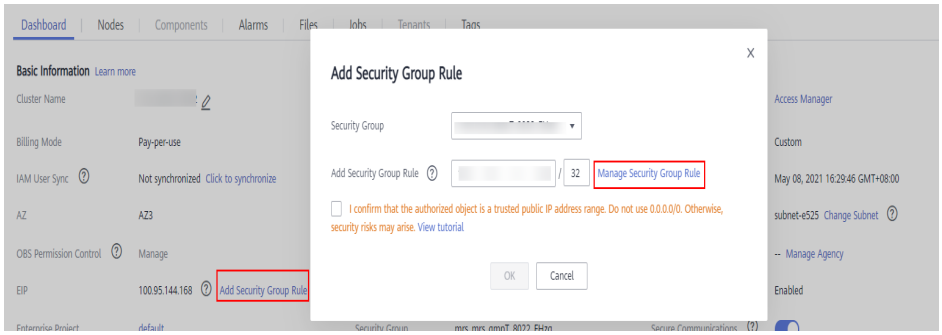
2. Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.


```
Mapping between public IP addresses and private IP addresses
100.95.[redacted] 172.16.0.120
100.95.[redacted] 172.16.0.42
100.93.[redacted] 172.16.0.62
100.95.[redacted] 172.16.0.200
100.93.[redacted] 172.16.0.139
100.93.[redacted] 172.16.0.214

hosts file in the cluster
172.16.0.120 node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
172.16.0.42 node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
172.16.0.62 node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
172.16.0.200 node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
172.16.0.139 node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
172.16.0.214 node-master2pVNu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVNu.ead64699-185a-4290-bbef-1a07e2f0459b.com.

hosts file that should be added to Windows
100.95.[redacted]3 node-group-1xzio0002.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0002.ead64699-185a-4290-bbef-1a07e2f0459b.com.
100.95.[redacted]6 node-master3vint.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master3VInT.ead64699-185a-4290-bbef-1a07e2f0459b.com.
100.93.[redacted] node-group-1xzio0003.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0003.ead64699-185a-4290-bbef-1a07e2f0459b.com.
100.95.[redacted]5 node-master1ceip.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master1CeIP.ead64699-185a-4290-bbef-1a07e2f0459b.com.
100.93.[redacted] node-group-1xzio0001.ead64699-185a-4290-bbef-1a07e2f0459b.com node-group-1XZIO0001.ead64699-185a-4290-bbef-1a07e2f0459b.com.
100.93.[redacted] node-master2pvnu.ead64699-185a-4290-bbef-1a07e2f0459b.com node-master2pVNu.ead64699-185a-4290-bbef-1a07e2f0459b.com.
```

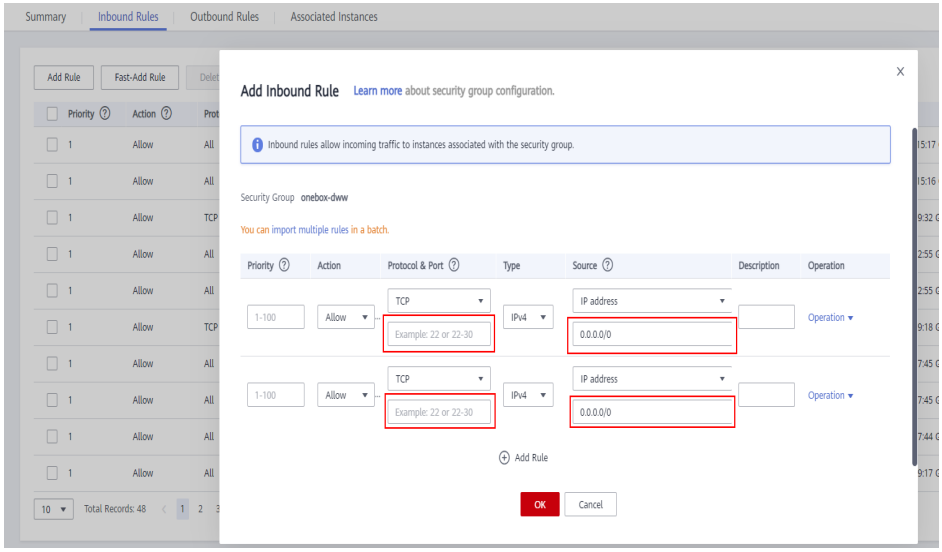
Step 2 Change the IP addresses in the **krb5.conf** file to the corresponding host names.

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-master1lnks.c0c17d76-481f-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 **PRNCIPAL_NAME** in the sample code to the username for security authentication.

----End

18.5.3 FAQ

18.5.3.1 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.

18.5.3.2 Problem performing GSS wrap Message Is Displayed Due to IBM JDK Exceptions

Question

IBM JDK is abnormal, and the **Problem performing GSS wrap** massage 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.

18.5.3.3 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 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)
```


19 Hive Development Guide (Normal Mode)

19.1 Overview

19.1.1 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, ORCFILE, 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**.

19.1.2 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.

19.1.3 Development Process

Figure 19-1 and **Table 19-1** illustrates each phase of the development process.

Figure 19-1 Hive application development process

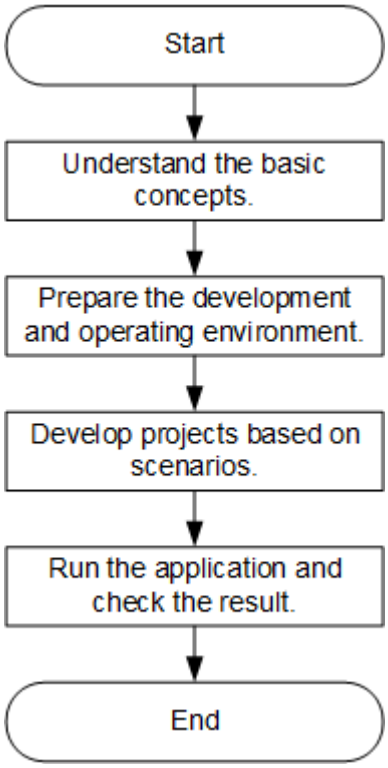


Table 19-1 Description of the Hive application development process

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**
Develope 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**

19.2 Preparing the Environment

19.2.1 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 19-2 JDBC development environment

Item	Description
OS	• 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	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. • For x86 nodes that run clients, the following JDKs can be used: – 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: – OpenJDK: 1.8.0_272 **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 com.ibm.jsse2.overrideDefaultTLS 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.**
IntelliJ IDEA installation and configuration	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. **NOTE** • 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	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.
7-zip	Used to decompress **.zip** and **.rar** packages. 7-Zip 16.04 is supported.

Table 19-3 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 shoul 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 19-4 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 shoul 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. Download and decompress the client software package.
     - **Log in to the FusionInsight Manager portal** and choose **Cluster** > **Dashboard** > **More** > **Download Client**. Set **Select Client Type** to **Configuration Files Only** 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 the Developer Account** is also stored in this directory. **Table 19-5** describes the main configuration files.

Table 19-5 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 19-6 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.

19.2.2 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.

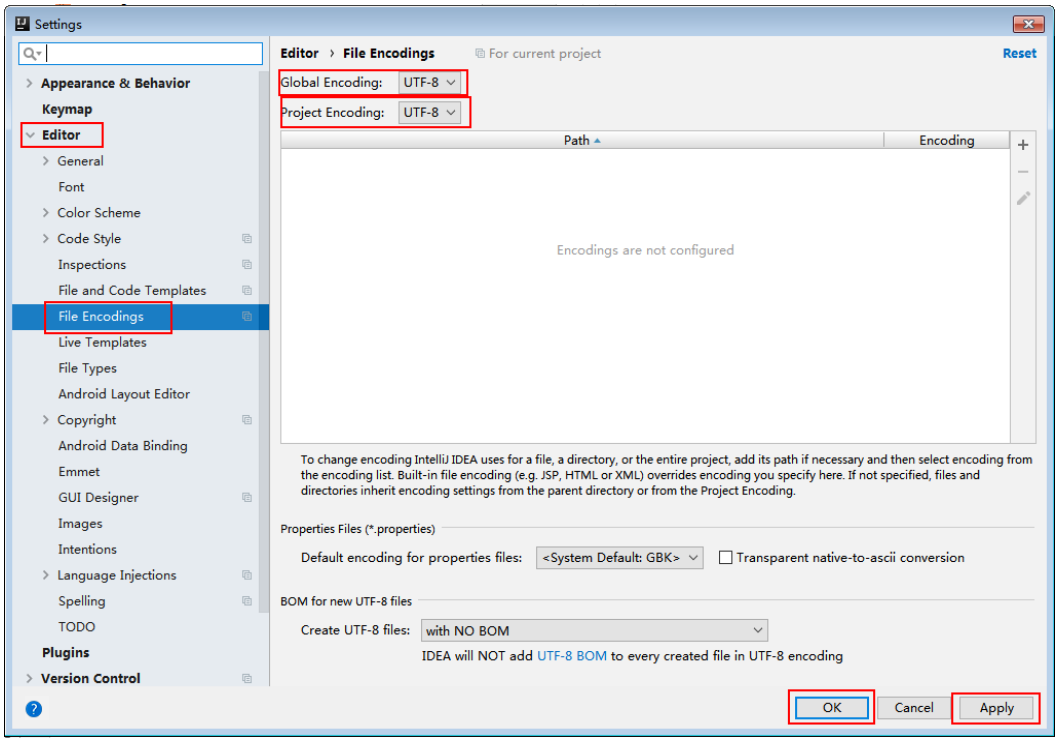
1. On the IntelliJ IDEA menu bar, choose **File** > **Open...**. The **Open File or Project** window is displayed.
2. In the displayed window, select the folder **hive-jdbc-example**, and click **OK**. On Windows, the path cannot contain any space.

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 19-2**.

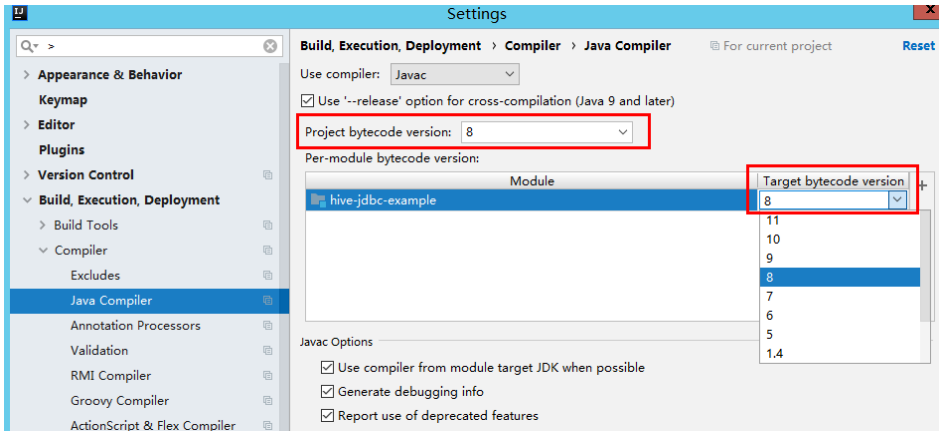
Figure 19-2 Setting the IntelliJ IDEA coding format



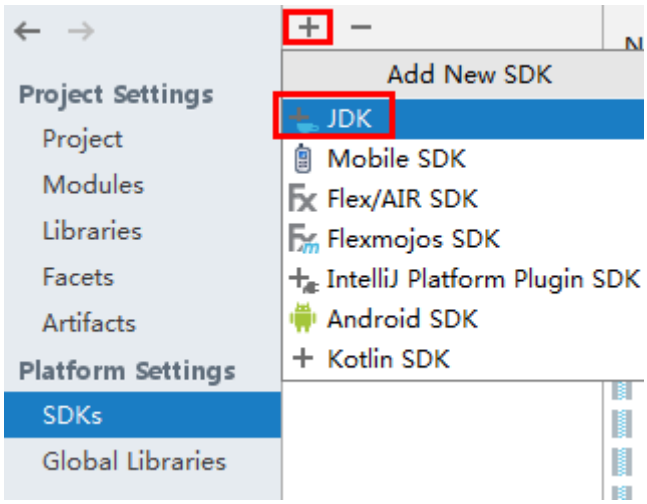
Step 5 Set the JDK of the project.

1. On the IntelliJ IDEA menu bar, choose **File** > **Settings...**. The **Settings** window is displayed.

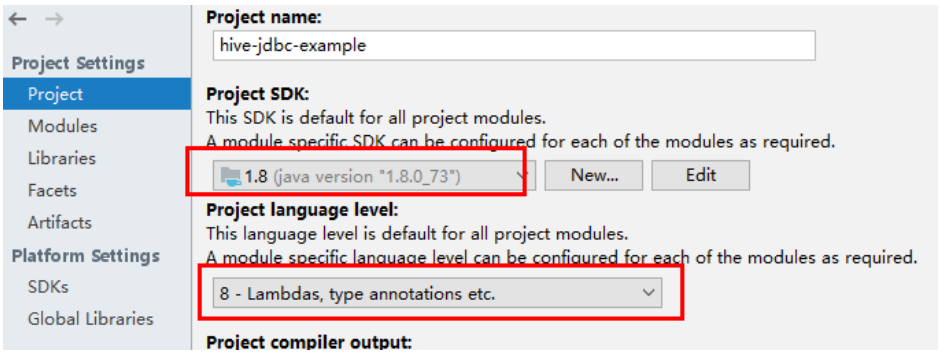
2. Choose **Build, Execution, Deployment** > **Compiler** > **Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **hive-jdbc-example**.



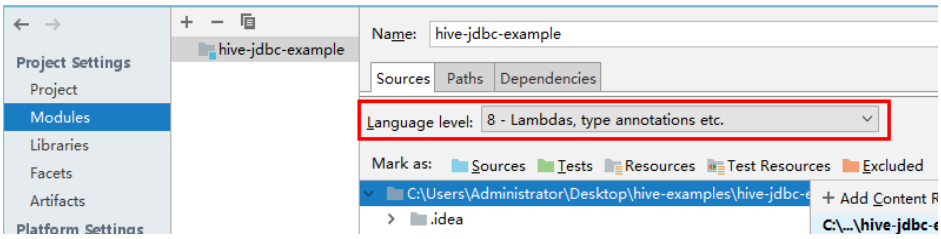
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File** > **Project Structure...**. The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



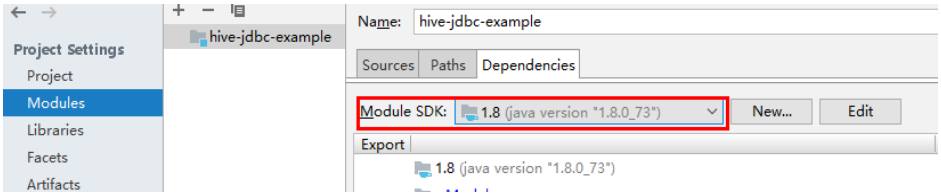
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** 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.

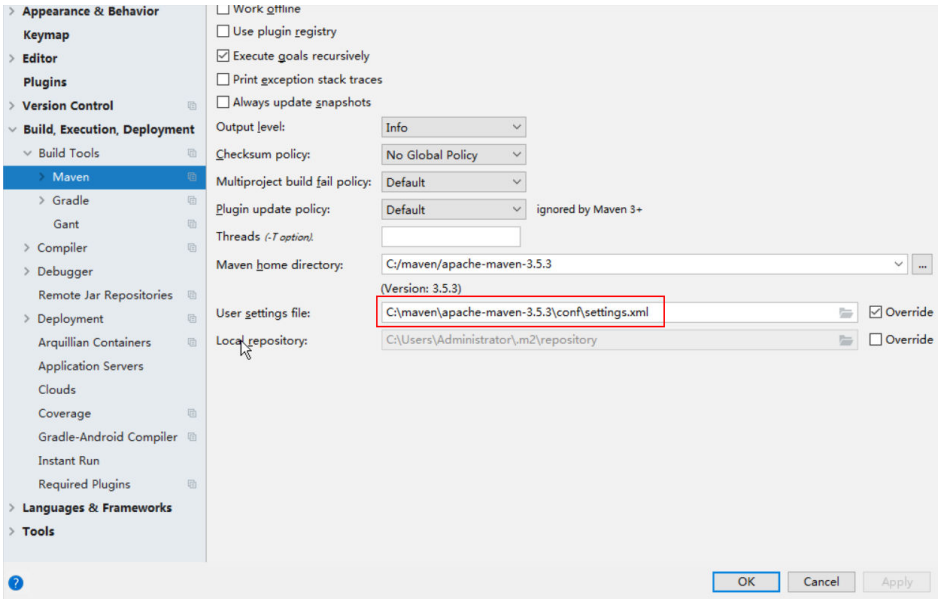


11. Click **Apply** and **OK**.

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 19-3 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

19.2.3 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.

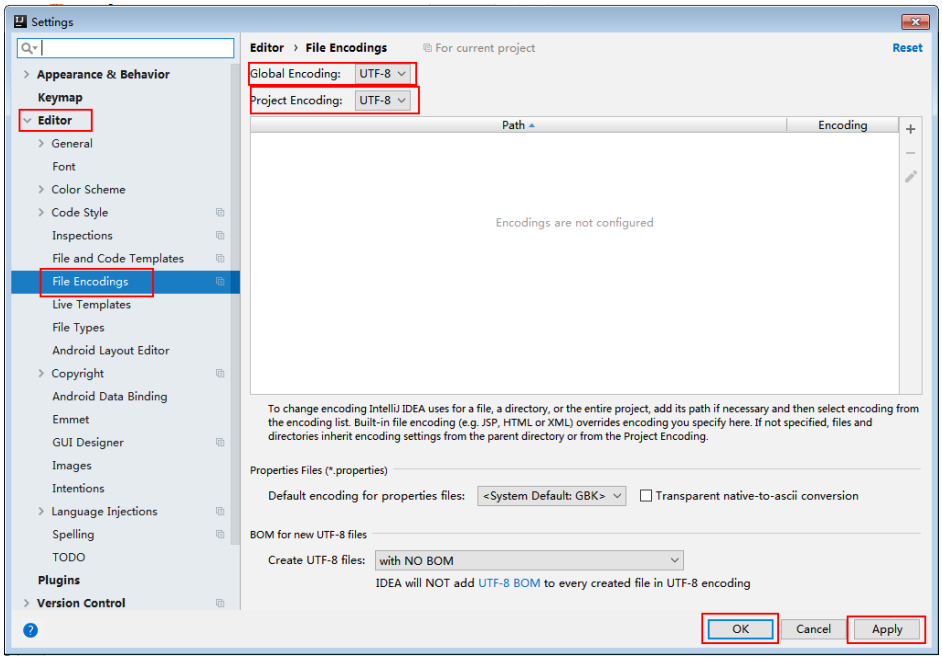
1. On the IntelliJ IDEA menu bar, choose **File** > **Open...**. The **Open File or Project** window is displayed.
2. In the displayed window, select the folder **hcatalog-example**, and click **OK**. On Windows, the path cannot contain any space.

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.

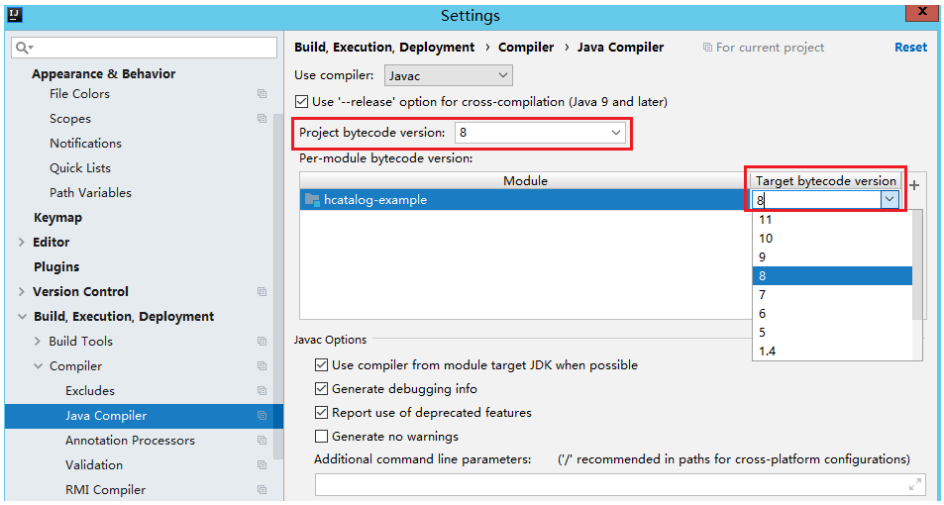
2. Choose **Editor** > **File Encodings** from the navigation tree. In the **Project Encoding** area and **Global Encoding** area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in **Figure 19-4**.

Figure 19-4 Setting the IntelliJ IDEA coding format

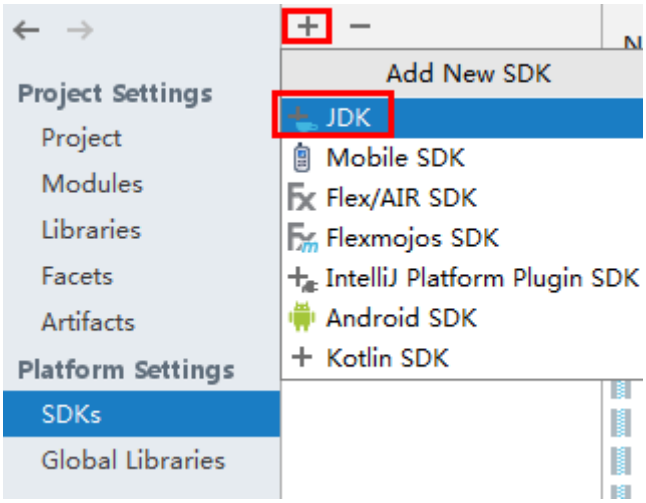


Step 4 Set the JDK of the project.

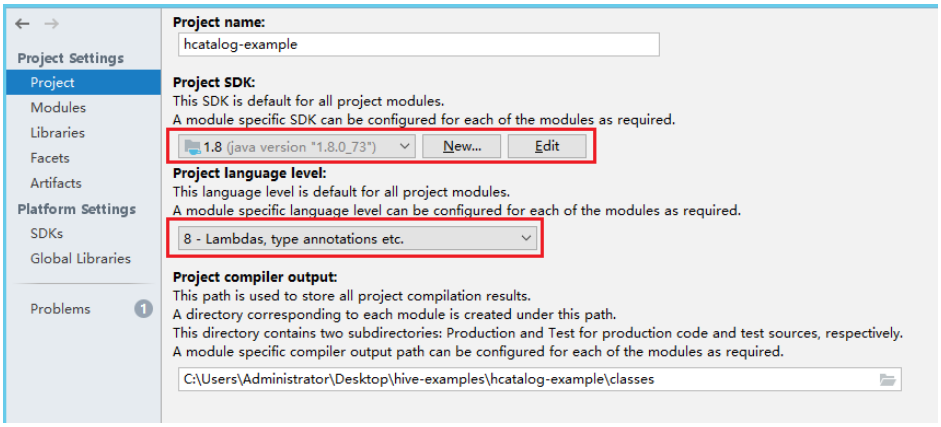
1. On the IntelliJ IDEA menu bar, choose **File** > **Settings...**. The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment** > **Compiler** > **Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **hive-jdbc-example**.



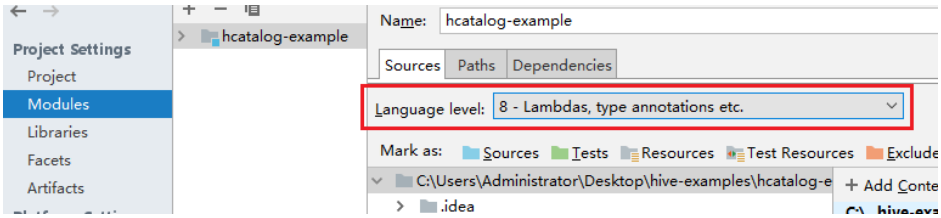
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File** > **Project Structure...**. The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



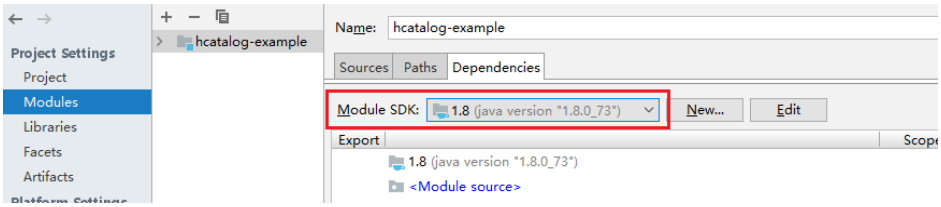
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** 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.

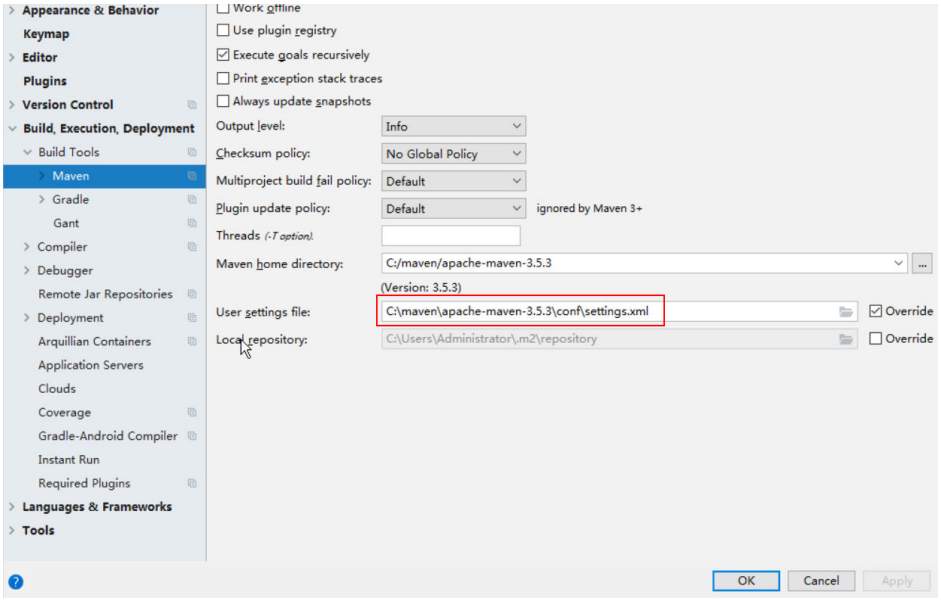


12. Click **Apply** and **OK**.

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 19-5 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

19.2.4 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, **hive_python_client 'show tables'**. This function applies to only simple SQL statements and depends on the ZooKeeper client.

----End

19.2.5 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. Versions earlier than MRS 3.2.0, 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

   The following information indicates that egg is installed successfully.

   ```
   Finished processing dependencies for future==0.18.2
   ```

   NOTICE

   If egg files of both **aarch64** and **x86_64** versions exist in the **dependency_python3.x** folder, you need to select one of the versions based on the operating system and run the **uname -p** command to check the current operating system architecture.

6. In MRS 3.2.0 and later versions, 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

19.3 Developing an Application

19.3.1 Typical Scenario Description

Scenarios

A user develops a Hive data analysis application for managing employee information described in **Table 19-7** and **Table 19-8**.

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 19-7 describes employee information.

Table 19-7 Employee information

ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hiring 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 19-8 describes employee contact information.

Table 19-8 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

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

19.3.2 Example Codes

19.3.2.1 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;

-- Use CREATE Like to create a table.
CREATE TABLE employees_like LIKE employees_info;
```



```
-- Use DESCRIBE to query the structures of employees_info, employees_like, and employees_as_select tables.
DESCRIBE employees_info;
DESCRIBE employees_like;
```

Extensions

- Create a partition.

  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.

  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 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);
  ```

- Configure Hive data encryption when creating a table.

  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.encryption.arc4.ARC4Codec;
  create table seq_Codec (key string, value string) stored as RCFile;
  ```

19.3.2.2 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).

Extensions

None

19.3.2.3 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.encryption.arc4.ARC4Codec;
  ```

- For details about UDFs, see **UDF**.

19.3.2.4 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

19.3.2.5 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 the Hive example in the **Development Specifications**. 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.

19.3.2.6 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);
```

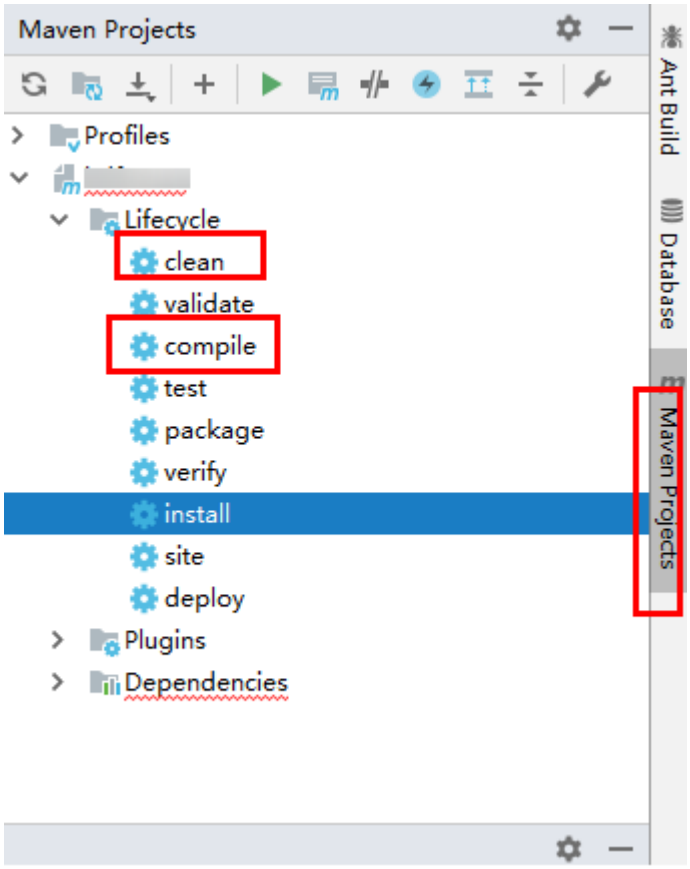
19.4 Commissioning Applications

19.4.1 Running JDBC and Viewing Results

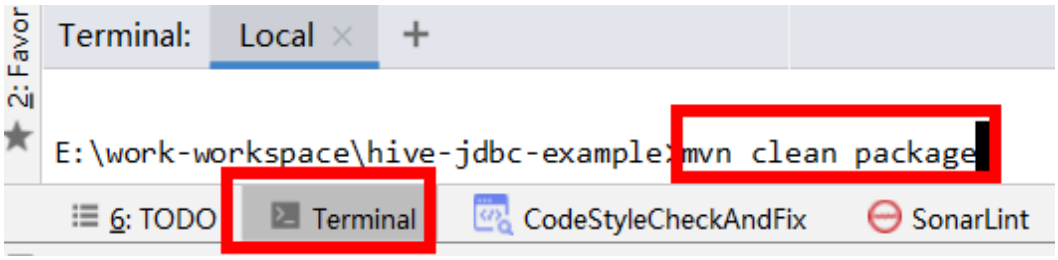
Running JDBC in CLI Mode

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 19-6 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  +
[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:

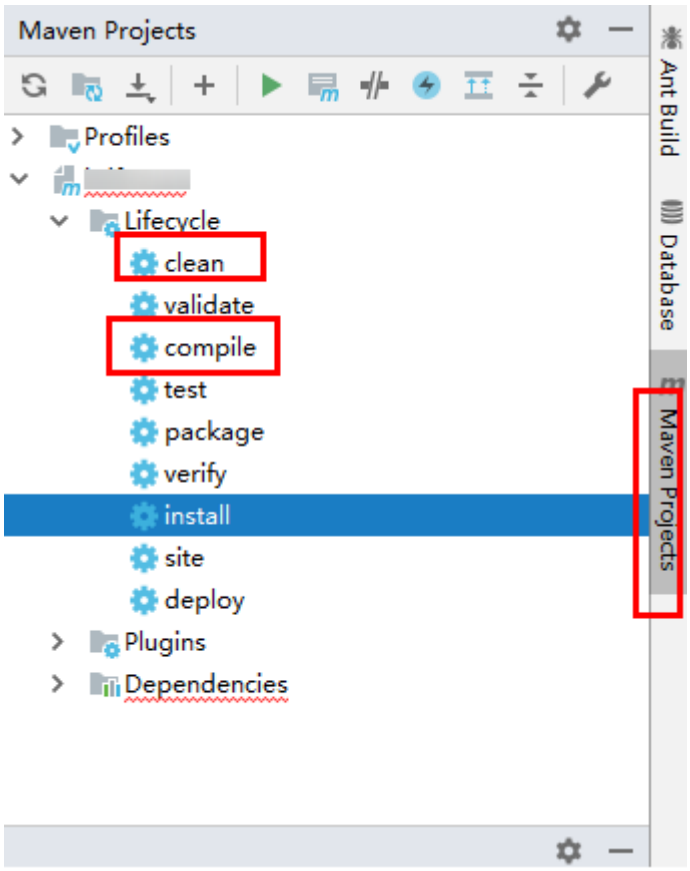
```
Create table success!
_c0
0
Delete table success!
```

----End

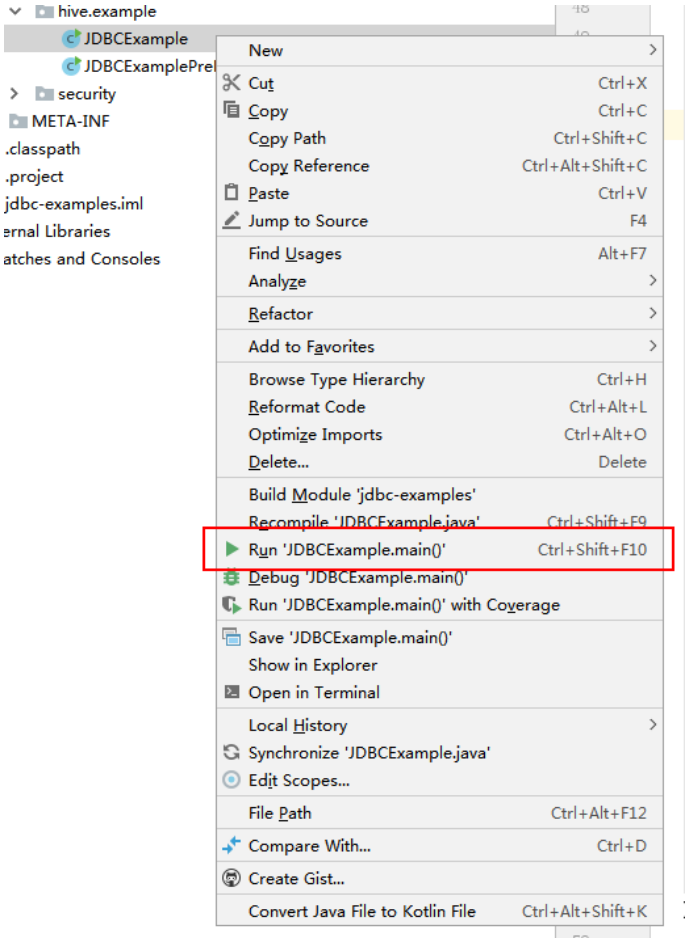
Running JDBC in IntelliJ IDEA Mode

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 19-7 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!
```

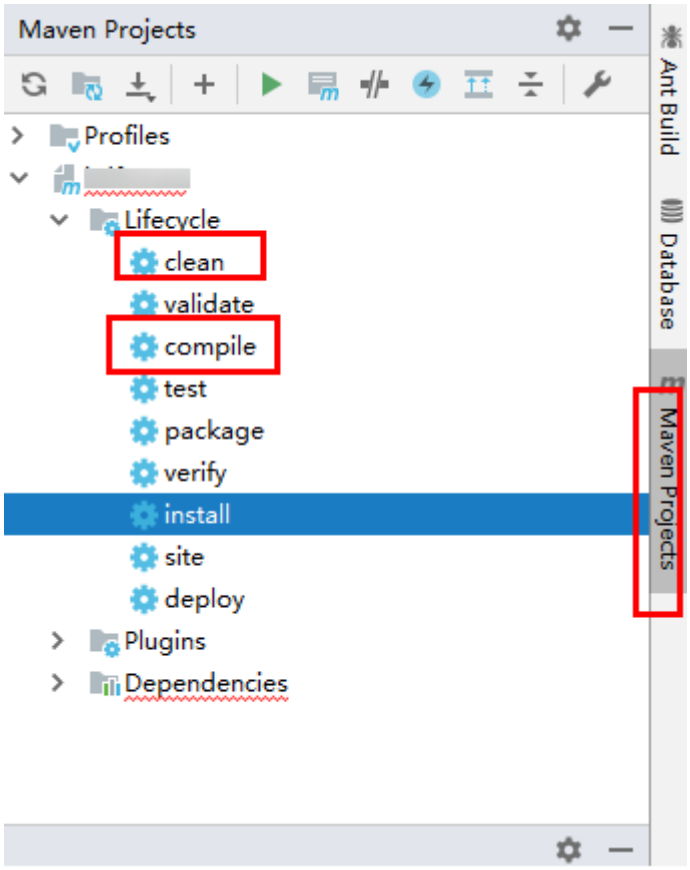
----End

19.4.2 Running HCatalog and Viewing Results

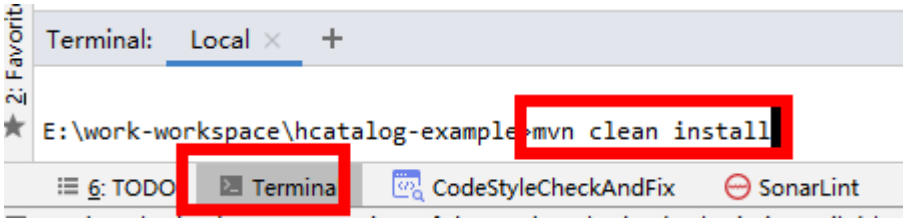
Running HCatalog Example Projects

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 19-8 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  +
[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-exa
[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-3.1.0.jar,$HCAT_HOME/lib/hive-metastore-3.1.0.jar,$HCAT_HOME/lib/hive-standalone-metastore-3.1.0.jar,$HIVE_HOME/lib/hive-exec-3.1.0.jar,$HCAT_HOME/lib/libfb303-0.9.3.jar,$HCAT_HOME/lib/slf4j-api-1.7.30.jar,$HCAT_HOME/lib/jdo-api-3.0.1.jar,$HCAT_HOME/lib/antlr-runtime-3.5.2.jar,$HCAT_HOME/lib/datanucleus-api-jdo-4.2.4.jar,$HCAT_HOME/lib/datanucleus-core-4.1.17.jar,$HCAT_HOME/lib/datanucleus-rdbms-fi-4.1.19.jar,$HCAT_HOME/lib/log4j-api-2.10.0.jar,$HCAT_HOME/lib/log4j-core-2.10.0.jar
export HADOOP_CLASSPATH=$HCAT_HOME/lib/hive-hcatalog-core-3.1.0.jar:$HCAT_HOME/lib/hive-metastore-3.1.0.jar:$HCAT_HOME/lib/hive-standalone-metastore-3.1.0.jar:$HIVE_HOME/lib/hive-exec-3.1.0.jar:$HCAT_HOME/lib/libfb303-0.9.3.jar:$HADOOP_HOME/etc/hadoop:$HCAT_HOME/conf:$HCAT_HOME/lib/slf4j-api-1.7.30.jar:$HCAT_HOME/lib/jdo-api-3.0.1.jar:$HCAT_HOME/lib/antlr-runtime-3.5.2.jar:$HCAT_HOME/lib/datanucleus-api-jdo-4.2.4.jar:$HCAT_HOME/lib/datanucleus-core-4.1.17.jar:$HCAT_HOME/lib/datanucleus-rdbms-fi-4.1.19.jar:$HCAT_HOME/lib/log4j-api-2.10.0.jar:$HCAT_HOME/lib/log4j-core-2.10.0.jar
```

NOTE

- **Change the version numbers of the JAR files specified in LIB_JARS and HADOOP_CLASSPATH based on the actual environment.** For example, if the version number of the **hive-hcatalog-core** JAR file in **$HCAT_HOME/lib** is **3.1.0-hw-ei-302001**, change **$HCAT_HOME/lib/hive-hcatalog-core-3.1.0.jar** in **LIB_JARS** to **$HCAT_HOME/lib/hive-hcatalog-core-3.1.0-hw-ei-302001.jar**.
- 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);**

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

19.4.3 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: 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

19.4.4 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:

```
[['default', '']]
 [{'comment': 'from deserializer', 'columnName': 'tab_name', 'type': 'STRING_TYPE'}]
 ['xx']
```

----End

19.5 More Information

19.5.1 Interface Reference

19.5.1.1 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.

19.5.1.2 Hive SQL

Hive SQL supports all features of Hive-3.1.0. For details, see **https://cwiki.apache.org/confluence/display/hive/languagemanual**.

Table 19-9 describes the extended Hive statements provided by FusionInsight.

Table 19-9 Extended Hive statements

Extended Syntax	Syntax Description	Syntax Example	Example Description
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 ("groupId"=" group1 ","locatorId"="locator1")] ...;	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**.	CREATE TABLE tab1 (id INT, name STRING) row format delimited fields terminated by '\t' stored as RCFILE TBLPROPERTIES(" groupId"=" group1 ","locatorId"="locator1");	The statement is used to create table **tab1** and specify locator1 on which the table data of **tab1** locates.

Extended Syntax	Syntax Description	Syntax Example	Example Description
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')]...;	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**.	create table encode_test(id INT, name STRING, phone STRING, address STRING) ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe' WITH SERDEPROPERTIES ('column.encode.indices'='2,3', 'column.encode.classname'='org.apache.hadoop.hive.serde2.SMS4Rewriter') STORED AS TEXTFILE;	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.
REMOVE TABLE *hbase_tablename* [WHERE where_condition];	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**.	remove table hbase_table1 where id = 1;	The statement is used to delete data that meets the criterion of "id = 1" from the table.

Extended Syntax	Syntax Description	Syntax Example	Example Description
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.hadoop.hive.contrib.fileformat.SpecifiedDelimiterInput-Format'** outputformat 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat';	The statement is used to create a hive table and specify that the table supports customized row delimiters. For details, see .	create table blu(time string, num string, msg string) row format delimited fields terminated by ',' **stored as inputformat 'org.apache.hadoop.hive.contrib.fileformat.SpecifiedDelimiterInput-Format'** outputformat 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat';	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.

19.5.1.3 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/db2?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	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 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 define=NAME=VALUE.
cmdenv	Set an environment variable using the syntax cmdenv=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.
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*.

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 the Hive rule in the **Development Specifications**.

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"
```

19.5.2 Hive of the Cluster in Normal Mode 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.

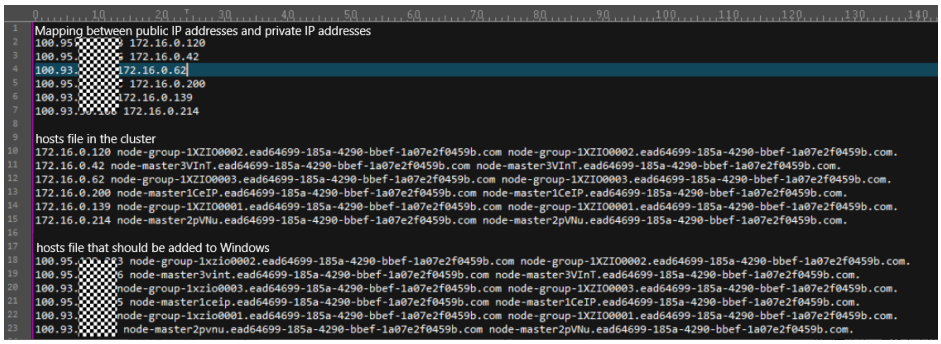
Procedure

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.

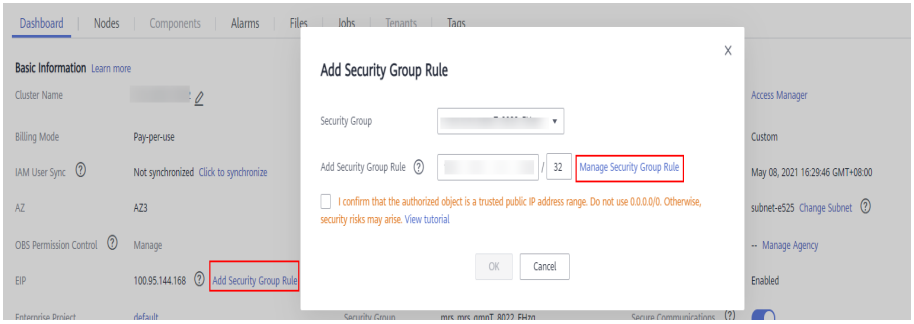
   For details, see **Virtual Private Cloud** > **User Guide** > **EIP** > **Assigning an EIP and Binding It to an ECS**.

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.

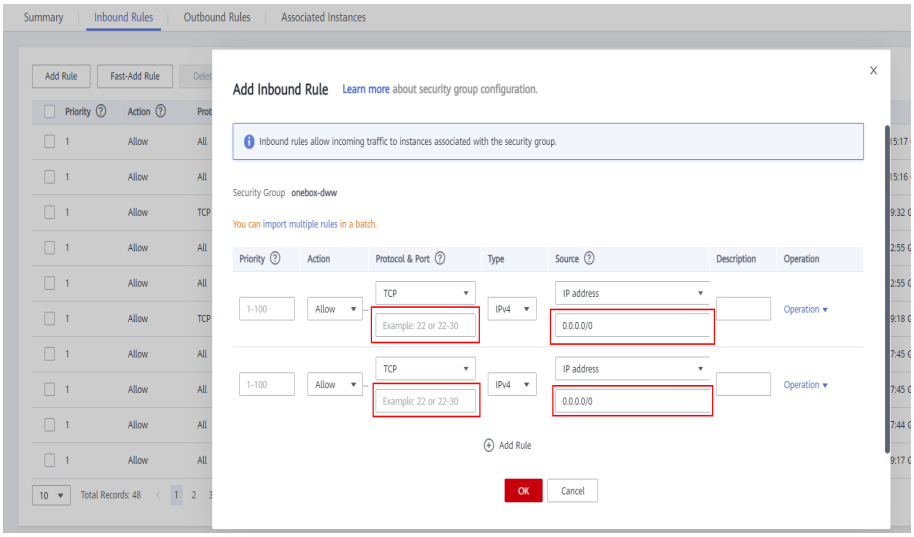


Step 2 Configure security group rules for the cluster.

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.

```
// Build JDBC URL
//   StringBuilder strBuilder = new StringBuilder("jdbc:hive2://").append(zkQuorum).append("/");
StringBuilder strBuilder = new StringBuilder("jdbc:hive2://node-master1lnks.c0c17d76-481f-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");
}
```

----End

19.5.3 FAQ

19.5.3.1 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.

20 IoTDB Development Guide (Security Mode)

20.1 Overview

20.1.1 Application Development Overview

Introduction to IoTDB

IoTDB is a data management engine that integrates collection, storage, and analysis of time series data. It features lightweight, high performance, and ease of use. It perfectly interconnects with the Hadoop and Spark ecosystems and meets the requirements of high-speed write and complex analysis and query on massive time series data in industrial IoT applications.

NOTE

This document applies only to MRS 3.2.0 or later.

20.1.2 Basic Concepts

The following uses an electric power scenario as an example to describe how to create a correct data model in IoTDB.

Figure 20-1 Hierarchical structure of attributes in an electric power scenario

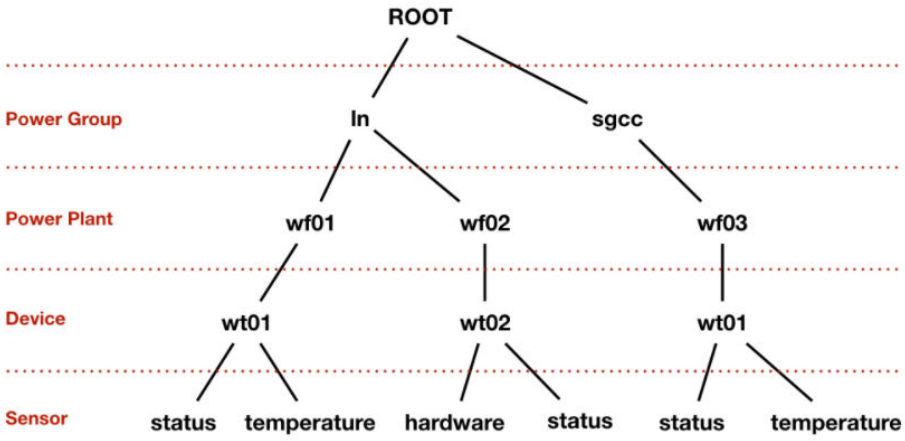


Figure 20-1 shows the power group layer, power plant layer, device layer, and sensor layer. **ROOT** is a root node, and each node at the sensor layer is a leaf node. According to the IoTDB syntax, the path from **ROOT** to a leaf node is separated by a dot (.). The complete path is used to name a time series in the IoTDB. For example, the time series name corresponding to the path on the left in Figure 20-1 is **ROOT.ln.wf01.wt01.status**.

Basic Concepts

- **Device**

  A device is a machine equipped with sensors in actual scenarios. In IoTDB, all sensors must have their corresponding devices.

- **Sensor**

  A sensor is a detection machine in actual scenarios. It can sense the information to be measured, and can transform the sensed information into an electrical signal or other desired form of information output and send it to IoTDB. In IoTDB, all data and paths stored are organized in units of sensors.

- **Storage group**

  You can set any prefix path as a storage group. If there are four time series, for example, **root.vehicle.d1.s1**, **root.vehicle.d1.s2**, **root.vehicle.d2.s1**, and **root.vehicle.d2.s2**, two devices **d1** and **d2** in the path **root.vehicle** may belong to the same owner or manufacturer, so **d1** and **d2** are closely related. In this case, the prefix path **root.vehicle** can be designated as a storage group, which will enable IoTDB to store the data of all devices under it in the same folder. Newly added devices in **root.vehicle** will also belong to this storage group.

  NOTE

  - A proper number of storage groups can improve performance. There is neither the slowdown of the system due to frequent I/O switching (which will also take up excessive memory and result in frequent memory-file switching) caused by too many storage files or folders, nor the block of write commands caused by too few storage files or folders (which reduces concurrency).
  - You need to balance the storage group settings of storage files according to their own data size and usage scenarios to achieve better system performance.

- **Time series**

  Time series is a core concept in IoTDB. The time series can be considered as the complete path of the sensor that generates the time series data. In IoTDB, all time series must start with root and end with the sensor.

20.1.3 Development Process

This section describes how to use Java APIs to develop IoTDB applications.

Figure 20-2 and **Table 20-1** describe the phases in the development process.

Figure 20-2 IoTDB application development process

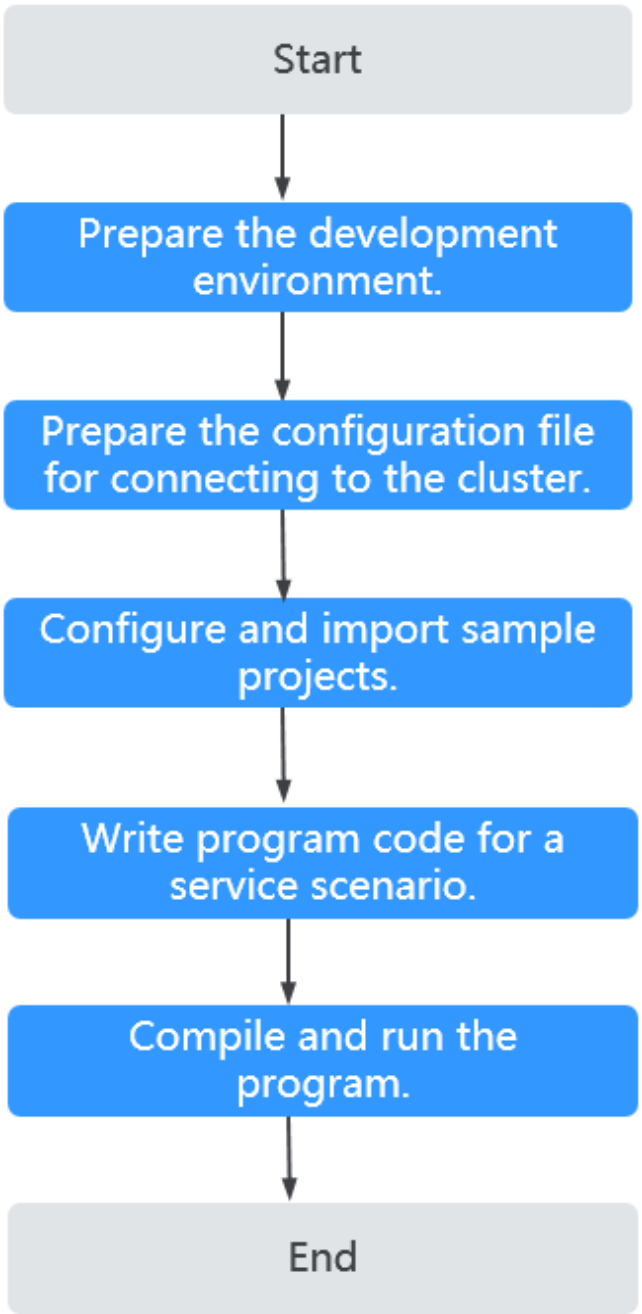


Table 20-1 Description of the IoTDB application development process

Phase	Description	Reference
Prepare the development environment.	Before developing an application, you need to prepare the development environment. You are advised to use the Java language and IntelliJ IDEA tool for development. In addition, you need to complete the initial configuration of the JDK and Maven.	**Preparing the Environment**
Prepare the Configuration File for Connecting to the Cluster.	During application development or running, you need to connect to the MRS cluster through cluster configuration files. Configuration files include cluster component information files and user files used for security authentication. You can obtain related content from the created MRS cluster. Nodes used for program commissioning or running must be able to communicate with nodes in the MRS cluster and the hosts domain name must be configured.	**Preparing the Configuration Files for Connecting to the Cluster**
Configure and import the sample project.	IoTDB provides multiple sample programs in different scenarios. You can obtain sample projects and import them to the local development environment for program learning.	**Configuring and Importing a Sample Projects**
Develop programs based on business scenarios.	Sample projects of the Java language are provided, including JDBC and Session connection modes. A sample project covers the entire process from creating a storage group, creating a time sequence, inserting data, to deleting a storage group.	**Application Development**
Compile and run the application.	Compile the developed application and submit it for running.	**Application Commissioning**

20.1.4 IoTDB Sample Project

To obtain an MRS sample project, visit **https://github.com/huaweicloud/huaweicloud-mrs-example** and switch to the branch that matches the MRS cluster version. Download the package to the local PC and decompress it to obtain the sample project of each component.

MRS provides the following IoTDB sample projects.

Table 20-2 IoTDB sample projects

Sample Project Location	Description
iotdb-examples/iotdb-flink-example	Program for using Flink to access IoTDB data, including FlinkIoTDBSink and FlinkIoTDBSource data. FlinkIoTDBSink can use Flink jobs to write time series data to IoTDB. FlinkIoTDBSource reads time series data from IoTDB through Flink jobs and prints the data. For details, see **IoTDB Flink**.
iotdb-examples/iotdb-jdbc-example	Java sample program for IoTDB JDBC to process data This example demonstrates how to use JDBC APIs to connect to IoTDB and run IoTDB SQL statements. For details, see **IoTDB JDBC**.
iotdb-examples/iotdb-kafka-example	Sample program for accessing IoTDB data through Kafka This program demonstrates how to send time series data to Kafka and then use multiple threads to write the data to IoTDB. For details, see **IoTDB Kafka**.
iotdb-examples/iotdb-session-example	Java sample program for IoTDB Session to process data This example demonstrates how to use Session to connect to IoTDB and run IoTDB SQL statements. For details, see **IoTDB Session**.
iotdb-examples/iotdb-udf-exmaple	This sample program describes how to implement a simple IoTDB user-defined function (UDF). For details, see **IoTDB UDF Program**.

20.2 Environment Preparations

20.2.1 Preparing the Environment

Table 20-3 describes the environment required for application development.

Table 20-3 Development environment

Item	Description
OS	• Development environment: Windows 7 or later versions • Operating environment: Windows or Linux If the program needs to be commissioned locally, the operating environment must be able to communicate with network on the cluster service plane.

Item	Description
JDK	Basic configurations of the development and operating environments. The version requirements are as follows: The server and client support only built-in OpenJDK 1.8.0_272. Other JDKs cannot be used. Customers' applications that need to reference the JAR files of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK. x86 client: Oracle JDK 1.8; IBM JDK: 1.8.5.11. **NOTE** For security purposes, the server supports only TLS V1.2 or later. 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**.
IntelliJ IDEA	Tool used for developing IoTDB applications. The version must be 2019.1 or other compatible version. **NOTE** • 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 OpenJDK, ensure that the JDK configured in IntelliJ IDEA is OpenJDK. • Do not use the same workspace or the sample project in the same path for different IntelliJ IDEA programs.
Maven	Basic configurations of the development environment. Maven is used for project management throughout the lifecycle of software development. Huawei provides an open-source mirror site, Huawei Mirrors. You can download the dependent JAR packages of the sample projects from this site. You can download the rest open-source JAR packages from the Maven central repository or other user-defined repositories. For details, see **Configuring Huawei Open Source Mirrors**
7-zip	This tool is used to decompress ***.zip** and ***.rar** files. **7-zip 16.04** is supported.

20.2.2 Preparing the Configuration Files for Connecting to the Cluster

Preparing for User Authentication

For an MRS cluster with Kerberos authentication enabled, you need to prepare a user who has the operation permission on related components for program authentication.

The following IoTDB permission configuration example is for reference only. You can modify the configuration as you need.

Step 1 Log in to FusionInsight Manager.

Step 2 Choose **System** > **Permission** > **Role**. On the displayed page, click **Create Role**.

1. Enter the role name, for example, **developrole**.
2. In the **Configure Resource Permission** table, choose *Name of the desired cluster* > **IoTDB** > **Common User Permission**, and select **Set Database** for the **root** directory.
3. Click **root**, select the desired storage group, select **Create**, **Modify**, **Write**, **Read**, and **Delete**, and click **OK**.

Step 3 Choose **User** in the navigation pane and click **Create** on the displayed page. Create a machine-machine user, for example, **developuser**.

- Add the **iotdbgroup** user group to **User Group**.
- Add the new role created in **Step 2** to **Role**.

Step 4 Log in to FusionInsight Manager as user **admin** and choose **System** > **Permission** > **User**. In the **Operation** column of **developuser**, choose **More** > **Download Authentication Credential**. Save the file and decompress it to obtain the **user.keytab** and **krb5.conf** files of the user.

----End

Generating an SSL Certificate for the IoTDB Client

If SSL encrypted transmission is enabled for the cluster and this is your first time running IoTDB sample code in the local Windows or Linux environment, perform the following operations to generate a client SSL certificate:

Step 1 Log in to the node where the client is installed as the client installation user.

Step 2 Switch to the IoTDB client installation directory, for example, **/opt/client**.

cd /opt/client

Step 3 Configure environment variables.

source bigdata_env

Step 4 Generate the client SSL certificate **truststore.jks**.

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

You are required to set a password.

Step 5 To run the sample code in the Linux environment, copy the generated **truststore.jks** file to the *Client installation directory*/**IoTDB/iotdb/conf** directory.

cp truststore.jks *Client installation directory*/**IoTDB/iotdb/conf**

----End

Preparing the Configuration Files of the Running Environment

During the development or a test run of the program, you need to use cluster configuration files to connect to an MRS cluster. The configuration files usually contain the cluster component information file and user files used for security authentication. You can obtain the required information from the created MRS cluster.

Nodes used for program debugging or running must be able to communicate with nodes within the MRS cluster, and the hosts domain name must be configured.

- Scenario 1: Preparing the configuration files required for debugging in the local Windows development environment
  a. Download and decompress the client software package.
     - Versions earlier than MRS 3.3.0, log in to FusionInsight Manager and choose **Cluster** > **Dashboard** > **More** > **Download Client**. Set **Select Client Type** to **Configuration Files Only**, and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.
     - MRS 3.3.0 or later, log in to FusionInsight Manager, and click **Download Client** in the upper right corner of **Homepage**. Set **Select Client Type** to **Configuration Files Only**, and click **OK**. After the client file package is generated, download it 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**. Then, decompress **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar** Decompress the package to the **D:\FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles** directory on the local PC.
  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 is normal.

     NOTE
     - If the client host is outside the cluster, configure network connections to the client to prevent errors when you run commands on the client.
     - The local **hosts** file in a Windows environment is stored, for example, in **C:\WINDOWS\system32\drivers\etc\hosts**.
- Scenario 2: Preparing the configuration files required for running the program in a Linux environment

a. Install the client. For example, install the client in the **/opt/client** directory.

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

b. Download the client configuration file to the active OMS node of the cluster.

   - Versions earlier than MRS 3.3.0, log in to FusionInsight Manager and choose **Cluster** > **Dashboard** > **More** > **Download Client**. Set **Select Client Type** to **Configuration Files Only**, select **Save to Path**, and click **OK** to download the client configuration file to the active OMS node of the cluster.
   - MRS 3.3.0 or later, log in to FusionInsight Manager, and click **Download Client** in the upper right corner of **Homepage**. Set **Select Client Type** to **Configuration Files Only**, select **Save to Path**, and click **OK** to download the client configuration file to the active OMS node of the cluster.

c. Log in to the active OMS node as user **root**, go to the directory where the client configuration files are stored (**/tmp/FusionInsight-Client** by default), decompress the software package, and obtain all configuration files in the **IoTDB/config** directory. Place the files in the **conf** directory where the compiled JAR package is stored on the client node, for example, **/opt/client/conf**.

   For example, if the client software package is **FusionInsight_Cluster_1_Services_Client.tar** and it is downloaded to the **/tmp/FusionInsight-Client** directory on the active management node, run the following commands:

   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 IoTDB/config/* root@*IP address of the client node***:/opt/client/conf**

d. 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 there is no required information, copy the content of the **hosts** file in the decompression directory to the **hosts** file on the node where the client is deployed, to ensure that the local host can communicate with each host in the cluster.

20.2.3 Configuring and Importing a Sample Projects

Context

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

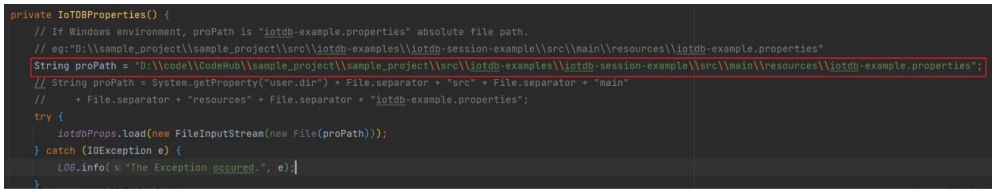
Procedure

Step 1 Obtain the sample project from the **src/iotdb-examples** directory in the directory where the sample code is decompressed by referring to **Obtaining Sample Projects from Huawei Mirrors**. You can select a sample based on the actual service scenario. For details about the samples, see **IoTDB Sample Project**.

Step 2 To debug **iotdb-flink-example**, **iotdb-jdbc-example**, **iotdb-kafka-example**, or **iotdb-session-example** sample code on the local Windows environment, perform the following operations:

- Store the authentication files **user.keytab** and **krb5.conf** obtained in **Preparing for User Authentication** and SSL certificate file **truststore.jks** to the **..\src\main\resources** directory of each sample project.
- Configure the **com.huawei.bigdata.iotdb.IoTDBProperties** class in the **..\src\main\resources** directory of each sample project, change the value of **proPath** in the **IoTDBProperties()** method to the absolute path of the **iotdb-example.properties** file.

  Figure 20-3 Configuring the proPath parameter



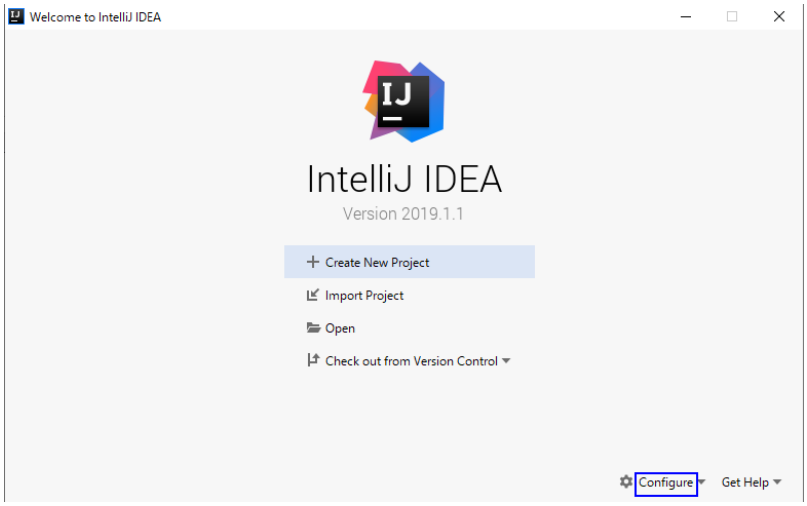
Step 3 Modify the **iotdb-example.properties** file in the **..\src\main\resources** directory of each sample project.

```
iotdb_ssl_enable=true
jdbc_url=jdbc:iotdb://IP address of the IoTDBServer instance:IoTDBServer RPC port
# Username for authentication
username=developuser
# User password for authentication. It is recommended that the password be stored in ciphertext and decrypted when being used.
password=xxx
# Location of the krb5.conf file in the downloaded authentication credential. In Linux, you are advised to use the krb5.conf file that has been uploaded to the IoTDB client installation directory/IoTDB/iotdb/conf directory. In Windows, use \\ to specify the absolute path of the file.
krb5_conf_dest=IoTDB client installation directory/iotdb/conf/krb5.conf
# Location of the user.keytab file in the downloaded authentication credential. In Linux, you are advised to use the user.keytab file that has been uploaded to the IoTDB client installation directory/IoTDB/iotdb/conf directory. In Windows, use \\ to specify the absolute path of the file.
key_tab_dest=IoTDB client installation directory/IoTDB/iotdb/conf/user.keytab
# Username corresponding to user.keytab followed by @HADOOP.COM
client_principal=developuser@HADOOP.COM
# Location of the truststore file. In Linux, you are advised to use the truststore.jks file that has been uploaded to the IoTDB client installation directory/IoTDB/iotdb/conf directory. In Windows, use \\ to specify the absolute path of the file.
iotdb_ssl_truststore=IoTDB client installation directory/IoTDB/iotdb/conf/truststore.jks
```

Step 4 After the IntelliJ IDEA and JDK tools are installed, configure the JDK in IntelliJ IDEA.

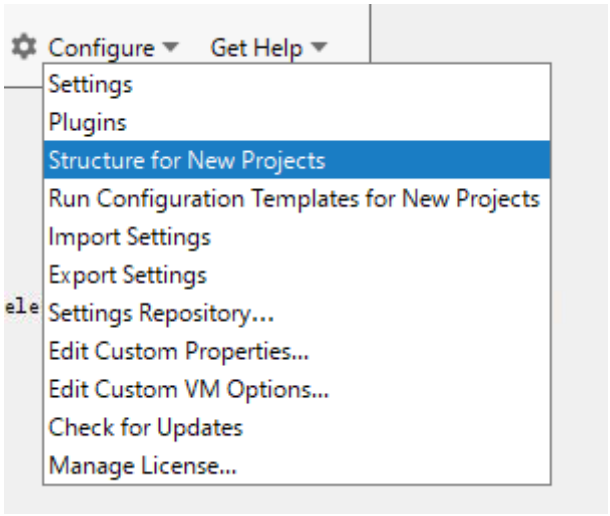
1. Start IntelliJ IDEA and choose **Configure**.

Figure 20-4 Quick Start



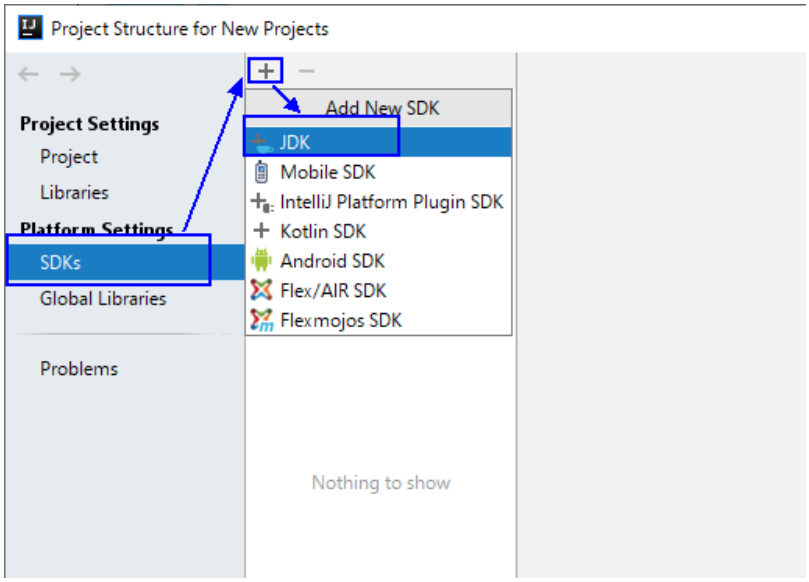
2. Select **Structure for New Projects** from the drop-down list.

Figure 20-5 Configure



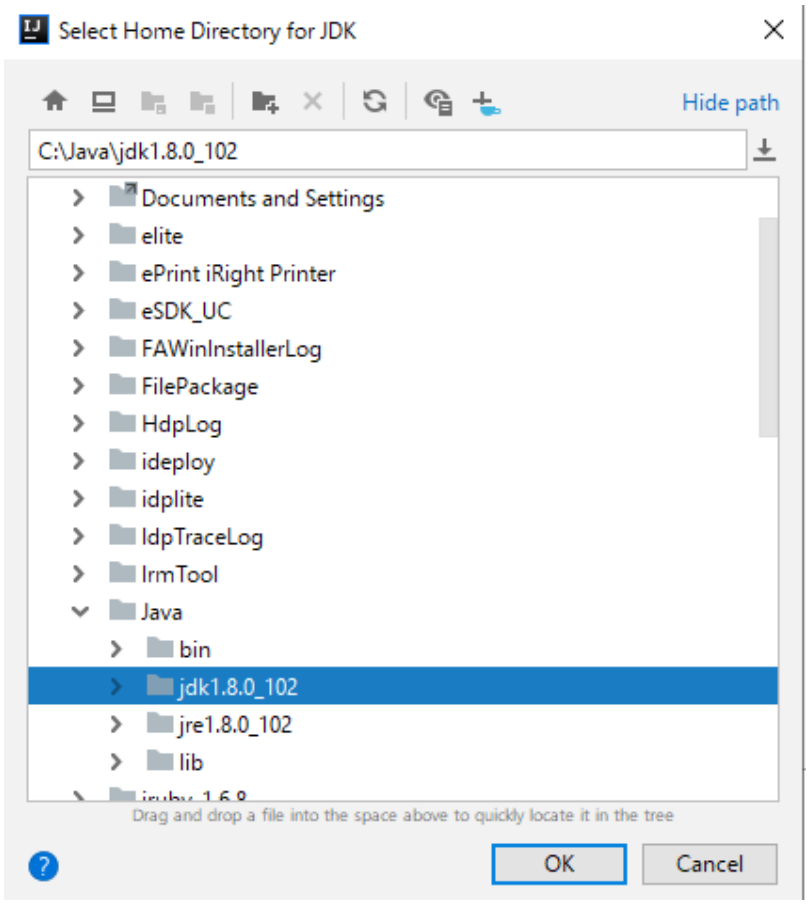
3. On the displayed **Project Structure for New Projects** page, select **SDKs**, click the plus sign (+), and click **JDK**.

Figure 20-6 Project Structure for New Projects



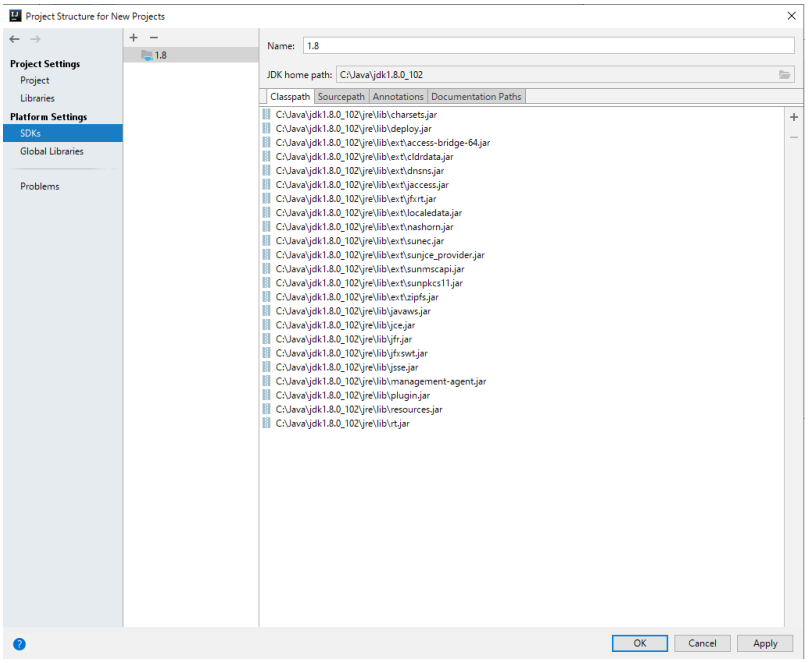
4. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 20-7 Select Home Directory for JDK



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

Figure 20-8 Completing the JDK configuration



NOTE

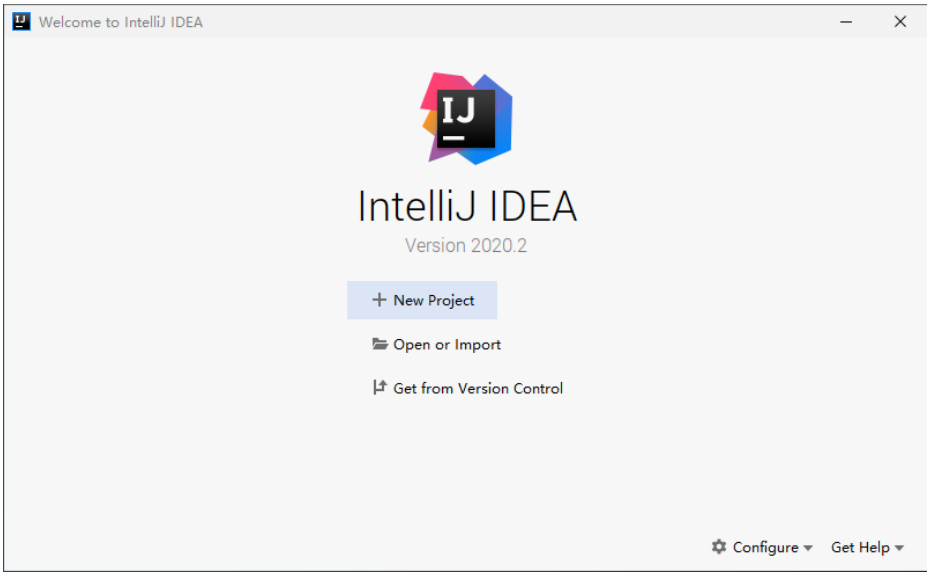
The operation procedure may vary according to the IDEA version.

Step 5 Import the sample project to the IntelliJ IDEA development environment.

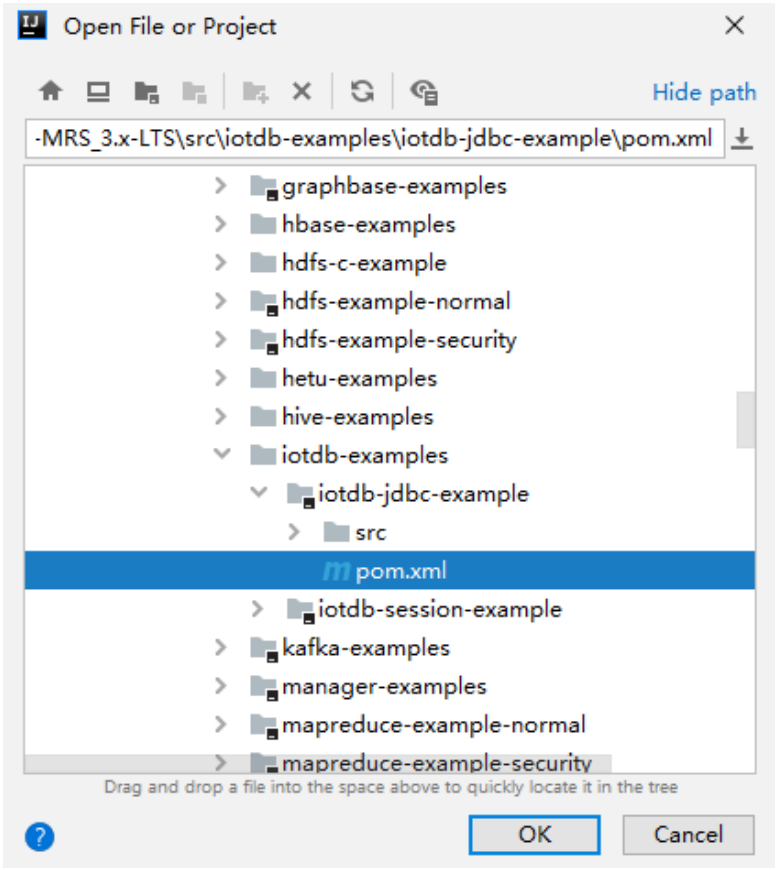
1. Start the IntelliJ IDEA, and click **Open or Import** on the quick start page.

   For the IDEA tool that has been used, you can choose **File** > **Import project...** on the main interface to import the sample project.

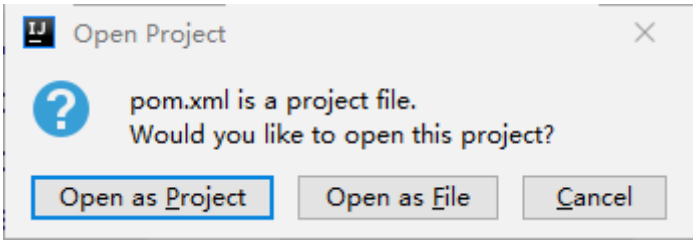
Figure 20-9 Open or Import (quick start page)



2. Select the **pom.xml** file of **iotdb-jdbc-example** in the sample project folder **iotdb-examples**, and click **OK**.

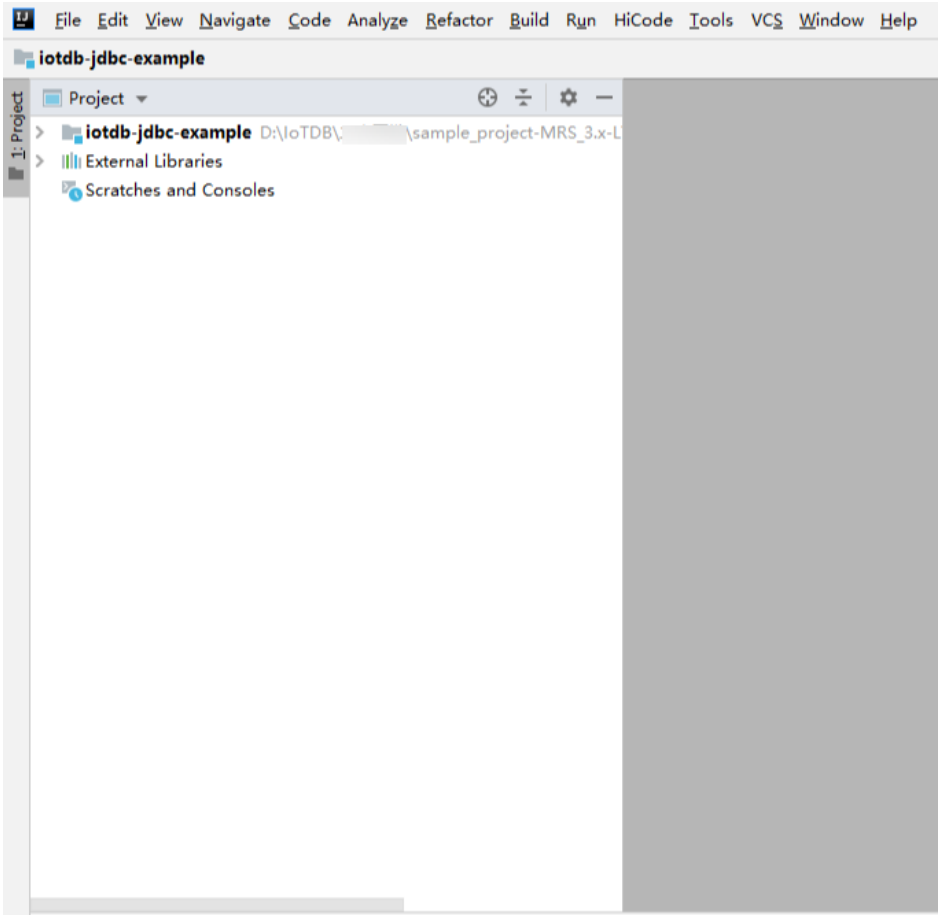


Select **Open as Project**.



3. After the import is complete, check that the imported sample projects are displayed on the IDEA home page.

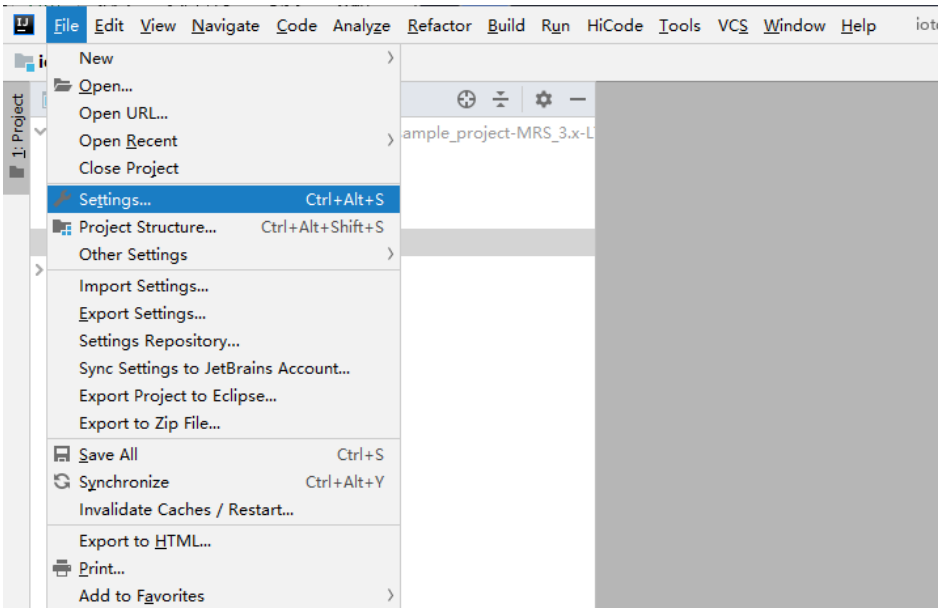
Figure 20-10 Successfully importing sample projects



Step 6 Set the Maven version used by the project.

1. Choose **File > Settings...** from the main menu of IntelliJ IDEA.

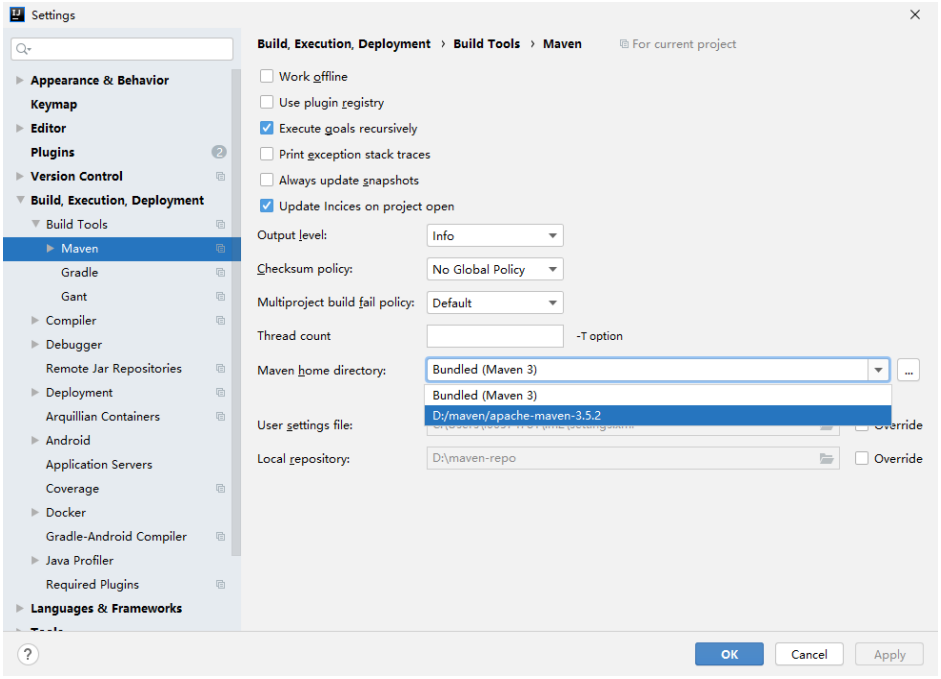
Figure 20-11 Settings



2. Choose **Build, Execution, Deployment** > **Maven** and set **Maven home directory** to the Maven version installed on the local host.

   Configure **User settings file** and **Local repository** based on the site requirements.

   Figure 20-12 Selecting the local Maven installation directory



3. Click **Apply** and **OK** to complete the configuration.

----End

20.3 Application Development

20.3.1 IoTDB JDBC

20.3.1.1 Java Example Code

Description

Run the IoTDB SQL statement in JDBC connection mode.

Sample Code

The following code snippet is used as an example. For complete code, see the **com.huawei.bigdata.iotdb.JDBCExample** class.

jdbc url includes the IP address, RPC port number, username, and password of the node where the IoTDBServer to be connected is located.

NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- To obtain the RPC port number, log in to FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB**, click **Configuration**, and click **All Configurations**, and search for **IOTDB_SERVER_RPC_PORT**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permission to operate the IoTDB service. For details, see **Preparing for User Authentication**.
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - *Authentication username* is the username for accessing IoTDB.
  - *Password* is the password for accessing IoTDB.

```
/**
 * In security mode, the default value of SSL_ENABLE is true. You need to import the truststore.jks file.
 * In security mode, you can also log in to FusionInsight Manager, choose Cluster > Services > IoTDB > Configuration, search for SSL in the search box, and change the value of SSL_ENABLE to false. After saving the configuration, restart the IoTDB service for the configuration to take effect. Modify the following configuration in the iotdb-client.env file in the Client installation directory/IoTDB/iotdb/conf directory on the client: iotdb_ssl_enable="false"
*/
    private static final String IOTDB_SSL_ENABLE = "true"; // Set it to the SSL_ENABLE value.
public static void main(String[] args) throws ClassNotFoundException, SQLException {
    Class.forName("org.apache.iotdb.jdbc.IoTDBDriver");
    // set iotdb_ssl_enable
    System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
    if ("true".equals(IOTDB_SSL_ENABLE)) {
      // set truststore.jks path
      System.setProperty("iotdb_ssl_truststore", "truststore file path");
    }
    try (Connection connection =
            DriverManager.getConnection("jdbc:iotdb://IP address of the IoTDBServer instance node:Port/",
"authentication username", "authentication user password");
         Statement statement = connection.createStatement()) {

      // set JDBC fetchSize
      statement.setFetchSize(10000);
      for (int i = 0; i <= 100; i++) {
        statement.addBatch(prepareInsertStatment(i));
      }
      statement.executeBatch();
      statement.clearBatch();

      ResultSet resultSet = statement.executeQuery("select ** from root where time <= 10");
      outputResult(resultSet);
      resultSet = statement.executeQuery("select count(**) from root");
      outputResult(resultSet);
      resultSet =
          statement.executeQuery(
              "select count(**) from root where time >= 1 and time <= 100 group by ([0, 100), 20ms, 20ms)");
      outputResult(resultSet);
    } catch (IoTDBSQLException e) {
      System.out.println(e.getMessage());
    }
  }
```

20.3.1.2 Using the keytab File for JDBC Authentication

Description

Use the keytab file for JDBC authentication.

Preparations

Log in to FusionInsight Manager, choose **System** > **Permission** > **User**, and download the user credential prepared in **Preparing the Developer Account**.

Sample Code

The following code snippet is used as an example. For complete code, see the **com.huawei.bigdata.iotdb.JDBCbyKerberosExample** class.

NOTE

- On FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB** > **Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- To obtain the RPC port number, log in to FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB**, click **Configuration**, and click **All Configurations**, and search for **IOTDB_SERVER_RPC_PORT**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permission to operate the IoTDB service. For details, see **Preparing the Developer Account**.
- Log in to FusionInsight Manager and choose **System** > **Permission** > **Domain and Mutual Trust**. On the page that is displayed, the value of **Local Domain** is the domain name.

```
package com.huawei.bigdata.iotdb;

import com.huawei.iotdb.client.security.IoTDBClientKerberosFactory;
import org.apache.iotdb.jdbc.IoTDBSQLException;
import org.ietf.jgss.GSSException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Base64;
import javax.security.auth.login.LoginException;

public class JDBCbyKerberosExample {

  /**
   * In security mode, the default value of SSL_ENABLE is true. You need to import the truststore.jks file.
   * In security mode, you can also log in to FusionInsight Manager, choose Cluster > Services > IoTDB > Configuration, search for SSL in the search box, and change the value of SSL_ENABLE to false. After saving the configuration, restart the IoTDB service for the configuration to take effect. Modify the following configuration in the iotdb-client.env file in the Client installation directory/IoTDB/iotdb/conf directory on the client: iotdb_ssl_enable="false"
   */
  private static final String IOTDB_SSL_ENABLE = "true"; // Set it to the SSL_ENABLE value.

  private static final String JAVA_KRB5_CONF = "java.security.krb5.conf";
  /**
   * Location of krb5.conf file
   */
  private static final String KRB5_CONF_DEST = "Location of the krb5.conf file in the downloaded authentication credential";
  /**
```

```
     * Location of keytab file
     */
    private static final String KEY_TAB_DEST = "Location of the user.keytab file in the downloaded
authentication credential";
    /**
     * User principal
     */
    private static final String CLIENT_PRINCIPAL = "User Principal (usually in the format of username@local
domain, for example, iotdb_admin@HADOOP.COM)";
   /**
    * Server principal, 'iotdb_server_kerberos_principal' in iotdb-datanode.properties
    */
    private static final String SERVER_PRINCIPAL = "iotdb/hadoop.hadoop.com@HADOOP.COM";// You can
obtain this parameter value by searching for iotdb_server_kerberos_principal in ${BIGDATA_HOME}/
FusionInsight_IoTDB_*/*_*_IoTDBServer/etc/iotdb-datanode.properties on any node with the IoTDB service
installed.

    /**
     * Get kerberos token as password
     * @return kerberos token
     * @throws LoginException loginException
     * @throws GSSException GSSException
     */
    public static String getAuthToken() throws LoginException, GSSException {
        IoTDBClientKerberosFactory kerberosHandler = IoTDBClientKerberosFactory.getInstance();
        System.setProperty(JAVA_KRB5_CONF, KRB5_CONF_DEST);
        kerberosHandler.loginSubjectFromKeytab(PRINCIPAL, KEY_TAB_DEST);
        byte[] tokens = kerberosHandler.generateServiceToken(PRINCIPAL);
        return Base64.getEncoder().encodeToString(tokens);
    }

    public static void main(String[] args) throws SQLException {
        // set iotdb_ssl_enable
        System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
        if ("true".equals(IOTDB_SSL_ENABLE)) {
          // set truststore.jks path
          System.setProperty("iotdb_ssl_truststore", "truststore file path");
        }

        try (Connection connection =
                    DriverManager.getConnection("jdbc:iotdb://IP address of the IoTDBServer instance
node:port number/", "Authentication username", getAuthToken());
                Statement statement = connection.createStatement()) {
            // set JDBC fetchSize
            statement.setFetchSize(10000);

            try {
                statement.execute("SET STORAGE GROUP TO root.sg1");
                statement.execute(
                        "CREATE TIMESERIES root.sg1.d1.s1 WITH DATATYPE=INT64, ENCODING=RLE,
COMPRESSOR=SNAPPY");
                statement.execute(
                        "CREATE TIMESERIES root.sg1.d1.s2 WITH DATATYPE=INT64, ENCODING=RLE,
COMPRESSOR=SNAPPY");
                statement.execute(
                        "CREATE TIMESERIES root.sg1.d1.s3 WITH DATATYPE=INT64, ENCODING=RLE,
COMPRESSOR=SNAPPY");
            } catch (IoTDBSQLException e) {
                System.out.println(e.getMessage());
            }
        } catch (GSSException | LoginException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

20.3.2 IoTDB Session

20.3.2.1 Java Example Code

Description

Run the IoTDB SQL statement in Session connection mode.

Sample Code

The following code snippet is used as an example. For complete code, see **com.huawei.bigdata.SessionExample**.

Change **HOST_1**, **HOST_2**, and **HOST_3** to the service IP addresses of the nodes accommodating IoTDBServer. Set the username and password in the **Session** object.

NOTE

- On Manager, choose **Cluster** > **Services** > **IoTDB** > **Instance** to view the service IP addresses of the nodes where the IoTDBServer to be connected is located.
- To obtain the RPC port number, log in to FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB**, click **Configuration**, and click **All Configurations**, and search for **IOTDB_SERVER_RPC_PORT**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permission to operate the IoTDB service. For details, see **Preparing for User Authentication**.
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - *Authentication username* is the username for accessing IoTDB.
  - *Password* is the password for accessing IoTDB.

```
  /**
   * In security mode, the default value of SSL_ENABLE is true. You need to import the truststore.jks file.
   * In security mode, you can also log in to FusionInsight Manager, choose Cluster > Services > IoTDB > Configuration, search for SSL in the search box, and change the value of SSL_ENABLE to false. After saving the configuration, restart the IoTDB service for the configuration to take effect. Modify the following configuration in the iotdb-client.env file in the Client installation directory/IoTDB/iotdb/conf directory on the client: iotdb_ssl_enable="false"
   */
   private static final String IOTDB_SSL_ENABLE = "true"; // Set it to the SSL_ENABLE value.
public static void main(String[] args)
    throws IoTDBConnectionException, StatementExecutionException {
  // set iotdb_ssl_enable
  System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
  if ("true".equals(IOTDB_SSL_ENABLE)) {
    // set truststore.jks path
    System.setProperty("iotdb_ssl_truststore", "truststore file path");
  }

  session = new Session(nodeUrls, Port, Authentication username,Authentication user password);
  session.open(false);

  // set session fetchSize
  session.setFetchSize(10000);

  try {
    session.setStorageGroup("root.sg1");
  } catch (StatementExecutionException e) {
    if (e.getStatusCode() != TSStatusCode.PATH_ALREADY_EXIST_ERROR.getStatusCode()) {
      throw e;
    }
```



```
    }

    createTimeseries();
    createMultiTimeseries();
    insertRecord();
    insertTablet();
    insertTablets();
    insertRecords();
    nonQuery();
    query();
    queryWithTimeout();
    rawDataQuery();
    queryByIterator();
    deleteData();
    deleteTimeseries();
    setTimeout();

    sessionEnableRedirect = new Session(nodeUrls, Authentication username, Authentication user password);
    sessionEnableRedirect.setEnableQueryRedirection(true);
    sessionEnableRedirect.open(false);

    // set session fetchSize
    sessionEnableRedirect.setFetchSize(10000);

    insertRecord4Redirect();
    query4Redirect();
    sessionEnableRedirect.close();
    session.close();
}
```

20.3.2.2 Using the Keytab File for Session Authentication

Description

Use the Keytab file for session authentication.

Preparations

Log in to FusionInsight Manager, choose **System** > **Permission** > **User**, and download the user credential prepared in **Preparing the Developer Account**.

Sample Code

The following code snippet is used as an example. For complete code, see the **com.huawei.bigdata.iotdb.SessionbyKerberosExample** class.

NOTE

- On FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB** > **Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- To obtain the RPC port number, log in to FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB**, click **Configuration**, and click **All Configurations**, and search for **IOTDB_SERVER_RPC_PORT**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permission to operate the IoTDB service. For details, see **Preparing the Developer Account**.
- Log in to FusionInsight Manager and choose **System** > **Permission** > **Domain and Mutual Trust**. On the page that is displayed, the value of **Local Domain** is the domain name.


```
package com.huawei.bigdata.iotdb;

import org.apache.iotdb.rpc.IoTDBConnectionException;
import org.apache.iotdb.rpc.StatementExecutionException;
import org.apache.iotdb.rpc.TSStatusCode;
import org.apache.iotdb.session.Session;
import org.apache.iotdb.tsfile.file.metadata.enums.CompressionType;
import org.apache.iotdb.tsfile.file.metadata.enums.TSDataType;
import org.apache.iotdb.tsfile.file.metadata.enums.TSEncoding;
import org.ietf.jgss.GSSException;
import java.util.Base64;
import javax.security.auth.login.LoginException;

public class SessionbyKerberosExample{
    private static Session session;
    private static final String ROOT_SG1_D1_S1 = "root.sg1.d1.s1";
    private static final String ROOT_SG1_D1_S2 = "root.sg1.d1.s2";
    private static final String ROOT_SG1_D1_S3 = "root.sg1.d1.s3";
    private static final String HOST = "127.0.0.1";
    private static final String PORT = "22260";

/**
 * In security mode, the default value of SSL_ENABLE is true. You need to import the truststore.jks file.
 * In security mode, you can also log in to FusionInsight Manager, choose Cluster > Services > IoTDB > Configuration, search for SSL in the search box, and change the value of SSL_ENABLE to false. After saving the configuration, restart the IoTDB service for the configuration to take effect. Modify the following configuration in the iotdb-client.env file in the Client installation directory/IoTDB/iotdb/conf directory on the client: iotdb_ssl_enable="false"
 */
    private static final String IOTDB_SSL_ENABLE = "true"; // Set it to the SSL_ENABLE value.

    private static final String JAVA_KRB5_CONF = "java.security.krb5.conf";
    /**
     * Location of krb5.conf file
     */
    private static final String KRB5_CONF_DEST = "Location of the krb5.conf file in the downloaded authentication credential";
    /**
     * Location of keytab file
     */
    private static final String KEY_TAB_DEST = "Location of the user.keytab file in the downloaded authentication credential";
    /**
     * User principal
     */

    private static final String CLIENT_PRINCIPAL = "User Principal (usually in the format of username@local domain, for example, iotdb_admin@HADOOP.COM)";
   /**
    * Server principal, 'iotdb_server_kerberos_principal' in iotdb-datanode.properties
    */
    private static final String SERVER_PRINCIPAL = "iotdb/hadoop.hadoop.com@HADOOP.COM";// You can obtain this parameter value by searching for iotdb_server_kerberos_principal in /opt/huawei/Bigdata/FusionInsight_IoTDB_*/*_*_IoTDBServer/etc/iotdb-datanode.properties on any node with the IoTDB service installed.

    /**
     * Get kerberos token as password
     * @return kerberos token
     * @throws LoginException loginException
     * @throws GSSException GSSException
     */
    public static String getAuthToken() throws LoginException, GSSException {
        IoTDBClientKerberosFactory kerberosHandler = IoTDBClientKerberosFactory.getInstance();
        System.setProperty(JAVA_KRB5_CONF, KRB5_CONF_DEST);
        kerberosHandler.loginSubjectFromKeytab(PRINCIPAL, KEY_TAB_DEST);
        byte[] tokens = kerberosHandler.generateServiceToken(PRINCIPAL);
        return Base64.getEncoder().encodeToString(tokens);
    }
```

```
public static void main(String[] args)
throws IoTDBConnectionException, StatementExecutionException, GSSException, LoginException {
// set iotdb_ssl_enable
System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
if ("true".equals(IOTDB_SSL_ENABLE)) {
    // set truststore.jks path
    System.setProperty("iotdb_ssl_truststore", "truststore file path");
}

session = new Session(HOST, PORT," Authentication username", getAuthToken());
session.open(false);

// set session fetchSize
session.setFetchSize(10000);

try {
    session.setStorageGroup("root.sg1");
} catch (StatementExecutionException e) {
    if (e.getStatusCode() != TSStatusCode.PATH_ALREADY_EXIST_ERROR.getStatusCode()) {
        throw e;
    }
}
createTimeseries();
}

private static void createTimeseries() throws IoTDBConnectionException, StatementExecutionException {
if (!session.checkTimeseriesExists(ROOT_SG1_D1_S1)) {
    session.createTimeseries(
        ROOT_SG1_D1_S1, TSDataType.INT64, TSEncoding.RLE, CompressionType.SNAPPY);
}
if (!session.checkTimeseriesExists(ROOT_SG1_D1_S2)) {
    session.createTimeseries(
        ROOT_SG1_D1_S2, TSDataType.INT64, TSEncoding.RLE, CompressionType.SNAPPY);
}
if (!session.checkTimeseriesExists(ROOT_SG1_D1_S3)) {
    session.createTimeseries(
        ROOT_SG1_D1_S3, TSDataType.INT64, TSEncoding.RLE, CompressionType.SNAPPY);
}
}
}
```

20.3.3 IoTDB Flink

20.3.3.1 FlinkIoTDBSink

Description

It is an integration of IoTDB and Flink. This module contains IoTDBSink and writes time series data into IoTDB using a Flink job.

Sample Code

This example shows how to send data from a Flink job to an IoTDB server.

- A simulated source SensorSource generates a data point every second.
- Flink uses IoTDBSink to consume produced data and write the data into IoTDB.

Session object parameters include the IP address, port number, username, and password of the node where the IoTDBServer is located.

NOTE

- On FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB** > **Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- To obtain the RPC port number, log in to FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB**, click **Configuration**, and click **All Configurations**, and search for **IOTDB_SERVER_RPC_PORT**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permissions to operate the IoTDB and Flink services. For details, see **Preparing for User Authentication**.
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - *Authentication username* is the username for accessing IoTDB.
  - *Password* is the password for accessing IoTDB.

```
public class FlinkIoTDBSink {
  public static void main(String[] args) throws Exception {
    // run the flink job on local mini cluster
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

    IoTDBSinkOptions options = new IoTDBSinkOptions();
    options.setHost("127.0.0.1");
    options.setPort(22260);
    options.setUser("Username");
    options.setPassword("Password");

    // If the server enables auto_create_schema, then we do not need to register all timeseries
    // here.
    options.setTimeseriesOptionList(
            Lists.newArrayList(
                    new IoTDBSinkOptions.TimeseriesOption(
                            "root.sg.d1.s1", TSDataType.DOUBLE, TSEncoding.GORILLA, CompressionType.SNAPPY)));

    IoTSerializationSchema serializationSchema = new DefaultIoTSerializationSchema();
    IoTDBSink ioTDBSink =
        new IoTDBSink(options, serializationSchema)
            // enable batching
            .withBatchSize(10)
            // how many connections to the server will be created for each parallelism
            .withSessionPoolSize(3);

    env.addSource(new SensorSource())
        .name("sensor-source")
        .setParallelism(1)
        .addSink(ioTDBSink)
        .name("iotdb-sink");

    env.execute("iotdb-flink-example");
  }

  private static class SensorSource implements SourceFunction<Map<String, String>> {
    boolean running = true;
    Random random = new SecureRandom();

    @Override
    public void run(SourceContext context) throws Exception {
      while (running) {
        Map<String, String> tuple = new HashMap();
        tuple.put("device", "root.sg.d1");
        tuple.put("timestamp", String.valueOf(System.currentTimeMillis()));
        tuple.put("measurements", "s1");
        tuple.put("types", "DOUBLE");
        tuple.put("values", String.valueOf(random.nextDouble()));
```



```
      context.collect(tuple);
      Thread.sleep(1000);
    }
  }

  @Override
  public void cancel() {
    running = false;
  }
 }
}
```

20.3.3.2 FlinkIoTDBSource

Description

It is an integration of IoTDB and Flink. This module contains IoTDBSource and reads time series data from IoTDB and prints the data using a Flink job.

Sample Code

This example shows how a Flink job reads time series data from a IoTDB server.

- Flink uses IoTDBSource to read data from a IoTDB server.
- To use IoTDBSource, you need to construct an IoTDBSource instance by specifying **IoTDBSourceOptions** and implementing the abstract method **convert()** in IoTDBSource. The **convert()** method defines how you want to convert row data.

Session object parameters include the IP address, port number, username, and password of the node where the IoTDBServer is located.

NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- To obtain the RPC port number, log in to FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB**, click **Configuration**, and click **All Configurations**, and search for **IOTDB_SERVER_RPC_PORT**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permissions to operate the IoTDB and Flink services. For details, see **Preparing for User Authentication**.
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - *Authentication username* is the username for accessing IoTDB.
  - *Password* is the password for accessing IoTDB.

```
public class FlinkIoTDBSource {
/**
 * In security mode, the default value of SSL_ENABLE is true. You need to import the truststore.jks file.
 * In security mode, you can also log in to FusionInsight Manager, choose Cluster > Services > IoTDB > Configuration, search for SSL in the search box, and change the value of SSL_ENABLE to false. After saving the configuration, restart the IoTDB service for the configuration to take effect. Modify the following configuration in the iotdb-client.env file in the Client installation directory/IoTDB/iotdb/conf directory on the client: iotdb_ssl_enable="false"
 */
  private static final String IOTDB_SSL_ENABLE = "true"; // Set it to the SSL_ENABLE value.
  static final String LOCAL_HOST = "127.0.0.1";
```



```
  static final String ROOT_SG1_D1_S1 = "root.sg1.d1.s1";
  static final String ROOT_SG1_D1 = "root.sg1.d1";

  public static void main(String[] args) throws Exception {
    // use session api to create data in IoTDB
    prepareData();

    // run the flink job on local mini cluster
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

    IoTDBSourceOptions ioTDBSourceOptions =
        new IoTDBSourceOptions(
            LOCAL_HOST, 22260, "Authentication username", "Password", "select s1 from " + ROOT_SG1_D1 +
" align by device");

    IoTDBSource<RowRecord> source =
        new IoTDBSource<RowRecord>(ioTDBSourceOptions) {
          @Override
          public RowRecord convert(RowRecord rowRecord) {
            return rowRecord;
          }
        };
    env.addSource(source).name("sensor-source").print().setParallelism(2);
    env.execute();
  }

  private static void prepareData()
      throws IoTDBConnectionException, StatementExecutionException, TTransportException {
    // set iotdb_ssl_enable
    System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
    if ("true".equals(IOTDB_SSL_ENABLE)) {
      // set truststore.jks path
      System.setProperty("iotdb_ssl_truststore", "truststore file path");
    }
    Session session = new Session(LOCAL_HOST, 22260, "Authentication username", "Password");
    session.open(false);
    try {
      session.setStorageGroup("root.sg1");
      if (!session.checkTimeseriesExists(ROOT_SG1_D1_S1)) {
        session.createTimeseries(
            ROOT_SG1_D1_S1, TSDataType.INT64, TSEncoding.RLE, CompressionType.SNAPPY);
        List<String> measurements = new ArrayList<>();
        measurements.add("s1");
        measurements.add("s2");
        measurements.add("s3");
        List<TSDataType> types = new ArrayList<>();
        types.add(TSDataType.INT64);
        types.add(TSDataType.INT64);
        types.add(TSDataType.INT64);

        for (long time = 0; time < 1000; time++) {
          List<Object> values = new ArrayList<>();
          values.add(1L);
          values.add(2L);
          values.add(3L);
          session.insertRecord(ROOT_SG1_D1, time, measurements, types, values);
        }
      }
    } catch (StatementExecutionException e) {
      if (e.getStatusCode() != TSStatusCode.PATH_ALREADY_EXIST_ERROR.getStatusCode()) {
        throw e;
      }
    }
  }
}
```

20.3.4 IoTDB Kafka

20.3.4.1 Java Example Code

Description

This section describes how to use Kafka to send data to IoTDB.

Sample Code

- Producer.java:

  This example shows how to send time series data to a Kafka cluster.

  a. Change the value of the **public final static String TOPIC** variable in the **KafkaProperties.java** file based on the site requirements, for example, **kafka-topic**.

  b. The default time series data format of this sample is *Device name*,*Timestamp*,*Value*, for example, **sensor_1,1642215835758,1.0**. You can change the value of **IOTDB_DATA_SAMPLE_TEMPLATE** in the **Constant.java** file based on the site requirements.

```
public static void main(String[] args) {
    // whether use security mode
    final boolean isSecurityModel = true;

    if (isSecurityModel) {
        try {
            LOG.info("Security mode start.");

            // Note: During security authentication, you need to manually change the account to a machine-machine one that you have applied for.
            LoginUtil.securityPrepare(Constant.USER_PRINCIPAL, Constant.USER_KEYTAB_FILE);
        } catch (IOException e) {
            LOG.error("Security prepare failure.", e);
            return;
        }
        LOG.info("Security prepare success.");
    }

    // whether to use the asynchronous sending mode
    final boolean asyncEnable = false;
    Producer producerThread = new Producer(KafkaProperties.TOPIC, asyncEnable);
}
```

  NOTE

  For details about the Kafka producer code, see **Producer API Sample**.

- KafkaConsumerMultThread.java:

  This example shows how to write data from a Kafka cluster to IoTDB using multiple threads. Kafka cluster data is generated by **Producer.java**.

  a. Change the value of the **public final static String TOPIC** variable in the **KafkaProperties.java** file based on the site requirements, for example, **kafka-topic**.

  b. Change the value of **CONCURRENCY_THREAD_NUM** in the **KafkaConsumerMultThread.java** file to adjust the number of consumer threads.

NOTICE

If multiple threads are used to consume Kafka cluster data, ensure that the number of consumed topic partitions is greater than 1.

c. Set the IP address, port number, username, and password of the node where the IoTDBServer is located in the IoTDBSessionPool object parameters.

NOTE

- On FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB** and click the **Instance** tab to view the IP address of the node where the IoTDBServer to be connected is located.
- To obtain the RPC port number, log in to FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB**, click **Configuration**, and click **All Configurations**, and search for **IOTDB_SERVER_RPC_PORT**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permission to operate the IoTDB service. For details, see **Preparing for User Authentication**.
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - *Authentication username* is the username for accessing IoTDB.
  - *Password* is the password for accessing IoTDB.

```
/**
 * In security mode, the default value of SSL_ENABLE is true. You need to import the truststore.jks file.
 * In security mode, you can also log in to FusionInsight Manager, choose Cluster > Services > IoTDB > Configuration, search for SSL in the search box, and change the value of SSL_ENABLE to false. After saving the configuration, restart the IoTDB service for the configuration to take effect. Modify the following configuration in the iotdb-client.env file in the Client installation directory/IoTDB/iotdb/conf directory on the client: iotdb_ssl_enable="false"
 */
    private static final String IOTDB_SSL_ENABLE = "true"; // Set it to the SSL_ENABLE value.
public static void main(String[] args) {
    // whether use security mode
    final boolean isSecurityModel = true;

    if (isSecurityModel) {
        try {
            LOG.info("Securitymode start.");

            // Note: During security authentication, you need to manually change the account to a machine-machine one.
            LoginUtil.securityPrepare(Constant.USER_PRINCIPAL, Constant.USER_KEYTAB_FILE);
        } catch (IOException e) {
            LOG.error("Security prepare failure.", e);
            return;
        }
        LOG.info("Security prepare success.");
    }
    // set iotdb_ssl_enable
    System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
    if ("true".equals(IOTDB_SSL_ENABLE)) {
      // set truststore.jks path
      System.setProperty("iotdb_ssl_truststore", "truststore file path");
    }

    // create IoTDB session connection pool
    IoTDBSessionPool sessionPool = new IoTDBSessionPool("127.0.0.1", 22260, "Authentication
```

```
username", "Password", 3);

    // start consumer thread
    KafkaConsumerMultThread consumerThread = new
KafkaConsumerMultThread(KafkaProperties.TOPIC, sessionPool);
    consumerThread.start();
}

    /**
     * consumer thread
     */
    private class ConsumerThread extends ShutdownableThread {
        private int threadNum;
        private String topic;
        private KafkaConsumer<String, String> consumer;
        private IoTDBSessionPool sessionPool;

        public ConsumerThread(int threadNum, String topic, Properties props, IoTDBSessionPool
sessionPool) {
            super("ConsumerThread" + threadNum, true);
            this.threadNum = threadNum;
            this.topic = topic;
            this.sessionPool = sessionPool;
            this.consumer = new KafkaConsumer<>(props);
            consumer.subscribe(Collections.singleton(this.topic));
        }

        public void doWork() {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(waitTime));
            for (ConsumerRecord<String, String> record : records) {
                LOG.info("Consumer Thread-" + this.threadNum + " partitions:" + record.partition() + "
record: "
                        + record.value() + " offsets: " + record.offset());

                // insert kafka consumer data to iotdb
                sessionPool.insertRecord(record.value());
            }
        }
    }
```

NOTE

For details about the Kafka consumer code, see **Multi-thread Producer Sample**.

20.3.5 IoTDB UDF Program

20.3.5.1 IoTDB UDF Sample Code

Description

This section describes how to implement a simple IoTDB user-defined function (UDF). For details, see section **UDF Sample Code and Operations**.

Sample Code

```
package com.huawei.bigdata.iotdb;
import org.apache.iotdb.udf.api.UDTF;
import org.apache.iotdb.udf.api.access.Row;
import org.apache.iotdb.udf.api.collector.PointCollector;
import org.apache.iotdb.udf.api.customizer.config.UDTFConfigurations;
import org.apache.iotdb.udf.api.customizer.parameter.UDFParameters;
import org.apache.iotdb.udf.api.customizer.strategy.RowByRowAccessStrategy;
import org.apache.iotdb.udf.api.type.Type;
import java.io.IOException;
```



```
public class UDTFExample implements UDTF {
    @Override
     public void beforeStart(UDFParameters parameters, UDTFConfigurations configurations) {
       configurations.setAccessStrategy(new RowByRowAccessStrategy()).setOutputDataType(Type.INT32);
    }

    @Override
    public void transform(Row row, PointCollector collector) throws IOException {
      collector.putInt(row.getTime(), -row.getInt(0));
    }
}
```

20.4 Application Commissioning

20.4.1 Commissioning Applications on Windows

20.4.1.1 Compiling and Running Applications

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.

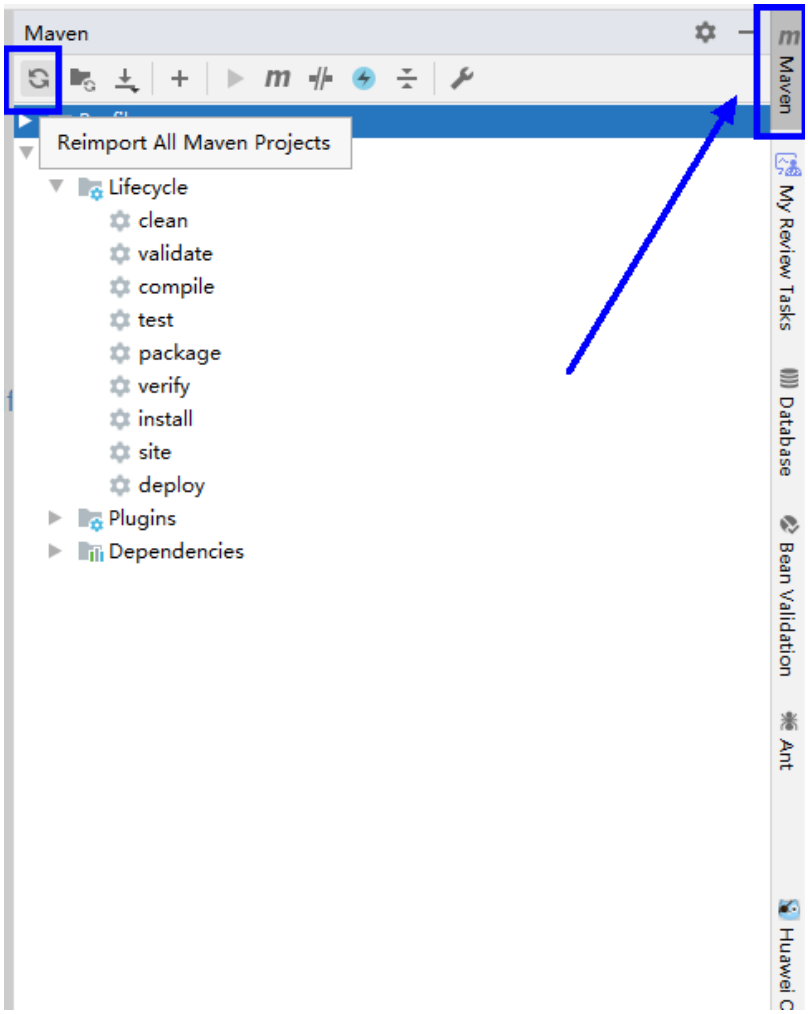
NOTE

- If the IBM JDK is used in the Windows environment, applications cannot be directly 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 20-13 reimport projects



Step 2 Compile and run the application.

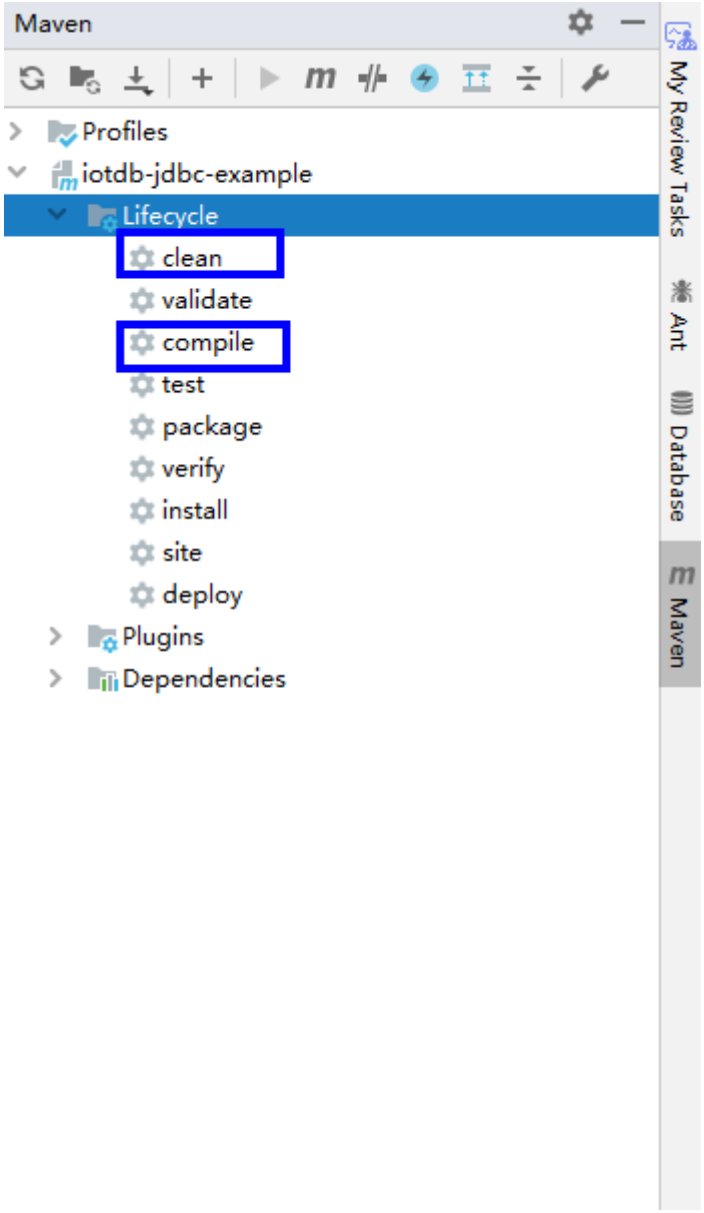
Modify the IP address, port number, login username, and password of the IoTDBServer node that matches the code.

1. Use either of the following two methods:
   - 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** and double-click **compile** to run the **compile** command of Maven.

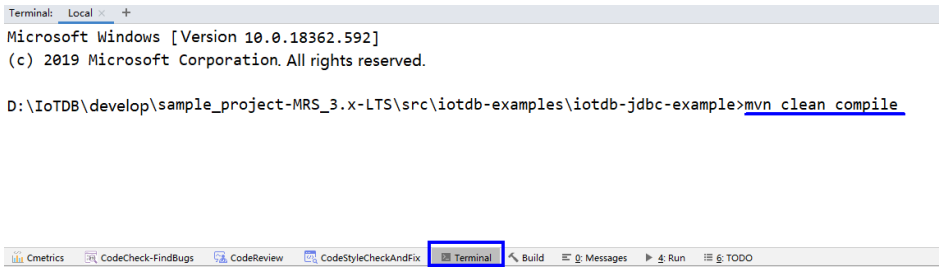
Figure 20-14 Maven tools clean and compile



- Method 2

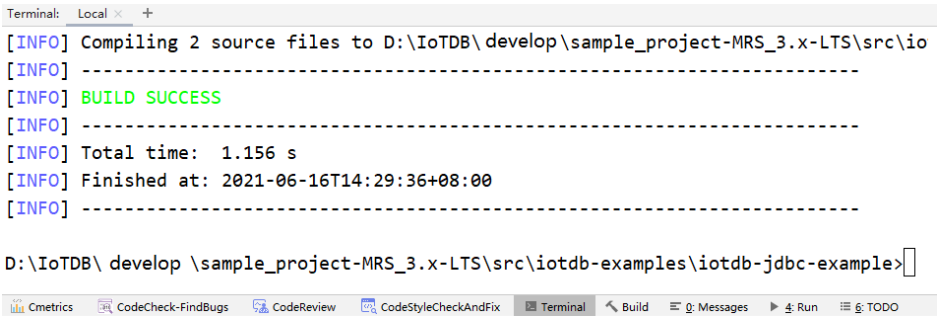
  Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

Figure 20-15 Enter **mvn clean compile** in the IDEA **Terminal** text box.



After the compilation is complete, the message "BUILD SUCCESS" is displayed.

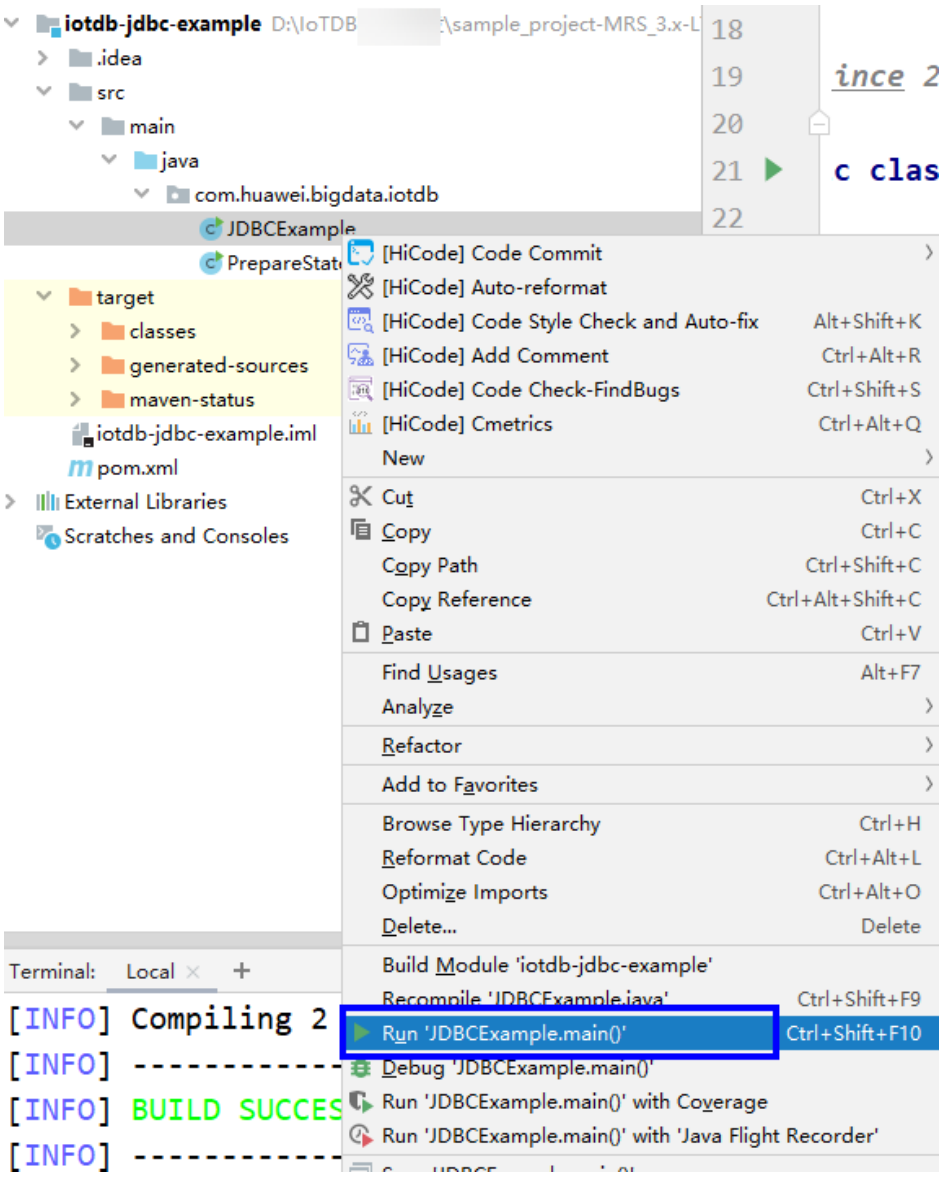
Figure 20-16 Compilation completed



2. Run the application. A JDBC application is used as an example. The operations for running other applications are the same.

   Right-click the **JDBCExample.java** file and choose **Run 'JDBCExample.main()** from the shortcut menu.

Figure 20-17 Running the application



----End

20.4.1.2 Viewing Commissioning Results

After the IoTDB application is run, you can view the running results in IntelliJ IDEA.

- The running result of the JDBCExample example application is as follows:

```
...
-------------------------
Time root.sg.d1.s1 root.company.line2.device1.temperature root.company.line2.device1.speed
root.company.line2.device2.speed root.company.line2.device2.status root.company.line1.device1.spin
root.company.line1.device1.status root.company.line1.device2.temperature
root.company.line1.device2.power root.sg1.d1.s3 root.sg1.d1.s1 root.sg1.d1.s2
0, null, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
1, null, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
2, null, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
3, null, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
4, null, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
```



```
5, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
6, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
7, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
8, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
9, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
10, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
--------------------------

--------------------------
count(root.sg.d1.s1) count(root.company.line2.device1.temperature)
count(root.company.line2.device1.speed) count(root.company.line2.device2.speed)
count(root.company.line2.device2.status) count(root.company.line1.device1.spin)
count(root.company.line1.device1.status) count(root.company.line1.device2.temperature)
count(root.company.line1.device2.power) count(root.sg1.d1.s3) count(root.sg1.d1.s1)
count(root.sg1.d1.s2)
8237, 1, 1, 1, 1, 1, 1, 1, 101, 101, 101
--------------------------

--------------------------
Time count(root.sg.d1.s1) count(root.company.line2.device1.temperature)
count(root.company.line2.device1.speed) count(root.company.line2.device2.speed)
count(root.company.line2.device2.status) count(root.company.line1.device1.spin)
count(root.company.line1.device1.status) count(root.company.line1.device2.temperature)
count(root.company.line1.device2.power) count(root.sg1.d1.s3) count(root.sg1.d1.s1)
count(root.sg1.d1.s2)
0, 0, 0, 0, 0, 0, 0, 0, 0, 19, 19, 19
20, 0, 0, 0, 0, 0, 0, 0, 0, 20, 20, 20
40, 0, 0, 0, 0, 0, 0, 0, 0, 20, 20, 20
60, 0, 0, 0, 0, 0, 0, 0, 0, 20, 20, 20
80, 0, 0, 0, 0, 0, 0, 0, 0, 20, 20, 20
--------------------------
```

- The running result of the FlinkIoTDBSink example application is as follows:

```
...
19:53:41.532 [flink-akka.actor.default-dispatcher-9] DEBUG
org.apache.flink.runtime.resourcemanager.StandaloneResourceManager - Received heartbeat from
5153e4ff24b25b13225f1bf67a4312d8.
19:53:41.800 [flink-akka.actor.default-dispatcher-9] DEBUG
org.apache.flink.runtime.jobmaster.JobMaster - Trigger heartbeat request.
19:53:41.800 [flink-akka.actor.default-dispatcher-10] DEBUG
org.apache.flink.runtime.taskexecutor.TaskExecutor - Received heartbeat request from
5153e4ff24b25b13225f1bf67a4312d8.
19:53:41.802 [flink-akka.actor.default-dispatcher-9] DEBUG
org.apache.flink.runtime.jobmaster.JobMaster - Received heartbeat from 7d6ef313-3f78-4cee-bbb1-
e234dcac6d30.
19:53:42.988 [pool-3-thread-1] DEBUG org.apache.iotdb.flink.IoTDBSink - send event successfully
19:53:42.988 [pool-6-thread-1] DEBUG org.apache.iotdb.flink.IoTDBSink - send event successfully
19:53:42.990 [pool-4-thread-1] DEBUG org.apache.iotdb.flink.IoTDBSink - send event successfully
19:53:45.990 [pool-7-thread-1] DEBUG org.apache.iotdb.flink.IoTDBSink - send event successfully
19:53:45.992 [pool-9-thread-1] DEBUG org.apache.iotdb.flink.IoTDBSink - send event successfully
19:53:45.994 [pool-5-thread-1] DEBUG org.apache.iotdb.flink.IoTDBSink - send event successfully
```

- The running result of the IoTDB Kafka example application is as follows:
  - Producer.java

```
...
[2022-01-15 15:12:34,221] INFO New Producer: start. (com.huawei.bigdata.iotdb.Producer)
[2022-01-15 15:12:39,369] INFO [Producer clientId=DemoProducer] Cluster ID:
uDtuaWS_QUK02EtuZQ4Xew (org.apache.kafka.clients.Metadata)
[2022-01-15 15:12:57,077] INFO The Producer have send 100 messages.
(com.huawei.bigdata.iotdb.Producer)
[2022-01-15 15:13:04,691] INFO The Producer have send 200 messages.
(com.huawei.bigdata.iotdb.Producer)
[2022-01-15 15:13:11,355] INFO The Producer have send 300 messages.
(com.huawei.bigdata.iotdb.Producer)
[2022-01-15 15:13:17,758] INFO The Producer have send 400 messages.
(com.huawei.bigdata.iotdb.Producer)
[2022-01-15 15:13:24,335] INFO The Producer have send 500 messages.
(com.huawei.bigdata.iotdb.Producer)
[2022-01-15 15:13:30,739] INFO The Producer have send 600 messages.
(com.huawei.bigdata.iotdb.Producer)
```



```
[2022-01-15 15:13:37,267] INFO The Producer have send 700 messages.
(com.huawei.bigdata.iotdb.Producer)
```

– KafkaConsumerMultThread.java

```
...
[2022-01-15 15:19:27,563] INFO Consumer Thread-1 partitions:1 record:
sensor_29,1642231023769,1.000000 offsets: 828
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-15 15:19:27,612] INFO Consumer Thread-1 partitions:1 record:
sensor_31,1642231023769,1.000000 offsets: 829
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-15 15:19:27,612] INFO Consumer Thread-0 partitions:0 record:
sensor_8,1642231023769,1.000000 offsets: 842
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-15 15:19:27,665] INFO Consumer Thread-1 partitions:1 record:
sensor_32,1642231023769,1.000000 offsets: 830
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-15 15:19:27,665] INFO Consumer Thread-0 partitions:0 record:
sensor_9,1642231023769,1.000000 offsets: 843
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-15 15:19:27,732] INFO Consumer Thread-1 partitions:1 record:
sensor_33,1642231023769,1.000000 offsets: 831
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-15 15:19:27,732] INFO Consumer Thread-0 partitions:0 record:
sensor_11,1642231023769,1.000000 offsets: 844
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-15 15:19:27,786] INFO Consumer Thread-0 partitions:0 record:
sensor_12,1642231023769,1.000000 offsets: 845
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-15 15:19:27,786] INFO Consumer Thread-1 partitions:1 record:
sensor_35,1642231023769,1.000000 offsets: 832
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
```

20.4.2 Commissioning JDBC and Session Applications on Linux

20.4.2.1 Compiling and Running Applications

Scenario

IoTDB applications can run in a Linux environment where an IoTDB client is installed. After the application code is developed, you can upload the JAR file to the prepared Linux environment. A Session application is used as an example. Operations for JDBC applications are the same as those for Session applications.

Prerequisites

- You have installed the IoTDB client.
- If the host where the client is installed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts** file on the node where the client is located. The host names and IP addresses must be mapped one by one.

Procedure

Step 1 Export a JAR file.

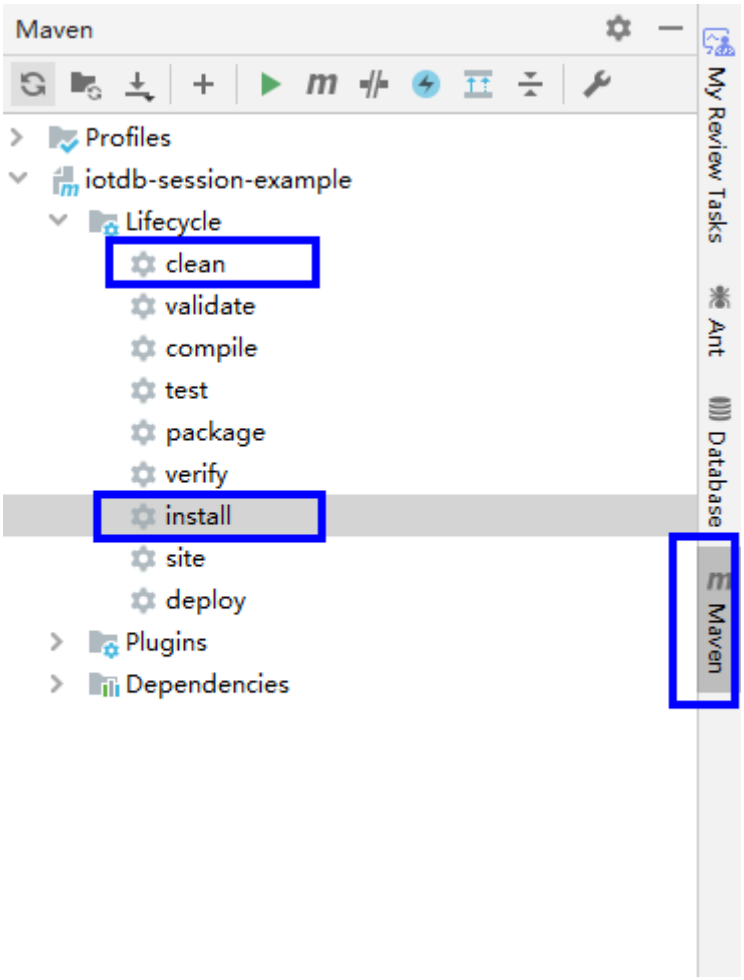
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

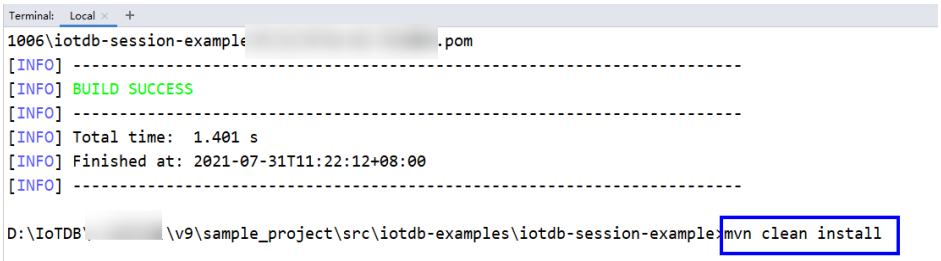
Choose **Maven** > *Sample project name* > **Lifecycle** > **install** and double-click **install** to run the **install** command of Maven.

Figure 20-18 Maven 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 file.

Figure 20-19 Entering mvn clean install in the IDEA Terminal text box



After the build is complete, message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

Step 2 Prepare the dependent JAR file.

1. Log in to the IoTDB client and import the JAR file generated in **Step 1** to the **lib** directory of the IoTDB client, for example, **/opt/client/IoTDB/iotdb/lib**.
2. In the root directory of the IoTDB client, for example, **/opt/client/IoTDB/iotdb**, create the **run.sh** script, modify the content as follows, and save the modification.

```
#!/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.iotdb.JDBCExample
```

   com.huawei.bigdata.iotdb.JDBCExample is an example. Use the actual code instead.

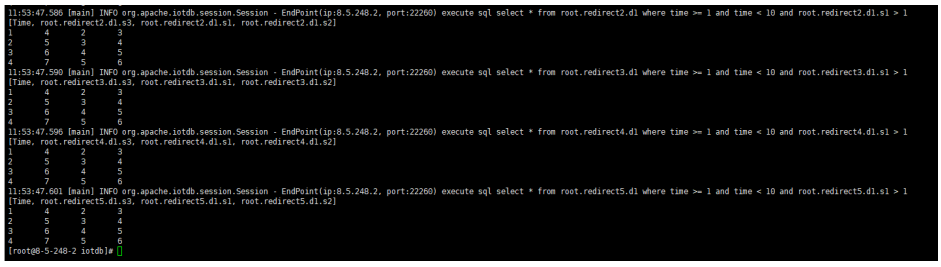
3. Run the **run.sh** script to run the JAR file.

   sh /opt/client/IoTDB/iotdb/run.sh

----End

20.4.2.2 Viewing Commissioning Results

If the application running is successful, the following information is displayed.



20.4.3 Commissioning Flink Applications on Flink Web UI and Linux

20.4.3.1 Compiling and Running Applications

Scenario

IoTDB applications can run in a Linux environment where the Flink client is installed and in an environment where the Flink web UI is installed. After the application code is developed, you can upload the JAR file to the prepared environment.

Prerequisites

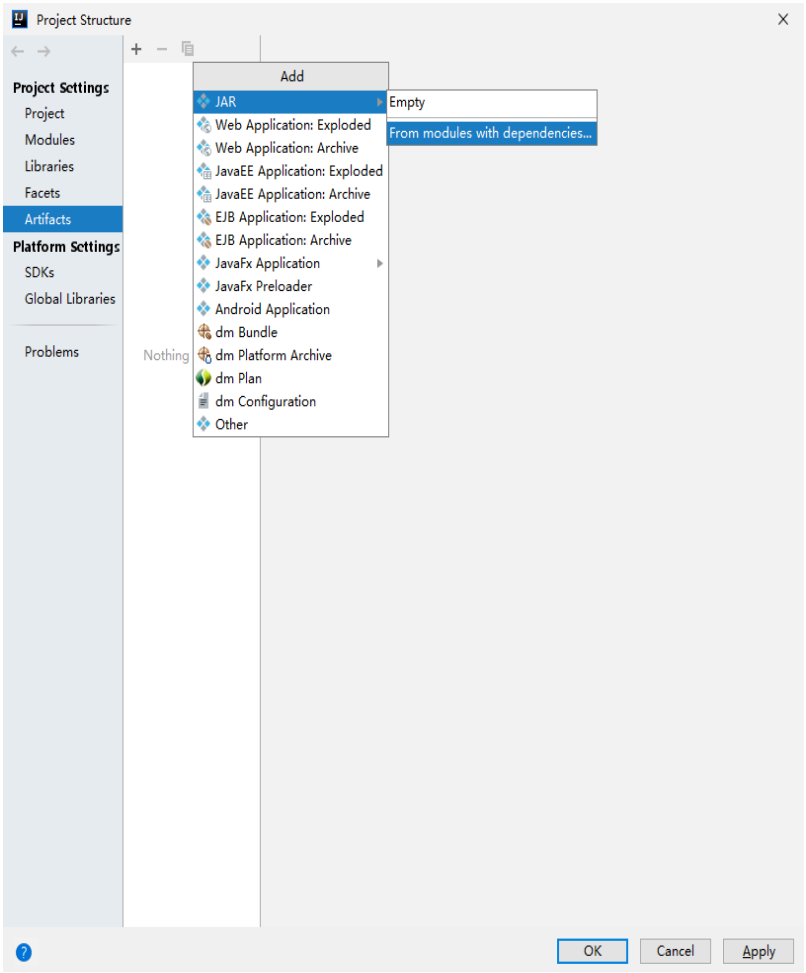
- The Flink component has been installed in the cluster and the FlinkServer instance has been added.
- The cluster client that contains the Flink service has been installed, for example, in the **/opt/client** directory.
- If the host where the client is installed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts** file on the node where the client is located. The host names and IP addresses must be mapped one by one.

Procedure

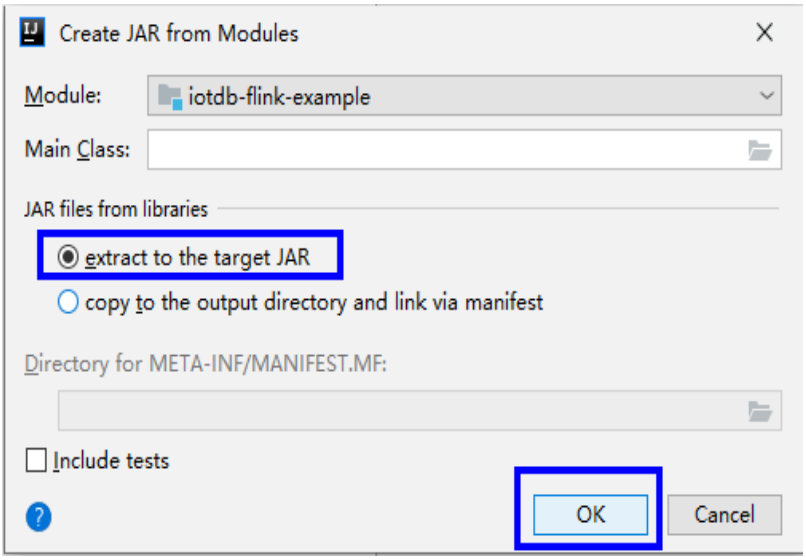
Step 1 Build a JAR file.

- In IntelliJ IDEA, configure **Artifacts** of the project before generating a JAR file.
  a. On the IDEA homepage, choose **File** > **Project Structures...** to go to the **Project Structure** page.
  b. On the **Project Structure** page, select **Artifacts**, click **+**, and select **From modules with dependencies...**.

Figure 20-20 Adding Artifacts



c. Select **extract to the target JAR** and click **OK**.

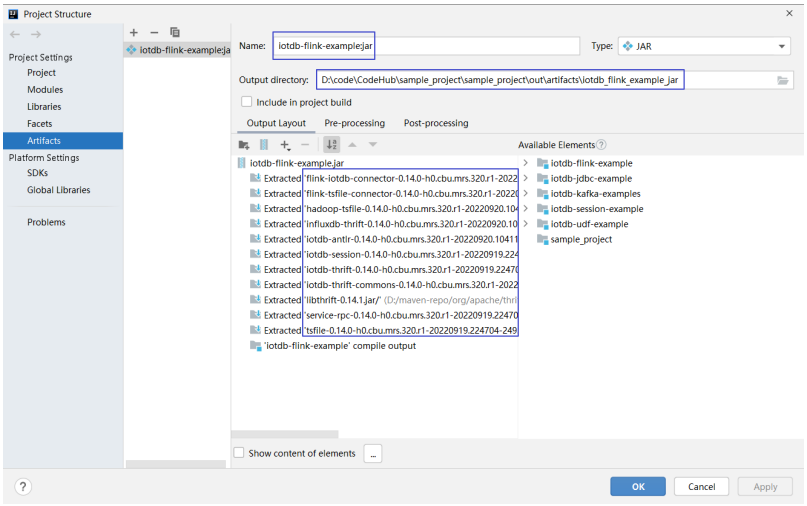


d. Set the name, type, and output path of the JAR file based on the site requirements.

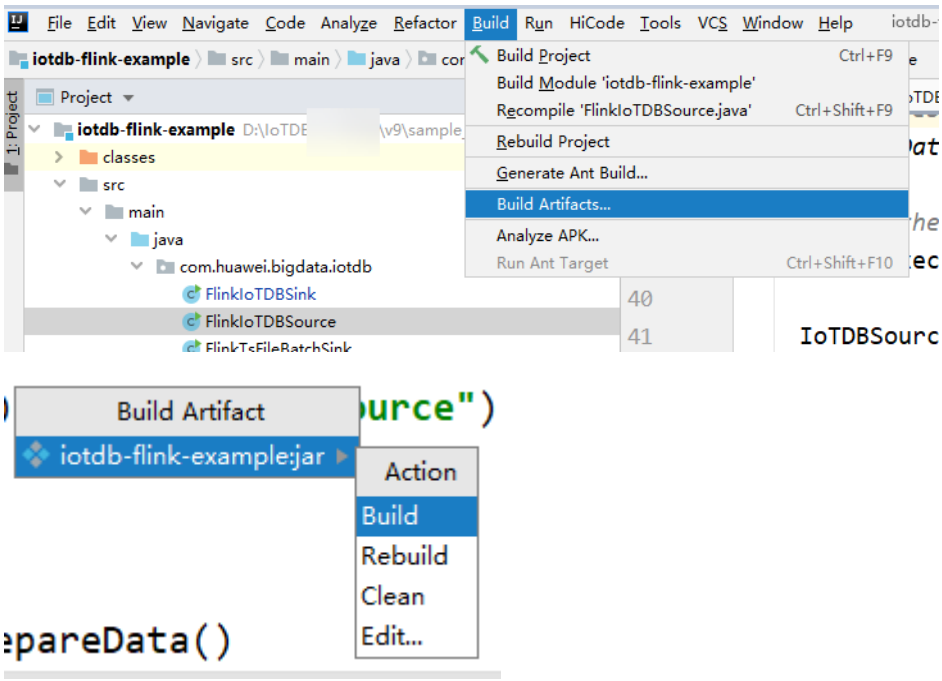
To avoid JAR file conflicts caused by unnecessary JAR files, you only need to load the following basic JAR files related to IoTDB:

- flink-iotdb-connector-*
- flink-tsfile-connector-*
- iotdb-session-*
- iotdb-thrift-*
- service-rpc-*
- tsfile-*

Click **OK**.



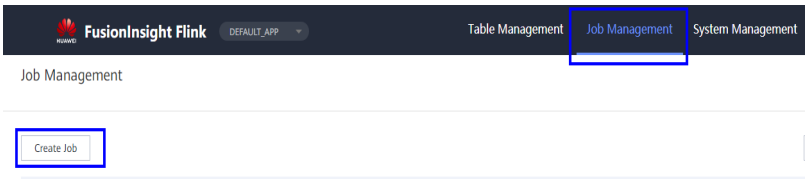
e. On the IDEA home page, choose **Build > Build Artifacts...**. On the **Build Artifact** page that is displayed, choose **Action** > **Build**.



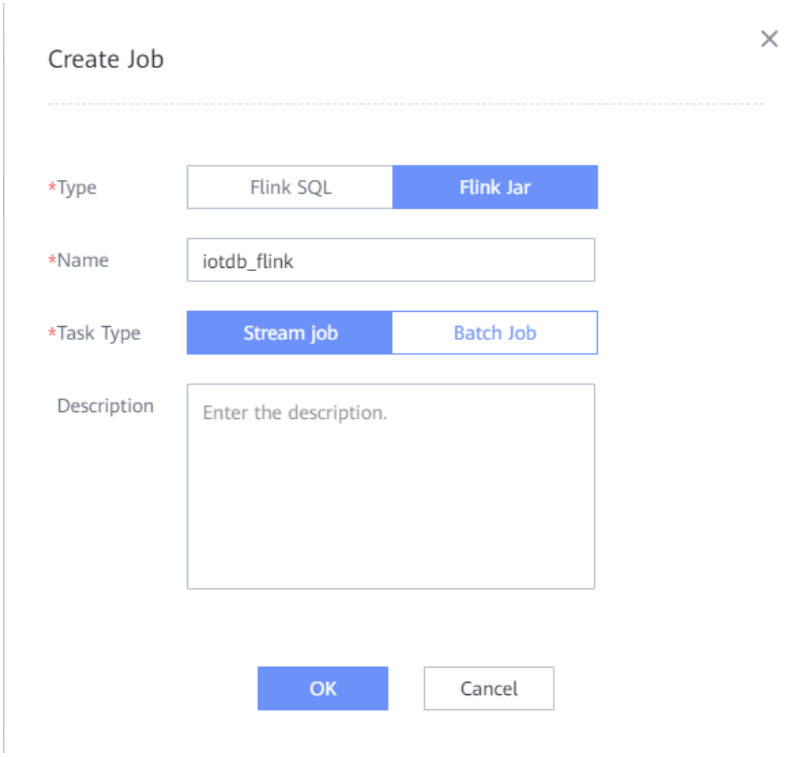
f. After the build is successful, the message "Build completed successfully" is displayed in the lower right corner, and the corresponding JAR file is generated in the **Output Directory** directory.

Step 2 (Scenario 1) Run a Flink job on the Flink web UI.

1. Log in to FusionInsight Manager of the cluster as a user who has the FlinkServer web UI management permission, choose **Cluster** > **Services** > **Flink**, and click the hyperlink next to **Flink WebUI** on the dashboard page to go to the FlinkServer web UI.
2. On the FusionInsight Flink web UI, choose **Job Management** > **Create Job** to create a job.

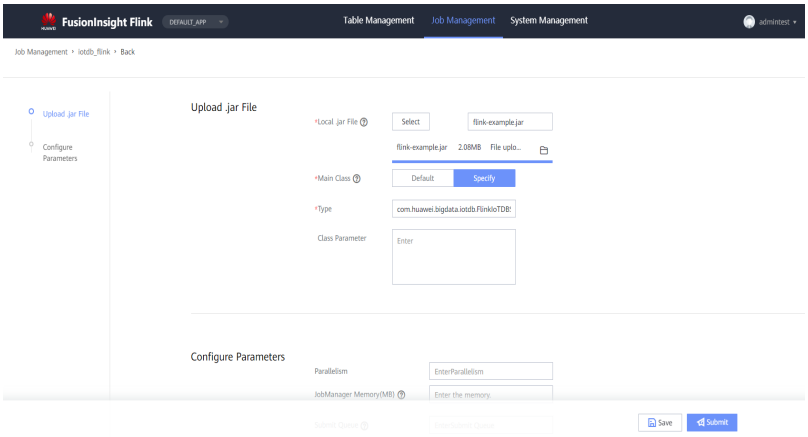


3. Set **Type** to **Flink Jar**, enter the name of the job to be created, select a task type, and click **OK**.



4. Upload the JAR file generated in **Step 1**, set **Main Class** to **Specify**, enter the class to be executed in **Class Parameter**, and click **Submit**.

   For example, set **Type** to **com.huawei.bigdata.iotdb.FlinkIoTDBSink** (development program that executes **FlinkIoTDBSink**) or **com.huawei.bigdata.iotdb.FlinkIoTDBSource** (development program that executes **FlinkIoTDBSource**).



Step 3 (Scenario 2) Submit a Flink job on the Flink client in the Linux environment.

1. Log in to the MRS client as a client installation user.
2. Run the following command to initialize environment variables:

   source /opt/client/bigdata_env
3. If Kerberos authentication is enabled for the cluster, perform **Step 3.4** to **Step 3.11**. If Kerberos authentication is disabled for the cluster, skip these steps.
4. Prepare a user for submitting Flink jobs.

   For details, see **Preparing the Developer Account**.

5. Log in to Manager using the created user, choose **System** > **Permission** > **User**. Locate the row that contains the new user and choose **More** > **Download Authentication Credential** in the **Operation** column.
6. Decompress the downloaded authentication credential package and copy the **user.keytab** file to the client node, for example, to the **/opt/client/Flink/flink/conf** directory on the client node.
7. In security mode, append 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 **/opt/client/Flink/flink/conf/flink-conf.yaml** file. Use commas (,) to separate IP addresses.
8. Run the following commands to configure security authentication by adding the **keytab** path and username to the **/opt/client/Flink/flink/conf/flink-conf.yaml** configuration file.
   security.kerberos.login.keytab: *<user.keytab file path>*
   security.kerberos.login.principal: *<Username>*

   Example:

   ```
   security.kerberos.login.keytab: /opt/client/Flink/flink/conf/user.keytab
   security.kerberos.login.principal: test
   ```
9. In the **bin** directory on the Flink client, run the following command to perform security hardening. For details, see **Authentication and Encryption**.

   sh generate_keystore.sh

   After the script is executed, enter a password for submitting jobs. Then, the value of SSL in **/opt/client/Flink/flink/conf/flink-conf.yaml** is automatically replaced.

   NOTE

   - In **Authentication and Encryption**, the generated **flink.keystore**, **flink.truststore**, and **security.cookie** are automatically filled in the corresponding configuration items in **flink-conf.yaml**.
   - The values of **security.ssl.key-password**, **security.ssl.keystore-password**, and **security.ssl.truststore-password** must be obtained using the Manager plaintext encryption API by running the following command:

     curl -k -i -u <*user name*>:<*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, *<password>* must be the same as the password used for issuing the certificate, and *x.x.x.x* indicates the floating IP address of Manager in the cluster.
10. Configure paths for the **flink.keystore** and **flink.truststore** files.
    - Absolute path: After the script is executed, the **flink.keystore** and **flink.truststore** file paths are automatically set to absolute paths in the **flink-conf.yaml** file. In this case, you need to place the **flink.keystore** and **flink.truststore** files in the **conf** directory to the absolute paths of the Flink client and each Yarn node in the cluster, respectively.
    - Relative path: Perform the following operations to set the **flink.keystore** and **flink.truststore** files to relative paths:
      i. In the **/opt/client/Flink/flink/conf/** directory, create a directory, for example, **ssl**.

         cd /opt/client/Flink/flink/conf

         mkdir ssl

ii. Move the **flink.keystore** and **flink.truststore** files to the new folder.

mv flink.keystore flink.truststore ssl/

iii. Change the values of the following parameters to relative paths in the **flink-conf.yaml** file:

```
security.ssl.keystore: ssl/flink.keystore
security.ssl.truststore: ssl/flink.truststore
```

11. Add the IP addresses of the nodes where the clients are located to the following configuration items in the **flink-conf.yaml** file. Use commas (,) to separate IP addresses.

```
web.access-control-allow-origin: xx.xx.xxx.xxx
jobmanager.web.allow-access-address: xx.xx.xxx.xxx
```

12. Upload the JAR file generated in **Step 1** to the Flink client node, for example, **/opt/client/Flink/flink**, and submit the job.

NOTICE

To submit or run jobs on Flink, the user must have the following permissions:

- If Ranger authentication is enabled, the current user must belong to the **hadoop** group or the user has been granted the **/flink** read and write permissions in Ranger.
- If Ranger authentication is disabled, the current user must belong to the **hadoop** group.

- If the **flink.keystore** and **flink.truststore** files are stored in the absolute path:

  Run the following commands to start a session and submit a job in the session: **com.huawei.bigdata.iotdb.FlinkIoTDBSink** is the application in **FlinkIoTDBSink**.

  yarn-session.sh -nm "*session-name*"

  flink run --class com.huawei.bigdata.iotdb.FlinkIoTDBSink /opt/client/Flink/flink/flink-example.jar

- If the **flink.keystore** and **flink.truststore** files are stored in the relative path:

  In the same directory of SSL, run the following commands to start a session and submit a job in the session. The SSL directory is a relative path. For example, if the SSL directory is **/opt/client/Flink/flink/conf/**, then run the following commands in this directory.

  com.huawei.bigdata.iotdb.FlinkIoTDBSink is the application in **FlinkIoTDBSink**.

  yarn-session.sh -t ssl/ -nm "*session-name***"**

  flink run --class com.huawei.bigdata.iotdb.FlinkIoTDBSink /opt/client/Flink/flink/flink-example.jar

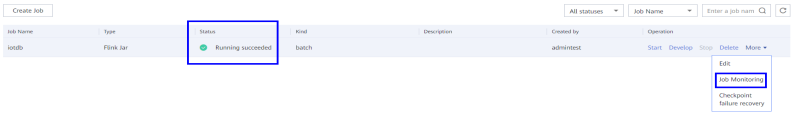
----End

20.4.3.2 Viewing Commissioning Results

1. Check whether the job is executed.

- Using the Flink web UI

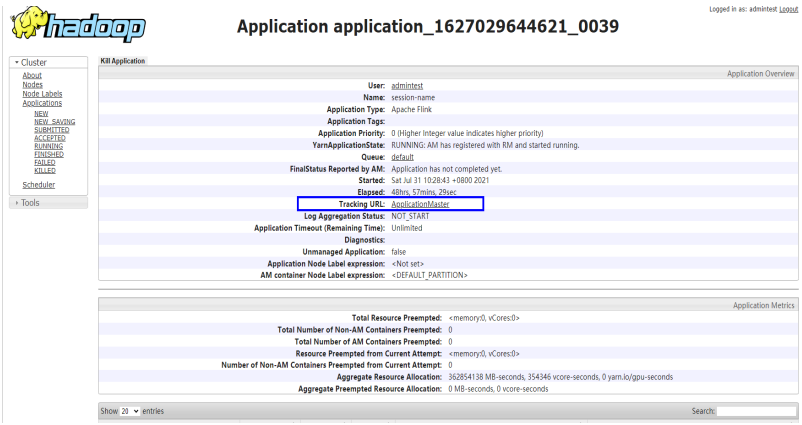
  If a success message is returned on the Flink web UI, the execution is successful. You can choose **More** > **Job Monitoring** in the **Operation** column to view detailed logs.

- Using the Flink client

  Log in to FusionInsight Manager as a running user, go to the native page of the Yarn service, find the application of the corresponding job, and click the application name to go to the job details page.

  - If the job is not complete, click **Tracking URL** to go to the native Flink service page and view the job running information.
  - If the job submitted in a session has been completed, you can click **Tracking URL** to log in to the native Flink service page to view job information.

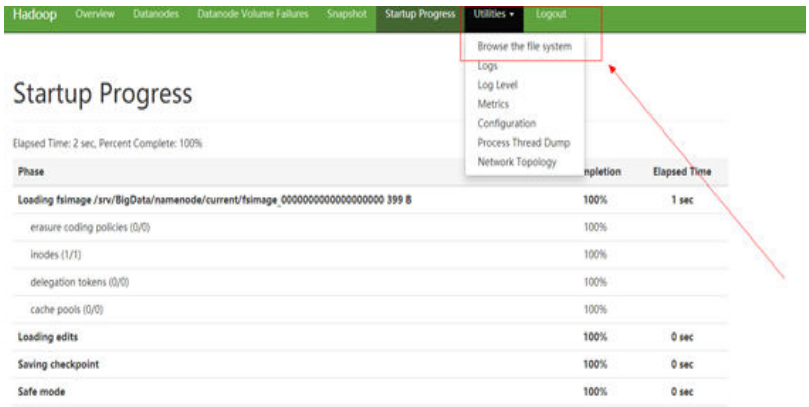
    

2. Verify the job execution result.
   - FlinkIoTDBSink execution result:
     - Run the following command on the IoTDB client and check whether the data has been written from Flink to IoTDB:

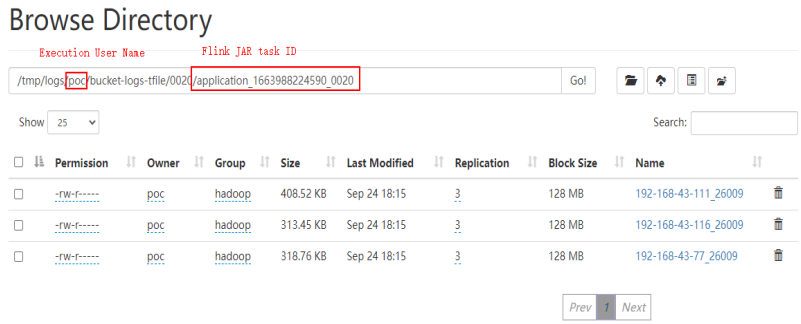
       select * from root.sg.d1

```
IoTDB@[redacted]:22260> select * from root.sg.d1
+----------------------------+---------------------+
|                        Time|        root.sg.d1.s1|
+----------------------------+---------------------+
|2022-09-24T16:19:13.076+08:00|  0.48050401096976925|
|2022-09-24T16:19:15.404+08:00|   0.5917976517293774|
|2022-09-24T16:19:16.404+08:00|    0.735936612423717|
|2022-09-24T16:19:23.869+08:00|   0.8386843031417548|
|2022-09-24T16:19:24.869+08:00|   0.6463236193893684|
|2022-09-24T16:19:25.870+08:00|   0.5954449837992902|
|2022-09-24T16:19:26.870+08:00|   0.5945272607886377|
|2022-09-24T16:19:27.871+08:00|   0.4929779102387244|
|2022-09-24T16:19:28.871+08:00|  0.34395251562673457|
|2022-09-24T16:19:29.872+08:00|   0.6177739062963853|
|2022-09-24T16:19:30.872+08:00|   0.7498794851618589|
|2022-09-24T16:19:31.873+08:00|   0.3148741420873038|
|2022-09-24T16:19:32.873+08:00|   0.6684900825420255|
|2022-09-24T16:19:33.874+08:00|  0.45938194501820595|
|2022-09-24T16:19:34.874+08:00|  0.24293321093096187|
|2022-09-24T16:19:35.875+08:00|   0.7094903559027376|
|2022-09-24T16:19:36.875+08:00|    0.934997431381603|
|2022-09-24T16:19:37.876+08:00|   0.9151513474234568|
|2022-09-24T16:19:38.876+08:00|   0.5625493998872735|
|2022-09-24T16:19:39.876+08:00|    0.991484539263849|
|2022-09-24T16:19:40.877+08:00|   0.8925889658869346|
|2022-09-24T16:19:41.878+08:00|   0.8873826211498665|
|2022-09-24T16:19:42.878+08:00|  0.44325532800273826|
|2022-09-24T16:19:43.879+08:00|  0.48251227562595245|
|2022-09-24T16:19:44.879+08:00|   0.1367102977099991|
|2022-09-24T16:19:45.880+08:00|  0.06494944420823234|
|2022-09-24T16:19:46.880+08:00|   0.4026376592496197|
|2022-09-24T16:19:47.881+08:00|   0.6740267414029771|
|2022-09-24T16:19:48.881+08:00|   0.7480089974928386|
```

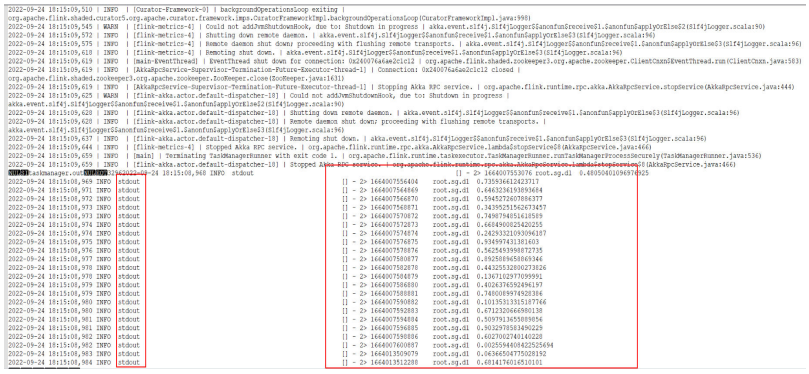
- FlinkIoTDBSource execution result:
  1) Log in to FusionInsight Manager as a running user and choose **Cluster** > **Services** > **HDFS**. Click the hyperlink next to **NameNode WebUI** to access the HDFS web UI.
  2) Choose **Utilities** > **Browse the file system**.



  3) Go to the **/tmp/logs/***Execution username*/**bucket-logs-tfile/***Task ID*/*Flink task ID* directory and download all files in the directory to the local PC.



4) Search for **root.sg.d1** in the files downloaded in **2.3)**. If the following information is displayed, data is successfully read from IoTDB.



20.4.4 Commissioning Kafka Applications on Linux

20.4.4.1 Compiling and Running Applications

Scenario

After code development is complete, you can run an IoTDB-Kafka sample application in the Linux environment.

Prerequisites

- You have installed the IoTDB and Kafka clients.
- If the host where the client is installed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts** file on the node where the client is located. The host names and IP addresses must be mapped one by one.

Procedure

Step 1 Export a JAR file.

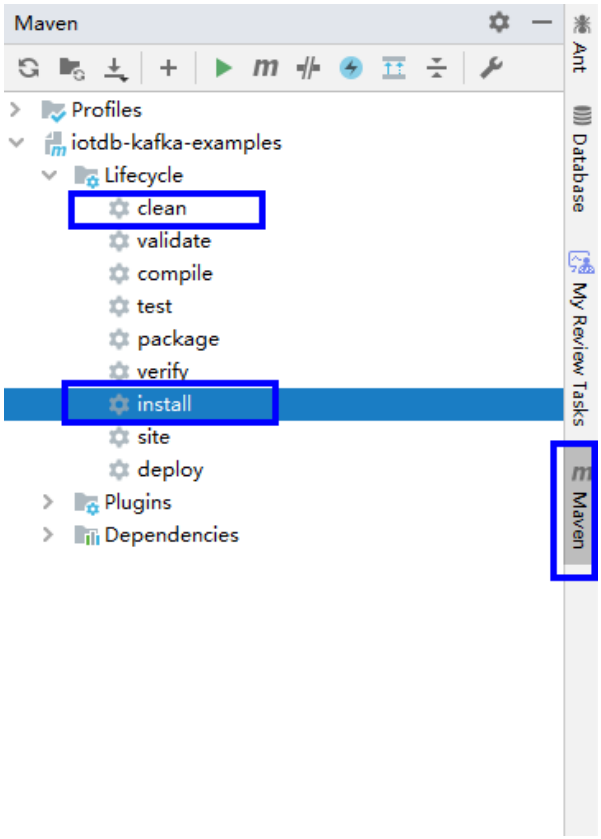
You can build a JAR file in either of the following ways:

- Method 1:

  Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

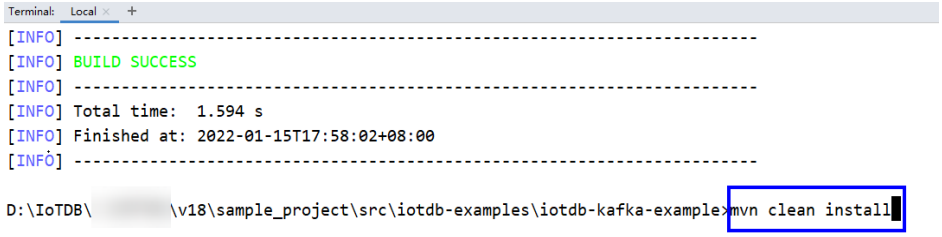
Choose **Maven** > *Sample project name* > **Lifecycle** > **install** and double-click **install** to run the **install** command of Maven.

Figure 20-21 Maven clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

  Figure 20-22 Entering **mvn clean install** in the IDEA Terminal text box

  After the build is complete, message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

Step 2 Prepare the dependent JAR file.

1. Go to the client installation directory, create the **lib** directory, and import the JAR package generated in **Step 1** to the **lib** directory, for example, **/opt/client/lib**.

2. Go to the Kafka client and copy the JAR package on which Kafka depends to the **lib** directory in **Step 2.1**. The following is an example directory:

   cp /opt/client/Kafka/kafka/libs/*.jar /opt/client/lib

3. Go to the IoTDB client and copy the JAR package on which IoTDB depends to the **lib** directory in **Step 2.1**. The following is an example directory:

   cp /opt/client/IoTDB/iotdb/lib/*.jar/opt/client/lib

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 3 Go to the **/opt/client** directory and ensure that the current user has the read permission on all files in the **src/main/resources** directory and the dependency library folder. In addition, ensure that JDK has been installed and Java environment variables have been set. Then, run the following command to run the sample project:

java -cp /opt/client/lib/*:/opt/client/src/main/resources com.huawei.bigdata.iotdb.KafkaConsumerMultThread

----End

20.4.4.2 Viewing Commissioning Results

If the application running is successful, the following information is displayed.

```
[2022-01-17 14:43:57,511] INFO Consumer Thread-1 partitions:1 record:sensor_89,1642401919971,1.000000
  offsets:6951  (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-17 14:43:57,511] INFO Consumer Thread-0 partitions:0 record:sensor_97,1642401919971,1.000000
  offsets:7129  (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-17 14:43:57,512] INFO Consumer Thread-0 partitions:0 record:sensor_98,1642401919971,1.000000
  offsets:7130  (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-17 14:43:57,512] INFO Consumer Thread-1 partitions:1 record:sensor_92,1642401919971,1.000000
  offsets:6952  (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-17 14:43:57,513] INFO Consumer Thread-0 partitions:0 record:sensor_99,1642401919971,1.000000
  offsets:7131  (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-17 14:43:57,513] INFO Consumer Thread-1 partitions:1 record:sensor_93,1642401919971,1.000000
  offsets:6953  (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
[2022-01-17 14:43:57,513] INFO Consumer Thread-1 partitions:1 record:sensor_95,1642401919971,1.000000
  offsets:6954  (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
```

20.4.5 Using a UDF

20.4.5.1 Registering a UDF

1. Build a JAR file.

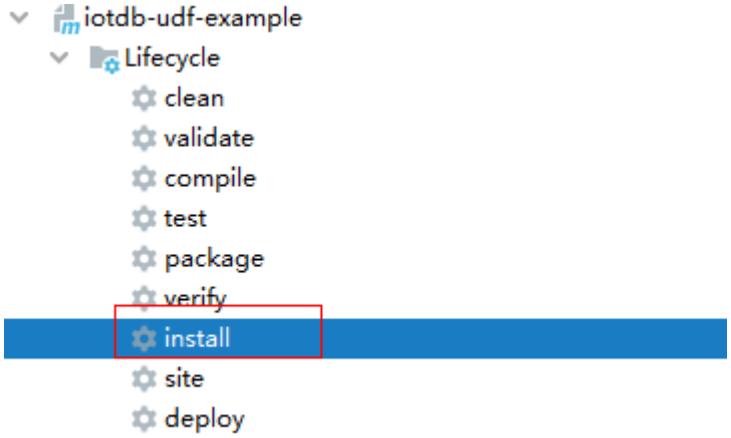
   You can build a JAR file in either of the following ways:

   - Method 1:

     Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.

     Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

Figure 20-23 Maven clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

  Figure 20-24 Building result after mvn clean install is entered

  After the build is complete, message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

2. Import the dependent JAR file.

   Log in to the node where IoTDBServer is located as user **root**, run **su - omm** to switch to user **omm**, and import the JAR file generated in **1** to the **$BIGDATA_HOME/FusionInsight_IoTDB_*/install/FusionInsight-IoTDB-*/iotdb/ext/udf** directory.

   NOTICE

   - During cluster deployment, ensure that a corresponding JAR package exists in the UDF JAR package path of each IoTDBServer node. You can modify IoTDB configuration **udf_root_dir** to specify the root path for the UDF to load JAR files.
   - To view the IP address of the node where IoTDBServer is deployed, log in to the MRS cluster management console, choose **Components**, click **IoTDB**, and click the **Instances** tab.

3. Run the following SQL statement to register the UDF:

   CREATE FUNCTION <UDF-NAME> **AS** '<UDF-CLASS-FULL-PATHNAME>'

   For example, you can run the following statement to register a UDF named **example**:

   ```
   CREATE FUNCTION example AS 'com.huawei.bigdata.iotdb.UDTFExample'
   ```


20.4.5.2 Querying a UDF

Basic SQL Syntax Supported

- SLIMIT / SOFFSET
- LIMIT / OFFSET
- NON ALIGN
- Queries with value filters
- Queries with time filters

NOTE

Currently, aligned time series are not supported in UDF queries. An error message is reported if you use UDF queries with aligned time series selected.

Queries with an Asterisk (*)

Assume that there are two time series (**root.sg.d1.s1** and **root.sg.d1.s2**).

- Execute the **SELECT example(*) from root.sg.d1** statement.

  The result set contains the results of **example(root.sg.d1.s1)** and **example(root.sg.d1.s2)**.
- Execute the **SELECT example(s1, *) from root.sg.d1** statement.

  The result set contains the results of **example(root.sg.d1.s1, root.sg.d1.s1)** and **example(root.sg.d1.s1, root.sg.d1.s2)**.
- Execute the **SELECT example(*, *) from root.sg.d1** statement.

  The result set contains the results of **example(root.sg.d1.s1, root.sg.d1.s1)**, **example(root.sg.d1.s2, root.sg.d1.s1)**, **example(root.sg.d1.s1, root.sg.d1.s2)**, and **example(root.sg.d1.s2, root.sg.d1.s2)**.

Queries with key-value pair attributes in UDF parameters

You can pass any number of key-value pair parameters to the UDF when constructing a UDF query. The key and value in a key-value pair must be enclosed in single or double quotation marks.

NOTICE

The key-value pair parameters can be passed in only after the time series have been passed in.

For example:

```
SELECT example(s1, 'key1'='value1', 'key2'='value2'), example(*, 'key3'='value3') FROM root.sg.d1;
SELECT example(s1, s2, 'key1'='value1', 'key2'='value2') FROM root.sg.d1;
```

Showing All Registered UDFs

Run the following SQL statement on the IoTDB client to view the registered UDFs:

```
SHOW FUNCTIONS
```


20.4.5.3 Deregistering a UDF

Syntax

DROP FUNCTION <UDF-NAME>

Example

Run the following command on the IoTDB client to deregister the UDF named **example**:

```
DROP FUNCTION example
```

20.5 More Information

20.5.1 Common APIs

20.5.1.1 Java API

IoTDB provides a connection pool (SessionPool) for native APIs. When using the APIs, you only need to specify the pool size to obtain connections from the pool. If you cannot get a connection in 60 seconds, a warning log will be printed, but the program continues to wait.

When a connection is used, it automatically returns to the pool and waits to be used next time. When a connection is damaged, it is deleted from the pool and a new connection is created to perform user operations again.

For query operations:

1. When SessionPool is used for query, the result set is SessionDataSetWrapper, the encapsulation class of SessionDataSet.
2. If a query result set is not traversed and you do not want to continue traversing, you need to call **closeResultSet** to release the connection.
3. If an exception is reported when you traverse a query result set, you need to call **closeResultSet** to release the connection.
4. You can call the **getColumnNames()** method of SessionDataSetWrapper to obtain the column names in the result set.

Table 20-4 Sessions APIs and corresponding parameters

Method	Description
• Session(String host, int rpcPort) • Session(String host, String rpcPort, String username, String password) • Session(String host, int rpcPort, String username, String password)	Initializes a session.
Session.open()	Opens a session.

Method	Description
Session.close()	Closes a session.
void setStorageGroup(String storageGroupId)	Sets a storage group.
• void deleteStorageGroup(String storageGroup) • void deleteStorageGroups(List<String> storageGroups)	Deletes one or more storage groups.
• void createTimeseries(String path, TSDataType dataType, TSEncoding encoding, CompressionType compressor, Map<String, String> props, Map<String, String> tags, Map<String, String> attributes, String measurementAlias) • void createMultiTimeseries(List<String> paths, List<TSDataType> dataTypes, List<TSEncoding> encodings, List<CompressionType> compressors, List<Map<String, String>> propsList, List<Map<String, String>> tagsList, List<Map<String, String>> attributesList, List<String> measurementAliasList)	Creates one or more time series.
• void deleteTimeseries(String path) • void deleteTimeseries(List<String> paths)	Deletes one or more time series.
• void deleteData(String path, long time) • void deleteData(List<String> paths, long time)	Deletes the data of one or more time series before or at a specific time point.
void insertRecord(String deviceId, long time, List<String> measurements, List<String> values)	Inserts a record, which contains the data of multiple measurement points of a device at a timestamp. Servers need to perform type inference, which may take extra time.
void insertTablet(Tablet tablet)	Inserts a tablet, which contains multiple rows of non-empty data blocks. The columns in each row are the same.
void insertTablets(Map<String, Tablet> tablet)	Inserts multiple tablets.

Method	Description
void insertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList, List<List<String>> valuesList)	Inserts multiple records. Servers need to perform type inference, which may take extra time.
void insertRecord(String deviceId, long time, List<String> measurements, List<TSDataType> types, List<Object> values)	Inserts a record, which contains the data of multiple measurement points of a device at a timestamp. With the data type information, servers do not need to perform type inference, improving the performance.
void insertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList, List<List<TSDataType>> typesList, List<List<Object>> valuesList)	Inserts multiple records. With the data type information, servers do not need to perform type inference, improving the performance.
submitApplication(SubmitApplication-Request request)	Used by the client to submit a new application to ResourceManager.
void insertRecordsOfOneDevice(String deviceId, List<Long> times, List<List<String>> measurementsList, List<List<TSDataType>> typesList, List<List<Object>> valuesList)	Inserts multiple records of the same device.
SessionDataSet executeRawDataQuery(List<String> paths, long startTime, long endTime)	Queries raw data. The interval includes the start time but does not include the end time.
SessionDataSet executeQueryState-ment(String sql)	Executes query statements.
void executeNonQueryState-ment(String sql)	Executes non-query statements.

Table 20-5 Test APIs

Method	Description
• void testInsertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList, List<List<String>> valuesList) • void testInsertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList, List<List<TSDataType>> typesList, List<List<Object>> valuesList)	Tests testInsertRecords. A response is returned immediately after data is transmitted to the server. No data is written.
• void testInsertRecord(String deviceId, long time, List<String> measurements, List<String> values) • void testInsertRecord(String deviceId, long time, List<String> measurements, List<TSDataType> types, List<Object> values)	Tests insertRecord. A response is returned immediately after data is transmitted to the server. No data is written.
void testInsertTablet(Tablet tablet)	Tests insertTablet. A response is returned immediately after data is transmitted to the server. No data is written.

21 IoTDB Development Guide (Normal Mode)

21.1 Overview

21.1.1 Application Development Overview

Introduction to IoTDB

IoTDB is a data management engine that integrates collection, storage, and analysis of time series data. It features lightweight, high performance, and ease of use. It perfectly interconnects with the Hadoop and Spark ecosystems and meets the requirements of high-speed write and complex analysis and query on massive time series data in industrial IoT applications.

NOTE

This document applies only to MRS 3.2.0 or later.

21.1.2 Basic Concepts

The following uses an electric power scenario as an example to describe how to create a correct data model in IoTDB.

Figure 21-1 Hierarchical structure of attributes in an electric power scenario

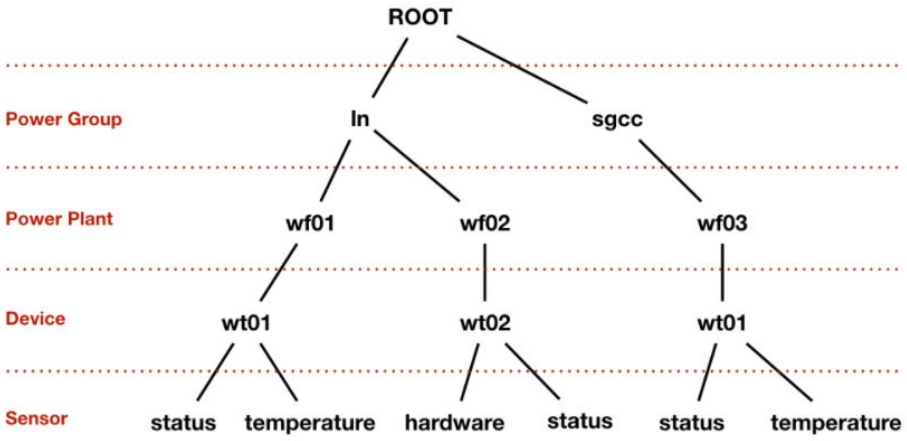


Figure 21-1 shows the power group layer, power plant layer, device layer, and sensor layer. **ROOT** is a root node, and each node at the sensor layer is a leaf node. According to the IoTDB syntax, the path from **ROOT** to a leaf node is separated by a dot (.). The complete path is used to name a time series in the IoTDB. For example, the time series name corresponding to the path on the left in the figure is **ROOT.ln.wf01.wt01.status**.

Basic Concepts

- **Device**

  A device is a machine equipped with sensors in actual scenarios. In IoTDB, all sensors must have their corresponding devices.

- **Sensor**

  A sensor is a detection machine in actual scenarios. It can sense the information to be measured, and can transform the sensed information into an electrical signal or other desired form of information output and send it to IoTDB. In IoTDB, all data and paths stored are organized in units of sensors.

- **Storage group**

  You can set any prefix path as a storage group. If there are four time series, for example, **root.vehicle.d1.s1**, **root.vehicle.d1.s2**, **root.vehicle.d2.s1**, and **root.vehicle.d2.s2**, two devices **d1** and **d2** in the path **root.vehicle** may belong to the same owner or manufacturer, so **d1** and **d2** are closely related. In this case, the prefix path **root.vehicle** can be designated as a storage group, which will enable IoTDB to store the data of all devices under it in the same folder. Newly added devices in **root.vehicle** will also belong to this storage group.

  NOTE

  - A proper number of storage groups can improve performance. There is neither the slowdown of the system due to frequent I/O switching (which will also take up excessive memory and result in frequent memory-file switching) caused by too many storage files or folders, nor the block of write commands caused by too few storage files or folders (which reduces concurrency).
  - You need to balance the storage group settings of storage files according to their own data size and usage scenarios to achieve better system performance.

- **Time series**

  Time series is a core concept in IoTDB. The time series can be considered as the complete path of the sensor that generates the time series data. In IoTDB, all time series must start with root and end with the sensor.

21.1.3 Development Process

This section describes how to use Java APIs to develop IoTDB applications.

Figure 21-2 and Table 21-1 describe the phases in the development process.

Figure 21-2 IoTDB application development process

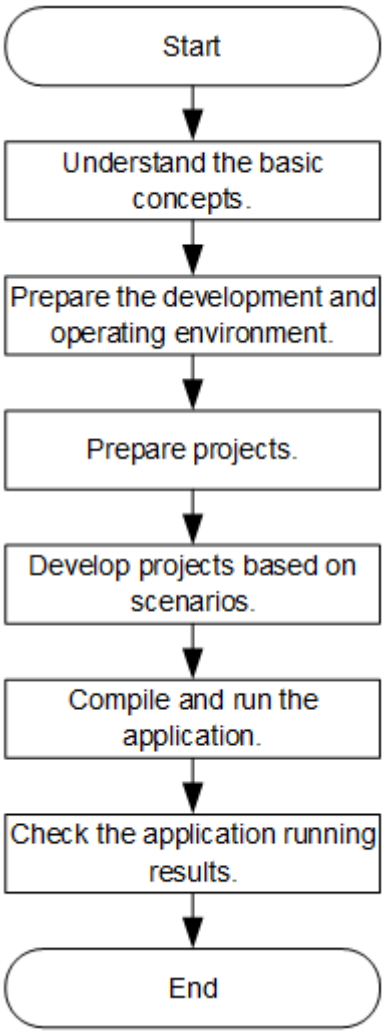


Table 21-1 Description of the IoTDB application development process

Phase	Description	Reference
Understand the basic concepts.	Before developing an application, learn basic concepts of IoTDB and understand the scenario requirements and design tables.	Basic Concepts

Phase	Description	Reference
Prepare the development and operating environment.	The Java language is recommended for IoTDB application development. You can use the IntelliJ IDEA tool.	**Preparing the Development and Running Environment**
Prepare projects.	IoTDB provides sample projects for different scenarios. You can import a sample project to learn the application. You can also create an IoTDB project according to the guide.	**Configuring and Importing a Sample Project**
Develop projects based on scenarios.	Sample projects of the Java language are provided, including JDBC and Session connection modes. A sample project covers the entire process from creating a storage group, creating a time sequence, inserting data, to deleting a storage group.	**Application Development**
Compile and run the application.	Compile the developed application and submit it for running.	**Application Commissioning**
View application running results.	View the application running status on IntelliJ IDEA.	**Application Commissioning**

21.1.4 IoTDB Sample Project

To obtain an MRS sample project, visit **https://github.com/huaweicloud/huaweicloud-mrs-example** and switch to the branch that matches the MRS cluster version. Download the package to the local PC and decompress it to obtain the sample project of each component.

MRS provides the following IoTDB sample projects.

Table 21-2 IoTDB sample projects

Sample Project Location	Description
iotdb-examples/iotdb-flink-example	Program for using Flink to access IoTDB data, including FlinkIoTDBSink and FlinkIoTDBSource data. FlinkIoTDBSink can use Flink jobs to write time series data to IoTDB. FlinkIoTDBSource reads time series data from IoTDB through Flink jobs and prints the data. For details, see **IoTDB Flink**.

Sample Project Location	Description
iotdb-examples/iotdb-jdbc-example	Java sample program for IoTDB JDBC to process data This example demonstrates how to use JDBC APIs to connect to IoTDB and run IoTDB SQL statements. For details, see **IoTDB JDBC**.
iotdb-examples/iotdb-kafka-example	Sample program for accessing IoTDB data through Kafka This program demonstrates how to send time series data to Kafka and then use multiple threads to write the data to IoTDB. For details, see **IoTDB Kafka**.
iotdb-examples/iotdb-session-example	Java sample program for IoTDB Session to process data This example demonstrates how to use Session to connect to IoTDB and run IoTDB SQL statements. For details, see **IoTDB Session**.
iotdb-examples/iotdb-udf-exmaple	This sample program describes how to implement a simple IoTDB user-defined function (UDF). For details, see **IoTDB UDF Program**.

21.2 Environment Preparations

21.2.1 Preparing the Development and Running Environment

Table 21-3 describes the environment required for application development.

Table 21-3 Development environment

Item	Description
OS	• Development environment: Windows 7 or later versions • Operating environment: Windows or Linux If the program needs to be commissioned locally, the operating environment must be able to communicate with network on the cluster service plane.

Item	Description
JDK	Basic configurations of the development and operating environments. The version requirements are as follows: The server and client support only built-in OpenJDK 1.8.0_272. Customers' applications that need to reference the JAR files of SDK to run in the application processes support Oracle JDK, IBM JDK, and OpenJDK. x86 client: Oracle JDK 1.8; IBM JDK: 1.8.5.11. **NOTE** For security purposes, the server supports only TLS V1.2 or later. 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-sslcontextgetinstancetls-oracle#matchsslcontext_tls**.
IntelliJ IDEA	Tool used for developing IoTDB applications. The version must be 2019.1 or other compatible version. **NOTE** • 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 OpenJDK, ensure that the JDK configured in IntelliJ IDEA is OpenJDK. • Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.
Maven	Basic configurations of the development environment. Maven is used 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 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. 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**. Then, decompress **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar** Decompress the package to the **D:\FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles** directory on the local PC.

b. 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 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. **Log in to FusionInsight Manager**. 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 directory of the cluster client, and copy all configuration files in the **FusionInsight_Cluster_1_Services_ClientConfig/IoTDB/config** directory to the client node to the **conf** directory where the compiled JAR package is stored, for example, **/opt/client/conf**, for subsequent commissioning.

  For example, if the client software package is **FusionInsight_Cluster_1_Services_Client.tar** and it is downloaded to the **/tmp/FusionInsight-Client** directory on the active management node, run the following commands:

  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 IoTDB/config/* root@*IP address of the client node***:/opt/client/conf**

The keytab file obtained in **Preparing the 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 is located, to ensure that the local host can communicate with each host in the cluster.

21.2.2 Preparing the Configuration Files for Connecting to the Cluster

Preparing the Configuration Files of the Running Environment

During the development or a test run of the program, you need to use cluster configuration files to connect to an MRS cluster. The configuration files usually contain the cluster component information file and user files used for security authentication. You can obtain the required information from the created MRS cluster.

Nodes used for program debugging or running must be able to communicate with nodes within the MRS cluster, and the hosts domain name must be configured.

- Scenario 1: Preparing the configuration files required for debugging in the local Windows development environment

  a. Download and decompress the client software package.

     - Versions earlier than MRS 3.3.0, log in to FusionInsight Manager and choose **Cluster** > **Dashboard** > **More** > **Download Client**. Set **Select Client Type** to **Configuration Files Only**, and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.
     - MRS 3.3.0 or later, log in to FusionInsight Manager, and click **Download Client** in the upper right corner of **Homepage**. Set **Select Client Type** to **Configuration Files Only**, and click **OK**. After the client file package is generated, download it 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**. Then, decompress **FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar** Decompress the package to the **D:\FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles** directory on the local PC.

  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 is normal.

NOTE

- If the client host is outside the cluster, configure network connections to the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored, for example, in **C:\WINDOWS\system32\drivers\etc\hosts**.

- Scenario 2: Preparing the configuration files required for running the program in a Linux environment

  a. Install the client. For example, install the client in the **/opt/client** directory.

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

  b. Download the client configuration file to the active OMS node of the cluster.

     - Versions earlier than MRS 3.3.0, log in to FusionInsight Manager and choose **Cluster** > **Dashboard** > **More** > **Download Client**. Set **Select Client Type** to **Configuration Files Only**, select **Save to Path**, and click **OK** to download the client configuration file to the active OMS node of the cluster.
     - MRS 3.3.0 or later, log in to FusionInsight Manager, and click **Download Client** in the upper right corner of **Homepage**. Set **Select Client Type** to **Configuration Files Only**, select **Save to Path**, and click **OK** to download the client configuration file to the active OMS node of the cluster.

  c. Log in to the active OMS node as user **root**, go to the directory where the client configuration files are stored (**/tmp/FusionInsight-Client** by default), decompress the software package, and obtain all configuration files in the **IoTDB/config** directory. Place the files in the **conf** directory where the compiled JAR package is stored on the client node, for example, **/opt/client/conf**.

     For example, if the client software package is **FusionInsight_Cluster_1_Services_Client.tar** and it is downloaded to the **/tmp/FusionInsight-Client** directory on the active management node, run the following commands:

     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 IoTDB/config/* root@*IP address of the client node***:/opt/client/conf**

  d. 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 there is no required information, copy the content of the **hosts** file in the decompression directory to the **hosts** file on the node where the client is deployed, to ensure that the local host can communicate with each host in the cluster.

21.2.3 Configuring and Importing a Sample Project

Context

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

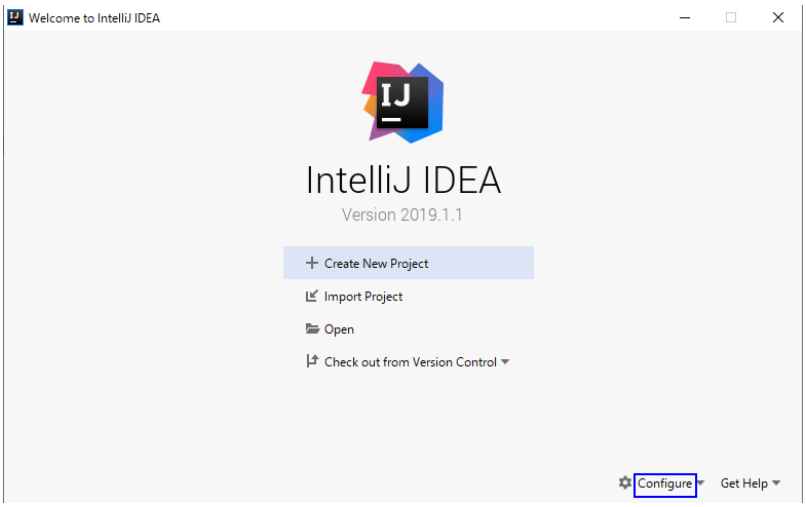
Procedure

Step 1 Obtain the sample project folder **iotdb-jdbc-example** in the **src/iotdb-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 tools are installed, configure the JDK in IntelliJ IDEA.

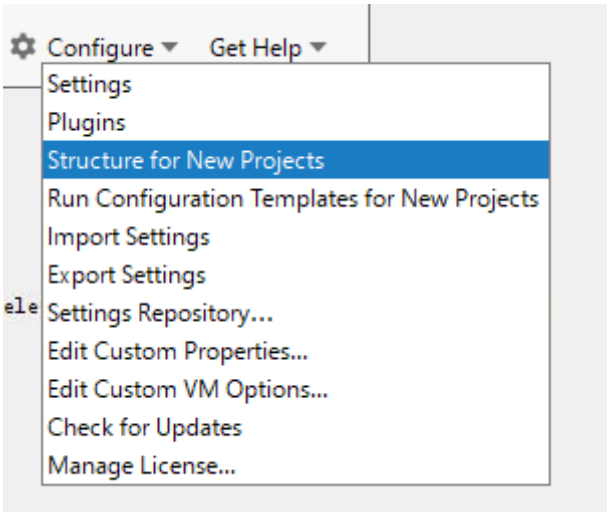
1. Start IntelliJ IDEA and choose **Configure**.

Figure 21-3 Quick Start



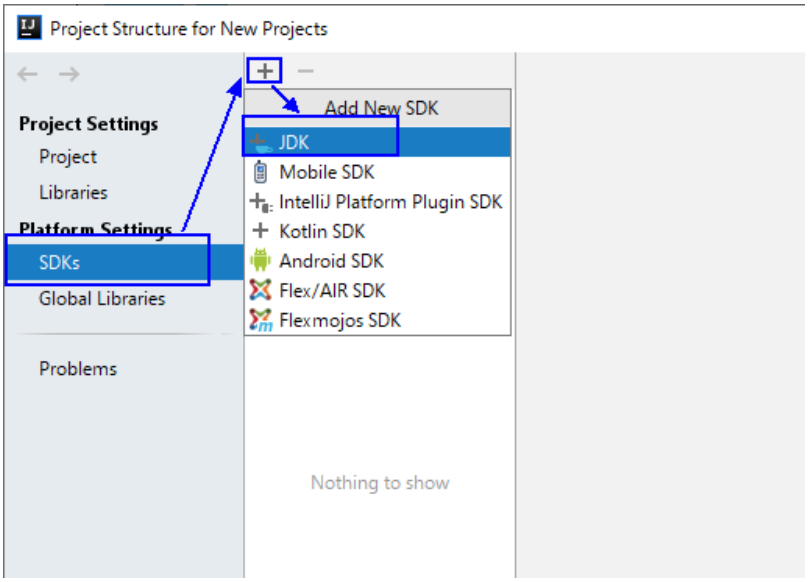
2. Select **Structure for New Projects** from the drop-down list.

Figure 21-4 Configure



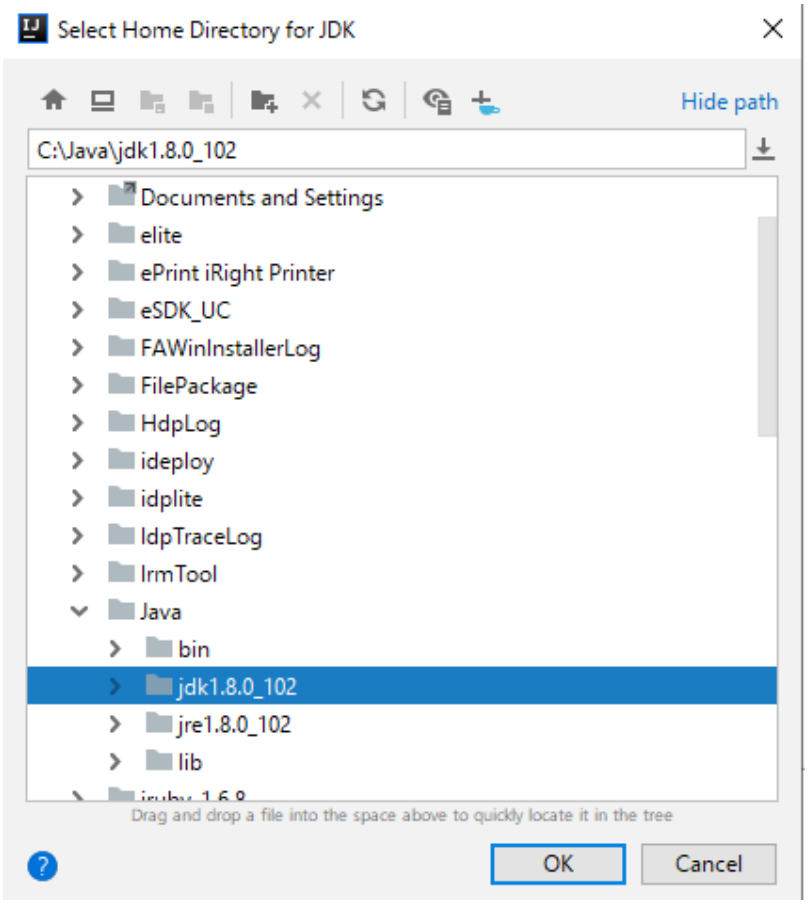
3. On the displayed **Project Structure for New Projects** page, select **SDKs**, click the plus sign (+), and click **JDK**.

Figure 21-5 Project Structure for New Projects



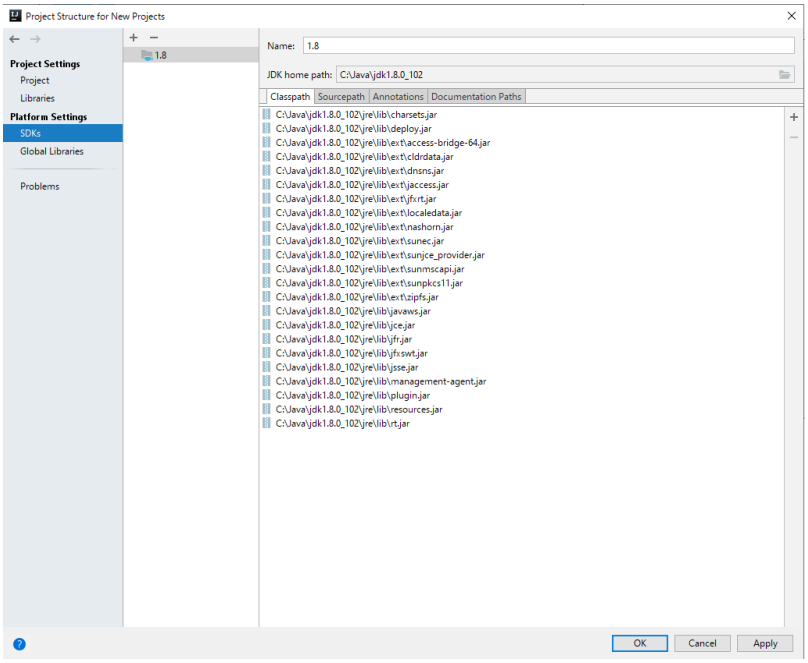
4. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 21-6 Select Home Directory for JDK



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

Figure 21-7 Completing the JDK configuration



NOTE

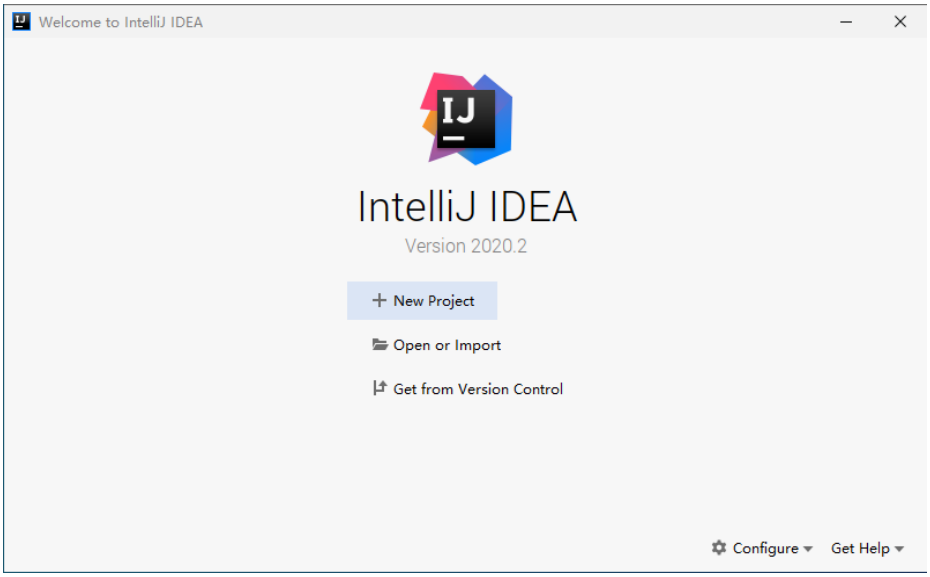
The operation procedure may vary according to the IDEA version.

Step 3 Import the sample project to the IntelliJ IDEA development environment.

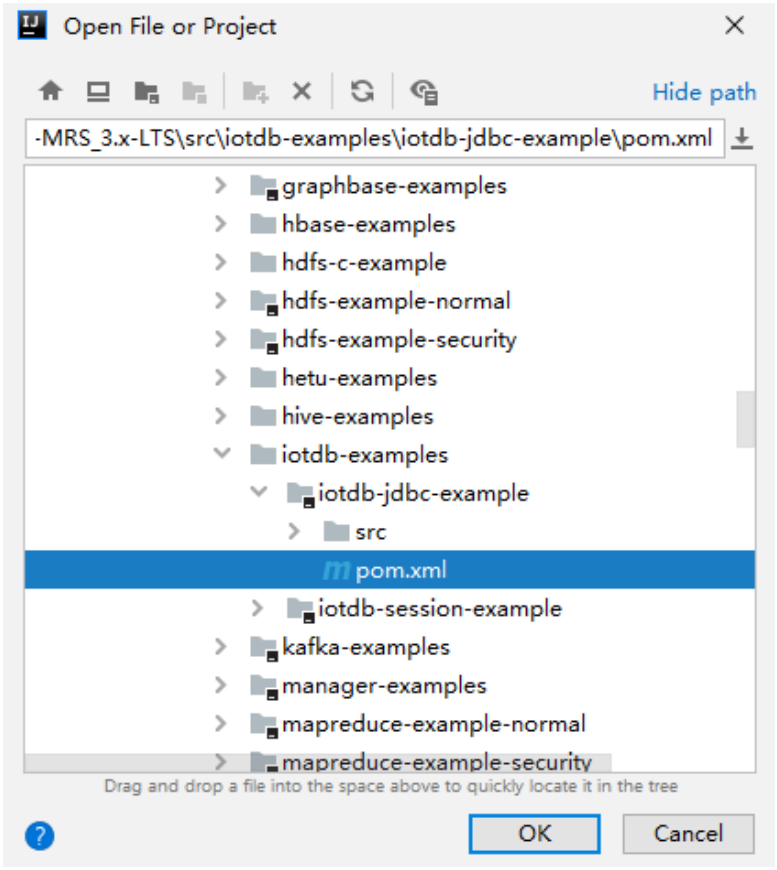
1. Start the IntelliJ IDEA, and click **Open or Import** on the quick start page.

   For the IDEA tool that has been used, you can choose **File** > **Import project...** on the main interface to import the sample project.

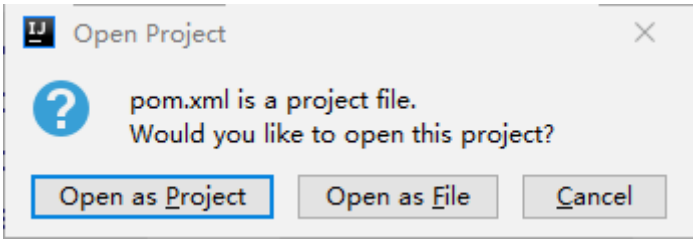
Figure 21-8 Open or Import (quick start page)



2. Select the **pom.xml** file of **iotdb-jdbc-example** in the sample project folder **iotdb-examples**, and click **OK**.

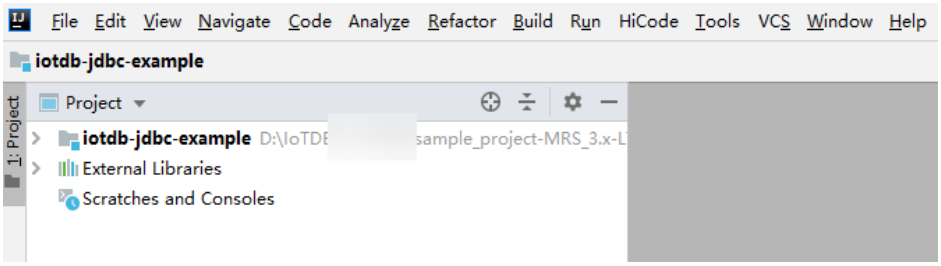


Select **Open as Project**.



3. After the import is complete, check that the imported sample projects are displayed on the IDEA home page.

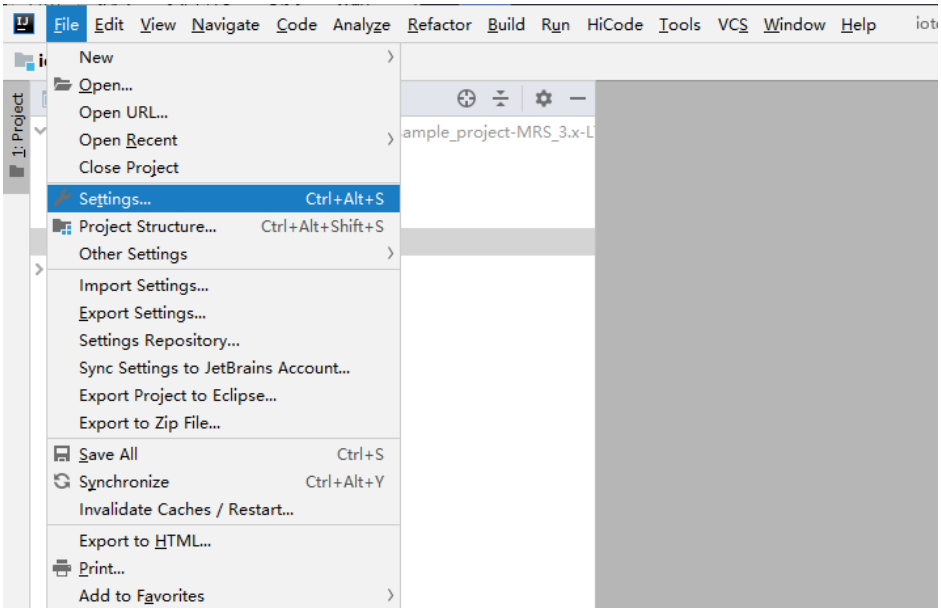
Figure 21-9 Successfully importing sample projects



Step 4 Set the Maven version used by the project.

1. Choose **File > Settings...** from the main menu of IntelliJ IDEA.

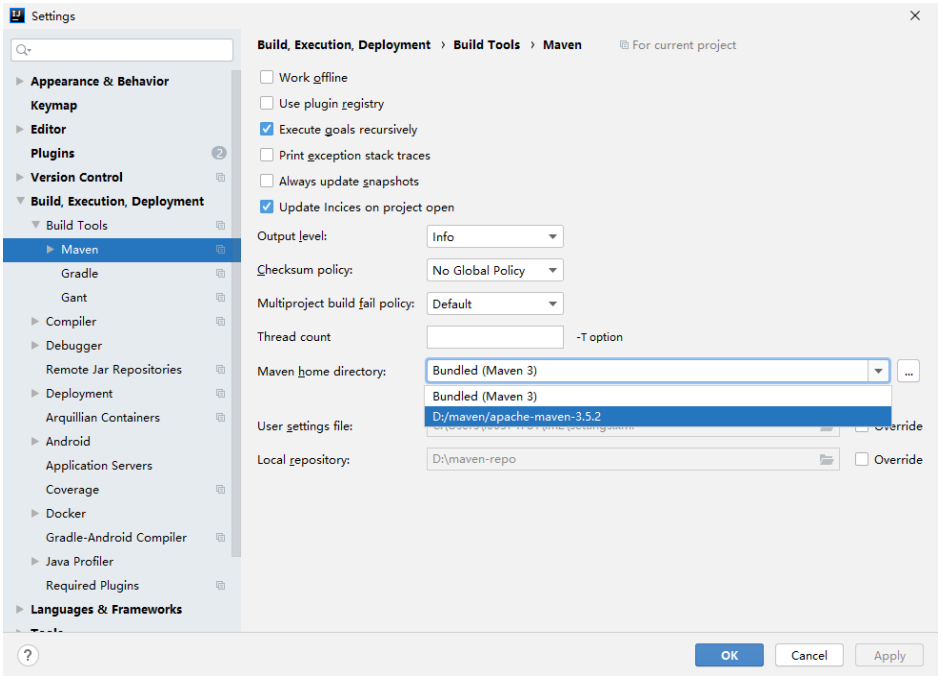
Figure 21-10 Settings



2. Choose **Build, Execution, Deployment** > **Maven** and set **Maven home directory** to the Maven version installed on the local host.

   Configure **User settings file** and **Local repository** based on the site requirements.

Figure 21-11 Selecting the local Maven installation directory



3. Click **Apply** and **OK** to complete the configuration.

----End

21.3 Application Development

21.3.1 IoTDB JDBC

21.3.1.1 Java Example Code

Description

Run the IoTDB SQL statement in JDBC connection mode.

Sample Code

The following code snippets are used as an example. For complete codes, see the **com.huawei.bigdata.iotdb.JDBCExample** class.

jdbc url includes the IP address, port number, username, and password of the node where the IoTDBServer to be connected is located.

NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- To obtain the RPC port number, log in to FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB**. Click **Configuration**, click **All Configurations**, and search for **IOTDB_SERVER_RPC_PORT**.
- In normal mode, IoTDB has a default user **root** after initial installation, and the password is **root**. This user is an administrator and has all permissions, which cannot be assigned, revoked, or deleted.

```
 private static final String IOTDB_SSL_ENABLE = "false";//To obtain the value, log in to FusionInsight Manager, choose Cluster > Services > IoTDB, click Configurations, search for SSL in the search box, and view the value of SSL_ENABLE.
public static void main(String[] args) throws ClassNotFoundException, SQLException {
    Class.forName("org.apache.iotdb.jdbc.IoTDBDriver");
    // set iotdb_ssl_enable
    System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
    if ("true".equals(IOTDB_SSL_ENABLE)) {
      // set truststore.jks path
      System.setProperty("iotdb_ssl_truststore", "truststore file path");
    }
    try (Connection connection =
            DriverManager.getConnection("jdbc:iotdb://127.0.0.1:22260/", "root", "Password");
        Statement statement = connection.createStatement()) {

      // set JDBC fetchSize
      statement.setFetchSize(10000);
      for (int i = 0; i <= 100; i++) {
        statement.addBatch(prepareInsertStatment(i));
      }
      statement.executeBatch();
      statement.clearBatch();

      ResultSet resultSet = statement.executeQuery("select ** from root where time <= 10");
      outputResult(resultSet);
      resultSet = statement.executeQuery("select count(**) from root");
      outputResult(resultSet);
      resultSet =
          statement.executeQuery(
```

```
            "select count(**) from root where time >= 1 and time <= 100 group by ([0, 100), 20ms, 20ms)");
    outputResult(resultSet);
  } catch (IoTDBSQLException e) {
    System.out.println(e.getMessage());
  }
}
```

21.3.2 IoTDB Session

21.3.2.1 Java Example Code

Description

Run the IoTDB SQL statement in Session connection mode.

Sample Code

The following code snippets are used as an example. For complete code, see **com.huawei.bigdata.SessionExample**.

Session object parameters include the IP address, port number, username, and password of the node where the IoTDBServer is located.

NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- To obtain the RPC port number, log in to FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB**. Click **Configuration**, click **All Configurations**, and search for **IOTDB_SERVER_RPC_PORT**.
- In normal mode, IoTDB has a default user **root** after initial installation, and the password is **root**. This user is an administrator and has all permissions, which cannot be assigned, revoked, or deleted.

```
 private static final String IOTDB_SSL_ENABLE = "true";//To obtain the value, log in to FusionInsight Manager, choose Cluster > Services > IoTDB, click Configurations, search for SSL in the search box, and view the value of SSL_ENABLE.
public static void main(String[] args)
    throws IoTDBConnectionException, StatementExecutionException {
  // set iotdb_ssl_enable
  System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
  if ("true".equals(IOTDB_SSL_ENABLE)) {
    // set truststore.jks path
    System.setProperty("iotdb_ssl_truststore", "truststore file path");
  }
  session = new Session(nodeUrls, "root", "Password");
  session.open(false);

  // set session fetchSize
  session.setFetchSize(10000);

  try {
    session.setStorageGroup("root.sg1");
  } catch (StatementExecutionException e) {
    if (e.getStatusCode() != TSStatusCode.PATH_ALREADY_EXIST_ERROR.getStatusCode()) {
      throw e;
    }
  }

  createTimeseries();
  createMultiTimeseries();
  insertRecord();
```

```
    insertTablet();
    insertTablets();
    insertRecords();
    nonQuery();
    query();
    queryWithTimeout();
    rawDataQuery();
    queryByIterator();
    deleteData();
    deleteTimeseries();
    setTimeout();

    sessionEnableRedirect = new Session(nodeUrls, "root", "Password");
    sessionEnableRedirect.setEnableQueryRedirection(true);
    sessionEnableRedirect.open(false);

    // set session fetchSize
    sessionEnableRedirect.setFetchSize(10000);

    insertRecord4Redirect();
    query4Redirect();
    sessionEnableRedirect.close();
    session.close();
  }
```

21.3.3 IoTDB Flink

21.3.3.1 FlinkIoTDBSink

Description

It is an integration of IoTDB and Flink. This module contains IoTDBSink and writes time series data into IoTDB using a Flink job.

Sample Code

This example shows how to send data from a Flink job to an IoTDB server.

- A simulated source SensorSource generates a data point every second.
- Flink uses IoTDBSink to consume produced data and write the data into IoTDB.

Session object parameters include the IP address, port number, username, and password of the node where the IoTDBServer is located.

NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- To obtain the RPC port number, log in to FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB**. Click **Configuration**, click **All Configurations**, and search for **IOTDB_SERVER_RPC_PORT**.
- In normal mode, IoTDB has a default user **root** after initial installation, and the password is **root**. This user is an administrator and has all permissions, which cannot be assigned, revoked, or deleted.

```
public class FlinkIoTDBSink {
  public static void main(String[] args) throws Exception {
    // run the flink job on local mini cluster
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
```



```
  IoTDBSinkOptions options = new IoTDBSinkOptions();
  options.setHost("127.0.0.1");
  options.setPort(22260);
  options.setUser("Username");
  options.setPassword("Password");

  // If the server enables auto_create_schema, then we do not need to register all timeseries
  // here.
  options.setTimeseriesOptionList(
          Lists.newArrayList(
                  new IoTDBSinkOptions.TimeseriesOption(
                          "root.sg.d1.s1", TSDataType.DOUBLE, TSEncoding.GORILLA, CompressionType.SNAPPY)));

  IoTSerializationSchema serializationSchema = new DefaultIoTSerializationSchema();
  IoTDBSink ioTDBSink =
      new IoTDBSink(options, serializationSchema)
          // enable batching
          .withBatchSize(10)
          // how many connections to the server will be created for each parallelism
          .withSessionPoolSize(3);

  env.addSource(new SensorSource())
      .name("sensor-source")
      .setParallelism(1)
      .addSink(ioTDBSink)
      .name("iotdb-sink");

  env.execute("iotdb-flink-example");
 }

 private static class SensorSource implements SourceFunction<Map<String, String>> {
  boolean running = true;
  Random random = new SecureRandom();

  @Override
  public void run(SourceContext context) throws Exception {
    while (running) {
      Map<String, String> tuple = new HashMap();
      tuple.put("device", "root.sg.d1");
      tuple.put("timestamp", String.valueOf(System.currentTimeMillis()));
      tuple.put("measurements", "s1");
      tuple.put("types", "DOUBLE");
      tuple.put("values", String.valueOf(random.nextDouble()));

      context.collect(tuple);
      Thread.sleep(1000);
    }
  }

  @Override
  public void cancel() {
    running = false;
  }
 }
}
```

21.3.3.2 FlinkIoTDBSource

Description

It is an integration of IoTDB and Flink. This module contains IoTDBSource and reads time series data from IoTDB and prints the data using a Flink job.

Sample Code

This example shows how a Flink job reads time series data from a IoTDB server.

- Flink uses IoTDBSource to read data from a IoTDB server.
- To use IoTDBSource, you need to construct an IoTDBSource instance by specifying **IoTDBSourceOptions** and implementing the abstract method **convert()** in IoTDBSource. The **convert()** method defines how you want to convert row data.

Session object parameters include the IP address, port number, username, and password of the node where the IoTDBServer is located.

NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- To obtain the RPC port number, log in to FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB**. Click **Configuration**, click **All Configurations**, and search for **IOTDB_SERVER_RPC_PORT**.
- In normal mode, IoTDB has a default user **root** after initial installation, and the password is **root**. This user is an administrator and has all permissions, which cannot be assigned, revoked, or deleted.

```
public class FlinkIoTDBSource {
  private static final String IOTDB_SSL_ENABLE = "true";//To obtain the value, log in to FusionInsight Manager, choose Cluster > Services > IoTDB, click Configurations, search for SSL in the search box, and view the value of SSL_ENABLE.
  static final String LOCAL_HOST = "127.0.0.1";
  static final String ROOT_SG1_D1_S1 = "root.sg1.d1.s1";
  static final String ROOT_SG1_D1 = "root.sg1.d1";

  public static void main(String[] args) throws Exception {
    // use session api to create data in IoTDB
    prepareData();

    // run the flink job on local mini cluster
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

    IoTDBSourceOptions ioTDBSourceOptions =
        new IoTDBSourceOptions(
            LOCAL_HOST, 22260, "root", "Password", "select s1 from " + ROOT_SG1_D1 + " align by device");

    IoTDBSource<RowRecord> source =
        new IoTDBSource<RowRecord>(ioTDBSourceOptions) {
          @Override
          public RowRecord convert(RowRecord rowRecord) {
            return rowRecord;
          }
        };
    env.addSource(source).name("sensor-source").print().setParallelism(2);
    env.execute();
  }

  private static void prepareData()
      throws IoTDBConnectionException, StatementExecutionException, TTransportException {
    // set iotdb_ssl_enable
    System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
    if ("true".equals(IOTDB_SSL_ENABLE)) {
      // set truststore.jks path
      System.setProperty("iotdb_ssl_truststore", "truststore file path");
    }
    Session session = new Session(LOCAL_HOST, 22260, "root", "Password");
    session.open(false);
    try {
      session.setStorageGroup("root.sg1");
      if (!session.checkTimeseriesExists(ROOT_SG1_D1_S1)) {
        session.createTimeseries(
            ROOT_SG1_D1_S1, TSDataType.INT64, TSEncoding.RLE, CompressionType.SNAPPY);
        List<String> measurements = new ArrayList<>();
```

```
      measurements.add("s1");
      measurements.add("s2");
      measurements.add("s3");
      List<TSDataType> types = new ArrayList<>();
      types.add(TSDataType.INT64);
      types.add(TSDataType.INT64);
      types.add(TSDataType.INT64);

      for (long time = 0; time < 1000; time++) {
        List<Object> values = new ArrayList<>();
        values.add(1L);
        values.add(2L);
        values.add(3L);
        session.insertRecord(ROOT_SG1_D1, time, measurements, types, values);
      }
    }
  } catch (StatementExecutionException e) {
    if (e.getStatusCode() != TSStatusCode.PATH_ALREADY_EXIST_ERROR.getStatusCode()) {
      throw e;
    }
  }
 }
}
```

21.3.4 IoTDB Kafka

21.3.4.1 Java Sample Code

Description

This example shows how to send data to IoTDB using Kafka.

Sample Code

- Producer.java:

  This example shows how to send time series data to a Kafka cluster.

  a. Change the value of the **public final static String TOPIC** variable in the **KafkaProperties.java** file based on the site requirements, for example, **kafka-topic**.

  b. The default time series data format of this sample is *Device name*,*Timestamp*,*Value*, for example, **sensor_1,1642215835758,1.0**. You can change the value of **IOTDB_DATA_SAMPLE_TEMPLATE** in the **Constant.java** file based on the site requirements.

  ```
  public static void main(String[] args) {
     // whether use security mode
     final boolean isSecurityModel = false;
     ...
     // whether to use the asynchronous sending mode
     final boolean asyncEnable = false;
     Producer producerThread = new Producer(KafkaProperties.TOPIC, asyncEnable);
  }
  ```

  NOTE

  For details about the Kafka producer code, see **Producer API Usage Sample**.

- KafkaConsumerMultThread.java:

  This example shows how to write data from a Kafka cluster to IoTDB using multiple threads. Kafka cluster data is generated by **Producer.java**.

a. Change the value of the **public final static String TOPIC** variable in the **KafkaProperties.java** file based on the site requirements, for example, **kafka-topic**.

b. Change the value of **CONCURRENCY_THREAD_NUM** in the **KafkaConsumerMultThread.java** file to adjust the number of consumer threads.

NOTICE

If multiple threads are used to consume Kafka cluster data, ensure that the number of consumed topic partitions is greater than 1.

c. Set the IP address, port number, username, and password of the node where the IoTDBServer is located in the IoTDBSessionPool object parameters.

NOTE

- On FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB** and click the **Instance** tab to view the IP address of the node where the IoTDBServer to be connected is located.
- To obtain the RPC port number, log in to FusionInsight Manager, choose **Cluster** > **Services** > **IoTDB**. Click **Configuration**, click **All Configurations**, and search for **IOTDB_SERVER_RPC_PORT**.

```
 private static final String IOTDB_SSL_ENABLE = "true";//To obtain the value, log in to FusionInsight Manager, choose Cluster > Services > IoTDB, click Configurations, search for SSL in the search box, and view the value of SSL_ENABLE.
public static void main(String[] args) {
    // whether use security mode
    final boolean isSecurityModel = false;
    ...

// set iotdb_ssl_enable
    System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
    if ("true".equals(IOTDB_SSL_ENABLE)) {
      // set truststore.jks path
      System.setProperty("iotdb_ssl_truststore", "truststore file path");
    }
    // create IoTDB session connection pool
    IoTDBSessionPool sessionPool = new IoTDBSessionPool("127.0.0.1", 22260, "root", "Password", 3);

    // start consumer thread
    KafkaConsumerMultThread consumerThread = new KafkaConsumerMultThread(KafkaProperties.TOPIC, sessionPool);
    consumerThread.start();
}

    /**
     * consumer thread
     */
    private class ConsumerThread extends ShutdownableThread {
        private int threadNum;
        private String topic;
        private KafkaConsumer<String, String> consumer;
        private IoTDBSessionPool sessionPool;

        public ConsumerThread(int threadNum, String topic, Properties props, IoTDBSessionPool sessionPool) {
            super("ConsumerThread" + threadNum, true);
            this.threadNum = threadNum;
            this.topic = topic;
            this.sessionPool = sessionPool;
```



```
            this.consumer = new KafkaConsumer<>(props);
            consumer.subscribe(Collections.singleton(this.topic));
        }

        public void doWork() {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(waitTime));
            for (ConsumerRecord<String, String> record : records) {
                LOG.info("Consumer Thread-" + this.threadNum + " partitions:" + record.partition() + "
record: "
                        + record.value() + " offsets: " + record.offset());

                // insert kafka consumer data to iotdb
                sessionPool.insertRecord(record.value());
            }
        }
    }
```

NOTE

For details about the Kafka consumer code, see **Multi-thread Producer Sample**.

21.3.5 IoTDB UDF Program

21.3.5.1 IoTDB UDF Sample Code

Description

This section describes how to implement a simple IoTDB user-defined function (UDF). For details, see section **UDF Sample Code and Operations**.

Sample Code

```
package com.huawei.bigdata.iotdb;
import org.apache.iotdb.udf.api.UDTF;
import org.apache.iotdb.udf.api.access.Row;
import org.apache.iotdb.udf.api.collector.PointCollector;
import org.apache.iotdb.udf.api.customizer.config.UDTFConfigurations;
import org.apache.iotdb.udf.api.customizer.parameter.UDFParameters;
import org.apache.iotdb.udf.api.customizer.strategy.RowByRowAccessStrategy;
import org.apache.iotdb.udf.api.type.Type;
import java.io.IOException;

public class UDTFExample implements UDTF {
    @Override
     public void beforeStart(UDFParameters parameters, UDTFConfigurations configurations) {
       configurations.setAccessStrategy(new RowByRowAccessStrategy()).setOutputDataType(Type.INT32);
    }

    @Override
    public void transform(Row row, PointCollector collector) throws IOException {
      collector.putInt(row.getTime(), -row.getInt(0));
    }
}
```

21.4 Application Commissioning

21.4.1 Commissioning Applications on Windows

21.4.1.1 Compiling and Running Applications

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.

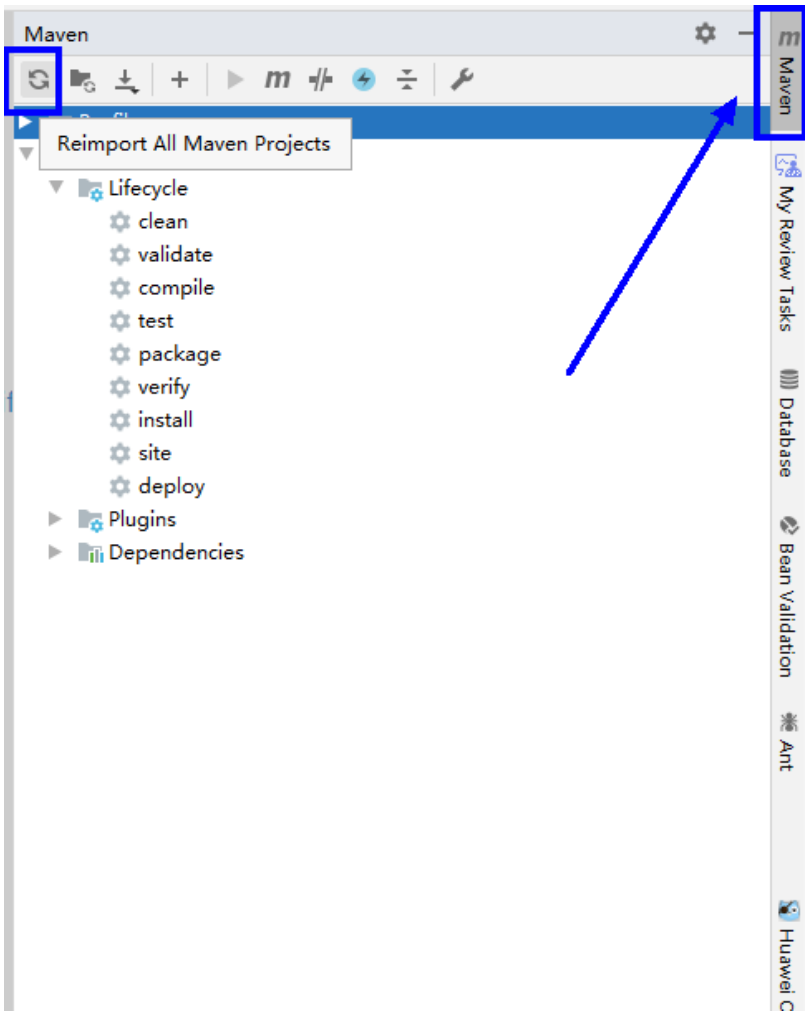
NOTE

- If the IBM JDK is used in the Windows environment, applications cannot be directly 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 21-12 reimport projects



Step 2 Compile and run the application.

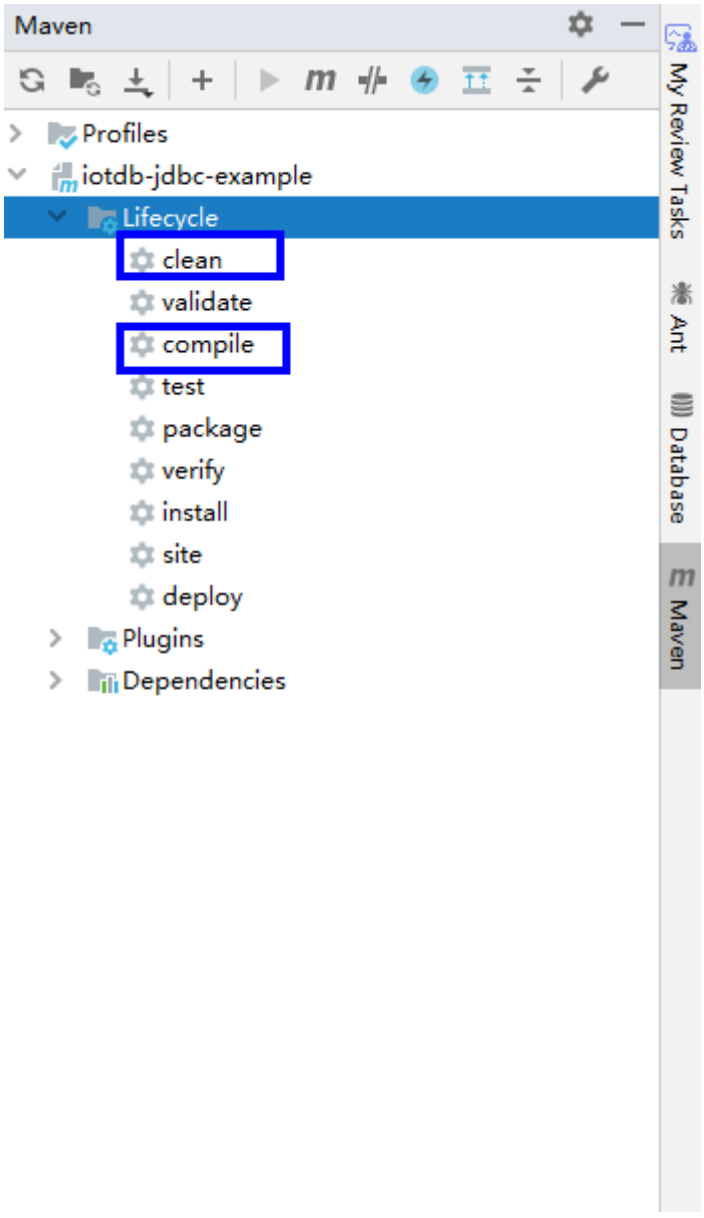
Modify the IP address, port number, login username, and password of the IoTDBServer node that matches the code.

1. Use either of the following two methods:
   - 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** and double-click **compile** to run the **compile** command of Maven.

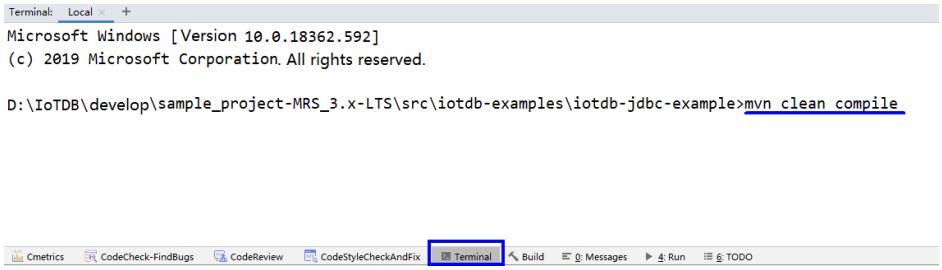
     Figure 21-13 Maven tools clean and compile

   - Method 2

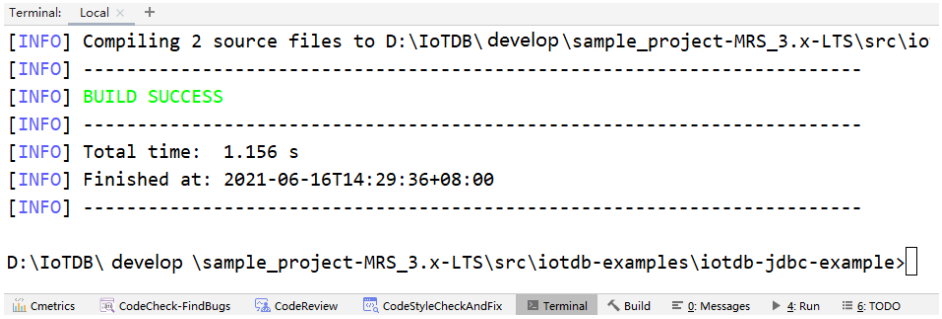
Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

Figure 21-14 Enter **mvn clean compile** in the IDEA **Terminal** text box.



After the compilation is complete, the message "BUILD SUCCESS" is displayed.

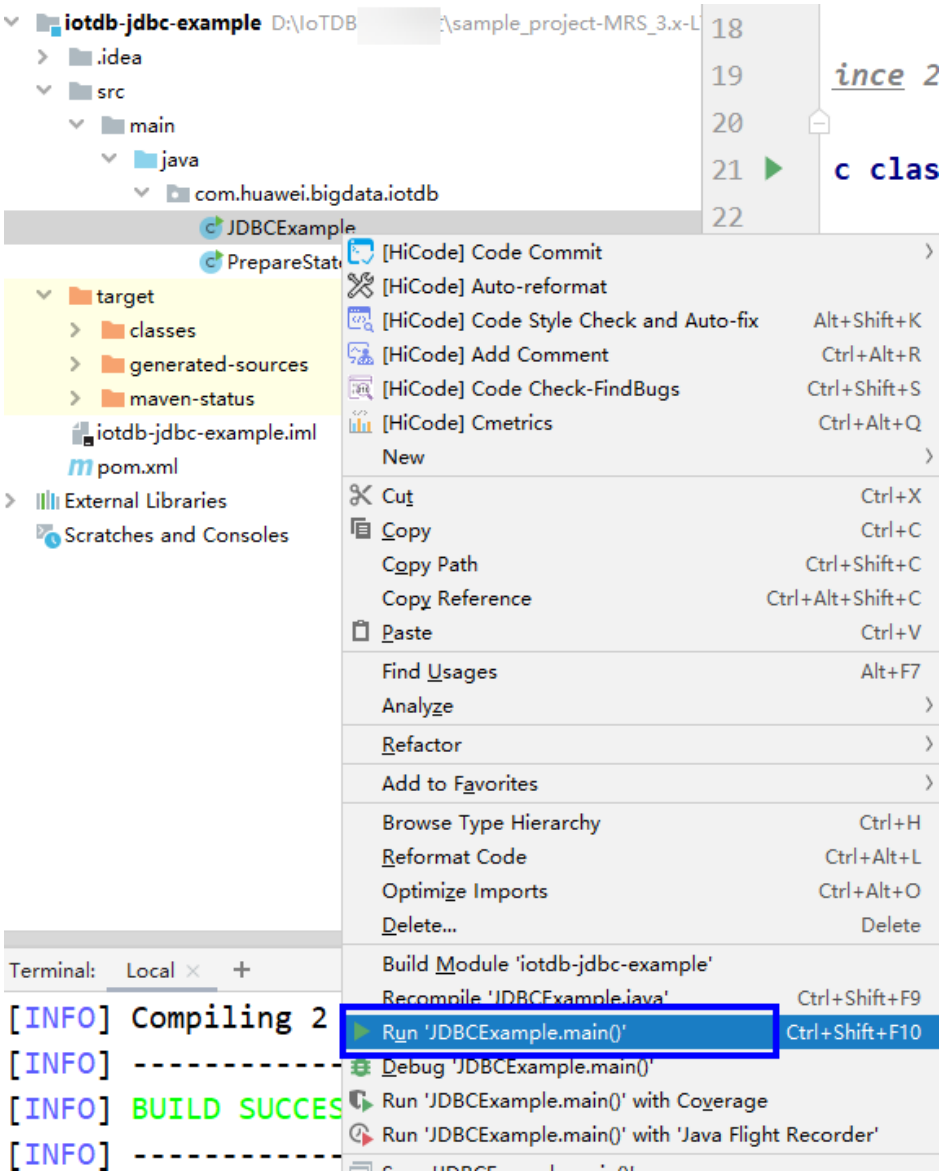
Figure 21-15 Compilation completed



2. Run the application.

   Right-click the **JDBCExample.java** file and choose **Run 'JDBCExample.main()** from the shortcut menu.

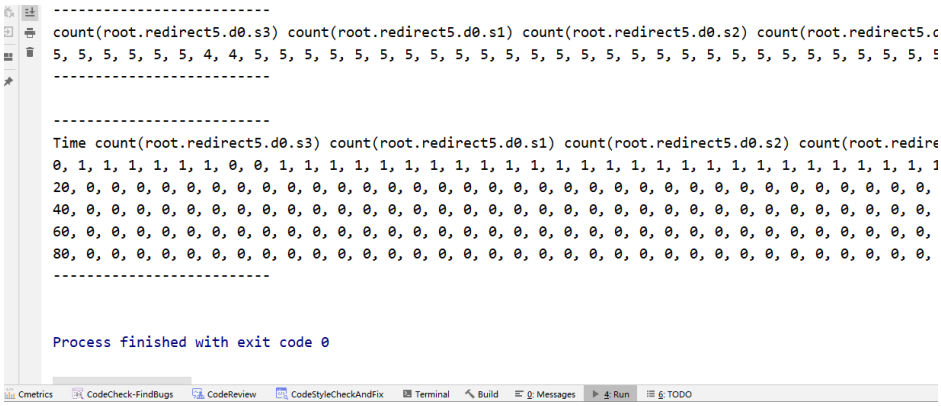
Figure 21-16 Running the application



----End

21.4.1.2 Viewing Commissioning Results

After the IoTDB application is run, you can view the running results in IntelliJ IDEA.



21.4.2 Commissioning JDBC and Session Applications on Linux

21.4.2.1 Compiling and Running Applications

Scenario

IoTDB applications can run in a Linux environment where an IoTDB client is installed. After the application code is developed, you can upload the JAR file to the prepared Linux environment. A Session application is used as an example. Operations for JDBC applications are the same as those for Session applications.

Prerequisites

- You have installed the IoTDB client.
- If the host where the client is installed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts** file on the node where the client is located. The host names and IP addresses must be mapped one by one.

Procedure

Step 1 Export a JAR file.

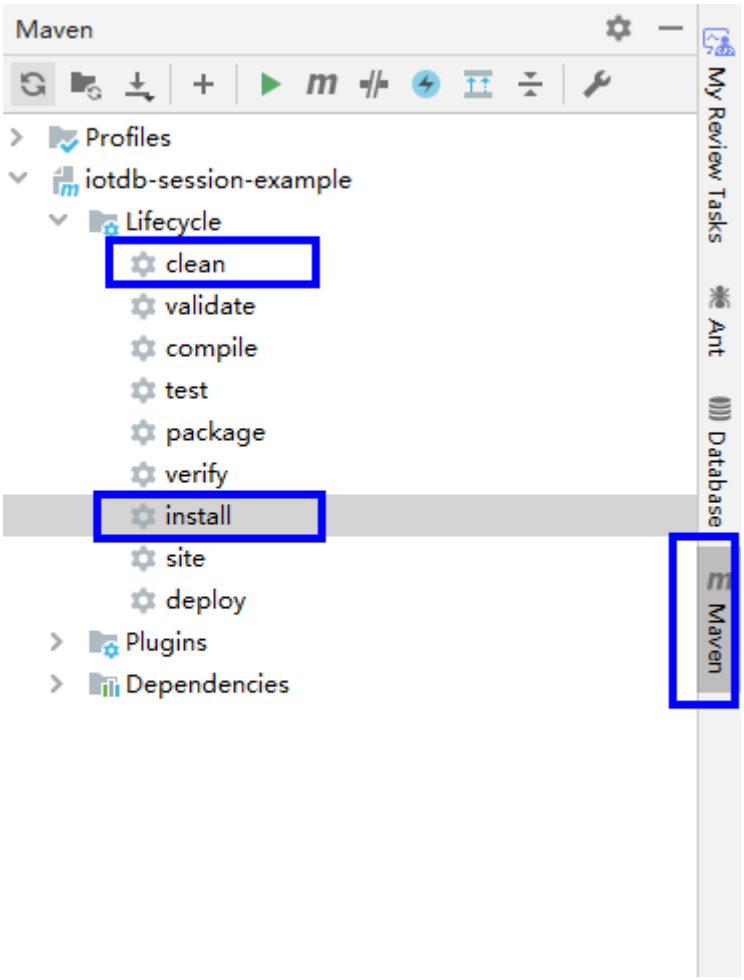
You can build a JAR file in either of the following ways:

- Method 1:

  Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

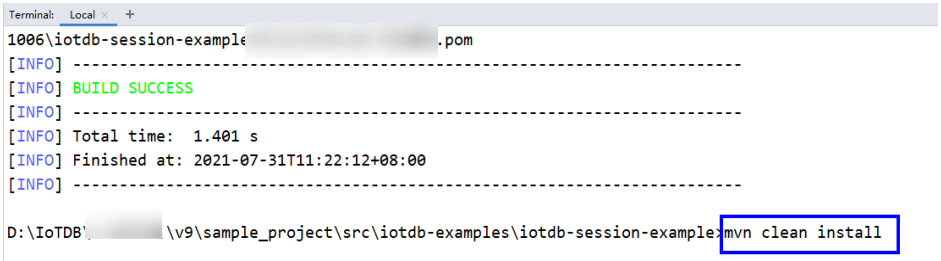
  Choose **Maven** > *Sample project name* > **Lifecycle** > **install** and double-click **install** to run the **install** command of Maven.

Figure 21-17 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 file.

  Figure 21-18 Entering mvn clean install 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 Prepare the dependent JAR file.

1. Log in to the IoTDB client and import the JAR file generated in **Step 1** to the **lib** directory of the IoTDB client, for example, **/opt/client/IoTDB/iotdb/lib**.

2. In the root directory of the IoTDB client, for example, **/opt/client/IoTDB/iotdb**, create the **run.sh** script, modify the content as follows, and save the modification.

```
#!/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.iotdb.JDBCExample
```

   com.huawei.bigdata.iotdb.JDBCExample is an example. Use the actual code instead.

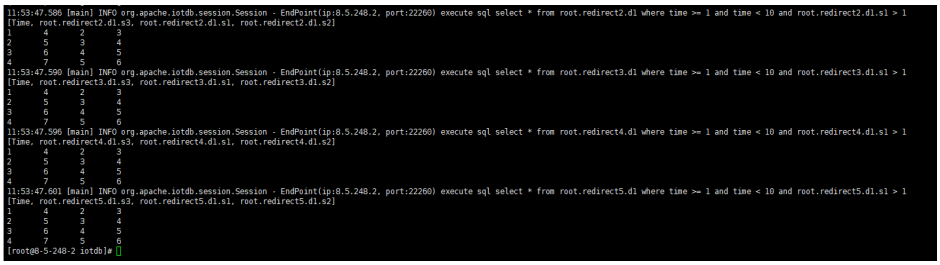
3. Run the **run.sh** script to run the JAR file.

   sh /opt/client/IoTDB/iotdb/run.sh

----End

21.4.2.2 Viewing Commissioning Results

If the application running is successful, the following information is displayed.



21.4.3 Commissioning Flink Applications on Flink Web UI and Linux

21.4.3.1 Compiling and Running Applications

Scenario

IoTDB applications can run in a Linux environment where the Flink client is installed and in an environment where the Flink web UI is installed. After the application code is developed, you can upload the JAR file to the prepared environment.

Prerequisites

- The Flink component has been installed in the cluster and the FlinkServer instance has been added.
- The cluster client that contains the Flink service has been installed, for example, in the **/opt/client** directory.

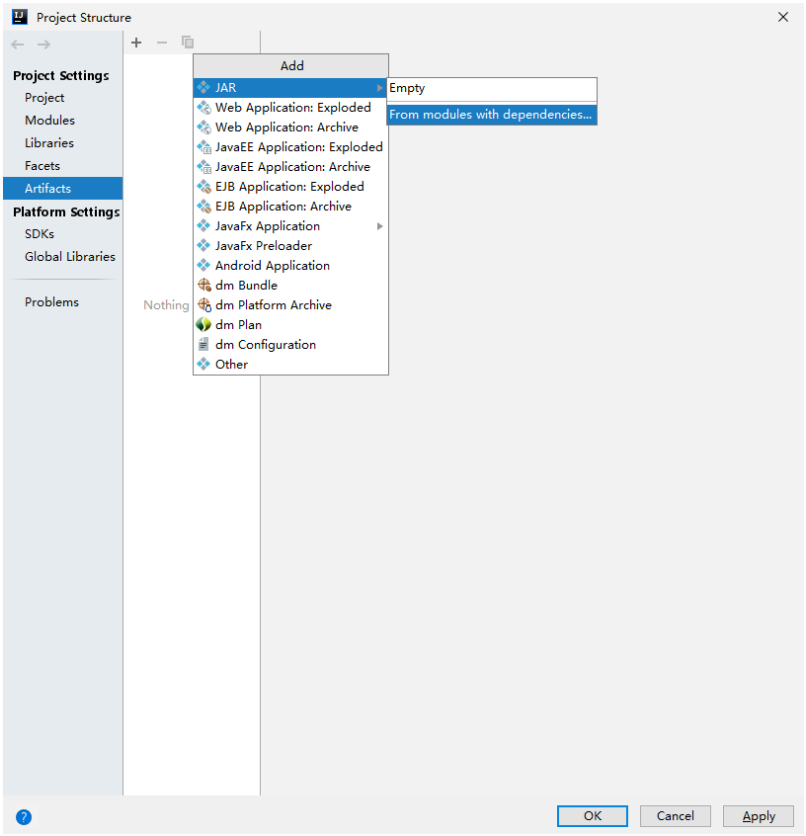
- If the host where the client is installed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts** file on the node where the client is located. The host names and IP addresses must be mapped one by one.

Procedure

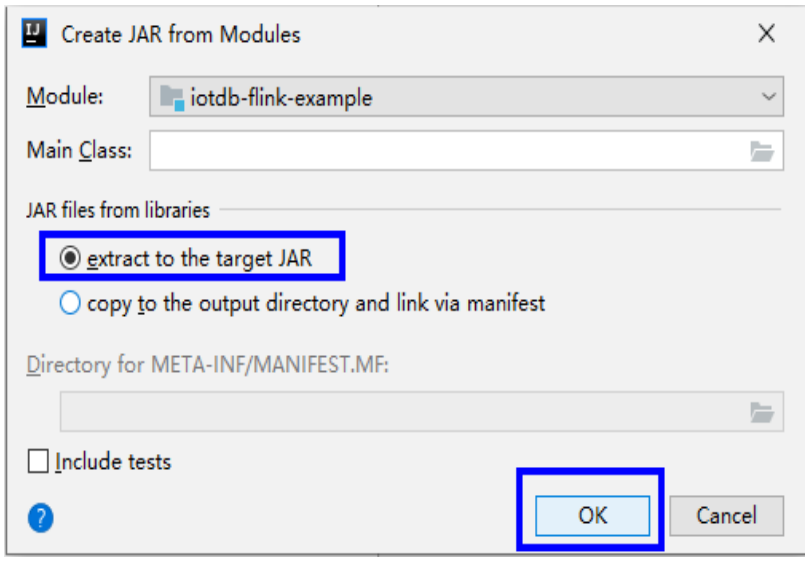
Step 1 Build a JAR file.

- In IntelliJ IDEA, configure **Artifacts** of the project before generating a JAR file.
  a. On the IDEA homepage, choose **File** > **Project Structures...** to go to the **Project Structure** page.
  b. On the **Project Structure** page, select **Artifacts**, click **+**, and select **From modules with dependencies...**.

  Figure 21-19 Adding Artifacts



  c. Select **extract to the target JAR** and click **OK**.

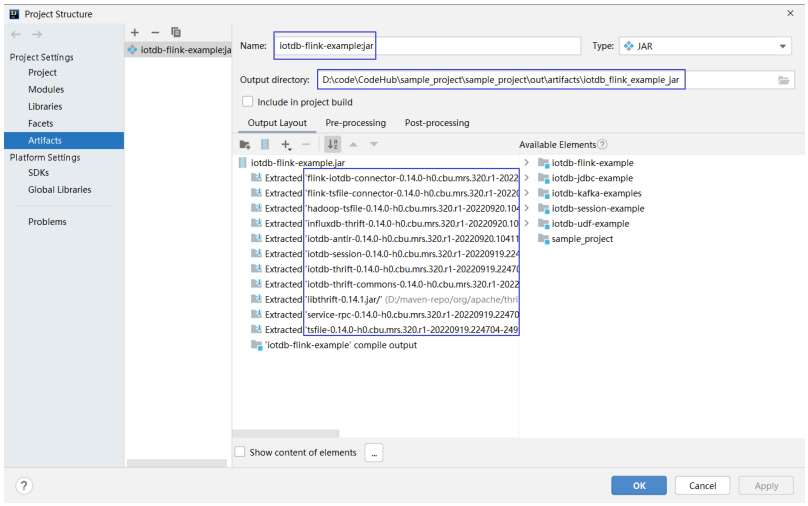


d. Set the name, type, and output path of the JAR file based on the site requirements.

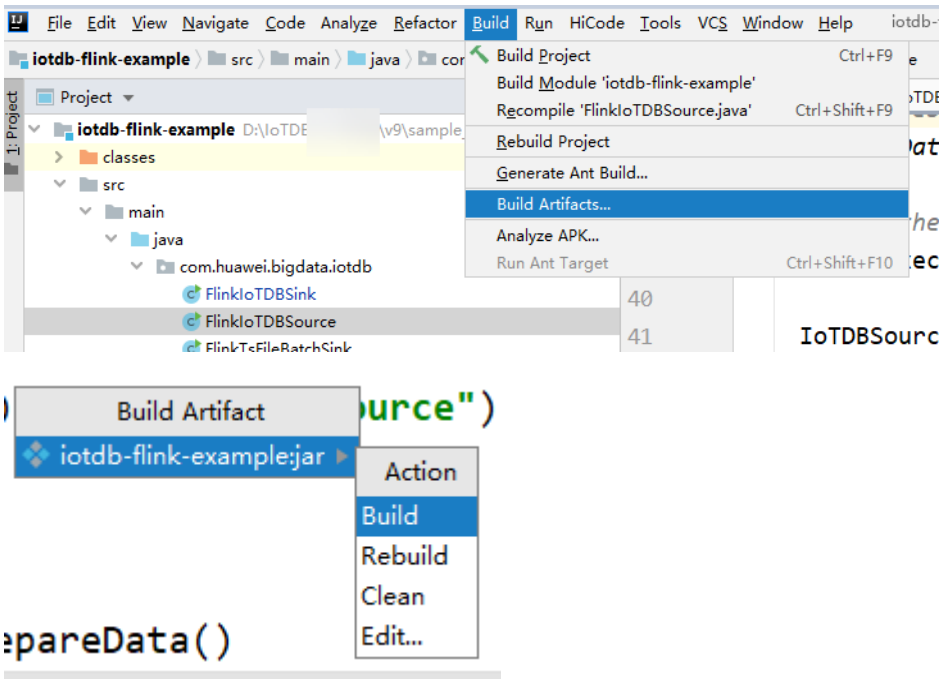
To avoid JAR file conflicts caused by unnecessary JAR files, you only need to load the following basic JAR files related to IoTDB:

- flink-iotdb-connector-*
- flink-tsfile-connector-*
- iotdb-session-*
- iotdb-thrift-*
- service-rpc-*
- tsfile-*

Click **OK**.



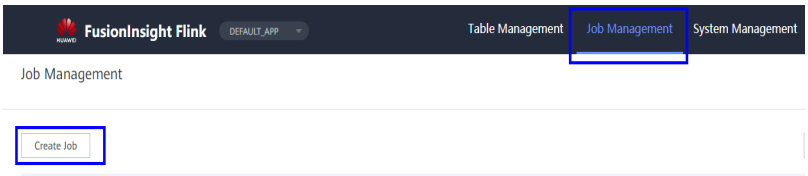
e. On the IDEA home page, choose **Build > Build Artifacts...**. On the **Build Artifact** page that is displayed, choose **Action** > **Build**.



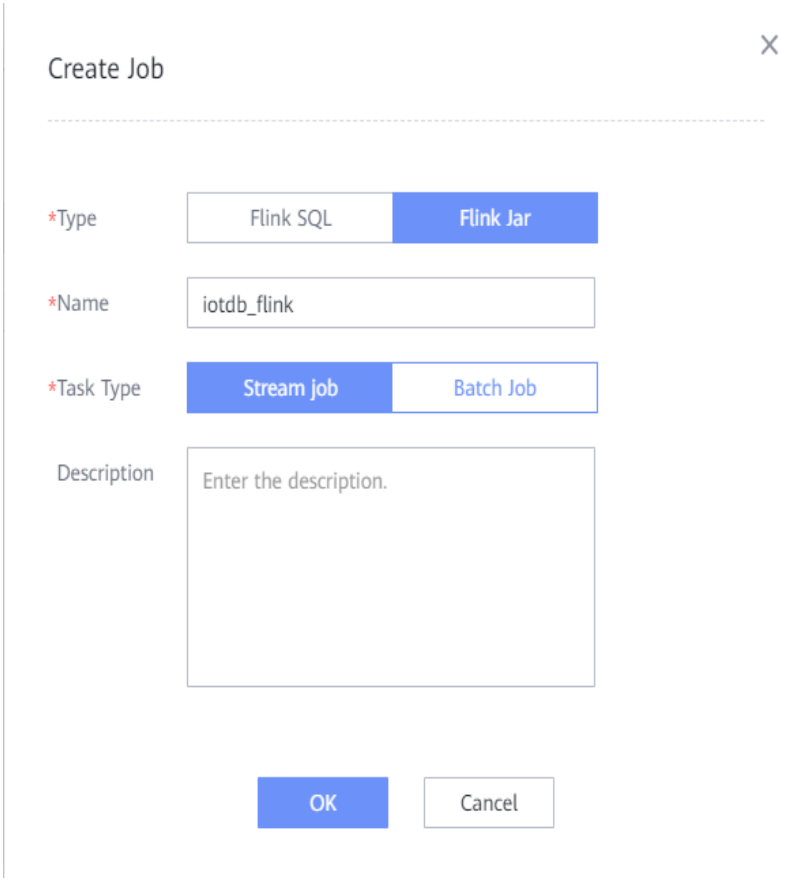
f. After the build is successful, the message "Build completed successfully" is displayed in the lower right corner, and the corresponding JAR file is generated in the **Output Directory** directory.

Step 2 (Scenario 1) Run a Flink job on the Flink web UI.

1. Log in to FusionInsight Manager of the cluster as a user who has the FlinkServer web UI management permission, choose **Cluster** > **Services** > **Flink**, and click the hyperlink next to **Flink WebUI** on the dashboard page to go to the FlinkServer web UI.
2. On the FusionInsight Flink web UI, choose **Job Management** > **Create Job** to create a job.

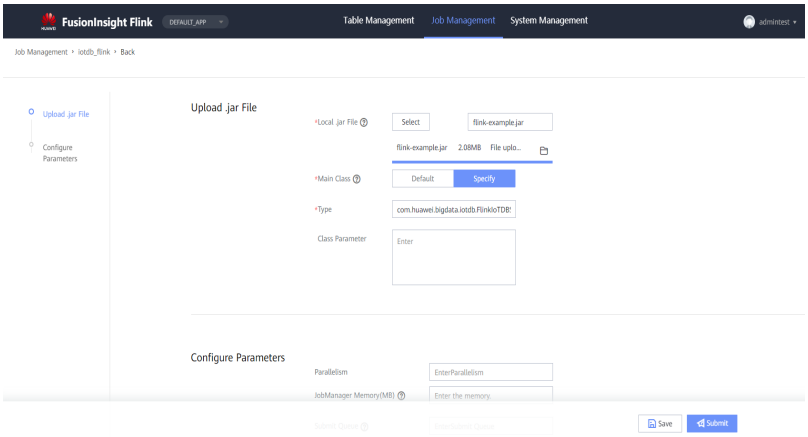


3. Set **Type** to **Flink Jar**, enter the name of the job to be created, select a task type, and click **OK**.



4. Upload the JAR file generated in **Step 1**, set **Main Class** to **Specify**, enter the class to be executed in **Class Parameter**, and click **Submit**.

   For example, set **Type** to **com.huawei.bigdata.iotdb.FlinkIoTDBSink** (development program that executes **FlinkIoTDBSink**) or **com.huawei.bigdata.iotdb.FlinkIoTDBSource** (development program that executes **FlinkIoTDBSource**).



Step 3 (Scenario 2) Submit a Flink job on the Flink client in the Linux environment.

- Save the JAR file generated in **Step 1** to the Flink running environment (Flink client) in the Linux environment, for example, **/opt/client**.
- Start the Flink cluster before running the Flink applications on Linux. Run the **yarn session** command on the Flink client to start the Flink cluster. The following is a command example:


```
bin/yarn-session.sh -jm 1024 -tm 1024
```

- Run the **flink-example.jar** sample application.

  Open another window on the terminal. Go to the Flink client directory and invoke the **bin/flink run** script to run code.

  ```
  bin/flink flink run --class com.huawei.bigdata.iotdb.FlinkIoTDBSink /opt/client/Flink/flink/flink-example.jar
  ```

  com.huawei.bigdata.iotdb.FlinkIoTDBSink is the application in **FlinkIoTDBSink**.

----End

21.4.3.2 Viewing Commissioning Results

1. Check whether the job is executed.
   - Using the Flink web UI

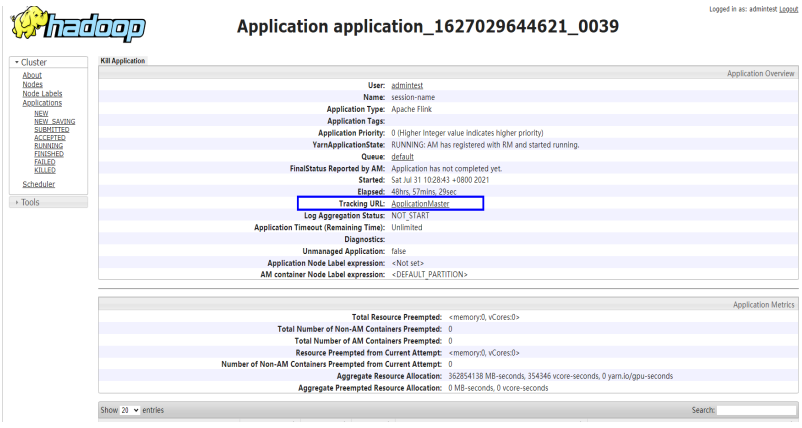
     If a success message is returned on the Flink web UI, the execution is successful. You can choose **More** > **Job Monitoring** in the **Operation** column to view detailed logs.

   - Using a Flink Client

     Log in to FusionInsight Manager as a running user, go to the native page of the Yarn service, find the application of the corresponding job, and click the application name to go to the job details page.

     - If the job is not complete, click **Tracking URL** to go to the native Flink service page and view the job running information.
     - If the job submitted in a session has been completed, you can click **Tracking URL** to log in to the native Flink service page to view job information.

2. Verify the job execution result.
   - FlinkIoTDBSink execution result:

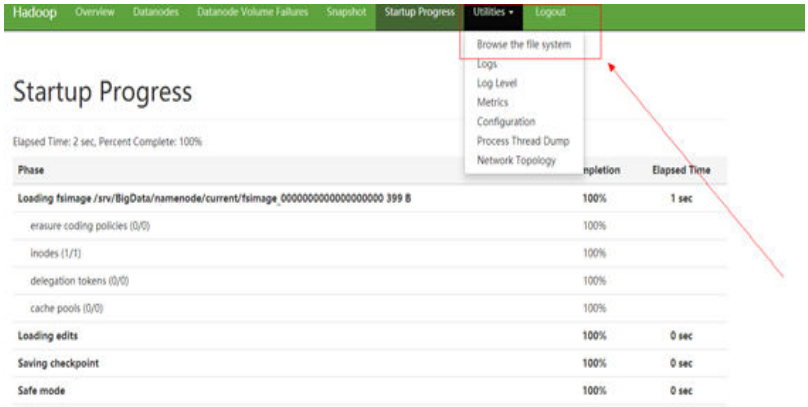
     Run the following command on the IoTDB client and check whether the data has been written from Flink to IoTDB:

SQLselect * from root.sg.d1

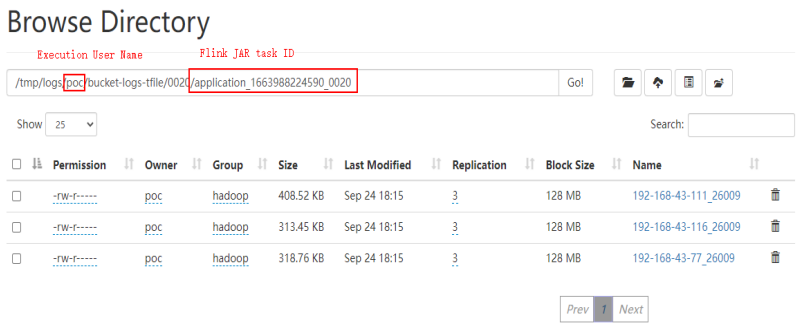
- FlinkIoTDBSource execution result:

  i. Log in to FusionInsight Manager as a running user and choose **Cluster** > **Services** > **HDFS**. Click the hyperlink next to **NameNode WebUI** to access the HDFS web UI.

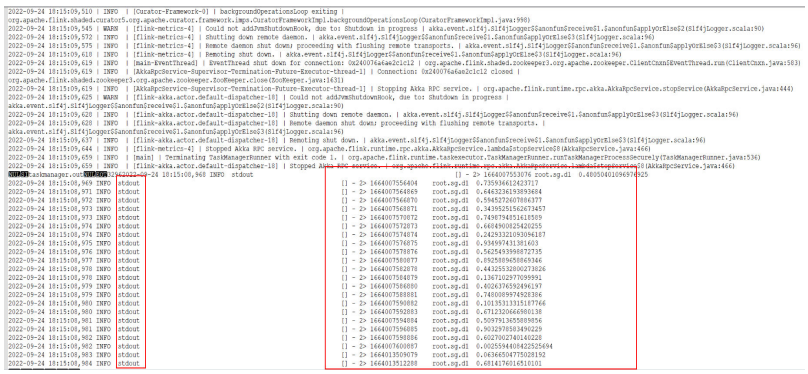
  ii. Choose **Utilities** > **Browse the file system**.



  iii. Go to the **/tmp/logs/***Execution user name*/**bucket-logs-tfile/***Task ID*/*Flink task ID* directory and download all files in the directory to the local PC.



  iv. Search for the **root.sg.d1** file from the files downloaded in **2.iii**. If the following information is displayed in the file, data is read from IoTDB.



21.4.4 Commissioning Kafka Applications on Linux

21.4.4.1 Compiling and Running Applications

Scenario

After code development is complete, you can run an IoTDB-Kafka sample application in the Linux environment.

Prerequisite

- You have installed the IoTDB and Kafka clients.
- If the host where the client is installed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts** file on the node where the client is located. The host names and IP addresses must be mapped one by one.

Procedure

Step 1 Export a JAR file.

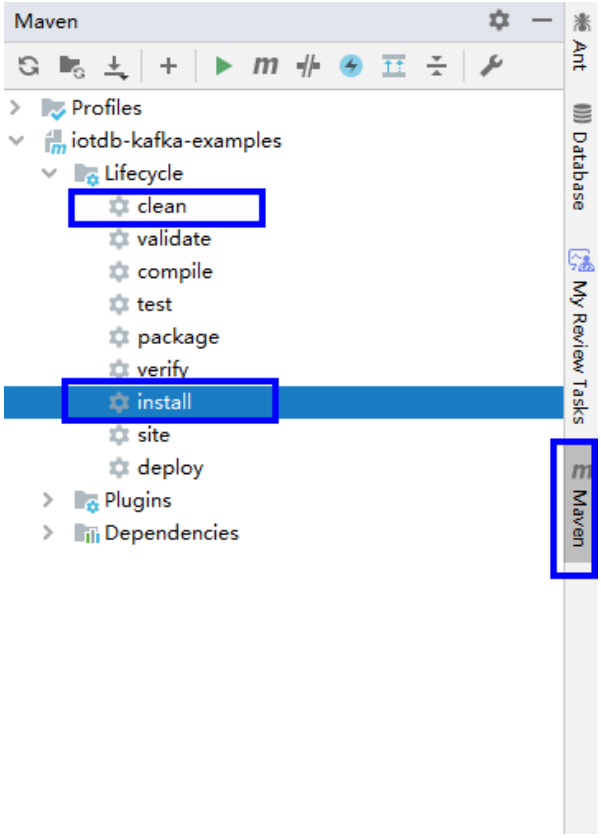
You can build a JAR file in either of the following ways:

- Method 1:

  Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

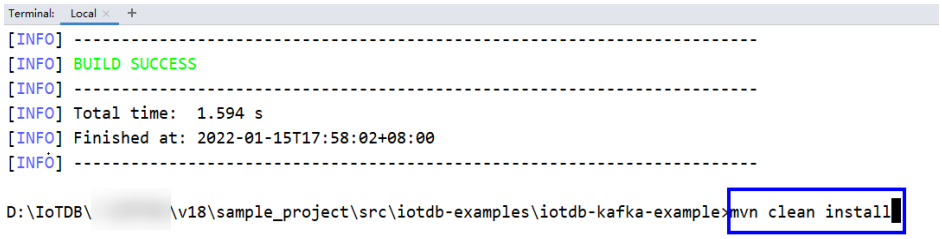
  Choose **Maven** > *Sample project name* > **Lifecycle** > **install** and double-click **install** to run the **install** command of Maven.

Figure 21-20 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 file.

  Figure 21-21 Entering **mvn clean install** 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 Prepare the dependent JAR file.

1. Go to the client installation directory, create the **lib** directory, and import the JAR package generated in **Step 1** to the **lib** directory, for example, **/opt/client/lib**.
2. Go to the Kafka client and copy the JAR package on which Kafka depends to the **lib** directory in **Step 2.1**. The following is an example directory:

   cp /opt/client/Kafka/kafka/libs/*.jar /opt/client/lib

3. Go to the IoTDB client and copy the JAR package on which IoTDB depends to the **lib** directory in **Step 2.1**. The following is an example directory:

   cp /opt/client/IoTDB/iotdb/lib/*.jar /opt/client/lib

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**.

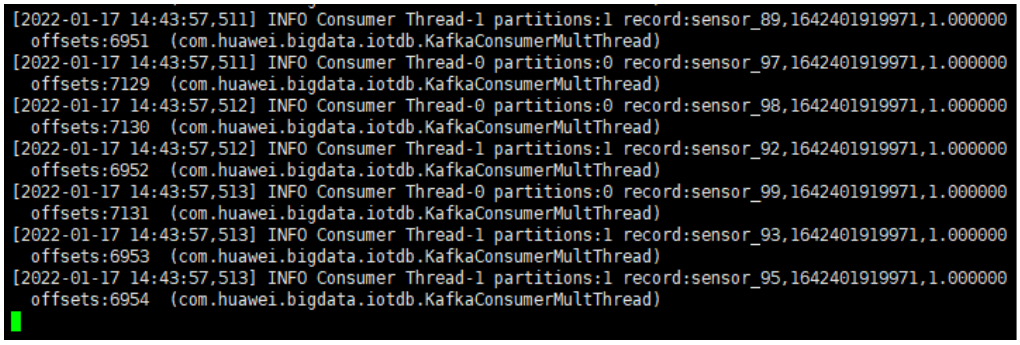
Step 3 Go to the **/opt/client** directory and ensure that the current user has the read permission on all files in the **src/main/resources** directory and the dependency library folder. In addition, ensure that JDK has been installed and Java environment variables have been set. Then, run the following command to run the sample project:

java -cp /opt/client/lib/*:/opt/client/src/main/resources com.huawei.bigdata.iotdb.KafkaConsumerMultThread

----End

21.4.4.2 Viewing Commissioning Results

The following information is displayed if the application is running properly:



21.4.5 Using a UDF

21.4.5.1 Registering a UDF

1. Build a JAR file.

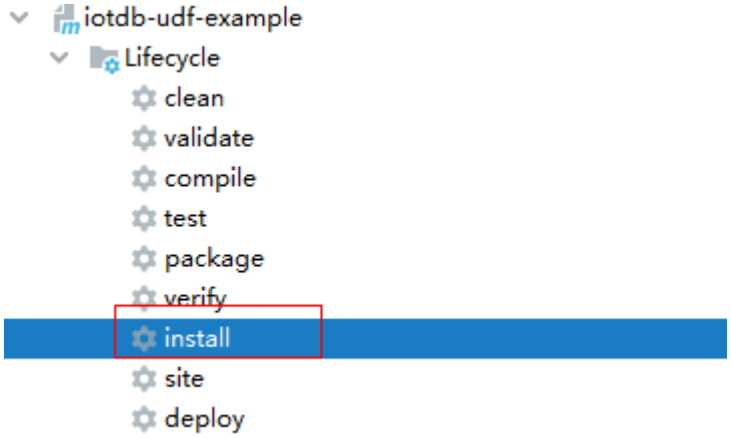
   You can build a JAR file in either of the following ways:

   - Method 1:

     Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.

     Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

Figure 21-22 Maven clean and install



– Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

Figure 21-23 Building result after mvn clean install is entered

After the build is complete, message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

2. Import the dependent JAR file.

   Log in to the node where IoTDBServer is located as user **root**, run **su - omm** to switch to user **omm**, and import the JAR file generated in **1** to the **$BIGDATA_HOME/FusionInsight_IoTDB_*/install/FusionInsight-IoTDB-*/iotdb/ext/udf** directory.

   NOTICE

   - During cluster deployment, ensure that a corresponding JAR package exists in the UDF JAR package path of each IoTDBServer node. You can modify IoTDB configuration **udf_root_dir** to specify the root path for the UDF to load JAR files.
   - To view the IP address of the node where IoTDBServer is deployed, log in to the MRS cluster management console, choose **Components**, click **IoTDB**, and click the **Instances** tab.

3. Run the following SQL statement to register the UDF:

   CREATE FUNCTION <UDF-NAME> **AS** '<UDF-CLASS-FULL-PATHNAME>'

   For example, you can run the following statement to register a UDF named **example**:

   ```
   CREATE FUNCTION example AS 'com.huawei.bigdata.iotdb.UDTFExample'
   ```


21.4.5.2 Querying a UDF

Basic SQL Syntax Supported

- SLIMIT / SOFFSET
- LIMIT / OFFSET
- NON ALIGN
- Queries with value filters
- Queries with time filters

NOTE

Currently, aligned time series are not supported in UDF queries. An error message is reported if you use UDF queries with aligned time series selected.

Queries with an Asterisk (*)

Assume that there are two time series (**root.sg.d1.s1** and **root.sg.d1.s2**).

- Execute the **SELECT example(*) from root.sg.d1** statement.

  The result set contains the results of **example(root.sg.d1.s1)** and **example(root.sg.d1.s2)**.
- Execute the **SELECT example(s1, *) from root.sg.d1** statement.

  The result set contains the results of **example(root.sg.d1.s1, root.sg.d1.s1)** and **example(root.sg.d1.s1, root.sg.d1.s2)**.
- Execute the **SELECT example(*, *) from root.sg.d1** statement.

  The result set contains the results of **example(root.sg.d1.s1, root.sg.d1.s1)**, **example(root.sg.d1.s2, root.sg.d1.s1)**, **example(root.sg.d1.s1, root.sg.d1.s2)**, and **example(root.sg.d1.s2, root.sg.d1.s2)**.

Queries with key-value pair attributes in UDF parameters

You can pass any number of key-value pair parameters to the UDF when constructing a UDF query. The key and value in a key-value pair must be enclosed in single or double quotation marks.

NOTICE

The key-value pair parameters can be passed in only after the time series have been passed in.

For example:

```
SELECT example(s1, 'key1'='value1', 'key2'='value2'), example(*, 'key3'='value3') FROM root.sg.d1;
SELECT example(s1, s2, 'key1'='value1', 'key2'='value2') FROM root.sg.d1;
```

Showing All Registered UDFs

Run the following SQL statement on the IoTDB client to view the registered UDFs:

```
SHOW FUNCTIONS
```


21.4.5.3 Deregistering a UDF

Syntax

DROP FUNCTION <UDF-NAME>

Example

Run the following command on the IoTDB client to deregister the UDF named **example**:

```
DROP FUNCTION example
```

21.5 More Information

21.5.1 Common APIs

21.5.1.1 Java API

IoTDB provides a connection pool (SessionPool) for native APIs. When using the APIs, you only need to specify the pool size to obtain connections from the pool. If you cannot get a connection in 60 seconds, a warning log will be printed, but the program continues to wait.

When a connection is used, it automatically returns to the pool and waits to be used next time. When a connection is damaged, it is deleted from the pool and a new connection is created to perform user operations again.

For query operations:

1. When SessionPool is used for query, the result set is SessionDataSetWrapper, the encapsulation class of SessionDataSet.
2. If a query result set is not traversed and you do not want to continue traversing, you need to call **closeResultSet** to release the connection.
3. If an exception is reported when you traverse a query result set, you need to call **closeResultSet** to release the connection.
4. You can call the **getColumnNames()** method of SessionDataSetWrapper to obtain the column names in the result set.

Table 21-4 Sessions APIs and corresponding parameters

Method	Description
• Session(String host, int rpcPort) • Session(String host, String rpcPort, String username, String password) • Session(String host, int rpcPort, String username, String password)	Initializes a session.
Session.open()	Opens a session.

Method	Description
Session.close()	Closes a session.
void setStorageGroup(String storageGroupId)	Sets a storage group.
• void deleteStorageGroup(String storageGroup) • void deleteStorageGroups(List<String> storageGroups)	Deletes one or more storage groups.
• void createTimeseries(String path, TSDataType dataType, TSEncoding encoding, CompressionType compressor, Map<String, String> props, Map<String, String> tags, Map<String, String> attributes, String measurementAlias) • void createMultiTimeseries(List<String> paths, List<TSDataType> dataTypes, List<TSEncoding> encodings, List<CompressionType> compressors, List<Map<String, String>> propsList, List<Map<String, String>> tagsList, List<Map<String, String>> attributesList, List<String> measurementAliasList)	Creates one or more time series.
• void deleteTimeseries(String path) • void deleteTimeseries(List<String> paths)	Deletes one or more time series.
• void deleteData(String path, long time) • void deleteData(List<String> paths, long time)	Deletes the data of one or more time series before or at a specific time point.
void insertRecord(String deviceId, long time, List<String> measurements, List<String> values)	Inserts a record, which contains the data of multiple measurement points of a device at a timestamp. Servers need to perform type inference, which may take extra time.
void insertTablet(Tablet tablet)	Inserts a tablet, which contains multiple rows of non-empty data blocks. The columns in each row are the same.
void insertTablets(Map<String, Tablet> tablet)	Inserts multiple tablets.

Method	Description
void insertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList, List<List<String>> valuesList)	Inserts multiple records. Servers need to perform type inference, which may take extra time.
void insertRecord(String deviceId, long time, List<String> measurements, List<TSDataType> types, List<Object> values)	Inserts a record, which contains the data of multiple measurement points of a device at a timestamp. With the data type information, servers do not need to perform type inference, improving the performance.
void insertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList, List<List<TSDataType>> typesList, List<List<Object>> valuesList)	Inserts multiple records. With the data type information, servers do not need to perform type inference, improving the performance.
submitApplication(SubmitApplication-Request request)	Used by the client to submit a new application to ResourceManager.
void insertRecordsOfOneDevice(String deviceId, List<Long> times, List<List<String>> measurementsList, List<List<TSDataType>> typesList, List<List<Object>> valuesList)	Inserts multiple records of the same device.
SessionDataSet executeRawDataQuery(List<String> paths, long startTime, long endTime)	Queries raw data. The interval includes the start time but does not include the end time.
SessionDataSet executeQueryState-ment(String sql)	Executes query statements.
void executeNonQueryState-ment(String sql)	Executes non-query statements.

Table 21-5 Test APIs

Method	Description
• void testInsertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList, List<List<String>> valuesList) • void testInsertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList, List<List<TSDataType>> typesList, List<List<Object>> valuesList)	Tests testInsertRecords. A response is returned immediately after data is transmitted to the server. No data is written.
• void testInsertRecord(String deviceId, long time, List<String> measurements, List<String> values) • void testInsertRecord(String deviceId, long time, List<String> measurements, List<TSDataType> types, List<Object> values)	Tests insertRecord. A response is returned immediately after data is transmitted to the server. No data is written.
void testInsertTablet(Tablet tablet)	Tests insertTablet. A response is returned immediately after data is transmitted to the server. No data is written.

22 Kafka Development Guide (Security Mode)

22.1 Overview

22.1.1 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**.

22.1.2 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.

22.1.3 Development Process

Kafka client roles include Producer and Consumer, which share the same application development process.

Figure 22-1 and **Table 22-1** show each phase of the development process.

Figure 22-1 Kafka client application development process

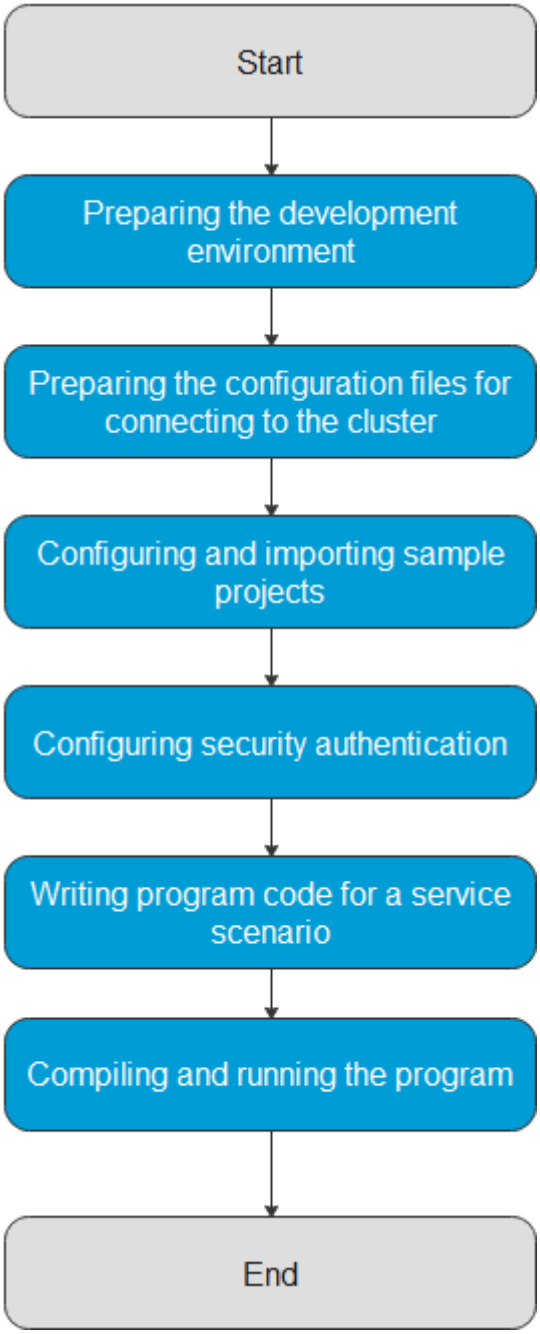


Table 22-1 Kafka client application development process description

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**
Preparing the configuration files for connecting to the cluster	During the development or a test run of the program, you need to use the cluster configuration files to connect to an MRS cluster. The configuration files usually contain the cluster component information file and user files used for security authentication. You can obtain the required information from the created MRS cluster. Nodes used for program debugging or running must be able to communicate with the nodes within the MRS cluster, and the hosts domain name must be configured.	**Preparing the Configuration Files for Connecting to the Cluster**
Configuring and importing sample projects	Kafka provides sample projects in various scenarios. sample projects can be imported for studying.	**Configuring and Importing a Sample Project**
Configuring security authentication	If you are using an MRS cluster with Kerberos authentication enabled, security authentication is required.	**Preparing for Security Authentication**

Phase	Description	Reference Document
Writing program code for a service scenario	Producer and Consumer API usage samples are provided and old APIs, new APIs, and multi-thread usage scenarios are covered, helping you quickly know Kafka APIs well. Compile and run the program. You can debug and run the program in the local Windows development environment, or compile the program into a JAR package and submit it to a Linux node.	**Developing an Application**
Compiling and running the program	This phase provides guidance for users to submit and run a developed program and then view the result.	**Application Commissioning**

22.1.4 Kafka Sample Project

To obtain an MRS sample project, visit **https://github.com/huaweicloud/huaweicloud-mrs-example** and switch to the branch that matches the MRS cluster version. Download the package to the local PC and decompress it to obtain the sample project of each component.

MRS provides the following Kafka sample projects:

Table 22-2 Kafka-related sample projects

Sample Project Location	Description
kafka-examples	1. Data production using a single thread. For details, see **Producer API Sample**. 2. Data consumption using a single thread. For details, see **Consumer API Sample**. 3. Data production using multiple threads. For details, see **Multi-thread Producer Sample**. 4. Data consumption using multiple threads. For details, see **Multi-thread Consumer Sample**. 5. The word counting function is implemented based on KafkaStreams. For details, see **KafkaStreams Sample**.

22.2 Environment Preparation

22.2.1 Preparing for Development and Operating Environment

Preparing Development Environment

Table 22-3 describes the environment required for secondary development.

Table 22-3 Development environment

Item	Description
OS	• 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	It is the basic configuration for the development environment. JDK 1.8 is used. IntelliJ IDEA 2019.1 or other compatible versions are used. **NOTE** • 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.

Item	Description
JDK installation	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. • For x86 nodes that run clients, the following JDKs can be used: – 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: – OpenJDK: 1.8.0_272 **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 com.ibm.jsse2.overrideDefaultTLS 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.**
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.

22.2.2 Preparing the Configuration Files for Connecting to the Cluster

Preparing User Information for Cluster Authentication

For an MRS cluster with Kerberos authentication enabled, prepare a user who has the operation permission on related components for program authentication.

The following Kafka permission configuration example is for reference only. You can modify the configuration as you need.

Step 1 Log in to FusionInsight Manager.

Step 2 Choose **Cluster** > **Services** > **Kafka**. On the displayed page, click **More** > **Enable Ranger** in the upper right corner. Check whether the button is grayed out.

- If it is grayed out, create a user and assign related operation rights to the user in Ranger.

a. Choose **System** > **Permission** > **User**. On the displayed page, click **Create**. On the displayed page, create a machine-machine user, for example, **developuser**.

Add the **kafkaadmin** user group to **User Group**.

b. Log in to the Ranger management page as the Ranger administrator **rangeradmin**.

c. On the home page, click the component plug-in name in the **KAFKA** area, for example, **Kafka**.

d. Click  in the **Action** column of the row containing the **all - topic** policy.

e. In the **Allow Conditions** area, add an allow condition. Select the user created in **Step 2.a** for **Select User**, and select **Select/Deselect All** for **Permissions**.

f. Click **Save**.

- If the button is available, create a user and grant related operation permissions to the user on Manager.

  a. Choose **User** in the navigation pane and click **Create** on the displayed page. Create a machine-machine user, for example, **developuser**. Add this user to the **kafkaadmin** user group.

  b. Click **OK**.

Step 3 Log in to FusionInsight Manager as user **admin** and choose **System** > **Permission** > **User**. In the **Operation** column of **developuser**, choose **More** > **Download Authentication Credential**. Save the file and decompress it to obtain the **user.keytab** and **krb5.conf** files of the user.

----End

Preparing the Configuration Files of the Running Environment

During the development or a test run of the program, you need to use the cluster configuration files to connect to an MRS cluster. The configuration files usually contain the cluster component information file and user files used for security authentication. You can obtain the required information from the created MRS cluster.

Nodes used for program debugging or running must be able to communicate with the nodes within the MRS cluster, and the hosts domain name must be configured.

- Scenario 1: Prepare the configuration files required for debugging in the local Windows development environment.

  a. Download and decompress the client.

     - **Log in to the FusionInsight Manager portal** and choose **Cluster** > **Dashboard** > **More** > **Download Client**. Set **Select Client Type** to **Configuration Files Only** 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. Then, continue to decompress this file.

b. Go to **Kafka\config** in the directory where the client configuration file is decompressed and obtain the Kafka configuration files listed in **Table 22-4**.

Table 22-4 Configuration files

File	Function
client.properties	Kafka client configuration information
consumer.properties	Kafka consumer configuration information
kafkaSecurityMode	whether to enable the security mode for Kafka.
producer.properties	Kafka producer configuration information
server.properties	Kafka server configuration information

c. Copy the **hosts** file content from the decompression directory to the **hosts** file of the local PC.

NOTE

- If you need to debug the application in the local Windows environment, ensure that the local PC can communicate with the hosts listed in the **hosts** file.
- If your PC cannot communicate with the network plane where the MRS cluster is deployed, you can bind an EIP to access the MRS cluster. For details, see **Kafka Access Configuration on Windows Using EIPs**.
- **C:\WINDOWS\system32\drivers\etc\hosts** is an example directory in a Windows environment for storing the local **hosts** file.

22.2.3 Configuring and Importing a Sample Project

Background

Obtain the Kafka development sample project and import the project to IntelliJ IDEA to learn the sample project and develop applications.

Prerequisites

- Ensure that the difference between the local environment 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 FusionInsight Manager.
- You have prepared the development environment and MRS cluster configuration files. For details, see **Preparing the Configuration Files for Connecting to the Cluster**.

Procedure

Step 1 Obtain the sample project folder.

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 Obtain configuration files.

If the Kafka sample code needs to be commissioned on the local Windows PC, place keytab files **user.keytab** and **krb5.conf** obtained in **Preparing User Information for Cluster Authentication** and all configuration files obtained in **Preparing the Configuration Files of the Running Environment** in the **kafka-examples\src\main\resources** directory of the sample project.

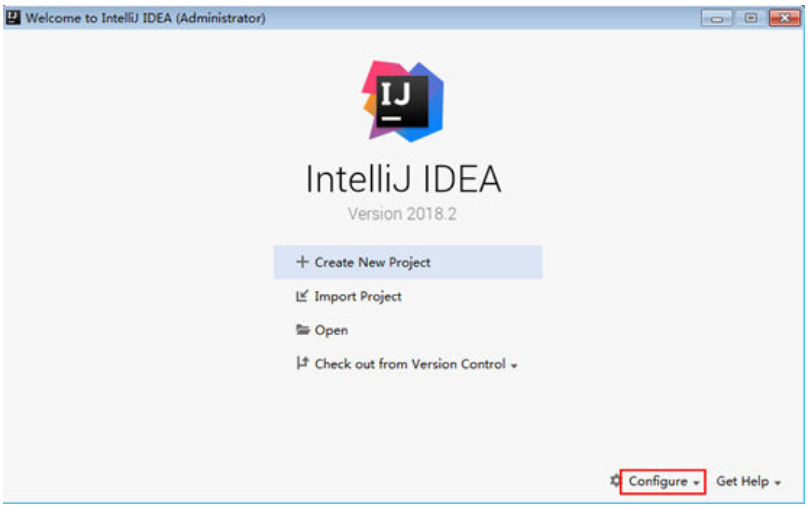
Step 3 Configure the JDK in IntelliJ IDEA after the IntelliJ IDEA and JDK tools are installed.

NOTE

The operation procedure may vary according to the IDEA version.

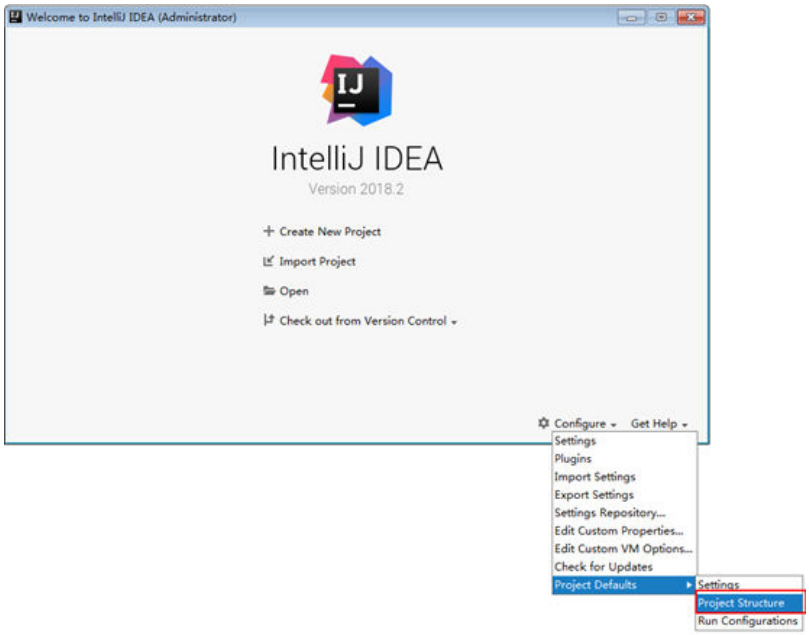
1. Start IntelliJ IDEA and choose **Configure**.

Figure 22-2 Quick Start



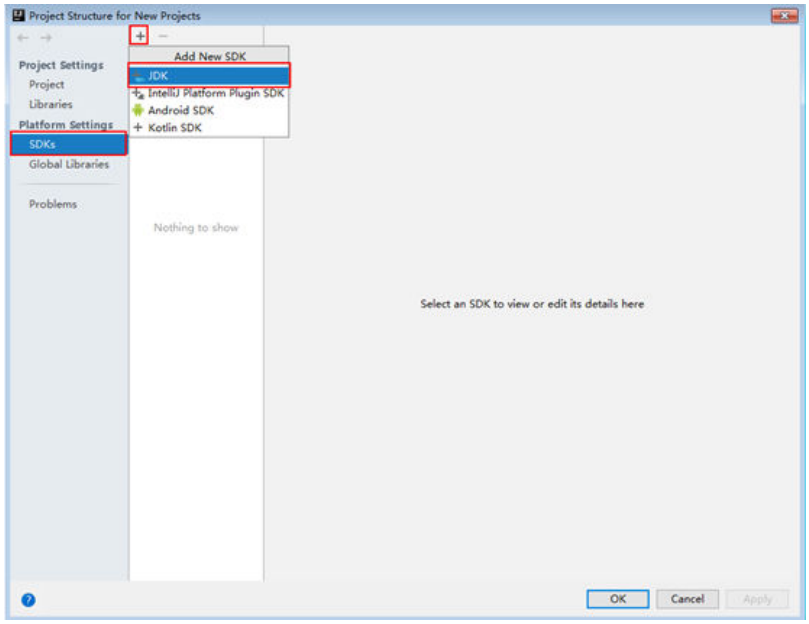
2. Choose **Project Defaults** > **Project Structure** from the drop-down list box.

Figure 22-3 Configure



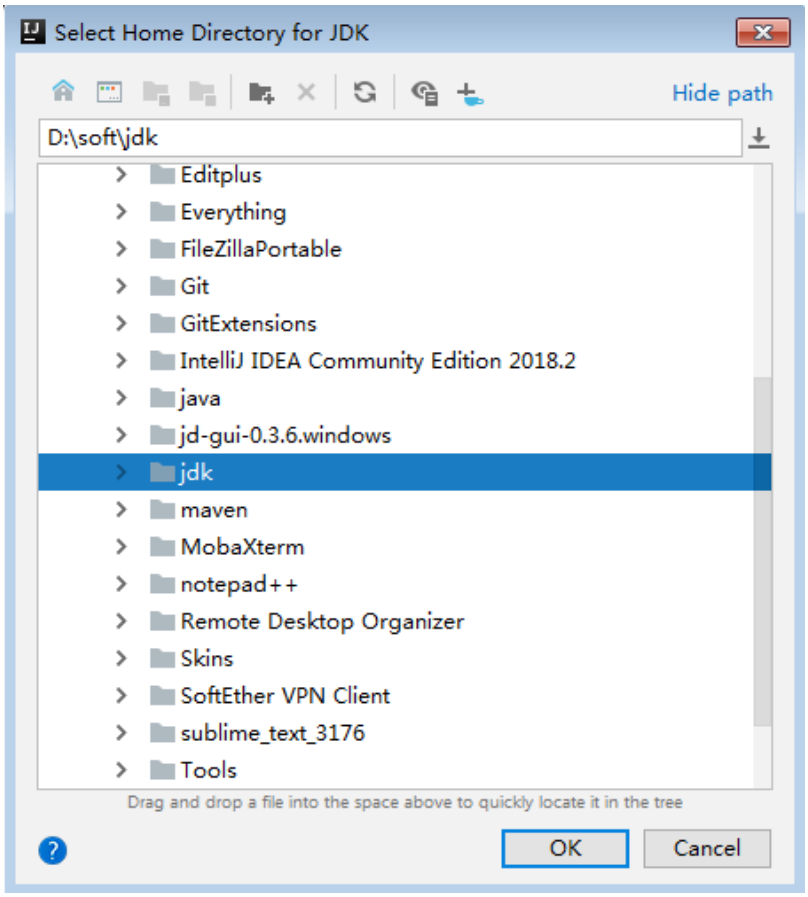
3. On the displayed **Project Structure for New Projects** page, select **SDKs**, click the plus sign (+), and click **JDK**.

Figure 22-4 Project Structure for New Projects



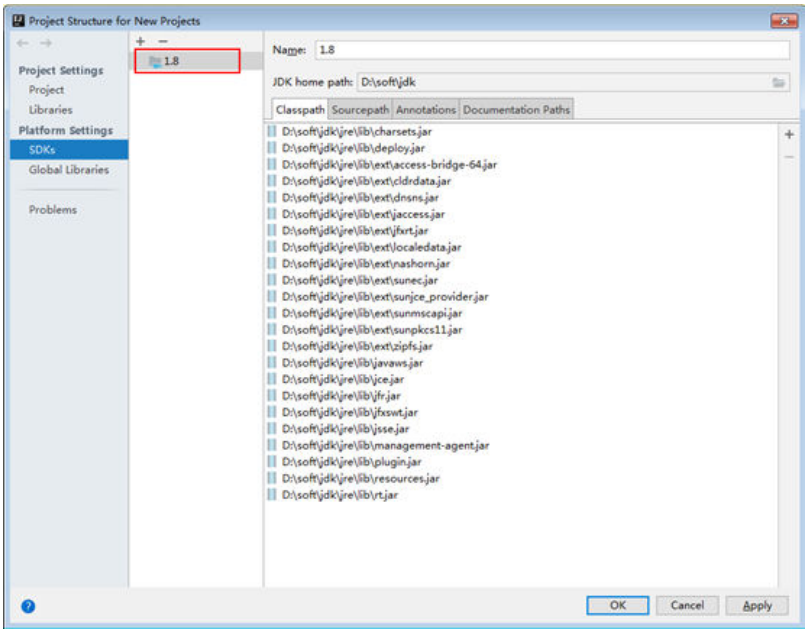
4. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 22-5 Select Home Directory for JDK



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

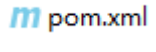
Figure 22-6 Completing the JDK configuration



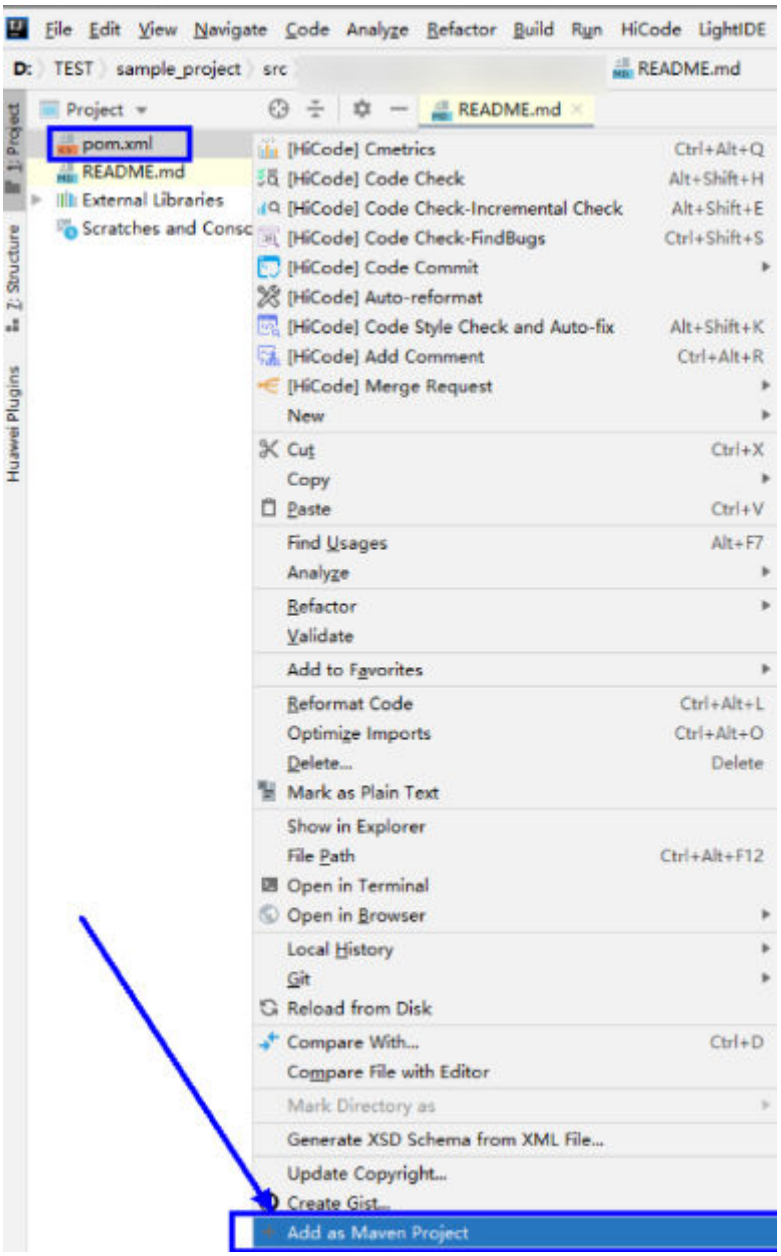
Step 4 Import the sample project to the IntelliJ IDEA development environment.

1. Start IntelliJ IDEA and choose **Open**.

   The **Browse Folder** dialog box is displayed.

2. Select a sample project folder, and click **OK**.
3. After the import is complete, check that the imported sample projects are displayed on the IDEA home page.
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 displayed as 

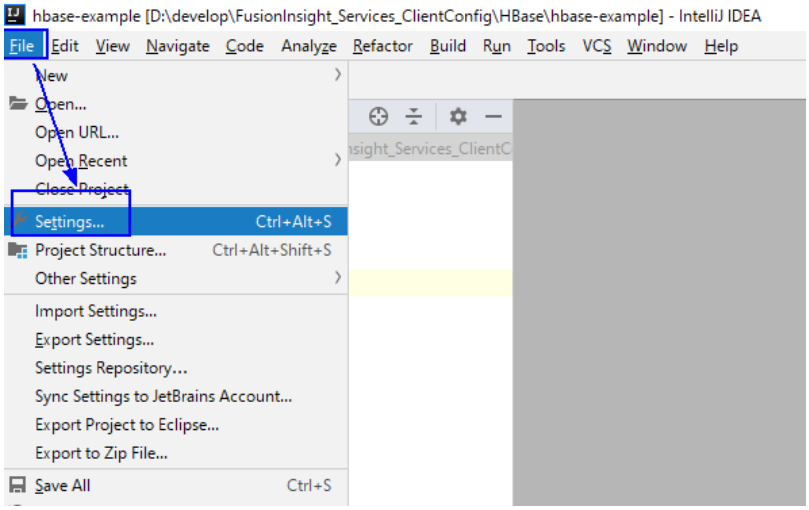
, go to the next step.



Step 5 Set the Maven version used by the project.

1. Choose **File** > **Settings...** from the main menu of IntelliJ IDEA.

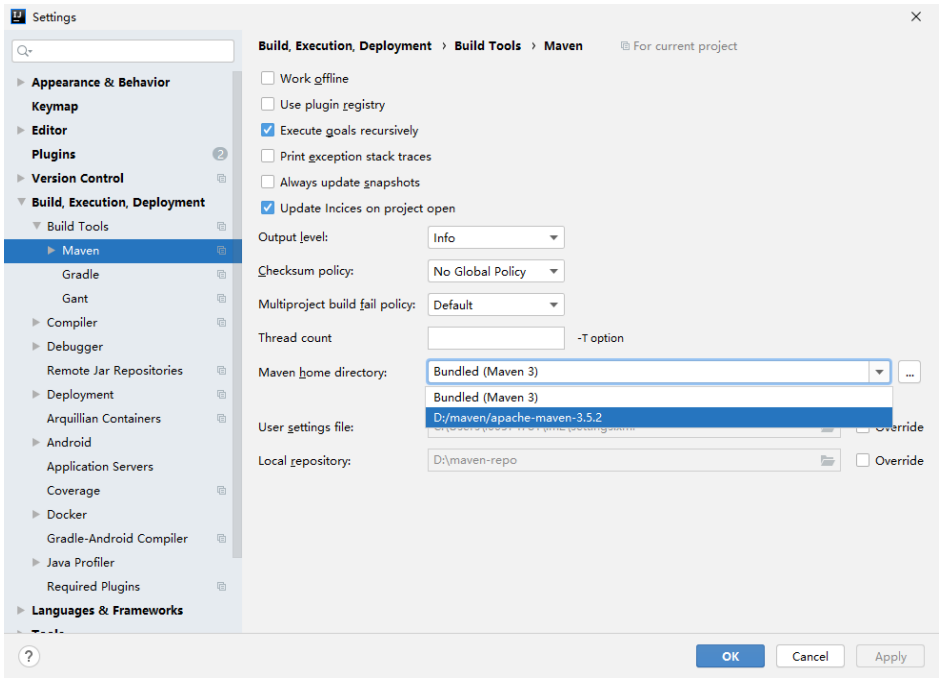
Figure 22-7 Settings



2. Choose **Build, Execution, Deployment** > **Maven** and set **Maven home directory** to the Maven version installed on the local PC.

   Set **User settings file** and **Local repository** as you need, and click **Apply** > **OK**.

Figure 22-8 Selecting the local Maven installation directory



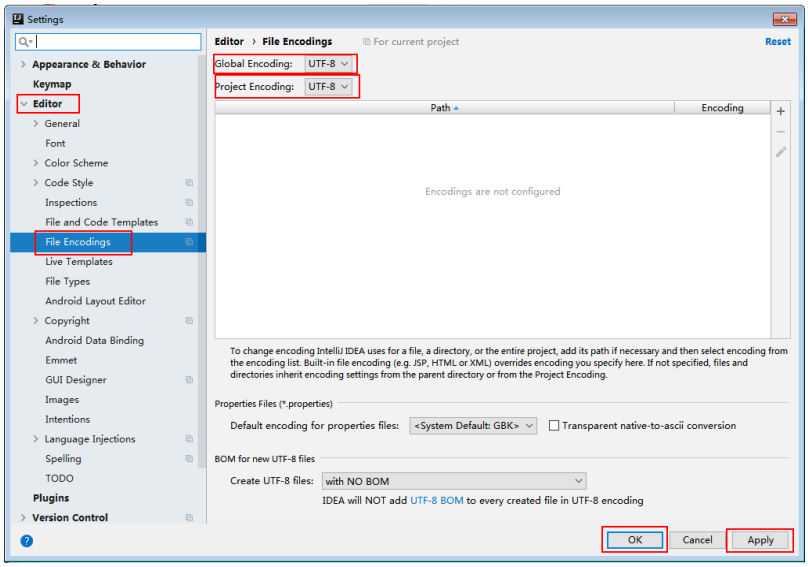
Step 6 Set the IntelliJ IDEA text file coding format to prevent garbled characters.

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

   The **Settings** window is displayed.

2. In the left navigation pane, choose **Editor** > **File Encodings**. In the **Project Encoding** and **Global Encoding** areas, set the parameter to **UTF-8**. Click **Apply** and **OK**, as shown in **Figure 22-9**.

Figure 22-9 Setting the IntelliJ IDEA encoding format



----End

22.2.4 Preparing for Security Authentication

22.2.4.1 SASL Kerberos Authentication

In a cluster with the security mode enabled, the components must be mutually authenticated before communicating with each other to ensure communication security. The Kafka, ZooKeeper, and Kerberos security authentications are required for Kafka application development. However, you only need to generate one JAAS file and configure related environment variables accordingly. LoginUtil related APIs can be used to complete these configurations.

Sample Code

The code snippets are contained in the LoginUtil class of the **com.huawei.bigdata.kafka.example.security** package.

```
   /**
    * keytab file name of the machine-machine account that the user applies for
    */
   private static final String USER_KEYTAB_FILE = "Keytab file name of the machine-machine account that the user applies for, for example, user.keytab";

  /**
   * Machine-machine account that the user applies for
   */
   private static final String USER_PRINCIPAL = "Machine-machine account that the 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 the Windows path.
       userKeyTableFile = userKeyTableFile.replace("\\", "\\\\");
       krbFile = krbFile.replace("\\", "\\\\");
```

```
    LoginUtil.setKrb5Config(krbFile);
    LoginUtil.setZookeeperServerPrincipal("zookeeper/hadoop.<System domain name>");
    LoginUtil.setJaasFile(USER_PRINCIPAL, userKeyTableFile);
}
```

NOTE

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.

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() {
        }
    };
  }
}
```

22.2.4.2 SASL/PLAINTEXT Authentication

Scenario

Kafka supports SASL/PLAINTEXT authentication for clusters with Kerberos authentication enabled.

Step 1 Configure SASL/PLAINTEXT authentication on the Kafka server.

1. Log in to FusionInsight Manager.
2. Choose **Cluster** > **Services** > **Kafka** and choose **Configurations** > **All Configurations**. Search for **sasl.enabled.mechanisms**, change the value to **GSSAPI,PLAIN**, and click **Save**.
3. Click **Dashboard**, click **More**, and select **Restart Service** to make the configuration take effect.

Step 2 Configure SASL/PLAINTEXT authentication on the Kafka client.

You only need to configure dynamic **jaas.conf** and set related authentication attributes on the Kafka client. For details, see the authentication sample code in **Producer** of the **com.huawei.bigdata.kafka.example.security** package.

```
public static Properties initProperties() {
    .............
    props.put("sasl.mechanism", "PLAIN");
    props.put("sasl.jaas.config","org.apache.kafka.common.security.plain.PlainLoginModule required
username=manager_user password=Password;");
}
```

NOTE

- *manager_user* is a human-machine user created on FusionInsight Manager and must have the production and consumption permissions for the topic that is being used.
- *Password* is the password of *manager_user*.
  - If the open-source **kafka-client** JAR package is used, the special characters in the password can only be the dollar sign ($).
  - If the MRS **kafka-client** JAR package is used, the special characters in the password are those supported by FusionInsight Manager (for example, ~`!?,.;-_'(){}[]/<>@#$%^&*+|\=).

----End

22.2.4.3 Kafka Token Authentication

Scenario

The token authentication mechanism is a lightweight authentication mechanism that does not require Kerberos authentication. It can be used in APIs.

Sample Code

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.

- The sample code of **Producer()** is as follows:

```
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;
}
```

- The sample code of **Consumer()** is as follows:

```
public static Properties initProperties() {
    Properties props = new Properties();
    KafkaProperties kafkaProc = KafkaProperties.getInstance();
        // Broker connection address
        props.put(BOOTSTRAP_SERVER, kafkaProc.getValues(BOOTSTRAP_SERVER, "localhost:21007"));
        // Group id
        props.put(GROUP_ID, kafkaProc.getValues(GROUP_ID, "DemoConsumer"));
        // Whether to automatically submit the offset.
        props.put(ENABLE_AUTO_COMMIT, kafkaProc.getValues(ENABLE_AUTO_COMMIT, "true"));
        // Interval for automatically submitting the offset.
        props.put(AUTO_COMMIT_INTERVAL_MS,
kafkaProc.getValues(AUTO_COMMIT_INTERVAL_MS,"1000"));
```



```
        // Session timeout
        props.put(SESSION_TIMEOUT_MS, kafkaProc.getValues(SESSION_TIMEOUT_MS, "30000"));
        // Deserialization class used by the message key value
        props.put(KEY_DESERIALIZER,
            kafkaProc.getValues(KEY_DESERIALIZER,
"org.apache.kafka.common.serialization.StringDeserializer"));
        // Deserialization class used by the message content
        props.put(VALUE_DESERIALIZER,
            kafkaProc.getValues(VALUE_DESERIALIZER,
"org.apache.kafka.common.serialization.StringDeserializer"));
        // Security protocol type
        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"));
    // 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

- Set **BOOTSTRAP_SERVERS** to the host name and port number of the Kafka broker node based on site requirements. You can choose **Cluster** > **Services** > **Kafka** > **Instance** on FusionInsight Manager to view the broker instance information.
- Set **SECURITY_PROTOCOL** to the protocol for connecting to Kafka. In this example, set this parameter to **SASL_PLAINTEXT**.
- **TOKENID** and **HMAC** are generated when you generate tokens. For how to generate a token, see **Kafka Token Authentication Mechanism Tool Usage**
- 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 occured.", 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();
    }
```

22.3 Developing an Application

22.3.1 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.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.

22.3.2 Typical Scenario Sample Code Description

22.3.2.1 Producer API Sample

Function

The following code snippet belongs to the run method in the **com.huawei.bigdata.kafka.example.Producer** class. It is used by the Producer APIs to consume messages in the security topic.

Sample Code

```
/**
* 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 a message record.
        ProducerRecord<Integer, String> record = new ProducerRecord<Integer, String>(topic, messageNo,
messageStr);

        if (isAsync) {
            // Send data asynchronously.
            producer.send(record, new DemoCallBack(startTime, messageNo, messageStr));
        } else {
            try {
                // Send data synchronously.
                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 occured : {}.", ie);
            } catch (ExecutionException ee) {
                LOG.info("The ExecutionException occured : {}.", ee);
            }
        }
        messageNo++;
    }
}
```


22.3.2.2 Consumer API Sample

Function

The following code snippet belongs to the **com.huawei.bigdata.kafka.example.Consumer** class. It is used to enable the Consumer APIs to subscribe to secure topics and consume messages.

Sample Code

```
/**
 * Consumer create a function.
 * @param topic Subscribed topic name
 */

public Consumer(String topic) {
    super("KafkaConsumerExample", false);
    // Initialize 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));
    // Submit the message consumption request.
    ConsumerRecords<Integer, String> records = consumer.poll(waitTime);
    // Process the message.
    for (ConsumerRecord<Integer, String> record : records) {
        LOG.info("[ConsumerExample], Received message: (" + record.key() + ", " + record.value() + ") at
offset " + record.offset());
    }
}
```

22.3.2.3 Multi-thread Producer Sample

Function

The multi-thread producer function is implemented based on the code sample described in **Producer API 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. They are used to enable multiple threads to produce data.

Sample Code

```
/**
 * Specify the key value as the current thread ID 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 a message.
        producer.send(new KeyedMessage<String, String>(sendTopic, key, messageStr));
LOG.info("Producer: send " + messageStr + " to " + sendTopic + " with key: " + key);
        messageCount++;
    }
}
```

22.3.2.4 Multi-thread Consumer Sample

Function

The multi-thread consumer function is implemented based on the code sample described in **Consumer API 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. They are used to implement concurrent consumption of messages in a specified topic.

Sample Code

```
/**
 * Start the multi-thread consumer function.
 */
public void run() {
    LOG.info("Consumer: start.");
    Properties props = Consumer.initProperties();
    // Start a specified number of consumer threads to consume data.
    // Note: If the value of this parameter is greater than the number of partitions of the topic to be
consumed, the excess threads cannot 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;

    /**
     * Constructor of the consumer thread class
     *
     * @param threadNum Thread ID
     * @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());
        }
    }
}
```

22.3.2.5 KafkaStreams Sample

Function

The section describes High Level Kafka Streams API and Low Level Kafka Streams API sample codes. Kafka Streams counts words in each message by reading messages in the input topic and outputs the result in key-value pairs by consuming data in the output topic.

High Level Kafka Streams API Sample Code

Step 1 The following code snippets are in the createWordCountStream method of the **com.huawei.bigdata.kafka.example.WordCountDemo** class.

```
static void createWordCountStream(final StreamsBuilder builder) {
    // Receive input records from the input topic.
    final KStream<String, String> source = builder.stream(INPUT_TOPIC_NAME);

    // Aggregate the calculation results of the key-value pairs.
    final KTable<String, Long> counts = source
            // Process the received records and split them based on the regular expression REGEX_STRING.
            .flatMapValues(value ->
Arrays.asList(value.toLowerCase(Locale.getDefault()).split(REGEX_STRING)))
            // Aggregate the calculation results of the key-value pairs.
            .groupBy((key, value) -> value)
            // Output the final result.
            .count();

    // Output the key-value pairs from the output topic.
    counts.toStream().to(OUTPUT_TOPIC_NAME, Produced.with(Serdes.String(), Serdes.Long()));
}
// keytab file name of the machine-machine account that a user applies for
private static final String USER_KEYTAB_FILE = "Change it to the real-world keytab file name.";
// Machine-machine account that a user applies for
private static final String USER_PRINCIPAL = "Change it to the real-world username."
```

----End

Low Level Kafka Streams API Sample Code

Step 1 The following code snippets are in the **com.huawei.bigdata.kafka.example.WordCountProcessorDemo** class.

```
private static class MyProcessorSupplier implements ProcessorSupplier<String, String> {
    @Override
    public Processor<String, String> get() {
        return new Processor<String, String>() {
            // ProcessorContext instance, which provides access to the metadata of the record being processed.
            private ProcessorContext context;
            private KeyValueStore<String, Integer> kvStore;

            @Override
            @SuppressWarnings("unchecked")
```



```
        public void init(ProcessorContext context) {
            // Retain the processor context on the local PC because it will be used in punctuate() and
commit().
            this.context = context;
            // Execute punctuate() 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 record as a key-value pair to the downstream processor.
                        context.forward(entry.key, entry.value.toString());
                    }
                }
            });
            // Search for the key-value state storage zone KEY_VALUE_STATE_STORE_NAME, which can be
used to memorize the recently received input records.
            this.kvStore = (KeyValueStore<String, Integer>)
context.getStateStore(KEY_VALUE_STATE_STORE_NAME);
        }

        // Process the received records of the 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() {
        }
    };
  }
}

// keytab file name of the machine-machine account that a user applies for
private static final String USER_KEYTAB_FILE = "Change it to the real-world keytab file name.";
// Machine-machine account that a user applies for
private static final String USER_PRINCIPAL = "Change it to the real-world username."
```

----End

22.4 Application Commissioning

22.4.1 Producer Sample Commissioning

Prerequisites

- Ensure that Windows is configured to allow Kafka access through an EIP if you need to debug applications on Windows. For details, see **Kafka Access Configuration on Windows Using EIPs**.
- Ensure that the current user has the read permission on all files in the **src/main/resources** directory and the dependent library file directory if you need

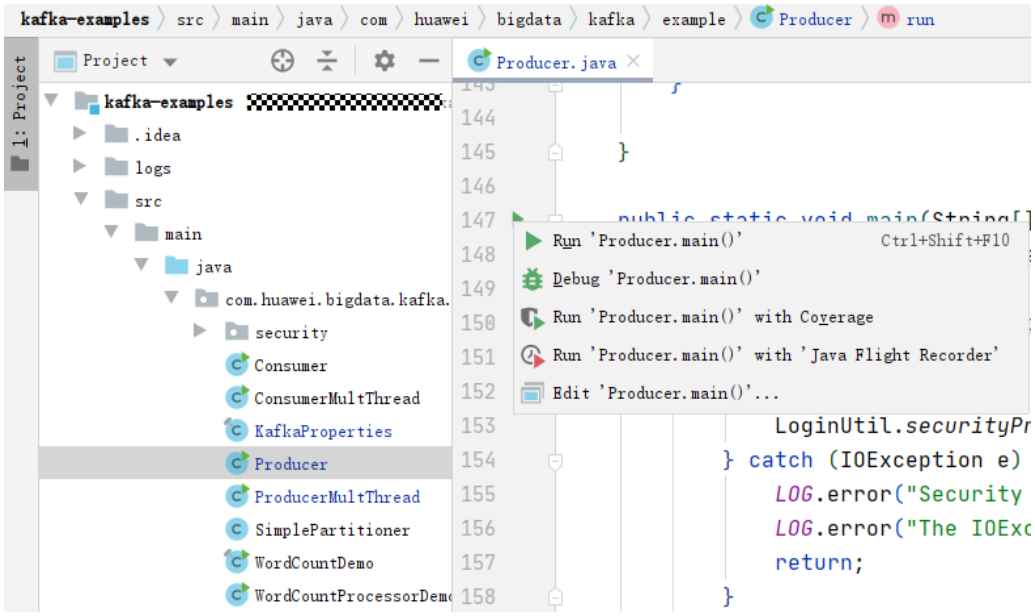
to debug applications on Linux. Ensure that the JDK has been installed and Java environment variables have been set.

Commissioning Applications on Windows

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.

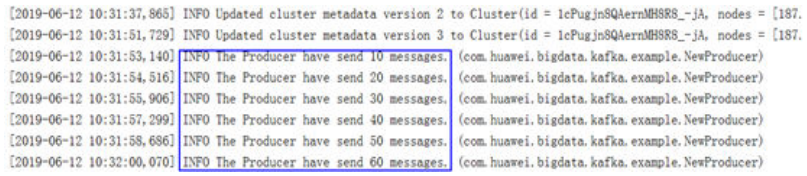
Step 2 Run Producer.java on IntelliJ IDEA, as shown in **Figure 22-10**.

Figure 22-10 Running Producer.java.



Step 3 Check the displayed console window and 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 22-11 Producer running window



----End

Commissioning Applications on Linux

Step 1 Export a JAR package.

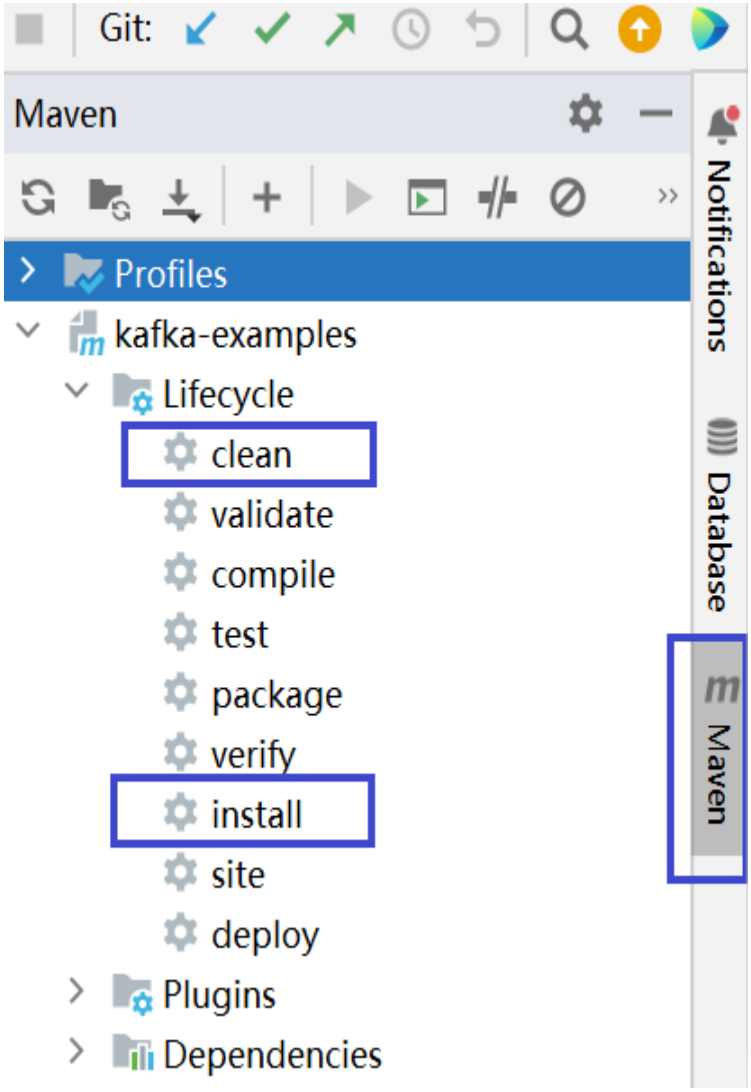
You can build a JAR file in either of the following ways:

- Method 1:

  Choose **Maven** > *Sample projectname* > **Lifecycle** > **clean**, double click **clean** to run the **clean** command of Maven.

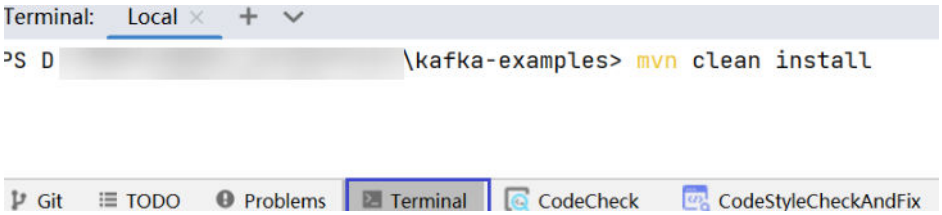
Choose **Maven** > *Sample project name* > **Lifecycle** > **install**, double click **install** to run the **install** command of Maven.

Figure 22-12 Maven tools: **clean** and **installl**



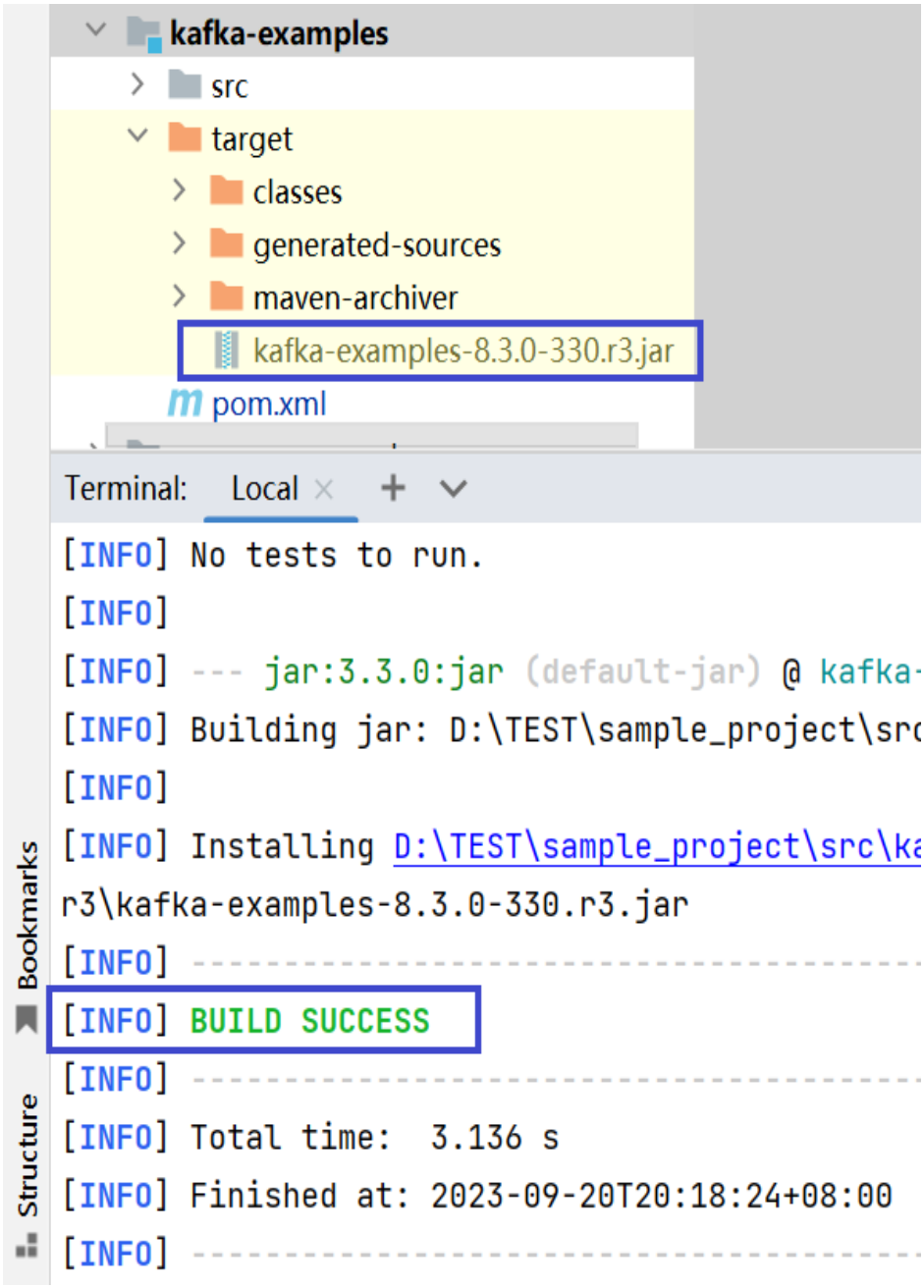
- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the **pom.xml** file.

Figure 22-13 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 22-14 When the compilation is completed, the JAR file is generated.



Step 2 Copy the JAR file generated during project compilation to the **/opt/hadoopclient/lib** directory.

Step 3 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 4 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

22.4.2 Consumer Sample Commissioning

Prerequisites

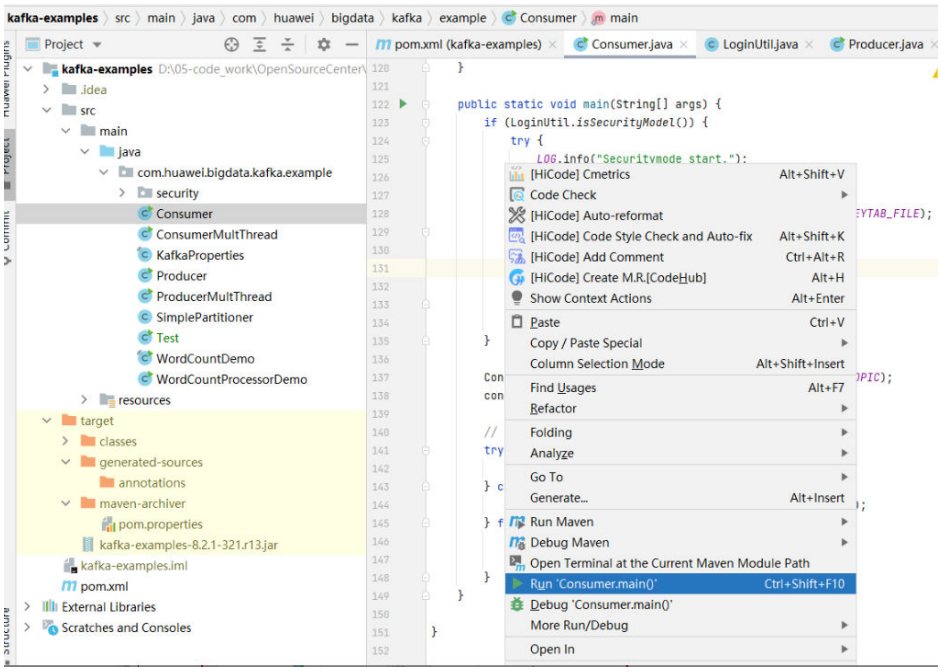
- Ensure that Windows is configured to allow Kafka access through an EIP if you need to debug applications on Windows. For details, see **Kafka Access Configuration on Windows Using EIPs**.
- Ensure that the current user has the read permission on all files in the **src/main/resources** directory and the dependent library file directory if you need to debug applications on Linux. Ensure that the JDK has been installed and Java environment variables have been set.

Commissioning Applications on Windows

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 Consumer.java on IntelliJ IDEA, as shown in **Figure 22-15**.

Figure 22-15 Running Consumer.java



Step 3 Click **Run**. In the displayed console window, you can see that Producer starts after Consumer has started and you can view the messages received in real time.

Figure 22-16 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.inter
[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: (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

Commissioning Applications on Linux

Step 1 Compile and generate a JAR package, and copy the JAR package to the **src/main/resources** directory at the same level as the dependent library folder. For details, see **Commissioning Applications on Linux**.

Step 2 Run the following command to run the consumer sample project:

java -cp /opt/client/lib/*:/opt/client/src/main/resources com.huawei.bigdata.kafka.example.Consumer

----End

22.4.3 High Level Streams Sample Commissioning

Commissioning Applications on Windows

For how to debug applications on Windows, see **Commissioning Applications on Windows**.

Commissioning Applications on Linux

Step 1 Compile and generate a JAR package, and copy the JAR package to the **src/main/resources** directory at the same level as the dependent library folder. For details, see **Commissioning Applications on Linux**.

Step 2 Log in to the cluster client node as the cluster installation user.

cd /opt/client

source bigdata_env

kinit *Component operation user (for example, developuser)*

Step 3 Create an input topic and an output topic. Ensure that the topic names are the same as those specified in the sample code. Set the cleanup policy of the output topic to **compact**.

kafka-topics.sh --create --zookeeper *IP address of the quorumpeer instance:ZooKeeper client connection port*/**kafka --replication-factor 1 --partitions 1 --topic** *Topic name*

To query the IP address of the quorumpeer instance, log in to FusionInsight Manager of the cluster, choose **Cluster** > **Services** > **ZooKeeper**, and click the **Instance** tab. Use commas (,) to separate multiple IP addresses. You can obtain the ZooKeeper client connection port by querying the ZooKeeper configuration parameter **clientPort**. The default value is **2181**.

Run the following commands:

kafka-topics.sh --create --zookeeper 192.168.0.17:2181/kafka --replication-factor 1 --partitions 1 --topic streams-wordcount-input

kafka-topics.sh --create --zookeeper 192.168.0.17:2181/kafka --replication-factor 1 --partitions 1 --topic streams-wordcount-output --config cleanup.policy=compact

Step 4 Run the following command to run the application after the topics are created:

java -cp /opt/client/lib/*:/opt/client/src/main/resources com.huawei.bigdata.kafka.example.WordCountDemo

Step 5 Open a new client window and run the following commands to use **kafka-console-producer.sh** to write messages to the input topic:

cd /opt/client

source bigdata_env

kinit *Component operation user (for example, developuser)*

kafka-console-producer.sh --broker-list *IP address of the broker instance:Kafka connection port(for example, 192.168.0.13:9092)* **--topic streams-wordcount-input --producer.config /opt/client/Kafka/kafka/config/producer.properties**

Step 6 Open a new client window and run the following commands to use **kafka-console-consumer.sh** to consume data from the output topic and view the result:

cd /opt/client

source bigdata_env

kinit *Component operation user (for example, developuser)*

kafka-console-consumer.sh --topic streams-wordcount-output --bootstrap-server *IP address of the broker instance:Kafka connection port* **--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**

Write a message to the input topic.

```
>This is Kafka Streams test
>test starting
>now Kafka Streams is running
>test end
```

The output is as follows:

```
this    1
is      1
kafka   1
streams 1
test    1
test    2
starting 1
now     1
kafka   2
streams 2
is      2
running 1
test    3
end     1
```

----End

22.4.4 Low level Streams API Sample Usage Guide

Commissioning Applications on Windows

For how to debug applications on Windows, see **Commissioning Applications on Windows**.

Commissioning Applications on Linux

Step 1 Compile and generate a JAR package, and copy the JAR package to the **src/main/resources** directory at the same level as the dependent library folder. For details, see **Commissioning Applications on Linux**.

Step 2 Log in to the node where the cluster client is installed as user **root**.

cd /opt/client

source bigdata_env

kinit *Component operation user (for example, developuser)*

Step 3 Create an input topic and an output topic. Ensure that the topic names are the same as those specified in the sample code. Set the cleanup policy of the output topic to **compact**.

kafka-topics.sh --create --zookeeper *IP address of the quorumpeer instance:ZooKeeper client connection port*/**kafka --replication-factor 1 --partitions 1 --topic** *Topic name*

To query the IP address of the quorumpeer instance, log in to FusionInsight Manager of the cluster, choose **Cluster** > **Services** > **ZooKeeper**, and click the **Instance** tab. Use commas (,) to separate multiple IP addresses. You can obtain the ZooKeeper client connection port by querying the ZooKeeper configuration parameter **clientPort**. The default value is **2181**.

Run the following commands:

kafka-topics.sh --create --zookeeper 192.168.0.17:2181/kafka --replication-factor 1 --partitions 1 --topic streams-wordcount-processor-input

kafka-topics.sh --create --zookeeper 192.168.0.17:2181/kafka --replication-factor 1 --partitions 1 --topic streams-wordcount-processor-output --config cleanup.policy=compact

Step 4 Run the following command to run the application after the topics are created:

java -cp /opt/client/lib/*:/opt/client/src/main/resources com.huawei.bigdata.kafka.example.WordCountDemo

Step 5 Open a new client window and run the following commands to use **kafka-console-producer.sh** to write messages to the input topic:

cd /opt/client

source bigdata_env

kinit *Component operation user (for example, developuser)*

kafka-console-producer.sh --broker-list *IP address of the broker instance:Kafka connection port(for example, 192.168.0.13:9092)* **--topic streams-wordcount-**

processor-input --producer.config /opt/client/Kafka/kafka/config/producer.properties

Step 6 Open a new client window and run the following commands to use **kafka-console-consumer.sh** to consume data from the output topic and view the result:

cd /opt/client

source bigdata_env

kinit *Component operation user (for example, developuser)*

kafka-console-consumer.sh --topic streams-wordcount-processor-output --bootstrap-server *IP address of the broker instance:Kafka connection port* **--consumer.config /opt/client/Kafka/kafka/config/consumer.properties --from-beginning --property print.key=true --property print.value=true**

Write a message to the input topic.

```
>This is Kafka Streams test
>test starting
>now Kafka Streams is running
>test end
```

The output is as follows:

```
this    1
is      1
kafka   1
streams 1
test    1
test    2
starting 1
now     1
kafka   2
streams 2
is      2
running 1
test    3
end     1
```

----End

22.4.5 Sample Code Running Guide for the Kafka Token Authentication Mechanism

Procedure

Step 1 Configure Kafka Token Authentication on the Kafka server.

1. Log in to FusionInsight Manager and choose **Cluster** > **Services** > **Kafka**. Click **Configurations**.
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.

NOTE

The value of **delegation.token.master.key** can be customized, for example, **Tokentest**.

3. Specify the SASL authentication mechanism used for the 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.

4. Log in to a component using Scram.

   Find the custom 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;**.

5. Log in to FusionInsight Manager and restart all Broker instances of the Kafka service.

Step 2 Configure Kafka Token Authentication on the Kafka client.

Generate a token for the user. For details, see **Kafka Token Authentication Mechanism Tool Usage**.

Step 3 Configure the secondary development sample project.

Configure the required sample codes in **Producer()** and **Consumer()** of the secondary development sample project. For details about the required sample codes, see **Kafka Token Authentication**.

Step 4 Run the sample code.

- For how to debug applications on Windows, see **Commissioning Applications on Windows**.
- For how to debug applications on Linux, see **Commissioning Applications on Linux**.

----End

22.5 More Information

22.5.1 External Interfaces

22.5.1.1 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

22.5.1.2 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 22-5 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.
value.serializer	Serialization of messages	This parameter specifies how to serialize transmitted messages.

Table 22-6 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 22-7 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 **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.

Parameter	Description	Remarks
value.deserializer	Deserialization of messages	This parameter specifies how to deserialize received messages.

Table 22-8 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.

22.5.1.3 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.

22.5.1.4 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", "XXXXX");
```

  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**.

22.5.2 Kafka Access Configuration on Windows Using EIPs

Scenario

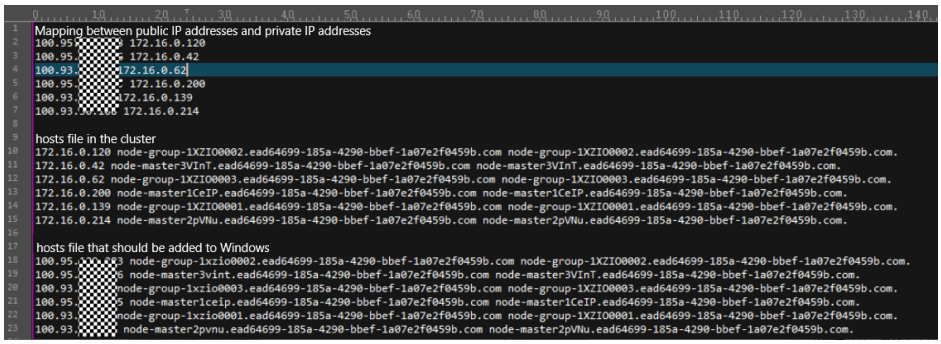
This section describes how to bind Elastic IP addresses (EIPs) to a cluster and configure Kafka files so that sample files can be compiled locally.

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.

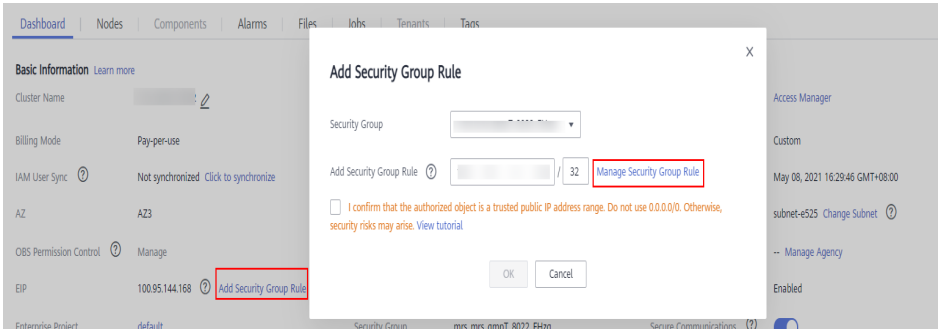
   For details, see **Virtual Private Cloud** > **User Guide** > **EIP** > **Assigning an EIP and Binding It to an ECS**.
2. Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.



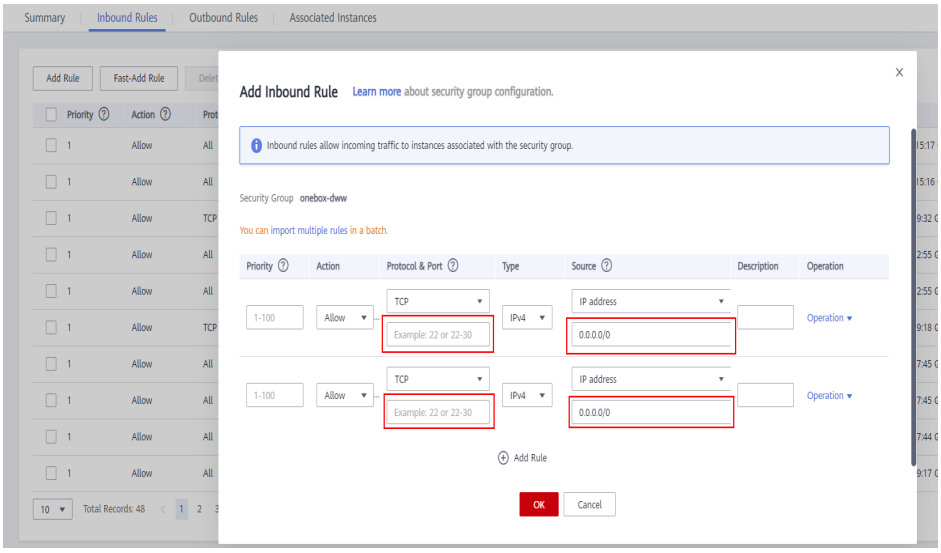
Step 2 Change the IP addresses in the **krb5.conf** file to the corresponding host names.

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 21007,21730TCP, 21731TCP/UDP, and 21732TCP/UDP.



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.

NOTE

If the current cluster is MRS 3.2.0-LTS.1 and you cannot access Kafka through the EIP after performing this step, perform the following operations:

1. Log in to FusionInsight Manager, choose **Cluster > Services > Kafka > Instances**, select all Broker instances, and choose **More > Stop Instance** to verify the administrator password and stop all Broker instances. (This operation affects services. Perform this operation during off-peak hours.)
2. Log in to the Broker node as the **root** user and modify the **server.properties** file.

   vi ${BIGDATA_HOME}/FusionInsight_HD_*/*_*_Broker/etc/server.properties

   Change **host.name** to the host name of the current Broker node, and change the values of **listeners** and **advertised.listeners** to **EXTERNAL_PLAINTEXT://***{Host name}:{port}*.
3. Log in to FusionInsight Manager, choose **Cluster > Services > Kafka > Instances**, select all Broker instances, and click **Start Instance**.
4. Bind an EIP to each Broker node in the MRS cluster.
5. On Windows, use the configured EIP and port of the Broker node to connect to the Kafka cluster and debug the code.

Step 5 Before running the sample code, change the Kafka connection string in the sample code to *hostname1*:**21007,** *hostname2*:**21007,** *hostname3*:**21007**,change the domain name in the code, and change the machine-machine account name and keytab file name applied by the user.

NOTE

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.

----End

22.5.3 FAQ

22.5.3.1 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

22.5.3.2 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

23 Kafka Development Guide (Normal Mode)

23.1 Overview

23.1.1 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**.

23.1.2 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.

23.1.3 Development Process

Kafka client roles include Producer and Consumer, which share the same application development process.

Figure 23-1 and Table 23-1 show each phase of the development process.

Figure 23-1 Kafka client application development process

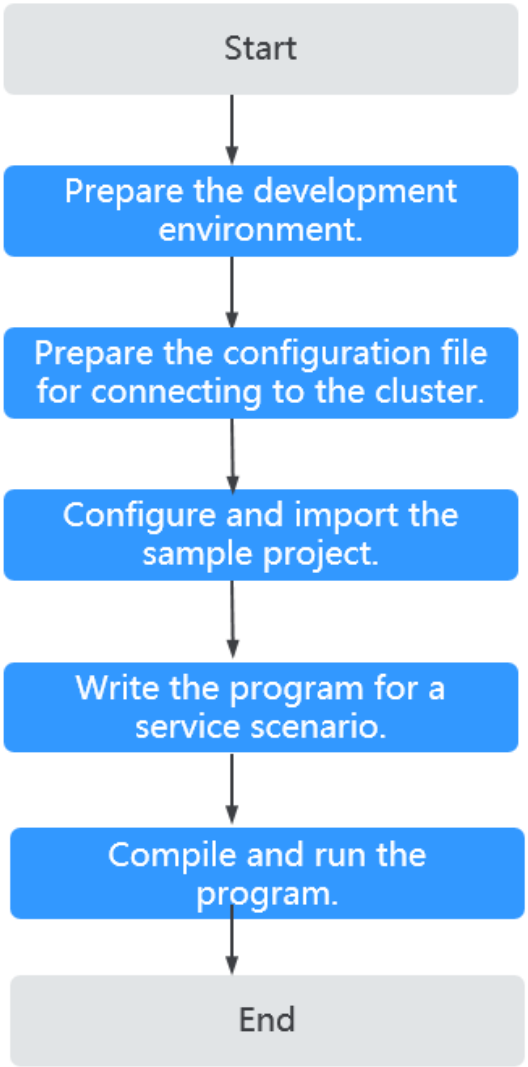


Table 23-1 Kafka client application development process description

Phase	Description	Reference Document
Prepare the development 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.	**Common Concepts**
Preparing the configuration files for connecting to the cluster	During the development or a test run of the program, you need to use the cluster configuration files to connect to an MRS cluster. The configuration files usually contain the cluster component information file. You can obtain the required information from the created MRS cluster. Nodes used for program debugging or running must be able to communicate with the nodes within the MRS cluster, and the hosts domain name must be configured.	**Preparing the Configuration Files for Connecting to the Cluster**
Configuring and importing sample projects	Kafka provides program samples for different scenarios. You can import the program samples for learning.	**Configuring and Importing Sample Projects**

Phase	Description	Reference Document
Writing program code for a service scenario	Producer and Consumer API usage samples are provided and old APIs, new APIs, and multi-thread usage scenarios are covered, helping you quickly know Kafka APIs well. Compile and run the program. You can debug and run the program in the local Windows development environment, or compile the program into a JAR package and submit it to a Linux node.	**Developing an Application**
Compiling and running the program	This phase provides guidance for users to submit and run a developed program and then view the result.	**Application Commissioning**

23.1.4 Kafka Sample Project

To obtain an MRS sample project, visit **https://github.com/huaweicloud/huaweicloud-mrs-example** and switch to the branch that matches the MRS cluster version. Download the package to the local PC and decompress it to obtain the sample project of each component.

MRS provides the following Kafka sample projects:

Table 23-2 Kafka-related sample projects

Sample Project Location	Description
kafka-examples	1. Data production using a single thread. For details, see **Producer API Usage Sample**. 2. Data consumption using a single thread. For details, see **Consumer API Usage Sample**. 3. Data production using multiple threads. For details, see **Multi-thread Producer Sample**. 4. Data consumption using multiple threads. For details, see **Multi-thread Consumer Sample**. 5. The word counting function is implemented based on KafkaStreams. For details, see **KafkaStreams Sample**.

23.2 Environment Preparation

23.2.1 Preparing for Development Environment

Preparing Development Environment

Table 23-3 describes the environment required for secondary development.

Table 23-3 Development environment

Item	Description
OS	• 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	It is the basic configuration for the development environment. JDK 1.8 is used. IntelliJ IDEA 2019.1 or other compatible versions are used. **NOTE** • 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.

Item	Description
JDK installation	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. • For x86 nodes that run clients, the following JDKs can be used: – 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: – OpenJDK: 1.8.0_272 **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 com.ibm.jsse2.overrideDefaultTLS 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.**
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.

23.2.2 Preparing the Configuration Files for Connecting to the Cluster

Preparing the Configuration Files of the Running Environment

During the development or a test run of the program, you need to use the cluster configuration files to connect to an MRS cluster. The configuration files usually contain the cluster component information file. You can obtain the required information from the created MRS cluster.

Nodes used for program debugging or running must be able to communicate with the nodes within the MRS cluster, and the hosts domain name must be configured.

- Scenario 1: Prepare the configuration files required for debugging in the local Windows development environment.
  a. Log in to FusionInsight Manager and choose **Cluster** > **Dashboard** > **More** > **Download Client**. In the dialog box that is displayed, set **Select Client Type** to **Configuration Files Only** and click **OK**. After the client

package is generated, download the package 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**. Then, continue to decompress this file.

b. Go to **Kafka\config** in the directory where the client configuration file is decompressed and obtain the Kafka configuration files listed in **Table 23-4**.

Table 23-4 Configuration files

File	Function
client.properties	Kafka client configuration information
consumer.properties	Kafka consumer configuration information
producer.properties	Kafka producer configuration information
server.properties	Kafka server configuration information

c. Copy the **hosts** file content from the decompression directory to the **hosts** file of the local PC.

NOTE

- If you need to debug the application in the local Windows environment, ensure that the local PC can communicate with the hosts listed in the **hosts** file.
- If your PC cannot communicate with the network plane where the MRS cluster is deployed, you can bind an EIP to access the MRS cluster. For details, see **Kafka Access Configuration on Windows Using EIPs**.
- **C:\WINDOWS\system32\drivers\etc\hosts** is an example directory in a Windows environment for storing the local **hosts** file.

- Scenario 2: Prepare the configuration files required for running the program in a Linux environment.

  a. Install the MRS cluster client on the node.

  For example, the client installation directory can be **/opt/client**.

  b. Obtain the configuration files.

  i. Log in to FusionInsight Manager, click **More** and select **Download Client** in the upper right corner of **Homepage**. Set **Select Client Type** to **Configuration Files Only**, select **Save to Path**, and click **OK** to download the client configuration file to the active OMS node of the cluster.

  ii. Log in to the active OMS node as user **root**, go to the directory where the client configuration file is stored (**/tmp/FusionInsight-Client/** by default), decompress the software package, and obtain

the configuration files listed in **Table 23-4** from the **Kafka/config** 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 OMS node, run the following commands:

cd /tmp/FusionInsight-Client

tar -xvf FusionInsight_Cluster_1_Services_Client.tar

tar -xvf FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar

cd FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles

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 of the **hosts** file in the decompression directory to the **hosts** file on the node where the client is located, to ensure that the local host can communicate with each host in the cluster.

23.2.3 Configuring and Importing Sample Projects

Step 1 Obtain the sample project folder.

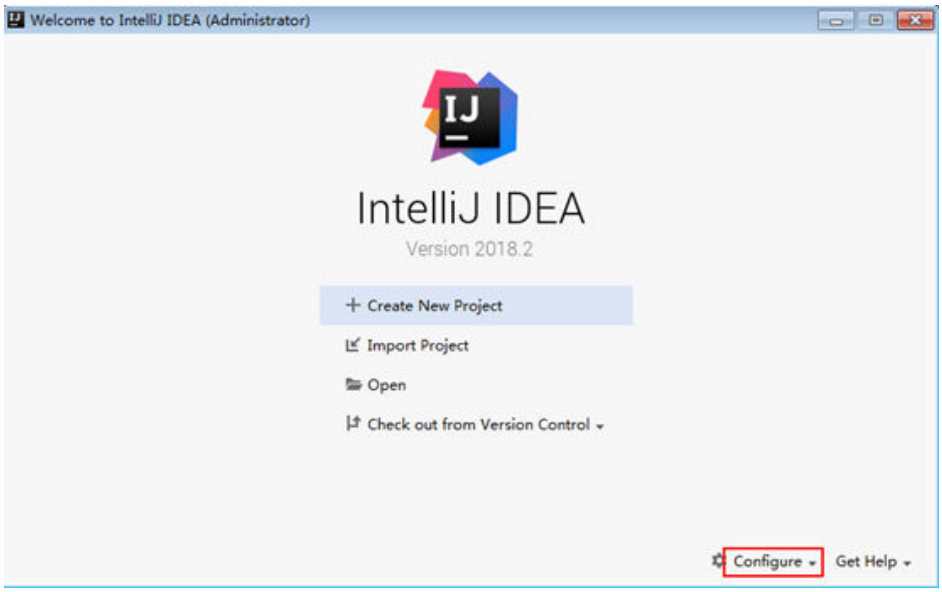
Obtain the sample project folder **kafka-examples** in the **src** directory where the sample code is decompressed. For details, see **Obtaining Sample Projects from Huawei Mirrors**.

Step 2 Copy the all the cluster configuration file obtained in section **Preparing the Configuration Files for Connecting to the Cluster** 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.

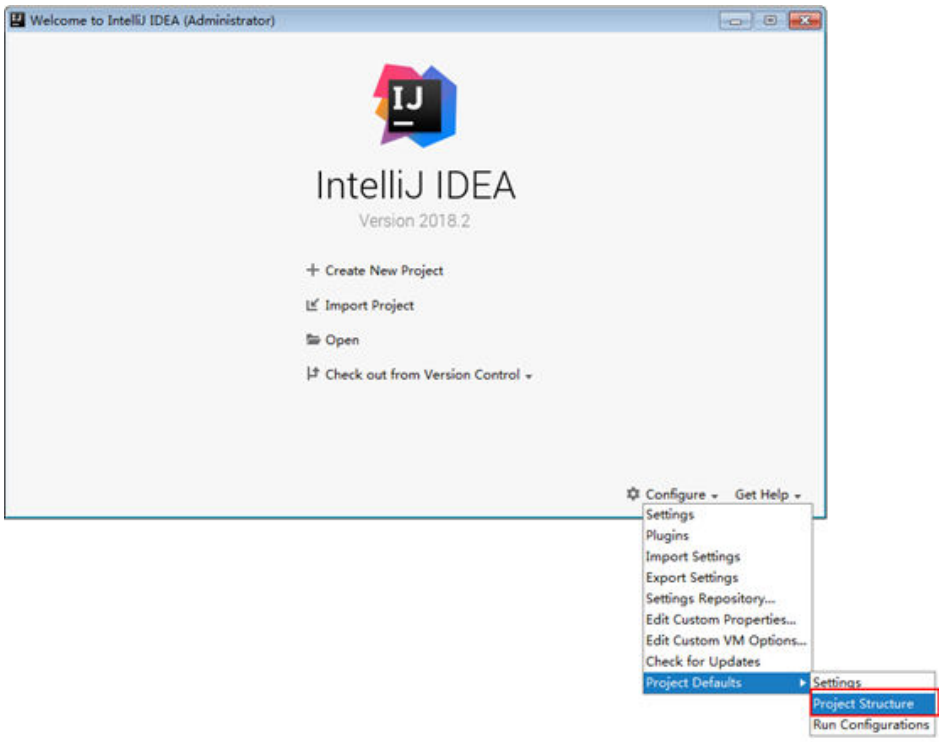
1. Open IntelliJ IDEA and click **Configure**.

Figure 23-2 Quick Start



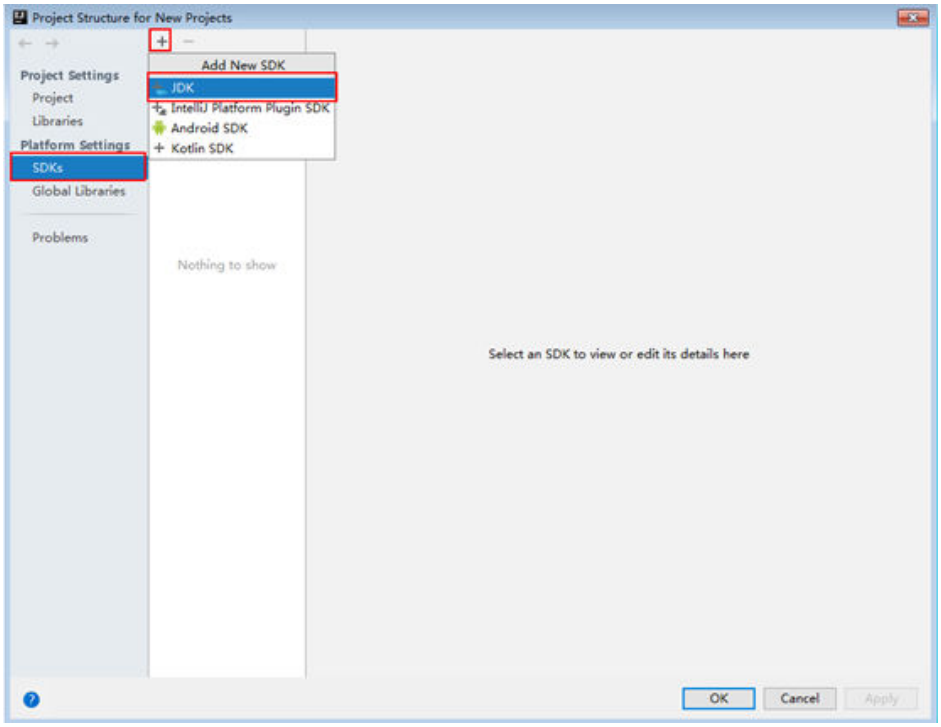
2. Choose **Project Defaults** > **Project Structure**.

Figure 23-3 Configure



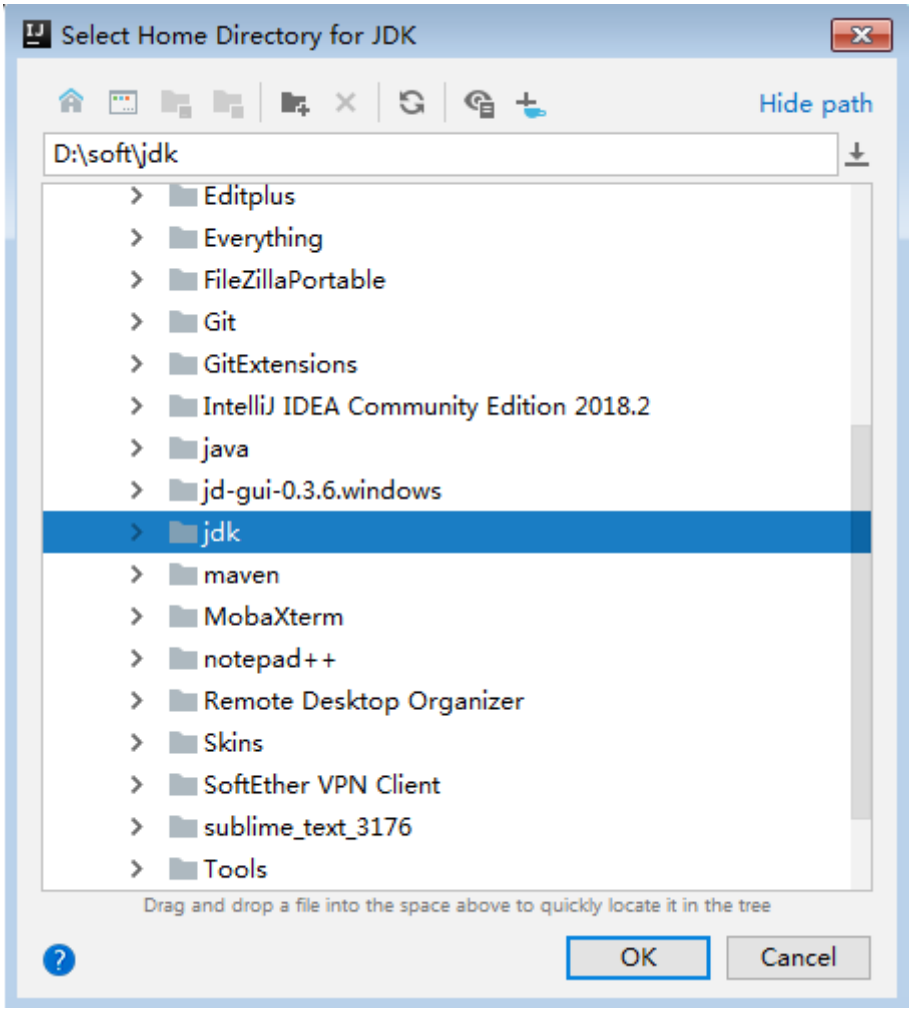
3. In the displayed **Project Structure for New Projects** interface, click **SDKs**. Then click the plus sign (+) and choose **JDK**.

Figure 23-4 Project Structure for New Projects



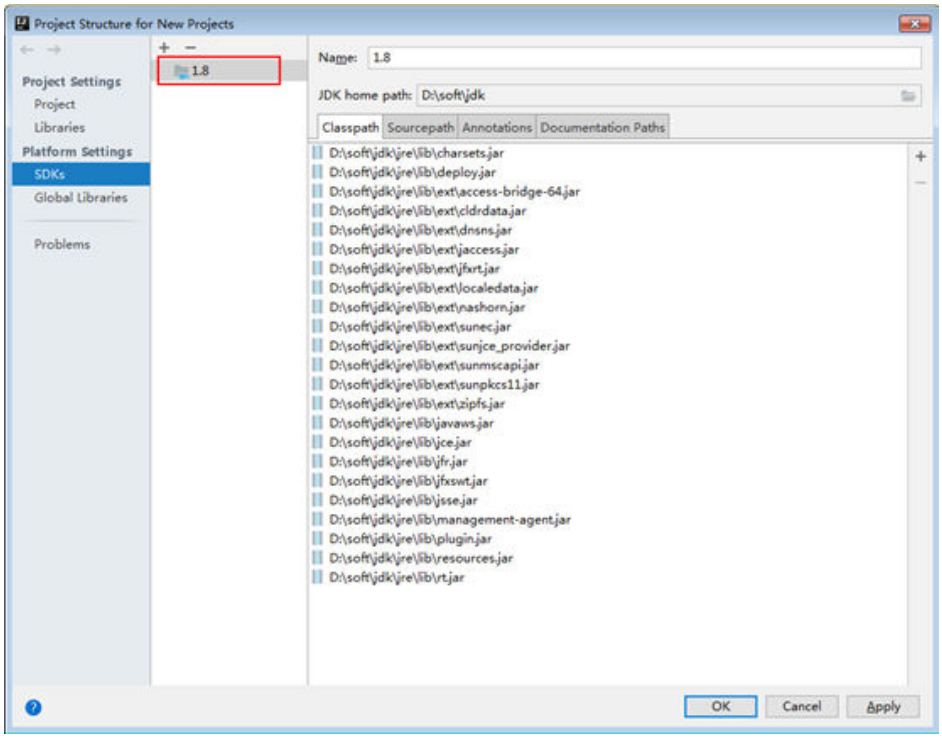
4. In the displayed **Select Home Directory for JDK** interface, select the JDK directory, and click **OK**.

Figure 23-5 Select Home Directory for JDK



5. Click **OK**.

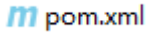
Figure 23-6 Completing the JDK configuration

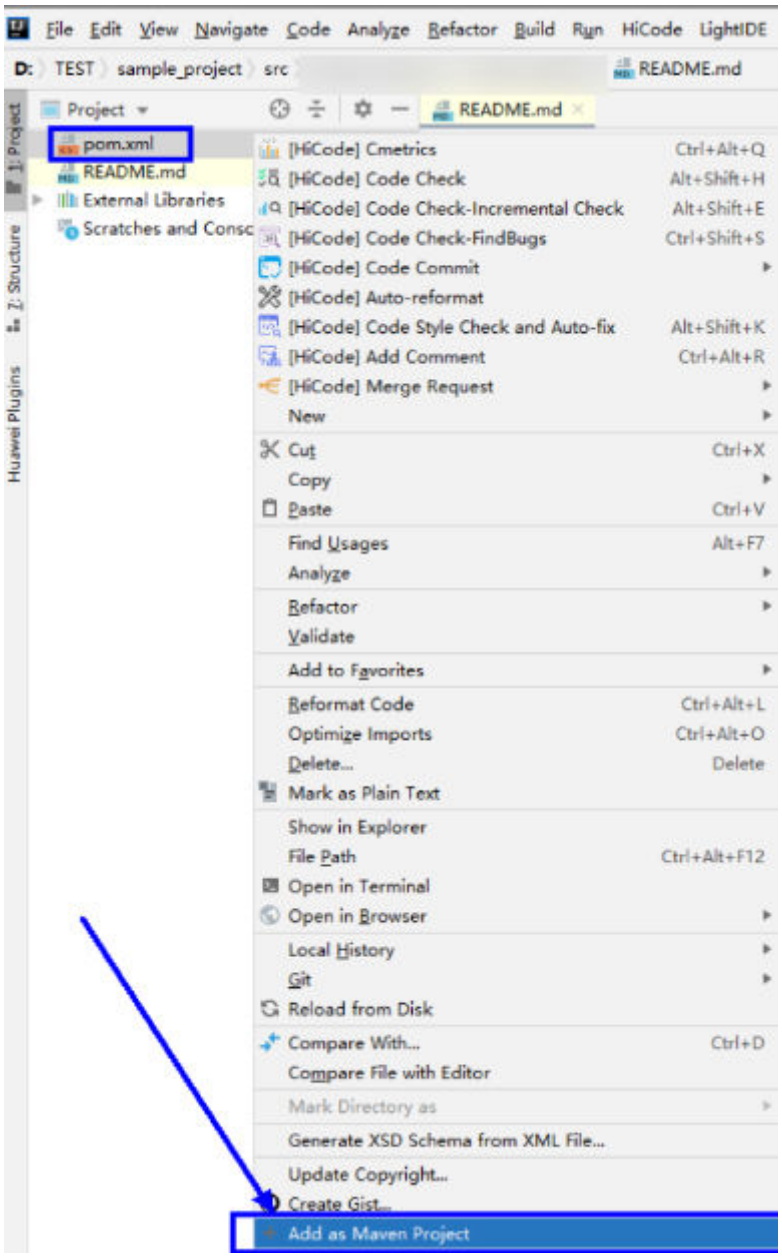


Step 4 Import the sample project to the IntelliJ IDEA development environment.

1. Start IntelliJ IDEA and choose **Open**.

   The **Browse Folder** dialog box is displayed.

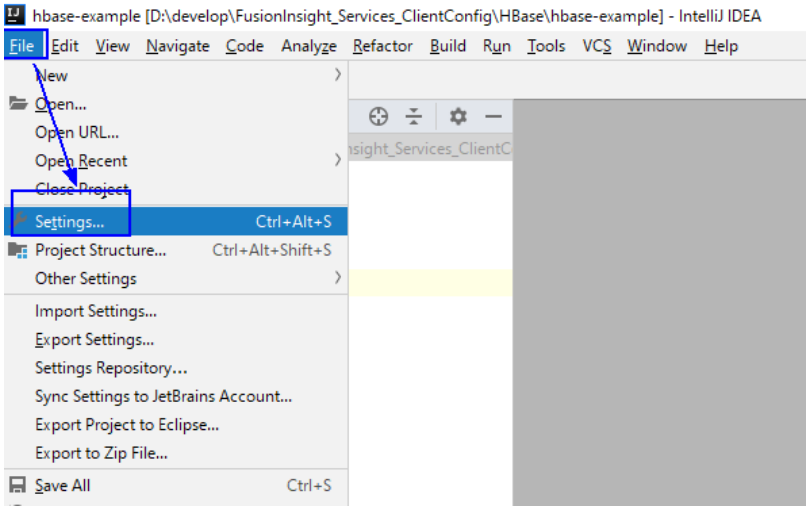
2. Select a sample project folder, and click **OK**.
3. After the import is complete, check that the imported sample projects are displayed on the IDEA home page.
4. Right-click **pom.xml** and choose **Add as Maven Project** from the shortcut menu to add the project as a Maven project. If the "pom.xml" icon is displayed as  , go to the next step.



Step 5 Set the Maven version used by the project.

1. Choose **File** > **Settings...** from the main menu of IntelliJ IDEA.

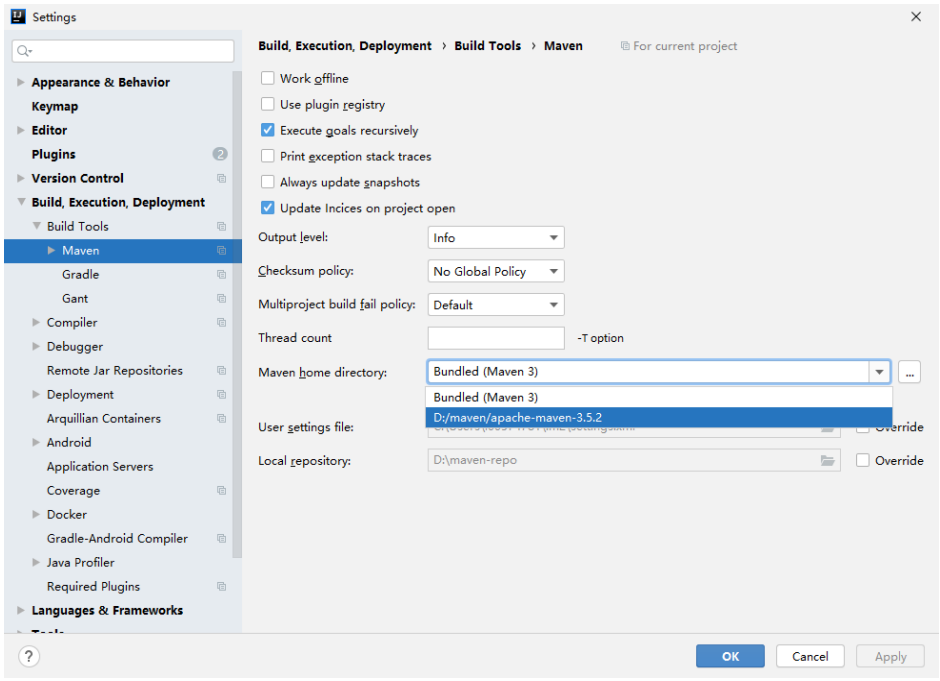
Figure 23-7 Settings



2. Choose **Build, Execution, Deployment** > **Maven** and set **Maven home directory** to the Maven version installed on the local PC.

   Set **User settings file** and **Local repository** as you need, and click **Apply** > **OK**.

Figure 23-8 Selecting the local Maven installation directory



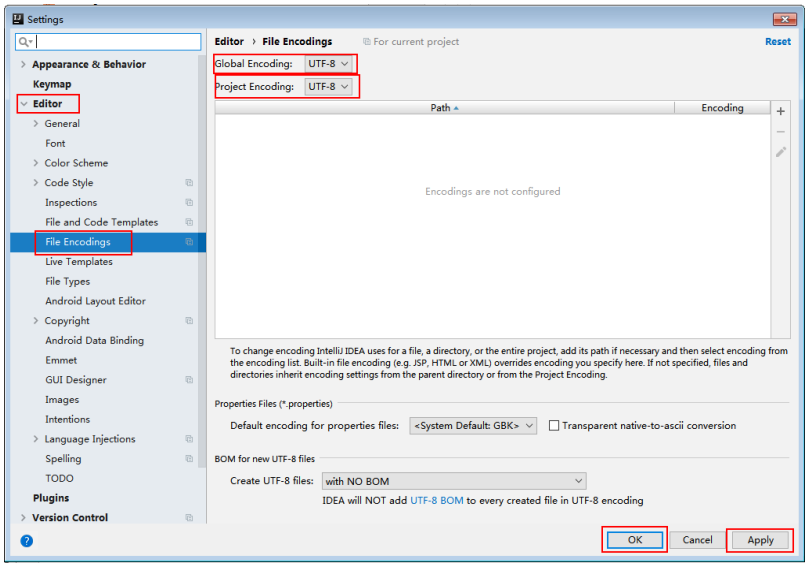
Step 6 Set the IntelliJ IDEA text file coding format to prevent garbled characters.

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

   The **Settings** window is displayed.

2. In the left navigation pane, choose **Editor** > **File Encodings**. In the **Project Encoding** and **Global Encoding** areas, set the parameter to **UTF-8**. Click **Apply** and **OK**, as shown in **Figure 23-9**.

Figure 23-9 Setting the IntelliJ IDEA encoding format



----End

23.3 Developing an Application

23.3.1 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.

23.3.2 Example Code Description

23.3.2.1 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 occured : {}.", ie);
            } catch (ExecutionException ee) {
                LOG.info("The ExecutionException occured : {}.", ee);
            }
        }
        messageNo++;
    }
}
```

23.3.2.2 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());
    }
}
```

23.3.2.3 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++;
     }
}
```


23.3.2.4 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());
        }
    }
}
```


23.3.2.5 KafkaStreams Sample

Function

The section describes High Level Kafka Streams API and Low Level Kafka Streams API sample codes. Kafka Streams counts words in each message by reading messages in the input topic and outputs the result in key-value pairs by consuming data in the output topic.

High Level Kafka Streams API Sample Code

Step 1 The following code snippets are in the createWordCountStream method of the **com.huawei.bigdata.kafka.example.WordCountDemo** class.

```
static void createWordCountStream(final StreamsBuilder builder) {
    // Receive input records from the input topic.
    final KStream<String, String> source = builder.stream(INPUT_TOPIC_NAME);

    // Aggregate the calculation results of the key-value pairs.
    final KTable<String, Long> counts = source
            // Process the received records and split them based on the regular expression REGEX_STRING.
            .flatMapValues(value ->
Arrays.asList(value.toLowerCase(Locale.getDefault()).split(REGEX_STRING)))
            // Aggregate the calculation results of the key-value pairs.
            .groupBy((key, value) -> value)
            // Output the final result.
            .count();

    // Output the key-value pairs from the output topic.
    counts.toStream().to(OUTPUT_TOPIC_NAME, Produced.with(Serdes.String(), Serdes.Long()));
}
// keytab file name of the machine-machine account that a user applies for
private static final String USER_KEYTAB_FILE = "Change it to the real-world keytab file name.";
// Machine-machine account that a user applies for
private static final String USER_PRINCIPAL = "Change it to the real-world username."
```

----End

Low Level Kafka Streams API Sample Code

Step 1 The following code snippets are in the **com.huawei.bigdata.kafka.example.WordCountProcessorDemo** class.

```
private static class MyProcessorSupplier implements ProcessorSupplier<String, String> {
    @Override
    public Processor<String, String> get() {
        return new Processor<String, String>() {
            // ProcessorContext instance, which provides access to the metadata of the record being processed.
            private ProcessorContext context;
            private KeyValueStore<String, Integer> kvStore;

            @Override
            @SuppressWarnings("unchecked")
            public void init(ProcessorContext context) {
                // Retain the processor context on the local PC because it will be used in punctuate() and
commit().
                this.context = context;
                // Execute punctuate() 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 record as a key-value pair to the downstream processor.
```



```
                        context.forward(entry.key, entry.value.toString());
                    }
                }
            });
            // Search for the key-value state storage zone KEY_VALUE_STATE_STORE_NAME, which can be
used to memorize the recently received input records.
            this.kvStore = (KeyValueStore<String, Integer>)
context.getStateStore(KEY_VALUE_STATE_STORE_NAME);
        }

        // Process the received records of the 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() {
        }
    };
  }
}

// keytab file name of the machine-machine account that a user applies for
private static final String USER_KEYTAB_FILE = "Change it to the real-world keytab file name.";
// Machine-machine account that a user applies for
private static final String USER_PRINCIPAL = "Change it to the real-world username."
```

----End

23.4 Application Commissioning

23.4.1 Producer Sample Commissioning

Prerequisites

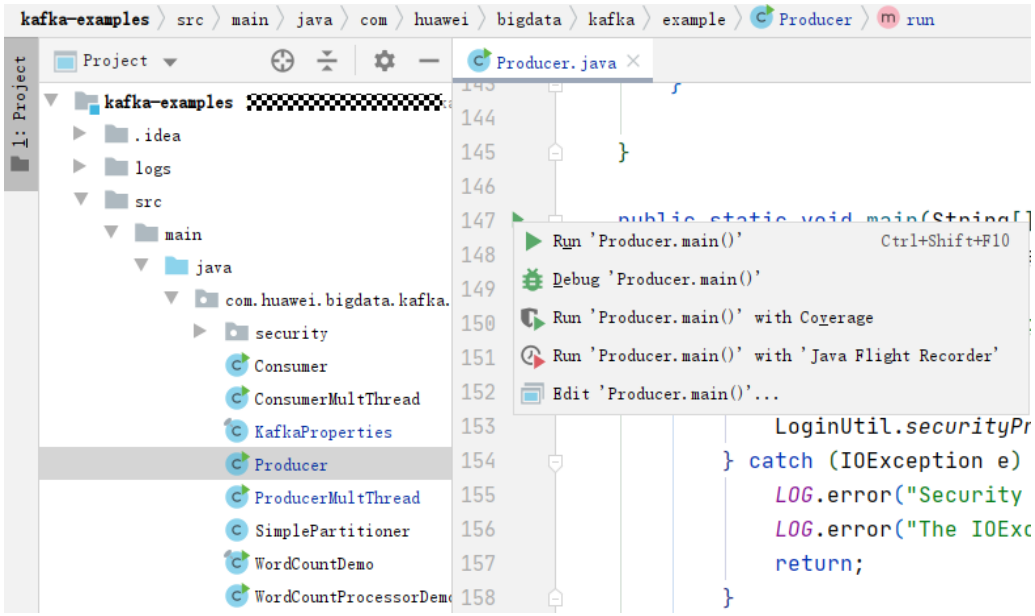
- Ensure that Windows is configured to allow Kafka access through an EIP if you need to debug applications on Windows. For details, see **Kafka Access Configuration on Windows Using EIPs**.
- Ensure that the current user has the read permission on all files in the **src/main/resources** directory and the dependent library file directory if you need to debug applications on Linux. Ensure that the JDK has been installed and Java environment variables have been set.

Commissioning Applications on Windows

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.

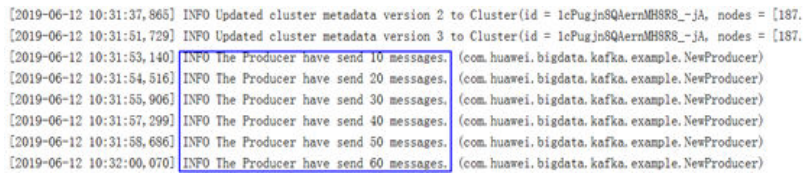
Step 2 Run Producer.java on IntelliJ IDEA, as shown in **Figure 23-10**.

Figure 23-10 Running Producer.java.



Step 3 Check the displayed console window and 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 23-11 Producer running window



----End

Commissioning Applications on Linux

Step 1 Export a JAR package.

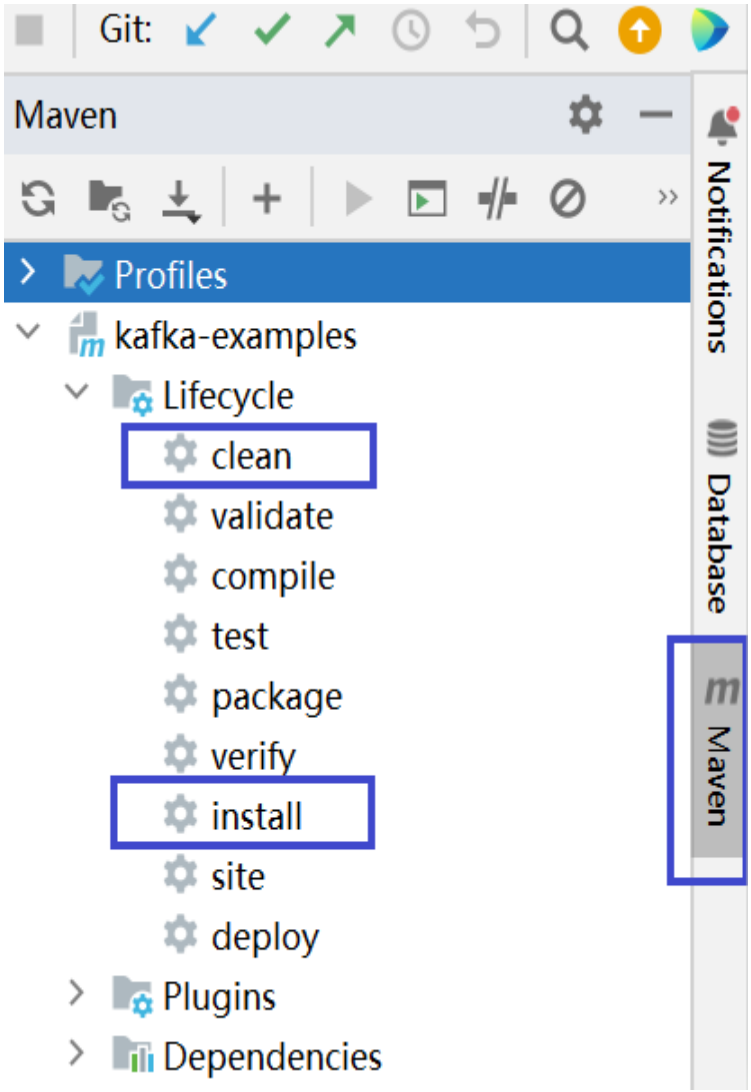
You can build a JAR file in either of the following ways:

- Method 1:

  Choose **Maven** > *Sample projectname* > **Lifecycle** > **clean**, double click **clean** to run the **clean** command of Maven.

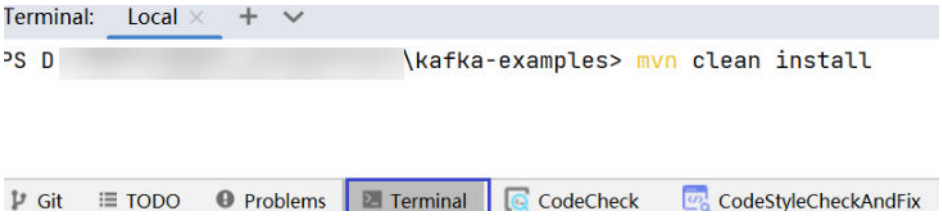
  Choose **Maven** > *Sample project name* > **Lifecycle** > **install**, double click **install** to run the **install** command of Maven.

Figure 23-12 Maven tools: **clean** and **installl**



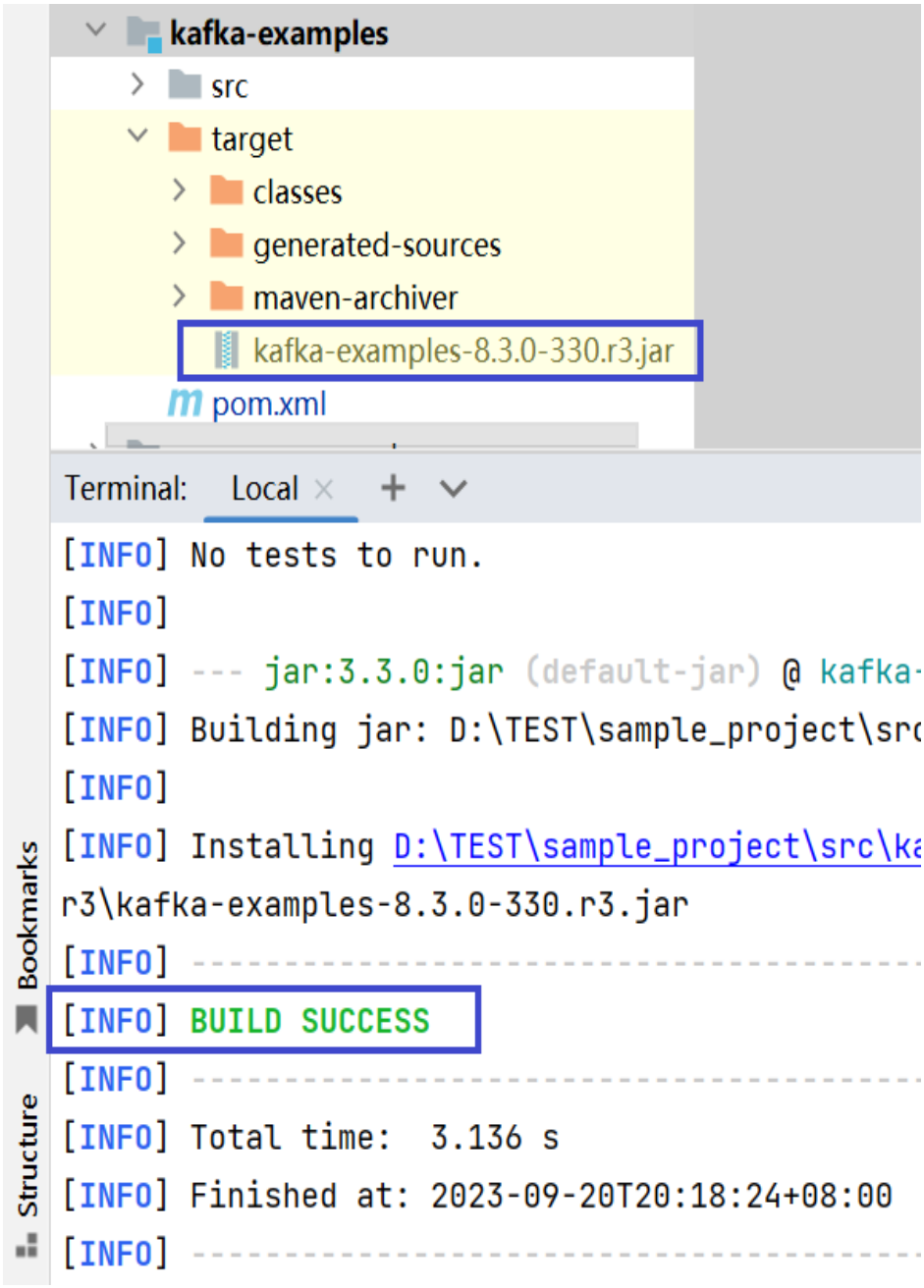
- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the **pom.xml** file.

Figure 23-13 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 23-14 When the compilation is completed, the JAR file is generated.



Step 2 Copy the JAR file generated during project compilation to the **/opt/hadoopclient/lib** directory.

Step 3 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 4 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

23.4.2 Consumer Sample Commissioning

Prerequisites

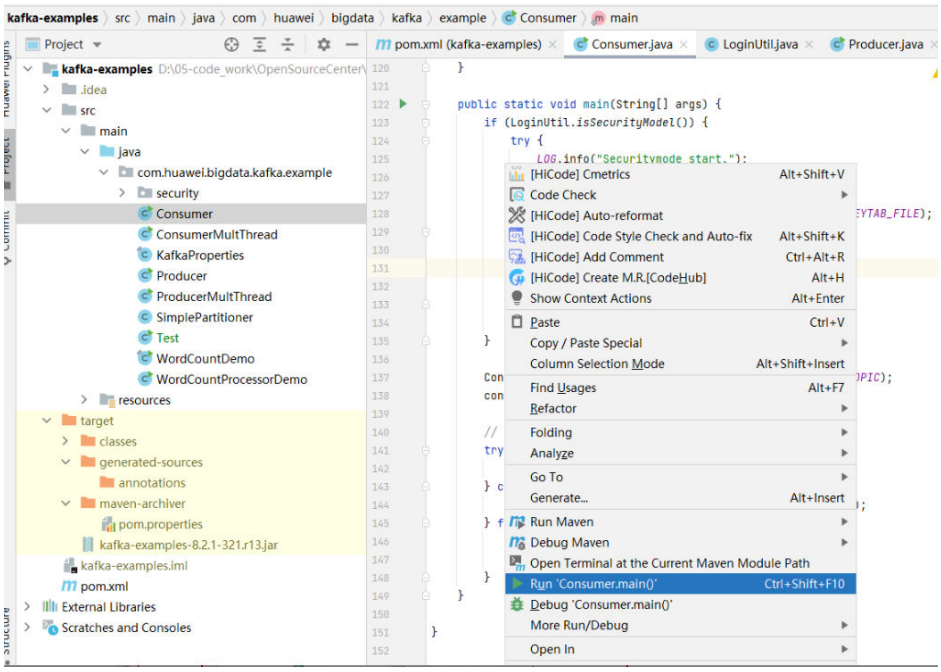
- Ensure that Windows is configured to allow Kafka access through an EIP if you need to debug applications on Windows. For details, see **Kafka Access Configuration on Windows Using EIPs**.
- Ensure that the current user has the read permission on all files in the **src/main/resources** directory and the dependent library file directory if you need to debug applications on Linux. Ensure that the JDK has been installed and Java environment variables have been set.

Commissioning Applications on Windows

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 Consumer.java on IntelliJ IDEA, as shown in **Figure 23-15**.

Figure 23-15 Running Consumer.java



Step 3 Click **Run**. In the displayed console window, you can see that Producer starts after Consumer has started and you can view the messages received in real time.

Figure 23-16 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.inter
[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: (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

Commissioning Applications on Linux

Step 1 Compile and generate a JAR package, and copy the JAR package to the **src/main/resources** directory at the same level as the dependent library folder. For details, see **Commissioning Applications on Linux**.

Step 2 Run the following command to run the consumer sample project:

java -cp /opt/client/lib/*:/opt/client/src/main/resources com.huawei.bigdata.kafka.example.Consumer

----End

23.4.3 High Level Streams Sample Commissioning

Commissioning Applications on Windows

For how to debug applications on Windows, see **Commissioning Applications on Windows**.

Commissioning Applications on Linux

Step 1 Compile and generate a JAR package, and copy the JAR package to the **src/main/resources** directory at the same level as the dependent library folder. For details, see **Commissioning Applications on Linux**.

Step 2 Log in to the cluster client node as the cluster installation user.

cd /opt/client

source bigdata_env

Step 3 Create an input topic and an output topic. Ensure that the topic names are the same as those specified in the sample code. Set the cleanup policy of the output topic to **compact**.

kafka-topics.sh --create --zookeeper *IP address of the quorumpeer instance:ZooKeeper client connection port*/**kafka --replication-factor 1 --partitions 1 --topic** *Topic name*

To query the IP address of the quorumpeer instance, log in to FusionInsight Manager of the cluster, choose **Cluster** > **Services** > **ZooKeeper**, and click the **Instance** tab. Use commas (,) to separate multiple IP addresses. You can obtain the ZooKeeper client connection port by querying the ZooKeeper configuration parameter **clientPort**. The default value is **2181**.

Run the following commands:

kafka-topics.sh --create --zookeeper 192.168.0.17:2181/kafka --replication-factor 1 --partitions 1 --topic streams-wordcount-input

kafka-topics.sh --create --zookeeper 192.168.0.17:2181/kafka --replication-factor 1 --partitions 1 --topic streams-wordcount-output --config cleanup.policy=compact

Step 4 Run the following command to run the application after the topics are created:

java -cp /opt/client/lib/*:/opt/client/src/main/resources com.huawei.bigdata.kafka.example.WordCountDemo

Step 5 Open a new client window and run the following commands to use **kafka-console-producer.sh** to write messages to the input topic:

cd /opt/client

source bigdata_env

kafka-console-producer.sh --broker-list *IP address of the broker instance:Kafka connection port(for example, 192.168.0.13:9092)* **--topic streams-wordcount-input --producer.config /opt/client/Kafka/kafka/config/producer.properties**

Step 6 Open a new client window and run the following commands to use **kafka-console-consumer.sh** to consume data from the output topic and view the result:

cd /opt/client

source bigdata_env

kafka-console-consumer.sh --topic streams-wordcount-output --bootstrap-server *IP address of the broker instance:Kafka connection port* **--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**

Write a message to the input topic.

```
>This is Kafka Streams test
>test starting
>now Kafka Streams is running
>test end
```

The output is as follows:

```
this    1
is      1
kafka   1
streams 1
test    1
test    2
starting 1
now     1
kafka   2
streams 2
is      2
running 1
test    3
end     1
```

----End

23.4.4 Low Level Streams Sample Commissioning

Commissioning Applications on Windows

For how to debug applications on Windows, see **Commissioning Applications on Windows**.

Commissioning Applications on Linux

Step 1 Compile and generate a JAR package, and copy the JAR package to the **src/main/resources** directory at the same level as the dependent library folder. For details, see **Commissioning Applications on Linux**.

Step 2 Log in to the node where the cluster client is installed as user **root**.

cd /opt/client

source bigdata_env

Step 3 Create an input topic and an output topic. Ensure that the topic names are the same as those specified in the sample code. Set the cleanup policy of the output topic to **compact**.

kafka-topics.sh --create --zookeeper *IP address of the quorumpeer instance:ZooKeeper client connection port*/**kafka --replication-factor 1 --partitions 1 --topic** *Topic name*

To query the IP address of the quorumpeer instance, log in to FusionInsight Manager of the cluster, choose **Cluster** > **Services** > **ZooKeeper**, and click the **Instance** tab. Use commas (,) to separate multiple IP addresses. You can obtain the ZooKeeper client connection port by querying the ZooKeeper configuration parameter **clientPort**. The default value is **2181**.

Run the following commands:

kafka-topics.sh --create --zookeeper 192.168.0.17:2181/kafka --replication-factor 1 --partitions 1 --topic streams-wordcount-processor-input

kafka-topics.sh --create --zookeeper 192.168.0.17:2181/kafka --replication-factor 1 --partitions 1 --topic streams-wordcount-processor-output --config cleanup.policy=compact

Step 4 Run the following command to run the application after the topics are created:

java -cp /opt/client/lib/*:/opt/client/src/main/resources com.huawei.bigdata.kafka.example.WordCountDemo

Step 5 Open a new client window and run the following commands to use **kafka-console-producer.sh** to write messages to the input topic:

cd /opt/client

source bigdata_env

kafka-console-producer.sh --broker-list *IP address of the broker instance:Kafka connection port(for example, 192.168.0.13:9092)* **--topic streams-wordcount-processor-input --producer.config /opt/client/Kafka/kafka/config/producer.properties**

Step 6 Open a new client window and run the following commands to use **kafka-console-consumer.sh** to consume data from the output topic and view the result:

cd /opt/client

source bigdata_env

kafka-console-consumer.sh --topic streams-wordcount-processor-output --bootstrap-server *IP address of the broker instance:Kafka connection port* **--**

consumer.config /opt/client/Kafka/kafka/config/consumer.properties --from-beginning --property print.key=true --property print.value=true

Write a message to the input topic.

```
>This is Kafka Streams test
>test starting
>now Kafka Streams is running
>test end
```

The output is as follows:

```
this    1
is      1
kafka   1
streams 1
test    1
test    2
starting 1
now     1
kafka   2
streams 2
is      2
running 1
test    3
end     1
```

----End

23.5 More Information

23.5.1 External Interfaces

23.5.1.1 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

23.5.1.2 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 23-5 Major parameters of a Producer

Parameter	Description	Remarks
bootstrap.servers	Broker address list	The Producer creates connections with the Broker based on this parameter.

Parameter	Description	Remarks
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.
value.serializer	Serialization of messages	This parameter specifies how to serialize transmitted messages.

Table 23-6 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.

Return Value	Interface Function	Description
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 23-7 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 **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 23-8 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.

23.5.2 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.

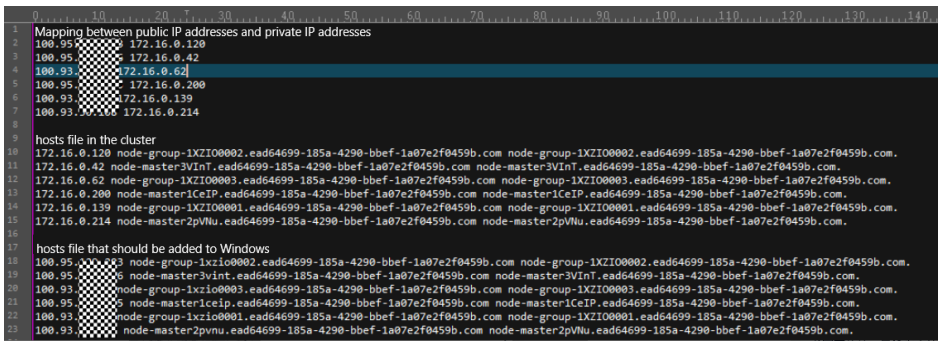
Procedure

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.

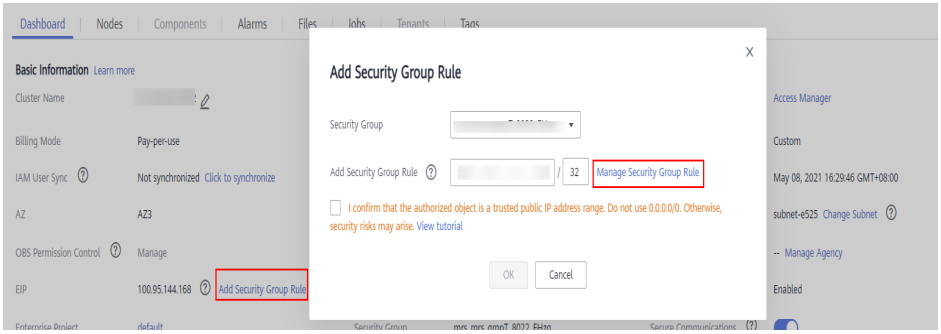
   For details, see **Virtual Private Cloud** > **User Guide** > **EIP** > **Assigning an EIP and Binding It to an ECS**.

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.

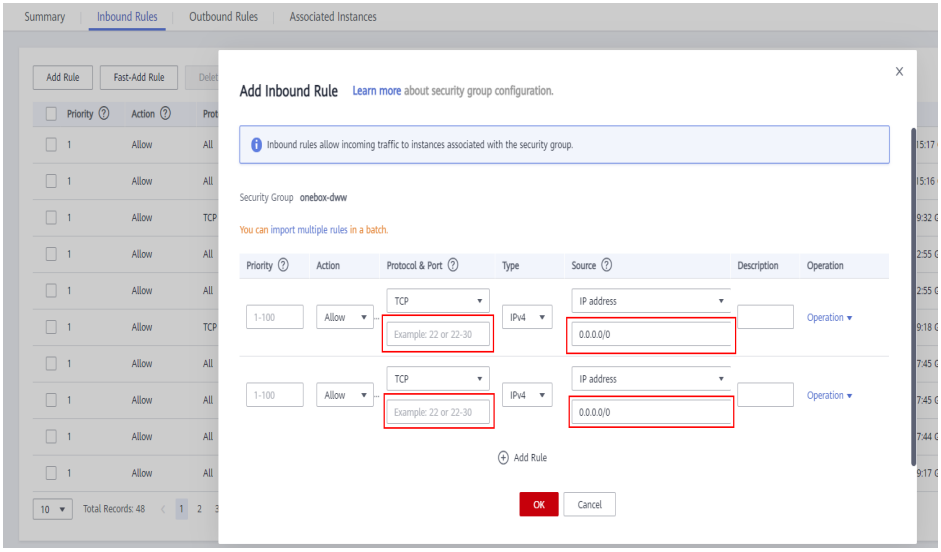


Step 2 Configure security group rules for the cluster.

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.expandor** parameter, save the configuration, and restart the Kafka cluster.

NOTE

If the current cluster is MRS 3.2.0-LTS.1 and you cannot access Kafka through the EIP after performing this step, perform the following operations:

1. Log in to FusionInsight Manager, choose **Cluster > Services > Kafka > Instances**, select all Broker instances, and choose **More > Stop Instance** to verify the administrator password and stop all Broker instances. (This operation affects services. Perform this operation during off-peak hours.)
2. Log in to the Broker node as the **root** user and modify the **server.properties** file.

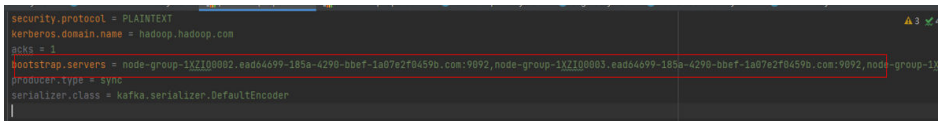
   vi ${BIGDATA_HOME}/FusionInsight_HD_*/*_*_Broker/etc/server.properties

   Change **host.name** to the host name of the current Broker node, and change the values of **listeners** and **advertised.listeners** to **EXTERNAL_PLAINTEXT://*{Host name}*:*{port}***.
3. Log in to FusionInsight Manager, choose **Cluster > Services > Kafka > Instances**, select all Broker instances, and click **Start Instance**.
4. Bind an EIP to each Broker node in the MRS cluster.
5. On Windows, use the configured EIP and port of the Broker node to connect to the Kafka cluster and debug the code.

Step 4 Before running the sample code, change the Kafka connection string in the sample code to *hostname1*:**9092,** *hostname2*:**9092,** *hostname3*:**9092**, change the domain name in the code.

NOTE

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.



----End

23.5.3 FAQ

23.5.3.1 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

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

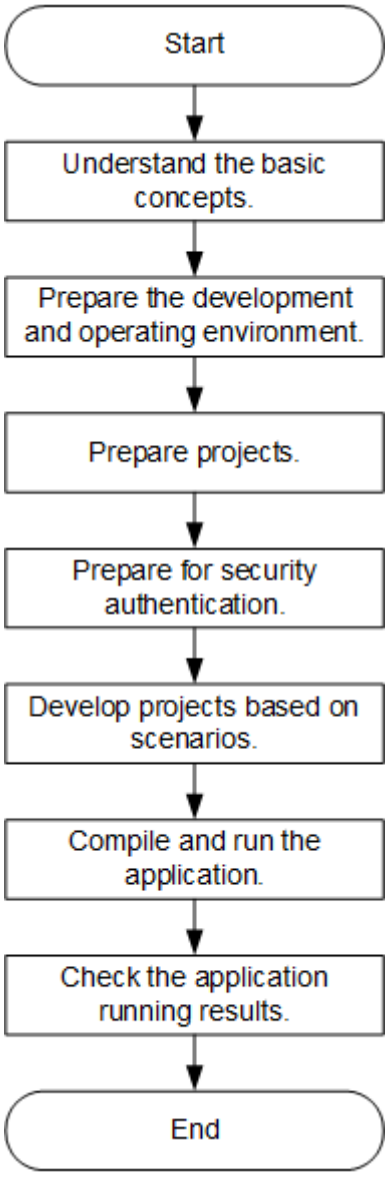


Table 24-1 Description of MapReduce development process

Stage	Description	Reference
Understand the basic concepts.	Before the application development, the basic concepts of MapReduce are required to be understood.	**Basic Concepts**

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 MapReduce is the MapReduce client. Install and configure the client based on the reference.	**Preparing for Development and Operating Environment**
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.	**Configuring and Importing Sample Projects** **Creating a New Project (Optional)**
Prepare for safety certification.	If a safe cluster is used, the safety certification must be performed.	**Preparing the Authentication Mechanism**
Develop projects based on scenarios.	Provide the example project. This helps users to learn about the programming interfaces of all Spark components quickly.	**Developing the Project**
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	**Commissioning the Application**
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.	**Commissioning the Application**

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	• 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	It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version. **NOTE** • 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.
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
JDK installation	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. • For x86 nodes that run clients, the following JDKs can be used: – 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: – OpenJDK: 1.8.0_272 **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 com.ibm.jsse2.overrideDefaultTLS 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.**

Preparation Item	Description
Developer account preparation	See **Preparing the 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 **Complete Client** (For MRS 3.3.0 or later, click **Download Client** in the upper right corner of the **Homepage**). 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-Clien

     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 the 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.

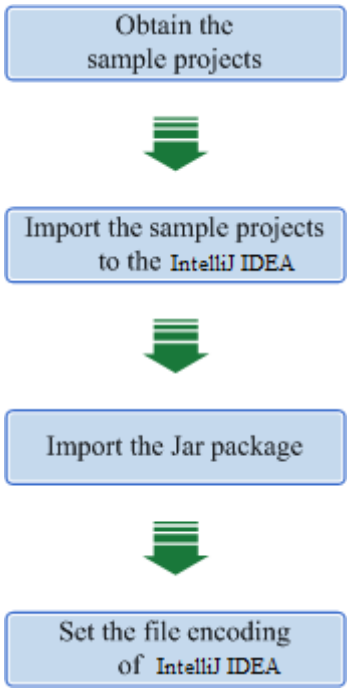
24.2.2 Configuring and Importing Sample Projects

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 the 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.

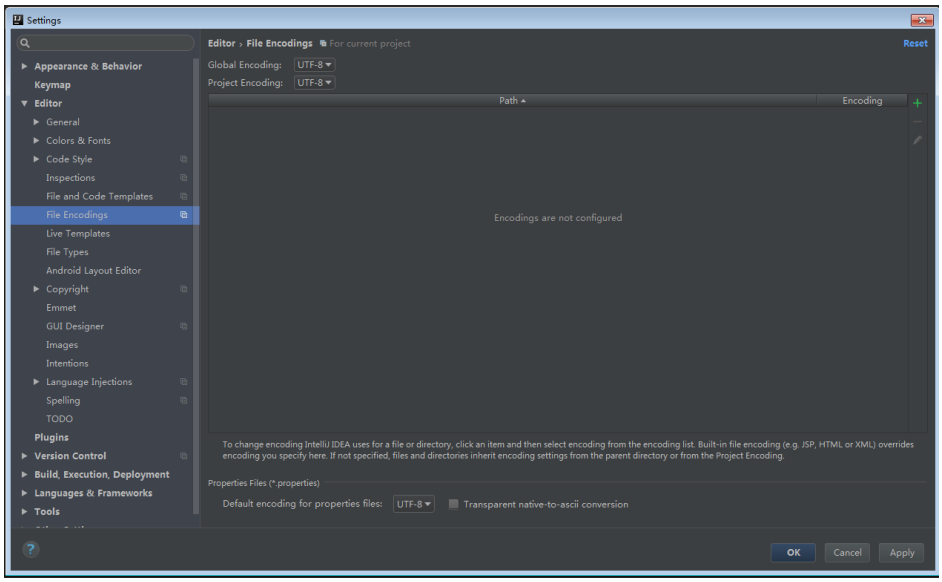
1. Open IntelliJ IDEA and choose **File > Open**.
2. Choose the directory of the example project **mapreduce-example-security**. Click **OK**.

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**

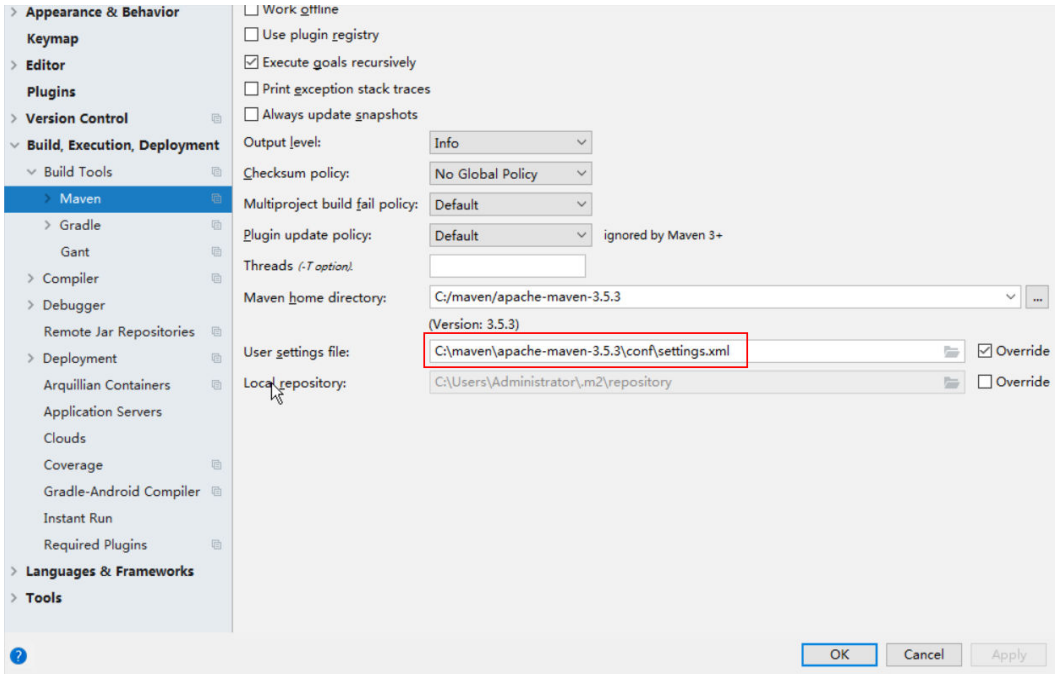
Figure 24-3 Setting the IntelliJ IDEA coding format



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)

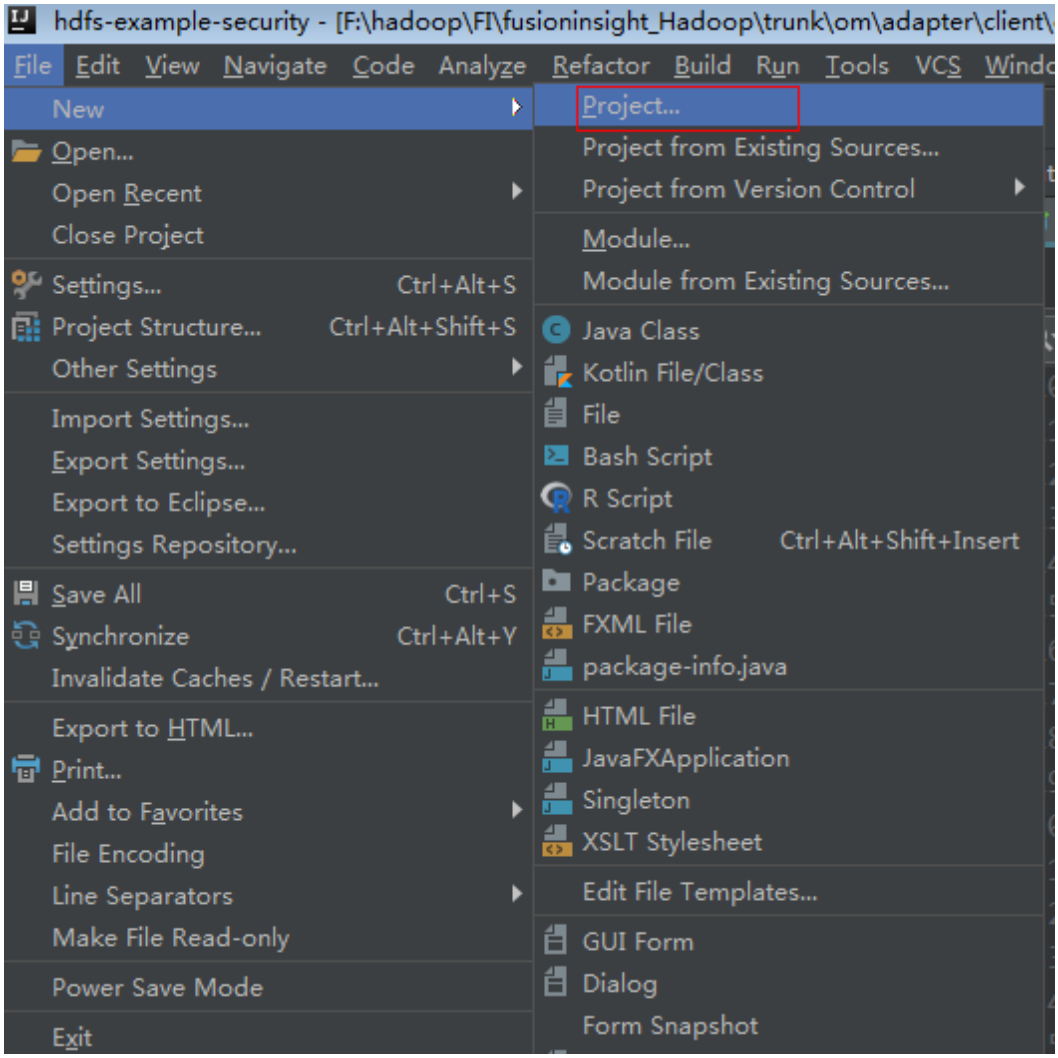
Scenario

Apart from importing MapReduce sample projects, you can also create a new MapReduce project using IntelliJ IDEA.

Procedure

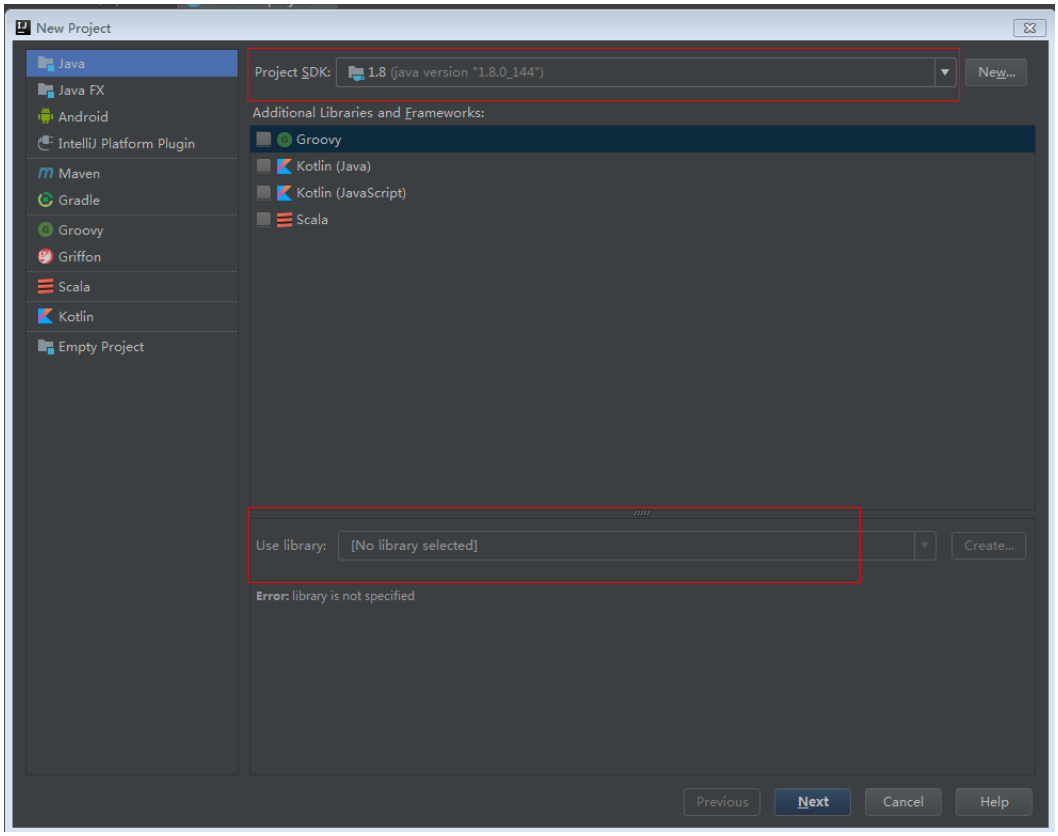
Step 1 Open IntelliJ IDEA and choose **File > New > Project**, as shown in **Figure 24-5**.

Figure 24-5 Creating a project



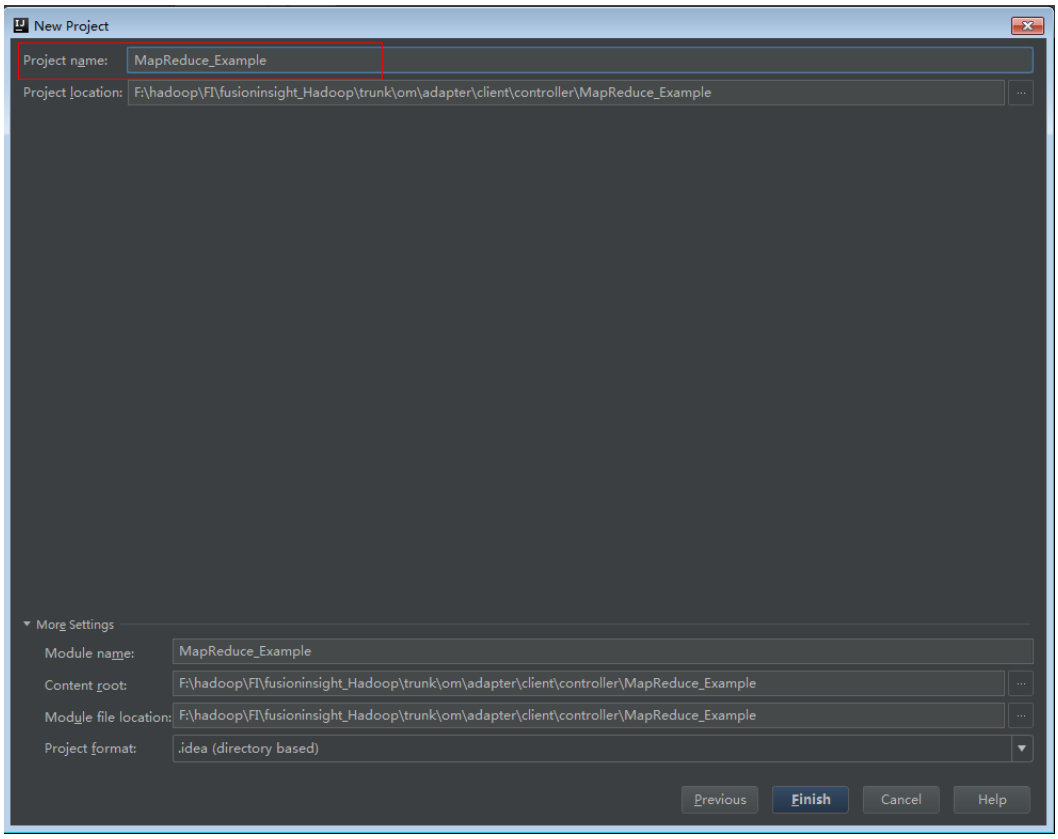
Step 2 On the **New Project** page, select **Java**, and then configure the JDK and other Java libraries required by the project. As shown in the following figure. After the configuration is complete, click **Next**.

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 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.

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.MultiComponentExampl.

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(config);

    // 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 dependecy 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.

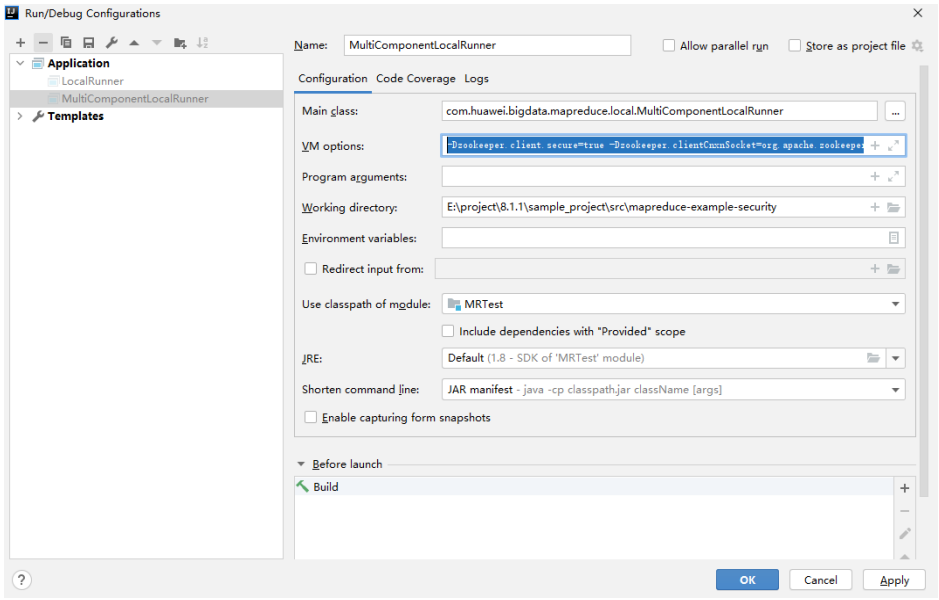
Create the **jaas_mr.conf** file in the **conf** directory and add the following content (**test** is the user corresponding to **user.keytab**):

```
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

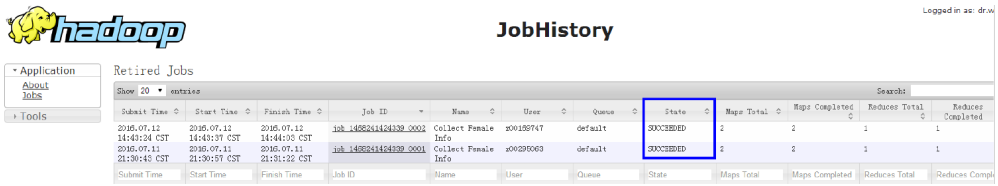
In the Windows environment, the following exception occurs but does not affect services.

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.

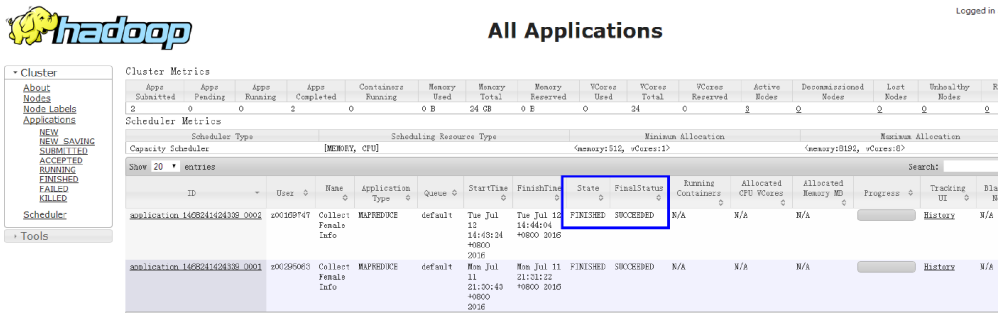
Figure 24-8 JobHistory Web UI



- **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, *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 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

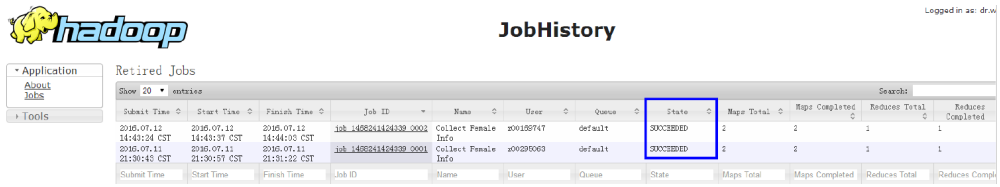
You must have permission to access WebUI. If not, contact the admin to obtain an account and password.

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.

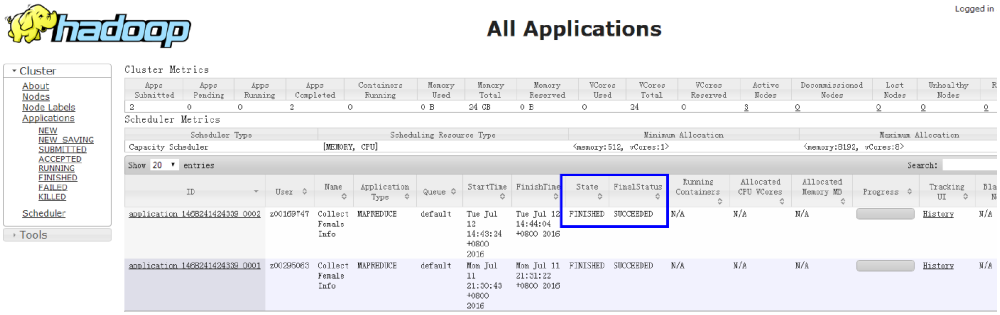
  Figure 24-10 JobHistory Web UI

- **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 **mapreduce.job.map.class** in **mapred-site.xml**.
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 **mapreduce.job.reduce.class in mapred-site.xml**.
setCombinerClass(Class<extends Reducer> cls)	Specifies the Combiner class of a MapReduce job. The Combiner class is empty by default. You can also configure **mapreduce.job.combine.class** in **mapred-site.xml**. 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 **TextInputFormat**. You can also configure **mapreduce.job.inputformat.class** in mapred-site.xml. 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 **mapreduce.job.jar** in **mapred-site.xml**.
setOutputFormat-Class(Class<extends OutputFormat> theClass)	A core interface used to specify the OutputFormat class of a MapReduce job. The default OutputFormat class is **TextOutputFormat**. You can also configure mapred.output.format.class in **mapred-site.xml**. In the default **TextOutputFormat**, each key and value are recorded in text. **OutputFormat** 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 **mapreduce.job.output.key.class** in **mapred-site.xml**.
setOutputValueClass(Class < > theClass)	A core interface used to specify the output value type of a MapReduce job. You can also configure **mapreduce.job.output.value.class** in **mapred-site.xml**.
setPartitionerClass(Class<extends Partitioner> theClass)	Specifies the Partitioner class of a MapReduce job. You can also configure **mapred.partitioner.class** in **mapred-site.xml**. This method is used to allocate Map output results to reduce classes. **HashPartitioner** 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.
setSortComparator-Class(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 **mapreduce.map.output.compress** and **mapreduce.map.output.compress.codec** in **mapred-site.xml**. 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: **VERY_HIGH, HIGH, NORMAL, LOW**, and **VERY_LOW**. The default priority is **NORMAL**. You can also configure **mapreduce.job.priority** in **mapred-site.xml**.

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 **mapreduce.job.maps** in **mapred-site.xml**. **NOTE** 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 **mapreduce.job.reduces** in **mapred-site.xml**. 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 **configure mapreduce.job.queuename** in **mapred-site.xml**.

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 theWebUI 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.

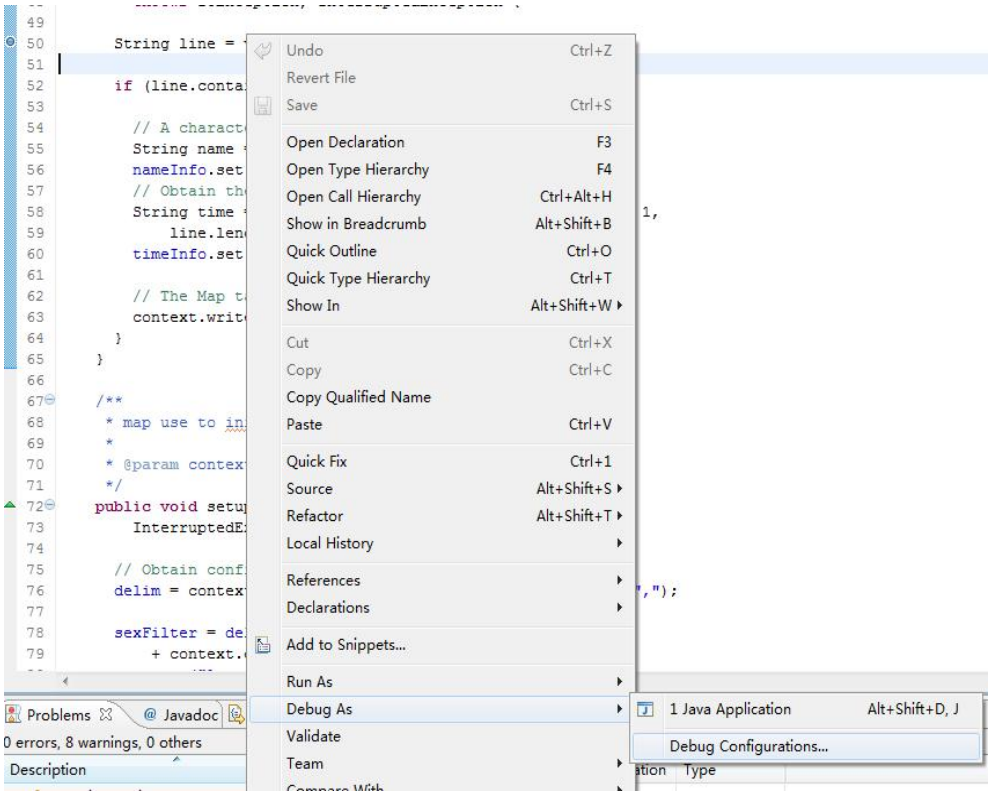
Step 3 Submit MapReduce tasks on the client, then the Map or Reduce tasks will be suspended and listen to the port 8000 to wait for remote debugging.

Step 4 In IDE, select the implementation class of MapReduce tasks and configure the remote debugging information to perform debugging.

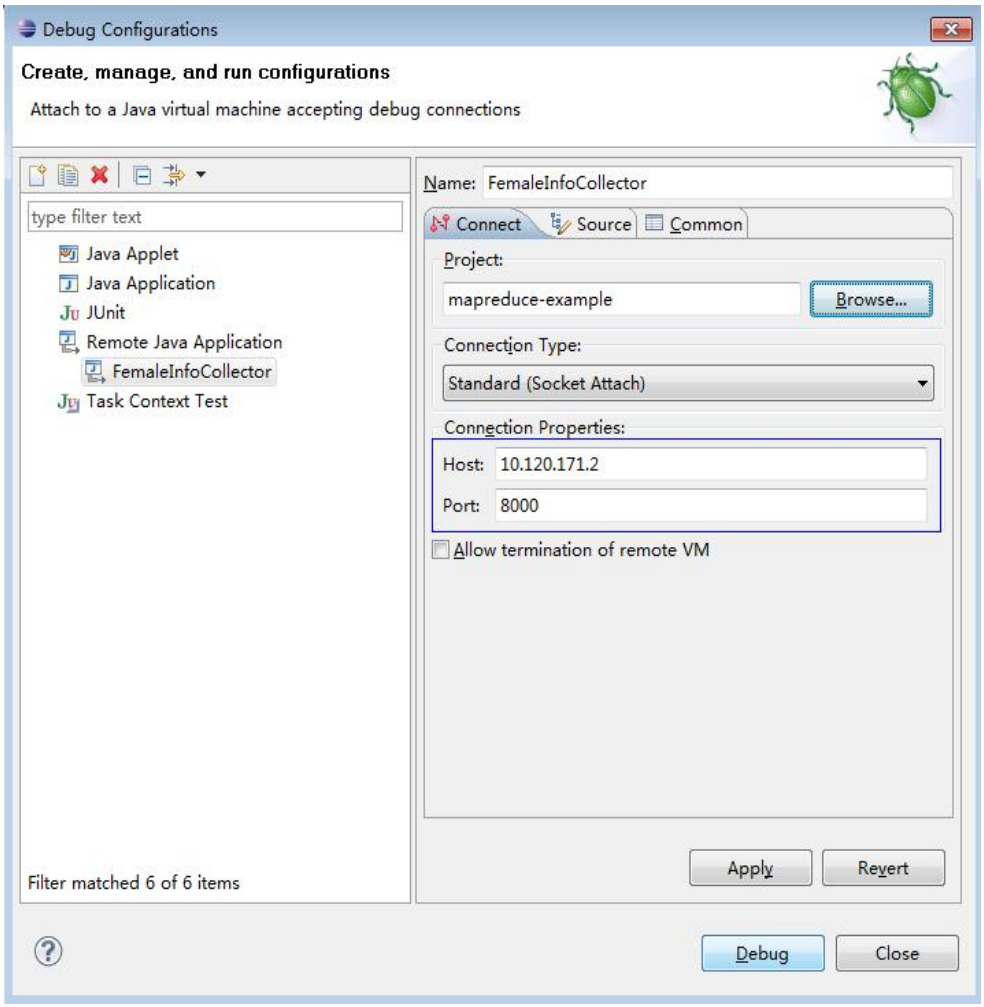
1. Double-click the area in the blue box to configure or cancel the break point.

```
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

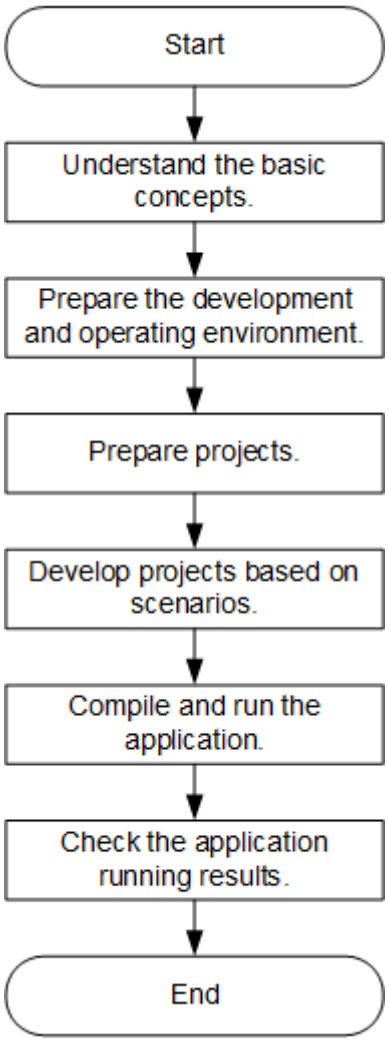


Table 25-1 Description of MapReduce development process

Stage	Description	Reference
Understand the basic concepts.	Before the application development, the basic concepts of MapReduce are required to be understood.	**Basic Concepts**
Prepare the development and operating environment.	Use IntelliJ IDEA and configure the development environment based on the reference. The running environment of MapReduce is the MapReduce client. Install and configure the client based on the reference.	**Preparing Development and Operating Environment**

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.	**Configuring and Importing Sample Projects** **Creating a New Project (Optional)**
Develop projects based on scenarios.	Provide the example project. This helps users to learn about the programming interfaces of all Spark components quickly.	**Developing the Project**
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	**Commissioning the Application**
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.	**Commissioning the Application**

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	• 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.

Preparation Item	Description
IntelliJ IDEA installation and configuration	It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version. **NOTE** • 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.
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
JDK installation	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. • For x86 nodes that run clients, the following JDKs can be used: – 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: – OpenJDK: 1.8.0_272 **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 com.ibm.jsse2.overrideDefaultTLS 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.**
7-zip	Used to decompress **.zip** and **.rar** 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** (For MRS 3.3.0 or later, click **Download Client** in the upper right corner of the **Homepage**). 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.

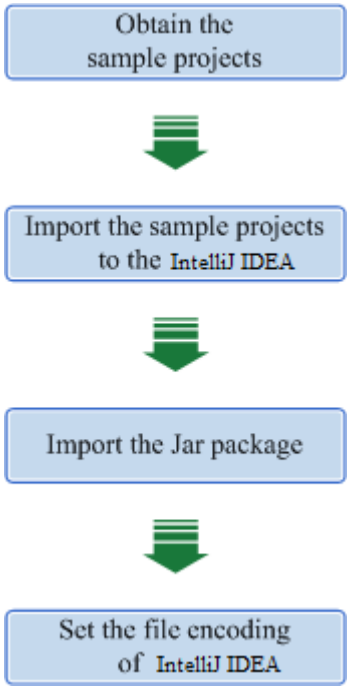
25.2.2 Configuring and Importing Sample Projects

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.

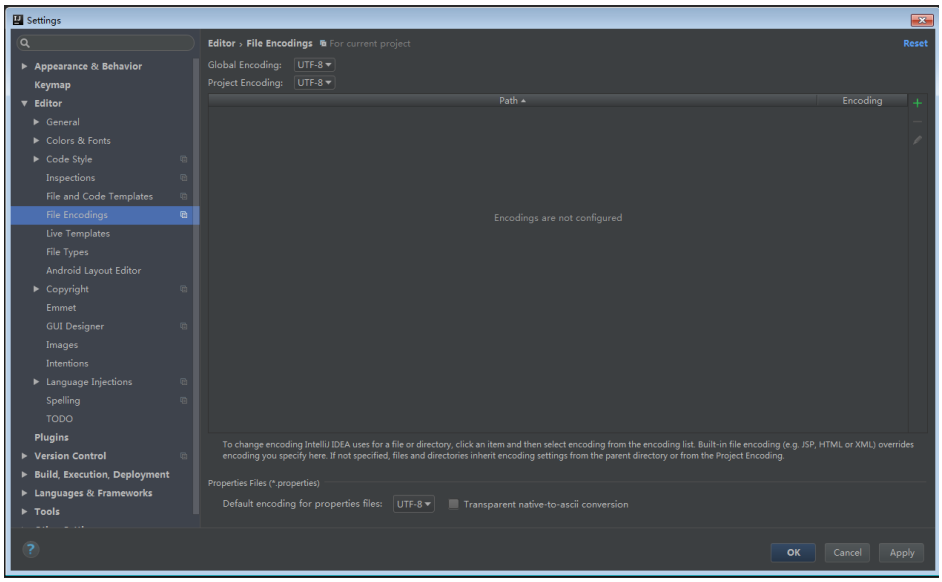
1. Open IntelliJ IDEA and choose **File > Open**.
2. Choose the directory of the example project **mapreduce-example-normal**. Click **OK**.

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**

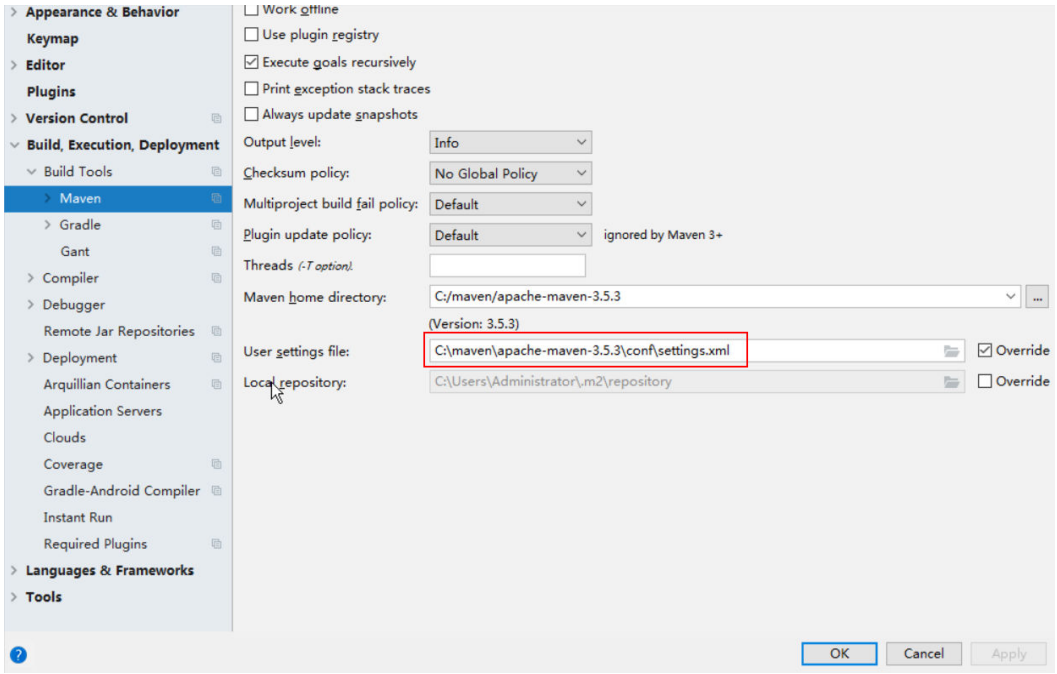
   Figure 25-3 Setting the IntelliJ IDEA coding format



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)

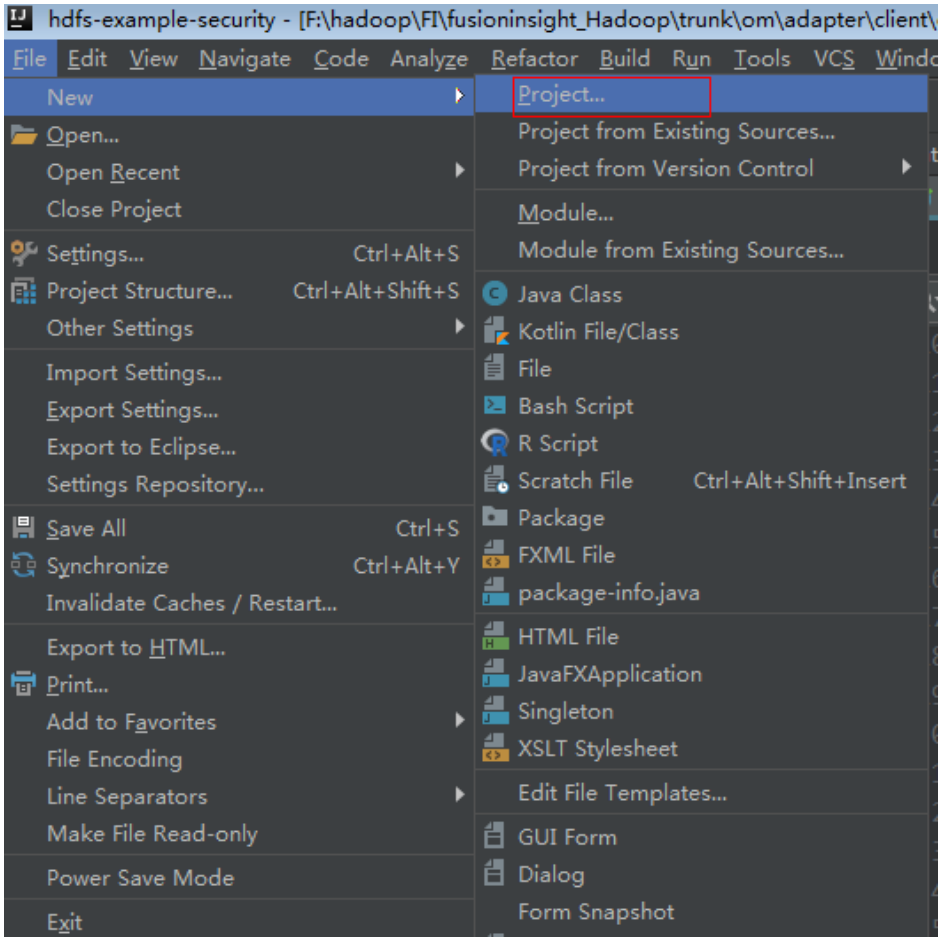
Scenario

Apart from importing MapReduce sample projects, you can also create a new MapReduce project using IntelliJ IDEA.

Procedure

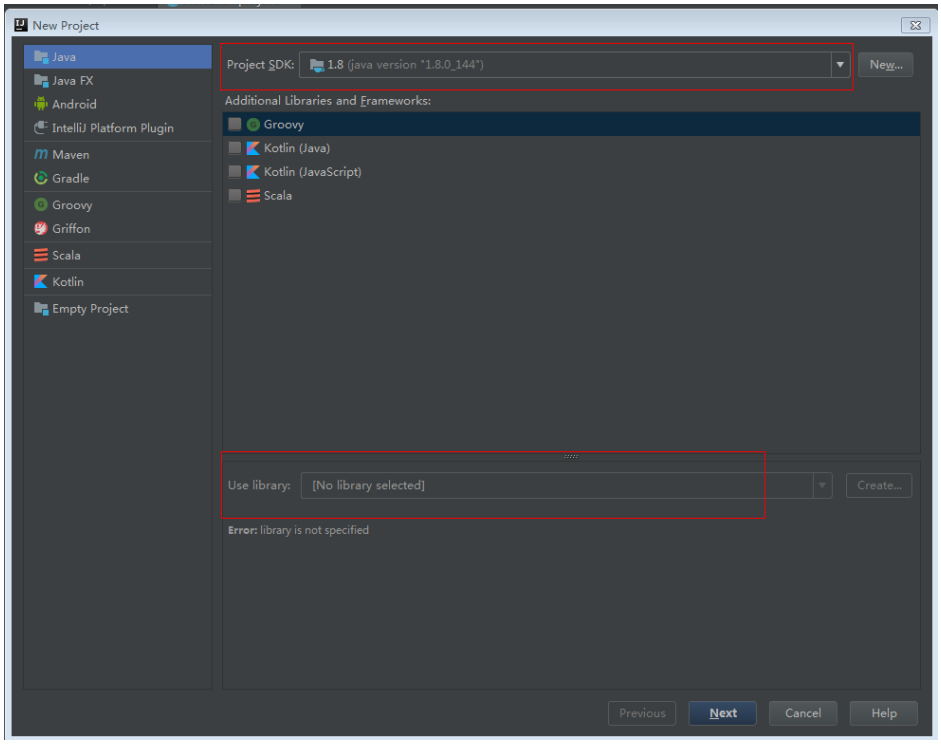
Step 1 Open IntelliJ IDEA and choose **File > New > Project**, as shown in **Figure 25-5**.

Figure 25-5 Creating a project



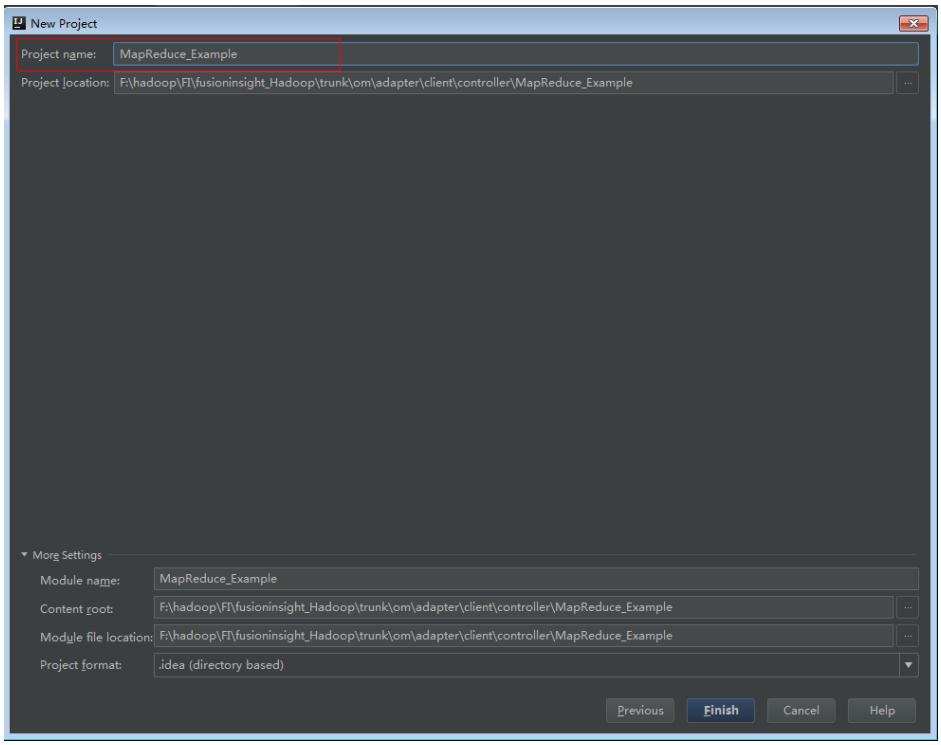
Step 2 On the **New Project** page, select **Java**, and then configure the JDK and other Java libraries required by the project. **Figure 25-6** shows the SDK information required for configuring a project. After the configuration is complete, click **Next**.

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 sex, 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");
```



```
      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 {
    String hiveClientProperties =
MultiComponentExample.class.getClassLoader().getResource("hiveclient.properties").getPath();
    // A file that contains configuration information.
    String file =  "file://" + hiveClientProperties;
    // In runtime, put the configuration information on HDFS.
    config.set("tmpfiles", file);
    // Clean up the desired directory before submitting the job.
    MultiComponentExample.cleanupBeforeRun();
    // 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");
    Class thriftType = Class.forName("org.apache.hadoop.hive.serde2.thrift.Type");
    // Add dependency jars to Job

    JarFinderUtil.addDependencyJars(config, hiveDriverClass, serviceThriftCLIClass, thriftCLIClass, thriftClass,
            hiveConfClass, hiveTransClass, hiveMetaClass, hiveShimClass, thriftType);
    // Add hive config file.
    config.addResource("hive-site.xml");
    // Add hbase config file
    Configuration conf = HBaseConfiguration.create(config);
    // 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 the path of 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));
    // Sets the output key type
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    // HBase use this utility class to add dependecy jars of hbase to MR job
    TableMapReduceUtil.addDependencyJars(job);
    // Submit the job to a remote environment for execution.
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
```


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

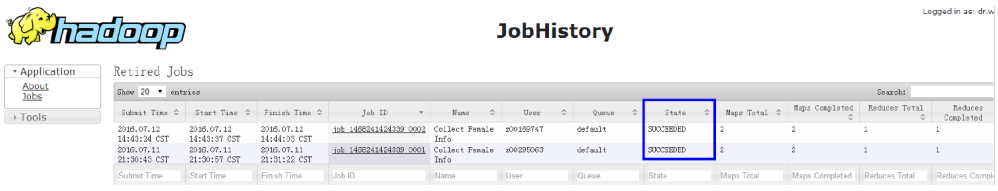
In the Windows environment, the following exception occurs but does not affect services.

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.

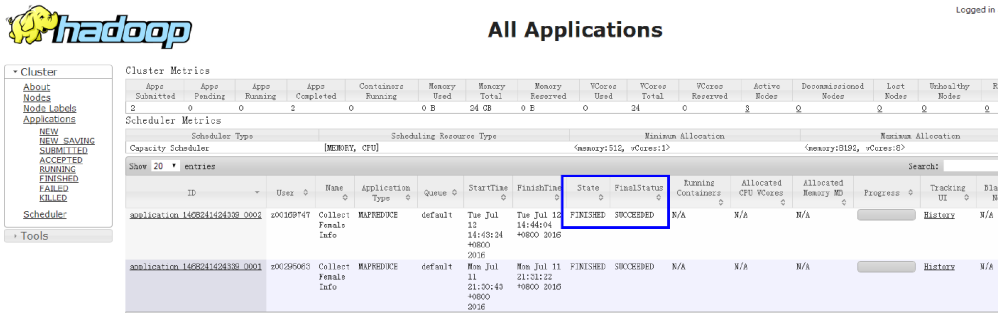
Figure 25-8 JobHistory Web UI



- **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

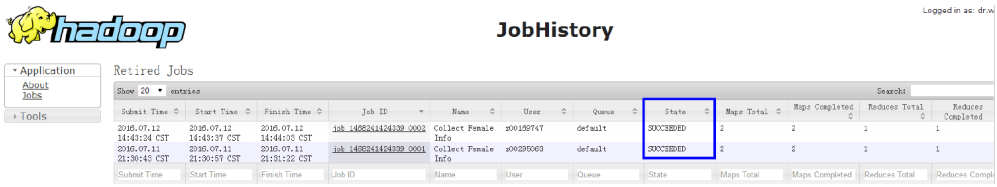
You must have permission to access WebUI. If not, contact the admin to obtain an account and password.

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.

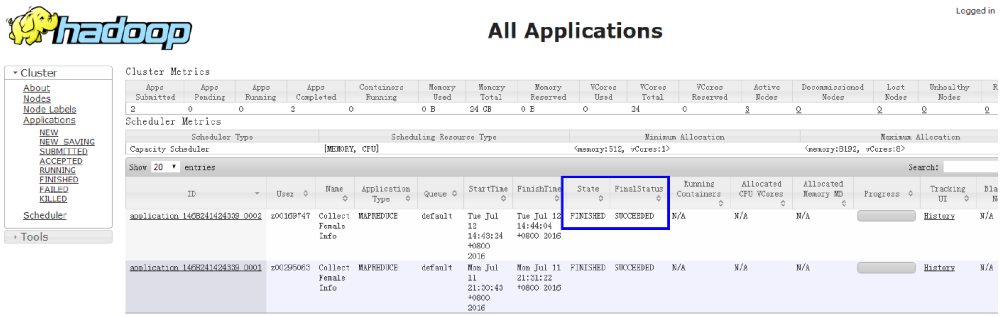
  Figure 25-10 JobHistory Web UI

- **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: **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 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 **mapreduce.job.map.class** in **mapred-site.xml**.
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 **mapreduce.job.reduce.class in mapred-site.xml**.

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 **mapreduce.job.combine.class** in **mapred-site.xml**. 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 **TextInputFormat**. You can also configure **mapreduce.job.inputformat.class** in mapred-site.xml. 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 setJar(String jar) or setJarByClass(Class< > cls). You can also configure **mapreduce.job.jar** in **mapred-site.xml**.
setOutputFormat-Class(Class<extends OutputFormat> theClass)	A core interface used to specify the OutputFormat class of a MapReduce job. The default OutputFormat class is **TextOutputFormat**. You can also configure mapred.output.format.class in **mapred-site.xml**. In the default **TextOutputFormat**, each key and value are recorded in text. **OutputFormat** 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 **mapreduce.job.output.key.class** in **mapred-site.xml**.
setOutputValueClass(Class< > theClass)	A core interface used to specify the output value type of a MapReduce job. You can also configure **mapreduce.job.output.value.class** in **mapred-site.xml**.

Interface	Description
setPartitionerClass(Class<extends Partitioner> theClass)	Specifies the Partitioner class of a MapReduce job. You can also configure **mapred.partitioner.class** in **mapred-site.xml**. This method is used to allocate Map output results to reduce classes. **HashPartitioner** 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.
setSortComparator-Class(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 **mapreduce.map.output.compress** and **mapreduce.map.output.compress.codec** in **mapred-site.xml**. 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: **VERY_HIGH, HIGH, NORMAL, LOW**, and **VERY_LOW**. The default priority is **NORMAL**. You can also configure **mapreduce.job.priority** in **mapred-site.xml**.

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 **mapreduce.job.maps** in **mapred-site.xml**. **NOTE** 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 **mapreduce.job.reduces** in **mapred-site.xml**. 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 **configure mapreduce.job.queuename** in **mapred-site.xml**.

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.

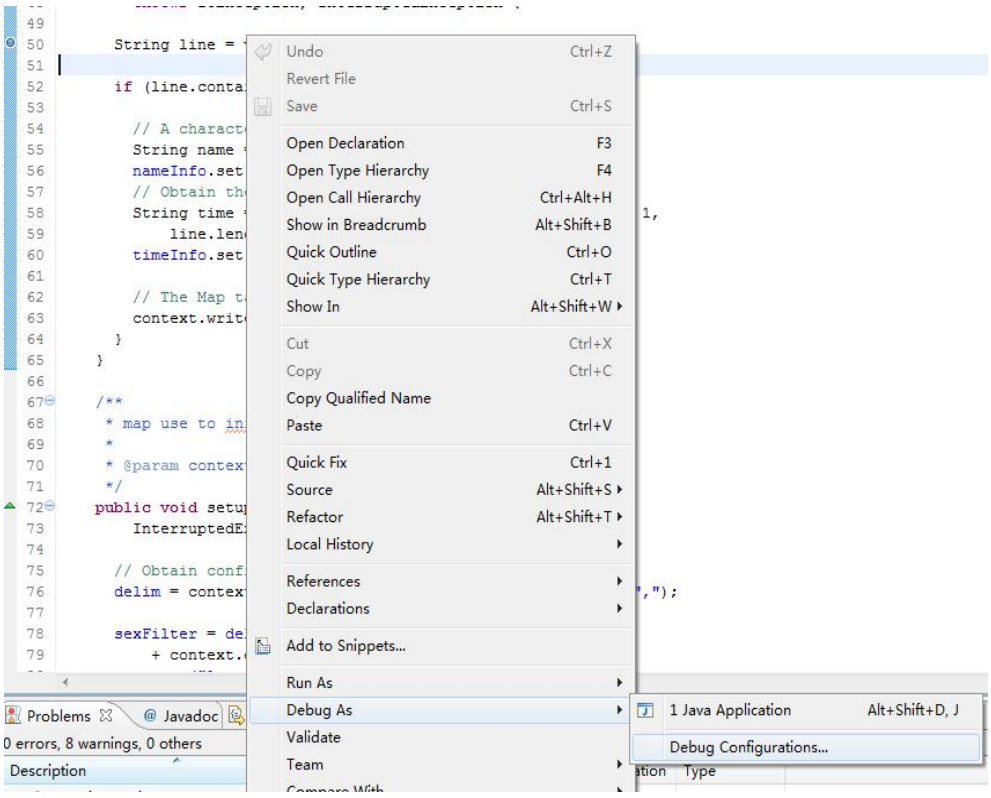
Step 3 Submit MapReduce tasks on the client, then the Map or Reduce tasks will be suspended and listen to the port 8000 to wait for remote debugging.

Step 4 In IDE, select the implementation class of MapReduce tasks and configure the remote debugging information to perform debugging.

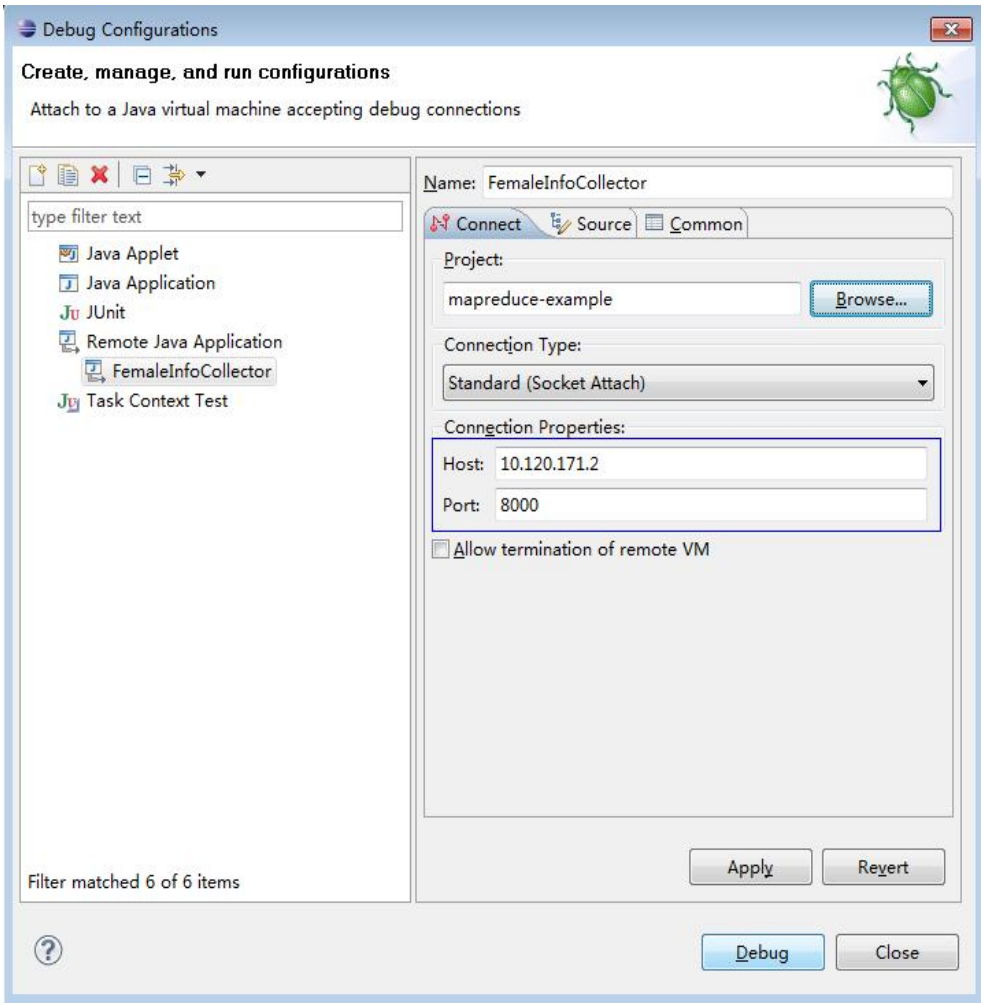
1. Double-click the area in the blue box to configure or cancel the break point.

```
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

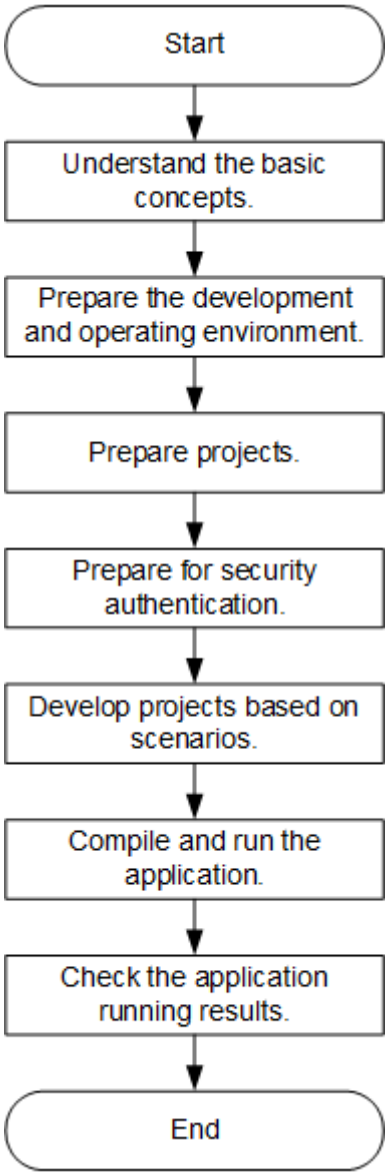


Table 26-1 Description of Oozie development process

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.	**Common Concepts**

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.	**Preparing Development and Operating Environment**
Prepare projects.	Oozie provides example projects for different scenarios. You can import an example project to learn the application.	**Downloading and Importing Sample Projects**
Prepare for security authentication.	If you use a security cluster, you need to perform security authentication.	**Preparing Authentication Mechanism Code**
Develop projects based on scenarios.	An example project based on the Java language is provided.	**Developing the Project**
Compile and run the application.	Compile the developed application and submit it for running.	**Commissioning the Application**
View application running results.	Application running results are written into a specified path. You can also view the application running status on the UI.	**Commissioning the Application**

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. 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	Basic configuration for the development and running 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. • For x86 nodes that run clients, the following JDKs can be used: – 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: – OpenJDK: 1.8.0_272 **NOTE** The server supports only TLS V1.2 or later to ensure security.
IDEA installation and configuration	It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version. **NOTE** • If the IBM JDK is used, ensure that the JDK configured in IDEA is the IBM JDK. • If the Oracle JDK is used, ensure that the JDK configured in IDEA is the Oracle JDK. • If the Open JDK is used, ensure that the JDK configured in IDEA is the Open 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 the Developer Account** for configuration.
7-zip	Used to decompress **.zip** and **.rar** packages. The 7-zip 16.04 is supported.

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 the 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 \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 5 Select the sample project to be run.

- For the **OozieMapReduceExcample** 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 **Start Action**
2	Define the MapReduce action.	For details, see **MapReduce Action**
3	Define the FS action.	For details, see **FS Action**
4	Define the end action.	For details, see **End Action**
5	Define the kill action.	For details, see **Kill Action**

   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 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 **job.properties**.
-run	Indicates the starts workflow.

For example:

oozie job -oozie https://10-1-130-10:21003/oozie -configjob.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
nameNode	Indicates the Hadoop distributed file system (HDFS) NameNode cluster address.
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
```

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>
            <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>
```

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 the 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	https://*Oozie service IP address*:21003/oozie/	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}/${examplesRoot}/apps/spark2x **NOTICE** Ensure that the path is the same as the path with the **<jar>** and **<spark-opts>** tags in the **src\main\resources\workflow.xml** file.
src\main\resources\workflow.xml	<jar> </jar>	Change **OoizeSparkHBase-1.0.jar** to the actual JAR package name.	<jar>${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\OozieSparkHBaseExample**, 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, **/opt/client/Spark2x/spark/conf**)	hive-site.xml	**/user/developuser/myjobs/apps/spark2x** directory in the HDFS.
	hbase-site.xml	

Initial File Path	File	Destination Path
Keytab file obtained in **Preparing the Developer Account**	user.keytab	
	krb5.conf	
Spark client directory (for example, **/opt/client/Spark2x/spark/jars**)	JAR package	Share HDFS **/user/oozie/share/lib/spark2x/** directory of Oozie.
JAR package of the sample projects to be used, for example, **OoizeSparkHBase-1.0.jar**	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. **NOTE** Change the path of **spark-archive-2x.zip** 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. **NOTE** Change the path of **spark-archive-2x.zip** in **<spark-opts>** based on the actual HDFS file path.

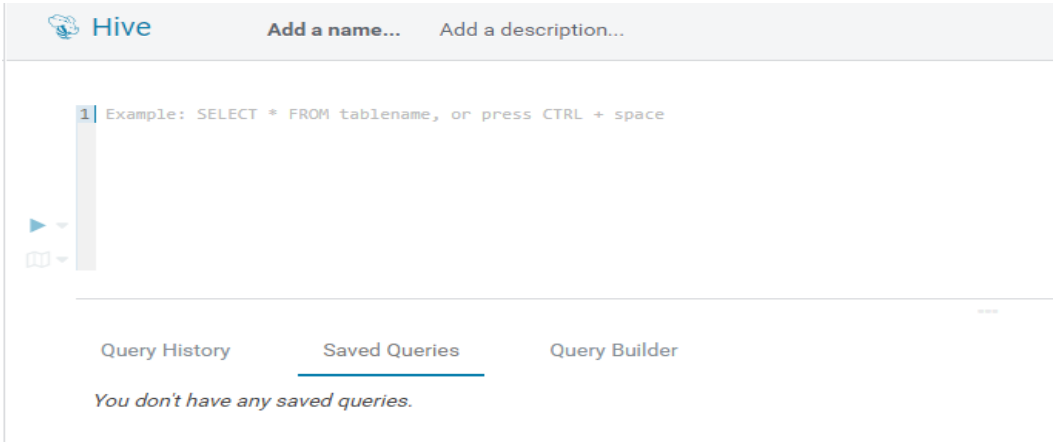
Step 6 Change the value of **hive.security.authenticator.manager** in the **hive-site.xml** file in the **/user/developuser/myjobs/apps/spark2x** directory of HDFS from **org.apache.hadoop.hive.ql.security.SessionStateUserMSGroupAuthenticator** to **org.apache.hadoop.hive.ql.security.SessionStateUserGroupAuthenticator**.

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_8.1.2.2/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 **job.properties**.
	-oozie <arg>	Indicates the Oozie server address.
	-haconfig <arg>	Indicates the path to the Oozie service configuration file **oozie-site.xml**.
	-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**.

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

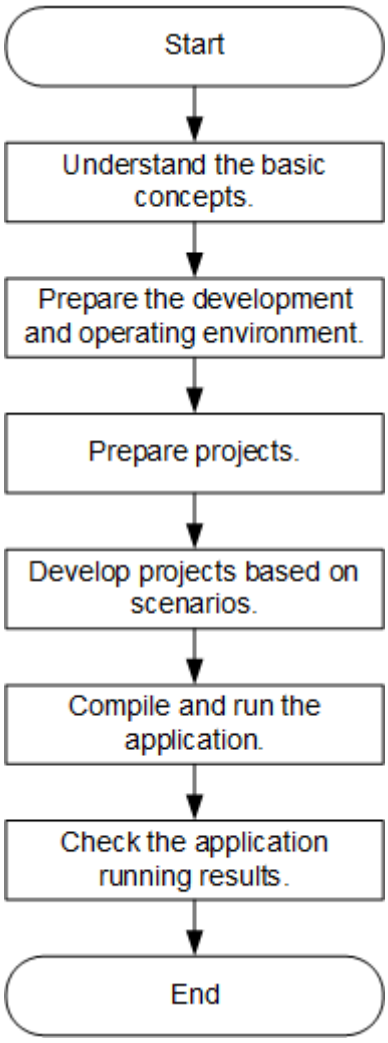


Table 27-1 Description of Oozie development process

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.	Common Concepts

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.	**Development and Operating Environment**
Prepare projects.	Oozie provides example projects for different scenarios. You can import an example project to learn the application.	**Downloading and Importing Sample Projects**
Develop projects based on scenarios.	An example project based on the Java language is provided.	**Developing the Project**
Compile and run the application.	Compile the developed application and submit it for running.	**Commissioning the Application**
View application running results.	Application running results are written into a specified path. You can also view the application running status on the UI.	**Commissioning the Application**

27.2 Environment Preparation

27.2.1 Development and Operating Environment

Table 27-2 describes the environment required for secondary development.

Table 27-2 Development environment

Item	Description
OS	Windows OS. Windows 7 or later is supported. If the program needs to be commissioned locally, the development and running environment must be able to communicate with the cluster service plane network.
JDK installation	Basic configuration for the development and running 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. ● For x86 nodes that run clients, the following JDKs can be used: – 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: – OpenJDK: 1.8.0_272 **NOTE** The server supports only TLS V1.2 or later to ensure security.
IDEA installation and configuration	configuration for the development environment. The version must be 2019.1 or other compatible version. **NOTE** ● If the IBM JDK is used, ensure that the JDK configured in IDEA is the IBM JDK. ● If the Oracle JDK is used, ensure that the JDK configured in IDEA is the Oracle JDK. ● If the Open JDK is used, ensure that the JDK configured in IDEA is the Open 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.
7-zip	Used to decompress **.zip** and **.rar** packages. The 7-zip 16.04 is supported.

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 **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 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 **OozieMapReduceExcample** 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 **Start Action**
2	Define the MapReduceaction.	For details, see **MapReduce Action**
3	Define the FS action.	For details, see **FS Action**
4	Define the end action.	For details, see **End Action**
5	Define the killaction.	For details, see **Kill Action**

   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 *client installation directory*/**bin/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 **job.properties**.
-run	Indicates the starts workflow.

For example:

/opt/client/Oozie/oozie-client-5.1.0-hw-ei-302002/bin/oozie job -oozie https://10-1-130-10:21003/oozie -configjob.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>
            <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://*Oozie service IP address*: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}/${examplesRoot}/apps/spark2x **NOTICE** Ensure that the path is the same as the path with the **<jar>** and **<spark-opts>** tags in the **src\main\resources\workflow.xml** file.
src\main\resources\workflow.xml	<jar> </jar>	Change **OoizeSparkHBase-1.0.jar** to the actual JAR package name.	<jar>${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\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.

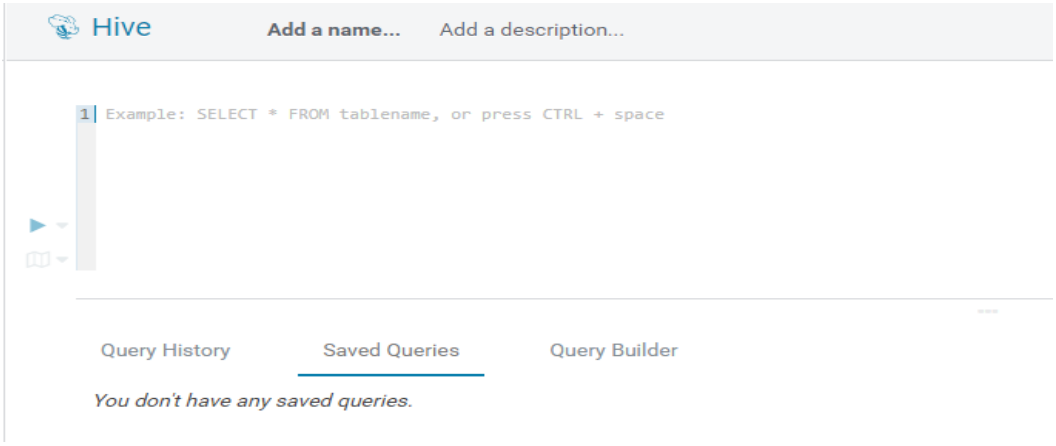
Table 27-15 Files to be uploaded

Initial File Path	File	Destination Path
Spark client directory (for example, **/opt/client/Spark2x/spark/conf**)	hive-site.xml hbase-site.xml	**/user/developuser/myjobs/apps/spark2x** directory in the HDFS.
Spark client directory (for example, **/opt/client/Spark2x/spark/jars**)	JAR package	Share HDFS **/user/oozie/share/lib/spark2x** directory of Oozie. **NOTE** This file must be uploaded as user **oozie**. Run the **su - oozie** command to switch to user **oozie**. 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. **NOTE** Change the path of **spark-archive-2x.zip** 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. **NOTE** Change the path of **spark-archive-2x.zip** 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.SessionStateUserMSGroupAuthenticator** 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_8.1.2.2/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 **job.properties**.
	-oozie <arg>	Indicates the Oozie server address.
	-haconfig <arg>	Indicates the path to the Oozie service configuration file **oozie-site.xml**.
	-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**.

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 Spark2x Development Guide (Security Mode)

28.1 Overview

28.1.1 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 Interface 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 28-1**.

Table 28-1 Spark APIs

Function	Description
Scala API	Indicates the API in Scala. For common interfaces of Spark Core, Spark SQL and Spark Streaming, see **Scala**. Since Scala is easy to read, users are advised to use Scala interfaces in the program development.
Java API	Indicates the API in Java. For common interfaces of Spark Core, Spark SQL and Spark Streaming, see **Java**.
Python API	Indicates the API in Python. For common interfaces of Spark Core, Spark SQL and Spark Streaming, see **Python**.

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.

28.1.2 Basic Concepts

Basic Concepts

- **RDD**

  Resilient Distributed Dataset (RDD) is a core concept of Spark. It indicates a read-only and partitionable 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.

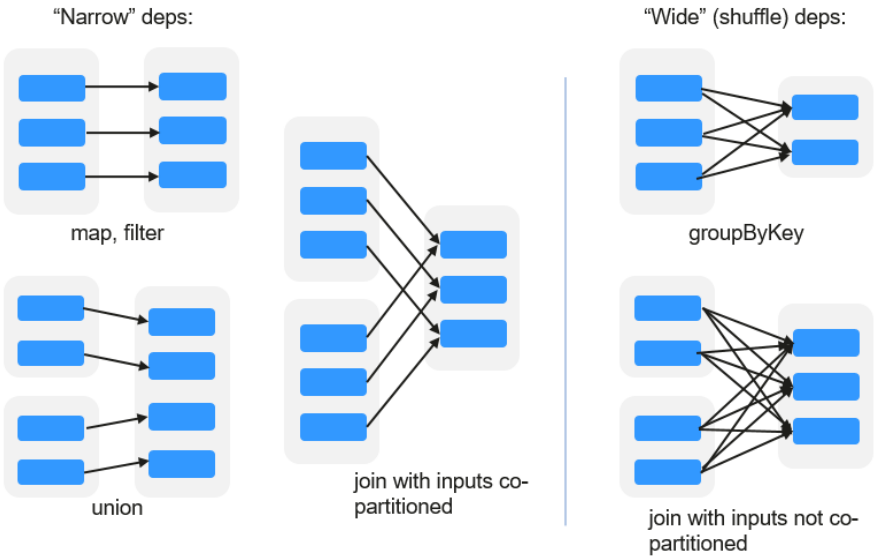
  RDD Storage

  - 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 28-1 RDD dependency



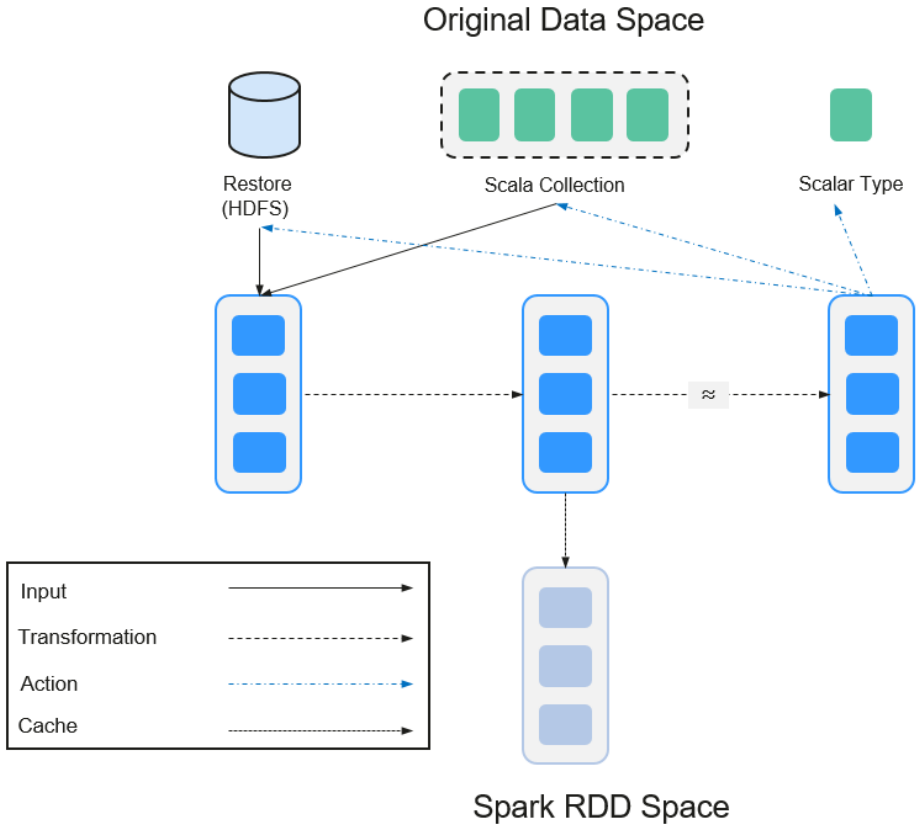
  - **Narrow dependency:** Each partition of the parent RDD is used by at most one partition of the child RDD partition.
  - **Wide dependency:** Partitions of the child RDD depend on all partitions of the parent RDD due to shuffle operations.

  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 28-2** 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 28-2 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()
```

a. The **textFile** operator reads log files from HDFS and returns **file** (as an RDD).
b. The filter operator filters rows with ERROR and assigns them to errors (a new RDD). The filter operator is a transformation.
c. The cache operator caches errors for future use.
d. 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).

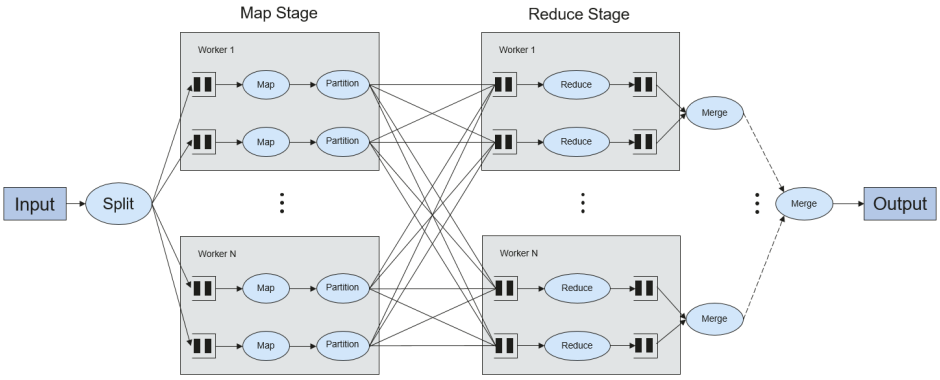
– Check points, such as checkpoint. When Lineage is long (which occurs frequently in graphics computation), it takes a long period of time to execute the whole sequence again when a fault occurs. In this case, checkpoint is used as the check point to write the current data to stable storage.

- **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 28-3 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. 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

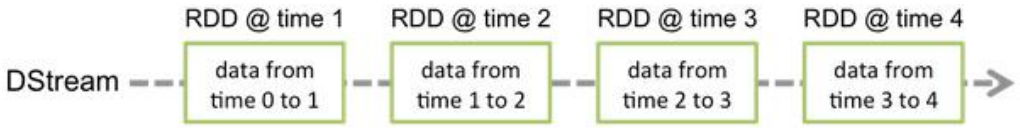
Dstream

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.

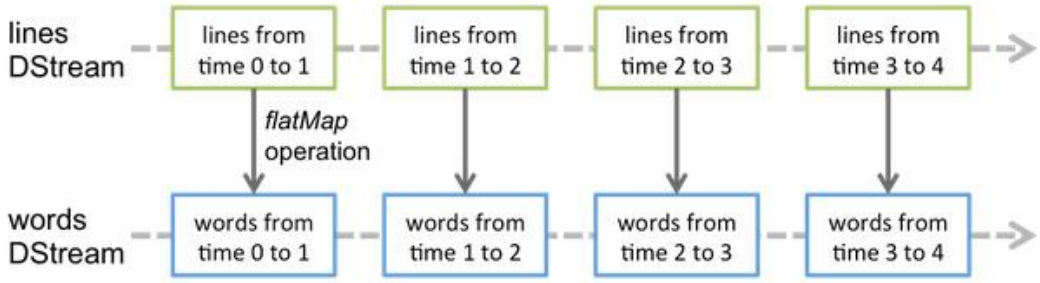
Each RDD in DStream contains the data of a partition as shown in **Figure 28-4**.

Figure 28-4 Relationship between DStream and RDD



All operators applied in DStream are translated to the operations in the lower RDD as shown in **Figure 28-5**. 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 28-5 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 a difference between the Appendix 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 28-6 Running process in micro-batch mode

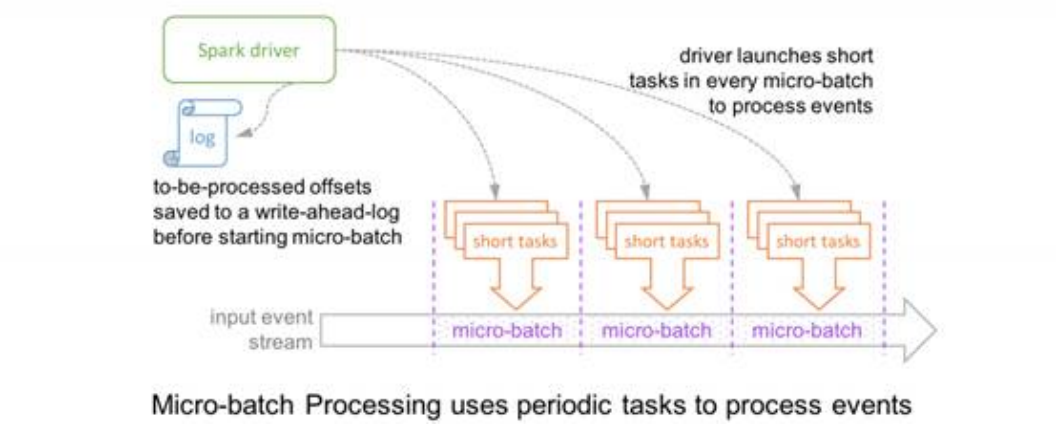
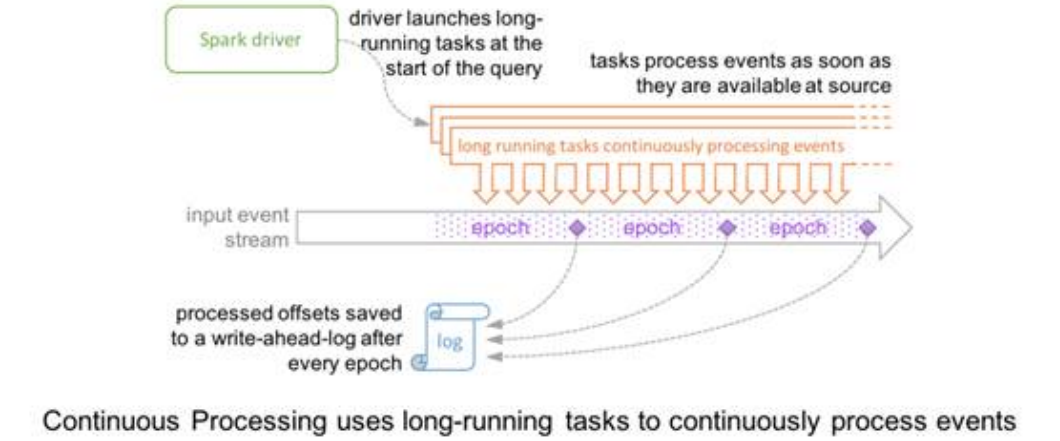


Figure 28-7 Running process in continuous mode



28.1.3 Development Process

Development Process of a Spark Application

Spark includes Spark Core, Spark SQL and Spark Streaming, whose development processes are the same.

All stages of the development process are shown and described in **Figure 28-8** and **Table 28-2**.

Figure 28-8 Spark development process

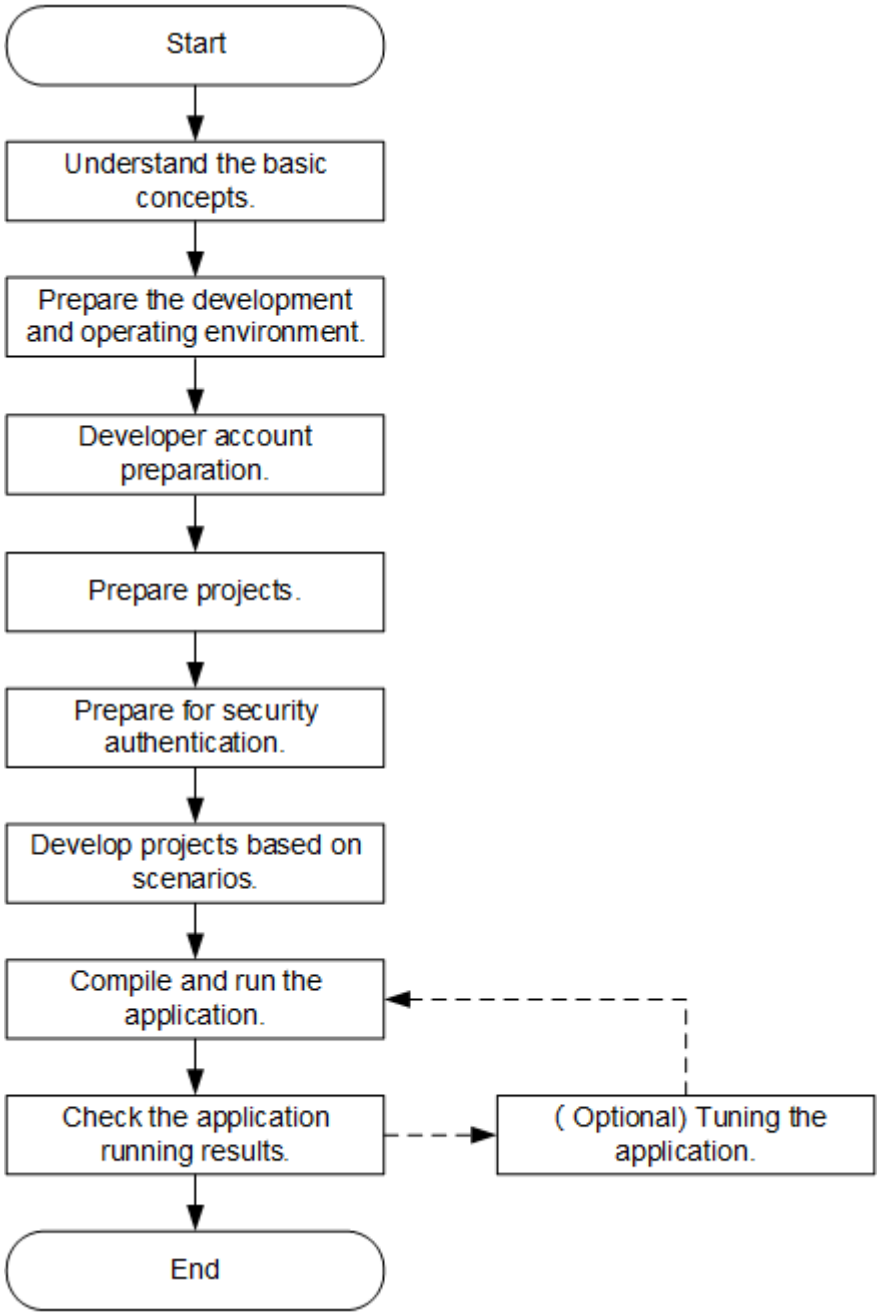


Table 28-2 Description of Spark development process

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.	**Basic Concepts**

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.	**Preparing for Development and Operating Environment**
Developer account preparation	Only the developer user with permission to access HDFS, YARN, Kafka, and Hive is allowed to run Spark sample projects.	**Preparing the Developer Account**
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.	**Configuring and Importing Sample Projects** **Creating a New Project (Optional)**
Prepare for safety authentication.	If a safe cluster is used, the safety authentication must be performed.	**Preparing for Security Authentication**
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.	**Developing the Project**
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	**Compiling and Running the Application**
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.	**Checking the Commissioning Result**
Tune the application.	Based on the application running results, tun the application to meet the requirements of the service scenario. After application tuning, compile and run the application again.	**Spark2x Performance Tuning**

28.2 Preparing for the Environment

28.2.1 Preparing for Development and Operating Environment

- Spark2x applications can be developed in Scala, Java, and Python. **Table 28-3** describes the development and operating environment to be prepared.

Table 28-3 Development environment

Preparation Item	Description
OS	• 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 runtime environment must be able to communicate with the cluster service plane network.
JDK installation	Basic configuration for the Java/Scala 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. • For x86 nodes that run clients, the following JDKs can be used: – 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: – OpenJDK: 1.8.0_272 **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 com.ibm.jsse2.overrideDefaultTLS 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.**

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. **NOTE** • 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.
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 2.12.14.
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 **Preparing the Developer Account** for configuration.
7-zip	Used to decompress **.zip** and **.rar** packages. The 7-Zip 16.04 is supported.
Python installation	Its version must be 3.7 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 **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\Spark2x\config** and manually import the configuration file to the configuration file directory (usually the **resources** folder) of the Spark sample project.

The keytab file obtained during the **Preparing the Developer Account** is also stored in this directory. **Table 28-4** describes the main configuration files.

Table 28-4 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 on the node. For example, install the client in the directory **/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/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 the Developer Account** is also stored in this directory. **Table 28-5** describes the main configuration files.

Table 28-5 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.

28.2.2 Configuring and Importing Sample Projects

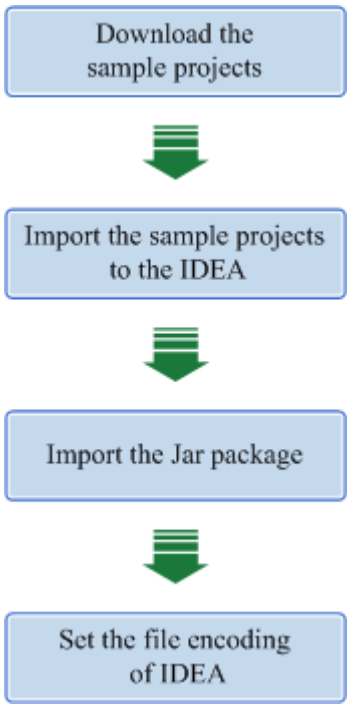
Scenario

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 28-9** shows the procedure of importing sample projects.

Figure 28-9 Procedure of importing sample projects



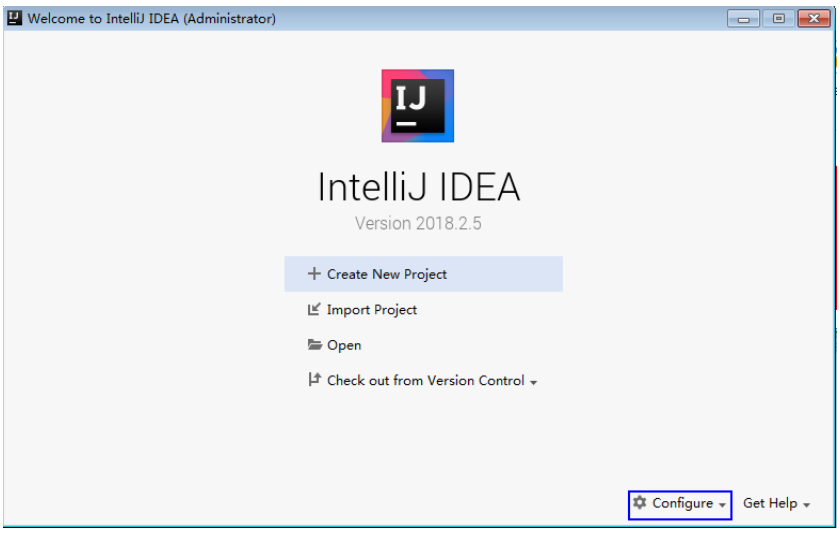
Procedure

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.

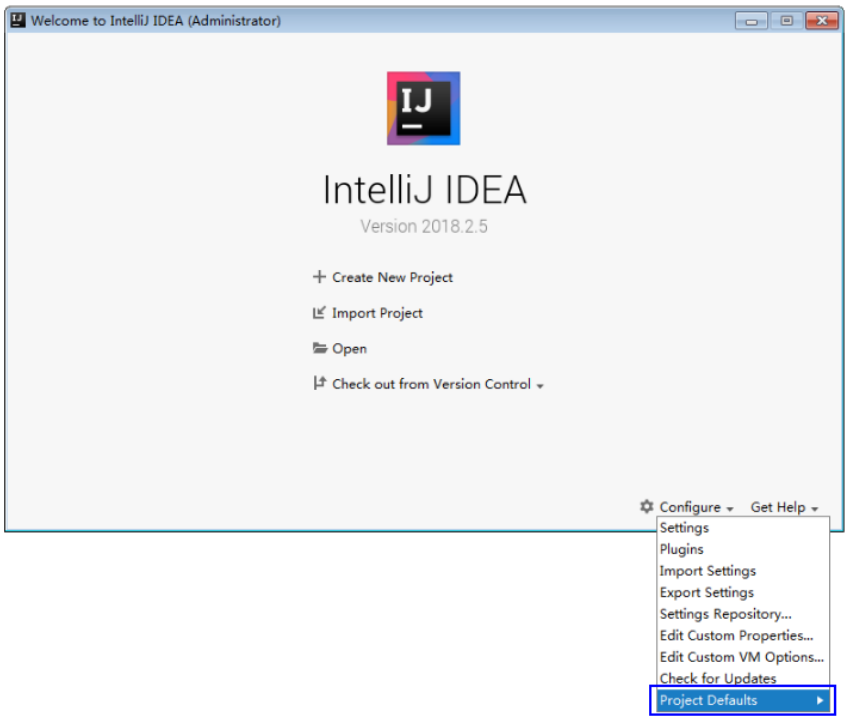
1. Start the IntelliJ IDEA and select **Configure**.

Figure 28-10 Quick Start



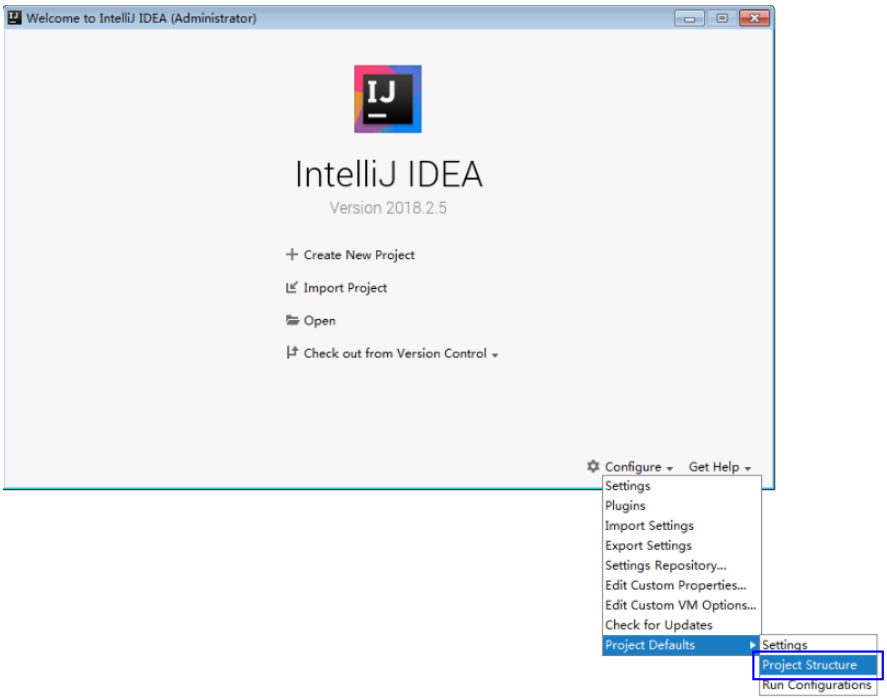
2. Select **Project Defaults** from the **Configure** drop-down list.

Figure 28-11 Configure



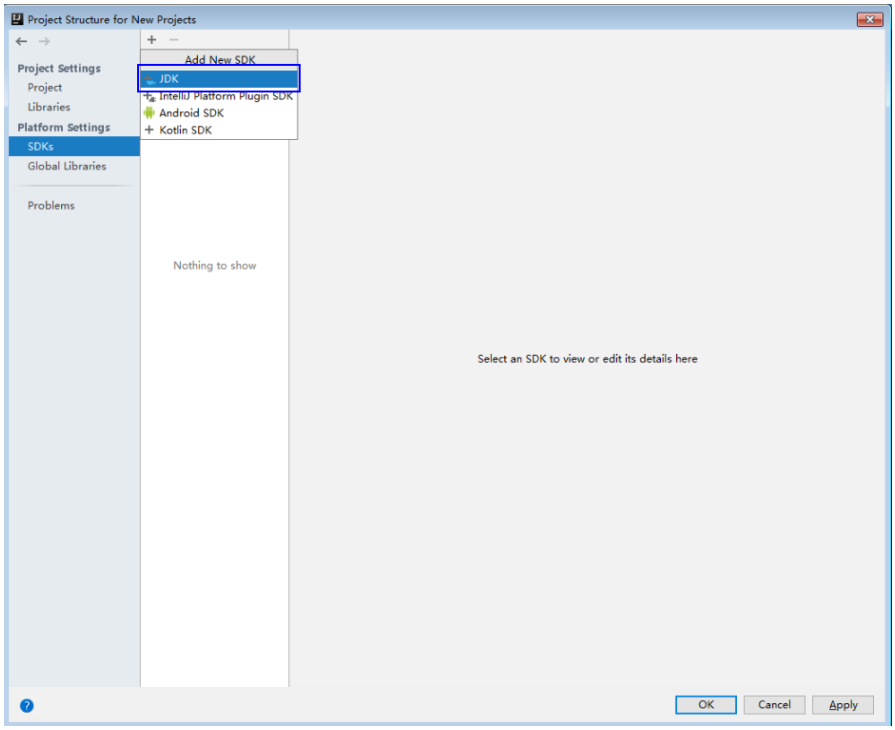
3. Select **Project Structure** from **Project Defaults**.

Figure 28-12 Project Defaults



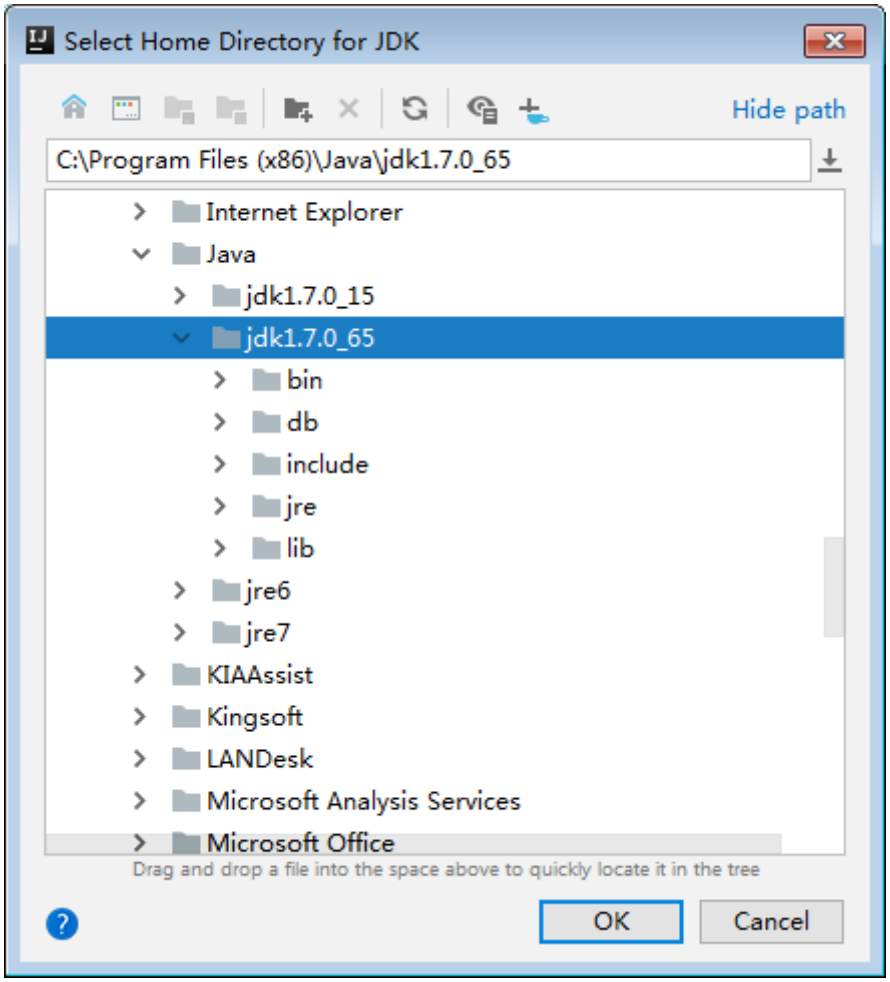
4. On the displayed **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 28-13 Add the JDK



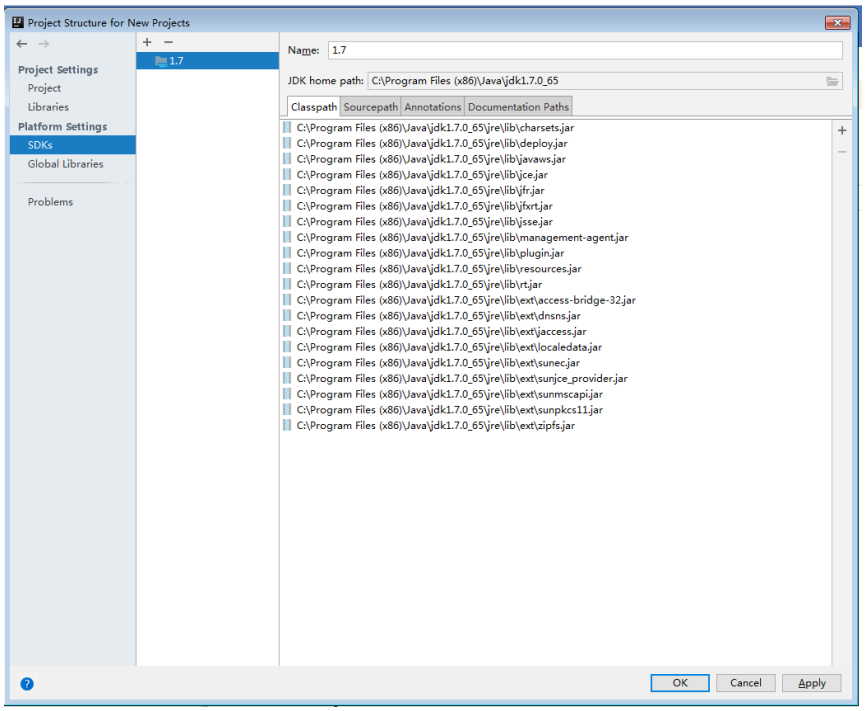
5. In the displayed **Select Home Directory for JDK** window, select a home directory for JDK and click **OK**.

Figure 28-14 Select a home directory for the JDK



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

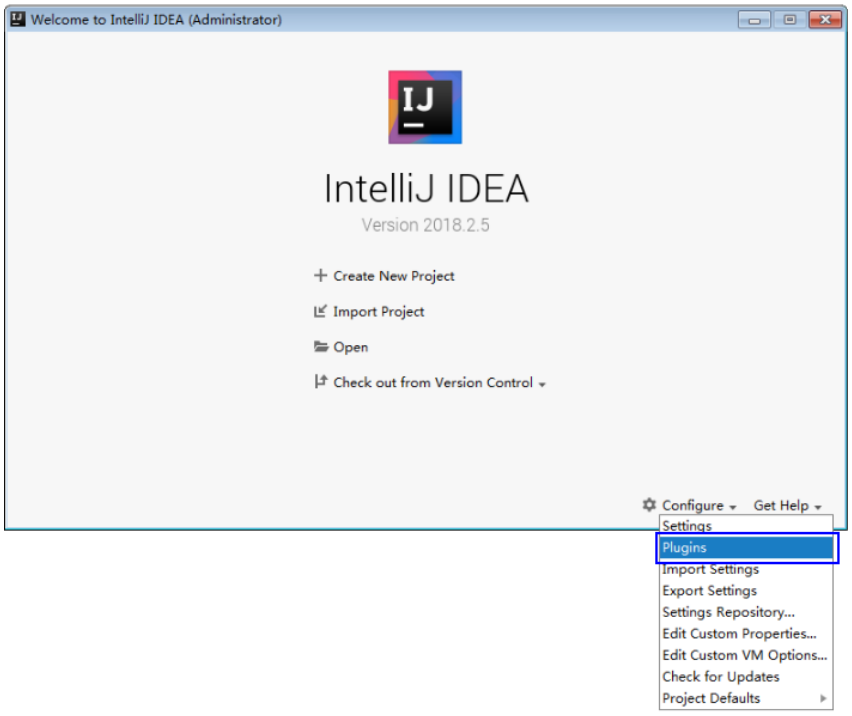
Figure 28-15 Complete the configuration



Step 3 (Optional) If the Scala development environment is used, install the Scala plug-in in IntelliJ IDEA.

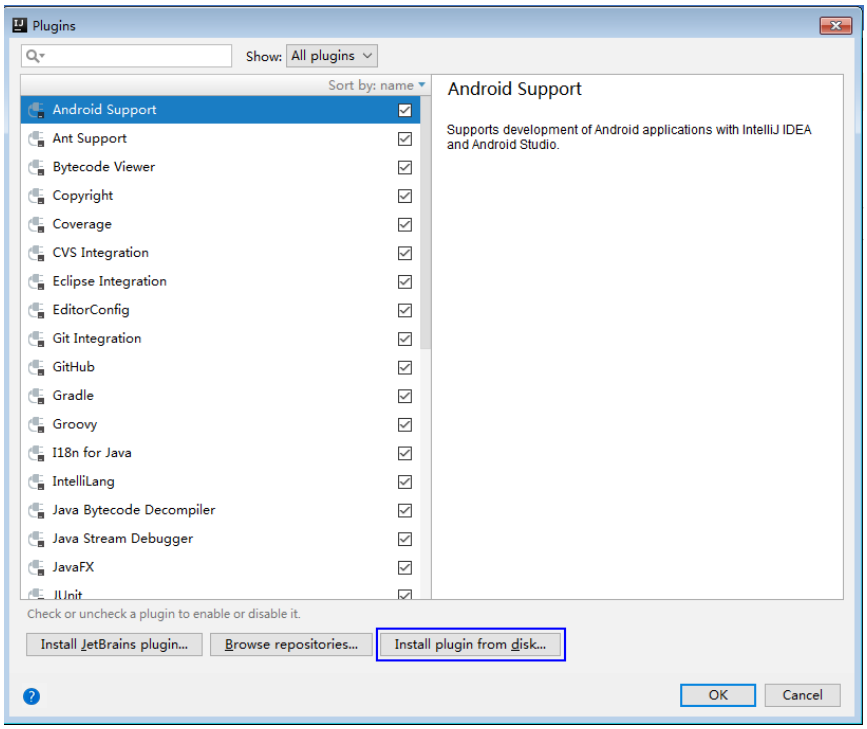
1. Select **Plugins** from the **Configure** drop-down list.

Figure 28-16 Plugins

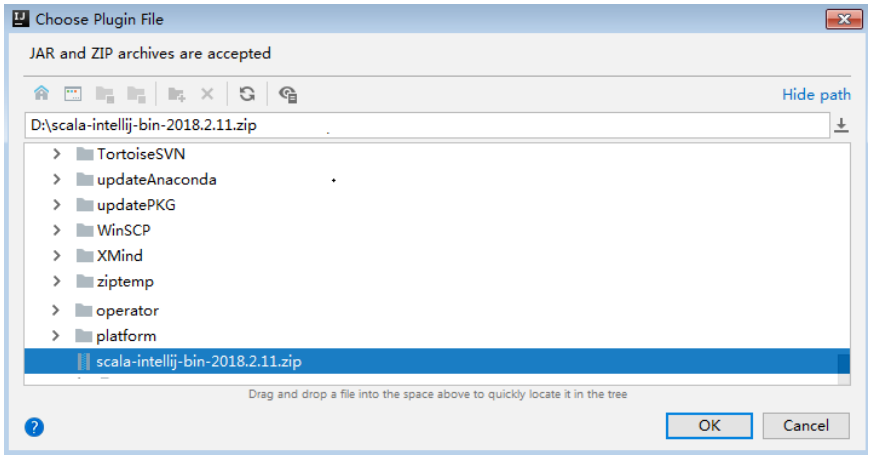


2. On the **Plugins** page, select **Install plugin from disk**.

Figure 28-17 Install plugin from disk

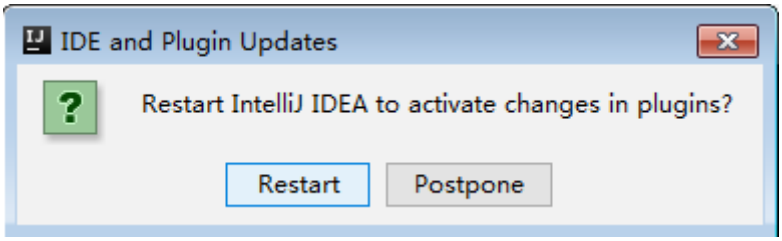


3. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.



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 28-18 Plugins Changed

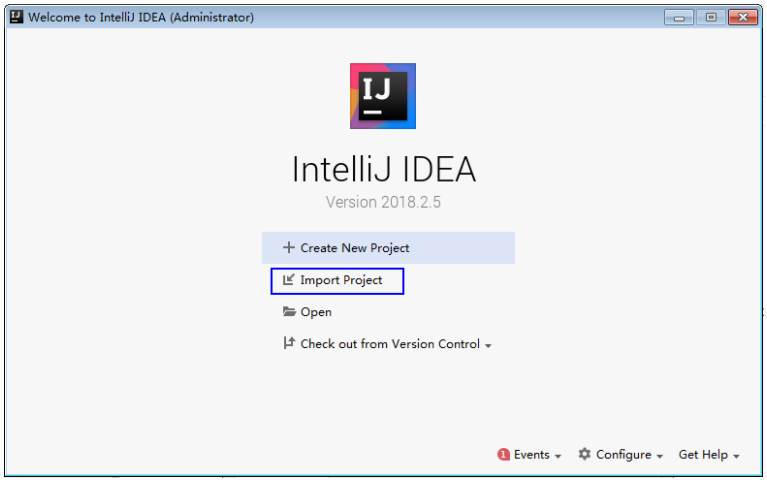


Step 4 Import the Java sample projects to the IDEA.

1. Start the IntelliJ IDEA, select **Import Project** on the **Quick Start** page.

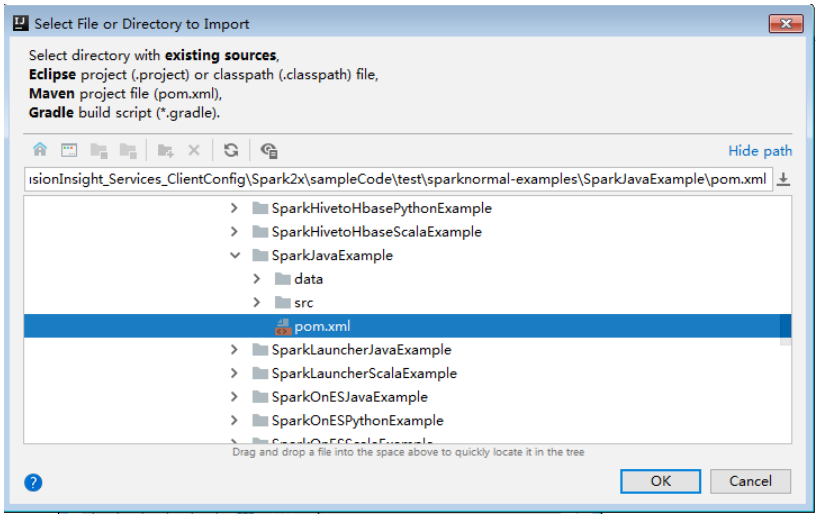
   Or, for the used IDEA tool, add projects directly from the IDEA homepage. Select **File** > **Import project...** to import projects.

   Figure 28-19 Import Project (on the Quick Start page)

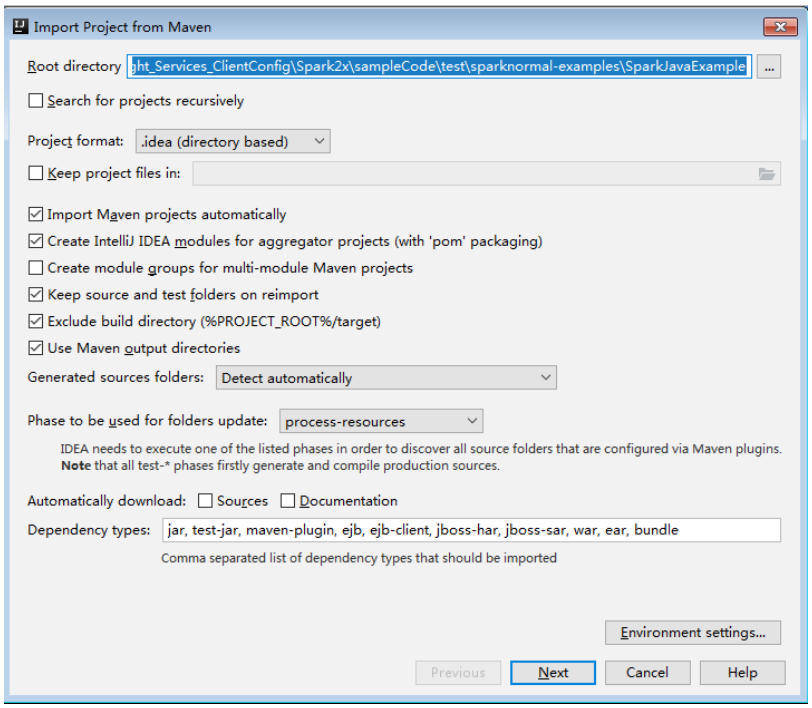
2. Select the directory to store the imported projects and the pom file, and click **OK**.

   Figure 28-20 Select File or Directory to Import

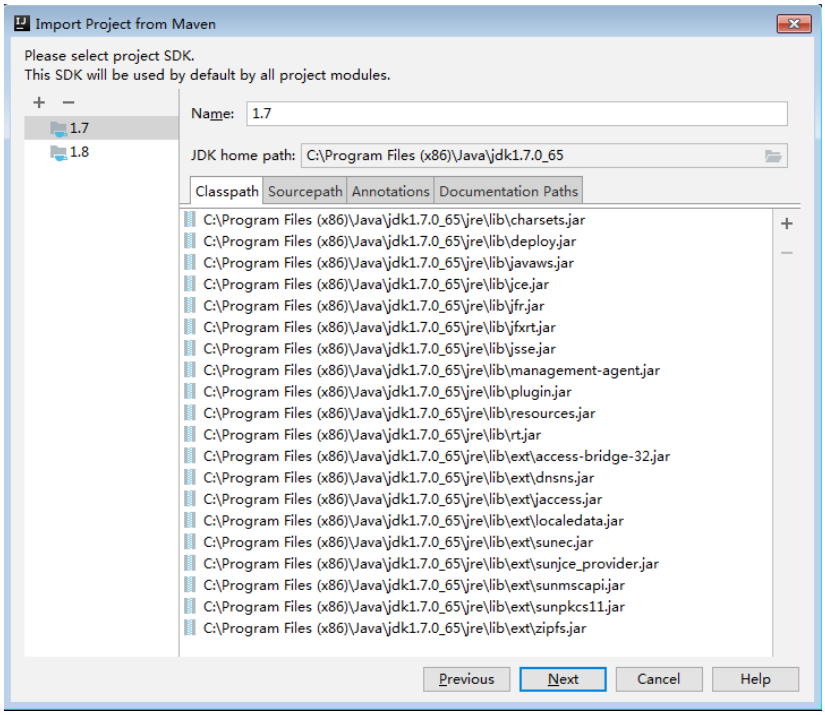
3. Confirm the import directory and project name, and click **Next**.

Figure 28-21 Import Project from Maven



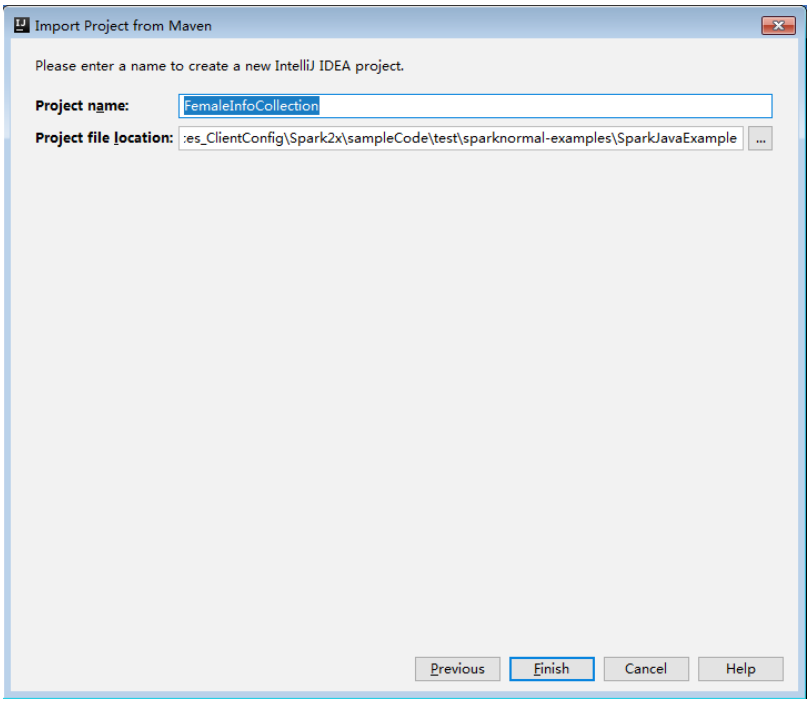
4. Select the projects to import and click **Next**.
5. Confirm the project JDK and click **Next**.

Figure 28-22 Select project SDK



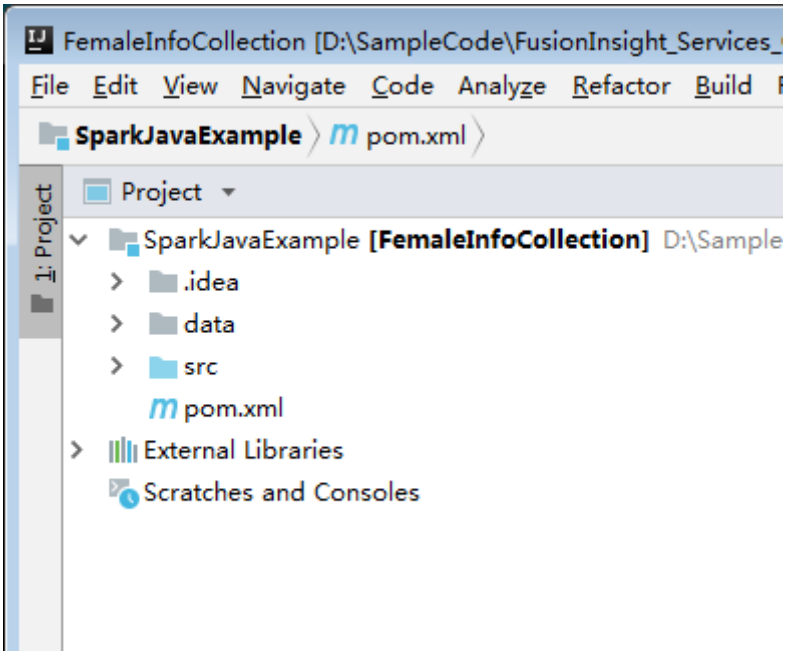
6. Confirm the project name and project file location, and click **Finish** to complete the import.

Figure 28-23 Confirm the project name and file location



7. After the import, the imported projects are displayed on the IDEA homepage.

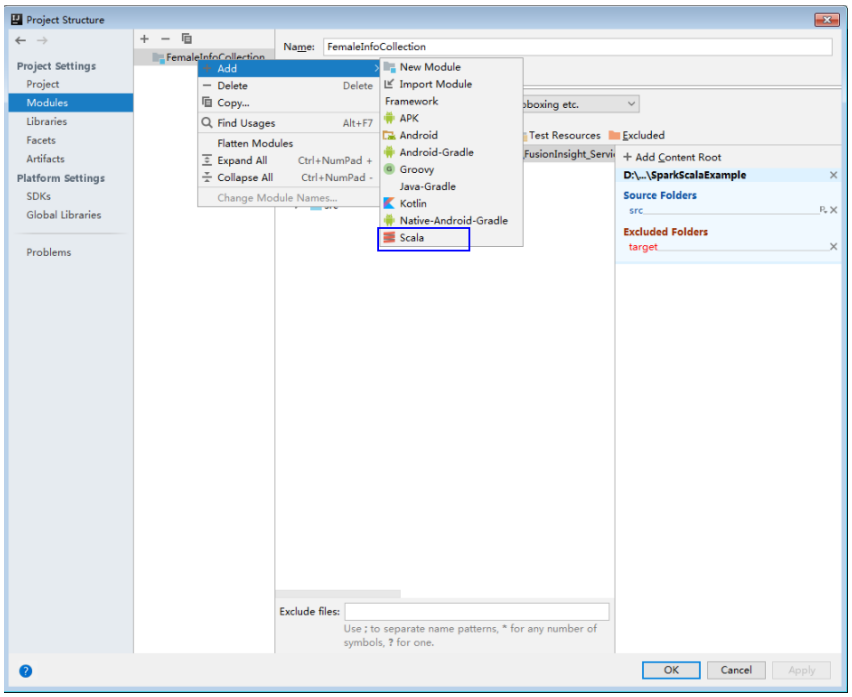
Figure 28-24 Imported project



Step 5 (Optional) If a sample application developed in Scala is imported, configure the language for the project.

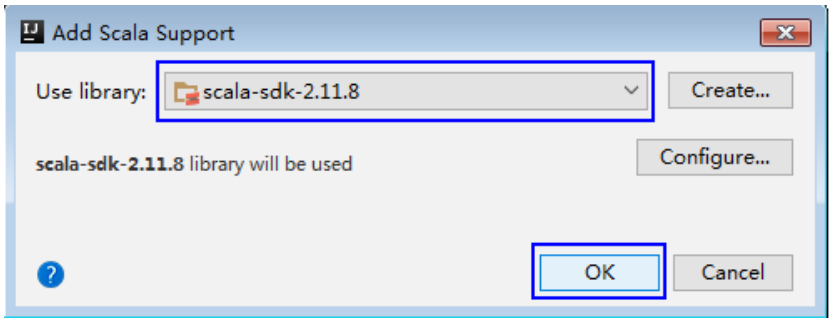
1. On the main page of the IDEA, choose **File** > **Project Structures...**to access the **Project Structure** page.
2. Choose **Modules**, right-click the project name, and choose **Add** > **Scala**.

Figure 28-25 Select the Scala language



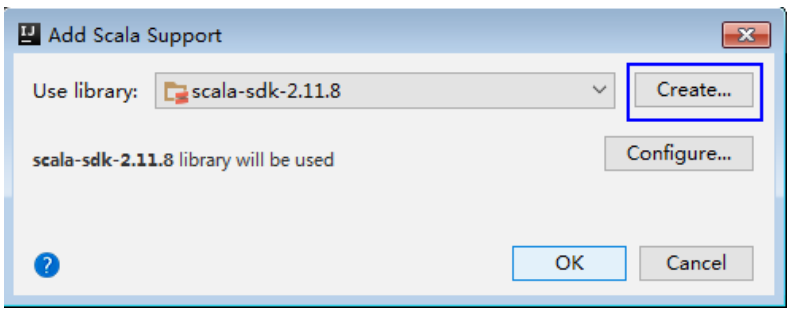
3. Wait until IDEA identifies Scala SDK, select the dependency JAR packages in the **Add Scala Support** dialog box, and then click **OK**

Figure 28-26 Add Scala Support



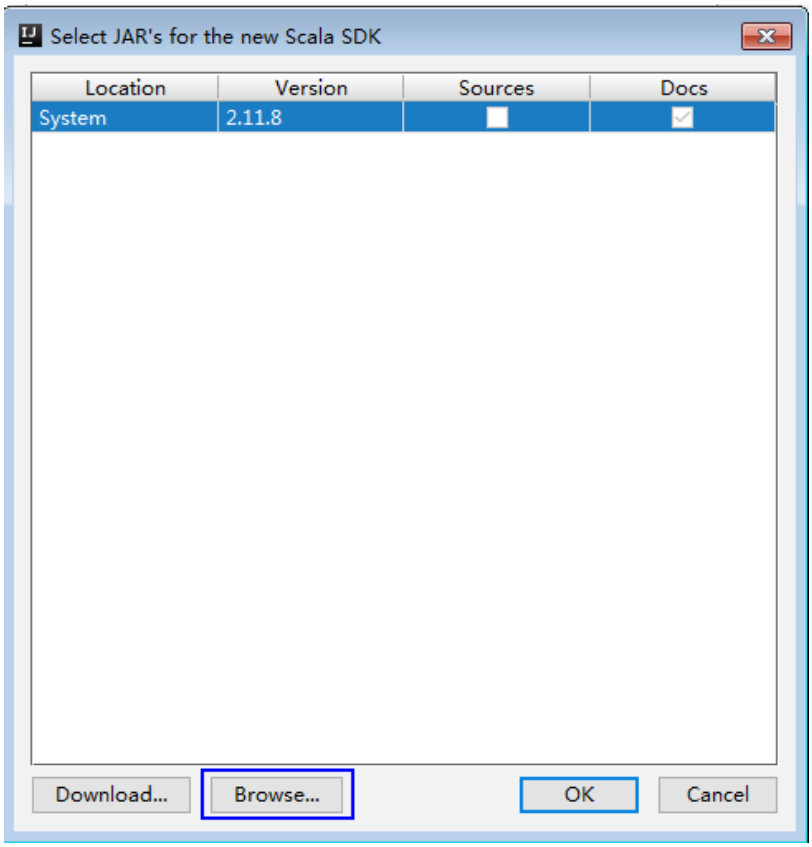
4. If IDEA fails to identify Scala SDK, you are required to create a Scala SDK.
   a. Click **Create**...

   Figure 28-27 Create...



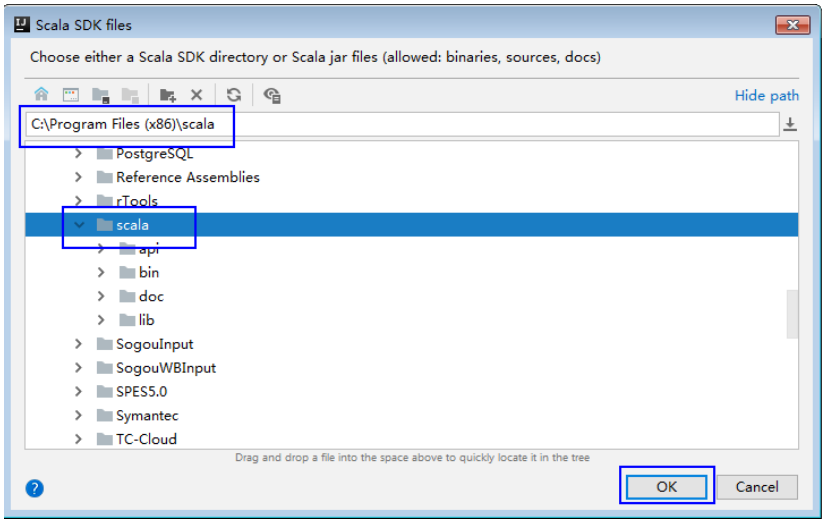
   b. On the **Select JAR';s for the new Scala SDK** page, click **Browse**...

Figure 28-28 Select JAR's for the new Scala SDK



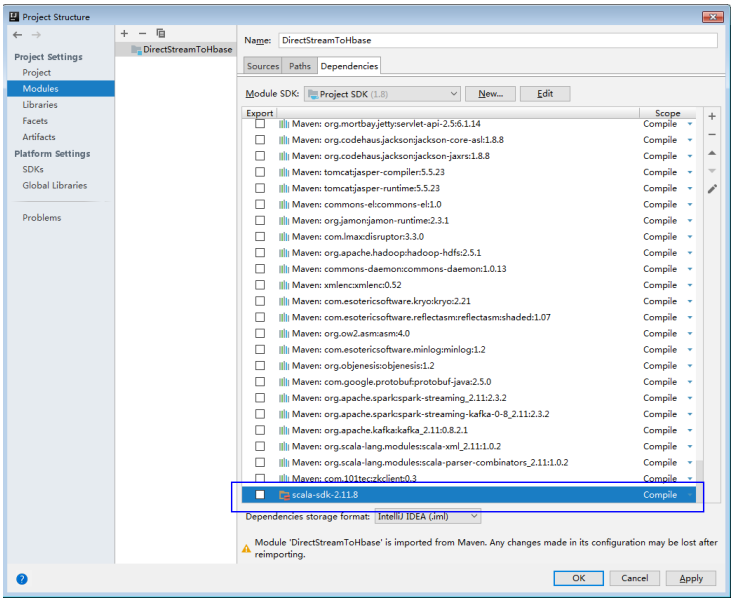
c. In the **Scala SDK files** window, select the scala sdk directory, and then click **OK**.

Figure 28-29 Scala SDK files



5. Click **OK**.

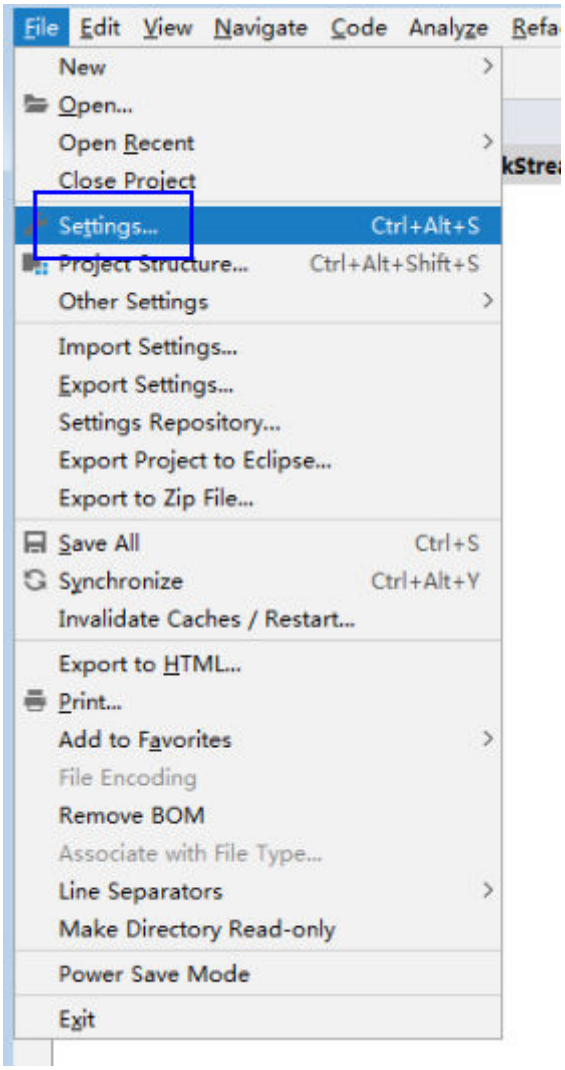
Figure 28-30 Successful configuration



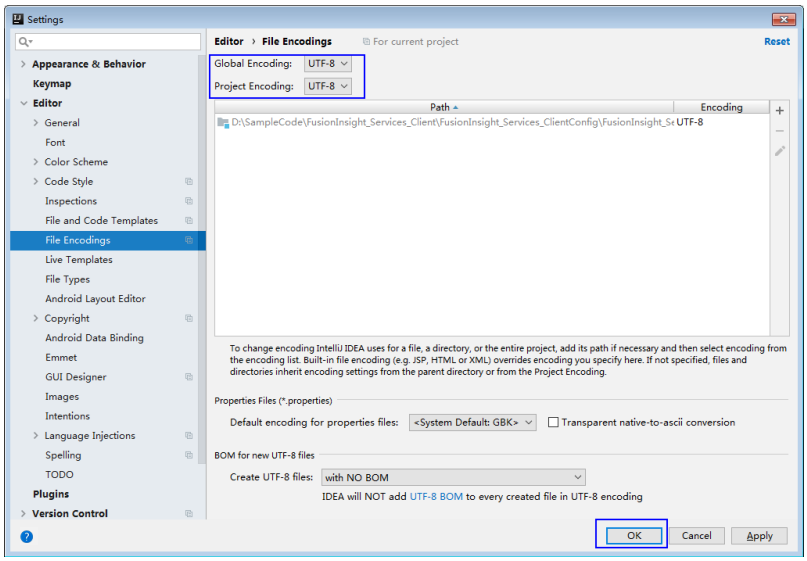
Step 6 Set the file encoding of IDEA and solve the display of garble characters.

1. On the IDEA homepage, choose **File** > **Settings...**.

Figure 28-31 Choose 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 28-6 Sample Code Path Description

Sample code project	Sample Name	Sample Development Language
SparkJavaExample	Spark Core Project	Java
SparkScalaExample	Spark Core Project	Scala
SparkPyhtonExample	Spark Core Project	Python
SparkSQLJavaExample	Spark SQL Project	Java
SparkSQLScalaExample	Spark SQL Project	Scala
SparkSQLPythonExample	Spark SQL Project	Python
SparkStreamingJavaExample	Streaming Connecting to Kafka0-8	Java
SparkStreamingScalaExample	Streaming Connecting to Kafka0-8	Scala
SparkStreamingPythonExample	Streaming Connecting to Kafka0-8	Python
SparkThriftServerJavaExample	Accessing the Spark SQL Through JDBC	Java
SparkThriftServerScalaExample	Accessing the Spark SQL Through JDBC	Scala

Sample code project	Sample Name	Sample Development Language
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
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

Sample code project	Sample Name	Sample Development Language
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
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

Sample code project	Sample Name	Sample Development Language
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
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
SparkRExample	Install SparkR	R
SparkOnHudiJavaExample	Using Spark to Perform Basic Hudi Operations	Java

Sample code project	Sample Name	Sample Development Language
SparkOnHudiPythonExample	Using Spark to Perform Basic Hudi Operations	Python
SparkOnHudiScalaExample	Using Spark to Perform Basic Hudi Operations	Scala

28.2.3 Creating a New Project (Optional)

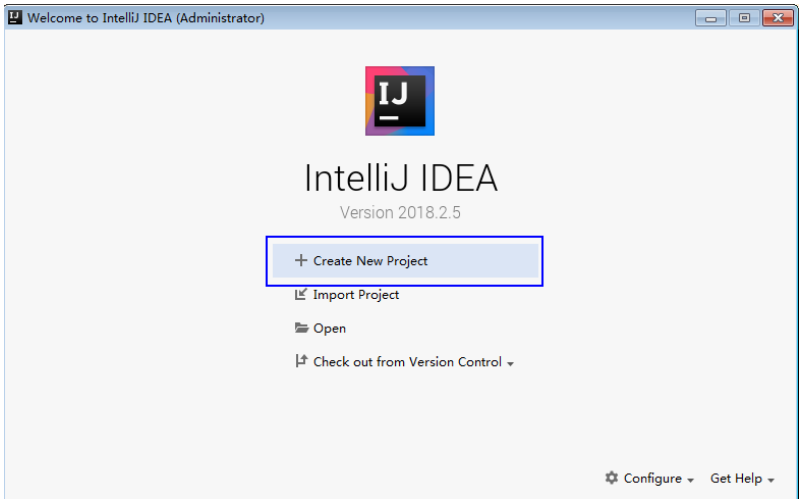
Scenario

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**.

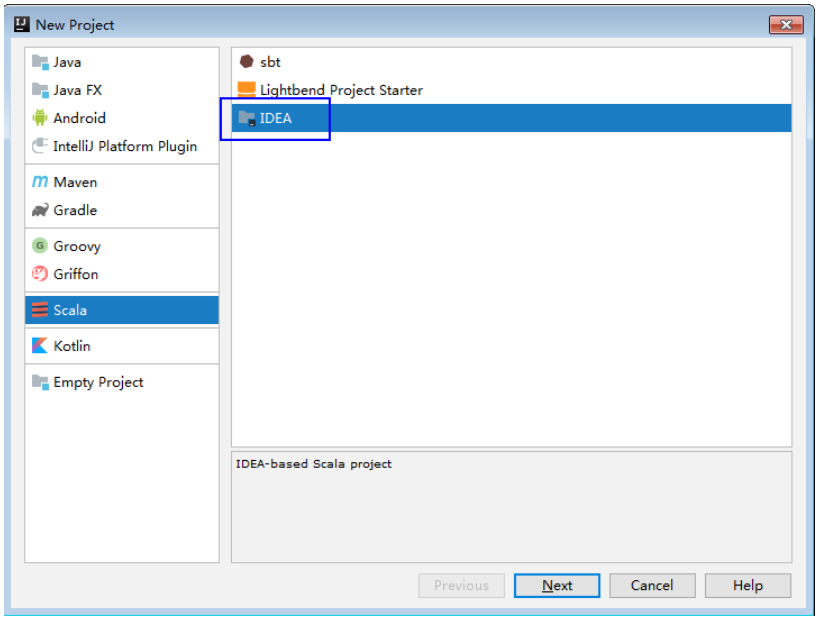
Figure 28-32 Create a project



Step 2 On the **New Project** page, select **Scala** as the development environment, select **IDEA**, and click **Next**.

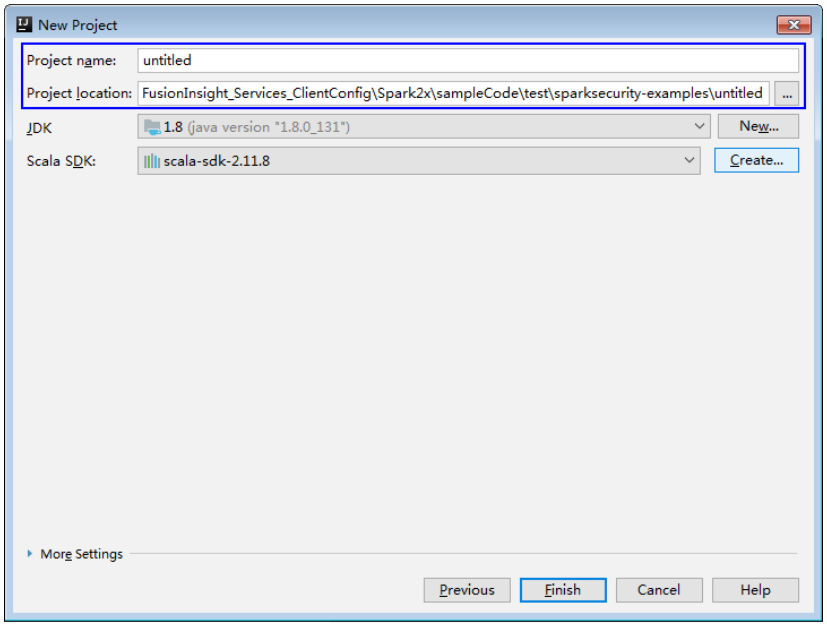
If a new project in Java is required, choose corresponding parameters.

Figure 28-33 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 28-34 Add the project information



----End

28.2.4 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.

Two security authentication methods are described as follows:

- 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 28-7 lists the authentication method for the example code in security cluster environment.

Table 28-7 Authentication methods

Example Code	Mode	Authentication Method
sparknormal-examples	yarn-client	By running command lines, configuring parameters, or adding code.
	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 example 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 28-8**. Modify the parameter value as required.

Table 28-8 Parameter Description

Parameter	Example Parameter Value	Description
userPrincipal	sparkuser	Users use the authenticated account Principal, Use the developer account prepared in **Preparing the Developer Account**.
userKeytabPath	/opt/FIclient/user.keytab	Users use the authenticated Keytab file, Copy the **user.keytab** 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 example 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 28-9**. Modify the parameter value as required.

Table 28-9 Parameter Description

Parameter	Example Parameter Value	Description
userPrincipal	sparkuser	Users use the authenticated account Principal, Use the developer account prepared in **Preparing the Developer Account**.
userKeytabPath	/opt/FIclient/user.keytab	Users use the authenticated Keytab file, Copy the **user.keytab** 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);
```

28.2.5 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

28.3 Developing the Project

28.3.1 Spark Core Project

28.3.1.1 Instance

Instance Description

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).

Step 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**.

Step 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**:

1. On the HDFS client, run the following commands to obtain the safety authentication:

   cd */opt/hadoopclient*

   source bigdata_env

   kinit *<component service user>*

2. On the Linux FusionInsight client, run ***hadoop fs -mkdir /tmp/input*** (a hdfs dfs command provides the same function).
3. Go to the /**tmp/input** directory on the HDFS client, on the Linux FusionInsight client, run ***hadoop fs -put*** *input_data1.txt /tmp/input* and ***hadoop fs -put*** *input_data2.txt /tmp/input*.

----End

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 example 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/FIclient/user.keytab --principal sparkuser */opt/female/SparkPythonExample/collectFemaleInfo.py <inputPath>*

  <inputPath> indicates the input path in HDFS.

28.3.1.2 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());
   }
```

28.3.1.3 Scala 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.

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()
```

28.3.1.4 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()
```

28.3.2 Spark SQL Project

28.3.2.1 Instance

Instance Description

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).

Step 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**.

Step 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**:

1. On the HDFS client, run the following commands to obtain the safety authentication:

   cd */opt/hadoopclient*

   source bigdata_env

   kinit *<component service user>*

2. On the Linux FusionInsight client, run ***hadoop fs -mkdir /tmp/input*** (a hdfs dfs command provides the same function).
3. Go to the **/tmp/input** directory on the HDFS client, on the Linux FusionInsight client, run ***hadoop fs -put*** *input_data1.txt /tmp/input* and ***hadoop fs -put*** *input_data2.txt /tmp/input*.

----End

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 example 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.

28.3.2.2 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 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**

28.3.2.3 Scala 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 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**

28.3.2.4 Python Example Code

Function

Collect information about female netizens who have spent more than 2 hours in online shopping on the weekend.

Example 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()
```

28.3.3 Accessing the Spark SQL Through JDBC

28.3.3.1 Instance

Scenario

Spark on HBase allows users to create HBase tables on the JDBCServer, store data to JDBCServer tables by running the HBase command, and perform other operations.

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.

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 example 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 example 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 example 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*

28.3.3.2 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.

```
String securityConfig = ";saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.<system domain name>@<system domain name>;user.principal=sparkuser;user.keytab=/opt/FIclient/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.

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

28.3.3.3 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 securityConfig = ";saslQop=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.

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

28.3.4 Spark on HBase

28.3.4.1 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 example 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 --deploy-mode cluster --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.
"""
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().
```

28.3.4.2 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.

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 example 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 threquired 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.
"""
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().
```

28.3.4.3 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 example 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

- 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.JavaHBaseBulkPutExample 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.JavaHBaseBulkPutExample --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.
"""
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()
```

28.3.4.4 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 **3.5.3 Using 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 example 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.)

  bin/spark-submit --master yarn --deploy-mode client --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkGetExample 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.JavaHBaseBulkGetExample --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.
"""
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()
```

28.3.4.5 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 **3.5.3 Using 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 example 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.)

  bin/spark-submit --master yarn --deploy-mode client --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkDeleteExample 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 HBaseButDeleteExample.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.JavaHBaseBulkDeleteExample --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 HBaseButDeleteExample.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, esure that the spark.yarn.security.credentials.hbase.enabledparameter in the spark-defaults.conffile under Spark2x/spark/conf/ is set to true on the Spark2x client.
"""
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()
```

28.3.4.6 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 example 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.)

  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.
"""
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()
```

28.3.4.7 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.

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 example 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.)

  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.enabledparameter in the spark-defaults.conffile under Spark2x/spark/conf/ is set 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()
```

28.3.4.8 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 **3.5.1 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 example 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.)

  bin/spark-submit --master yarn --deploy-mode client --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseDistributedScanExample 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.JavaHBaseDistributedScanExample --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.
"""
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()
```

28.3.4.9 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 **3.5.7 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 example 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.)

  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.JavaHBaseMapPartitionExample --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.
"""
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()
```


28.3.4.10 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 example 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.JavaHBaseStreamingBulkPutExample --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.
"""
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 insatnce and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseStreamingBulkPutExample().execute(spark._jsc, sys.argv)
# Stop the SparkSession instance.
spark.stop()
```

28.3.5 Reading Data from HBase and Write It Back to HBase

28.3.5.1 Instance

Scenario

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 user1 in the current day is 100 CNY.

In table2, key=1 and cf:cid=1000 indicate that the history consumption amount of user1 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 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 example 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**.
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 example code.

  bin/spark-submit --jars *{Client path}***/Spark/spark/jars/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*

28.3.5.2 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"));//colomn 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 colomn family and colomn 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();
      }
    }
  }
 }

}
```

28.3.5.3 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")) //colomn 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 colomn family and colomn 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 MyRegistrator extends KryoRegistrator {
  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])
  }
}
```

28.3.5.4 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.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()
```

28.3.6 Reading Data from Hive and Write It to HBase

28.3.6.1 Instance

Scenario

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 user1 in the current day is 100 CNY.

In table2, key=1 and cf:cid=1000 indicate that the history consumption amount of user1 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 *<component service user>*

      kinit the user to be *<authenticated>*

b. On a Linux-based HDFS client, run the ***hadoop fs -mkdir /tmp/input*** command (or ***hdfs dfs -mkdir /tmp/input***) to create the **/tmp/input** directory.

c. On a Linux-base HDFS client, run ***hadoop fs -put*** *log1.txt /tmp/input* command to import data files.

Step 2 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 example 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**.
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 example 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*

28.3.6.2 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.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
          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();
        }
      }
    }
  }
}
```


28.3.6.3 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.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()
      }
    }
  }
 }

}
```

28.3.6.4 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()
```

28.3.7 Streaming Connecting to Kafka0-10

28.3.7.1 Instance

Scenario

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 include the absolute path of the Kafka JAR package of the Spark client, for example: /opt/client/Spark2x/spark/jars/*:/opt/client/Spark2x/spark/jars/streamingClient010/*

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 example 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
- kafka_2.12-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 the dependency package whose version number does not contain "hw-ei", obtain them from the Maven central repository.

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):

- Example 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.
  ```

- Spark Streaming Write To Kafka 0-10 example 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.JavaDstreamKafkaWriter /opt/SparkStreamingKafka010JavaExample-1.0.jar <groupId> <brokers> <topics>

28.3.7.2 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("sasl.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> sentData = new ArrayList();
   sentData.add("kafka_writer_test_msg_01");
   sentData.add("kafka_writer_test_msg_02");
   sentData.add("kafka_writer_test_msg_03");

   // Create Java RDD queue
   Queue<JavaRDD<String>> sent = new LinkedList();
   sent.add(ssc.sparkContext().parallelize(sentData));

   // 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();
```



```
  }
}
```

28.3.7.3 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.

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()
 }
}
```

28.3.8 Structured Streaming Project

28.3.8.1 Instance

Scenario

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 package 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 example 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> where **<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 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 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 example 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 example 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>

28.3.8.2 Java Example 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();
 }
}
```

28.3.8.3 Scala Example 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(_.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()
 }
}
```

28.3.8.4 Python Example 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.

Example 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()
```

28.3.9 Structured Streaming Stream-Stream Join

28.3.9.1 Overview

Scenario

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 section "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 example 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.

- Add configuration items to **$SPARK_HOME/conf/jaas.conf**:

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=false
useTicketCache=true
debug=false;
};
```

- 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;
};
```

- 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 cope the kafka-clientsjar 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.
```


28.3.9.2 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)
println(showStaticsQuery.status)
          println("-------------get clickStatic progress---------")
          //println(clickStaticsQuery.lastProgress)
println(clickStaticsQuery.status)
          Thread.sleep(10000)
        }
      }
    }).start

    spark.streams.awaitAnyTermination()

  }
}
```

28.3.10 Structured Streaming Status Operation

28.3.10.1 Scenario

Scenario

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 example 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.

- Add configuration items to **$SPARK_HOME/conf/jaas.conf**:
  ```
  KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=false
  useTicketCache=true
  debug=false;
  };
  ```
- Add configuration items to **$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;
  };
  ```
- 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 --driver-java-options "-Djava.security.auth.login.config=<local path>/jaas.conf"* ***--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 cope the kafka-clientsjar file from the **$SPARK_HOME/jars/streamingClient010** directory to the **$SPARK_HOME/jars** directory. Otherwise, the "class not found" error is reported.

28.3.10.2 Scala Sample Code

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()
}
```

28.3.11 Concurrent Access from Spark to HBase in Two Clusters

28.3.11.1 Scenario

Scenario Description

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:

```
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.

28.3.11.2 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
}
```

28.3.12 Synchronizing HBase Data from Spark to CarbonData

28.3.12.1 Instance

Instance Description

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**.
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
```

28.3.12.2 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);
}
```

28.3.13 Using Spark to Perform Basic Hudi Operations

28.3.13.1 Instance

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

1. Log in to the Spark client node and run the following commands:

   source *Client installation directory*/**bigdata_env**

   source *Client installation directory*/**Hudi/component_env**

   kinit *Hudi development user*
2. 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 project.

     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> 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 project.

     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 project.

     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.

28.3.13.2 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()}
```

28.3.13.3 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))
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)
```

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()
```

28.3.13.4 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> writeStatues = client.compact(instant.get());
   client.commitCompaction(instant.get(), writeStatues, Option.empty());
}
```

28.3.14 Compiling User-defined Configuration Items for Hudi

28.3.14.1 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<>();
      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\"},
{\"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;
  }

}
```

28.3.14.2 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>
  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;
```

```
  }
}
```

28.4 Commissioning the Application

28.4.1 Commissioning Applications on Windows

28.4.1.1 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.

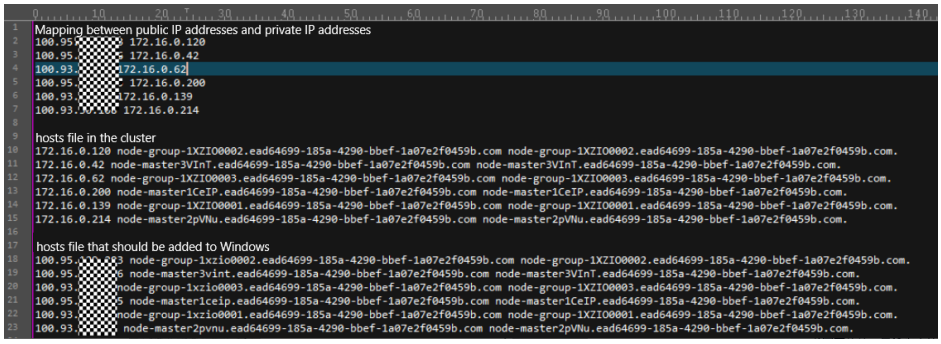
Procedure

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**.

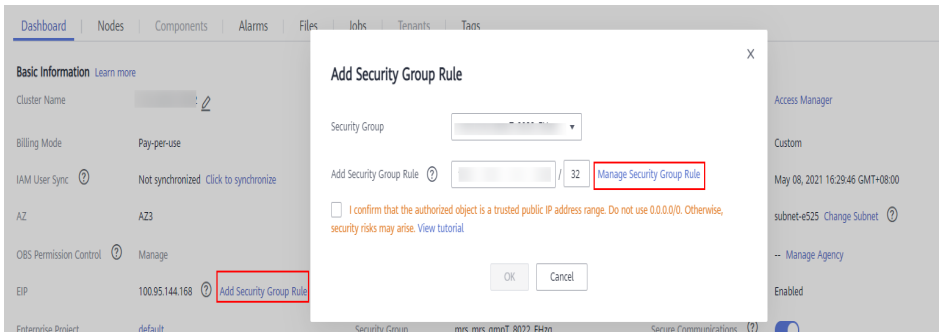
2. Record the mapping between the public IP addresses and private IP addresses. Change the private IP addresses in the **hosts** file to the corresponding public IP addresses.



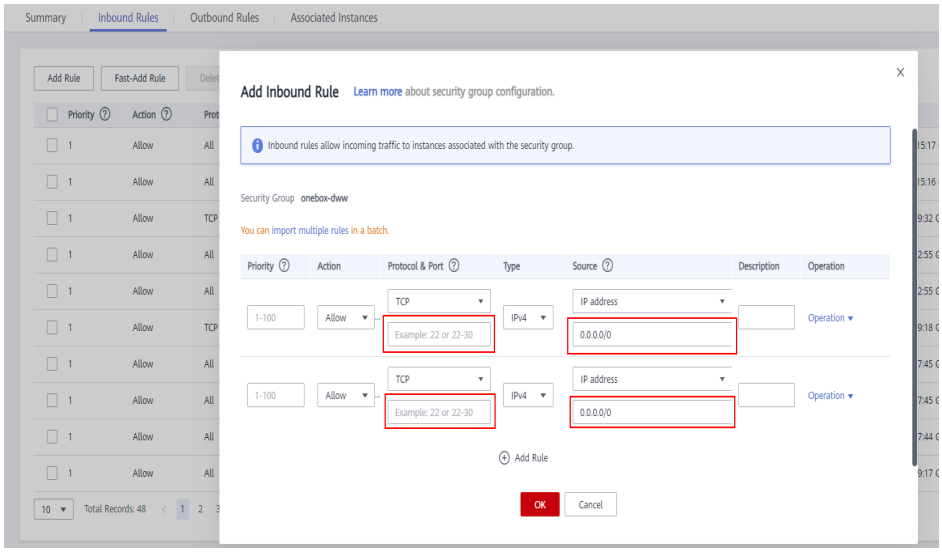
Step 2 Change the IP addresses in the **krb5.conf** file to the corresponding host names.

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.

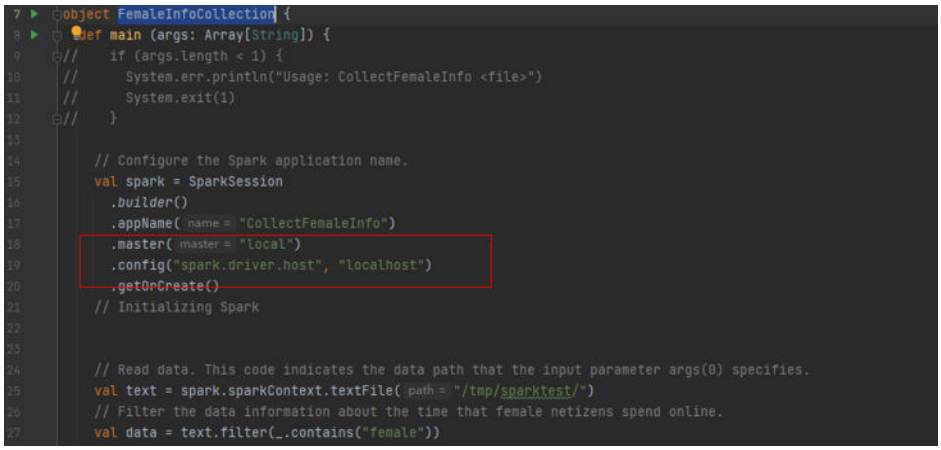
Add the following content to the **hdfs-site.xml** file:

```
<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 **PRNCIPAL_NAME** in the sample code to the username for security authentication.



----End

28.4.1.2 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_VENDER = "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";
```

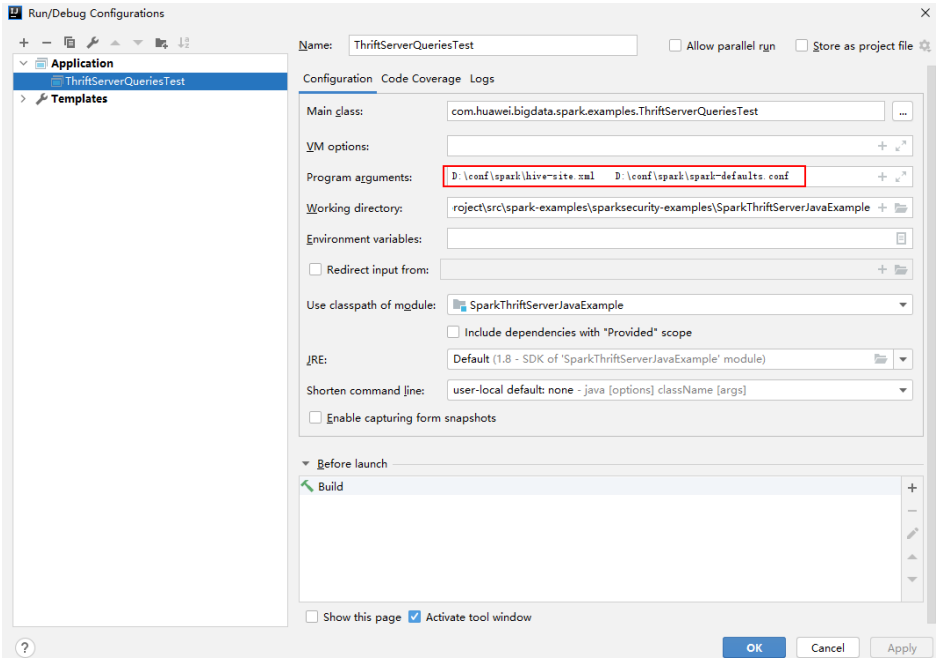
2. Change **user.principal** and **user.keytab** in the string concatenated by **securityConfig** to the corresponding username and path. Note that the path of the **keytab** file must use slashes (/).

```
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

28.4.1.3 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
```

28.4.2 Commissioning an Application in Linux

28.4.2.1 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

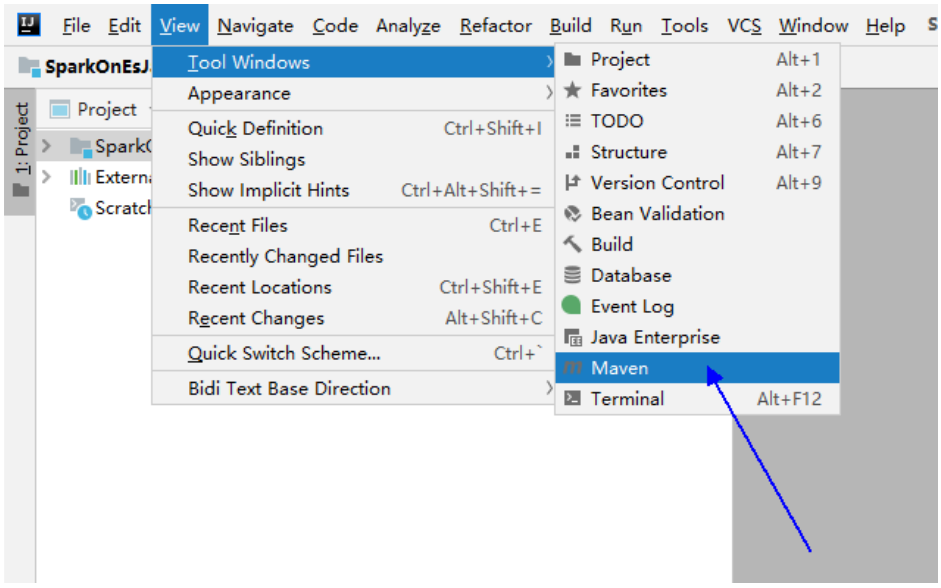
- The Spark application developed in Python does not need to build Artifacts as a jar. You just need to copy the sample projects to the compiler.
- It is needed to ensure that the version of Python installed on the worker and driver is consistent, otherwise the following error will be reported: "Python 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 **Methods for Obtaining Sample Projects**.

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.

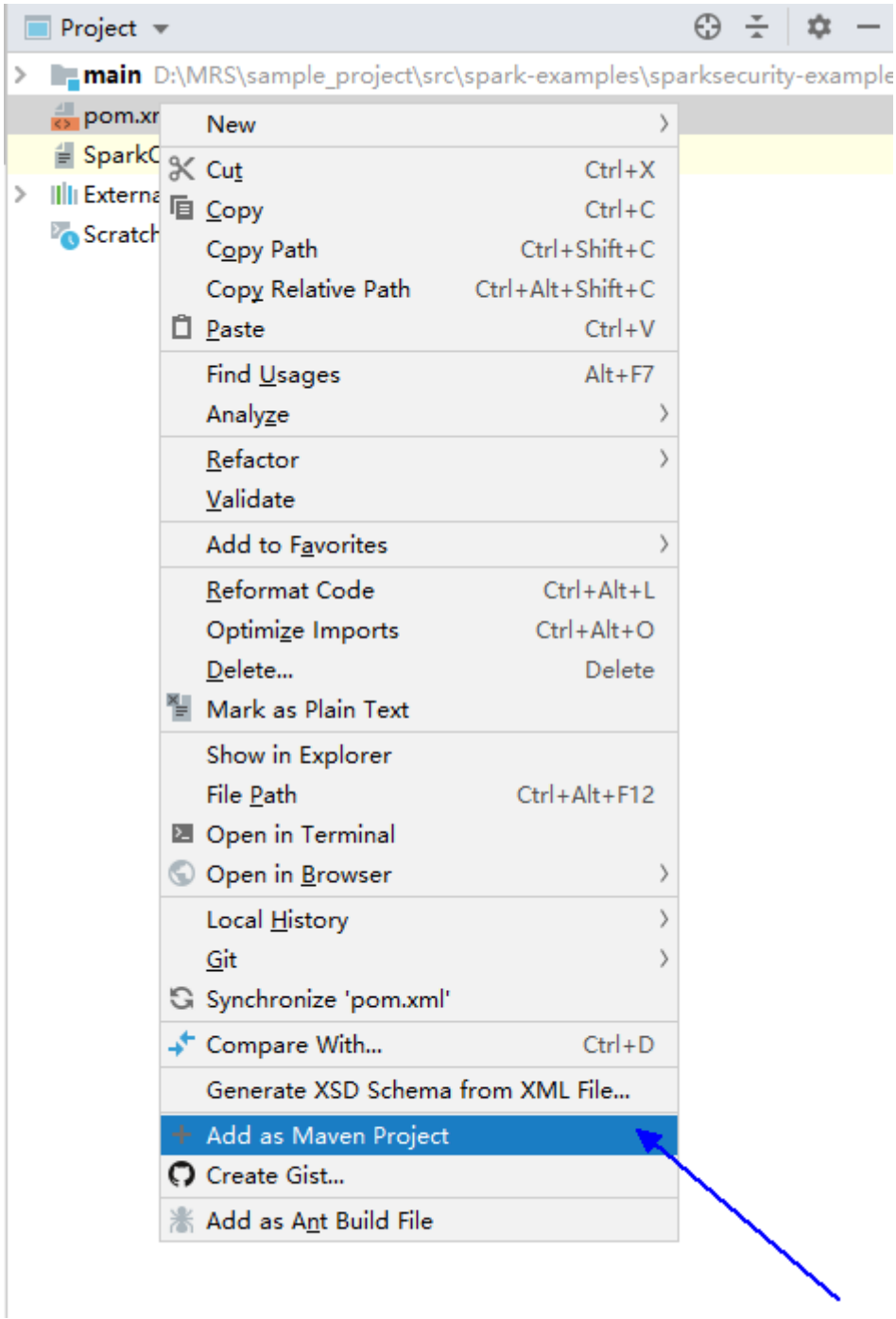
Figure 28-35 Openning the Maven tool window



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.

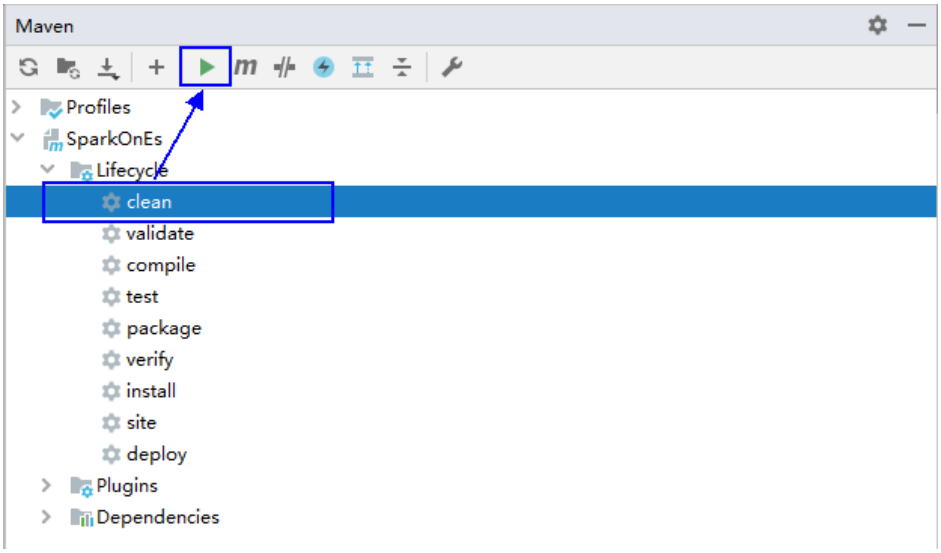
Figure 28-36 Adding a Maven project



Step 2 Use Maven to generate a JAR file.

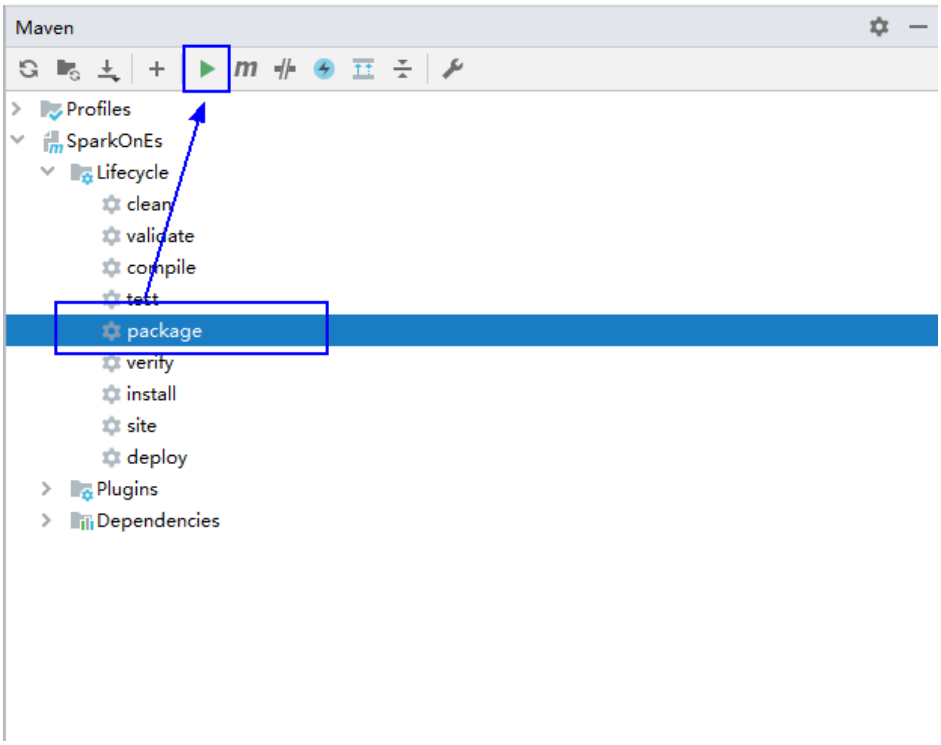
1. In the Maven tool window, select **clean** from **Lifecycle** to execute the Maven building process.

Figure 28-37 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 28-38 Selecting **package** from **Lifecycle** and execute the Maven build process.



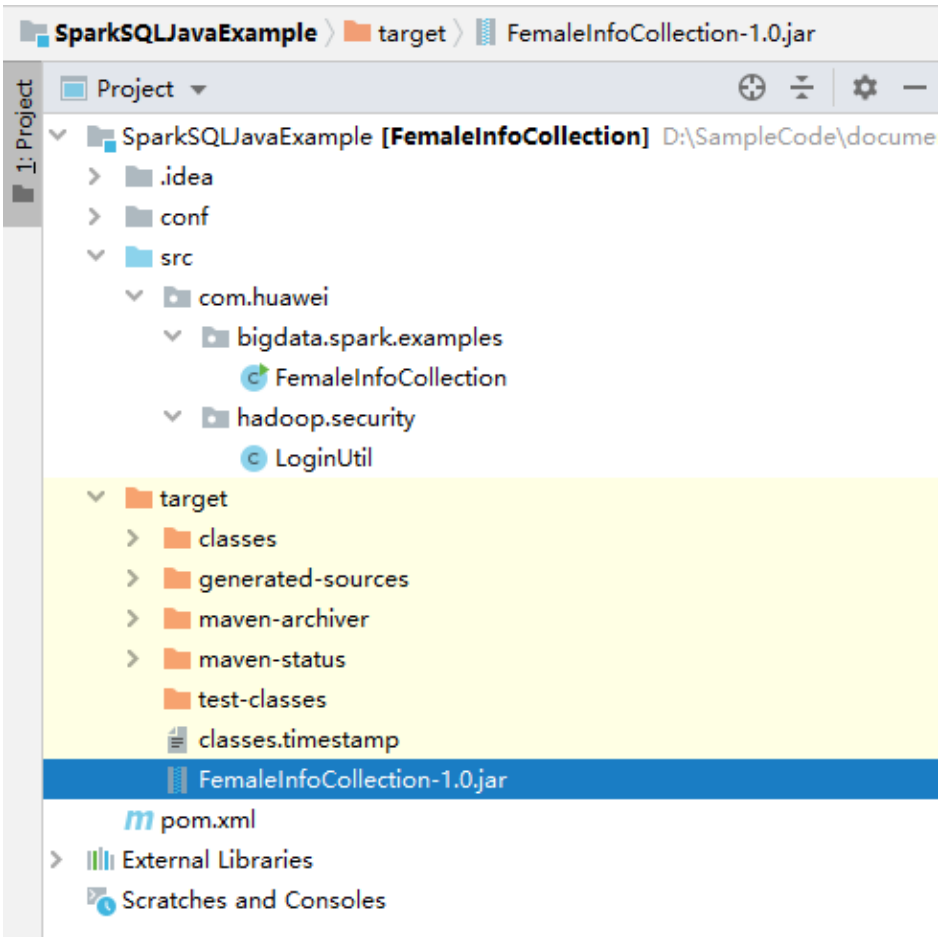
If the following information is displayed in **Run:**, the packaging is successful.

Figure 28-39 Packaging success message

```
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ FemaleInfoCollection ---
[INFO]
[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 28-40 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

28.4.2.2 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 WebUI.
- 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 the final column **ApplicationMaster** of the application information to access the Spark UI.

  – 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. Log related information can be referenced to**Spark2x Logs**.

28.5 More Information

28.5.1 Common APIs

28.5.1.1 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 level class, including MEMORY_ONLY, DISK_ONLY, and MEMORY_AND_DISK.

The JavaRDD supports two types of operations, transformation and action. **Table 28-10** and **Table 28-11** show the common methods.

Table 28-10 Transformation

Method	Description
<R> JavaRDD<R> map(Function<T,R> f)	Return a new RDD by applying a function to all elements of this RDD.
JavaRDD<T> filter(Function<T,Boolean> f)	Return a new RDD containing only the elements that satisfy a predicate.
<U> JavaRDD<U> flatMap(FlatMapFunction<T,U> f)	Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
JavaRDD<T> sample(boolean withReplacement, double fraction, long seed)	Return a sampled subset of this RDD.
JavaRDD<T> distinct(int numPartitions)	Return a new RDD containing the distinct elements in this RDD.
JavaPairRDD<K,Iterable<V>> groupByKey(int numPartitions)	Group the values for each key in the RDD into a single sequence. Hash-partitions the resulting RDD with into numPartitions partitions.
JavaPairRDD<K,V> reduceByKey(Function2<V,V,V> func, int numPartitions)	Merge the values for each key using an associative reduce function. This will also perform the merging locally on each mapper before sending results to a reducer, similarly to a "combiner" in MapReduce. Output will be hash-partitioned with numPartitions partitions.

Method	Description
JavaPairRDD<K,V> sortByKey(boolean ascending, int numPartitions)	Sort the RDD by key, so that each partition contains a sorted range of the elements. Calling collect or save on the resulting RDD will return or output an ordered list of records (in the save case, they will be written to multiple part-X files in the filesystem, in order of the keys).
JavaPairRDD<K,scala.Tuple2<V,W>> join(JavaPairRDD<K,W> other)	Return an RDD containing all pairs of elements with matching keys in this and other. Each pair of elements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in this and (k, v2) is in other. Performs a hash join across the cluster.
JavaPairRDD<K,scala.Tuple2<Iterable<V>,Iterable<W>>> cogroup(JavaPairRDD<K,W> other, int numPartitions)	For each key k in this or other, return a resulting RDD that contains a tuple with the list of values for that key in this as well as other.
JavaPairRDD<T,U> cartesian(JavaRDDLike<U,?> other)	Return the Cartesian product of this RDD and another one, that is, the RDD of all pairs of elements (a, b) where a is in this and b is in other.

Table 28-11 Action

Method	Description
T reduce(Function2<T,T,T> f)	Reduces the elements of this RDD using the specified commutative and associative binary operator.
java.util.List<T> collect()	Return an array that contains all of the elements in this RDD.
long count()	Return the number of elements in the RDD.
T first()	Return the first element in this RDD.
java.util.List<T> take(int num)	Take the first num elements of the RDD. This currently scans the partitions *one by one*, so it will be slow if a lot of partitions are required. In that case, use collect() to get the whole RDD instead.
java.util.List<T> takeSample(boolean withReplacement, int num, long seed)	Return a fixed-size sampled subset of this RDD in an array

Method	Description
void saveAsTextFile(String path, Class<? extends org.apache.hadoop.io.compress.CompressionCodec> codec)	Save this RDD as a compressed text file, using string representations of elements.
java.util.Map<K,Object> countByKey()	Count the appearance times of each key.
void foreach(VoidFunction<T> f)	Applies a function f to all elements of this RDD.
java.util.Map<T,Long> countByValue()	Return the count of each unique value in this RDD as a map of (value, count) pairs. The final combine step happens locally on the master, equivalent to running a single reduce task.

Table 28-12 New APIs of Spark core

API	Description
public java.util.concurrent.atomic.AtomicBoolean isSparkContextDown()	Determines whether sparkContext is shut down completely. The initial value is **false**. The value **true** indicates that sparkContext is shut down completely. The value **false** indicates that sparkContext is not shut down. For example, **jsc.sc().isSparkContextDown().get() == true** indicates that sparkContext is shut down completely.

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 28-13 Spark Streaming methods

Method	Description
JavaReceiverInputDStream<java.lang.String> socketStream(java.lang.String hostname,int port)	It 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)	It 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()	It starts the Spark Streaming calculation.
void awaitTermination()	It terminates the await of the process, which is similar to pressing Ctrl+C.
void stop()	It stops the Spark Streaming calculation.
<T> JavaDStream<T> transform(java.util.List<JavaDStream<?>> dstreams,Function2<java.util.List<JavaRDD<?>>,Time,JavaRDD<T>> transformFunc)	It 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)	It creates a unified Dstream from multiple DStreams with the same type and sliding window.

Table 28-14 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 28-15 Spark SQL methods

Method	Description
Row[] collect()	Return an array containing all DataFrame columns.
long count()	Return the number of DataFrame rows.
DataFrame describe(java.lang.String... cols)	Count the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
Row first()	Return the first row.
Row[] head(int n)	Return the first n rows.
void show()	Display the first 20 rows in table.
Row[] take(int n)	Return the first n rows in the DataFrame.

Table 28-16 Basic DataFrame Functions

Method	Description
void explain(boolean extended)	Print the logical plan and physical plan of the SQL.
void printSchema()	Print the schema information to the console.
registerTempTable	Register the DataFrame as a temporary table, whose period is bound to the SQLContext.
DataFrame toDF(java.lang.String... colNames)	Return a DataFrame whose columns are renamed.
DataFrame sort(java.lang.String sortCol,java.lang.String... sortCols)	Based on different columns, sort columns in ascending or descending orders.

Method	Description
GroupedData rollup(Column... cols)	Perform multiple-dimension crankback on the specified columns in the DataFrame.

28.5.1.2 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 level class, including MEMORY_ONLY, DISK_ONLY, and MEMORY_AND_DISK.

The RDD supports two types of operations, transformation and action. **Table 28-17** and **Table 28-18** show the common methods.

Table 28-17 Transformation

Method	Description
map[U](f: (T) => U): RDD[U]	Return a new RDD by applying a function to all elements of this RDD.
filter(f: (T) => Boolean): RDD[T]	Return a new RDD containing only the elements that satisfy a predicate.
flatMap[U](f: (T) => TraversableOnce[U]) (implicit arg0: ClassTag[U]): RDD[U]	Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.

Method	Description
sample(withReplacement: Boolean, fraction: Double, seed: Long = Utils.random.nextLong): RDD[T]	Return a sampled subset of this RDD.
union(other: RDD[T]): RDD[T]	Return a new RDD, contains source RDD and the group of RDD's elements.
distinct([numPartitions: Int]): RDD[T]	Return a new RDD containing the distinct elements in this RDD.
groupByKey(): RDD[(K, Iterable[V])]	Group the values for each key in the RDD into a single sequence. Hash-partitions the resulting RDD with into numPartitions partitions.
reduceByKey(func: (V, V) => V[, numPartitions: Int]): RDD[(K, V)]	Merge the values for each key using an associative reduce function. This will also perform the merging locally on each mapper before sending results to a reducer, similarly to a "combiner" in MapReduce. Output will be hash-partitioned with numPartitions partitions.
sortByKey(ascending: Boolean = true, numPartitions: Int = self.partitions.length): RDD[(K, V)]	Sort the RDD by key, so that each partition contains a sorted range of the elements. Calling collect or save on the resulting RDD will return or output an ordered list of records (in the save case, they will be written to multiple part-X files in the filesystem, in order of the keys).
join[W](other: RDD[(K, W)][, numPartitions: Int]): RDD[(K, (V, W))]	Return an RDD containing all pairs of elements with matching keys in this and other. Each pair of elements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in this and (k, v2) is in other. Performs a hash join across the cluster.
cogroup[W](other: RDD[(K, W)], numPartitions: Int): RDD[(K, (Iterable[V], Iterable[W]))]	For each key k in this or other, return a resulting RDD that contains a tuple with the list of values for that key in this as well as other.
cartesian[U](other: RDD[U])(implicit arg0: ClassTag[U]): RDD[(T, U)]	Return the Cartesian product of this RDD and another one, that is, the RDD of all pairs of elements (a, b) where a is in this and b is in other.

Table 28-18 Action

Method	Description
reduce(f: (T, T) => T):	Reduces the elements of this RDD using the specified commutative and associative binary operator.
collect(): Array[T]	Return an array that contains all of the elements in this RDD.
count(): Long	Return the number of elements in the RDD.
first(): T	Return the first element in this RDD.
take(num: Int): Array[T]	Take the first num elements of the RDD. This currently scans the partitions *one by one*, so it will be slow if a lot of partitions are required. In that case, use collect() to get the whole RDD instead.
takeSample(withReplacement: Boolean, num: Int, seed: Long = Utils.random.nextLong): Array[T]	Return a fixed-size sampled subset of this RDD in an array
saveAsTextFile(path: String): Unit	Save this RDD as a compressed text file, using string representations of elements.
saveAsSequenceFile(path: String, codec: Option[Class[_ <: CompressionCodec]] = None): Unit	Extra functions available on RDDs of (key, value) pairs to create a Hadoop SequenceFile, through an implicit conversion.
countByKey(): Map[K, Long]	Count 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]	Return the count of each unique value in this RDD as a map of (value, count) pairs. The final combine step happens locally on the master, equivalent to running a single reduce task.

Table 28-19 New APIs of Spark core

API	Description
isSparkContextDown:AtomicBoolean	Determines whether sparkContext is shut down completely. The initial value is **false**. The value **true** indicates that sparkContext is shut down completely. The value **false** indicates that sparkContext is not shut down. For example, **sc.isSparkContextDown.get() == true** indicates that sparkContext is shut down completely.

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 28-20 Spark Streaming methods

Method	Description
socketTextStream(hostname: String, port: Int, storageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK_SER_2): ReceiverInputDStream[String]	Reduces the elements of this RDD using the specified commutative and associative binary operator.
start():Unit	Return an array that contains all of the elements in this RDD.
awaitTermination(timeout: long):Unit	Return the number of elements in the RDD.

Method	Description
stop(stopSparkContext: Boolean, stopGracefully: Boolean): Unit	Return the first element in this RDD.
transform[T] (dstreams: Seq[DStream[_]], transformFunc: (Seq[RDD[_]], Time) ? RDD[T])(implicit arg0: ClassTag[T]): DStream[T]	Take the first num elements of the RDD. This currently scans the partitions *one by one*, so it will be slow if a lot of partitions are required. In that case, use collect() to get the whole RDD instead.
UpdateStateByKey(func)	Return a fixed-size sampled subset of this RDD in an array
window(windowLength, slideInterval)	Save this RDD as a compressed text file, using string representations of elements.
countByWindow(windowLength, slideInterval)	Count the appearance times of each key.
reduceByWindow(func, windowLength, slideInterval)	Applies a function f to all elements of this RDD.
join(otherStream, [numTasks])	Return the count of each unique value in this RDD as a map of (value, count) pairs. The final combine step happens locally on the master, equivalent to running a single reduce task.
DStreamKafkaWriter.writeToKafka()	Writes data from DStream into Kafka in batch.
DStreamKafkaWriter.writeToKafkaBySingle()	Writes data from DStream into Kafka one by one.

Table 28-21 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 28-22 Common Actions methods

Method	Description
collect(): Array[Row]	Return an array containing all DataFrame columns.
count(): Long	Return the number of DataFrame rows.
describe(cols: String*): DataFrame	Count the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
first(): Row	Return the first row.
Head(n:Int): Row	Return the first n rows.
show(numRows: Int, truncate: Boolean): Unit	Display the first 20 rows in table.
take(n:Int): Array[Row]	Return the first n rows in the DataFrame.

Table 28-23 Basic DataFrame Functions

Method	Description
explain(): Unit	Print the logical plan and physical plan of the SQL.
printSchema(): Unit	Print the schema information to the console.
registerTempTable(tableName: String): Unit	Register the DataFrame as a temporary table, whose period is bound to the SQLContext.
toDF(colNames: String*): DataFrame	Return a DataFrame whose columns are renamed.

28.5.1.3 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 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.
- pyspark.SparkConf: Spark application configuration class, which is used to configure the application name, execution model, 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: A broadcast variable created with SparkContext.broadcast(). Access its value through value.
- pyspark.StorageLevel: data storage level class, including MEMORY_ONLY, DISK_ONLY, and MEMORY_AND_DISK.
- pyspark.sql.SQLContext: Main entry point for SparkSQL functionality. A SQLContext can be used create SchemaRDDs, register SchemaRDDs as tables, execute SQL over tables, cache tables, and read parquet files.
- 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 28-24** and **Table 28-25** show the common methods.

Table 28-24 Transformation

Method	Description
map(f, preservesPartition-ing=False)	Return a new RDD by applying a function to all elements of this RDD.
filter(f)	Return a new RDD containing only the elements that satisfy a predicate.
flatMap(f, preservesPartition-ing=False)	Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
sample(withReplacement, fraction, seed=None)	Return a sampled subset of this RDD.
union(rdds)	Return a new RDD, contains source RDD and the group of RDD's elements.

Method	Description
distinct([numPartitions: Int]): RDD[T]	Return a new RDD containing the distinct elements in this RDD.
groupByKey(): RDD[(K, Iterable[V])]	Group the values for each key in the RDD into a single sequence. Hash-partitions the resulting RDD with into numPartitions partitions.
reduceByKey(func, numPartitions=None)	Merge the values for each key using an associative reduce function. This will also perform the merging locally on each mapper before sending results to a reducer, similarly to a "combiner" in MapReduce. Output will be hash-partitioned with numPartitions partitions.
sortByKey(ascending=True, numPartitions=None, keyfunc=function <lambda>)	Sort the RDD by key, so that each partition contains a sorted range of the elements. Calling collect or save on the resulting RDD will return or output an ordered list of records (in the save case, they will be written to multiple part-X files in the filesystem, in order of the keys).
join(other, numPartitions)	Return an RDD containing all pairs of elements with matching keys in this and other. Each pair of elements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in this and (k, v2) is in other. Performs a hash join across the cluster.
cogroup(other, numPartitions)	For each key k in this or other, return a resulting RDD that contains a tuple with the list of values for that key in this as well as other.
cartesian(other)	Return the Cartesian product of this RDD and another one, that is, the RDD of all pairs of elements (a, b) where a is in this and b is in other.

Table 28-25 Action

Method	Description
reduce(f)	Reduces the elements of this RDD using the specified commutative and associative binary operator.
collect()	Return an array that contains all of the elements in this RDD.
count()	Return the number of elements in the RDD.
first()	Return the first element in this RDD.

Method	Description
take(num)	Take the first num elements of the RDD. This currently scans the partitions *one by one*, so it will be slow if a lot of partitions are required. In that case, use collect() to get the whole RDD instead.
takeSample(withReplacement, num, seed)	Return a fixed-size sampled subset of this RDD in an array
saveAsTextFile(path, compressionCodecClass)	Save this RDD as a compressed text file, using string representations of elements.
saveAsSequenceFile(path, compressionCodecClass=None)	Extra functions available on RDDs of (key, value) pairs to create a Hadoop SequenceFile, through an implicit conversion.
countByKey()	Count the appearance times of each key.
foreach(func)	Applies a function f to all elements of this RDD.
countByValue()	Return the count of each unique value in this RDD as a map of (value, count) pairs. The final combine step happens locally on the master, equivalent to running a single reduce task.

Spark Streaming Common Interfaces

Spark Streaming mainly uses the following classes:

- pyspark.streaming.StreamingContext: A StreamingContext is the main entry point for Spark Streaming functionality. Besides the basic information (such as, cluster URL and job name) to internally create a SparkContext, it provides methods used to create DStreams from various input sources.
- 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 28-26 Spark Streaming common interfaces

Method	Description
socketTextStream(hostname, port, storageLevel)	Reduces the elements of this RDD using the specified commutative and associative binary operator.
start()	Return an array that contains all of the elements in this RDD.
awaitTermination(timeout)	Return the number of elements in the RDD.
stop(stopSparkContext, stopGraceFully)	Return the first element in this RDD.
UpdateStateByKey(func)	Take the first num elements of the RDD. This currently scans the partitions *one by one*, so it will be slow if a lot of partitions are required. In that case, use collect() to get the whole RDD instead.
window(windowLength, slideInterval)	Return a fixed-size sampled subset of this RDD in an array
countByWindow(windowLength, slideInterval)	Save this RDD as a compressed text file, using string representations of elements.
reduceByWindow(func, windowLength, slideInterval)	Count the appearance times of each key.
join(other,numPartitions)	Applies a function f to all elements of this RDD. Return the count of each unique value in this RDD as a map of (value, count) pairs. The final combine step happens locally on the master, equivalent to running a single reduce task.

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: A variant of Spark SQL that integrates with 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 28-27 Spark SQL common Actions

Method	Description
collect()	Return an array containing all DataFrame columns.
count()	Return the number of DataFrame rows.
describe()	Count the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
first()	Return the first row.
head(n)	Return the first n rows.
show()	Display the first 20 rows in table.
take(num)	Return the first n rows in the DataFrame.

Table 28-28 Basic DataFrame Functions

Method	Description
explain()	Print the logical plan and physical plan of the SQL.
printSchema()	Print the schema information to the console.
registerTempTable(name)	Register the DataFrame as a temporary table, whose period is bound to the SQLContext.
toDF()	Return a DataFrame whose columns are renamed.

28.5.1.4 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=false** and restart Spark2x (Change the value of this parameter for the JobHistory2x, JDBCServer2x, and SparkResource2x instances.) .

NOTE

HTTPS-based access is different from HTTP-based access. When you access JobHistory of Spark2x using HTTPS, 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, use either of the following methods:

- Modify the SSL protocol configured for the cluster. For example, if the curl command supports only the TLSv1 protocol (TLSv1 has security vulnerabilities and must be used with caution), perform the following steps:
  1. Log in to FusionInsight Manager and choose **Cluster** > *Name of the desired cluster* > **Services** > **Spark2x** > **Configurations** > **All Configurations**.
  2. Search for **ssl** in the search box. Check whether the value of **spark.ssl.historyServer.protocol** for JobHistory contains **TLSv1**. If it does not, add **TLSv1** to the value.
  3. Clear the value of the **spark.ssl.historyServer.enabledAlgorithms** parameter for JobHistory.
  4. Click **Save Configuration** and then click **OK**. Restart the Spark2x service or JobHistory instance.
- 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 command 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://192.168.227.16:4040/api/v1/applications"
    ```


"192.168.227.16" indicates the service IP address of the JobHistory node; "4040" 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:

  After running this command, you can query information about all Spark applications in the current cluster. **Table 28-29** describes the parameters in response to this command.

  Table 28-29 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://192.168.227.16:4040/api/v1/applications/
application_1517290848707_0008"
```

    "192.168.227.16" indicates the service IP address of the JobHistory node; "4040" indicates the port number of the JobHistory node; "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:

    After running this command, you can query the information about a Spark application. For the description of parameters in response to this command, see **Table 28-29**.

- Obtain the 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"
```

    "192.168.195.232" indicates the service IP address of the master node of the ResourceManager; "8090" indicates the port number of the ResourceManager; "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-224:26010/node/containerlogs/
container_1478570725074_0049_01_000002/admin/stdout?start=-4096",
    "stderr" : "https://XTJ-224:26010/node/containerlogs/
container_1478570725074_0049_01_000002/admin/stderr?start=-4096"
  }
} ]
```

- Analysis:

  After running this command, you can query information about all Executors (including the driver) of the current application. **Table 28-30** describes the parameters in response to this command.

  Table 28-30 Parameter description

Parameter	Description
id	Executor ID.
hostPort	IP address and port number of the node where the Executor resides. Format: *IP address:port number*.
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; "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://192.168.227.16:4040/api/v1/applications/
application_1478570725074_0004/SQL"
```

"192.168.227.16" indicates the service IP address of the JobHistory node; "4040" indicates the port number of the JobHistory node; "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" : [ ],
    "successedJobs" : [ 0 ],
    "failedJobs" : [ ]
  } ],
  "sqls" : [ {
    "id" : 0,
    "status" : "COMPLETED",
    "description" : "getCallSite at SQLExecution.scala:48",
    "submissionTime" : "2016/11/08 15:39:00",
    "duration" : "2 s",
    "runningJobs" : [ ],
    "successedJobs" : [ 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). **Table 28-31** describes the parameters in response to this command.

Table 28-31 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.
successedJobs	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; "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
      } ],
      "sqls" : [ {
        "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. **Table 28-32** describes the parameters in the queried session information; **Table 28-33** describes the parameters in the queried SQL statement information.

    Table 28-32 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.

Parameter	Description
finishTime	Time when the session ends the connection.
duration	Connection duration of the session.
totalExecute	Number of SQL statements executed by the session.

Table 28-33 SQL parameter description

Parameter	Description
user	User who executes the SQL statement.
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; "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.

28.5.2 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 ***./bin/spark-shell*** to log in the interactive interface of Scala, obtain data from 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()

  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

28.5.3 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.

JDBCServer starts a JDBC service on port 22550 of the installation node by default. (If you want to change the port, configure the **hive.server2.thrift.port** parameter.) You can connect to JDBCServer using Beeline or running the JDBC client code to run SQL statements.

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 **kinit** 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=sparkthriftserver2x;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.

28.5.4 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.

Left Table	Right Table	Supported Join Type	Description
		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.
		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	**path**: file path, which is mandatory. **maxFilesPerTrigger**: maximum number of files in each trigger. The default value is infinity. **latestFirst**: whether to process limited number of new files. The default value is **false**. **fileNameOnly**: whether the file name is used as the new file for verification instead of using the complete path. (Default value: **false**)	Supported	Paths with wildcard are supported, but multiple paths separated by commas (,) are not supported. Files must be placed in a given directory in atomic mode, which can be implemented through file movement in most file systems.
Socket Source	**host**: IP address of the connected node, which is mandatory. **port**: connected port, which is mandatory.	Not supported	-

Sources	Supported Options	Fault Tolerance Supported	Description
Rate Source	**rowsPerSecond**: number of rows generated per second. The default value is **1**. **rampUpTime**: rising time before the speed specified by **rowsPerSecond** is reached. **numPartitions**: concurrency degree for generating data rows.	Supported	-
Kafka Source	For details, see **https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html**	Supported	-

3. Fault tolerance list of Sink

Sinks	Supported Output Mode	Supported Options	Fault Tolerance	Description
File Sink	Append	**Path**: 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 **https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html**.	at-least-once	For details, see **https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html**.
Foreach Sink	Append, Update, Complete	None	Depends on ForeachWriter.	For details, see **https://spark.apache.org/docs/3.1.1/structured-streaming-programming-guide.html#using-foreach**.

Sinks	Supported Output Mode	Supported Options	Fault Tolerance	Description
ForeachBatch Sink	Append, Update, Complete	None	Depends on operator.	For details, see **https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#using-foreach-and-foreachbatch**.
Console Sink	Append, Update, Complete	**numRows**: number of rows printed in each round. The default value is **20**. **truncate**: whether to clear the output when the output is too long. The default value is **true**.	Not supported	-
Memory Sink	Append, Complete	None	Not supported. In complete mode, the entire table is rebuilt after the query is restarted.	-

28.5.5 FAQ

28.5.5.1 How to Add a User-Defined Library

Question

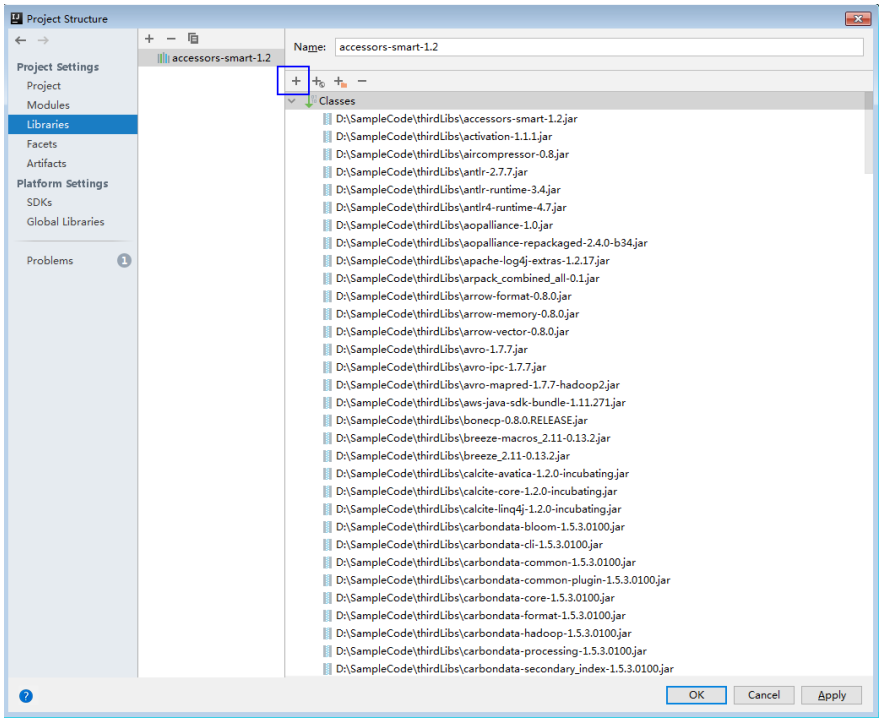
In the development of the Spark application, users may add a user-defined dependent library which is different from the sample project. This section describes how to add a dependent library with user-defined codes into the project.

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 28-41 Add the Dependent Library

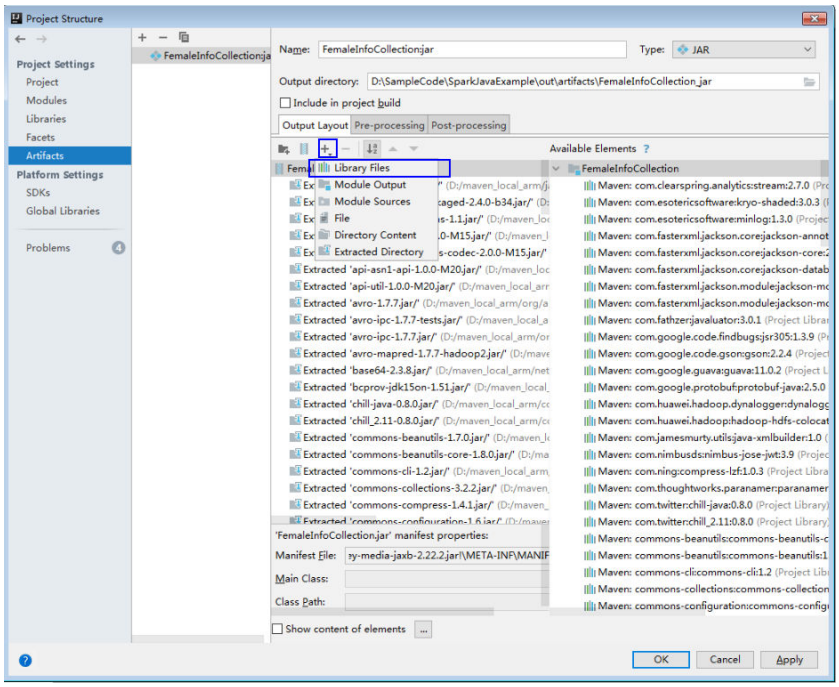


Step 3 Click **Apply** to load the dependent library and click **OK** to complete the configuration.

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.

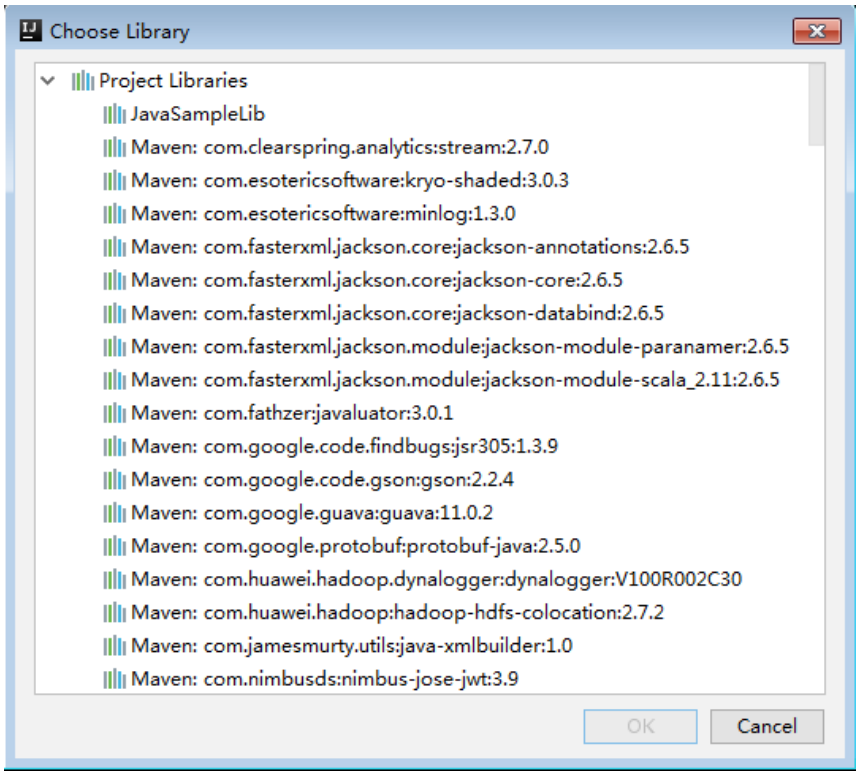
1. On the **Project Structure** page, select the **Artifacts** tab.
2. Click the icon "**+**" and select **Library Files** in the right window to add the dependent library.

Figure 28-42 Add Library Files



3. Choose the dependent library to be added and click **OK**.

Figure 28-43 Choose Libraries



4. Click **Apply** to load the dependent library and click **OK** to complete the configuration.

----End

28.5.5.2 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**.

28.5.5.3 Why the "Class Does not Exist" Error Is Reported While the SparkStresmingKafka 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_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, 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_2.12-2.4.0.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**.

28.5.5.4 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.

28.5.5.5 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
```



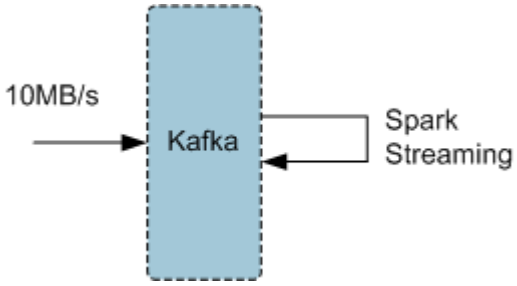
```
(Logging.scala:68)
2016-03-02 17:46:19,155 | INFO | [kafka-network-thread-21005-2] | Closing socket connection to /
10.91.8.208. | kafka.network.Processor (Logging.scala:68)
2016-03-02 17:46:19,270 | INFO | [kafka-network-thread-21005-0] | Closing socket connection to /
10.91.8.208 due to invalid 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 28-44 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.

28.5.5.6 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.newArray(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$$anon$3$$anonfun$run$1.apply$mcV
$sp(TaskResultGetter.scala:66)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.util.Utils$.logUncaughtExceptions(Utils.scala:1716)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3.run(TaskResultGetter.scala:56)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)
Exception in thread "task-result-getter-2" java.lang.OutOfMemoryError: Java heap space
at java.lang.reflect.Array.newArray(Native Method)
at java.lang.reflect.Array.newInstance(Array.java:75)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1671)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1707)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
at org.apache.spark.serializer.JavaDeserializationStream.readObject(JavaSerializer.scala:71)
at org.apache.spark.serializer.JavaSerializerInstance.deserialize(JavaSerializer.scala:91)
at org.apache.spark.scheduler.DirectTaskResult.value(TaskResult.scala:94)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$$anonfun$run$1.apply$mcV
$sp(TaskResultGetter.scala:66)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.util.Utils$.logUncaughtExceptions(Utils.scala:1716)
at org.apache.spark.scheduler.TaskResultGetter$$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'
```

28.5.5.7 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_1463550673665_0007	zwn	Spark Pi	SPARK	tenant_zwn	Sat May 28 11:59:27 +0800 2016	Sat May 28 11:59:51 +0800 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A		History	N/A
application_1463550673665_0006	zwn	org.apache.spark.examples.SparkPi	SPARK	tenant_zwn	Sat May 28 11:58:29 +0800 2016	Sat May 28 11:59:00 +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*

28.5.5.8 How to Perform Remote Debugging Using IDEA?

Question

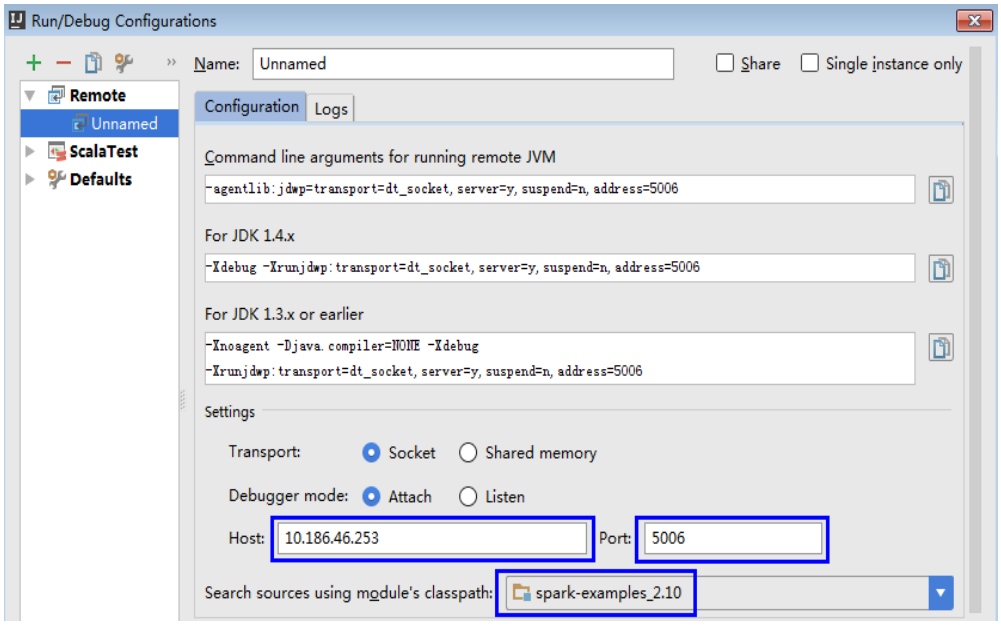
How to perform remote debugging using IDEA during Spark secondary development?

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. On the displayed window, click + at the upper left corner. Then on the drop-down menu, choose **Remote**, as shown in **Figure 28-45**.

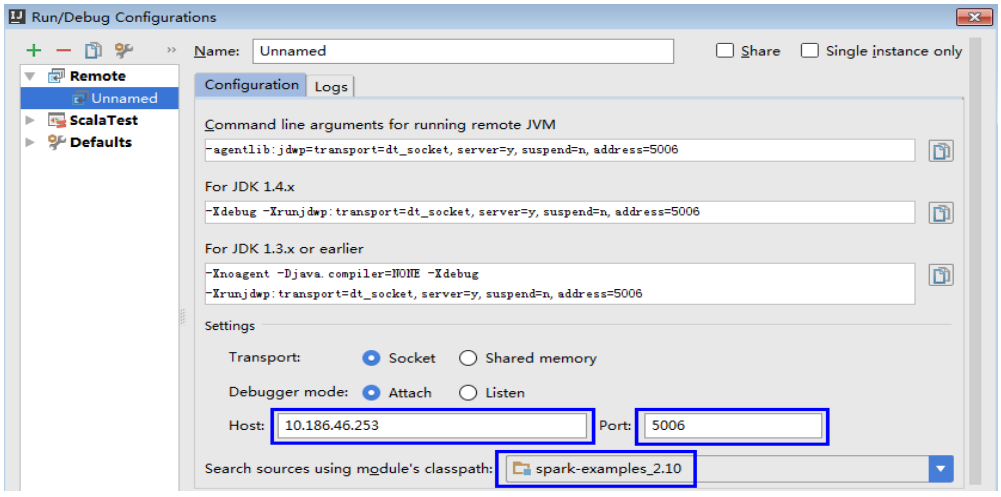
Figure 28-45 Choosing **Remote**



3. Configure the **Host**, **Port**, and **Search source using module's classpath**, as shown in **Figure 28-46**.

   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 28-46 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 -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5006*** to the **For JDK1.4.xdebugging** command obtained from **3**

   Figure 28-47 Command for running Spark

```
[root@host2 bin]#./spark-submit --master yarn-client --driver-java-options "-Xde
bug -Xrunjdwp: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 28-48** illustrates how to select the breakpoint of the code in row 29 of SparkPi.scala.

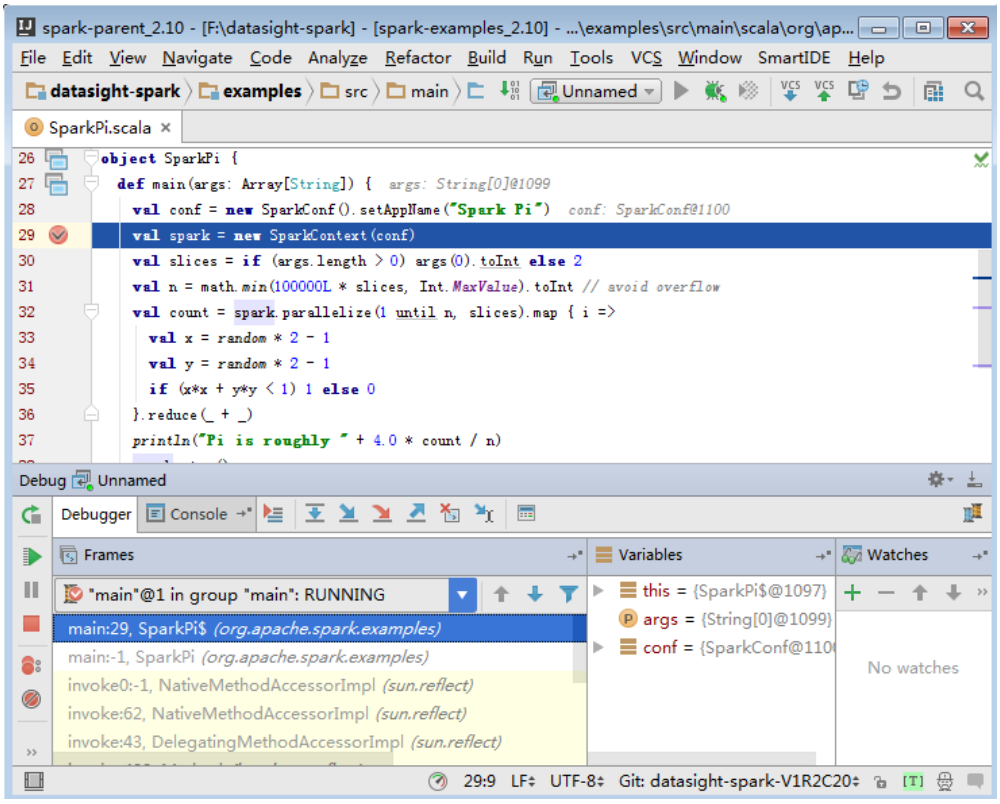
   Figure 28-48 Setting the breakpoint

```
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 28-49**.

Figure 28-49 Debugging



28.5.5.9 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.

Step 1 Define the org.apache.spark.launcher.SparkLauncher class. The SparkLauncherJavaExample and SparkLauncherScalaExample are provided by default as example code. You can modify the input parameters of example code 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 requirements 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 example 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 th client. The dependency package of the example 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. Th 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 WebUI 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

28.5.5.10 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.

28.5.5.11 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.

28.5.5.12 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 or folder is a high-risk operation. Ensure that the file or folder is no longer required before performing this operation.

28.5.5.13 Support for Third-party JAR Packages on x86 and TaiShan Platforms

Question

The .jar package (for example, UDF package) written by users has two versions: x86 and TaiShan. How to enable Spark2x to support .jar package for both the x86 and TaiShan platforms?

Answer

Use the hybrid solution.

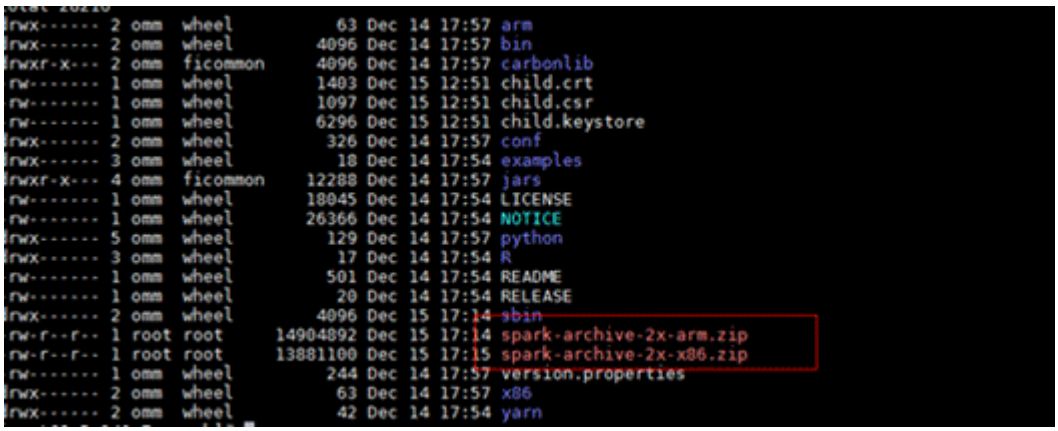
Step 1 Go to the installation directory of the Spark2x sparkResource on the server. The cluster may be installed on multiple nodes. You can go to any installation node and run the cd command to go to the installation directory of the sparkResource.

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/*



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 the **local spark-archive-2x-arm.zip** and **spark-archive-2x-x86.zip** files.

Step 6 Perform **Step 1** to **Step 2** for other sparkResource nodes.

Step 7 Log in to the webUI 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.

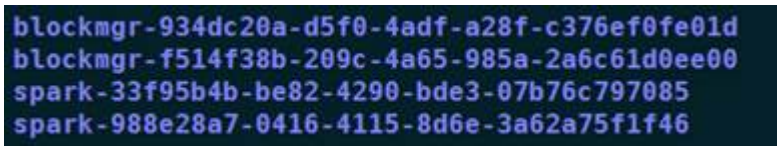
----End

28.5.5.14 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 28-50 Residual directory example



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.

28.5.5.15 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

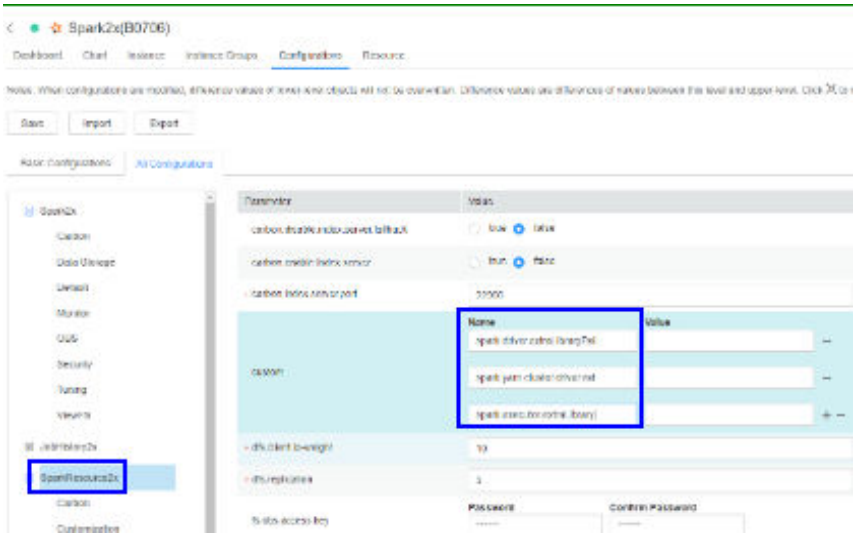
Solution 1:

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.

28.5.5.16 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.

28.5.5.17 Common JAR File Conflicts

Symptom

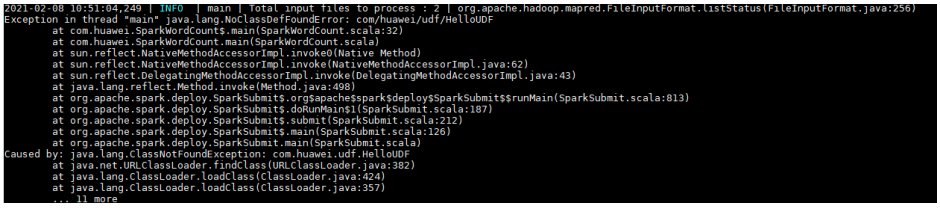
Spark can interconnect with many third-party tools. Therefore, it depends on a large number of third-party packages. Some packages are provided by MRS. As a result, the versions of the JAR files used by the code may be different from those of the JAR files provided by the cluster. In this case, JAR file 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**

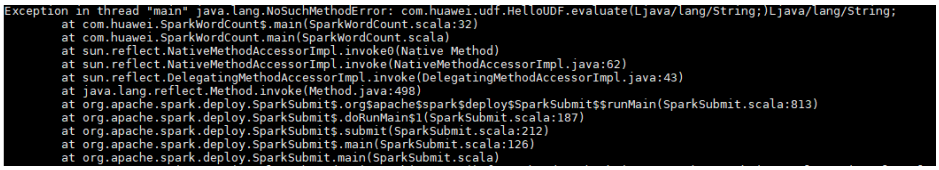
Cause Analysis

The following uses UDF customization as an example:



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.



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>
```


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.

29 Spark2x Development Guide (Normal Mode)

29.1 Overview

29.1.1 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 Interface 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 29-1**.

Table 29-1 Spark APIs

Function	Description
Scala API	Indicates the API in Scala. For common interfaces of Spark Core, Spark SQL and Spark Streaming, see **Scala** . Since Scala is easy to read, users are advised to use Scala interfaces in the program development.
Java API	Indicates the API in Java. For common interfaces of Spark Core, Spark SQL and Spark Streaming, see **Java**.
Python API	Indicates the API in Python. For common interfaces of Spark Core, Spark SQL and Spark Streaming, see **Python**.

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.

29.1.2 Basic Concepts

Basic Concepts

- **RDD**

  Resilient Distributed Dataset (RDD) is a core concept of Spark. It indicates a read-only and partitionable 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.

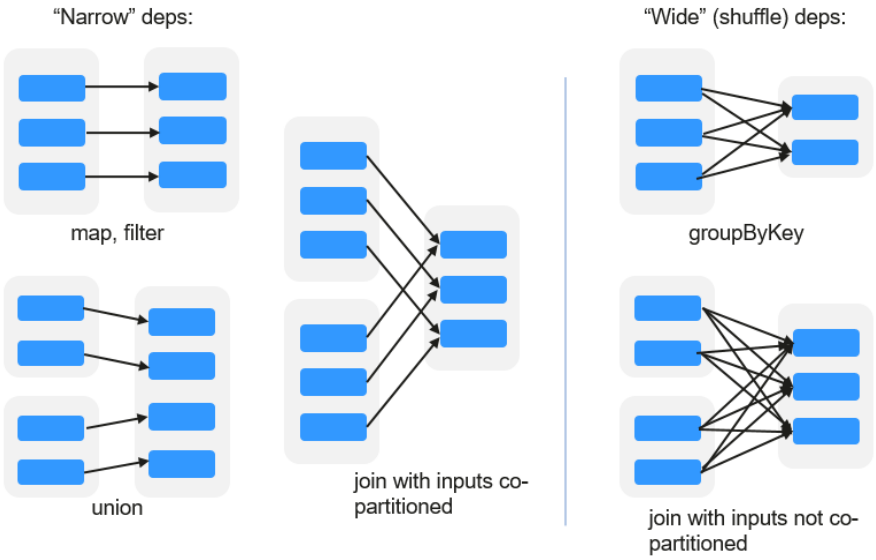
  RDD Storage

  - 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 29-1 RDD dependency



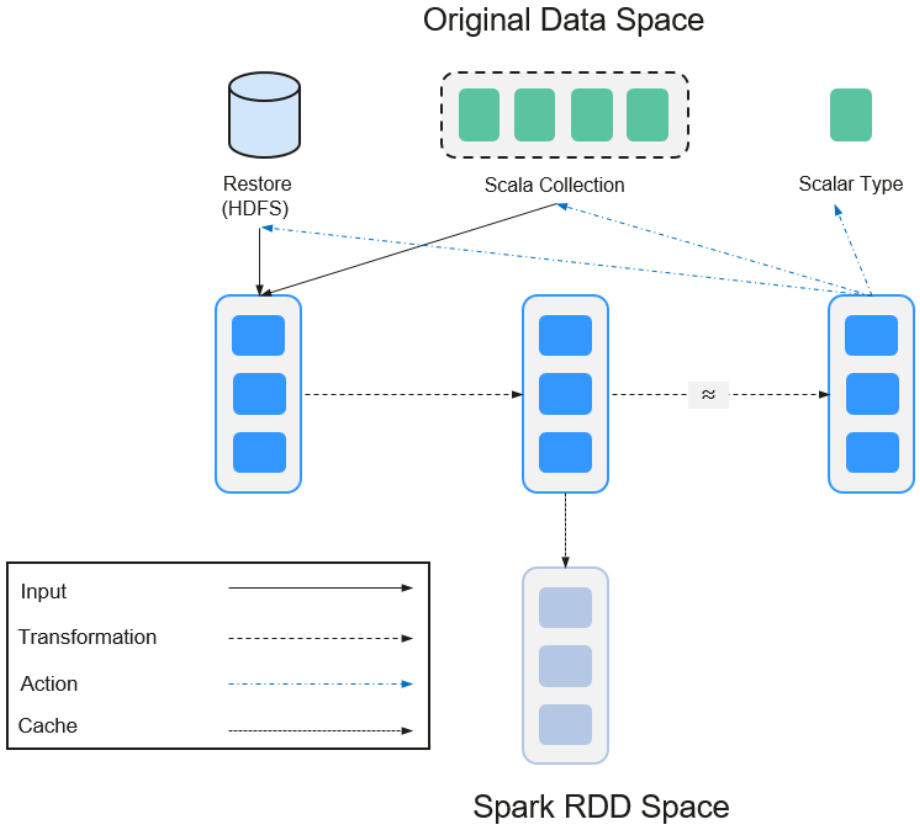
  - **Narrow dependency:** Each partition of the parent RDD is used by at most one partition of the child RDD partition.
  - **Wide dependency:** Partitions of the child RDD depend on all partitions of the parent RDD due to shuffle operations.

  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 29-2** 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 29-2 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()
```

a. The **textFile** operator reads log files from the HDFS and returns **file** (as an RDD).
b. The filter operator filters rows with ERROR and assigns them to errors (a new RDD). The filter operator is a transformation.
c. The cache operator caches errors for future use.
d. 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).

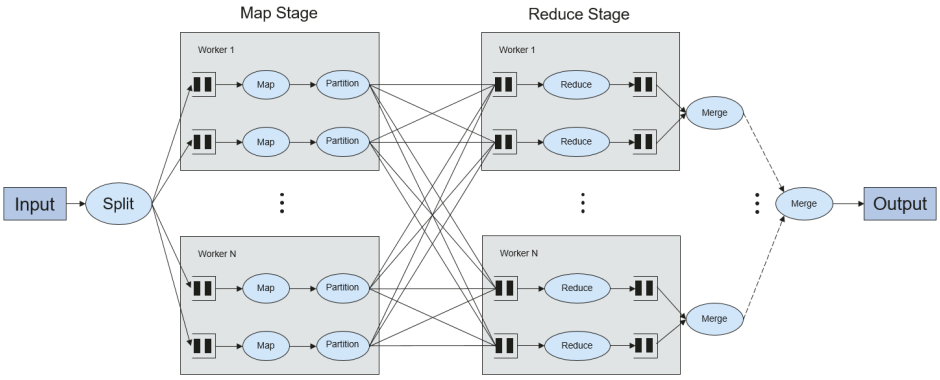
– Check points, such as checkpoint. When Lineage is quite long (which occurs frequently in graphics computation), it takes a long period of time to execute the whole sequence again when a fault occurs. In this case, checkpoint is used as the check point to write the current data to stable storage.

- **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 29-3 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. 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

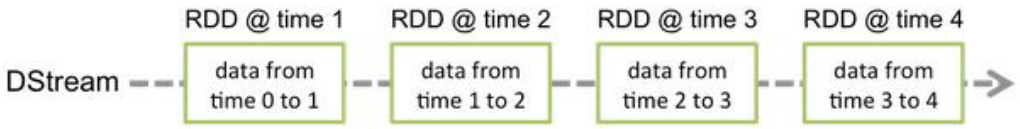
Dstream

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.

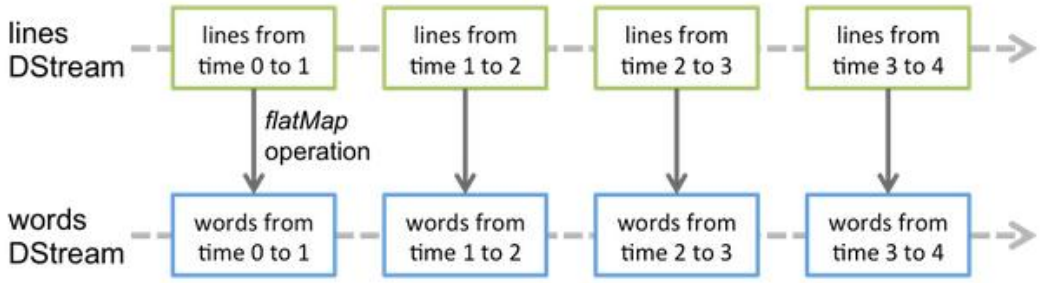
Each RDD in the DStream contains the data of a partition as shown in **Figure 29-4**.

Figure 29-4 Relationship between DStream and RDD



All operators applied in the DStream are translated to the operations in the lower RDD as shown in **Figure 29-5**. 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 29-5 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 a difference between the Appendix 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 29-6 Running process in micro-batch mode

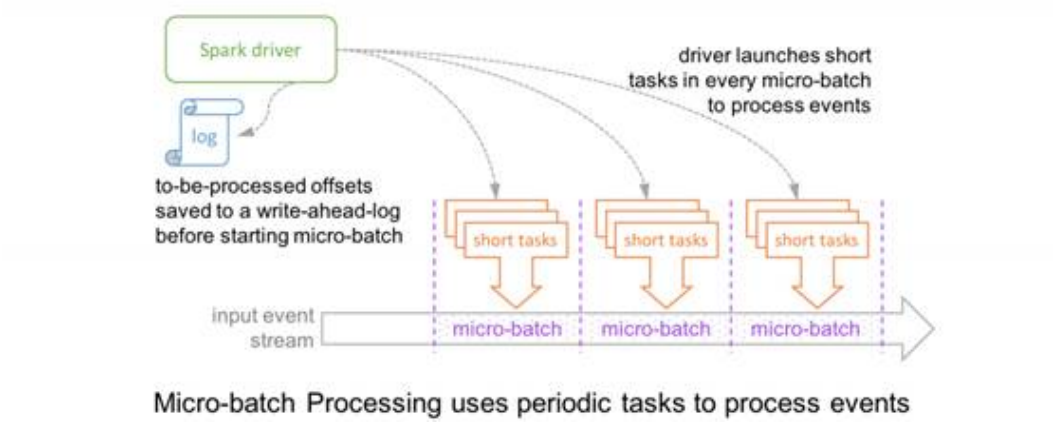
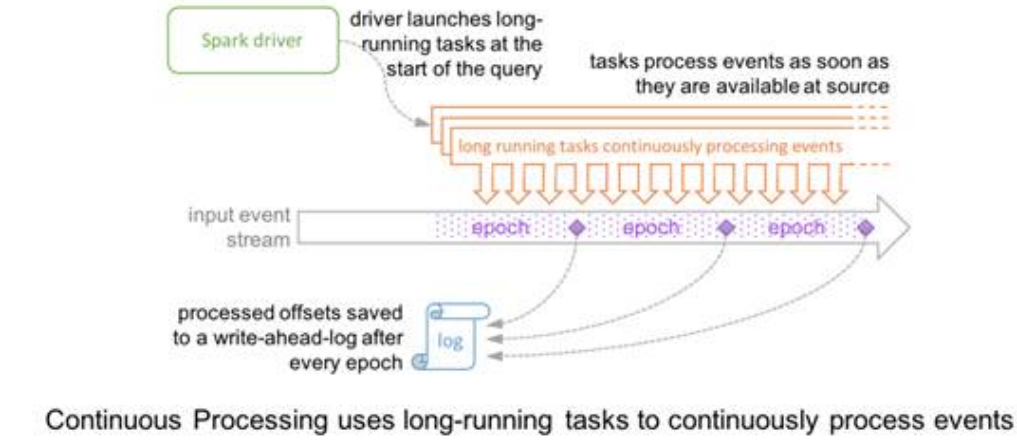


Figure 29-7 Running process in continuous mode



29.1.3 Development Process

Spark includes Spark Core, Spark SQL and Spark Streaming, whose development processes are the same.

All stages of the development process are shown and described in **Figure 29-8** and **Table 29-2**.

Figure 29-8 Spark development process

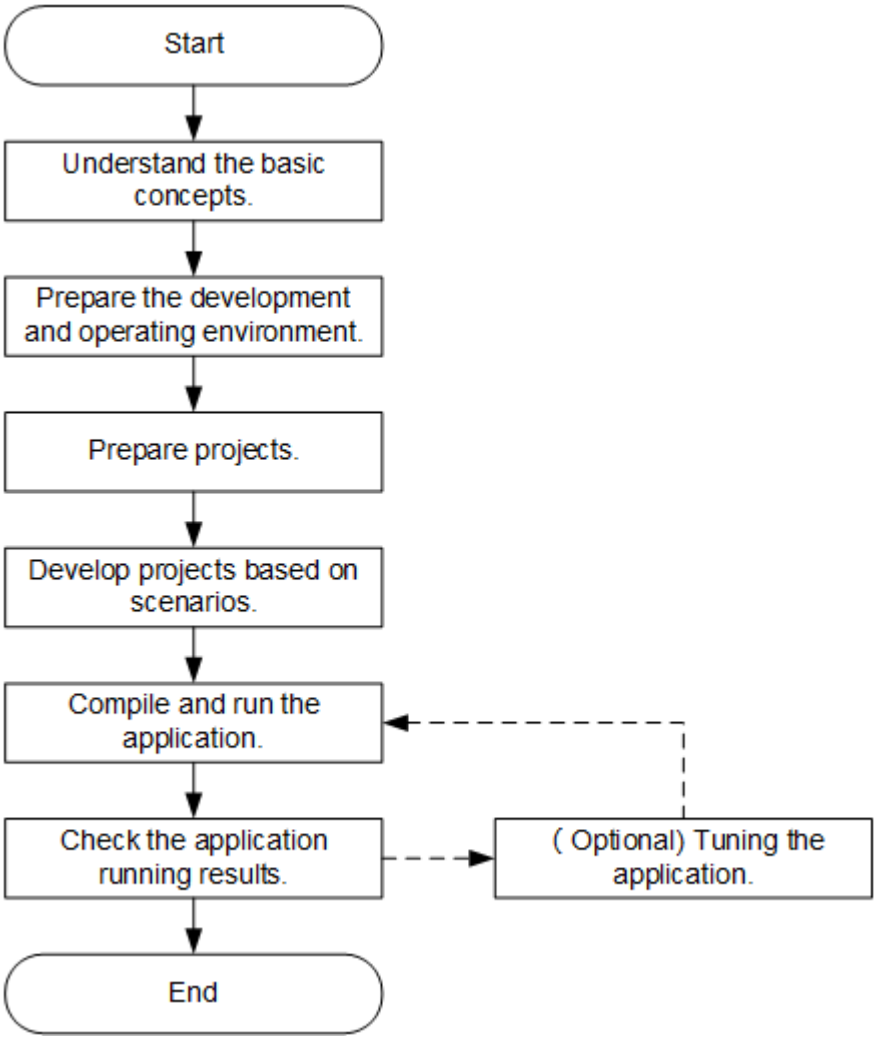


Table 29-2 Description of Spark development process

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.	**Basic Concepts**
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.	**Development and Operating Environment**

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.	**Configuring and Importing Sample Projects** **Creating a New Project (Optional)**
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.	**Developing the Project**
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	**Compiling and Running the Application**
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.	**Checking the Commissioning Result**
Tune the application.	Based on the application running results, tun the application to meet the requirements of the service scenario. After application tuning, compile and run the application again.	**Spark2x Performance Tuning**

29.2 Preparing for the Environment

29.2.1 Development and Operating Environment

Table 29-3 describes the environment required for secondary development.

Table 29-3 Development environment

Preparation Item	Description
OS	• 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 runtime environment must be able to communicate with the cluster service plane network.
JDK installation	Basic configuration for the Java/Scala 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. • For x86 nodes that run clients, the following JDKs can be used: – 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: – OpenJDK: 1.8.0_272 **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 com.ibm.jsse2.overrideDefaultTLS 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.**
IntelliJ IDEA installation and configuration	It is a tool used to develop Spark applications. Version 2019.1 or other compatible versions are recommended. **NOTE** • 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.
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.

Preparation Item	Description
Scala installation	It is the basic configuration for the Scala development environment. The required version is 2.12.14.
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 **.zip** and **.rar** packages, 7-Zip 16.04 is supported.
Python installation	Its version must be 3.7 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 **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\Spark2x\config** and manually import the configuration file to the configuration file directory (usually the **resources** folder) of the Spark sample project.

     Table 29-4 describes the main configuration files.

Table 29-4 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 on the node. For example, install the client in the directory **/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/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 29-5 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.

29.2.2 Configuring and Importing Sample Projects

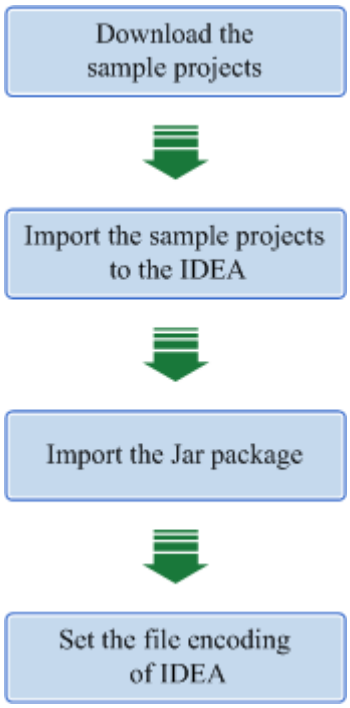
Scenario

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 29-9** shows the procedure of importing sample projects.

Figure 29-9 Procedure of importing sample projects



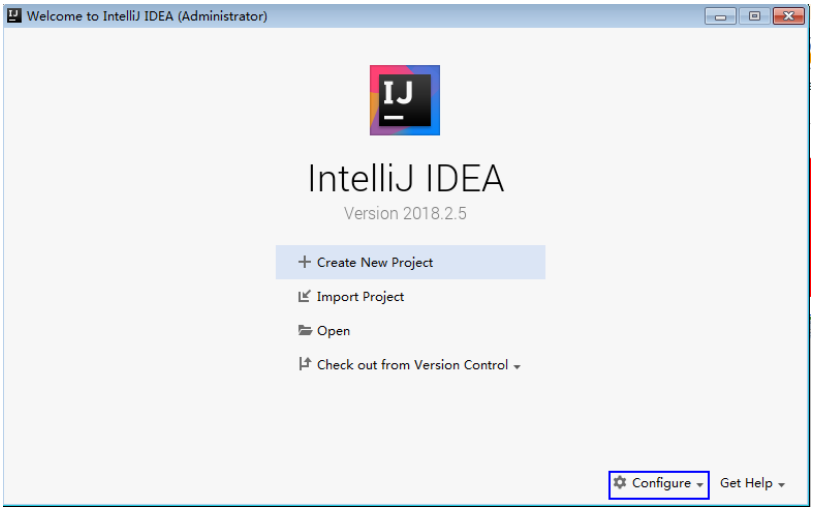
Procedure

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.

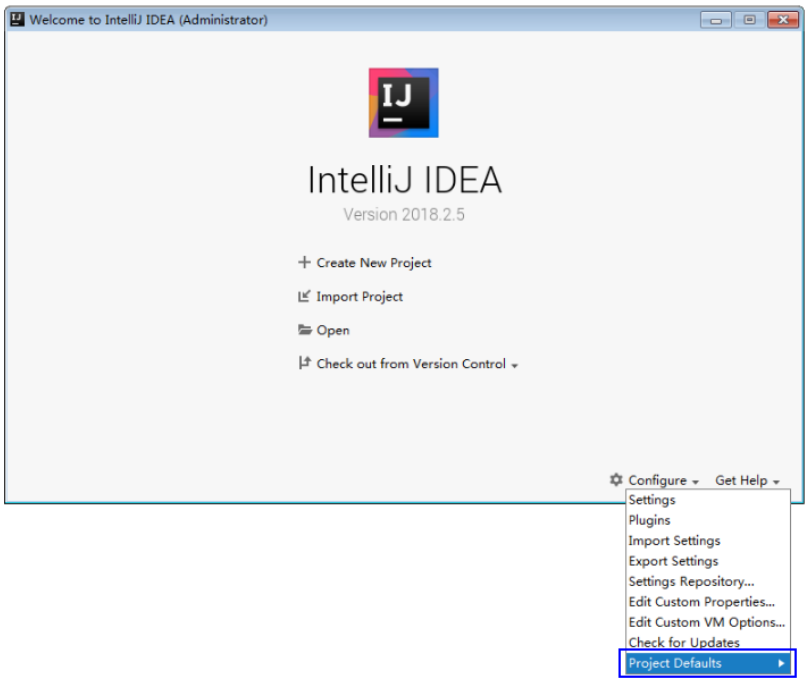
1. Start the IntelliJ IDEA and select **Configure**.

Figure 29-10 Quick Start



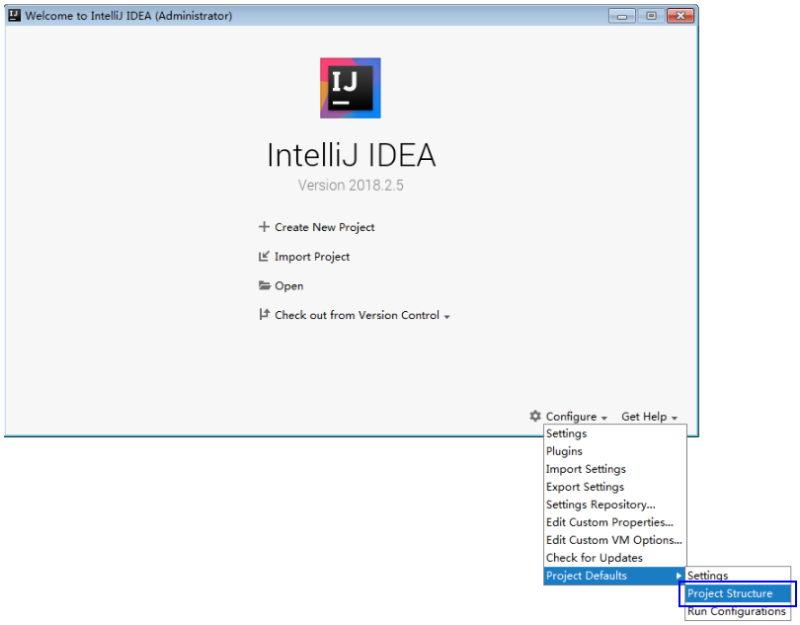
2. Select **Project Defaults** from the **Configure** drop-down list.

Figure 29-11 Configure



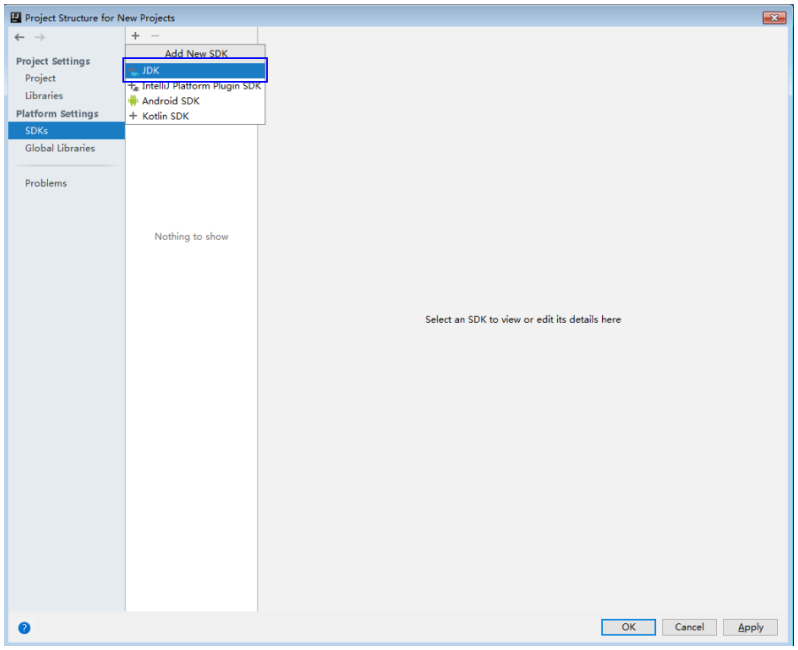
3. Select **Project Structure** from **Project Defaults**.

Figure 29-12 Project Defaults



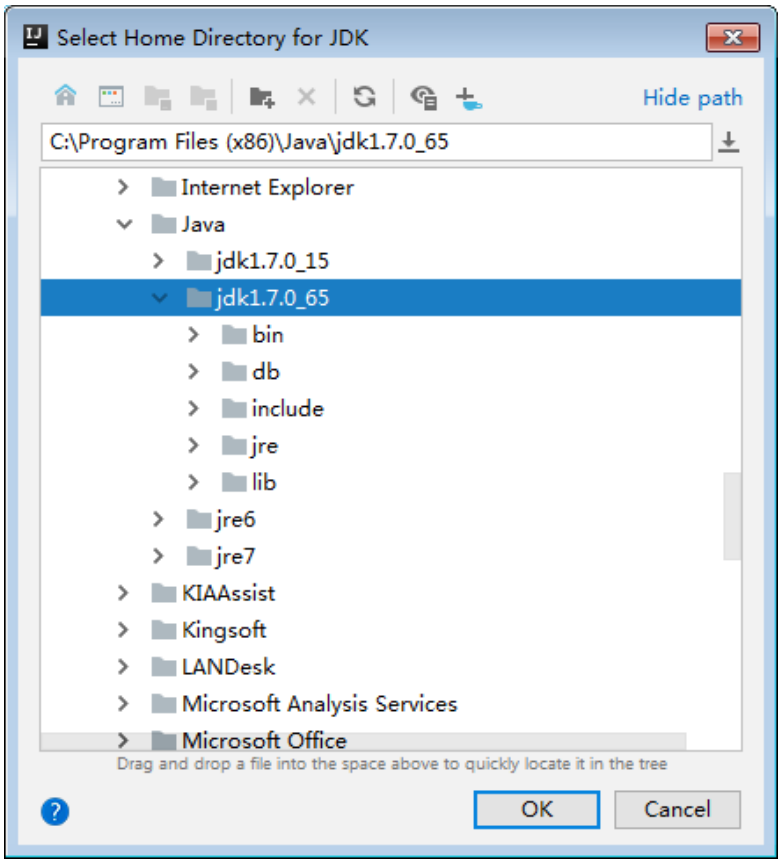
4. On the displayed **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 29-13 Add the JDK



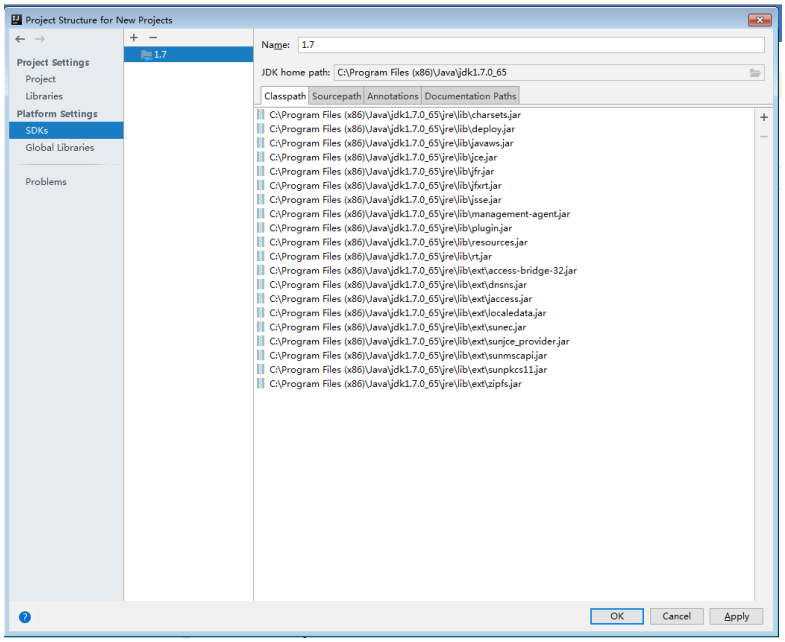
5. In the displayed **Select Home Directory for JDK** window, select a home directory for the JDK and click **OK**.

Figure 29-14 Select a home directory for the JDK



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

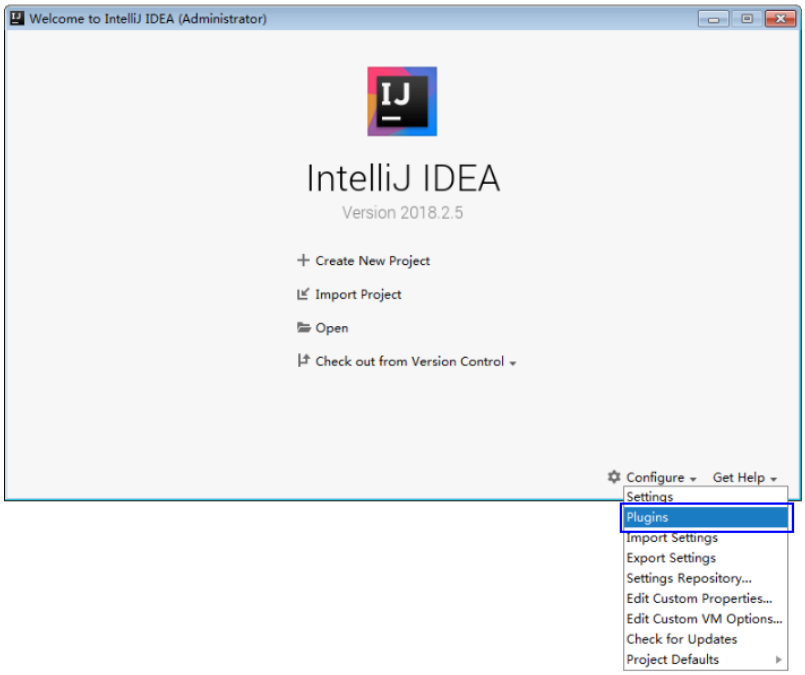
Figure 29-15 Complete the configuration



Step 3 (Optional) If the Scala development environment is used, install the Scala plug-in in IntelliJ IDEA.

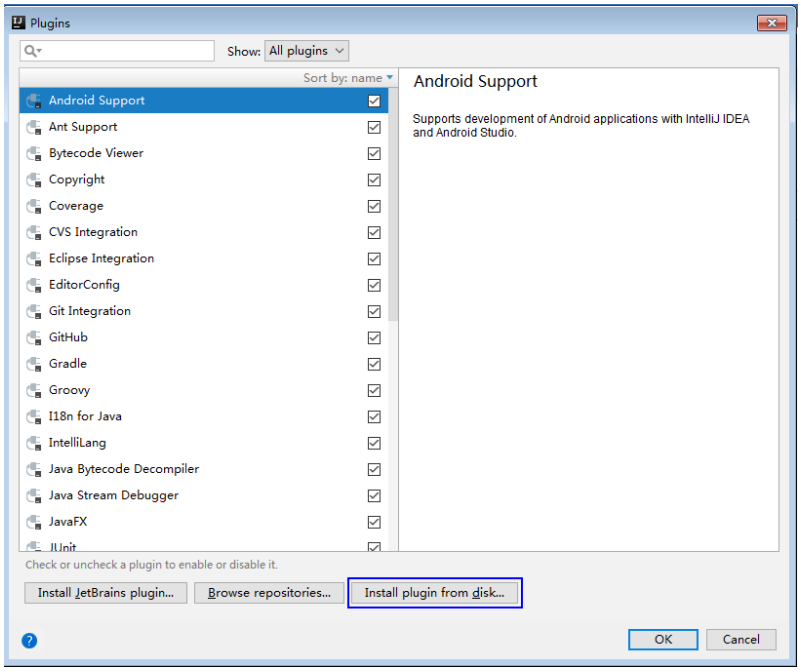
1. Select **Plugins** from the **Configure** drop-down list.

Figure 29-16 Plugins

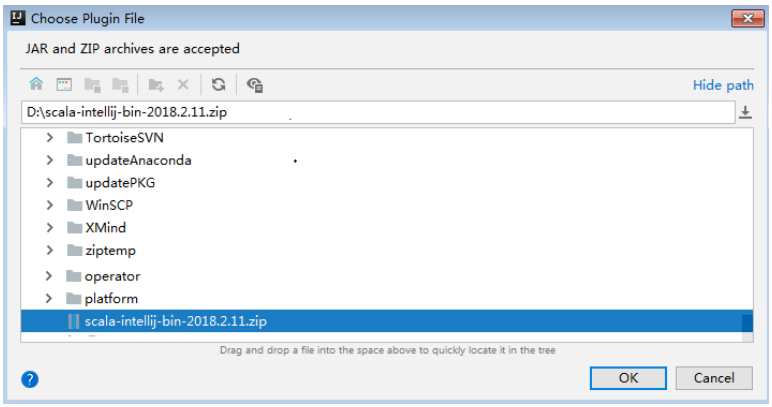


2. On the **Plugins** page, select **Install plugin from disk**.

Figure 29-17 Install plugin from disk

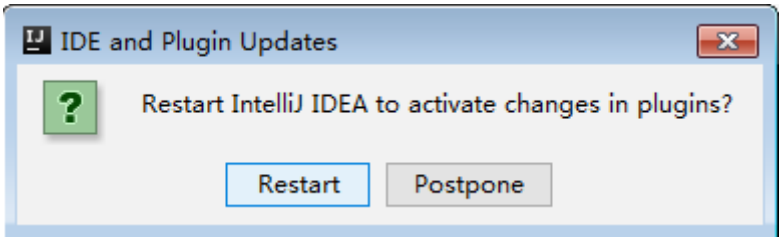


3. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.



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 29-18 Plugins Changed

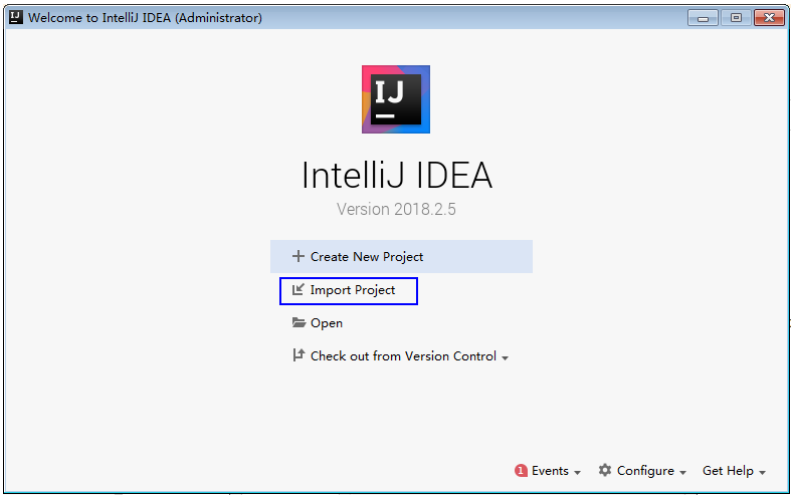


Step 4 Import the Java sample projects to the IDEA.

1. Start the IntelliJ IDEA, select **Import Project** on the **Quick Start** page.

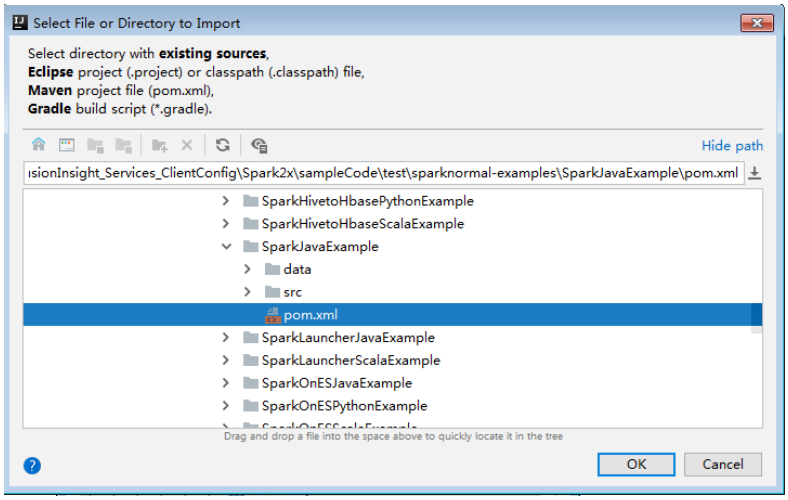
   Or, for the used IDEA tool, add projects directly from the IDEA home page. Select **File** > **Import project...** to import projects.

Figure 29-19 Import Project (on the Quick Start page)



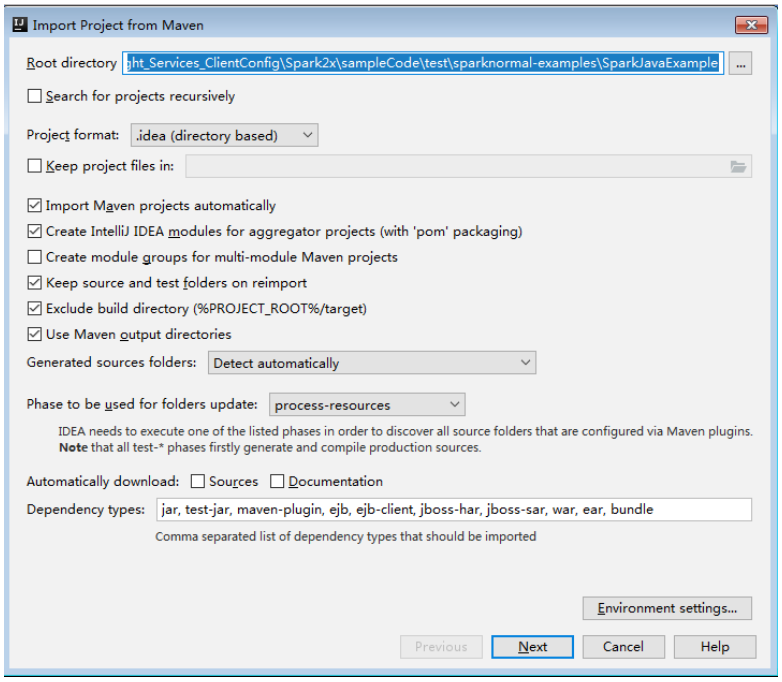
2. Select the directory to store the imported projects and the pom file, and click **OK**.

Figure 29-20 Select File or Directory to Import



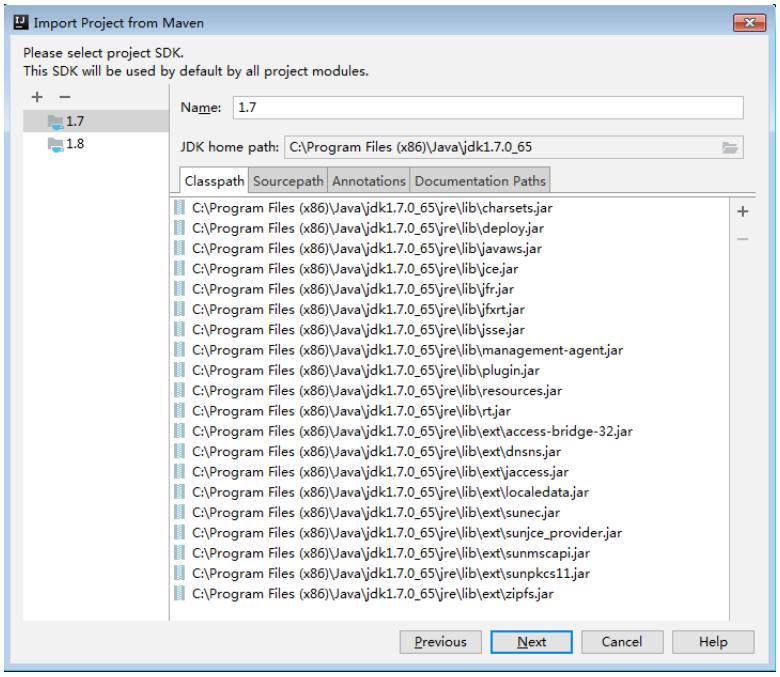
3. Confirm the import directory and project name, and click **Next**.

Figure 29-21 Import Project from Maven



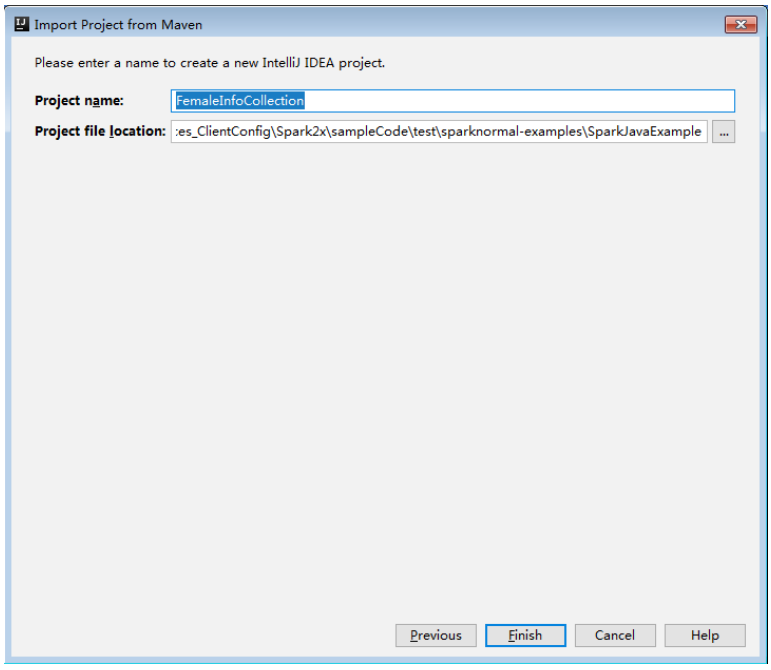
4. Select the projects to import and click **Next**.
5. Confirm the project JDK and click **Next**.

Figure 29-22 Select project SDK



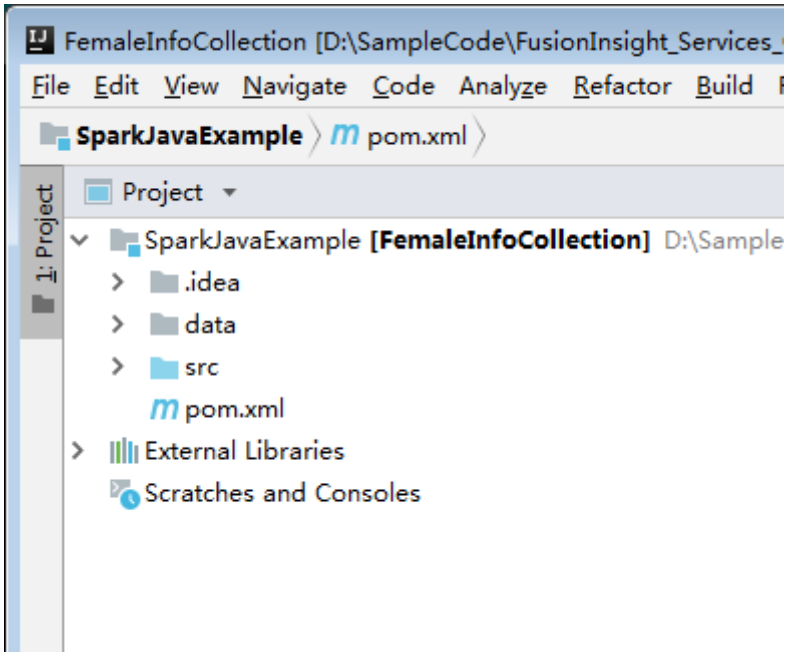
6. Confirm the project name and project file location, and click **Finish** to complete the import.

Figure 29-23 Confirm the project name and file location



7. After the import, the imported projects are displayed on the IDEA home page.

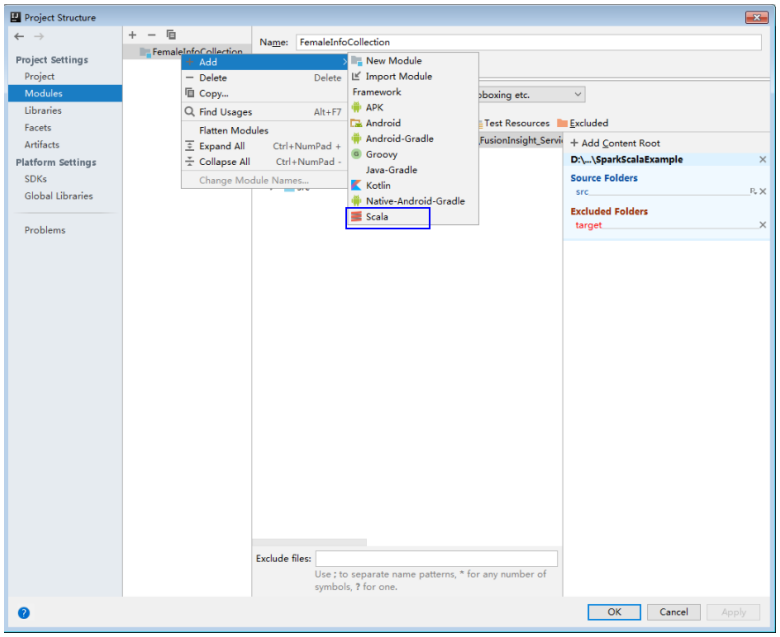
Figure 29-24 Imported project



Step 5 (Optional) If a sample application developed in Scala is imported, configure the language for the project.

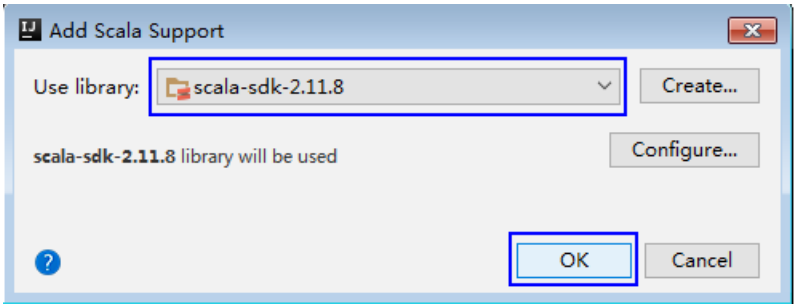
1. On the main page of the IDEA, choose **File > Project Structures...** to access the **Project Structure** page.
2. Choose **Modules**, right-click the project name, and choose **Add** > **Scala**.

Figure 29-25 Select the Scala language



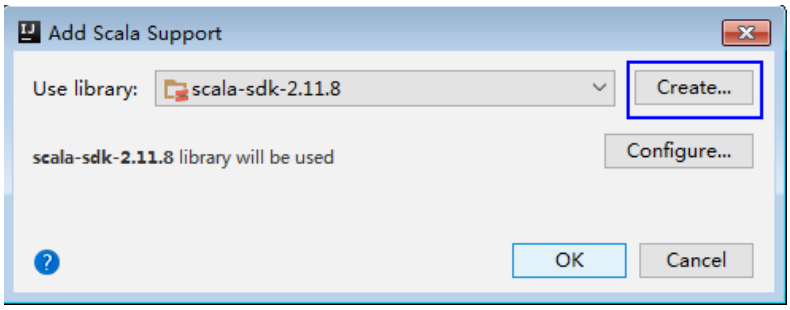
3. Wait until IDEA identifies Scala SDK, select the dependency JAR packages in the **Add Scala Support** dialog box, and then click **OK**.

Figure 29-26 Add Scala Support



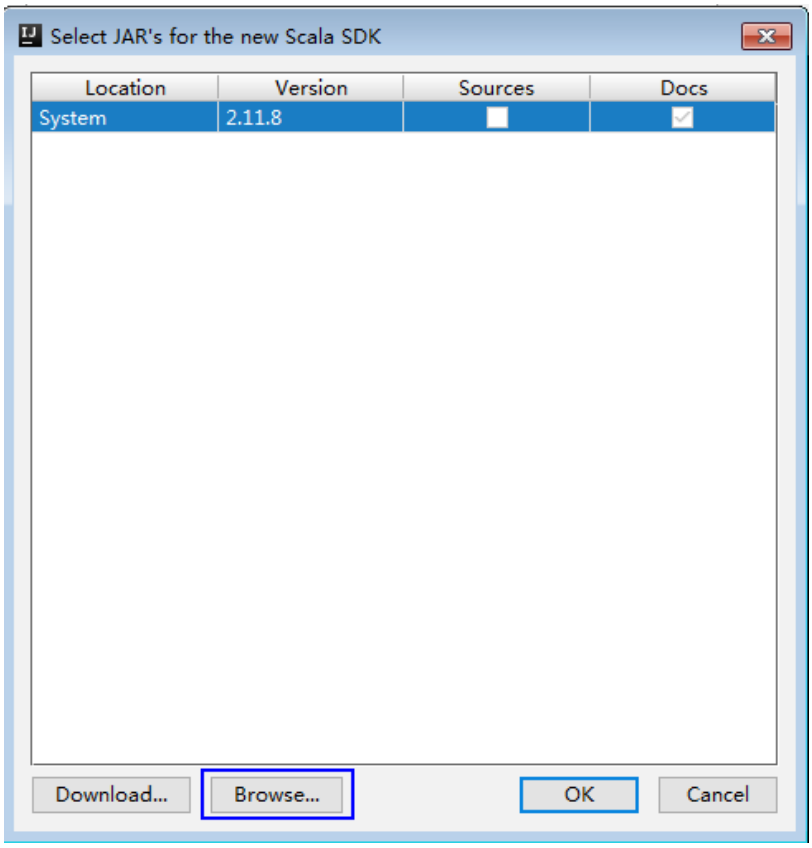
4. If IDEA fails to identify Scala SDK, you are required to create a Scala SDK.
   a. Click **Create**...

Figure 29-27 Create...



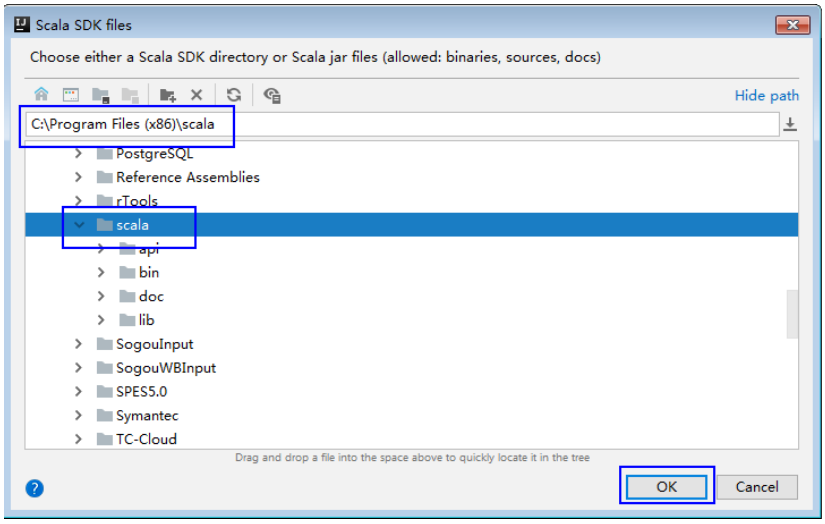
   b. On the **Select JAR';s for the new Scala SDK** page, click **Browse**...

Figure 29-28 Select JAR's for the new Scala SDK



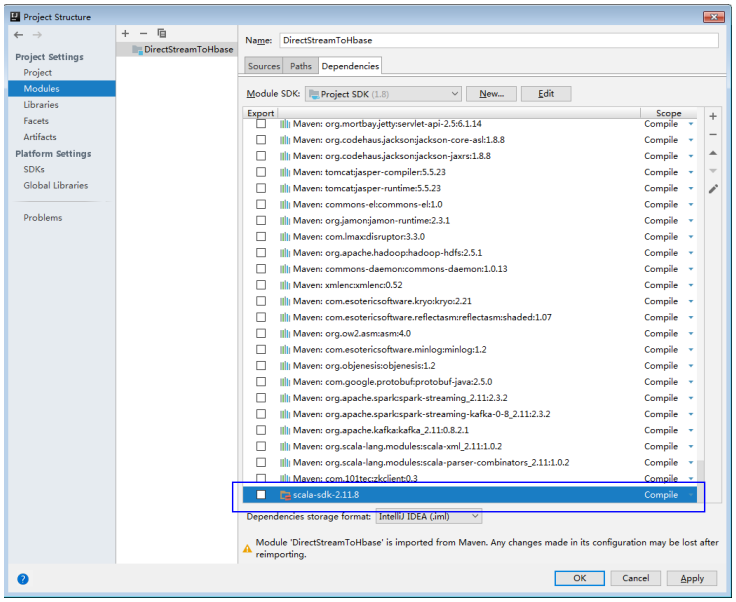
c. In the **Scala SDK files** window, select the scala sdk directory, and then click **OK**.

Figure 29-29 Scala SDK files



5. Click **OK**.

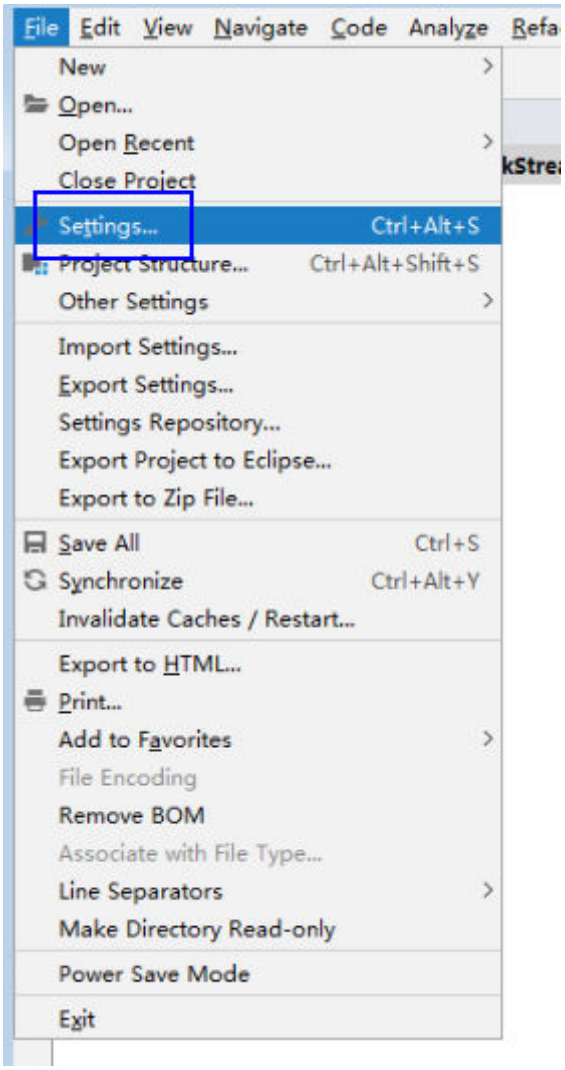
Figure 29-30 Successful configuration



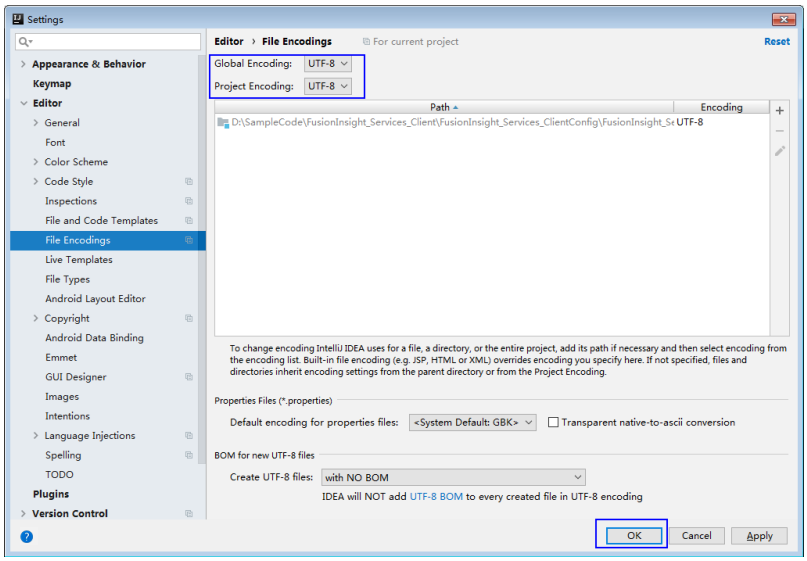
Step 6 Set the file encoding of IDEA and solve the display of garble characters.

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

Figure 29-31 Choose 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 29-6 Sample Code Path Description

Sample code project	Sample Name	Sample Development Language
SparkJavaExample	Spark Core Project	Java
SparkScalaExample	Spark Core Project	Scala
SparkPyhtonExample	Spark Core Project	Python
SparkSQLJavaExample	Spark SQL Project	Java
SparkSQLScalaExample	Spark SQL Project	Scala
SparkSQLPythonExample	Spark SQL Project	Python
SparkStreamingJavaExample	Streaming Connecting to Kafka0-8	Java
SparkStreamingScalaExample	Streaming Connecting to Kafka0-8	Scala
SparkStreamingPythonExample	Streaming Connecting to Kafka0-8	Python
SparkThriftServerJavaExample	Accessing the Spark SQL Through JDBC	Java
SparkThriftServerScalaExample	Accessing the Spark SQL Through JDBC	Scala

Sample code project	Sample Name	Sample Development Language
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
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

Sample code project	Sample Name	Sample Development Language
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
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

Sample code project	Sample Name	Sample Development Language
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
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

29.2.3 Creating a New Project (Optional)

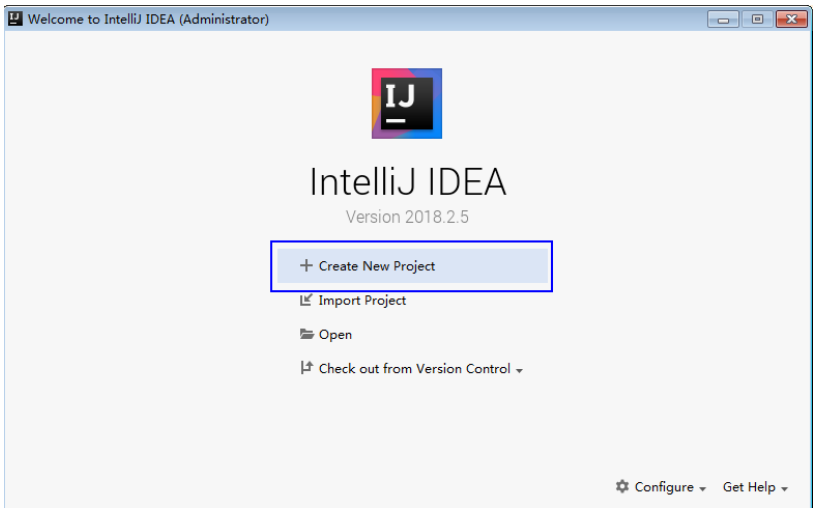
Scenario

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**.

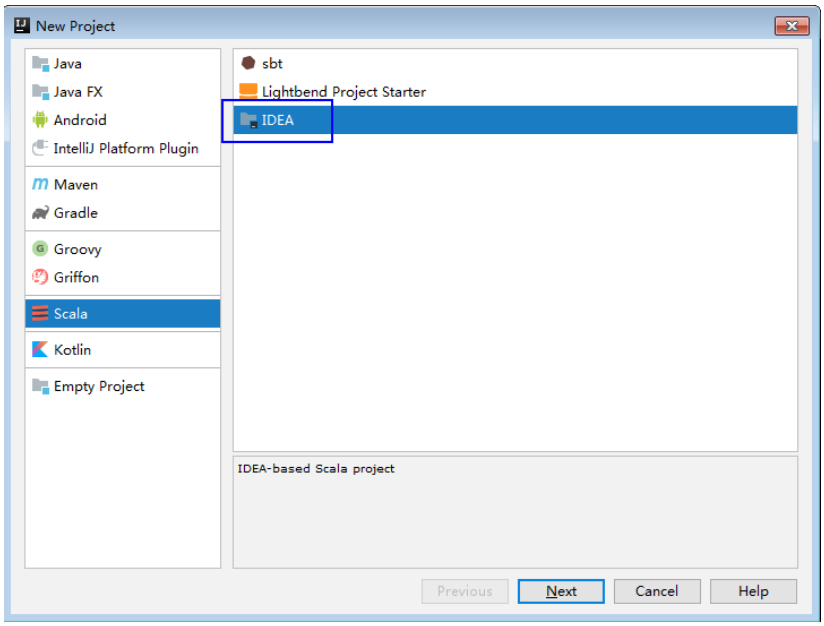
Figure 29-32 Create a project



Step 2 On the **New Project** page, select **Scala** as the development environment, select **IDEA**, and click **Next**.

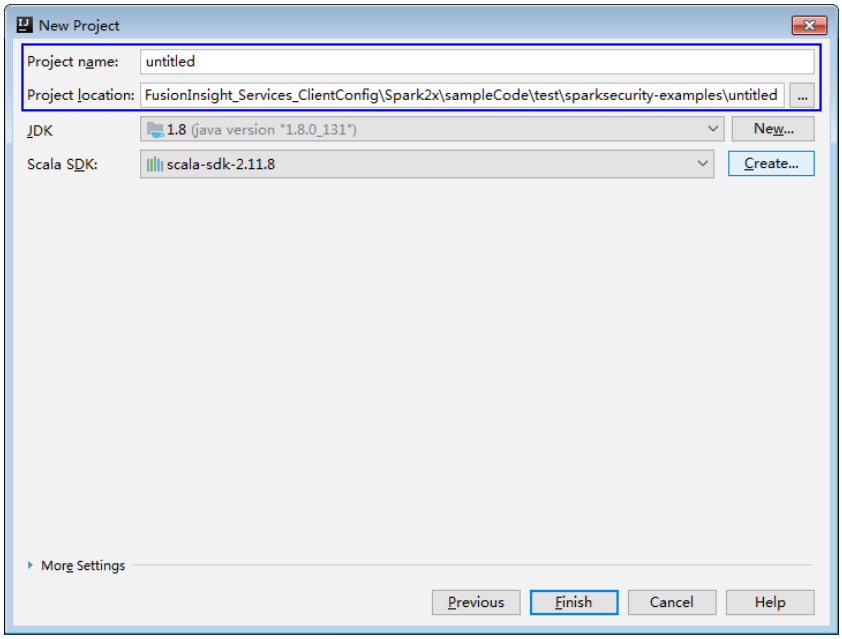
If a new project in Java is required, choose corresponding parameters.

Figure 29-33 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 29-34 Add the project information



----End

29.2.4 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

29.3 Developing the Project

29.3.1 Spark Core Project

29.3.1.1 Instance

Instance Description

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).

Step 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.

Step 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**:

1. On the Linux FusionInsight client, run ***hadoop fs -mkdir /tmp/input*** (a hdfs dfs command provides the same function).
2. Go to the /**tmp/input** directory on the HDFS client, on the Linux FusionInsight client, run ***hadoop fs -put*** *input_data1.txt /tmp/input* and ***hadoop fs -put*** *input_data2.txt /tmp/input*.

----End

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.

29.3.1.2 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());
   }
```


29.3.1.3 Scala 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.

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()
```

29.3.1.4 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()
```

29.3.2 Spark SQL Project

29.3.2.1 Instance

Instance Description

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).

Step 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.

Step 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**:

1. On the Linux FusionInsight client, run ***hadoop fs -mkdir /tmp/input*** (a hdfs dfs command provides the same function).
2. Go to the /**tmp/input** directory on the HDFS client, on the Linux FusionInsight client, run ***hadoop fs -put*** *input_data1.txt /tmp/input* and ***hadoop fs -put*** *input_data2.txt /tmp/input*.

----End

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.

29.3.2.2 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 (The class name and file name must be the same as those in the actual code. The following is only an example).

Example 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**

29.3.2.3 Scala 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 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**

29.3.2.4 Python Example Code

Function

Collect information about female netizens who have spent more than 2 hours in online shopping on the weekend.

Example 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()
```

29.3.3 Accessing the Spark SQL Through JDBC

29.3.3.1 Instance

Scenario

Spark on HBase allows users to create HBase tables on the JDBCServer, store data to JDBCServer tables by running the HBase command, and perform other operations.

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 example 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 example 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*

29.3.3.2 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

29.3.3.3 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

29.3.4 Spark on HBase

29.3.4.1 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/

  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.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:]
 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 and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.AvroSource().execute(spark._jsc)
# Stop SparkSession spark.stop().
```


29.3.4.2 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

  o 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 threquired 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()
```

29.3.4.3 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 *'b*ulktable','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.JavaHBaseBulkPutExample 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.JavaHBaseBulkPutExample 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()
```

29.3.4.4 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 **3.5.3 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 spark-hbase-python-test-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()
```

29.3.4.5 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 **3.5.3 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.JavaHBaseBulkDeleteExample 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 HBaseButDeleteExample.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.JavaHBaseBulkDeleteExample 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 HBaseButDeleteExample.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()
```

29.3.4.6 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()
```

29.3.4.7 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','fc1'*

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.JavaHBaseForEachPartitionExample 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.JavaHBaseForEachPartitionExample 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(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);
         }
         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()
```

29.3.4.8 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 **3.5.1 Performing Operations 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.JavaHBaseDistributedScanExample 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.JavaHBaseDistributedScanExample 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 int 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()
```


29.3.4.9 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 **3.5.7 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.

```
# -*- 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()
```

29.3.4.10 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.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 --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 eample, 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()
```

29.3.5 Reading Data from HBase and Write It Back to HBase

29.3.5.1 Instance

Scenario

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 user1 in the current day is 100 CNY.

In table2, key=1 and cf:cid=1000 indicate that the history consumption amount of user1 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 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.

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 example code.

  bin/spark-submit --jars --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*

29.3.5.2 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"));//colomn 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 colomn family and colomn 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();
      }
    }
  }
 }

}
```

29.3.5.3 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")) //colomn 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 colomn family and colomn 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 MyRegistrator extends KryoRegistrator {
 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])
  }
}
```

29.3.5.4 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.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()
```

29.3.6 Reading Data from Hive and Write It to HBase

29.3.6.1 Instance

Scenario

Assume that table **person** of Hive stores the user consuption 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 comsuption amount of user1 in the current day is 100 CNY.

In table2, key=1 and cf:cid=1000 indicate that the history comsuption amount of user1 is 1000 CNY.

The Spark application shall achieve the following function:

Add the current consumption amount (100) to the history consumption amount (1000).

The runing 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 a Linux-based HDFS client, run the ***hadoop fs -mkdir /tmp/input*** command (or ***hdfs dfs -mkdir /tmp/input***) to create the **/tmp/input** directory.
   b. On a Linux-base HDFS client, run ***hadoop fs -put*** *log1.txt /tmp/input* command to import data files.

Step 2 Ensure that JDBCServer is started. Use the Beeline command 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.

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 example 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*

29.3.6.2 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
        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();
      }
    }
  }
 }
}
```

29.3.6.3 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]) {

  // 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.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()
        }
      }
    }
  }

}
```

29.3.6.4 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()
```

29.3.7 Streaming Connecting to Kafka0-10

29.3.7.1 Instance

Scenario

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 include the absolute path of the Kafka JAR package of the Spark client, for example: /opt/client/Spark2x/spark/jars/*:/opt/client/Spark2x/spark/jars/streamingClient010/*

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.

NOTE

- For the dependency package whose version number contains "hw-ei", download from the Huawei open-source image site.
- For the dependency package whose version number does not contain "hw-ei", obtain them from the Maven central repository.

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[*]}")**

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):

- **Example 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>

- Spark Streaming Write To Kafka 0-10 example 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.JavaDstreamKafkaWriter /opt/SparkStreamingKafka010JavaExample-1.0.jar <groupId> <brokers> <topics>

29.3.7.2 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 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");

    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> sentData = new ArrayList();
  sentData.add("kafka_writer_test_msg_01");
  sentData.add("kafka_writer_test_msg_02");
  sentData.add("kafka_writer_test_msg_03");

  // Create Java RDD queue
  Queue<JavaRDD<String>> sent = new LinkedList();
  sent.add(ssc.sparkContext().parallelize(sentData));

  // 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();
 }
}
```

29.3.7.3 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;
      }
    }).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()
  }
}
```

29.3.8 Structured Streaming Project

29.3.8.1 Instance

Scenario

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 package 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.

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.
- **<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, which can be a local path or an HDFS path.

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-clientsjar 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 example 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 example 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>*

29.3.8.2 Java Example 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.

```
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();
```



```
  }
}
```

29.3.8.3 Scala Example 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()
  }
}
```


29.3.8.4 Python Example 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.

Example 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()
```


29.3.9 Structured Streaming Stream-Stream Join

29.3.9.1 Overview

Scenario

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 the 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 section **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 project, you need to specify **<kafkaBootstrapServers>**, **<maxEventDelay>**, **<reqTopic>**, **<showTopic>**, **<maxShowDelay>**, **<clickTopic>**, **<maxClickDelay>**, **<triggerInterver>**, and **<checkpointDir**.

- **<kafkaBootstrapServers>** indicates the Kafka address for obtaining metadata.
- **<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.
- **<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, which can be a local path or an HDFS path.

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-clientsjar 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>*

29.3.9.2 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)
println(showStaticsQuery.status)
          println("-------------get clickStatic progress---------")
          //println(clickStaticsQuery.lastProgress)
println(clickStaticsQuery.status)
          Thread.sleep(10000)
        }
      }
    }).start

    spark.streams.awaitAnyTermination()

  }
}
```

29.3.10 Structured Streaming Status Operation

29.3.10.1 Scenario

Scenario

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 sample project, you need to specify **<brokers>**, **<subscribe-type>**, **<topic>**, and **<checkpointLocation>**.

- **<brokers>** indicates the Kafka address for obtaining metadata.
- **<subscribe-type>** indicates the Kafka consumption mode.
- <topic> indicates the Kafka topic to be consumed.
- <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.

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 cope the kafka-clientsjar file from the **$SPARK_HOME/jars/streamingClient010** directory to the **$SPARK_HOME/jars** directory. Otherwise, the "class not found" error is reported.

29.3.10.2 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()
  }
```

29.3.11 Synchronizing HBase Data from Spark to CarbonData

29.3.11.1 Instance

Instance Description

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

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
```

29.3.11.2 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);
}
```

29.3.12 Using Spark to Perform Basic Hudi Operations

29.3.12.1 Instance

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

1. Log in to the Spark client node and run the following commands:

   source *Client installation directory*/**bigdata_env**

   source *Client installation directory*/**Hudi/component_env**

2. 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 project.

     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 project.

     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 project.

     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.

29.3.12.2 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()}
```

29.3.12.3 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))
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)
```


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()
```


29.3.12.4 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> writeStatues = client.compact(instant.get());
```



```
    client.commitCompaction(instant.get(), writeStatues, Option.empty());
}
```

29.3.13 Compiling User-defined Configuration Items for Hudi

29.3.13.1 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\"},
{\"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;
    }

}
```

29.3.13.2 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>
    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;
    }
}
```

29.4 Commissioning the Application

29.4.1 Commissioning Applications on Windows

29.4.1.1 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.

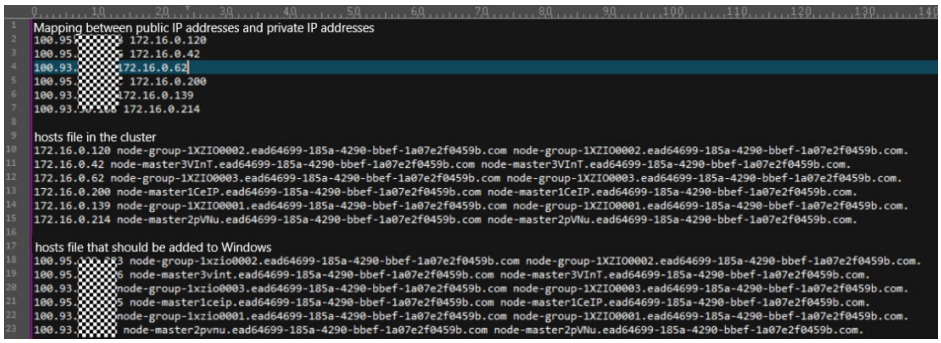
Procedure

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.

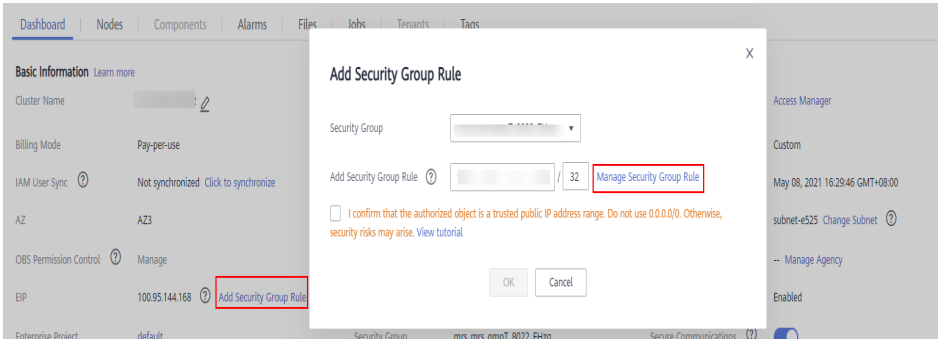
   For details, see **Virtual Private Cloud** > **User Guide** > **EIP** > **Assigning an EIP and Binding It to an ECS**.

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.

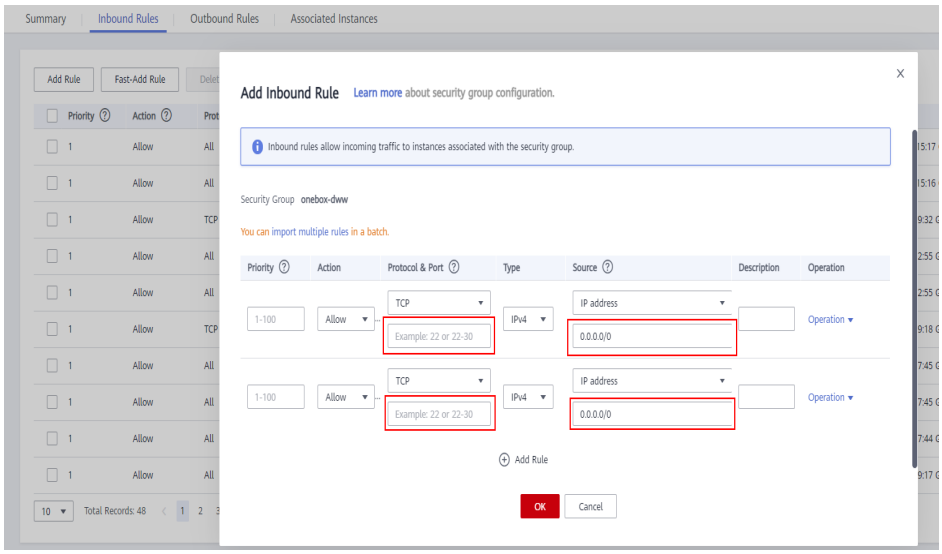


Step 2 Configure security group rules for the cluster.

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**, 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 4 Before running the sample code, add **.master("local").config("spark.driver.host", "localhost")** to **SparkSession** to set the local running mode for Spart.

```
object FemaleInfoCollection {
  def main (args: Array[String]) {
//    if (args.length < 1) {
//      System.err.println("Usage: CollectFemaleInfo <file>")
//      System.exit(1)
//    }

    // Configure the Spark application name.
    val spark = SparkSession
      .builder()
      .appName("CollectFemaleInfo")
      .master("local")
      .config("spark.driver.host", "localhost")
      .getOrCreate()
    // Initializing Spark


    // Read data. This code indicates the data path that the input parameter args(0) specifies.
    val text = spark.sparkContext.textFile("/tmp/sparktest/")
    // Filter the data information about the time that female netizens spend online.
    val data = text.filter(_.contains("female"))
```

----End

29.4.1.2 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.

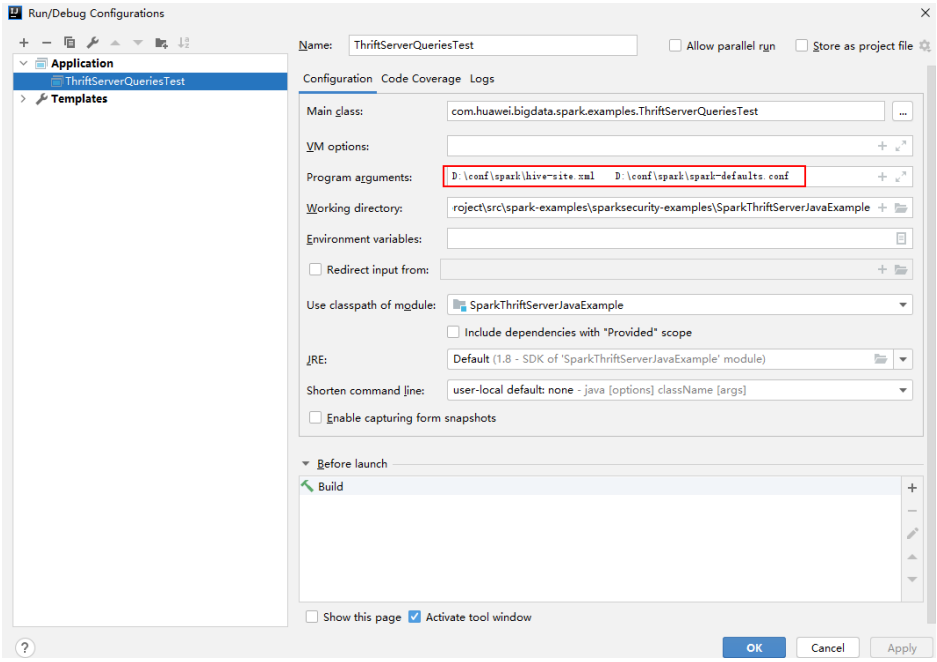
1. Create a data text file on Linux and save the following data to the data file:
   ```
   Miranda,32
   Karlie,23
   Candice,27
   ```
2. On the HDFS client running the Linux OS, run the **hadoop fs -mkdir /data** command (or the **hdfs dfs** command) to create a directory.
3. On the HDFS client running 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

29.4.1.3 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
```

29.4.2 Commissioning an Application in Linux

29.4.2.1 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

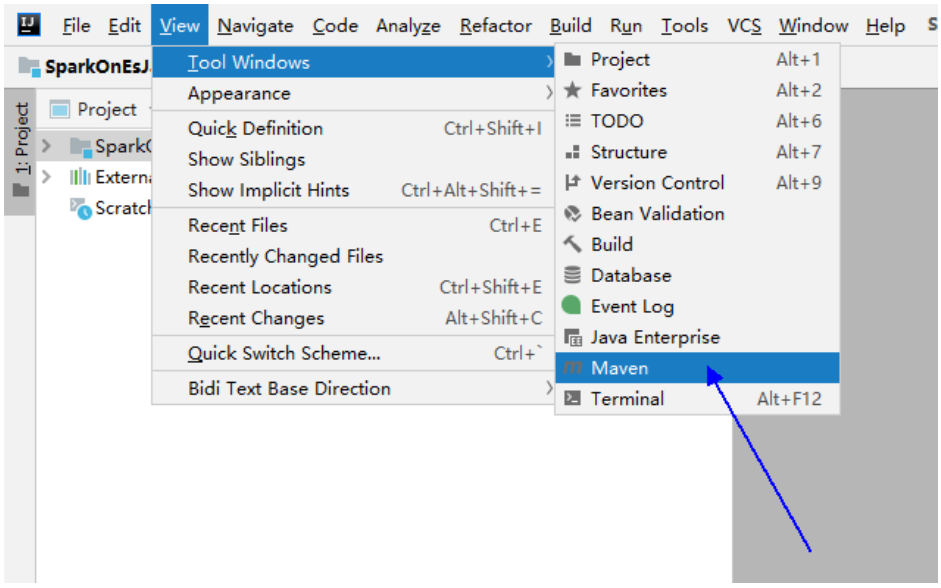
- The Spark application developed in Python does not need to build Artifacts as a jar. You just need to copy the sample projects to the compiler.
- It is needed to ensure that the version of Python installed on the worker and driver is consistent, otherwise the following error will be reported: "Python 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 **Methods for Obtaining Sample Projects**.

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.

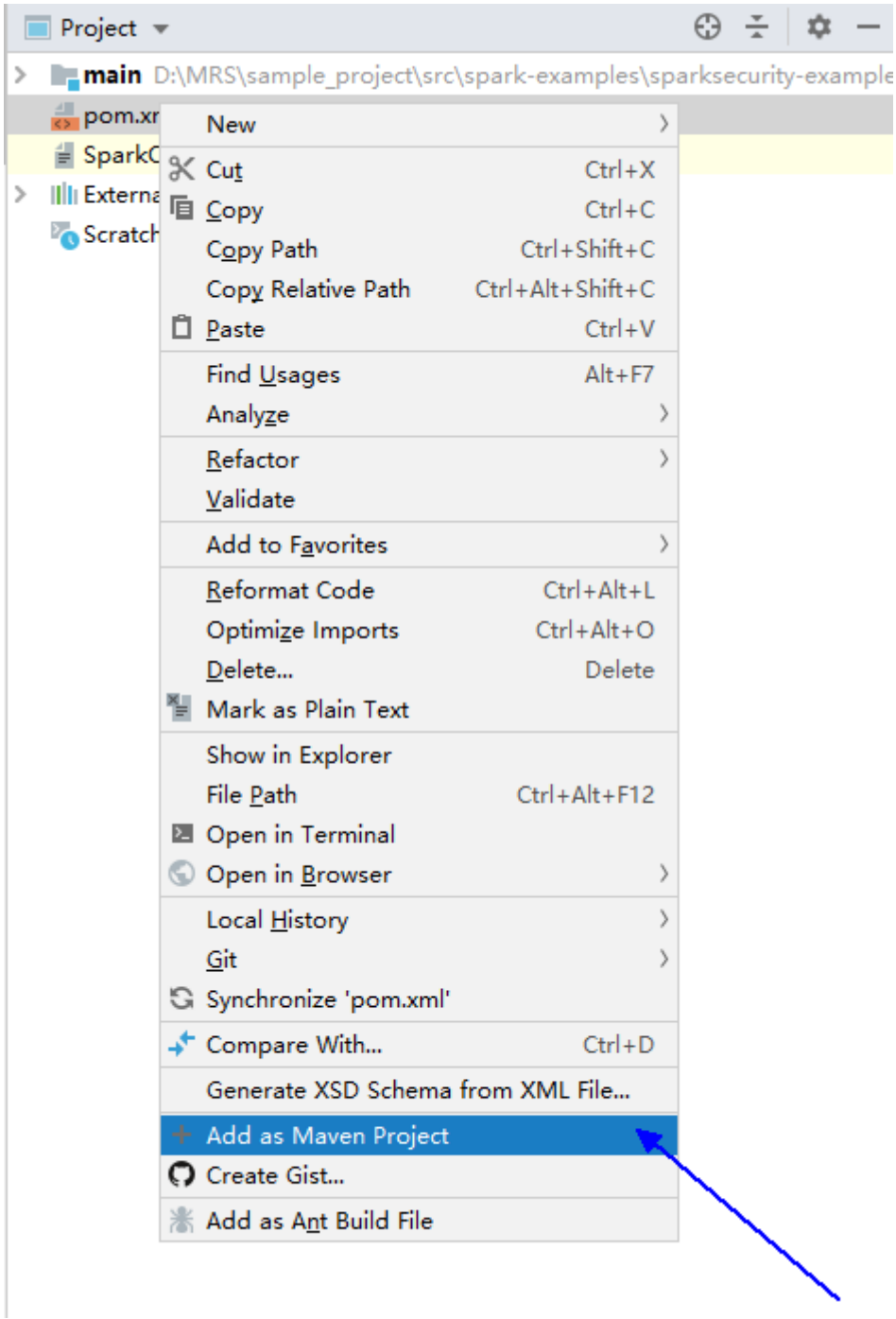
Figure 29-35 Openning the Maven tool window



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.

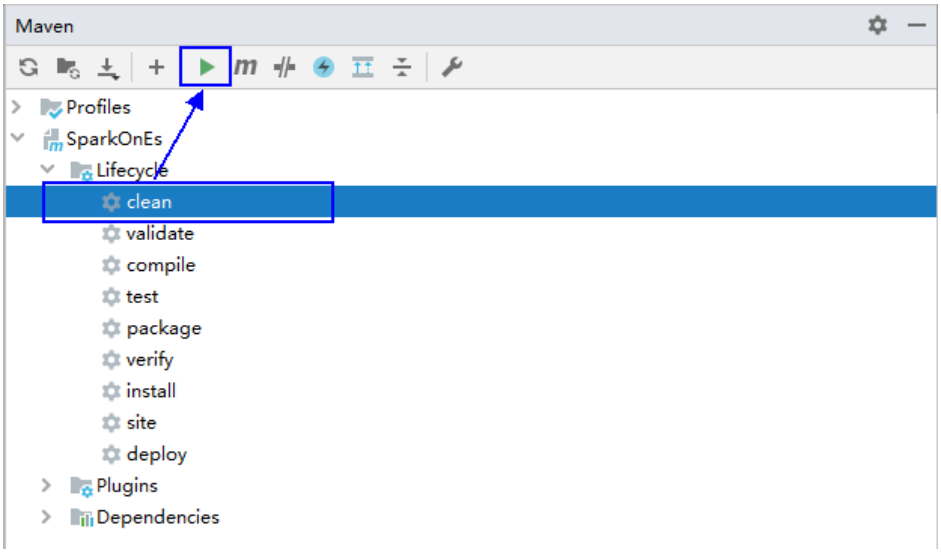
Figure 29-36 Adding a Maven project



Step 2 Use Maven to generate a JAR file.

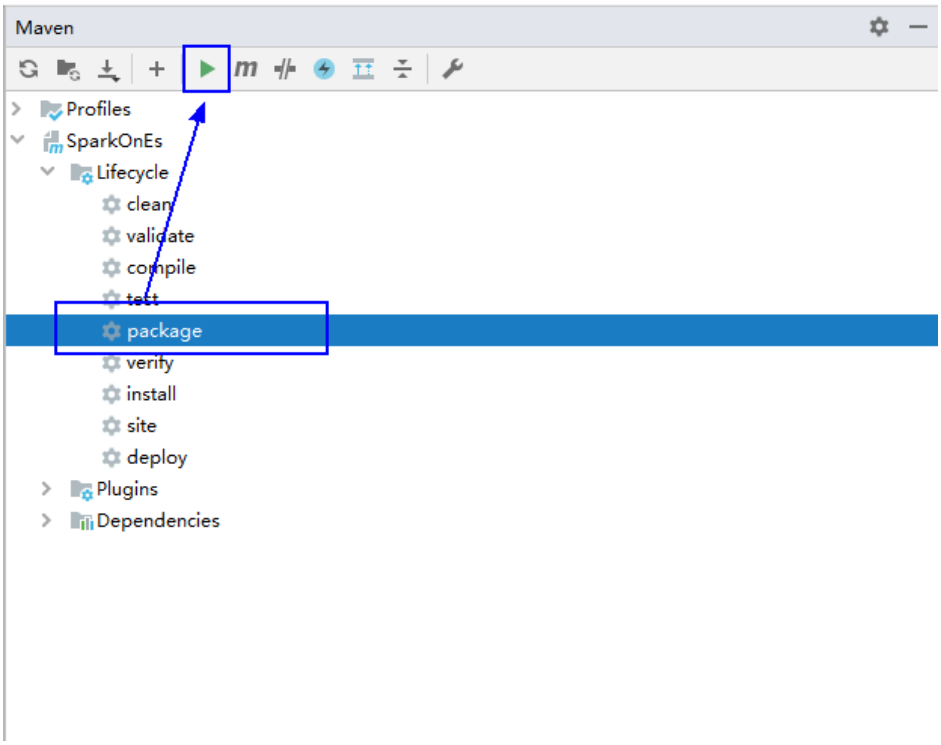
1. In the Maven tool window, select **clean** from **Lifecycle** to execute the Maven building process.

Figure 29-37 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 29-38 Selecting **package** from **Lifecycle** and execute the Maven build process.



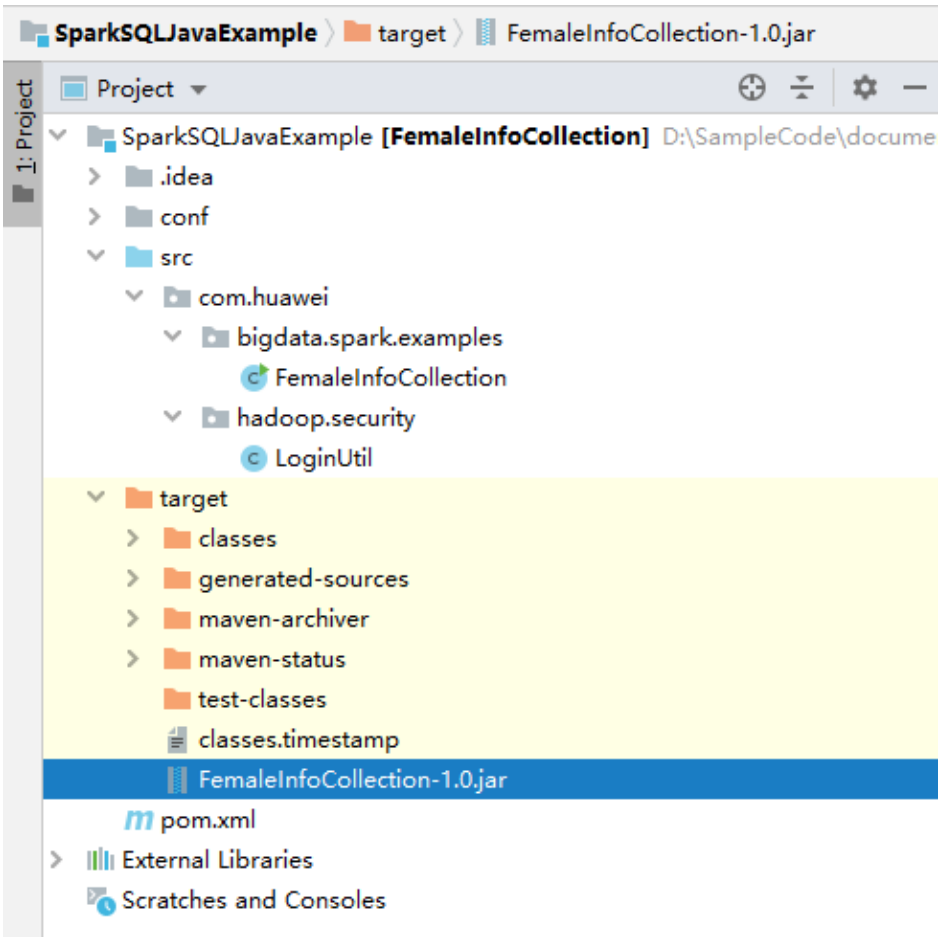
If the following information is displayed in **Run:**, the packaging is successful.

Figure 29-39 Packaging success message

```
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ FemaleInfoCollection ---
[INFO]
[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 29-40 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

29.4.2.2 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 WebUI.
- 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 the final column **ApplicationMaster** of the application information to access the Spark UI.

  – 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. *Log related information can be referenced to**Spark2x Logs**.*

29.5 More Information

29.5.1 Common APIs

29.5.1.1 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 level class, including MEMORY_ONLY, DISK_ONLY, and MEMORY_AND_DISK.

The JavaRDD supports two types of operations, transformation and action. **Table 29-7** and **Table 29-8** show the common methods.

Table 29-7 Transformation

Method	Description
<R> JavaRDD<R> map(Function<T,R> f)	Return a new RDD by applying a function to all elements of this RDD.
JavaRDD<T> filter(Function<T,Boolean> f)	Return a new RDD containing only the elements that satisfy a predicate.
<U> JavaRDD<U> flatMap(FlatMapFunction<T,U> f)	Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
JavaRDD<T> sample(boolean withReplacement, double fraction, long seed)	Return a sampled subset of this RDD.
JavaRDD<T> distinct(int numPartitions)	Return a new RDD containing the distinct elements in this RDD.
JavaPairRDD<K,Iterable<V>> groupByKey(int numPartitions)	Group the values for each key in the RDD into a single sequence. Hash-partitions the resulting RDD with into numPartitions partitions.
JavaPairRDD<K,V> reduceByKey(Function2<V,V,V> func, int numPartitions)	Merge the values for each key using an associative reduce function. This will also perform the merging locally on each mapper before sending results to a reducer, similarly to a "combiner" in MapReduce. Output will be hash-partitioned with numPartitions partitions.

Method	Description
JavaPairRDD<K,V> sortByKey(boolean ascending, int numPartitions)	Sort the RDD by key, so that each partition contains a sorted range of the elements. Calling collect or save on the resulting RDD will return or output an ordered list of records (in the save case, they will be written to multiple part-X files in the filesystem, in order of the keys).
JavaPairRDD<K,scala.Tuple2<V,W>> join(JavaPairRDD<K,W> other)	Return an RDD containing all pairs of elements with matching keys in this and other. Each pair of elements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in this and (k, v2) is in other. Performs a hash join across the cluster.
JavaPairRDD<K,scala.Tuple2<Iterable<V>,Iterable<W>>> cogroup(JavaPairRDD<K,W> other, int numPartitions)	For each key k in this or other, return a resulting RDD that contains a tuple with the list of values for that key in this as well as other.
JavaPairRDD<T,U> cartesian(JavaRDDLike<U,?> other)	Return the Cartesian product of this RDD and another one, that is, the RDD of all pairs of elements (a, b) where a is in this and b is in other.

Table 29-8 Action

Method	Description
T reduce(Function2<T,T,T> f)	Reduces the elements of this RDD using the specified commutative and associative binary operator.
java.util.List<T> collect()	Return an array that contains all of the elements in this RDD.
long count()	Return the number of elements in the RDD.
T first()	Return the first element in this RDD.
java.util.List<T> take(int num)	Take the first num elements of the RDD. This currently scans the partitions *one by one*, so it will be slow if a lot of partitions are required. In that case, use collect() to get the whole RDD instead.
java.util.List<T> takeSample(boolean withReplacement, int num, long seed)	Return a fixed-size sampled subset of this RDD in an array

Method	Description
void saveAsTextFile(String path, Class<? extends org.apache.hadoop.io.compress.CompressionCodec> codec)	Save this RDD as a compressed text file, using string representations of elements.
java.util.Map<K,Object> countByKey()	Count the appearance times of each key.
void foreach(VoidFunction<T> f)	Applies a function f to all elements of this RDD.
java.util.Map<T,Long> countByValue()	Return the count of each unique value in this RDD as a map of (value, count) pairs. The final combine step happens locally on the master, equivalent to running a single reduce task.

Table 29-9 New APIs of Spark core

API	Description
public java.util.concurrent.atomic.AtomicBoolean isSparkContextDown()	Determines whether sparkContext is shut down completely. The initial value is **false**. The value **true** indicates that sparkContext is shut down completely. The value **false** indicates that sparkContext is not shut down. For example, **jsc.sc().isSparkContextDown().get() == true** indicates that sparkContext is shut down completely.

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 29-10 Spark Streaming methods

Method	Description
JavaReceiverInputDStream<java.lang.String> socketStream(java.lang.String hostname,int port)	It 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)	It 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()	It starts the Spark Streaming calculation.
void awaitTermination()	It terminates the await of the process, which is similar to pressing Ctrl+C.
void stop()	It stops the Spark Streaming calculation.
<T> JavaDStream<T> transform(java.util.List<JavaDStream<?>> dstreams,Function2<java.util.List<JavaRDD<?>>,Time,JavaRDD<T>> transformFunc)	It 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)	It creates a unified Dstream from multiple DStreams with the same type and sliding window.

Table 29-11 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 29-12 Spark SQL methods

Method	Description
Row[] collect()	Return an array containing all DataFrame columns.
long count()	Return the number of DataFrame rows.
DataFrame describe(java.lang.String... cols)	Count the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
Row first()	Return the first row.
Row[] head(int n)	Return the first n rows.
void show()	Display the first 20 rows in table.
Row[] take(int n)	Return the first n rows in the DataFrame.

Table 29-13 Basic DataFrame Functions

Method	Description
void explain(boolean extended)	Print the logical plan and physical plan of the SQL.
void printSchema()	Print the schema information to the console.
registerTempTable	Register the DataFrame as a temporary table, whose period is bound to the SQLContext.
DataFrame toDF(java.lang.String... colNames)	Return a DataFrame whose columns are renamed.
DataFrame sort(java.lang.String sortCol,java.lang.String... sortCols)	Based on different columns, sort columns in ascending or descending orders.

Method	Description
GroupedData rollup(Column... cols)	Perform multiple-dimension crankback on the specified columns in the DataFrame.

29.5.1.2 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 level class, including MEMORY_ONLY, DISK_ONLY, and MEMORY_AND_DISK.

The RDD supports two types of operations, transformation and action. **Table 29-14** and **Table 29-15** show the common methods.

Table 29-14 Transformation

Method	Description
map[U](f: (T) => U): RDD[U]	Return a new RDD by applying a function to all elements of this RDD.
filter(f: (T) => Boolean): RDD[T]	Return a new RDD containing only the elements that satisfy a predicate.
flatMap[U](f: (T) => TraversableOnce[U]) (implicit arg0: ClassTag[U]): RDD[U]	Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.

Method	Description
sample(withReplacement: Boolean, fraction: Double, seed: Long = Utils.random.nextLong): RDD[T]	Return a sampled subset of this RDD.
union(other: RDD[T]): RDD[T]	Return a new RDD, contains source RDD and the group of RDD's elements.
distinct([numPartitions: Int]): RDD[T]	Return a new RDD containing the distinct elements in this RDD.
groupByKey(): RDD[(K, Iterable[V])]	Group the values for each key in the RDD into a single sequence. Hash-partitions the resulting RDD with into numPartitions partitions.
reduceByKey(func: (V, V) => V[, numPartitions: Int]): RDD[(K, V)]	Merge the values for each key using an associative reduce function. This will also perform the merging locally on each mapper before sending results to a reducer, similarly to a "combiner" in MapReduce. Output will be hash-partitioned with numPartitions partitions.
sortByKey(ascending: Boolean = true, numPartitions: Int = self.partitions.length): RDD[(K, V)]	Sort the RDD by key, so that each partition contains a sorted range of the elements. Calling collect or save on the resulting RDD will return or output an ordered list of records (in the save case, they will be written to multiple part-X files in the filesystem, in order of the keys).
join[W](other: RDD[(K, W)][, numPartitions: Int]): RDD[(K, (V, W))]	Return an RDD containing all pairs of elements with matching keys in this and other. Each pair of elements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in this and (k, v2) is in other. Performs a hash join across the cluster.
cogroup[W](other: RDD[(K, W)], numPartitions: Int): RDD[(K, (Iterable[V], Iterable[W]))]	For each key k in this or other, return a resulting RDD that contains a tuple with the list of values for that key in this as well as other.
cartesian[U](other: RDD[U])(implicit arg0: ClassTag[U]): RDD[(T, U)]	Return the Cartesian product of this RDD and another one, that is, the RDD of all pairs of elements (a, b) where a is in this and b is in other.

Table 29-15 Action

Method	Description
reduce(f: (T, T) => T):	Reduces the elements of this RDD using the specified commutative and associative binary operator.
collect(): Array[T]	Return an array that contains all of the elements in this RDD.
count(): Long	Return the number of elements in the RDD.
first(): T	Return the first element in this RDD.
take(num: Int): Array[T]	Take the first num elements of the RDD. This currently scans the partitions *one by one*, so it will be slow if a lot of partitions are required. In that case, use collect() to get the whole RDD instead.
takeSample(withReplacement: Boolean, num: Int, seed: Long = Utils.random.nextLong): Array[T]	Return a fixed-size sampled subset of this RDD in an array
saveAsTextFile(path: String): Unit	Save this RDD as a compressed text file, using string representations of elements.
saveAsSequenceFile(path: String, codec: Option[Class[_ <: CompressionCodec]] = None): Unit	Extra functions available on RDDs of (key, value) pairs to create a Hadoop SequenceFile, through an implicit conversion.
countByKey(): Map[K, Long]	Count 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]	Return the count of each unique value in this RDD as a map of (value, count) pairs. The final combine step happens locally on the master, equivalent to running a single reduce task.

Table 29-16 New APIs of Spark core

API	Description
isSparkContextDown:AtomicBoolean	Determines whether sparkContext is shut down completely. The initial value is **false**. The value **true** indicates that sparkContext is shut down completely. The value **false** indicates that sparkContext is not shut down. For example, **sc.isSparkContextDown.get() == true** indicates that sparkContext is shut down completely.

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 29-17 Spark Streaming methods

Method	Description
socketTextStream(hostname: String, port: Int, storageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK_SER_2): ReceiverInputDStream[String]	Reduces the elements of this RDD using the specified commutative and associative binary operator.
start():Unit	Return an array that contains all of the elements in this RDD.
awaitTermination(timeout: long):Unit	Return the number of elements in the RDD.

Method	Description
stop(stopSparkContext: Boolean, stopGracefully: Boolean): Unit	Return the first element in this RDD.
transform[T] (dstreams: Seq[DStream[_]], transformFunc: (Seq[RDD[_]], Time) ? RDD[T])(implicit arg0: ClassTag[T]): DStream[T]	Take the first num elements of the RDD. This currently scans the partitions *one by one*, so it will be slow if a lot of partitions are required. In that case, use collect() to get the whole RDD instead.
UpdateStateByKey(func)	Return a fixed-size sampled subset of this RDD in an array
window(windowLength, slideInterval)	Save this RDD as a compressed text file, using string representations of elements.
countByWindow(windowLength, slideInterval)	Count the appearance times of each key.
reduceByWindow(func, windowLength, slideInterval)	Applies a function f to all elements of this RDD.
join(otherStream, [numTasks])	Return the count of each unique value in this RDD as a map of (value, count) pairs. The final combine step happens locally on the master, equivalent to running a single reduce task.
DStreamKafkaWriter.writeToKafka()	Writes data from the DStream into Kafka in batch.
DStreamKafkaWriter.writeToKafkaBySingle()	Writes data from the DStream into Kafka one by one.

Table 29-18 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: An instance of the Spark SQL execution engine that integrates with data stored in Hive.

Table 29-19 Common Actions methods

Method	Description
collect(): Array[Row]	Return an array containing all DataFrame columns.
count(): Long	Return the number of DataFrame rows.
describe(cols: String*): DataFrame	Count the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
first(): Row	Return the first row.
Head(n:Int): Row	Return the first n rows.
show(numRows: Int, truncate: Boolean): Unit	Display the first 20 rows in table.
take(n:Int): Array[Row]	Return the first n rows in the DataFrame.

Table 29-20 Basic DataFrame Functions

Method	Description
explain(): Unit	Print the logical plan and physical plan of the SQL.
printSchema(): Unit	Print the schema information to the console.
registerTempTable(tableName: String): Unit	Register the DataFrame as a temporary table, whose period is bound to the SQLContext.
toDF(colNames: String*): DataFrame	Return a DataFrame whose columns are renamed.

29.5.1.3 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 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.
- pyspark.SparkConf: Spark application configuration class, which is used to configure the application name, execution model, 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: A broadcast variable created with SparkContext.broadcast(). Access its value through value.
- pyspark.StorageLevel: data storage level class, including MEMORY_ONLY, DISK_ONLY, and MEMORY_AND_DISK.
- pyspark.sql.SQLContext: Main entry point for SparkSQL functionality. A SQLContext can be used create SchemaRDDs, register SchemaRDDs as tables, execute SQL over tables, cache tables, and read parquet files.
- 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 29-21** and **Table 29-22** show the common methods.

Table 29-21 Transformation

Method	Description
map(f, preservesPartition-ing=False)	Return a new RDD by applying a function to all elements of this RDD.
filter(f)	Return a new RDD containing only the elements that satisfy a predicate.
flatMap(f, preservesPartition-ing=False)	Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
sample(withReplacement, fraction, seed=None)	Return a sampled subset of this RDD.
union(rdds)	Return a new RDD, contains source RDD and the group of RDD's elements.

Method	Description
distinct([numPartitions: Int]): RDD[T]	Return a new RDD containing the distinct elements in this RDD.
groupByKey(): RDD[(K, Iterable[V])]	Group the values for each key in the RDD into a single sequence. Hash-partitions the resulting RDD with into numPartitions partitions.
reduceByKey(func, numPartitions=None)	Merge the values for each key using an associative reduce function. This will also perform the merging locally on each mapper before sending results to a reducer, similarly to a "combiner" in MapReduce. Output will be hash-partitioned with numPartitions partitions.
sortByKey(ascending=True, numPartitions=None, keyfunc=function <lambda>)	Sort the RDD by key, so that each partition contains a sorted range of the elements. Calling collect or save on the resulting RDD will return or output an ordered list of records (in the save case, they will be written to multiple part-X files in the filesystem, in order of the keys).
join(other, numPartitions)	Return an RDD containing all pairs of elements with matching keys in this and other. Each pair of elements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in this and (k, v2) is in other. Performs a hash join across the cluster.
cogroup(other, numPartitions)	For each key k in this or other, return a resulting RDD that contains a tuple with the list of values for that key in this as well as other.
cartesian(other)	Return the Cartesian product of this RDD and another one, that is, the RDD of all pairs of elements (a, b) where a is in this and b is in other.

Table 29-22 Action

Method	Description
reduce(f)	Reduces the elements of this RDD using the specified commutative and associative binary operator.
collect()	Return an array that contains all of the elements in this RDD.
count()	Return the number of elements in the RDD.
first()	Return the first element in this RDD.

Method	Description
take(num)	Take the first num elements of the RDD. This currently scans the partitions *one by one*, so it will be slow if a lot of partitions are required. In that case, use collect() to get the whole RDD instead.
takeSample(withReplacement, num, seed)	Return a fixed-size sampled subset of this RDD in an array
saveAsTextFile(path, compressionCodecClass)	Save this RDD as a compressed text file, using string representations of elements.
saveAsSequenceFile(path, compressionCodecClass=None)	Extra functions available on RDDs of (key, value) pairs to create a Hadoop SequenceFile, through an implicit conversion.
countByKey()	Count the appearance times of each key.
foreach(func)	Applies a function f to all elements of this RDD.
countByValue()	Return the count of each unique value in this RDD as a map of (value, count) pairs. The final combine step happens locally on the master, equivalent to running a single reduce task.

Spark Streaming Common Interfaces

Spark Streaming mainly uses the following classes:

- pyspark.streaming.StreamingContext: A StreamingContext is the main entry point for Spark Streaming functionality. Besides the basic information (such as, cluster URL and job name) to internally create a SparkContext, it provides methods used to create DStreams from various input sources.
- 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: 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 29-23 Spark Streaming common interfaces

Method	Description
socketTextStream(hostname, port, storageLevel)	Reduces the elements of this RDD using the specified commutative and associative binary operator.
start()	Return an array that contains all of the elements in this RDD.
awaitTermination(timeout)	Return the number of elements in the RDD.
stop(stopSparkContext, stopGraceFully)	Return the first element in this RDD.
UpdateStateByKey(func)	Take the first num elements of the RDD. This currently scans the partitions *one by one*, so it will be slow if a lot of partitions are required. In that case, use collect() to get the whole RDD instead.
window(windowLength, slideInterval)	Return a fixed-size sampled subset of this RDD in an array
countByWindow(windowLength, slideInterval)	Save this RDD as a compressed text file, using string representations of elements.
reduceByWindow(func, windowLength, slideInterval)	Count the appearance times of each key.
join(other,numPartitions)	Applies a function f to all elements of this RDD. Return the count of each unique value in this RDD as a map of (value, count) pairs. The final combine step happens locally on the master, equivalent to running a single reduce task.

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: A variant of Spark SQL that integrates with 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 29-24 Spark SQL common Actions

Method	Description
collect()	Return an array containing all DataFrame columns.
count()	Return the number of DataFrame rows.
describe()	Count the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
first()	Return the first row.
head(n)	Return the first n rows.
show()	Display the first 20 rows in table.
take(num)	Return the first n rows in the DataFrame.

Table 29-25 Basic DataFrame Functions

Method	Description
explain()	Print the logical plan and physical plan of the SQL.
printSchema()	Print the schema information to the console.
registerTempTable(name)	Register the DataFrame as a temporary table, whose period is bound to the SQLContext.
toDF()	Return a DataFrame whose columns are renamed.

29.5.1.4 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/ODBC 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 normal mode, the JobHistory supports only the HTTP protocol. Therefore, use the HTTP protocol in the URL of the following command.

- Obtaining information about all applications on the JobHistory node:
  - Command:

    ```
    curl http://192.168.227.16:4040/api/v1/applications?mode=monitoring --insecure
    ```

    "192.168.227.16" indicates the service IP address of the JobHistory node; "4040" 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:

    After running this command, you can query information about all Spark applications in the current cluster. **Table 29-26** describes the parameters in response to this command.

    Table 29-26 Parameter description

Parameter	Description
id	Application ID.
name	Application name.

Parameter	Description
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 http://192.168.227.16:4040/api/v1/applications/application_1517290848707_0008?mode=monitoring --insecure
    ```
    "192.168.227.16" indicates the service IP address of the JobHistory node; "4040" indicates the port number of the JobHistory node; "pplication_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:

    After running this command, you can query the information about a Spark application. For the description of parameters in response to this command, see **Table 29-26**.
- Obtain the information about the Executor of a running application:
  - Command of alive executors list:
    ```
    curl http://192.168.169.84:8088/proxy/application_1478570725074_0046/api/v1/applications/application_1478570725074_0046/executors?mode=monitoring --insecure
    ```
  - Command of all executors(alive&dead) list:
    ```
    curl http://192.168.169.84:8088/proxy/application_1478570725074_0046/api/v1/applications/application_1478570725074_0046/allexecutors?mode=monitoring --insecure
    ```
    "192.168.195.232" indicates the service IP address of the master node of the ResourceManager; "8088" indicates the port number of the ResourceManager; "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-224:26010/node/containerlogs/
container_1478570725074_0049_01_000002/admin/stdout?start=-4096",
    "stderr" : "https://XTJ-224:26010/node/containerlogs/
container_1478570725074_0049_01_000002/admin/stderr?start=-4096"
  }
} ]
```


- Analysis:

  After running this command, you can query information about all Executors (including the driver) of the current application. **Table 29-27** describes the parameters in response to this command.

  Table 29-27 Parameter description

Parameter	Description
id	Executor ID.
hostPort	IP address and port number of the node where the Executor resides. Format: I*P address:port number*.
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 http://192.168.195.232:8088/proxy/application_1476947670799_0053/api/v1/applications/
    application_1476947670799_0053/SQL?mode=monitoring --insecure
    ```

    "192.168.195.232" indicates the service IP address of the master node of the ResourceManager; "8088" indicates the port number of the

ResourceManager; "application_1476947670799_0053" indicates the application ID in YARN; .

- JobHistory command:

```
curl http://192.168.227.16:4040/api/v1/applications/application_1478570725074_0004/SQL?mode=monitoring --insecure
```

"192.168.227.16" indicates the service IP address of the JobHistory node; "4040" indicates the port number of the JobHistory node; "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" : [ ],
    "successedJobs" : [ 0 ],
    "failedJobs" : [ ]
  } ],
  "sqls" : [ {
    "id" : 0,
    "status" : "COMPLETED",
    "description" : "getCallSite at SQLExecution.scala:48",
    "submissionTime" : "2016/11/08 15:39:00",
    "duration" : "2 s",
    "runningJobs" : [ ],
    "successedJobs" : [ 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). **Table 29-28** describes the parameters in response to this command.

Table 29-28 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.
successedJobs	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 http://192.168.195.232:8088/proxy/application_1476947670799_0053/api/v1/applications/application_1476947670799_0053/sqlserver?mode=monitoring --insecure
    ```

    "192.168.195.232" indicates the service IP address of the master node of the ResourceManager; "8088" indicates the port number of the ResourceManager; "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
      } ],
      "sqls" : [ {
        "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. **Table 29-29** describes the parameters in the queried session information; **Table 29-30** describes the parameters in the queried SQL statement information.

    Table 29-29 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.

Parameter	Description
finishTime	Time when the session ends the connection.
duration	Connection duration of the session.
totalExecute	Number of SQL statements executed by the session.

Table 29-30 SQL parameter description

Parameter	Description
user	User who executes the SQL statement.
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.

- Streaming related: obtaining the average input frequency, average scheduling delay, average execution duration, and average value of the overall delay.
  - Command:
    ```
    curl http://192.168.195.232:8088/proxy/application_1477722033672_0008/api/v1/applications/application_1477722033672_0008/streaming/statistics?mode=monitoring --insecure
    ```
    "192.168.195.232" indicates the service IP address of the master node of the ResourceManager; "8088" indicates the port number of the ResourceManager; "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.

29.5.2 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 ***./bin/spark-shell*** 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.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 "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

29.5.3 JDBCServer Interface

Overview

The JDBCServeris another implement of HiveServer2 in the Hive. The Spark SQL is used to process the SQL statement at its bottom. Therefore, theJDBCServer has better performance than the Hive.

The JDBCServeris a JDBC interface. Users can log in to theJDBCServerand access the Spark SQL data through the JDBC. When theJDBCServeris 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 theJDBCServeris 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 theJDBCServer, at least the host name and port of theJDBCServer must be configured. If Hive data is required, the uris of the hive metastore needs to be provided.

JDBCServer starts a JDBC service on port 22550 of the installation node by default. (If you want to change the port, configure the **hive.server2.thrift.port** parameter.) You can connect to JDBCServer using Beeline or running the JDBC client code to run SQL statements.

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**.

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=sparkthriftserver2x;"

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 randomJDBCServerinstance is selected for the connection to the client.

JDBC Client Codes

Log in to the JDBCServerby 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 JDBCServerHA solution and timeout of configuring theJDBCServer.

- The JDBCServer HA solution is described as follows:

  When multiple active nodes of JDBCServer provide 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.

29.5.4 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.
		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	**path**: file path, which is mandatory. **maxFilesPerTrigger**: maximum number of files in each trigger. The default value is infinity. **latestFirst**: whether to process limited number of new files. The default value is **false**. **fileNameOnly**: whether the file name is used as the new file for verification instead of using the complete path. (Default value: **false**)	Supported	Paths with wildcard are supported, but multiple paths separated by commas (,) are not supported. Files must be placed in a given directory in atomic mode, which can be implemented through file movement in most file systems.

Sources	Supported Options	Fault Tolerance Supported	Description
Socket Source	**host**: IP address of the connected node, which is mandatory. **port**: connected port, which is mandatory.	Not supported	-
Rate Source	**rowsPerSecond**: number of rows generated per second. The default value is **1**. **rampUpTime**: rising time before the speed specified by **rowsPerSecond** is reached. **numPartitions**: concurrency degree for generating data rows.	Supported	-
Kafka Source	For details, see **https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html**	Supported	-

3. Fault tolerance list of Sink

Sinks	Supported Output Mode	Supported Options	Fault Tolerance	Description
File Sink	Append	**Path**: 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 **https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html**.	at-least-once	For details, see **https://spark.apache.org/docs/3.1.1/structured-streaming-kafka-integration.html**.

Sinks	Supported Output Mode	Supported Options	Fault Tolerance	Description
Foreach Sink	Append, Update, Complete	None	Depends on ForeachWriter.	For details, see **https://spark.apache.org/docs/3.1.1/structured-streaming-programming-guide.html#using-foreach**.
ForeachBatch Sink	Append, Update, Complete	None	Depends on operator.	For details, see **https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#using-foreach-and-foreachbatch**.
Console Sink	Append, Update, Complete	**numRows**: number of rows printed in each round. The default value is **20**. **truncate**: whether to clear the output when the output is too long. The default value is **true**.	Not supported	-
Memory Sink	Append, Complete	None	Not supported. In complete mode, the entire table is rebuilt after the query is restarted.	-

29.5.5 FAQ

29.5.5.1 How to Add a User-Defined Library

Question

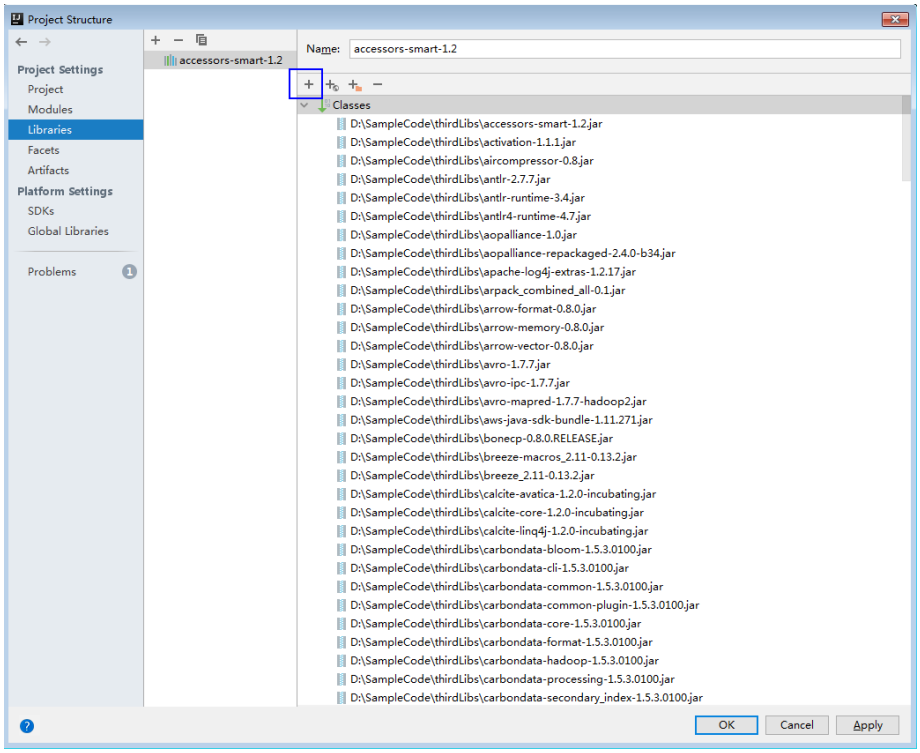
In the development of the Spark application, users may add a user-defined dependent library which is different from the sample project. This section describes how to add a dependent library with user-defined codes into the project.

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 29-41 Add the Dependent Library

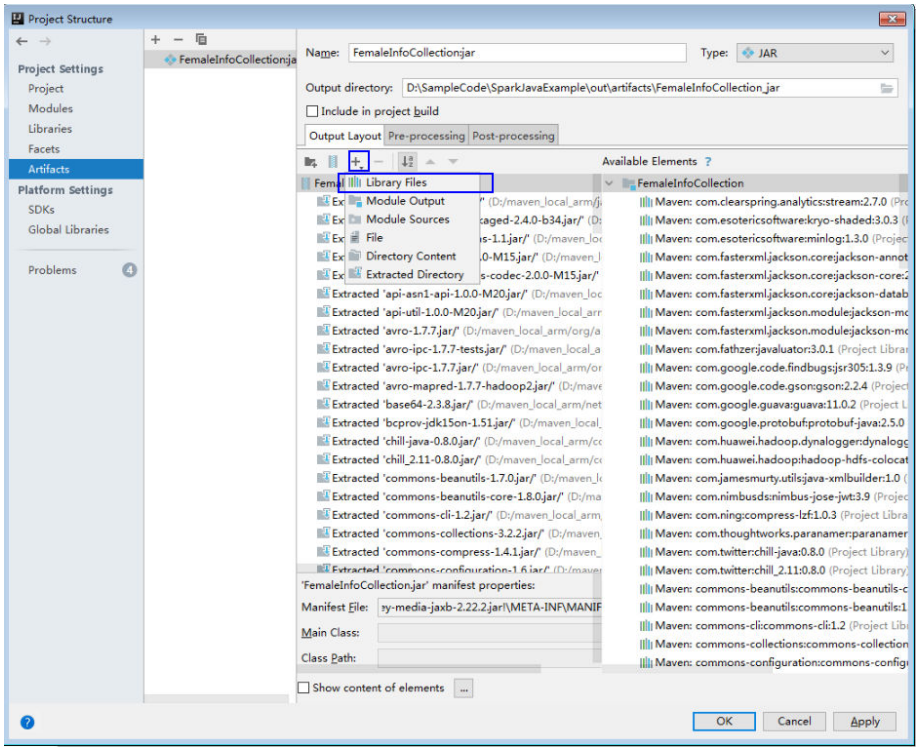


Step 3 Click **Apply** to load the dependent library and click **OK** to complete the configuration.

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.

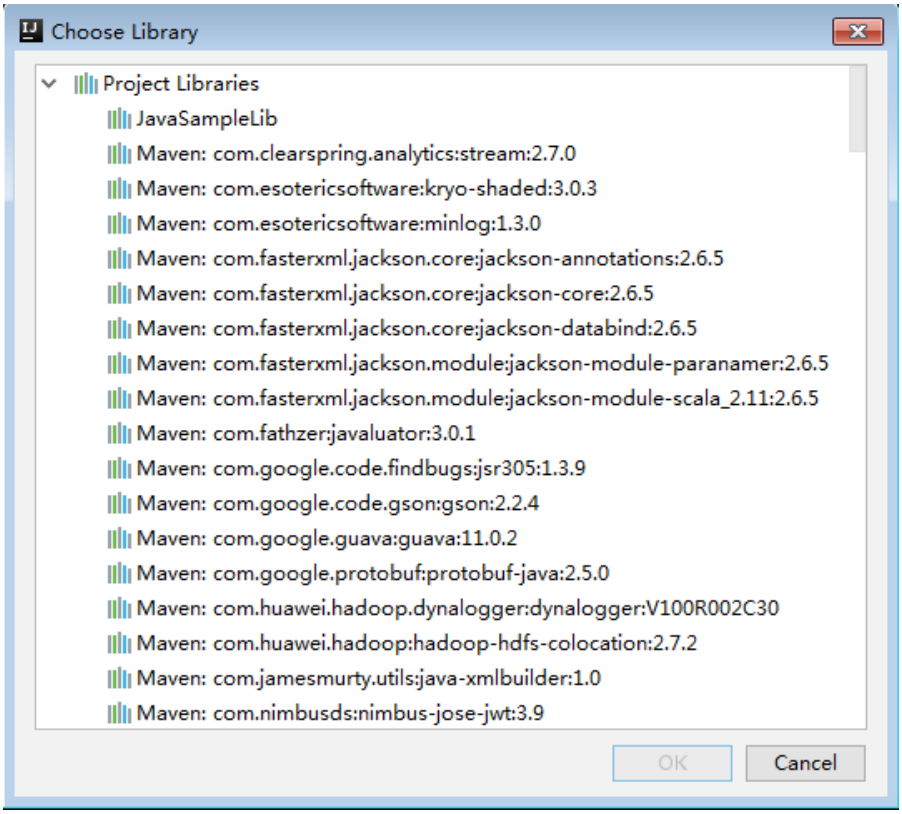
1. On the **Project Structure** page, select the **Artifacts** tab.
2. Click **+** and select **Library Files** in the right window to add the dependent library.

Figure 29-42 Add Library Files



3. Choose the dependent library to be added and click **OK**.

Figure 29-43 Choose Libraries



4. Click **Apply** to load the dependent library and click **OK** to complete the configuration.

----End

29.5.5.2 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 Stuctures...**.
2. Select **Libraries** and select the Jars packages that are automatically imported. Right-click the mouse and select **Delete**.

29.5.5.3 Why the "Class Does not Exist" Error Is Reported While the SparkStresmingKafka 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_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, 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 $SPARK_CLIENT_HOME/jars/streamingClient010/kafka-clients-2.4.0.jar,$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**.

29.5.5.4 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:

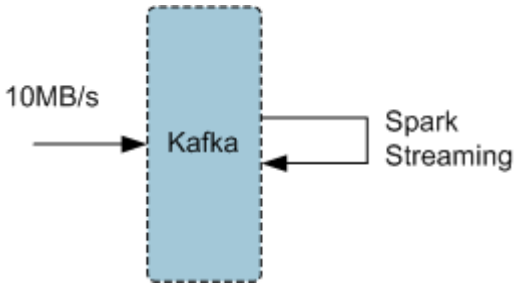
```
2016-03-02 17:46:19,017 | INFO | [kafka-network-thread-21005-1] | Closing socket connection to /10.91.8.208 due to invalid request: Request of length
 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. | kafka.network.Processor (Logging.scala:68)
2016-03-02 17:46:19,155 | 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 29-44 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.

29.5.5.5 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.newArray(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$$anon$3$$anonfun$run$1.apply$mcV
$sp(TaskResultGetter.scala:66)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.util.Utils$.logUncaughtExceptions(Utils.scala:1716)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3.run(TaskResultGetter.scala:56)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)
Exception in thread "task-result-getter-2" java.lang.OutOfMemoryError: Java heap space
at java.lang.reflect.Array.newArray(Native Method)
at java.lang.reflect.Array.newInstance(Array.java:75)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1671)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1707)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
at org.apache.spark.serializer.JavaDeserializationStream.readObject(JavaSerializer.scala:71)
at org.apache.spark.serializer.JavaSerializerInstance.deserialize(JavaSerializer.scala:91)
at org.apache.spark.scheduler.DirectTaskResult.value(TaskResult.scala:94)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$$anonfun$run$1.apply$mcV
$sp(TaskResultGetter.scala:66)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.util.Utils$.logUncaughtExceptions(Utils.scala:1716)
at org.apache.spark.scheduler.TaskResultGetter$$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.

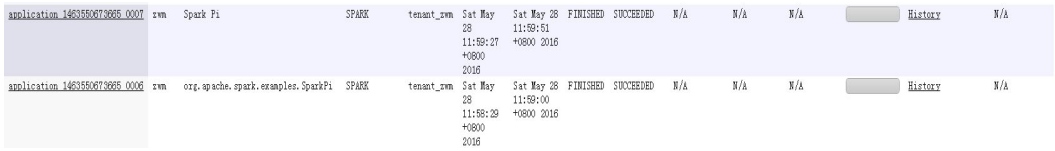
If you expect the Spark Core application to exit forcibly in the event of OOM error, add the following information to the configuration option **spark.driver.extraJavaOptions** in the Spark client configuration file **$SPARK_HOME/conf/spark-defaults.conf** when you start the Spark Core application for the first time:

```
-XX:OnOutOfMemoryError='kill -9 %p'
```


29.5.5.6 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_1463550673665_0007	zwn	Spark Pi	SPARK	tenant_zwn	Sat May 28 11:59:27 +0800 2016	Sat May 28 11:59:51 +0800 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A		History	N/A
application_1463550673665_0006	zwn	org.apache.spark.examples.SparkPi	SPARK	tenant_zwn	Sat May 28 11:58:29 +0800 2016	Sat May 28 11:59:00 +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*

29.5.5.7 How to Perform Remote Debugging Using IDEA?

Question

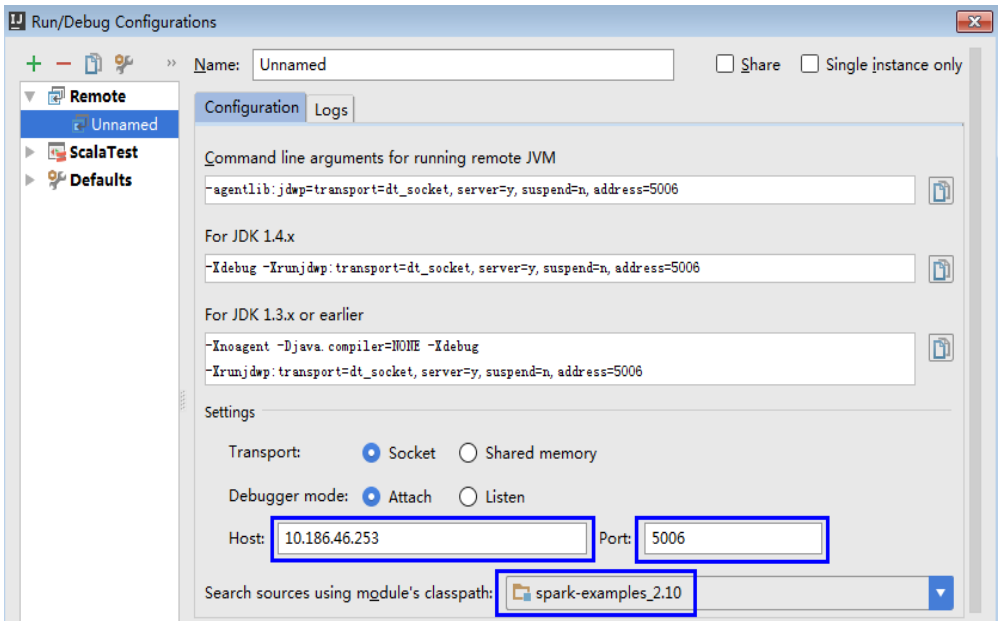
How to perform remote debugging using IDEA during Spark secondary development?

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. On the displayed window, click  at the upper left corner. Then on the drop-down menu, choose **Remote**, as shown in **Figure 29-45**.

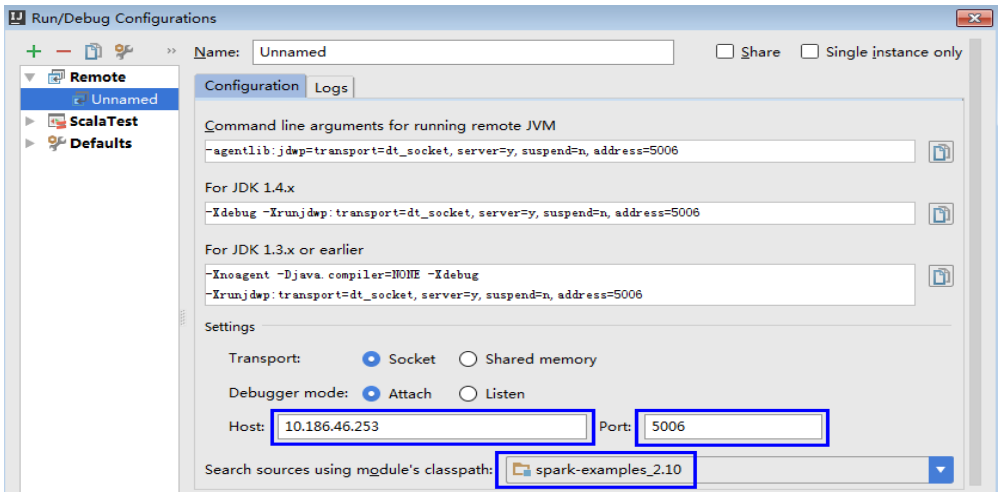
Figure 29-45 Choosing **Remote**



3. Configure the **Host**, **Port**, and **Search source using module's classpath**, as shown in **Figure 29-46**.

   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 29-46 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 -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5006*** to the **For JDK1.4.xdebugging** command obtained from **3**

Figure 29-47 Command for running Spark

```
[root@host2 bin]#./spark-submit --master yarn-client --driver-java-options "-Xde
bug -Xrunjdwp: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 29-48** illustrates how to select the breakpoint of the code in row 29 of SparkPi.scala.

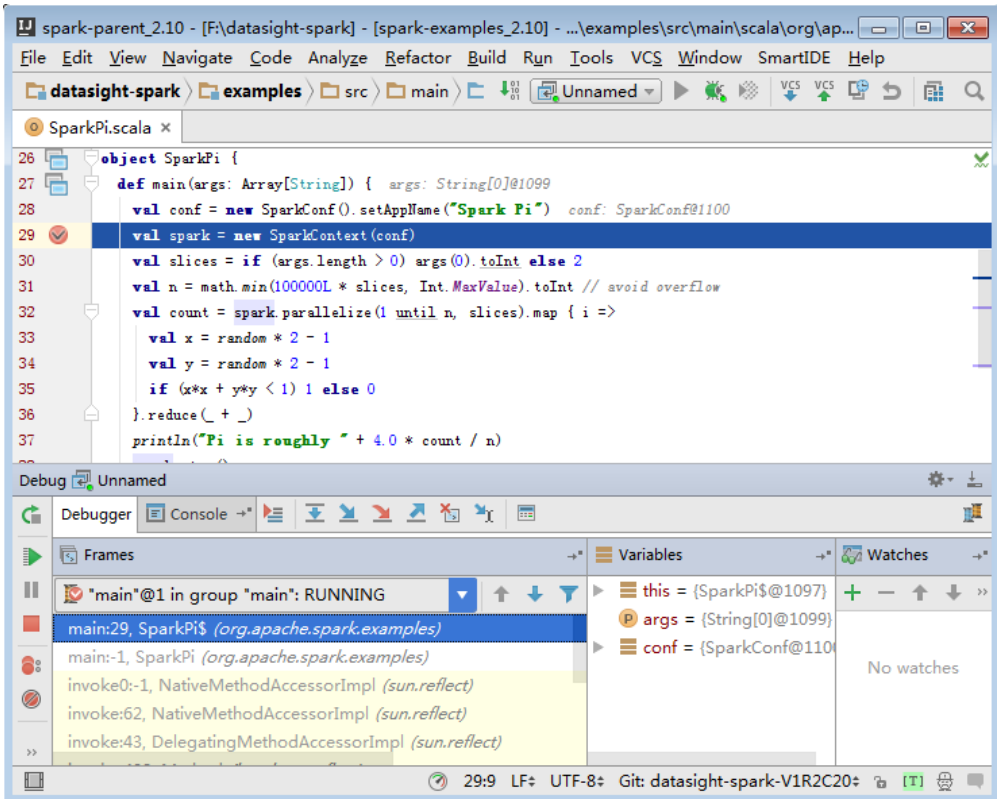
   Figure 29-48 Setting the breakpoint

```
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 29-49**.

Figure 29-49 Debugging



29.5.5.8 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.

Step 1 Define the org.apache.spark.launcher.SparkLauncher class. The SparkLauncherJavaExample and SparkLauncherScalaExample are provided by default as example code. You can modify the input parameters of example code 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 requirements and configure constant values such as the main class of the user-compiled Spark application. For details about different scenarios, see .

- 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 example 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 **Developing the Project**.
   - 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 th client. The dependency package of the example 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. Th 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 WebUI 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

29.5.5.9 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.

29.5.5.10 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.

29.5.5.11 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.

29.5.5.12 Support for Third-party JAR Packages on x86 and TaiShan Platforms

Question

The .jar package (for example, UDF package) written by users has two versions: x86 and TaiShan. How to enable Spark2x to support .jar package for both the x86 and TaiShan platforms?

Answer

Use the hybrid solution.

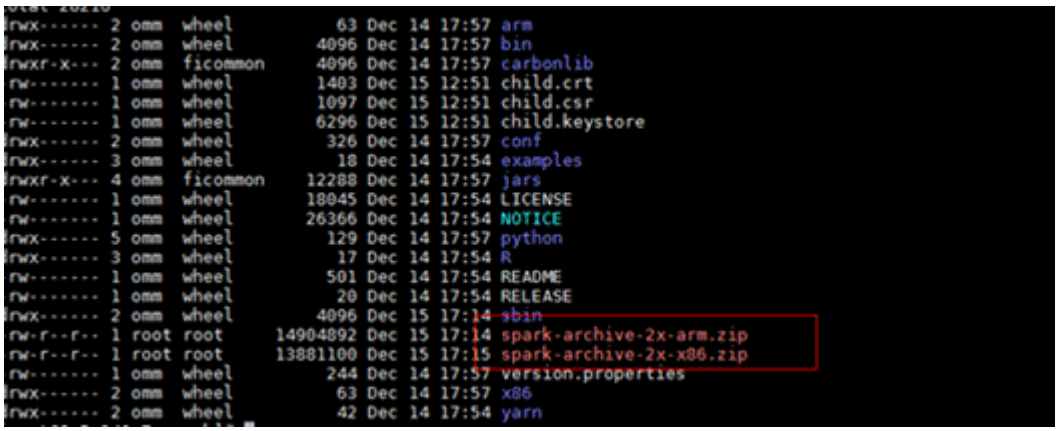
Step 1 Go to the installation directory of the Spark2x sparkResource on the server. The cluster may be installed on multiple nodes. You can go to any installation node and run the cd command to go to the installation directory of the sparkResource.

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/*



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 the **local spark-archive-2x-arm.zip** and **spark-archive-2x-x86.zip** files.

Step 6 Perform **Step 1** to **Step 2** for other sparkResource nodes.

Step 7 Log in to the webUI 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.

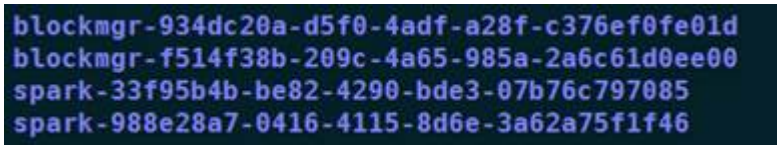
----End

29.5.5.13 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 29-50 Residual directory example



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.

29.5.5.14 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

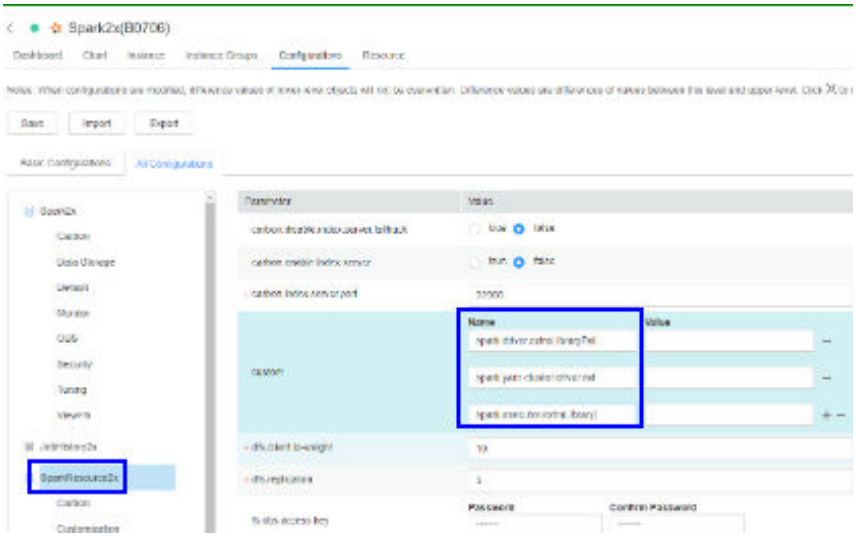
Solution 1:

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.

29.5.5.15 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.

29.5.5.16 Common JAR File Conflicts

Symptom

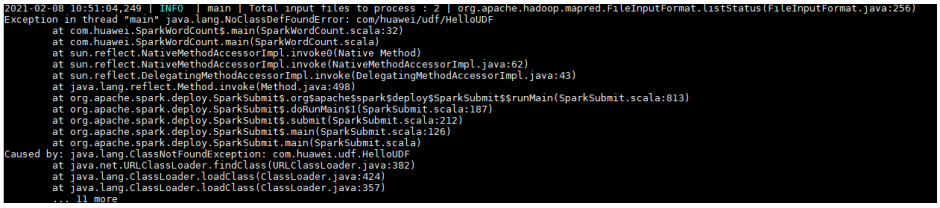
Spark can interconnect with many third-party tools. Therefore, it depends on a large number of third-party packages. Some packages are provided by MRS. As a result, the versions of the JAR files used by the code may be different from those of the JAR files provided by the cluster. In this case, JAR file 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**

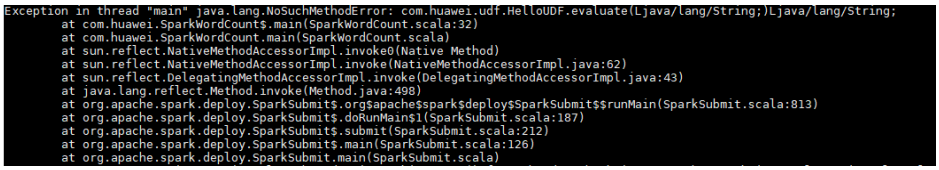
Cause Analysis

The following uses UDF customization as an example:



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.



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>
```


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.

30 YARN Development Guide (Security Mode)

30.1 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.

30.2 Interfaces

30.2.1 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 varables, and run the command. The command format is as follows.

yarn [--config *confdir***]** *COMMAND*

Table 30-1 describes the commands.

NOTE

The version 8.0.2.1 is used as an example. Replace it with the actual version number.

Table 30-1 Common commands

Command	Description
resourcemanager	Runs a ResourceManager. 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. • **export YARN_CONF_DIR=${BIGDATA_HOME}/FusionInsight_HD_8.1.2.2/1_10_ResourceManager/etc** • **export HADOOP_CONF_DIR=${BIGDATA_HOME}/FusionInsight_HD_8.1.2.2/1_10_ResourceManager/etc**
nodemanager	Runs a NodeManager. 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. • **export YARN_CONF_DIR=${BIGDATA_HOME}/FusionInsight_HD_8.1.2.2/1_10_NodeManager/etc** • **export export HADOOP_CONF_DIR=${BIGDATA_HOME}/FusionInsight_HD_8.1.2.2/1_10_NodeManager/etc**
rmadmin	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 *CLASSNAME*.
top	Runs the cluster usage monitor tool.
-Dmapreduce.job.hdfs-servers	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 *NAMESERVICE* with the HDFS NameService name. 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**.

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_ADMINGROUPS: 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
```

30.2.2 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 30-2 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 30-3 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 30-4 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.

30.2.3 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/"
     ```
     In the commands, 10-120-85-2 is the hostname of the main node. ResourceManager and 8090 is the port number of ResourceManager.
   - 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 30-5 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.

Information	Description
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 30-6 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.

Information	Description
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.

30.2.4 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
    - 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 30-7 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 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 30-8 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.

Attribute	Type	Description
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.
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.

Attribute	Type	Description
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/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 30-9 Parameters of all queues

Attribute	Type	Description
queuelist	array	Array of queue names.

Attribute	Type	Description
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.
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. 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 30-10 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.
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.

Attribute	Type	Description
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.
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 30-11 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	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.

Attribute	Type	Description
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 30-12 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.
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>
```

31 YARN Development Guide (Normal Mode)

31.1 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.

31.2 Interfaces

31.2.1 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 31-1 describes the commands.

NOTE

The version 8.1.2.2 is used as an example. Replace it with the actual version number.

Table 31-1 Common commands

Command	Description
resourcemanager	Runs a ResourceManager. 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. • **export YARN_CONF_DIR=${BIGDATA_HOME}/FusionInsight_HD_8.1.2.2/1_10_ResourceManager/etc** • **export HADOOP_CONF_DIR=${BIGDATA_HOME}/FusionInsight_HD_8.1.2.2/1_10_ResourceManager/etc**
nodemanager	Runs a NodeManager. 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. • **export YARN_CONF_DIR=${BIGDATA_HOME}/FusionInsight_HD_8.1.2.2/1_10_NodeManager/etc** • **export export HADOOP_CONF_DIR=${BIGDATA_HOME}/FusionInsight_HD_8.1.2.2/1_10_NodeManager/etc**
rmadmin	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 *CLASSNAME*.
top	Runs the cluster usage monitor tool.
-Dmapreduce.job.hdfs-servers	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 *NAMESERVICE* with the HDFS NameService name. 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**.

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 rigths.

  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_ADMINGROUPS: 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 rigths.

  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
```

31.2.2 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 31-2 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 31-3 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 31-4 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.

31.2.3 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/"
     ```

     In the commands, 10-120-85-2 is the hostname of the main node ResourceManager and 8088 is the port number of ResourceManager.
   - 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 31-5 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 31-6 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.

31.2.4 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
    - 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 31-7 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 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 31-8 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 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 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 31-9 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.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.

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. 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 31-10 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.

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 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 31-11 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	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 *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 31-12 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.

Attribute	Type	Description
resource_select	String	Resource select expression.

- Query **policiesxmlconf**
  - URL

    GET http://<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>
```


32 Manager Management Development Guide

32.1 Overview

32.1.1 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.

32.1.2 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: **http://**<*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, **http://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, **YWRtaW46QWRtaW5AMTIz** is generated. Then the basic authentication flag is added to obtain **Basic YWRtaW46QWRtaW5AMTIz**. 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.

32.1.3 Development Process

This document describes FusionInsight Manager application development based on the Java API.

Figure 32-1 shows and **Table 32-1** describes each phase of the development process.

Figure 32-1 FusionInsight Manager application development process

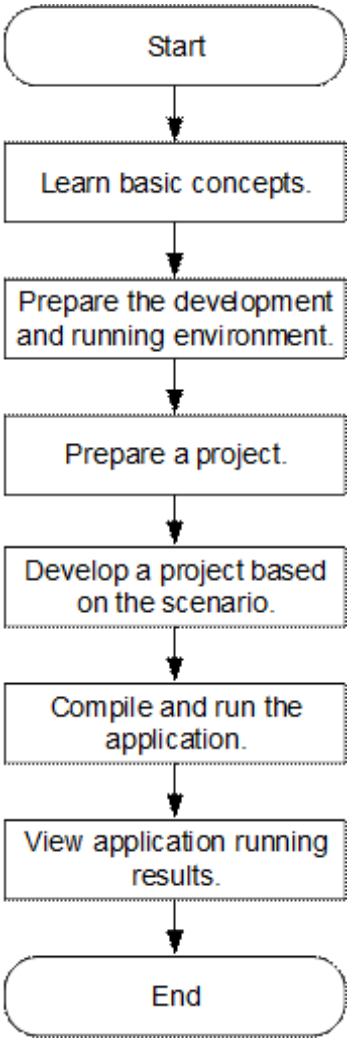


Table 32-1 Description of the FusionInsight Manager development process

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**

32.2 Environment Preparation

32.2.1 Preparing Development and Running Environments

Table 32-2 describes the development and running environment required for development.

Table 32-2 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	Basic configuration for the development and running environment. The JDK must meet the following requirements: 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. For example, JDK of versions later than 1.8.x must be used if FusionInsight Manager 8.1.2.2 is used.
IntelliJ IDEA installation and configuration	Tool used for developing HBase applications. The version must be **2019.1** or other compatible version.

32.2.2 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.

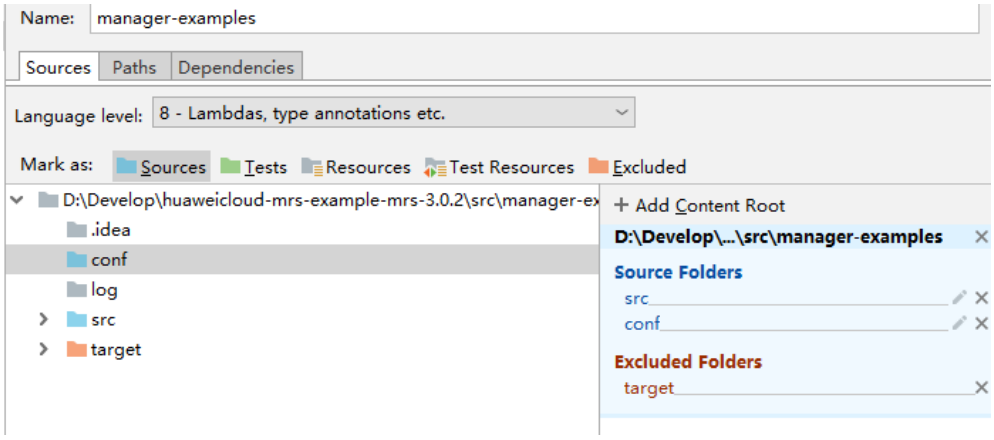
1. Start IntelliJ IDEA, and then choose **File** > **New** > **Project from Existing Sources**. In the **Select File or Directory to Import** dialog box, select the sample code folder.
2. Select the **pom.xml** file from the sample project folder, choose **Import project from external model** > **Maven**, 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.

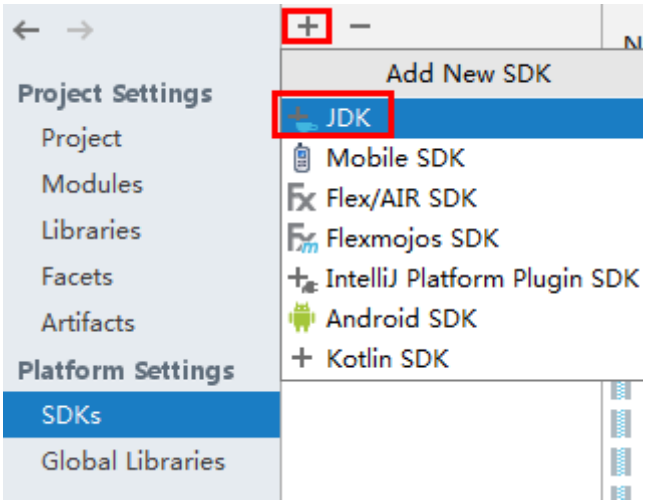
   After the project is imported, choose **File** > **Project Structure** on the menu bar of Intellij IDEA. In the displayed window, choose **Project Settings** > **Modules**.

   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**.

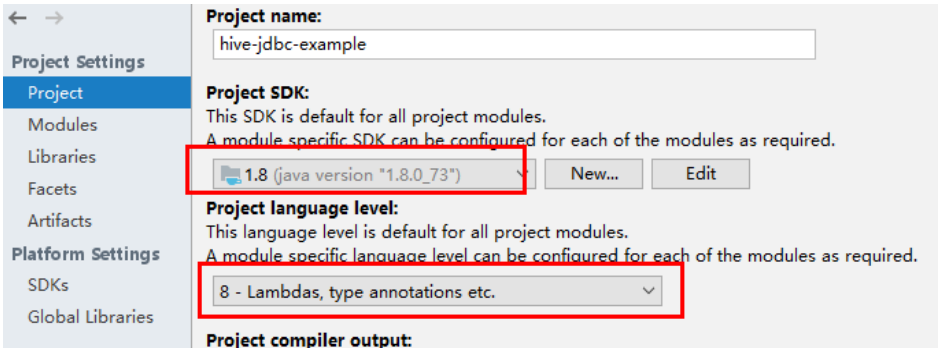


Step 3 Set the JDK of the project.

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**.

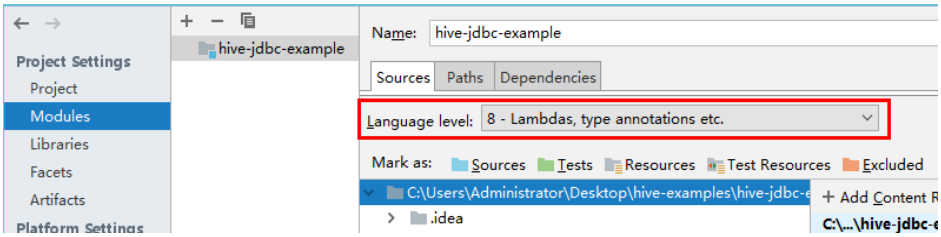


3. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
4. After selecting the JDK, click **Apply**.
5. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.

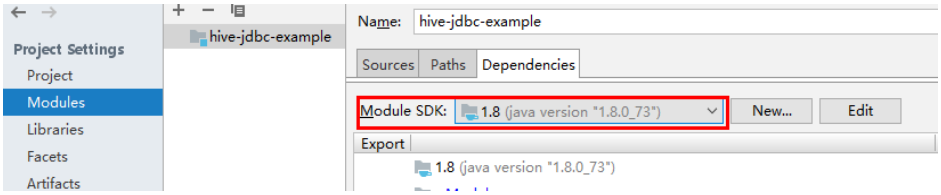


6. Click **Apply**.

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.

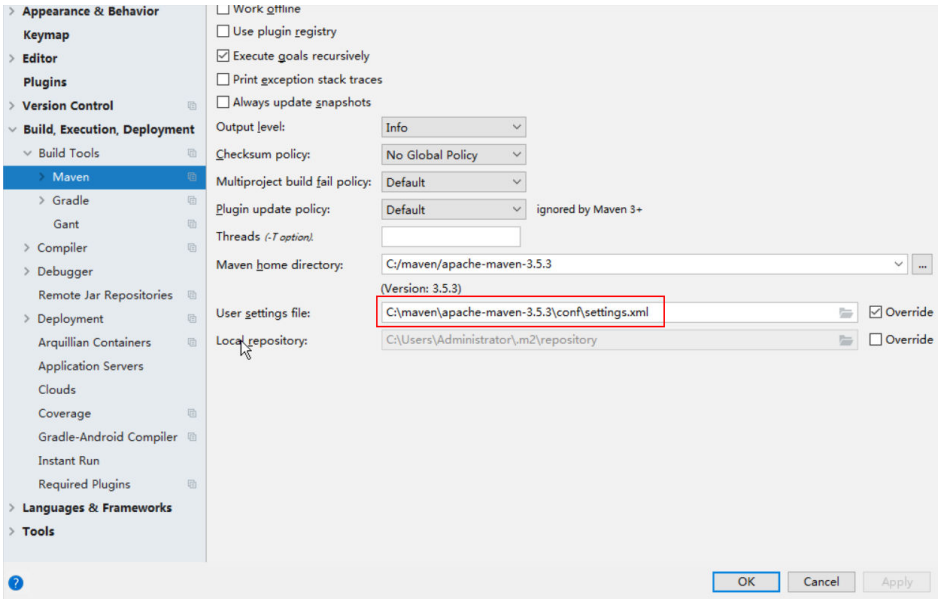


8. Click **Apply** and **OK**.

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**.

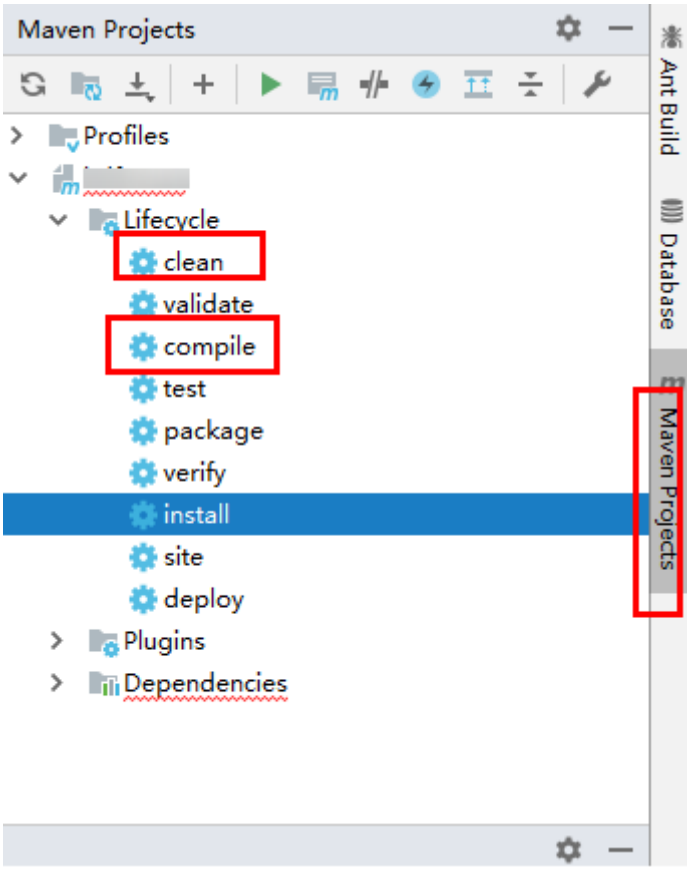
Figure 32-2 Directory for storing the **settings.xml** file



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 32-3 Maven Projects page



----End

32.3 Developing an Application

32.3.1 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.

32.3.2 Development Guideline

Function Decomposition

Decompose the function based on the scenario, as listed in **Table 32-3**.

Table 32-3 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**.

32.3.3 Example Code Description

32.3.3.1 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_Address:28443/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 UserManager class in the **rest** package.

```
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 = "";
```

32.3.3.2 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 UserManager class in the **rest** package.

```
//Access the FusionInsight Manager interface to add users.
operationName = "AddUser";
operationUrl = webUrl + ADD_USER_URL;
jsonFilePath = "./conf/addUser.json";
httpManager.sendHttpPostRequest(httpClient, operationUrl, jsonFilePath, operationName)
```

32.3.3.3 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 UserManager class in the **rest** package.

```
//Access the FusionInsight Manager interface to search the user list.
operationName = "QueryUserList";
operationUrl = webUrl + QUERY_USER_LIST_URL;
String responseLineContent = httpManager.sendHttpGetRequest(httpClient, operationUrl,
operationName);
LOG.info("The {} response is {}.", operationName, responseLineContent);
```

32.3.3.4 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 UserManager class in the **rest** package.

```
//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);
```

32.3.3.5 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 UserManager class in the **rest** package.

```
    //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.");
```

32.3.3.6 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 ExportUsers class in the **rest** package.

```
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);
```

32.4 Application Commissioning

32.4.1 Commissioning an Application in the Windows OS

32.4.1.1 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.

32.4.1.2 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 YWRtaW46QmlnZGF0YV8yMDEz .
basicAuth.BasicAuthAccess.constructAuthentication(BasicAuthAccess.java:122)
2020-10-19 14:22:52,695 INFO  [main] 3. Send first access request, usename 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_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"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","user
Source":"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_MANAGER_USER","iamCustomPolicyUser":false},
{"userName":"developuser","userType":"MM","description":"","password":"","createTime":"2020-10-15T19:16:37+08:00","defaultUser":false,"primaryGroup":"elasticsearch","locked":false,"userRoles":["System_administrator"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":"MRS_MANAGER_USER","iamCustomPolicyUser":false},
{"userName":"hue1","userType":"HM","description":"","password":"","createTime":"2020-10-09T17:39:57+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":"supergroup","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 sendHttpPutRequest for userOperation ModifyUser.
basicAuth.HttpManager.sendHttpPutRequest(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.sendHttpPutRequest(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] sendHttpPutRequest completely.
basicAuth.HttpManager.sendHttpPutRequest(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.sendHttpDeleteRequest(HttpManager.java:389)
2020-10-19 14:22:56,299 INFO [main] Enter sendHttpDeleteMessage for operation DeleteUser.
basicAuth.HttpManager.sendHttpDeleteRequest(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] sendHttpDeleteMessage for DeleteUser completely.
basicAuth.HttpManager.sendHttpDeleteRequest(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, sendHttpPostRequest, sendHttpGetRequest, sendHttpPutRequest, and sendHttpDeleteRequest 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, usename 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
```


32.5 More Information

32.5.1 External Interfaces

32.5.1.1 Java API

Typical APIs

Table 32-4 describes the typical methods of developing FusionInsight Manager REST APIs.

Table 32-4 restApiDemo.src.rest.BasicAuthAcces

Method	Description
loginAndAccess (String webUrl,String userName,String password,String userTLSVersion)	• 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 32-5 restApiDemo.src.rest.BasicAuthAcces.HttpManager

Method	Description
sendHttpGetRequest(HttpClient httpClient, String operationUrl, String operationName)	• Parameter type: HttpClient httpCient: 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)	• Parameter type: HttpClient httpCient: 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)	• Parameter type: HttpClient httpCient: 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)	• Parameter type: HttpClient httpCient: 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)	• Parameter type: HttpClient httpCient: 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)	• Parameter type: HttpClient httpCient: 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.

32.5.2 FAQ

32.5.2.1 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

32.5.2.2 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 32-4 Log indicating operation failure

```
2018-10-25 16:08:10,907 INFO  [main] 3. Send first access request, usename is admin. basicAuth.BasicAuthAccess.loginAndAcces
2018-10-25 16:08:11,720 INFO  [main] First access status is HTTP/1.1 401  basicAuth.BasicAuthAccess.firstAccessResp(BasicAutl
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.

32.5.2.3 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 32-5 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

32.5.2.4 An Operation Fails and "illegal character in path at index 57" Is Displayed in Logs

Question

An application execution fails. **Figure 32-6** shows the log information.

Figure 32-6 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.

32.5.2.5 Run the curl Command to Access REST APIs

Description

Users can run the **openssl version** command to view the OpenSSL version of the system. If the version is earlier than OpenSSL 1.0.1, install the OpenSSL of a later version for the operating system so that TLSv1.1 and TLSv1.2 can be used to interact with the cluster system.

Operation example and procedure

- Send the GET method to access the user list query interface and save **jsessionid** to the **jessionid.txt** file.

  curl -k -i --basic -u *<user name>*:*<password>* **-c** */tmp/jsessionid.txt* **'https://***x.x.x.x***:28443/web/api/v2/permission/users?limit=10&offset=0&filter=&order=ASC&order_by=userName'**

- Use **jsessionid** 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***","confirmPassword":"***xxx***","userGroups":["***supergroup***"],"userRoles":[],"primaryGroup":"***supergroup***","description":"***Add user***"}' 'https://***x.x.x.x***: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":"","confirmPassword":"","userGroups":["***supergroup***","***hadoopmanager***"],"primaryGroup":"***supergroup***","userRoles":[],"description":"***Modify user***"}' 'https://***x.x.x.x***: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://***x.x.x.x***:28443/web/api/v2/permission/users**

