MapReduce Service

Development Guide

Issue 05

Date 2024-08-16

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 Introduction to MRS Application Development

MRS is a unified platform for enterprise-level big data storage, query, and analysis. It helps enterprises quickly build massive data processing systems and discover new value points and business opportunities by analyzing and mining massive data.

MRS provides sample programs for common service scenarios of each component. Developers can develop and compile data applications based on the sample projects. The JAR packages on which the sample projects depend can be downloaded from the HUAWEI CLOUD open-source image site. The JAR packages of other communities can be downloaded from the Maven public repository.

Developer Capability Requirements

- You have a certain understanding of the components in the big data domain.
- You are familiar with how to use the ECS and the MRS development components.
- You have a certain understanding of the Maven build mode and usage methods.
- You already have some knowledge of the Java syntax.

MRS Application Development Process

The following figure shows the MRS component application development process. For details about the development and compilation operations of each component application, see the corresponding chapter in the component development guide.

Figure 1-1 MRS Application Development Process

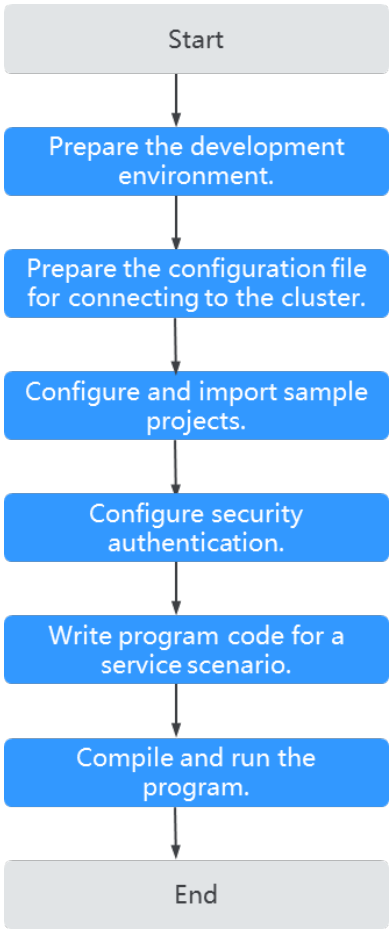


Table 1-1 MRS Application Development Process

stage	Description
Preparing the Development Environment	Before developing applications, prepare the development environment. The IntelliJ IDEA tool is recommended. In addition, you need to configure the JDK and Maven locally.
Preparing the Configuration File for Connecting to the Cluster	During application development or running, you need to connect to the MRS cluster through the cluster configuration file. The configuration file contains 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.
Configuring and Importing a Sample Project	MRS provides multiple sample programs for different component scenarios. You can obtain sample projects and import them to the local development environment for program learning.

stage	Description
Configuring Security Authentication	When connecting to the MRS cluster with Kerberos authentication enabled, configure the user with the related resource access permission in the application for security authentication.
Develop programs based on business scenarios.	Develop programs based on the actual service scenario and invoke component interfaces to implement corresponding functions.
Compiling and Running the Program	Compile and run the developed 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 the Linux node for running.

2 Obtaining the MRS Application Development Sample Project

Process for Building an MRS Sample Project

The MRS sample project construction process consists of the following steps:

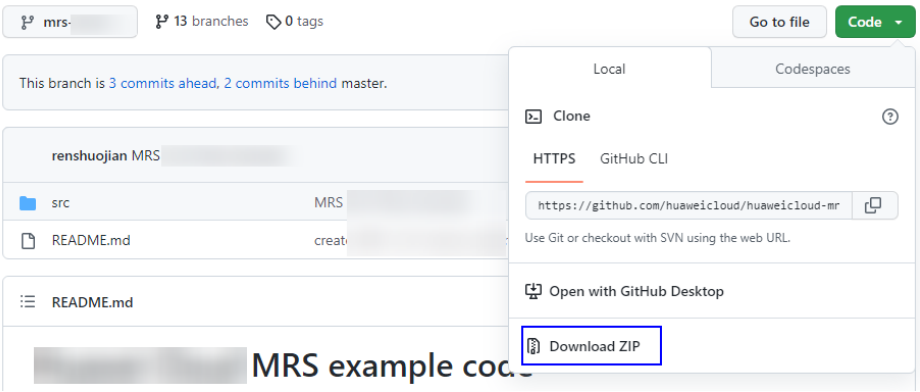
1. Download the Maven project source code and configuration file of the sample project. For details, see.**Obtaining the MRS Sample Project**.
2. For details about how to configure the Maven image repository of the SDK in Huawei image site, see.**Configuring Huawei Open Source Image Repository**.
3. This section describes how to build a complete Maven project and compile and develop it based on user requirements.

Obtaining the MRS Sample Project

You can download the MRS sample project from**https://github.com/huaweicloud/huaweicloud-mrs-example**.

Switch the branch to the version branch matching the MRS cluster. , for example, mrs-3.2.0.1, download the package to the local host and decompress it to obtain the sample code project corresponding to each component.

Figure 2-1 Downloading MRS Sample Project Code



You can download the sample project corresponding to the MRS LTS version from the following website:

- MRS 3.2.0-LTS.1: **https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-3.2.0.1**
- MRS 3.1.2-LTS.3: **https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-3.1.2**

You can download the sample project of the common MRS version from the following website:

- MRS 3.0.2: **https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-3.0.2**
- MRS 3.1.0: **https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-3.1.0**
- MRS 3.1.5: **https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-3.1.5**
- MRS 2.1.x: **https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-2.1**
- MRS 1.9.x: **https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-1.9**
- MRS 1.8.x: **https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-1.8**

Configuring Huawei Open Source Image Repository

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

NOTE

Before using the development tool to download the dependent JAR package in the local environment, ensure that the following information is available:

- The local network is normal.

  Uses a browser and visit Huawei Mirrors to check whether the website can be accessed. If the access is abnormal, connect the local network.

- The proxy is disabled for the development tool.

  Take the IntelliJ IDEA development tool of version 2020.2 as an example. Choose **File** > **Settings** > **Appearance & Behavior** > **System Settings** > **HTTP Proxy**, select **No proxy**, and click **OK** to save the configuration.

The configuration method of an open source image is as follows:

Step 1 Ensure that 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.

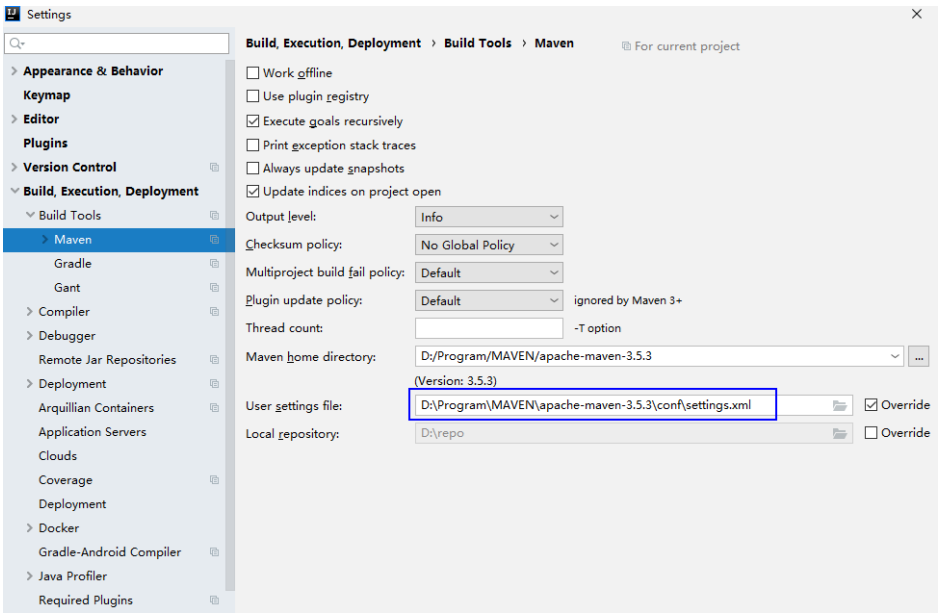
```
<profile>
   <id>huaweicloudsdk</id>
   <repositories>
      <repository>
         <id>huaweicloudsdk</id>
         <url>https://repo.huaweicloud.com/repository/maven/huaweicloudsdk/</url>
         <releases><enabled>true</enabled></releases>
         <snapshots><enabled>true</enabled></snapshots>
      </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 encoding and JDK. Add the following information to the profiles node 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

Sample projects for MRS components

The MRS sample code library provides sample projects for basic functions of each component. For details about the sample projects provided by each component in the current version, see.**Table 2-1**.

Table 2-1 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.

Component	Sample Project Location		Description
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.
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	

Component	Sample Project Location		Description
		FlinkSqlJavaExample	SQL job submission through Jar jobs on the client
		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.

3 Sample Projects of MRS Components

You need to obtain the sample projects from **Obtaining the MRS Application Development Sample Project**, **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 (2.*x*)

Component	Sample Project Location	Description
Alluxio	alluxio-examples	Use Alluxio to connect the storage system sample program through a public interface. The example writes and reads files.

Component	Sample Project Location	Description
Flink	flink-examples	The following sample programs are provided: ● DataStream program Java/Scale program for constructing DataStream with Flink This project analyzes user log data based on service requirements, reads text data, generates DataStreams, filters data that meets specified conditions, and obtains results. ● Program that produces and consumes data in Kafka 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. ● Asynchronous checkpointing Java/Scala program for Flink asynchronous checkpointing. In this project, the program uses custom operator 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. ● Stream SQL join Flink streaming 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.
HBase	hbase-examples	HBase data read and write This program calls HBase APIs to create user tables, import user data, add and query user information, and create secondary indexes for user tables.

Component	Sample Project Location	Description
HDFS	hdfs-examples	Java program for HDFS file operations. This program creates HDFS folders, writs files, appends file content, reads files, and deletes files or folders.
Hive	hive-examples	The following JDBC/HCatalog sample programs are provided: ● Java program for Hive JDBC to process data In this project, JDBC APIs are used to connect Hive and perform data operations. JDBC APIs are called to create tables, load data, and query data. ● Java program for Hive HCatalog to process data HCatalog APIs are used to define and query MRS Hive metadata with Hive CLI.
Impala	impala-examples	Java program for Impala JDBC to process data In this project, the JDBC APIs are called to connect Impala and perform data operations in Impala. JDBC APIs are called to create tables, load data, and query data.
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.
MapReduce	mapreduce-examples	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.

Component	Sample Project Location		Description
Presto	presto-examples		The following JDBC/HCatalog sample programs are provided: ● Java program for Presto JDBC to process data In this project, the JDBC APIs are called to connect Presto and perform data operations in Presto. JDBC APIs are called to create tables, load data, and query data. ● Java program for Presto HCatalog to process data
OpenTSDB	opentsdb-examples		OpenTSDB APIs are called to collect monitoring information in a large-scale cluster and query data in seconds. This program can write, query, and delete data.
Spark	spark-examples	SparkHbasetoHbaseJavaExample	Java/Scala 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.
		SparkHbasetoHbaseScalaExample	
		SparkHivetoHbaseJavaExample	Java/Scala 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.
		SparkHivetoHbaseScalaExample	
		SparkJavaExample	Java/Python/Scala program of Spark Core tasks The program reads text data from HDFS and then calculates and analyzes the data.
		SparkPythonExample	
		SparkScalaExample	
		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 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 the HBase, and the read data is filtered.
		SparkOnHbaseScalaExample	
		SparkSQLJavaExample	Java/Scala program of Spark SQL tasks The program reads text data from HDFS and then calculates and analyzes the data.
		SparkSQLScalaExample	
		SparkStreamingJavaExample	Java/Scala program used by Spark Streaming to receive data from Kafka and perform statistical analysis This program analyzes user log data based on service requirements, reads text data, generates DataStreams, filters data that meets specified conditions, and obtains results.
		SparkStreamingScalaExample	
		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	
		SparkStreamingtoHbaseJavaExample	Java/Scala 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.
		SparkStreamingtoHbaseScalaExample	
		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.
		SparkStructuredStreamingScalaExample	

Component	Sample Project Location		Description
		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	
Storm	storm-examples	storm-common-examples	Constructor of Storm topologies and Spout/Bolt The program can create Spout, Bolt, and Topology.
		storm-hbase-examples	Interaction between Storm and HBase of MRS The program submits the Storm topology and stores the data to the WordCount table of HBase.
		storm-hdfs-examples	Interaction between Storm and HDFS of MRS The program submits the Storm topology and stores the data to HDFS.
		storm-jdbc-examples	Accessing MRS Storm with JDBC The program uses Storm topology to insert data into a table.
		storm-kafka-examples	Interaction between Storm and Kafka of MRS The program uses the Storm topology to send data to Kafka and display the data.
		storm-obs-examples	Interaction between Storm and OBS of MRS The program submits the Storm topology and stores the data to OBS.

4 Alluxio Development Guide

4.1 Alluxio Application Development Overview

4.1.1 Introduction to Alluxio Application Development

Introduction to Alluxio

Alluxio is an open source data orchestration technology for analytics and AI for the cloud. It bridges data-driven applications and storage systems by moving data from the storage layer closer to data-driven applications, making it easier and faster to access. In addition, applications can be connected to multiple storage systems through a common interface.

Alluxio provides the following features:

- Provides in-memory I/O throughput, and makes data-driven applications that can be elastically scaled cost effective.
- Simplified cloud and object storage access
- Simplified data management and a single point of access to multiple data sources
- Easy application deployment

Introduction to Alluxio Interface Development

Alluxio supports program development using Java. For details about APIs, see **https://docs.alluxio.io/os/javadoc/2.0/index.html**.

4.1.2 Common Concepts of Alluxio

Masters

A JournalNode consists of two processes: Alluxio Master that processes user requests and manages metadata of the Journal storage system and Alluxio Job Master that schedules file system operations.

Workers

Manages local resources (such as memory, SDD, and HDD) that can be configured by users and performs data operations on underlying storage.

Client

The Alluxio Client supports Java API, Shell, and HTTP REST API.

- Java API

  Provides an application interface for the Alluxio. This guide describes how to use the Java API to develop Alluxio client.
- Shell

  Provides shell commands to perform operations on the Alluxio.
- HTTP REST API

  Provides other APIs except Shell and Java APIs. You can use these APIs to query information. For details, see **Alluxio APIs**.

Namespace

Transparent naming mechanism: ensures that the namespaces of Alluxio and underlying storage systems are the same.

Unified namespace: Alluxio provides a mounting API, which can be used to access data in multiple data sources in Alluxio.

4.1.3 Alluxio Application Development Process

Figure 4-1 and **Table 4-1** describe the phases in the development process.

Figure 4-1 Alluxio application development process

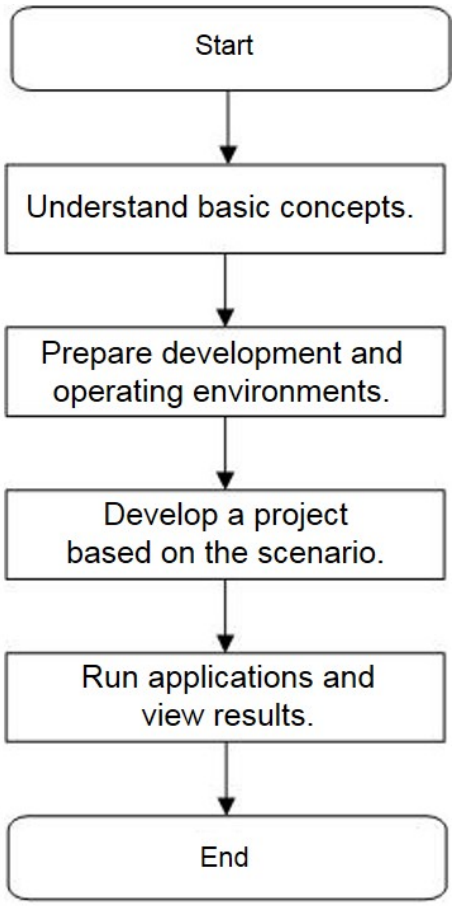


Table 4-1 Description of the Alluxio application development process

Phase	Description	Reference
Understand basic concepts.	Before application development, learn basic concepts of Alluxio.	**Common Concepts of Alluxio**
Prepare development and operating environments.	The Java language is recommended for the development of Alluxio client applications, and Maven is recommended for constructing projects. The running environment of the sample application consists of nodes of the VPC cluster where the MRS service is deployed.	**Alluxio Development Environment**

Phase	Description	Reference
Develop a project based on the scenario.	Presto provides a Java sample project and a sample project of data query.	**Alluxio Development Plan**
Run applications and view results.	This phase provides guidance for users to submit a developed application for running and view the result.	**Commissioning an Alluxio Application**

4.2 Preparing an Alluxio Application Development Environment

4.2.1 Alluxio Development Environment

Table 4-2 describes the local environment required for application development. You also need to prepare a Linux environment for verifying whether the application is running properly.

Table 4-2 Development environment

Item	Description
OS	Development environment: Windows 7 or later version is recommended. Operating environment: Linux system
Installation of JDK and Maven	Basic configuration of the development environment: Java JDK 8 or later, Maven 3.3.9 or later
Installation and configuration of Eclipse or IntelliJ IDEA	It is a tool used to develop Alluxio applications.
Network	The client must be interconnected with the Alluxio server on the network.

4.2.2 Preparing an Alluxio Application Development Environment

- Install Eclipse and JDK in the Windows development environment.

  The recommended JDK version is 1.8, and the Eclipse version is 4.3.2 or later.

NOTE

- If you use IBM JDK, ensure that the JDK configured in Eclipse is IBM JDK.
- If you use Oracle JDK, ensure that the JDK configured in Eclipse is Oracle JDK.
- If you use ODBC for secondary development, ensure that JDK 1.8 or later is used.
- Do not use the same workspace and the sample project in the same path for different Eclipse programs.

- Prepare a Linux environment for testing application running status.

Preparing a Running and Commissioning Environment

Step 1 On the ECS management console, apply for a new ECS for user application development, running, and commissioning.

- Select EulerOS as the ECS OS and the required version.
- The security group of the ECS must be the same as that of the Master node in an MRS cluster.
- The ECS and the MRS cluster must be in the same VPC.
- The ECS NIC and the MRS cluster must be in the same network segment.

Step 2 On the EIP page, apply for an EIP and bind it to the ECS. For details, see **Assigning an EIP and Binding It to an ECS**.

Step 3 Configure an inbound or outbound rule for the security group. For details, see **Configuring Security Group Rules**.

Step 4 Download a client program.

1. Log in to **MRS Manager**.
2. Choose **Services** > **Download Client** to download the complete client to the remote host, that is, download the client program to the newly applied ECS.

Step 5 Log in to the node where the downloaded client is located, and then install the client as user **root**.

1. Run the following command to decompress the client package:

   cd /opt

   tar -xvf /opt/MRS_Services_Client.tar

2. Run the following command to verify the installation file package:

   sha256sum -c /opt/MRS_Services_ClientConfig.tar.sha256

   MRS_Services_ClientConfig.tar:OK

3. Run the following command to decompress the installation file package:

   tar -xvf MRS_Services_ClientConfig.tar

4. Run the following command to install the client to a specified directory (absolute path), for example, **/opt/client**. The directory is automatically created.

   cd /opt/MRS_Services_ClientConfig

   sh install.sh /opt/client

   Components client installation is complete.

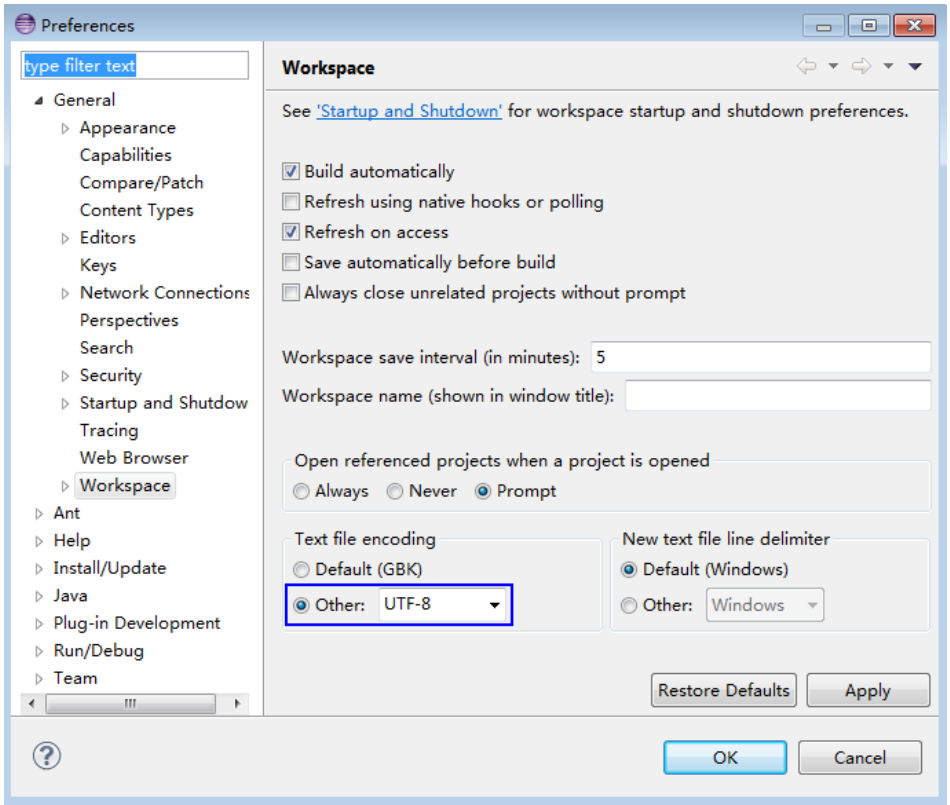
----End

4.2.3 Importing and Configuring Alluxio Sample Projects

1. Download the sample project from **Obtaining the MRS Sample Project** to the local PC.
2. Import the sample project to the Eclipse development environment.
   a. Start Eclipse and choose **File** > **Import**. In the **Import** dialog box, select **Existing Maven Projects**, and click **next**.
   b. Click **Browse** in the **Import Maven Projects** window. The **Select Root Folder** dialog box is displayed.
   c. Select the **alluxio-examples** sample project folder, and click **OK**.
   d. Click **Finish** in the **Import Maven Projects** window.
3. Set an Eclipse text file encoding format to prevent garbled characters.
   a. On the Eclipse menu bar, choose **Window** > **Preferences**.

      The **Preferences** window is displayed.
   b. In the navigation tree, choose **General** > **Workspace**. In the **Text file encoding** area, select **Other** and set the value to **UTF-8**. Click **Apply** and then **OK**. **Figure 4-2** shows the settings.

Figure 4-2 Setting the Eclipse encoding format



4.3 Developing an Alluxio Application

4.3.1 Alluxio Development Plan

Scenario Description

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

Service operation objects of Alluxio are files. File operations covered by sample codes include creating a folder, reading a file, and writing data to a file. You can learn how to perform other operations on the Alluxio, such as setting file access permissions, based on sample codes.

Sample codes are described in the following sequence:

1. Initializing the file system
2. Writing data to a file
3. Reading a file

Development guidelines

1. Invoke the **create** API in FileSystem to obtain the file system client.
2. Invoke the **createFile** API in FileSystem to create a file.
3. Invoke the **write** API in FileOutStream to write a file.
4. Invoke the **openFile** API in FileSystem to create a file.
5. Invoke the **in** API in fileSystem to read the file.

4.3.2 Initializing Alluxio

Function Description

Before using APIs provided by Alluxio, you need to initialize Alluxio. The process is as follows:

1. Load the HDFS service configuration file.
2. Instantiate Filesystem.
3. Use HDFS APIs.

Sample Code

The following provides code snippets. For complete codes, see the **ExampleClient** class.

```
/**
* load configurations from alluxio-site.properties
* @throws IOException
*/
private void loadConf() throws IOException {
    InputStream fileInputStream = null;
    alluxioConf = new Properties();
    File propertiesFile = new File(PATH_TO_ALLUXIO_SITE_PROPERTIES);
    try {
        fileInputStream = new FileInputStream(propertiesFile);
        alluxioConf.load(fileInputStream);
    }
    catch (FileNotFoundException e) {
```



```
            System.out.println(PATH_TO_ALLUXIO_SITE_PROPERTIES + "does not exist. Exception: " + e);
        }
        catch (IOException e) {
            System.out.println("Failed to load configuration file. Exception: " + e);
        }
        finally{
            close(fileInputStream);
        }
    }

    /**
    * build Alluxio instance
    */
    private void instanceBuild() throws IOException {
    // get filesystem
        InstancedConfiguration conf = new InstancedConfiguration(ConfigurationUtils.defaults());
        conf.set(PropertyKey.MASTER_RPC_ADDRESSES, alluxioConf.get("alluxio.master.rpc.addresses"));
        FileSystemContext fsContext = FileSystemContext.create(conf);
        fSystem = FileSystem.Factory.create(fsContext);
    }
```

4.3.3 Writing Data to an Alluxio File

Function Description

The process of writing data to a file is as follows:

1. Instantiate a FileSystem.
2. Use the FileSystem instance to obtain various types of resources for writing data to files.
3. Write the data to a specified file in Alluxio.

Sample Code

```
/**
* create file,write file
*/
private void write() throws IOException {
    final String content = "hi, I am bigdata. It is successful if you can see me.";
    FileOutStream out = null;
    try {
        AlluxioURI path = new AlluxioURI(testFilePath);
        out = fSystem.createFile(path);
        out.write(content.getBytes());
    }
    catch (Exception e){
        System.out.println("Failed to write file. Exception:" + e);
    }
    finally {
        close(out);
    }
}
```

4.3.4 Reading an Alluxio File

Function Description

Read data from a specified file in Alluxio.

Sample Code

```
/**
* read file
```



```
* @throws java.io.IOException
*/
private void read() throws IOException {
    AlluxioURI path = new AlluxioURI(testFilePath);
    FileInStream in = null;
    try{
        in = fSystem.openFile(path);
        byte[] buffer = new byte[1024];
        int len;
        String content = "";
        while((len = in.read(buffer)) != -1){
            String bufferStr = new String(buffer,0, len);
            content += bufferStr;
        }
        System.out.println(content);
    }
    catch (Exception e){
        System.out.println("Failed to read file. Exception:" + e);
    }
    finally {
        close(in);
    }
}
```

4.4 Commissioning an Alluxio Application

Running the Alluxio Client and Viewing Results

Step 1 Run the **mvn clean compile assembly:single** command to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **alluxio-examples-mrs-1.9-jar-with-dependencies.jar**.

Step 2 Create a directory as the running directory in the running and commissioning environment, for example, **/opt/alluxio_examples** (Linux), and create the **conf** subdirectory in the directory.

Copy **alluxio-examples-mrs-1.9-jar-with-dependencies.jar** exported in **Step 1** to **/opt/alluxio_examples**.

Copy the configuration file **/opt/client/Alluxio/alluxio/conf/alluxio-site.properties** from the client to the **conf** directory.

NOTE

When the Alluxio cluster is started, each Alluxio server process (including the master and worker processes) attempts to read the **alluxio-site.properties** file from the **${CLASSPATH}**, **${HOME}/.alluxio/**, **/etc/alluxio/**, and **${ALLUXIO_HOME}/conf** directories in sequence. When the **alluxio-site.properties** file is read in a directory, the remaining directories are skipped. Therefore, store the **alluxio-site.properties** file in an appropriate directory based on the site requirements.

Step 3 In Linux, run the sample program.

```
chmod +x /opt/alluxio_examples -R
cd /opt/alluxio_examples
java -jar alluxio-examples-mrs-1.9-jar-with-dependencies.jar /testFlie.txt
```

Step 4 In the CLI, view the query results of the sample code.

If the following information is displayed, the sample project execution is successful in Linux.


```
hi, I am bigdata. It is successful if you can see me.
```

----End

4.5 Alluxio APIs

Java API

Alluxio APIs comply with the Alluxio Parent API standard. For details, see **https://docs.alluxio.io/os/javadoc/2.0/index.html**.

HTTP REST API

- Master REST API: **https://docs.alluxio.io/os/restdoc/2.0/master/index.html**
- Worker REST API: **https://docs.alluxio.io/os/restdoc/2.0/worker/index.html**
- Proxy REST API: **https://docs.alluxio.io/os/restdoc/2.0/proxy/index.html**
- Job REST API: **https://docs.alluxio.io/os/restdoc/2.0/job/index.html**

5 Flink Development Guide

5.1 Flink Application Development Overview

5.1.1 Introduction to Flink Application Development

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

The entire Flink system consists of three parts:

- Client

  Flink client is used to submit jobs (streaming jobs) to Flink.

- TaskManager

  TaskManager is a service execution node of Flink. It executes specific tasks. A Flink system can have multiple TaskManagers. These TaskManagers are equivalent to each other.

- JobManager

  JobManager 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 is selected as the active JobManager, 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

- HA

  Active/standby JobManagers, preventing single point of failure (SPOF)
- Scale-out

  Manual scale-out supported by TaskManagers

Flink DataStream APIs can be developed in Scala and Java, as shown in **Table 5-1**.

Table 5-1 Flink DataStream APIs

Function	Description
Scala API	API in Scala, which can be used for data processing, such as filtering, joining, windowing, and aggregation. Since Scala is easy to read, you are advised to use Scala APIs to develop applications.
Java API	API in Java, which can be used for data processing, such as filtering, joining, windowing, and aggregation.

For details about Flink, visit **https://flink.apache.org/**.

5.1.2 Common Concepts of Flink Application Development

- **DataStream**

  A DataStream is the minimum data unit processed by 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.
- **Topology**

  A topology represents an execution task of a user. A topology is composed of the input (for example, Kafka source), output (for example, Kafka sink), and data transformations.
- **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 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.

5.1.3 Flink Application Development Process

Figure 5-1 and Table 5-2 describe the phases in the development process.

Figure 5-1 Flink application development process

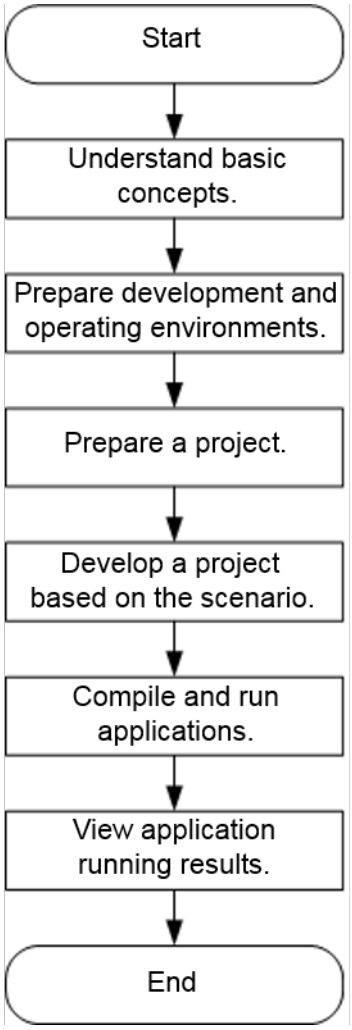


Table 5-2 Description of the Flink application development process

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

Phase	Description	Reference
Prepare development and operating environments.	Flink applications can be developed in Scala or Java. You are advised to use the IDEA tool to configure development environments in different languages according to the guide. The Flink operating environment is a Flink client. Install and configure the client according to the guide.	**Preparing a Local Application Development Environment**
Prepare a project.	Flink provides sample projects. You can import a sample project to learn the application. You can also create a Flink project according to the guide.	**Configuring and Importing a Flink Sample Project**
Develop a project based on the scenario.	Sample projects in Scala and Java are provided to help you quickly understand APIs of Flink components.	**Flink DataStream Development Plan**
Compile and run an application.	You can compile the developed application and submit it for running.	**Compiling and Running a Flink Application**
View application running results.	Application running results are stored in a path specified by you. You can also view application running status on the UI.	**Viewing the Running Result of a Flink Application**
Tune the application.	Tune the application to meet certain service requirements. After the application tuning is complete, the application needs to be compiled and run again.	**Flink Performance Tuning Suggestions**

5.2 Preparing a Flink Application Development Environment

5.2.1 Preparing a Local Application Development Environment

Table 5-3 describes the environment required for application development. You also need to prepare a Linux environment for verifying whether the application is running properly.

Table 5-3 Development environment

Item	Description
OS	• Development environment: Windows OS • Operating environment: Linux system
JDK installation	Basic configurations of the development and operating environments. The version requirements are as follows: The server and client of an MRS cluster support only built-in Oracle JDK 1.8, which cannot be replaced. If users' applications need to reference the JAR files of the SDK class in the user application processes, Oracle JDK and IBM JDK are supported. • Oracle JDK versions: 1.7 and 1.8 • IBM JDK versions: 1.7.8.10, 1.7.9.40, and 1.8.3.0 **NOTE** • If JDK 1.7 is used as the development environment, the running environment of Flink clusters can be JDK 1.7 or JDK 1.8. • If JDK 1.8 is used as the development environment, the running environment of Flink clusters must be JDK 1.8. If JDK 1.7 is used as the running environment, error of incorrect JDK version is reported.
IDEA installation and configuration	Tool used for developing Flink applications. The required version is 14.1.7.
Scala installation	Basic configuration for the Scala development environment. The required version is 2.11.12.
Scala plugin installation	Basic configuration for the Scala development environment. The required version is 1.5.4.
Preparing a development user	For details, see **Preparing a Flink Application Development User**.
Installing a client	For details, see **Installing the Flink Client**.

5.2.2 Preparing a Flink Application Development User

The development user is used to run the sample project. In a security cluster, only users with the permissions on HDFS, YARN, Kafka, and Flink are allowed to run Flink sample projects.

Prerequisites

Kerberos authentication has been enabled for the MRS cluster. Skip this step if Kerberos authentication is not enabled for the cluster.

Procedure

Step 1 Log in to MRS Manager and choose **System** > **Manage Role** > **Create Role**.

1. Enter a role name, for example, *flinkrole*.
2. In **Permission**, choose **HDFS** > **File System** > **hdfs://hacluster/** and select **Read**, **Write**, and **Execute**. After you finish configuring this service, click **Service** in the **Permission** area.
3. In **Permission**, choose **Yarn** > **Scheduler Queue** > **root**. Select **Submit** for **default**, and click **OK**.

   NOTE

   After you submit applications, WARN logs are printed on the client based on your configuration about the preceding role. The WARN log is generated because Flink obtains the remaining resource value from YARN for detection and evaluation. However, the operation requires the admin permission, which is not granted to you. Ignore the WARN log because it does not affect the job submission. Content of the WARN log is as follows:

   ```
   Get node resource from yarn cluster. Yarn cluster occur exception:
   org.apache.hadoop.yarn.exceptions.YarnPermissionDeniedException: User flinkuser does not have privilage to see, admin only
   ```

Step 2 On MRS Manager, choose **System** > **Manage User Group** > **Create User Group** to create a user group for the sample project, for example, **flinkgroup**.

Step 3 On MRS Manager, choose **System** > **Manage User** > **Create User** to create a user for the sample project. Enter a username, for example, **flinkuser**. Set **User Type** to **Human-machine**, and select **flinkgroup** and **hadoop** in **User Group**. Select **flinkrole** in **Assign Rights by Role**, and click **OK**.

NOTE

- You can use this user only after changing the password of user **flinkuser** on the client.
- 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, user **flinkuser** is added to the **kafkaadmin** user group.
- If a user wants to run a sample project (in Scala or Java) of an **application of producing and consuming data in Kafka**, the user needs to be added to the **kafkaadmin** group.

Step 4 On MRS Manager, choose **System** > **Manage User** and select **flinkuser**. Download an authentication credential file, save the file and decompress it to obtain the **keytab** and **krb5.conf** files, and copy the **krb5.conf** file to the **/etc** directory of the client. They are used for security authentication in the sample project. For

details how to use them, see **Preparing the Flink Application Security Authentication**.

----End

5.2.3 Installing the Flink Client

Flink uses a Windows environment for development. It is recommended that the running environment be deployed on a Linux OS because the MRS client cannot be installed in a Windows environment. Perform the following operations to configure the client.

Procedure

Step 1 Install the Flink client.

1. Ensure that the Flink component has been installed on a server.
2. Download a Flink client program.
   a. Log in to MRS Manager.
   b. Choose **Service** > **Flink** > **Download Client**, set **Client Type** to **All client files**, set **Download to** to **Server**, and click **OK** to download the client to the server.
3. Run the following commands to decompress the **MRS_Flink_Client.tar** client installation package:

   tar -xvf /tmp/MRS-client/MRS_Flink_Client.tar

   tar -xvf /tmp/MRS-client/MRS_Flink_ClientConfig.tar
4. Go to the directory (**/tmp/MRS-client/MRS_Flink_ClientConfig**) where the client installation package is decompressed, run the **./install.sh***#{client_install_home}* command to install the client.

   Example: **./install.sh /opt/flinkclient**

   NOTE

   If Kerberos authentication is enabled for the cluster and you need to use the client on a node outside the cluster, add the IP address of the node where the client is located to the **jobmanager.web.allow-access-address** configuration item in the Flink configuration file **flink-conf.yaml** of the client. If Kerberos authentication is not enabled for the cluster, you do not need to modify this configuration item.

Step 2 Configure network connections for the client.

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.

1. Confirm that the client can communicate with each host.
2. Add the mapping between the server host name and IP address to the **hosts** file on the client.
3. If the yarn-client mode is used, add the mapping between the client host name and IP address to the **hosts** file on the ResourceManager node of YARN.

NOTE

The file path is **/etc/hosts** on Linux and **C:\Windows\System32\drivers\etc\hosts** on Windows.

4. Verify the consistency of time between the client and the cluster. Ensure that the difference between the client time and the Flink cluster time is less than 5 minutes.
5. Verify that the configuration items in the Flink client configuration file **flink-conf.yaml** are correctly configured.

----End

5.2.4 Configuring and Importing a Flink Sample Project

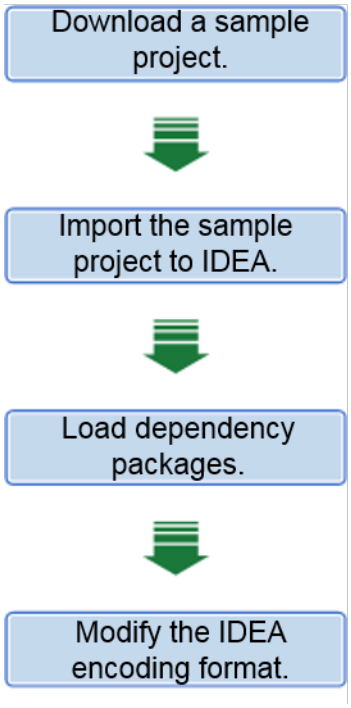
Scenario

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

Methods to import Java and Scala projects are the same.

The following example describes how to import Java sample code. **Figure 5-2** shows the operation process.

Figure 5-2 Procedure of importing a sample project



Procedure

Step 1 Download the sample project to the local computer by referring to **Obtaining the MRS Application Development Sample Project**.

Step 2 Copy the client installation package downloaded in **Installing the Flink Client** to the Windows server.

Step 3 Decompress **MRS_Flink_Client.tar** on the Windows server to obtain the **MRS_Flink_ClientConfig.tar**. Decompress the **MRS_Flink_ClientConfig.tar** to obtain the **MRS_Flink_ClientConfig** folder.

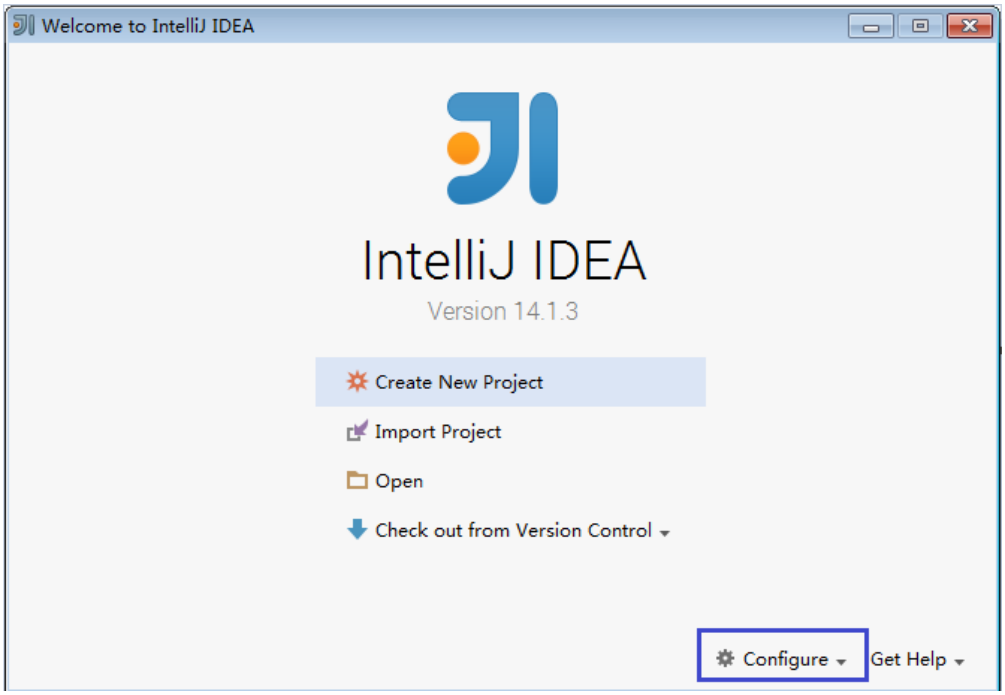
Step 4 Double-click the **flink_install.bat** script in the **MRS_Flink_ClientConfig/Flink** directory. After the installation is complete, the **lib** and **examples** folders are generated.

- The **lib** folder contains only the JAR files on which Flink depends. Find JAR files on which Kafka depends in the installation directory of the Kafka component on the server and add them.
- The **examples** folder contains the open source sample JAR file.

Step 5 Before importing the sample project, configure JDK for IntelliJ IDEA.

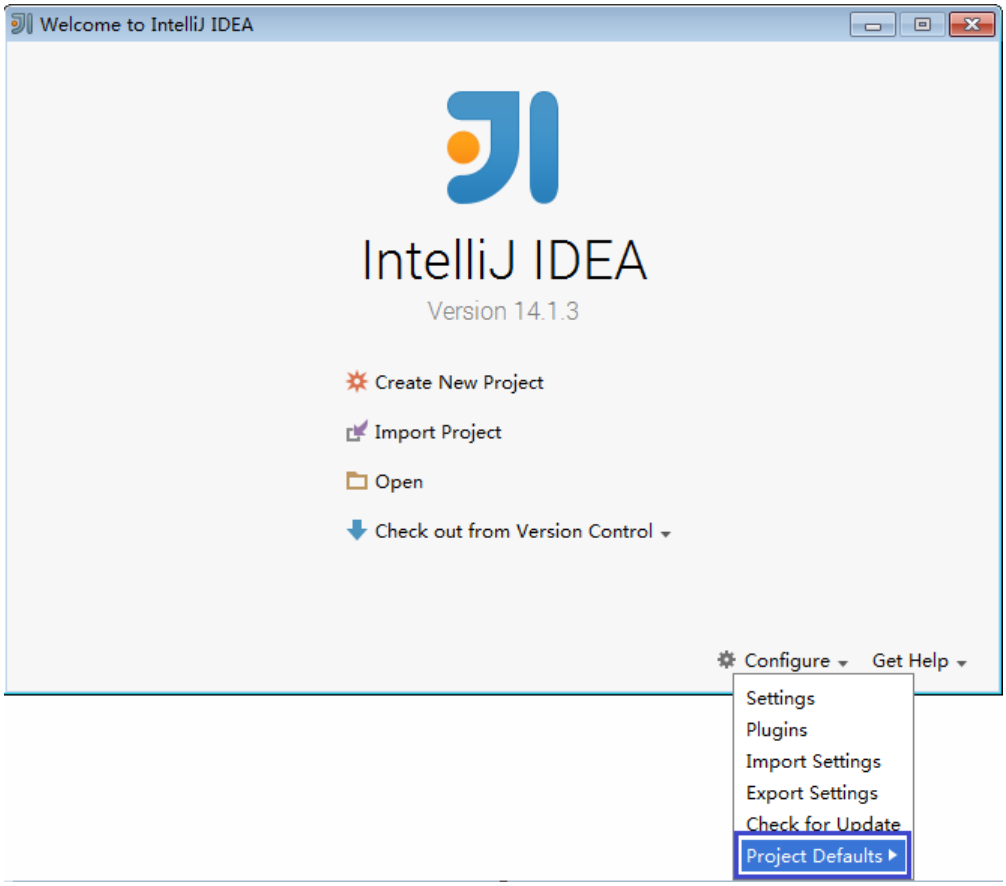
1. Start IntelliJ IDEA and click **Configure**.

   Figure 5-3 Clicking Configure

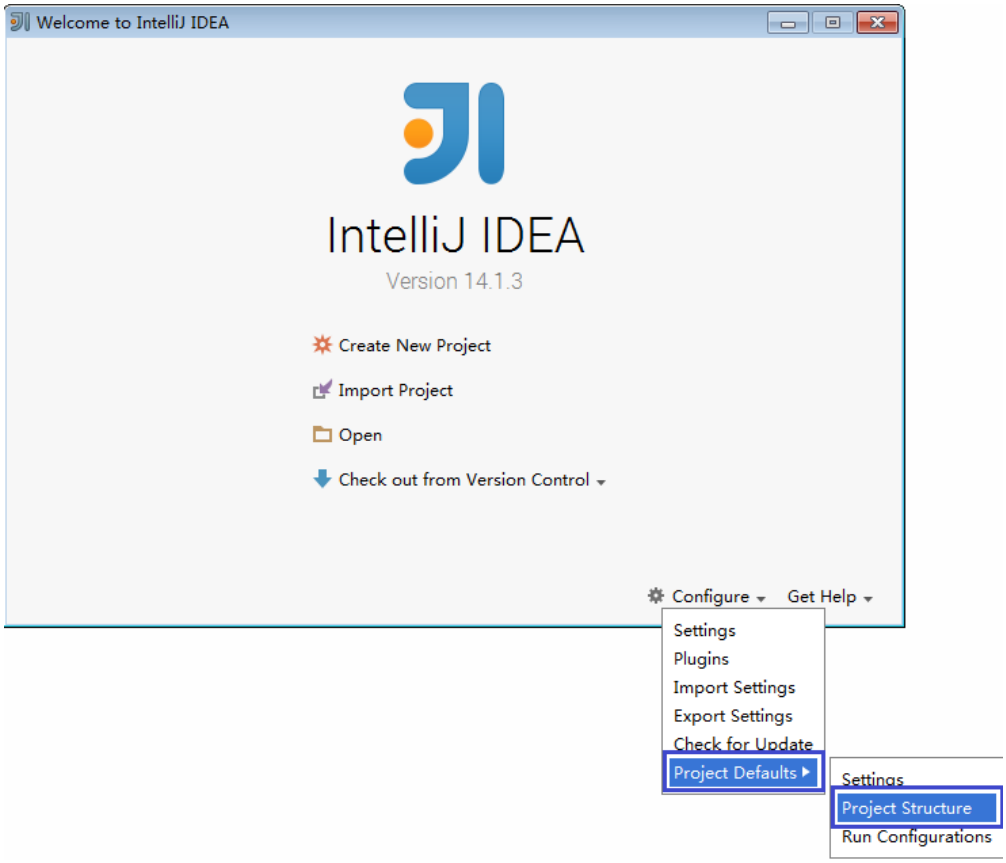
2. Choose **Project Defaults** from the **Configure** drop-down list.

Figure 5-4 Choosing Project Defaults



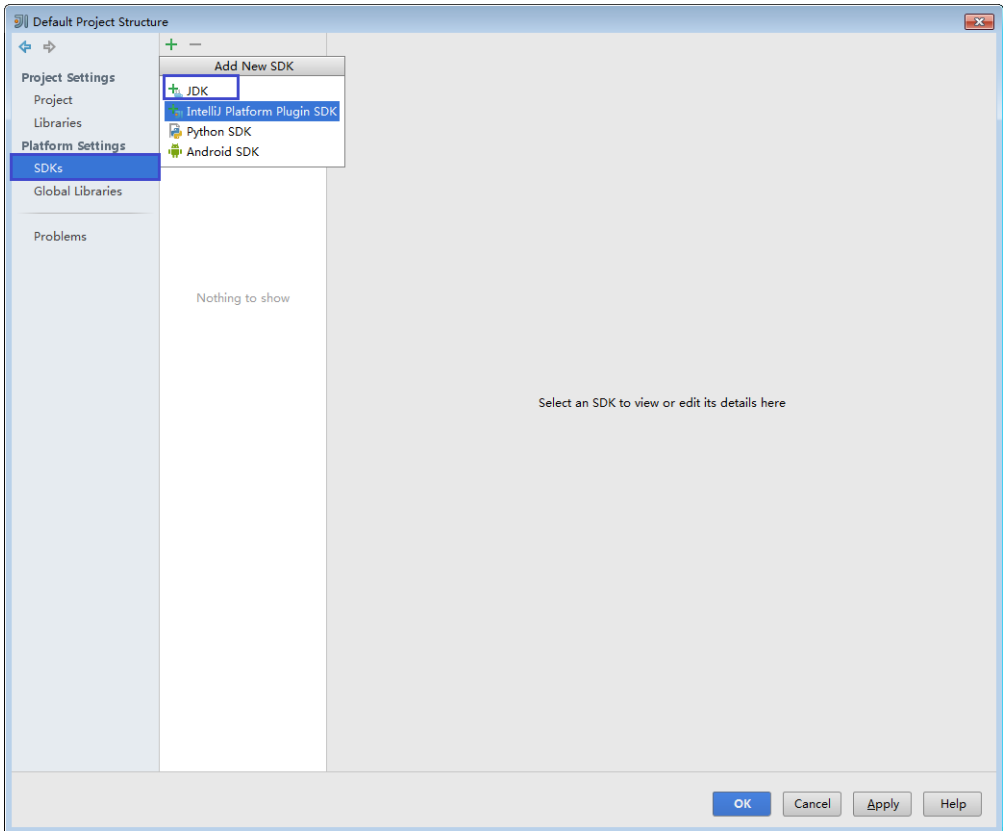
3. Choose **Project Structure** on the **Project Defaults** page.

Figure 5-5 Project Defaults



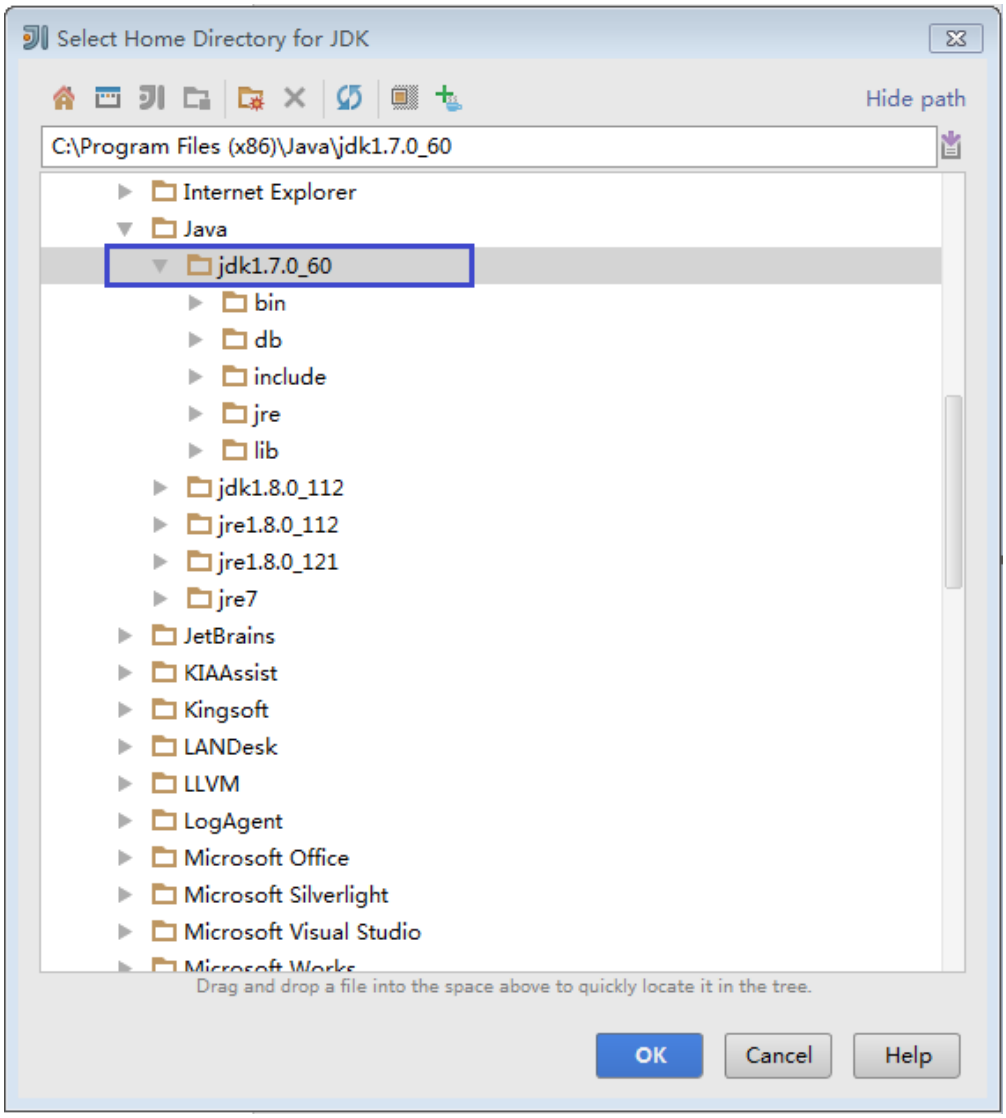
4. On the **Project Structure** page, select **SDKs** and click the green plus sign to add the JDK.

Figure 5-6 Adding the JDK



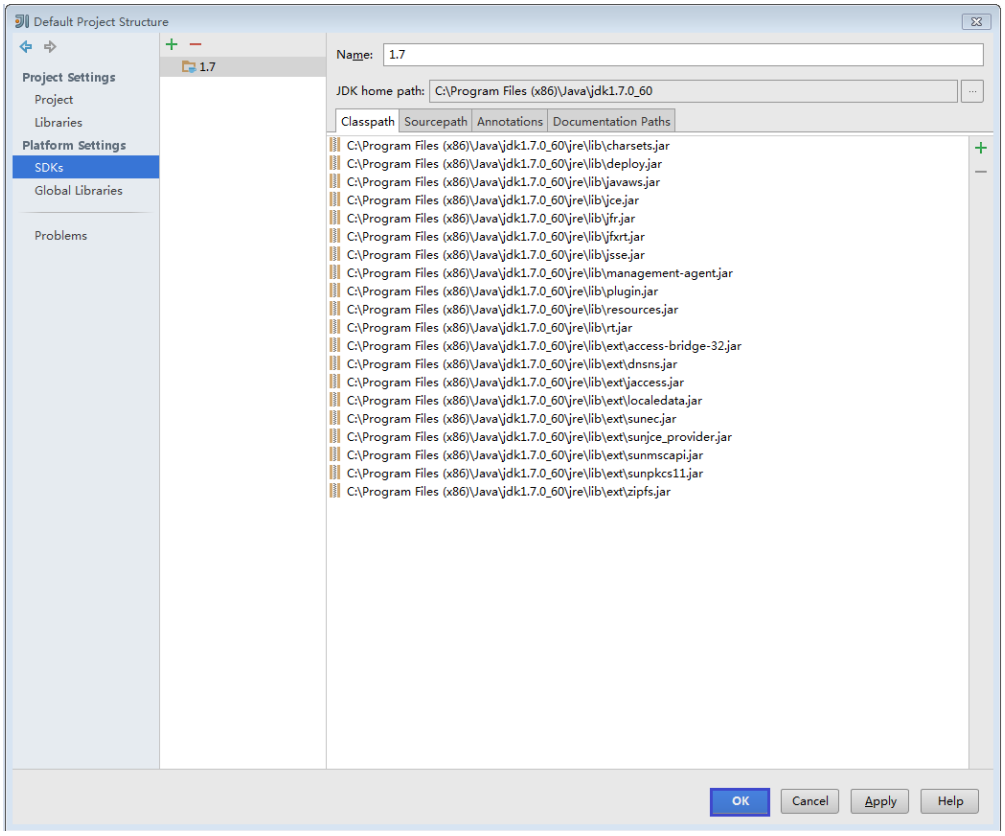
5. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 5-7 Selecting the JDK directory



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

Figure 5-8 Completing the JDK configuration

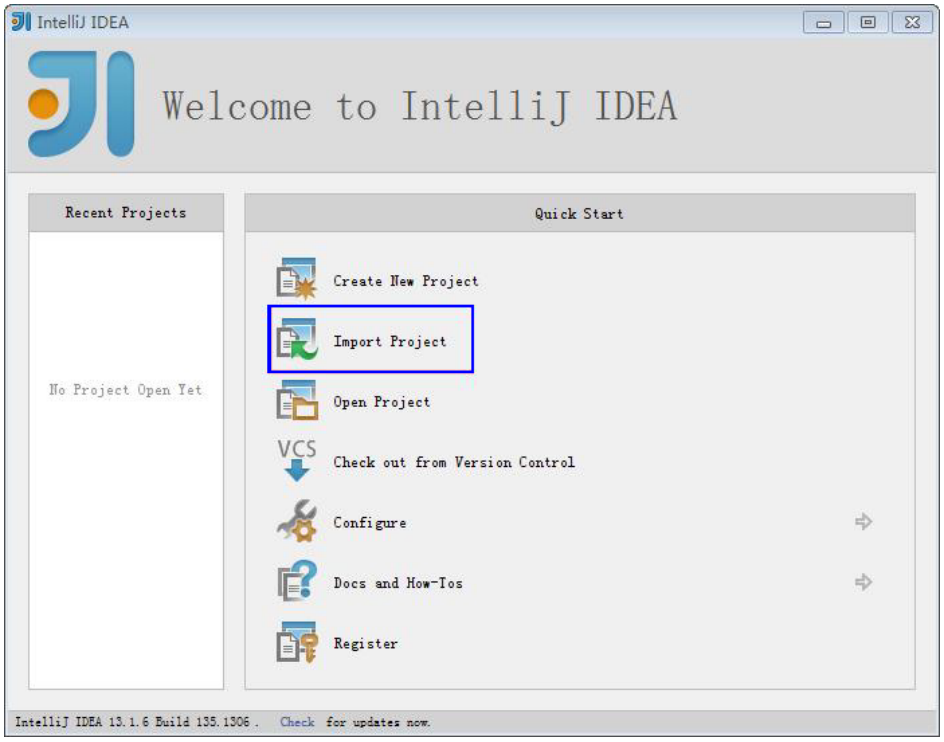


Step 6 Import the Java sample project to IDEA.

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

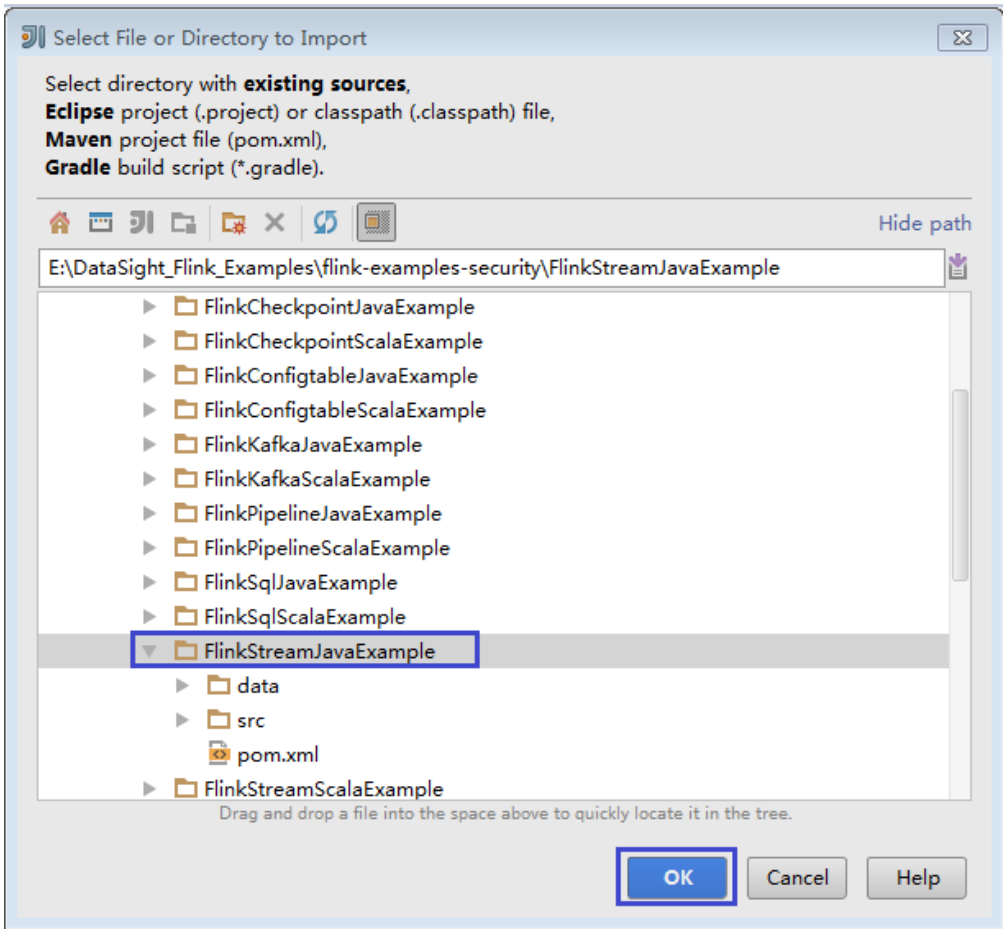
   Alternatively, for the used IDEA tool, add the project directly from the IDEA home page. Choose **File** > **Import project...** to import a project.

Figure 5-9 Importing the project (on the **Quick Start** page)



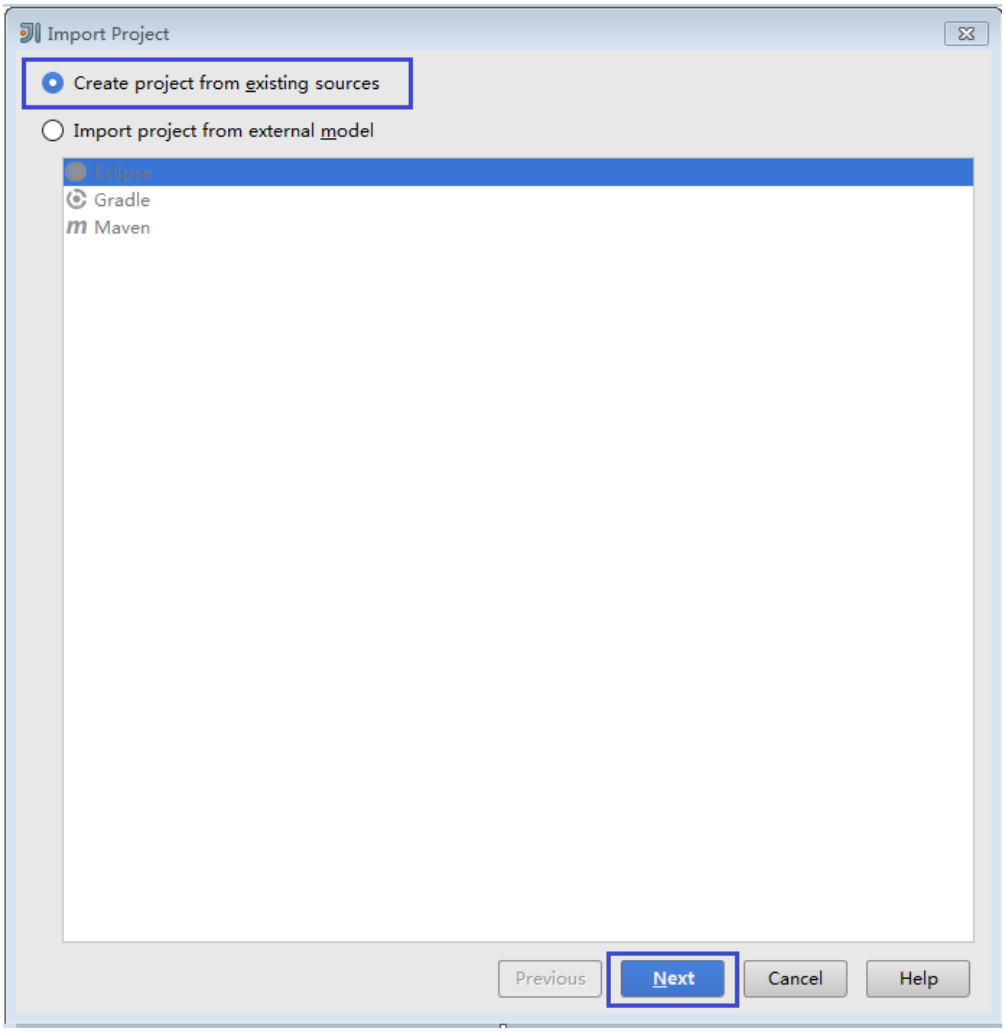
2. Select a path for storing the sample project to be imported and click **OK**.

Figure 5-10 Select File or Directory to Import



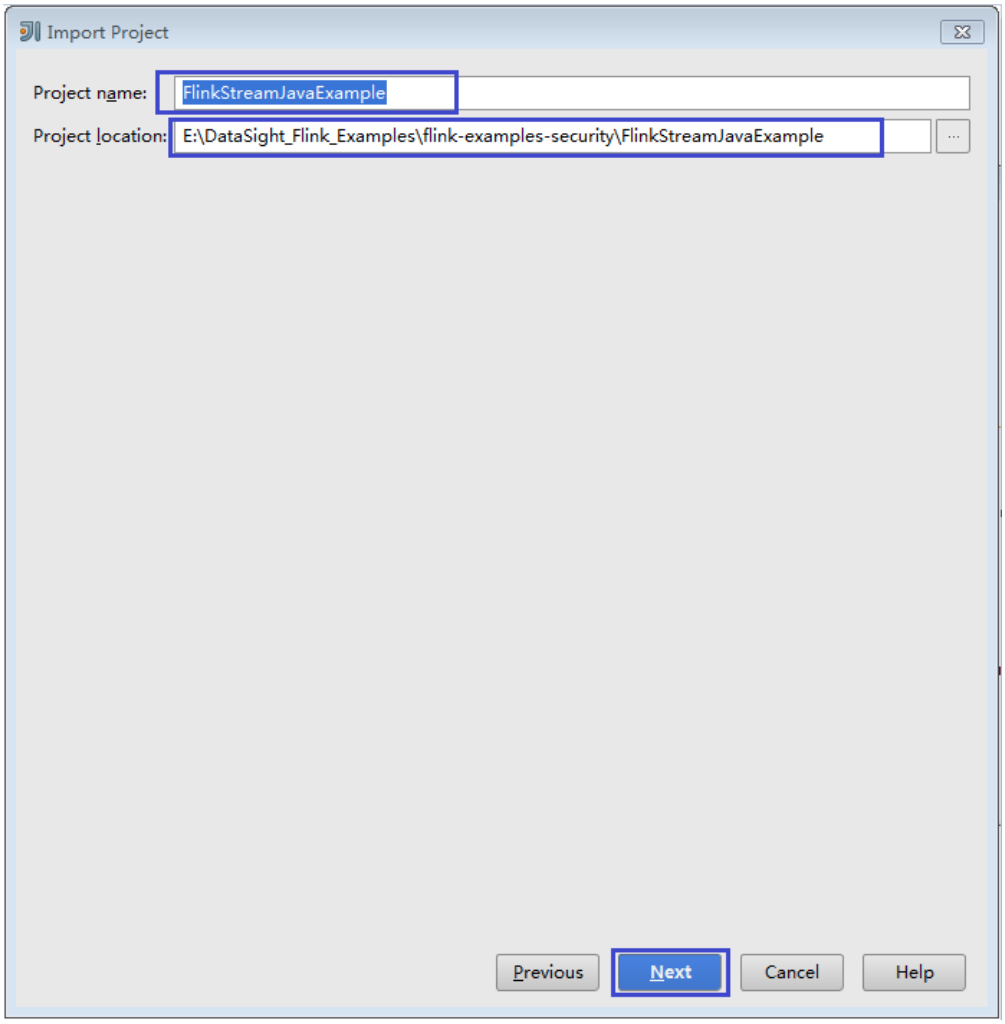
3. Select **Create project from existing sources** and click **Next**.

Figure 5-11 Create project from existing sources



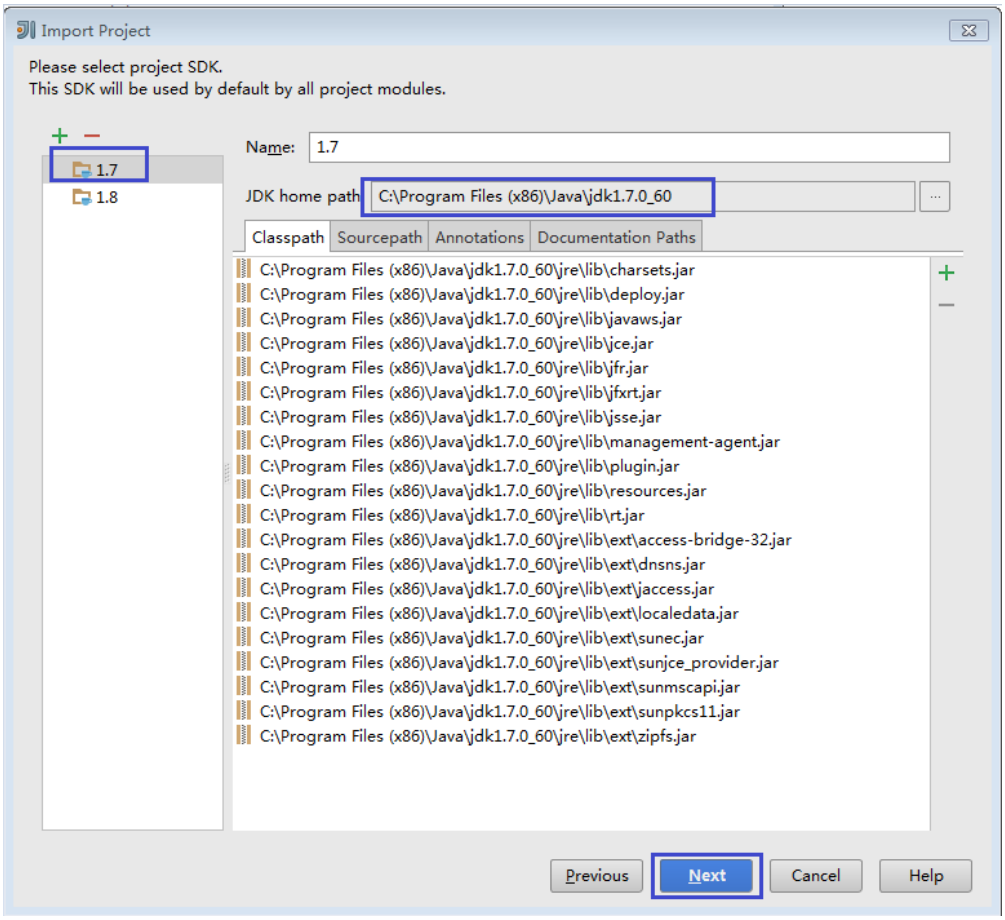
4. Confirm the project location and project name and click **Next**.

Figure 5-12 Import Project



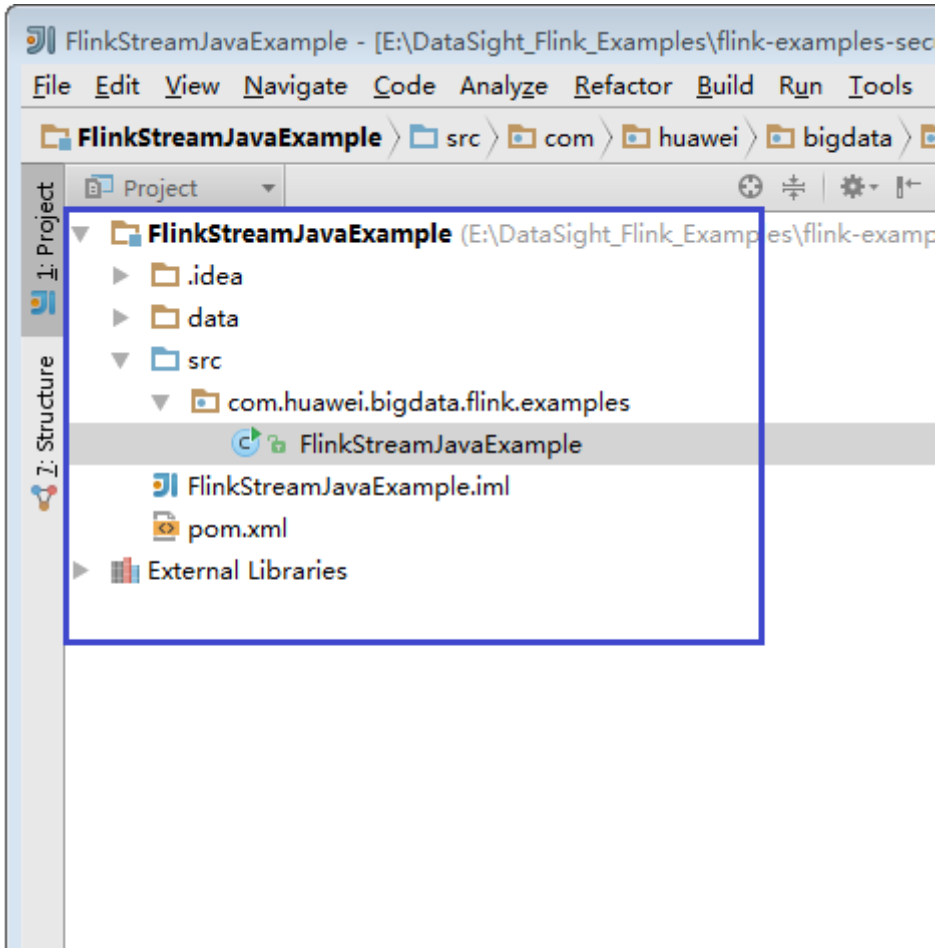
5. Retain the default value of the **root** directory for the project to be imported and click **Next**.
6. Retain the default dependency library that is automatically recognized by IDEA and the suggested module structure and click **Next**.
7. Confirm the JDK used by the project and click **Next**.

Figure 5-13 Selecting the project SDK



8. After the import is complete, click **Finish**. Then the imported sample project is displayed on the IDEA home page.

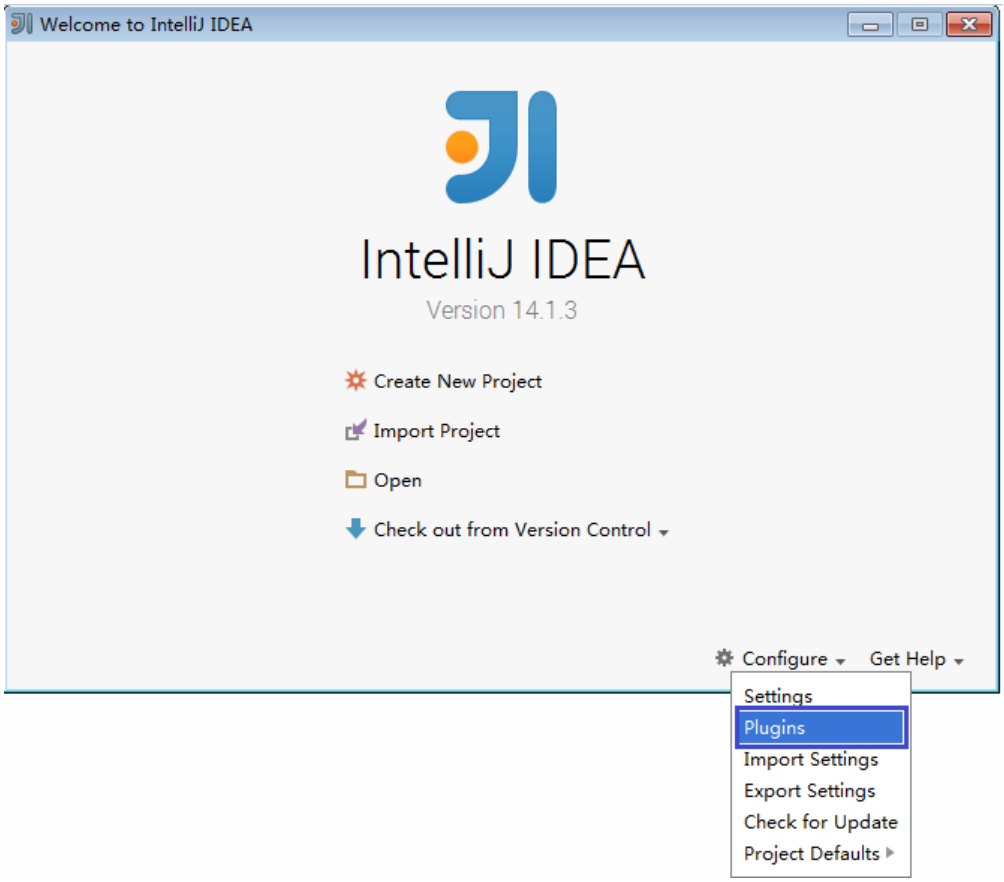
Figure 5-14 Imported project



Step 7 (Optional) If a Scala sample application is imported, install the Scala plug-in on the IntelliJ IDEA.

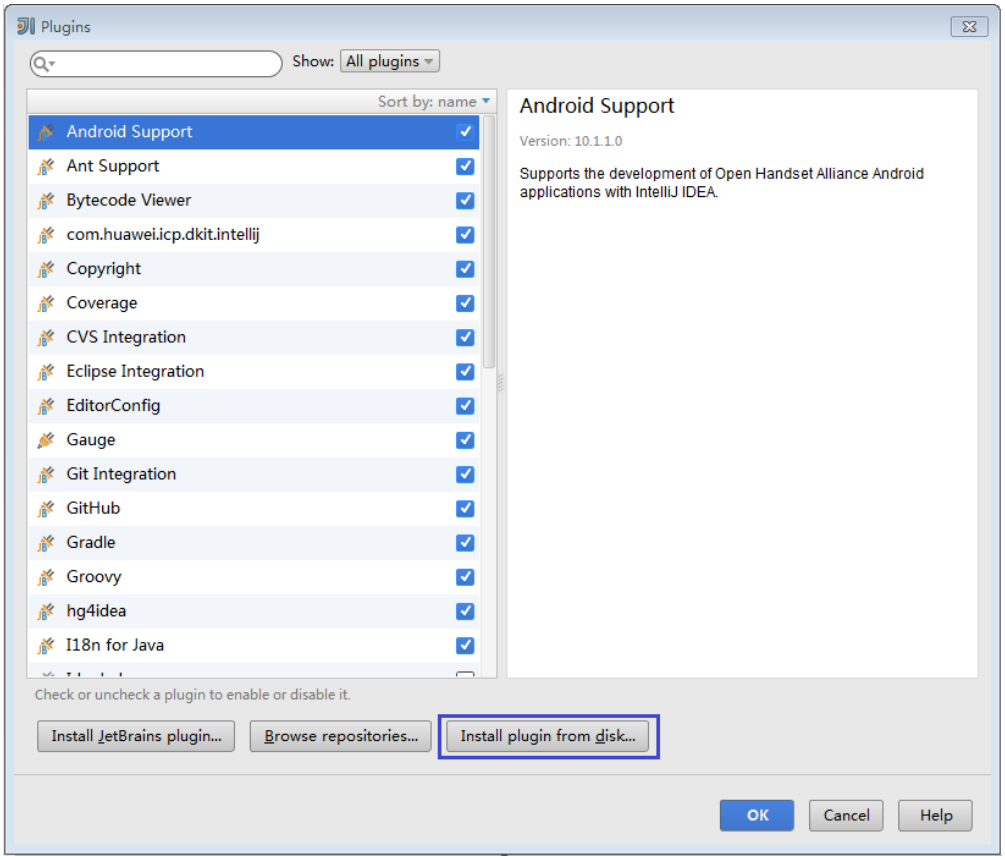
1. Choose **Plugins** from the **Configure** drop-down list.

Figure 5-15 Plugins



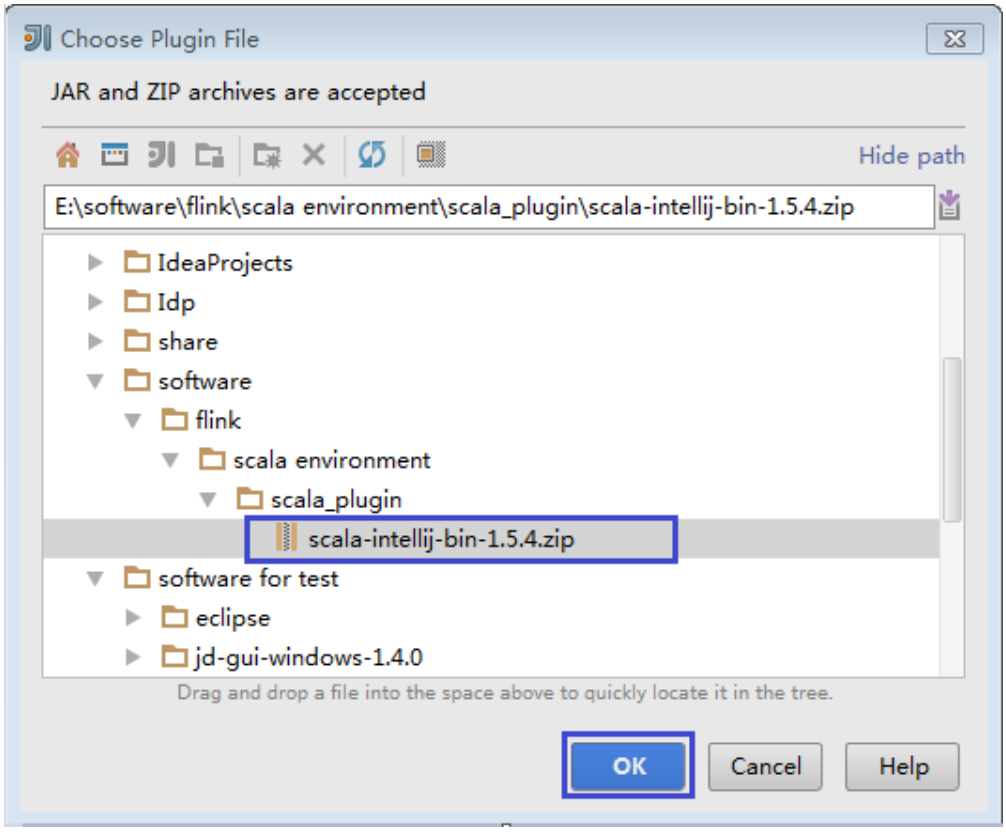
2. On the **Plugins** page, choose **Install plugin from disk**.

Figure 5-16 Choosing **Install plugin from disk**



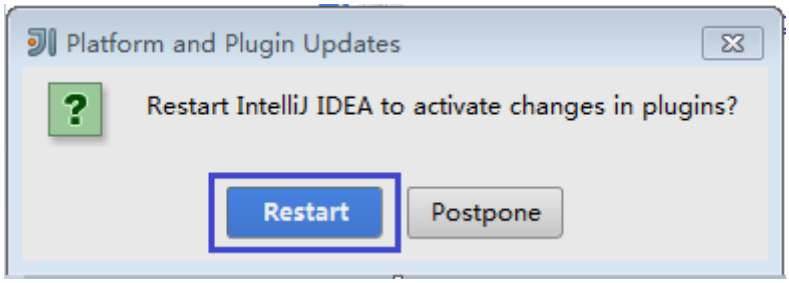
3. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.

Figure 5-17 Choose Plugin File



4. On the **Plugins** page, click **Apply** to install the Scala plugin.
5. On the **Platform and Plugin Updates** page that is displayed, click **Restart** to make the configurations take effect.

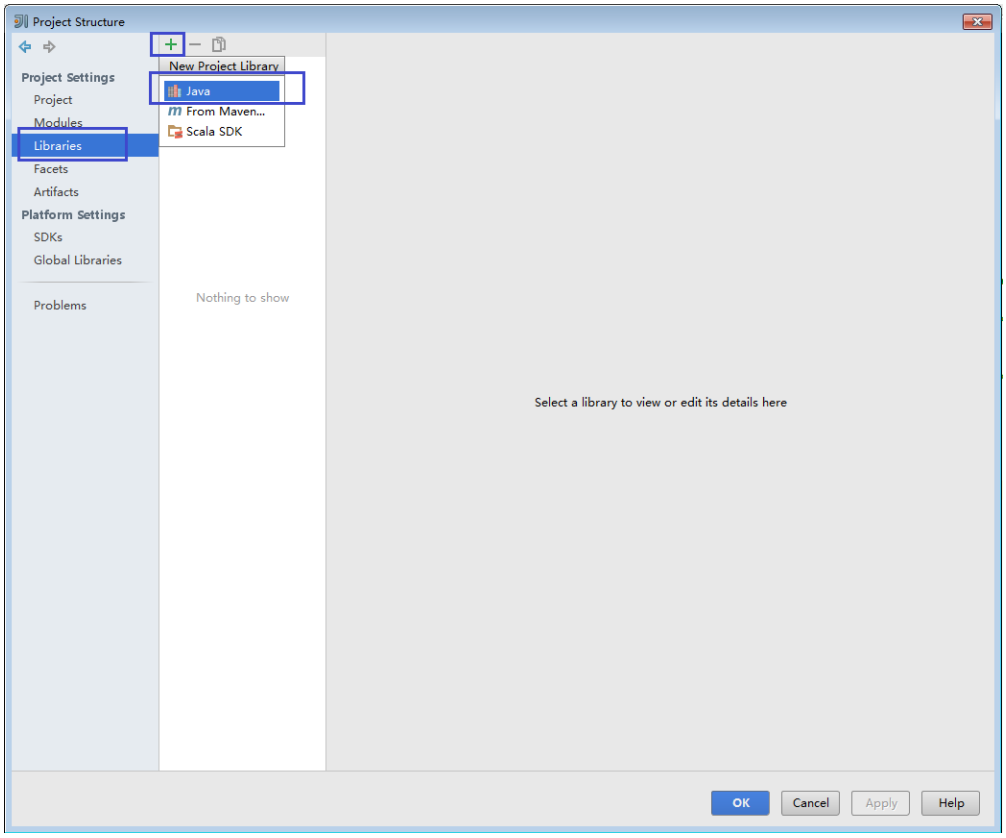
Figure 5-18 Platform and Plugin Updates



Step 8 Import the dependency JAR file for the sample project.

1. On the IDEA home page, choose **File** > **Project Structures...** to go to the **Project Structure** page.
2. Select **Libraries** and click **+** to add the dependency package of **Java**.

Figure 5-19 Add Java



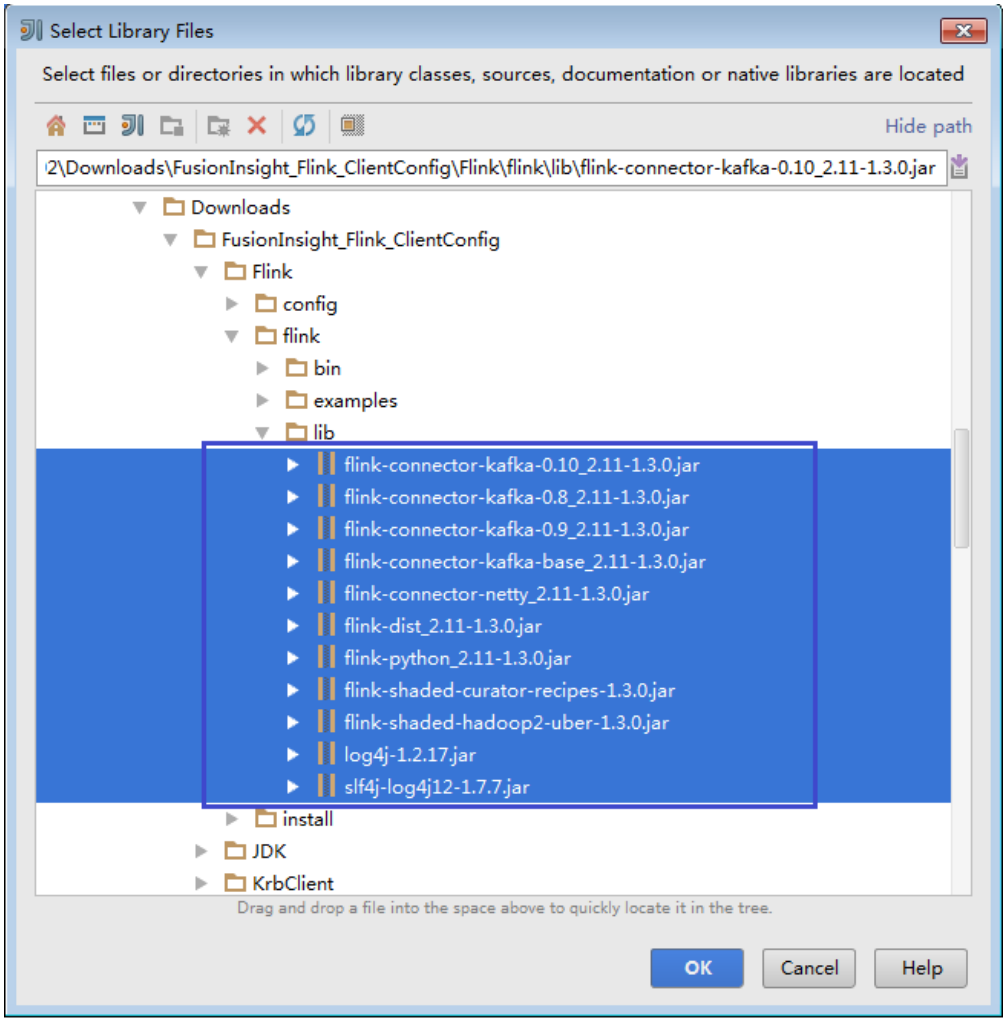
3. On the **Select Library Files** page, select all JAR files in the **lib** directory, and click **OK**.

   Flink dependency package: Select all JAR files in the **lib** directory. Alternatively, select a minimum of corresponding JAR files based on various sample projects.

   NOTE

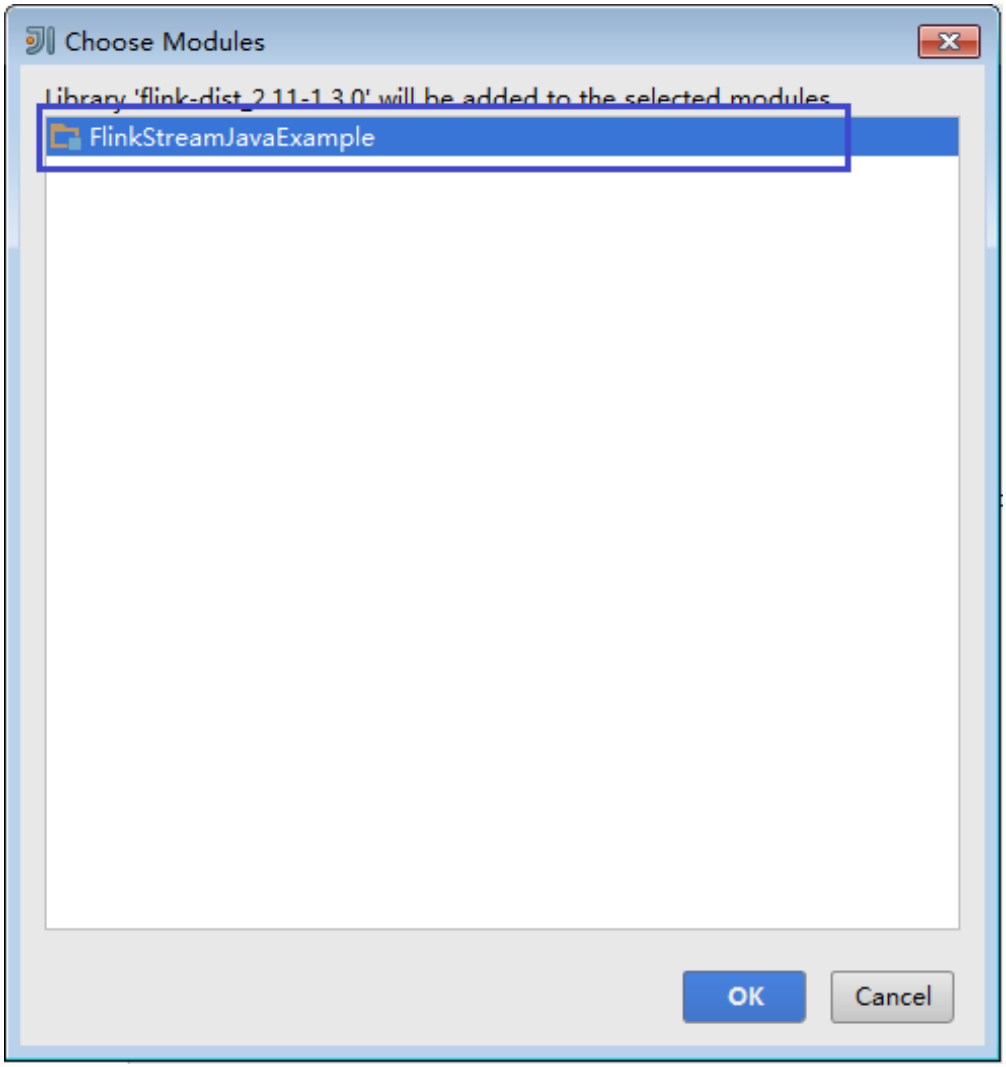
   If other MRS components are used in the sample code, obtain them from the installation directory of these MRS components.

Figure 5-20 Select Library Files



On the **Choose Modules** page, select all modules for the sample project. Then, click **OK**.

Figure 5-21 Choose Modules

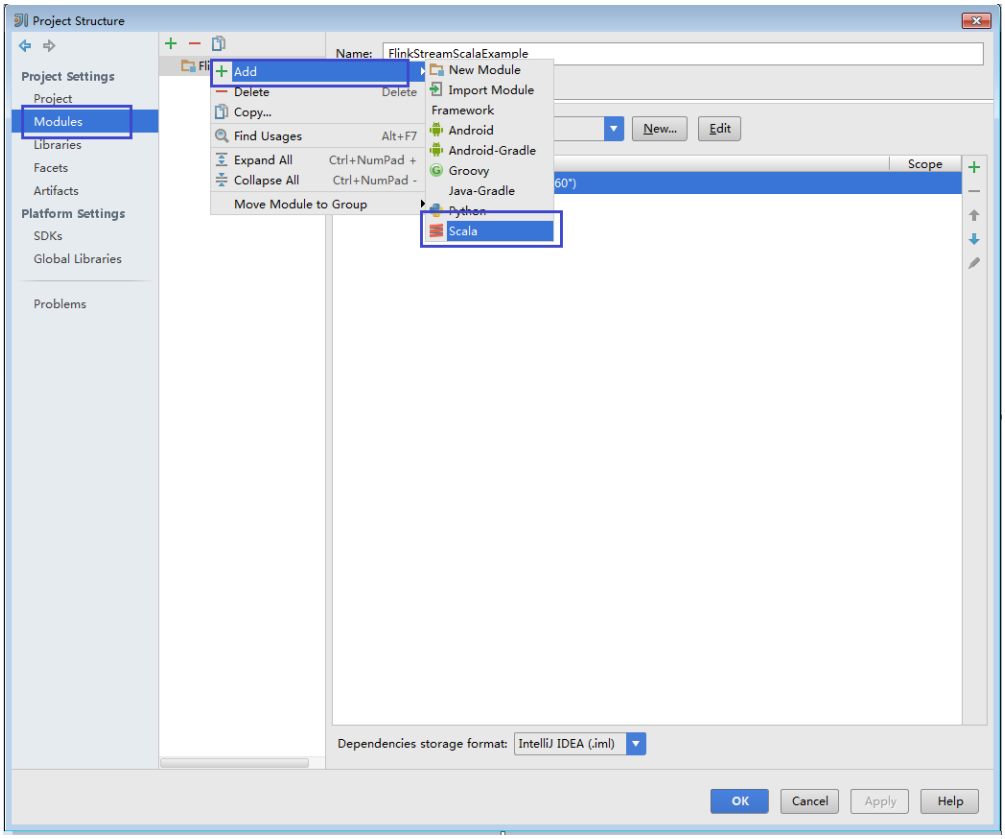


4. Click **OK** to complete the project library import.

Step 9 (Optional) If a Scala sample application is imported, configure a language for the project.

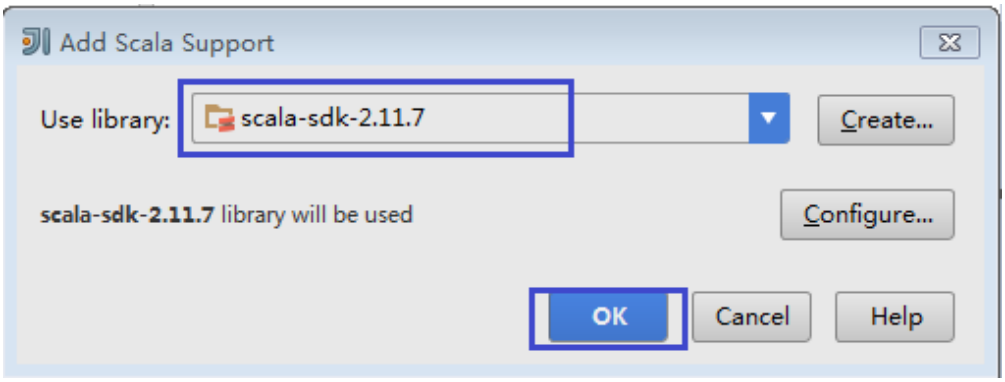
1. On the IDEA home page, choose **File** > **Project Structures...** to go to the **Project Structure** page.
2. Choose **Modules**, right-click a project name, and choose **Add** > **Scala**.

Figure 5-22 Selecting Scala



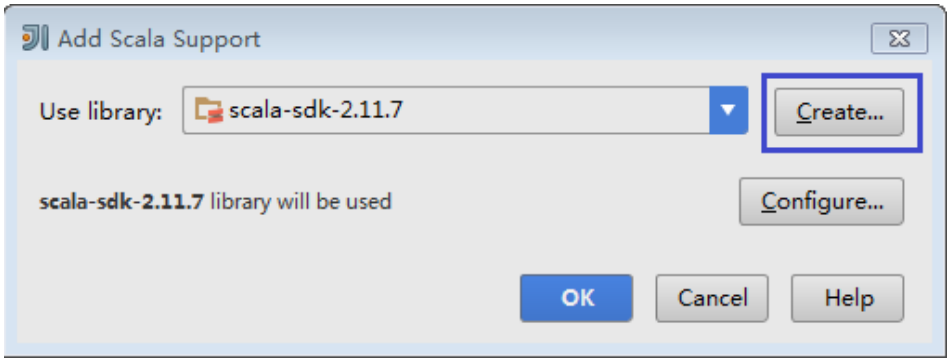
3. Wait until IDEA identifies the Scala SDK, choose the dependency JAR files on the **Add Scala Support** page, and then click **OK**.

Figure 5-23 Add Scala Support



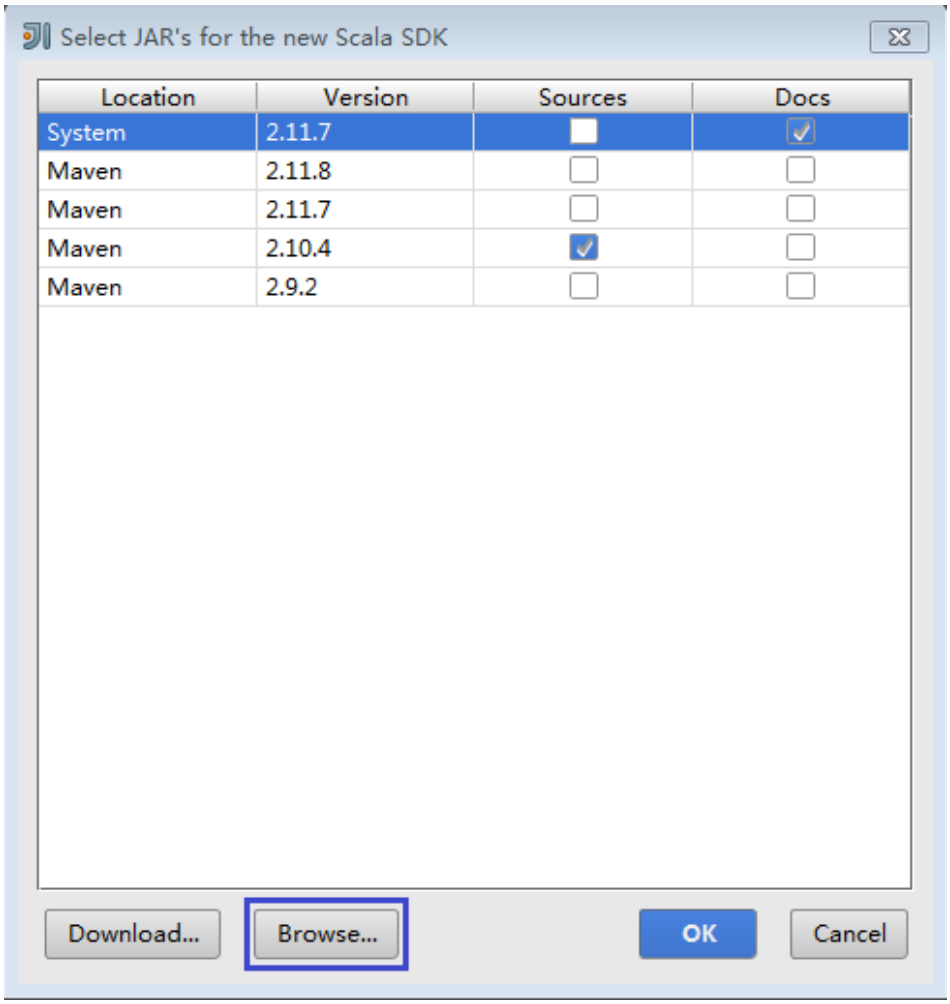
4. If IDEA fails to identify the Scala SDK, you need to create a Scala SDK.
   a. Click **Create ...**.

Figure 5-24 Clicking **Create...**



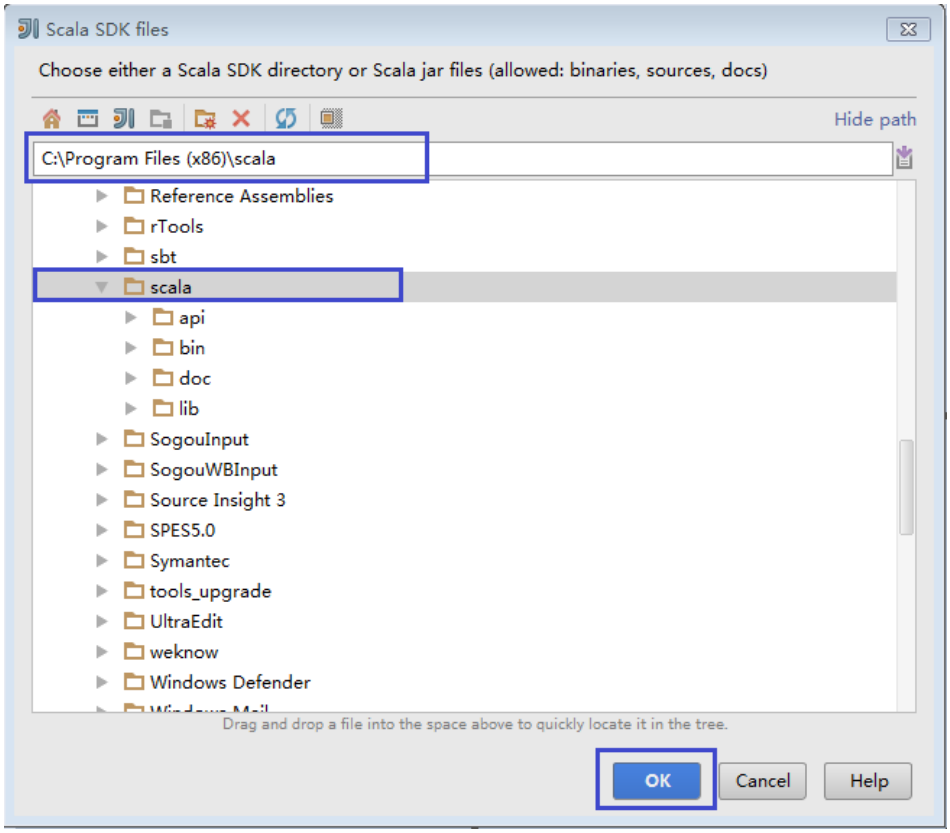
b. On the **Select JAR's for the new Scala SDK** page, click **Browse...**.

Figure 5-25 Select JAR's for the new Scala SDK



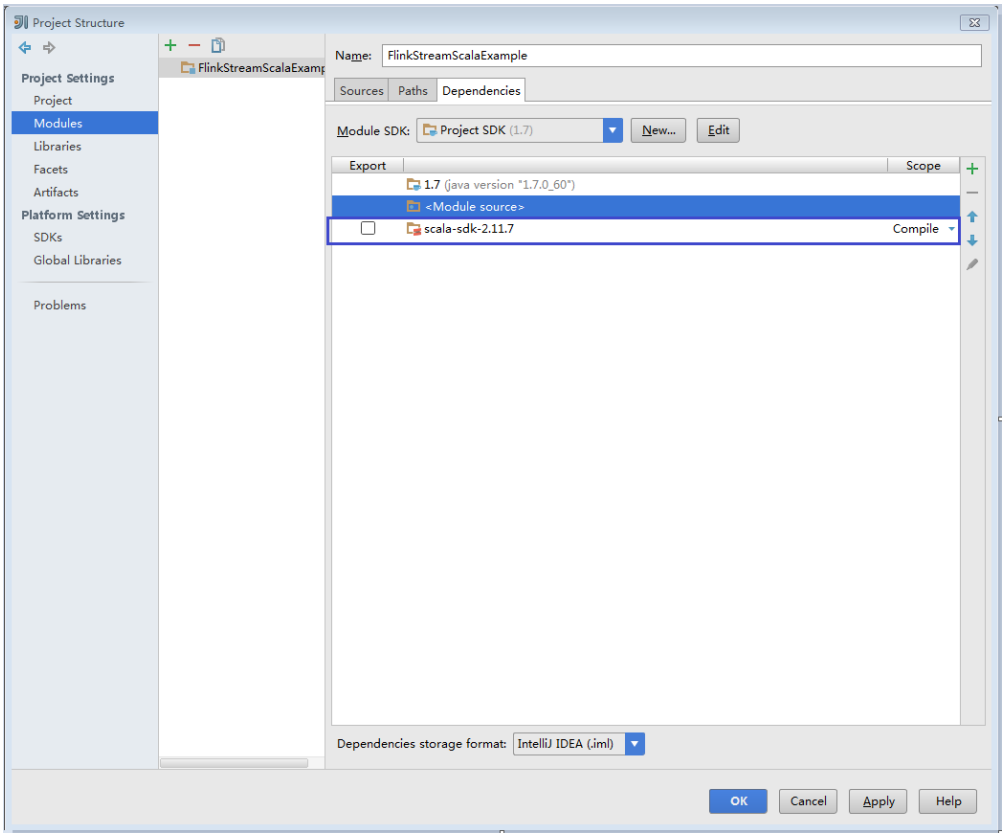
c. On the **Scala SDK files** page, select the **scala sdk** directory, and then click **OK**.

Figure 5-26 Scala SDK files



5. Click OK.

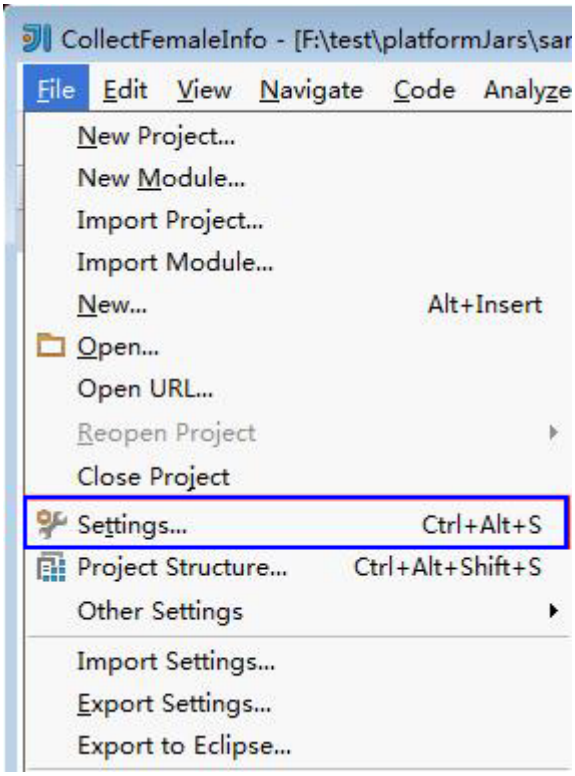
Figure 5-27 Configuration successful



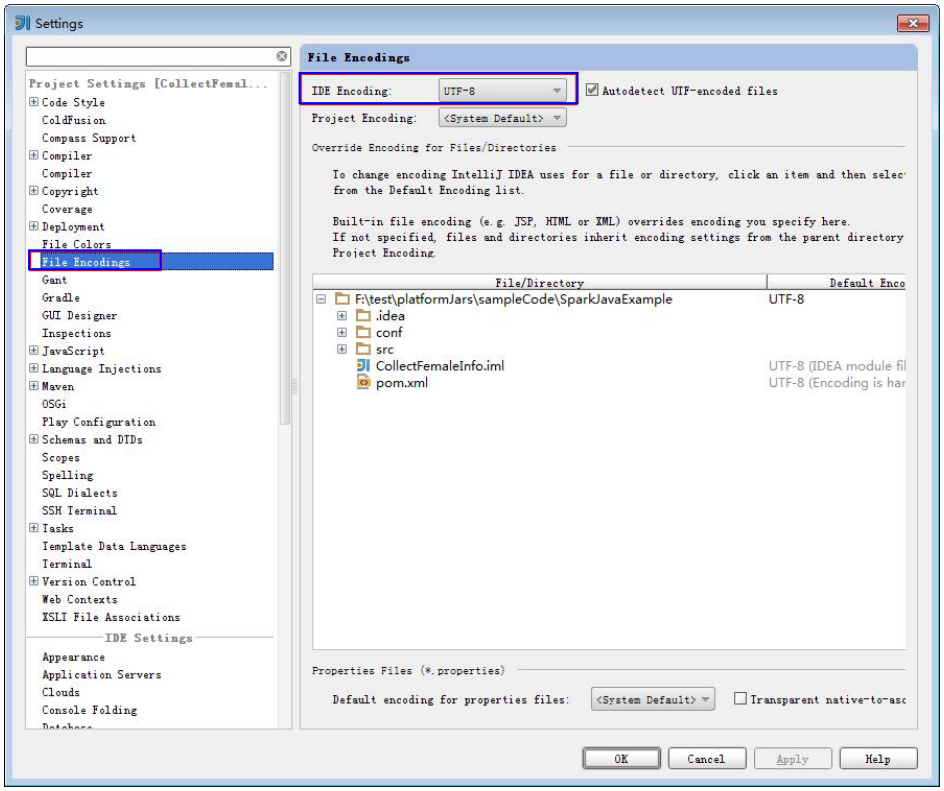
Step 10 Configure the text file encoding format of IDEA to prevent garbled characters.

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

Figure 5-28 Choosing **Settings**



2. On the **Settings** page, unfold **Editor**, and choose **File Encodings**. Select **UTF-8** from the **IDE Encoding** and **Project Encoding** drop-down lists on the right. Click **Apply**.



3. Click **OK** to complete the encoding settings.

----End

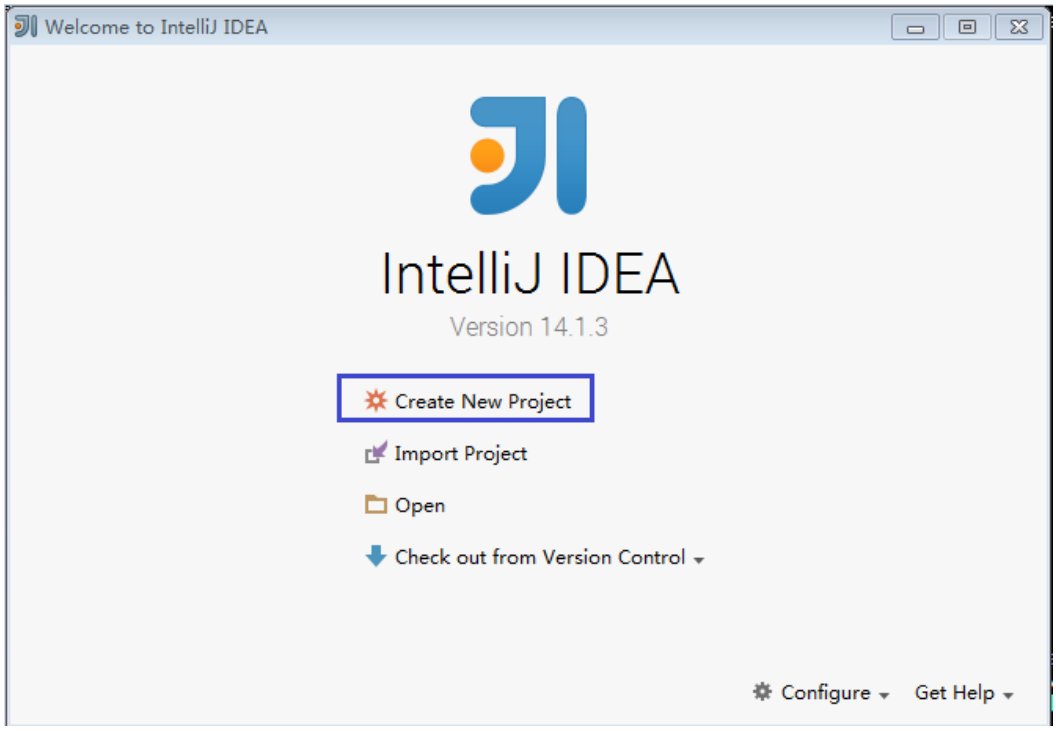
5.2.5 (Optional) Creating Flink Sample Projects

In addition to importing Flink sample projects, you can use IDEA to create a Flink project. The following describes how to create a Scala project.

Procedure

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

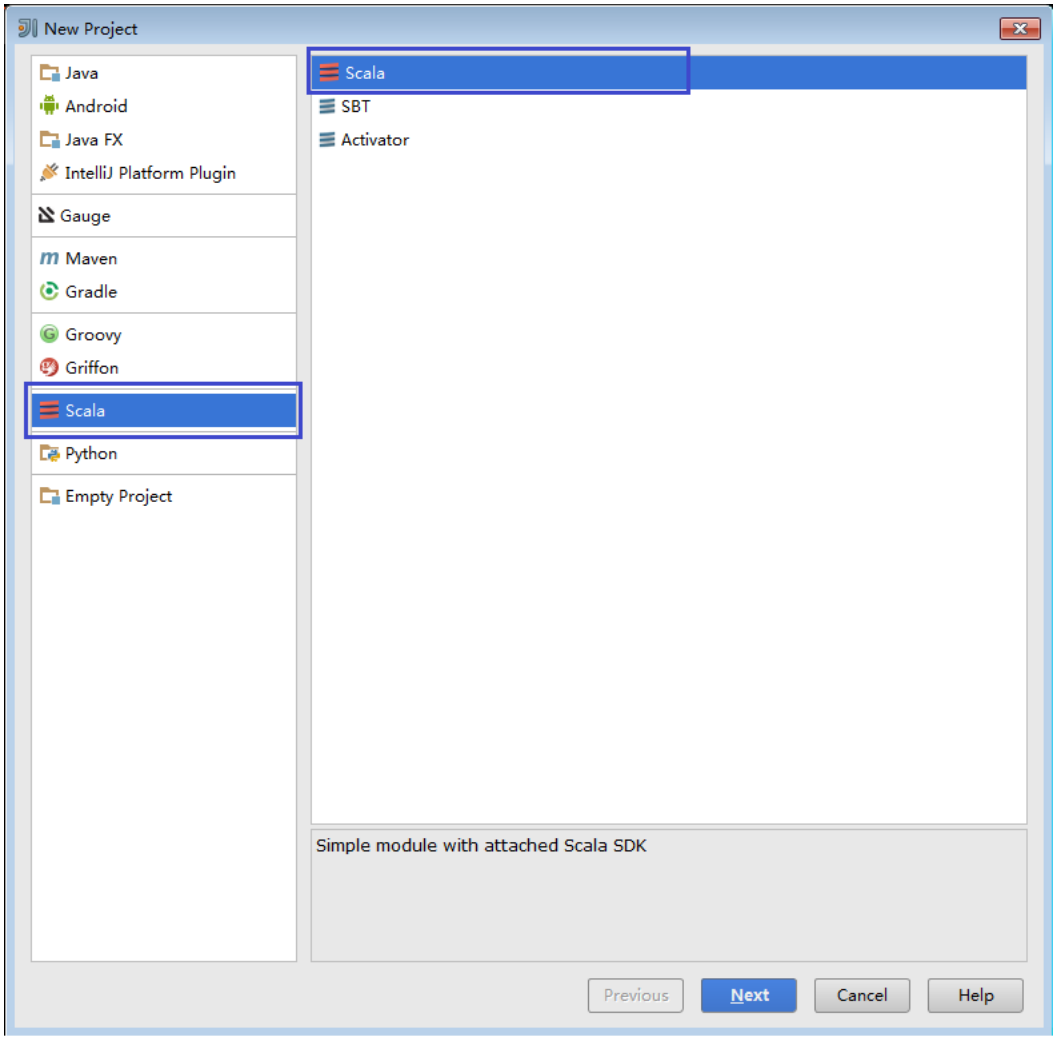
Figure 5-29 Creating a project



Step 2 On the **New Project** page, choose **Scala** > **Scala Module** and click **Next**.

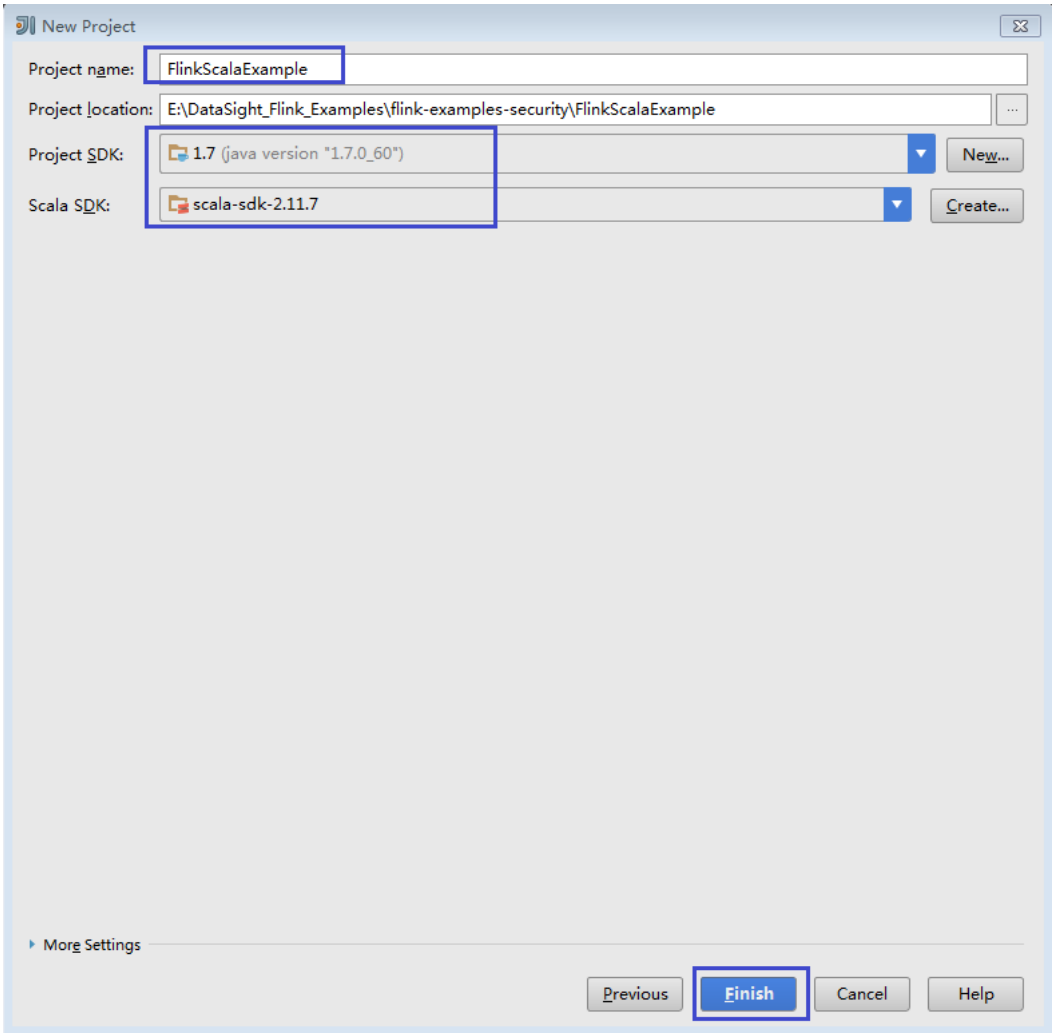
If you need to create a Java project, select the corresponding parameter.

Figure 5-30 Selecting a development environment



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

Figure 5-31 Entering the project information



----End

5.2.6 Preparing the Flink Application Security Authentication

If Kerberos authentication is enabled for the MRS cluster, perform the following steps to prepare a development user. If Kerberos authentication is not enabled, skip the following steps.

In a security cluster environment, the components must be mutually authenticated before communicating with each other to ensure communication security.

When submitting a Flink application, you need to communicate with Yarn and HDFS. Security authentication needs to be configured for the Flink application to be submitted to ensure that the Flink application can work properly.

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

Security Authentication

Flink uses the following two authentication modes:

- Kerberos authentication: It is used between the Flink YARN client and YARN ResourceManager, JobManager and ZooKeeper, JobManager and HDFS, TaskManager and HDFS, Kafka and TaskManager, as well as TaskManager and ZooKeeper.
- Internal authentication mechanism of Yarn: It is used between Yarn ResourceManager and ApplicationMaster.

  NOTE

  Flink JobManager and YARN ApplicationMaster are in the same process.

Table 5-4 Security authentication mode

Security Authentication Mode	Description	Configuration
Kerberos authentication	Currently, only keytab authentication is supported.	1. Download the user keytab file from the KDC server, and place the keytab file to a folder on the host of the Flink client (for example, **/home/flinkuser/keytab**). 2. Configure the following parameters in the **${FLINK_HOME}/conf/flink-conf.yaml** file: a. Keytab file path `security.kerberos.login.keytab: /home/flinkuser/keytab/user.keytab` **NOTE** **/home/flinkuser/keytab/** indicates the directory for storing the keytab file. b. Principal name (developer username). `security.kerberos.login.principal:flinkuser` c. In HA mode, if Zookeeper is configured, ZooKeeper Kerberos authentication must be configured as follows: `zookeeper.sasl.disable: false` `security.kerberos.login.contexts: Client` d. If Kerberos authentication is required between the Kafka client and Kafka broker, configure it as follows: `security.kerberos.login.contexts: Client,KafkaClient`
Internal authentication of YARN	The user does not need to configure this internal authentication mode.	-

NOTE

One Flink cluster belongs to only one user. One user can create multiple Flink clusters.

Encrypted Transmission

Flink uses the following three encrypted transmission modes:

- Encrypted transmission inside YARN: It is used between the Flink YARN client and YARN ResourceManager, as well as YARN ResourceManager and JobManager.
- SSL transmission: It is used between the Flink YARN client and JobManager, JobManager and TaskManager, as well as TaskManagers.
- Encrypted transmission inside Hadoop: It is used between JobManager and HDFS, TaskManager and HDFS, JobManager and ZooKeeper, and TaskManager and ZooKeeper.

NOTE

You do not need to configure encryption inside YARN and Hadoop, but need to configure SSL transmission.

To configure SSL transmission, configure the **flink-conf.yaml** file on the client.

Step 1 Turn on the SSL switch and set SSL encryption algorithms. **Table 5-5** describes the parameters. Set the parameters based on site requirements.

Table 5-5 Parameters

Parameter	Example Value	Description
security.ssl.internal.enabled	true	Switch to enable internal SSL
akka.ssl.enabled	true	Switch to enable Akka SSL
blob.service.ssl.enabled	true	Switch to enable SSL of the BLOB channels
taskmanager.data.ssl.enabled	true	Switch to enable SSL for communications between TaskManagers
security.ssl.algorithms	TLS_RSA_WITH_AES_128_CBC_SHA256	SSL encryption algorithms

The following parameters are not included in the default Flink configurations of MRS. You can add them if necessary. If you enable SSL for external connections, the proxy of YARN cannot access the Flink page. This is because Yarn does not support the HTTPS proxy. There can be security risks if a configuration file contains the authentication password. You are advised to delete the configuration file or use other secure methods to keep the password.

Parameter	Example Value	Description
security.ssl.rest.enabled	true	Switch to enable external SSL
security.ssl.rest.keystore	${path}/flink.keystore	Path for storing the keystore
security.ssl.rest.keystore-password	123456	Password of the keystore. The value **123456** indicates a user-defined password.
security.ssl.rest.key-password	123456	Password of the SSL key. The value **123456** indicates a user-defined password.
security.ssl.rest.truststore	${path}/flink.truststore	Path for storing the truststore
security.ssl.rest.truststore-password	123456	Password of the truststore. The value **123456** indicates a user-defined password.

NOTE

Enabling SSL for data transmission between TaskManagers may pose great impact on system performance. You need to take both security and performance into consideration.

Step 2 In the **bin** directory of the Flink client, run the **sh generate_keystore.sh** *<Password>* command. The configuration items in **Table 5-6** are set by default. You can also set the configuration items yourself. There can be security risks if a command contains the authentication password. You are advised to disable the command recording function (history) before running the command.

Table 5-6 Parameters

Parameter	Example Value	Description
security.ssl.internal.keystore	${path}/flink.keystore	Path for storing the **keystore** file. **flink.keystore** indicates the name of the **keystore** file generated by the **generate_keystore.sh*** tool.

Parameter	Example Value	Description
security.ssl.internal.keystore-password	123456	Password of the keystore. The value **123456** indicates a user-defined password.
security.ssl.internal.key-password	123456	Password of the SSL key. The value **123456** indicates a user-defined password.
security.ssl.internal.truststore	${path}/flink.truststore	Path for storing the **truststore** file. **flink.truststore** indicates the name of the **truststore** file generated by the **generate_keystore.sh*** tool.
security.ssl.internal.truststore-password	123456	Password of the truststore. The value **123456** indicates a user-defined password.

If SSL for external connections is enabled, that is, **security.ssl.rest.enabled** is set to **true**, the following parameters need to be set:

Parameter	Example Value	Description
security.ssl.rest.keystore	${path}/flink.keystore	Path for storing the keystore
security.ssl.rest.keystore-password	123456	Password of the keystore. The value **123456** indicates a user-defined password.
security.ssl.rest.key-password	123456	Password of the SSL key. The value **123456** indicates a user-defined password.
security.ssl.rest.truststore	${path}/flink.truststore	Path for storing the truststore
security.ssl.rest.truststore-password	123456	Password of the truststore. The value **123456** indicates a user-defined password.

path indicates a user-defined directory that is used to store configuration files of the SSL keystore and truststore. The commands vary according to the relative path and absolute path. The details are as follows:

NOTE

- Configure the file path storing the **keystore** or **truststore** file to a relative path, and the Flink client directory where the command is executed can directly access this relative path.
  security.ssl.internal.keystore: ssl/flink.keystore
  security.ssl.internal.truststore: ssl/flink.truststore
- 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 nodes.
  security.ssl.internal.keystore: /opt/client/Flink/flink/conf/flink.keystore
  security.ssl.internal.truststore: /opt/client/Flink/flink/conf/flink.truststore

- Configure the file path storing the **keystore** or **truststore** file to a relative path, and the Flink client directory where the command is executed can directly access this relative path. Flink can transfer the **keystore** and **truststore** files using either of the following methods:
  - Add the **-t** option to the **CLI yarn-session.sh** command of Flink to transfer the **keystore** and **truststore** files to execution nodes. Example:
    ```
    ./bin/yarn-session.sh -t ssl/ -n 2
    ```
  - Add the **-yt** option to the **flink run** command to transfer the **keystore** and **truststore** files to execution nodes. Example:
    ```
    ./bin/flink run -yt ssl/ -ys 3 -yn 3 -m yarn-cluster -c com.huawei.SocketWindowWordCount lib/flink-eg-1.0.jar --hostname r3-d3 --port 9000
    ```

    NOTE

    - In the preceding example, **ssl/** is the sub-directory of the Flink client directory and is used to store configuration files of the SSL keystore and truststore.
    - The relative path of **ssl/** must be accessible from the current path where the Flink client command is executed.
- If the **keystore** or **truststore** file path is an absolute path, the **keystore** and **truststore** files must exist in the absolute path on Flink Client and all nodes. In addition, the user who submits the job must have permission to read the files.

  Either of the following methods can be used to run applications. The **-t** or **-yt** option does not need to be added to transfer the **keystore** and **truststore** files.
  - Run the **CLI yarn-session.sh** command of Flink to execute applications. Example:
    ```
    ./bin/yarn-session.sh -n 2
    ```
  - Run the **flink run** command to execute applications. Example:
    ```
    ./bin/flink run  -ys 3 -yn 3 -m yarn-cluster -c com.huawei.SocketWindowWordCount lib/flink-eg-1.0.jar --hostname r3-d3 --port 9000
    ```

----End

5.3 Developing Flink Applications

5.3.1 DataStream Application

5.3.1.1 Flink DataStream Development Plan

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

NOTE

The DataStream application can run in Windows- and Linux-based environments.

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours in real time.
- 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 columns are separated by comma (,).

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 Planning

Data of the DataStream sample project is stored in TXT format.

Store the **log1.txt** and **log2.txt** files in a path of the user development program, for example, **/opt/log1.txt** and **/opt/log2.txt**.

Development Guidelines

Collect statistics on female netizens who dwell on online shopping for more than 2 hours on the weekend.

To achieve the objective, the process is as follows:

1. Read text data, generate DataStreams, and parse data to generate UserRecord information.

2. Filter data information of the time that female netizens spend online.
3. Perform keyby operation based on the name and gender, and summarize the total time that each female netizen spends online within a time window.
4. Filter data about netizens whose online duration exceeds the threshold, and obtain the results.

5.3.1.2 Flink DataStream Java Sample Code

Function Description

Collect statistics on female netizens who continuously dwell on online shopping for more than 2 hours and print statistics directly.

Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.flink.example.stream.FlinkStreamJavaExample**.

```
    // Parameter description:
    // <filePath> is the path for reading text files. The paths are separated by commas (,).
    // <windowTime> is the time span of the statistics data. The unit is minute.
public class FlinkStreamJavaExample {
    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.flink.examples.stream.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("*****************************************************************************************");
       // Read text path information. The 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 is used to set the time window. The default value is 2 minutes per time window. One
time window is sufficient to read all data in the text.
        final int windowTime = ParameterTool.fromArgs(args).getInt("windowTime", 2);
       // Construct an execution environment and use eventTime to process the data obtained in a time
window.
        final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
        env.setParallelism(1);
       // Read the text data stream.
        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 the data, construct the entire data processing logic, and calculate and print the 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();
       // Invoke execute to trigger the execution.
        env.execute("FemaleInfoCollectionPrint java");
    }
    // Construct a keyBy keyword 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);
        }
    }
    // Resolve the text line data and construct the 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]));
    }
    // Define the UserRecord data structure and rewrite the 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;
        }
    }
    // Construct a class inherited from AssignerWithPunctuatedWatermarks to set 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 start execution, 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
```


5.3.1.3 Flink DataStream Scala Sample Code

Function Description

Collect statistics on female netizens who continuously dwell on online shopping for more than 2 hours in real time and print statistics directly.

Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.flink.example.stream.FlinkStreamScalaExample**.

```
    // Parameter description:
    // filePath is the path for reading text files. The paths are separated by commas (,).
    // windowTime is the time span of the statistics data. The unit is minute.
object FlinkStreamScalaExample {
  def main(args: Array[String]) {
    // 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.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("*******************************************************************************************")

    // Read text path information. The 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 is used to set the time window. The default value is 2 minutes per time window. One
time window is sufficient to read all data in the text.
    val windowTime = ParameterTool.fromArgs(args).getInt("windowTime", 2)

    // Construct an execution environment and use eventTime to process the data obtained in a time
window.
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
    env.setParallelism(1)

    // Read the text data stream.
    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 the data, construct the entire data processing logic, and calculate and print the 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()

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

  // Resolve the text line data and construct the 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)
  }

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


  // Construct a class inherited from AssignerWithPunctuatedWatermarks to set 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 start execution, 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)
```

5.3.2 Application for Producing and Consuming Data in Kafka

5.3.2.1 Development Plan of Kafka Data Producing and Consuming

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

Develop a Flink application that can output prefixed message content in real time.

Data Planning

Flink sample project data is stored in Kafka. Data is sent to and obtained from Kafka (user with Kafka permission required).

Step 1 Ensure that a cluster containing HDFS, YARN, Flink, and Kafka has been successfully installed.

Step 2 Create a topic.

1. Configure the user permission for creating topics on the server.

   For a security cluster with Kerberos authentication enabled, change the value of the Kafka broker configuration parameter **allow.everyone.if.no.acl.found** to **true**. After the configuration is complete, restart Kafka. You do not need to configure this parameter for normal clusters with Kerberos authentication disabled.

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

NOTE

flinkuser is created by yourself and has permission to create Kafka topics. For details, see **Preparing a Flink Application Development User**.

The command for creating a topic is as follows:

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

Table 5-7 Parameters

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

For example, run the following command in the Kafka client path. In the following command example, the values of the IP:port of the ZooKeeper cluster are 10.96.101.32:2181,10.96.101.251:2181,10.96.101.177:2181, and the topic name is **topic1**.

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

Step 3 If Kerberos authentication is enabled for the cluster, perform this step for security authentication. Otherwise, skip this step.

- Configuration of Kerberos authentication

  a. Client configuration

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

  ```
  security.kerberos.login.keytab: /home/demo/flink/release/flink-1.2.1/keytab/admin.keytab
  security.kerberos.login.principal: admin
  security.kerberos.login.contexts: Client,KafkaClient
  security.kerberos.login.use-ticket-cache: false
  ```

  b. Running parameter

  The following is an example of running parameters about the **SASL_PLAINTEXT** protocol:

  ```
  --topic topic1 --bootstrap.servers 10.96.101.32:21007 --security.protocol SASL_PLAINTEXT  --sasl.kerberos.service.name kafka //10.96.101.32:21007 indicates the IP:port of the Kafka server.
  ```

----End

Development Guidelines

1. Start the Flink Kafka Producer application to send data to Kafka.

2. Start the Flink Kafka Consumer application to receive data from Kafka. Ensure that topics of Kafka Consumer are consistent with that of Kafka Producer.
3. Add a prefix to data content and print the result.

5.3.2.2 Java Sample Code of Kafka Data Producing and Consuming

Function Description

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

If you need to interconnect with Kafka in security mode before application development, **kafka-client-xx.x.x.jar** of MRS is required. You can obtain the JAR file in the MRS client directory.

Sample Code

The following example shows the main logic code of Kafka Consumer and Kafka Producer.

For the complete codes, see **com.huawei.bigdata.flink.examples.WriteIntoKafka** and **com.huawei.flink.example.kafka.ReadFromKafka**.

```
// Kafka 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 parallelism.
      env.setParallelism(1);
      // Parse the running parameters.
      ParameterTool paraTool = ParameterTool.fromArgs(args);
     // Construct a StreamGraph and write the data generated from self-defined sources to Kafka.
      DataStream<String> messageStream = env.addSource(new SimpleStringGenerator());
      messageStream.addSink(new FlinkKafkaProducer010<>(paraTool.get("topic"),
          new SimpleStringSchema(),
          paraTool.getProperties()));
      // Invoke execute to trigger the execution.
      env.execute();
    }
   // Customize the sources and generate a message every other 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;
      }
    }
  }
// Kafka Consumer code
public class ReadFromKafka {
    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.flink.example.kafka.ReadFromKafka" +
          " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21005");
      System.out.println("./bin/flink run --class com.huawei.flink.example.kafka.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
("*************************************************************************************************");
     // Construct the execution environment.
      StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
     // Set parallelism.
      env.setParallelism(1);
      // Parse the running parameters.
      ParameterTool paraTool = ParameterTool.fromArgs(args);
    //Construct a StreamGraph, read data from Kafka, and print the data in a new line.
      DataStream<String> messageStream = env.addSource(new
FlinkKafkaConsumer010<>(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();
      // Invoke execute to trigger the execution.
      env.execute();
    }
  }
```

5.3.2.3 Scala Sample Code of Kafka Data Producing and Consuming

Function Description

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

If you need to interconnect with Kafka in security mode before application development, **kafka-client-xx.x.x.jar** of MRS is required. You can obtain the JAR file in the MRS client directory.

Sample Code

The following example shows the main logic code of Kafka Consumer and Kafka Producer.

For the complete codes, see **com.huawei.bigdata.flink.examples.WriteIntoKafka** and **com.huawei.flink.example.kafka.ReadFromKafka**.

```
// Kafka Producer code
object WriteIntoKafkaScala {
```



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

   System.out.println("./bin/flink run --class com.huawei.flink.example.kafka.WriteIntoKafkaScala" +

     " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21005")

   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.
   val env = StreamExecutionEnvironment.getExecutionEnvironment
   // Set parallelism.
   env.setParallelism(1)
   // Parse the running parameters.
   val paraTool = ParameterTool.fromArgs(args)
   // Construct a StreamGraph and write the data generated from self-defined sources to Kafka.
   val messageStream: DataStream[String] = env.addSource(new SimpleStringGeneratorScala)

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

 }
}
   // Customize the sources and generate a message every other second.
class SimpleStringGeneratorScala 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
 }
}
// Kafka Consumer code
object ReadFromKafkaScala {
 def main(args: Array[String]) {
   // Print the command reference for flink run.
   System.out.println("use command as: ")

   System.out.println("./bin/flink run --class com.huawei.flink.example.kafka.ReadFromKafkaScala" +

     " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21005")

   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.
  val env = StreamExecutionEnvironment.getExecutionEnvironment
  // Set parallelism.
  env.setParallelism(1)
  // Parse the running parameters.
  val paraTool = ParameterTool.fromArgs(args)
  //Construct a StreamGraph, read data from Kafka, and print the data in a new line.
  val messageStream = env.addSource(new FlinkKafkaConsumer(

    paraTool.get("topic"), new SimpleStringSchema, paraTool.getProperties))

  messageStream

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

 }

}
```

5.3.3 Asynchronous Checkpoint Mechanism Application

5.3.3.1 Development Plan of Flink Asynchronous Checkpoint

Assume that you want to collect data volume in a 4-second time window every other second and the status of operators must be strictly consistent. That is, if an application recovers from a failure, the status of all operators must the same.

Data Planning

1. Customized operators generate about 10,000 pieces of data per second.
2. Generated data is of four tuples (Long, String, String, and Integer).
3. Statistic results are printed on the devices.
4. Printed data is of the Long type.

Development Guidelines

1. A source operator sends 10,000 pieces of data and injects the data to a window operator every other second.
2. The window operator collects the data volume statistics of the last 4 seconds every other second.
3. The statistics is printed to the device every other second. For details, see **Viewing the Running Result of a Flink Application**.
4. A checkpoint is triggered every other 6 seconds and the checkpoint result is stored in HDFS.

5.3.3.2 Java Sample Code of Flink Asynchronous Checkpoint

Sample Code

Assume that you want to collect data volume in a 4-second time window every other second and the status of operators must be strictly consistent.

- Snapshot data

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

```
import java.io.Seriablizale;
// As a part of the snapshot, this class saves the user-defined status.
public class UDFState implements Serializable {
    private long count;
    // Initialize the user-defined status.
    public UDFState() {
        count = 0L;
    }
    // Set the user-defined status.
    public void setState(long count) {
       this.count = count;
    }
    // Obtain the user-defined status.
    public long geState() {
        return this.count;
    }
}
```

- Data source with checkpoints

  The code snippet of a source operator pauses 1 second every time after sending 10,000 pieces of data. When a snapshot is created, the code saves the total number of sent data pieces in UDFState. When the snapshot is used for restoration, the number of sent data pieces saved 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.SourceFunction;

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

public class SimpleSourceWithCheckPoint implements SourceFunction<Tuple4<Long, String, String,
Integer>>, ListCheckpointed<UDFState> {

    private long count = 0;
    private boolean isRunning = true;
    private String alphabet = "justtest";

    @Override
    public List<UDFState> snapshotState(long l, long l1) throws Exception
    {
        UDFState udfState = new UDFState();
        List<UDFState> udfStateList = new ArrayList<UDFState>();
        udfState.setCount(count);
        udfStateList.add(udfState);
        return udfStateList;
    }

    @Override
    public void restoreState(List<UDFState> list) throws Exception
    {
        UDFState udfState = list.get(0);
        count = udfState.getCount();
    }

    @Override
    public void run(SourceContext<Tuple4<Long, String, String, Integer>> sourceContext) throws
Exception
    {
        Random random = new Random();
        while (isRunning) {
            for (int i = 0; i < 10000; i++) {
```



```
                sourceContext.collect(Tuple4.of(random.nextLong(), "hello" + count, alphabet, 1));
                count ++;
            }
            Thread.sleep(1000);
        }
    }

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

- Definition of a window with a checkpoint

  This code snippet is about a window operator and is used to calculate the number or tuples in a 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;

public class WindowStatisticWithChk implements WindowFunction<Tuple4<Long, String, String,
Integer>, Long, Tuple, TimeWindow>, ListCheckpointed<UDFState> {

    private long total = 0;

    @Override
    public List<UDFState> snapshotState(long l, long l1) throws Exception
    {
        UDFState udfState = new UDFState();
        List<UDFState> list = new ArrayList<UDFState>();
        udfState.setCount(total);
        list.add(udfState);
        return list;
    }

    @Override
    public void restoreState(List<UDFState> list) throws Exception
    {
        UDFState udfState = list.get(0);
        total = udfState.getCount();
    }

    @Override
    public void apply(Tuple tuple, TimeWindow timeWindow, Iterable<Tuple4<Long, String, String,
Integer>> iterable, Collector<Long> collector) throws Exception
    {
        long count = 0L;
        for (Tuple4<Long, String, String, Integer> tuple4 : iterable) {
            count ++;
        }
        total += count;
        collector.collect(total);
    }
}
```

- Application code

  This code snippet is about the definition of StreamGraph and detailed service implementation process. The processing time is used as time to trigger the window.


```
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.runtime.state.StateBackend;
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
   {
      String chkPath = ParameterTool.fromArgs(args).get("chkPath", "hdfs://hacluster/flink/
checkpoints/");
      StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

      env.setStateBackend((StateBackend) new FsStateBackend((chkPath)));
      env.enableCheckpointing(6000, CheckpointingMode.EXACTLY_ONCE);
      env.addSource(new SimpleSourceWithCheckPoint())
            .keyBy(0)
            .window(SlidingProcessingTimeWindows.of(Time.seconds(4), Time.seconds(1)))
            .apply(new WindowStatisticWithChk())
            .print();

      env.execute();
   }
}
```

5.3.3.3 Scala Sample Code of Flink Asynchronous Checkpoint

Sample Code

Assume that you want to collect data volume in a 4-second time window every other second and the status of operators must be strictly consistent.

- Formats of sent data

  ```
  case class SEvent(id: Long, name: String, info: String, count: Int)
  ```

- Snapshot data

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

  ```
     // User-defined status
  class UDFStateScala extends Serializable{
    private var count = 0L

    // Set the user-defined status.
    def setState(s: Long) = count = s

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

- Data source with checkpoints

  The code snippet of a source operator pauses 1 second every time after sending 10,000 pieces of data. When a snapshot is created, the code saves the total number of sent data pieces in UDFState. When the snapshot is used for restoration, the number of sent data pieces saved 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

  case class SEvent(id: Long, name: String, info: String, count: Int)
  ```



```
// This class is a source operator with a checkpoint.
class SEventSourceWithChk extends RichSourceFunction[SEvent] with
ListCheckpointed[UDFStateScala]{
  private var count = 0L
  private var isRunning = true
  private val alphabet =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuv
wxyzABCDEFGHIJKLMNOPQRSTUVWZYX0987654321"

  // The logic of a source operator is to inject 10,000 tuples to the StreamGraph every second.
  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)
    }
  }

  // Invoked when a 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[UDFStateScala] = {
    val udfList: util.ArrayList[UDFStateScala] = new util.ArrayList[UDFStateScala]
    val udfState = new UDFStateScala
    udfState.setState(count)
    udfList.add(udfState)
    udfList
  }

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

- Definition of a window with a checkpoint

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

// This class is a window operator with a checkpoint.
class WindowStatisticWithChk extends WindowFunction[SEvent, Long, Tuple, TimeWindow] with
ListCheckpointed[UDFStateScala]{
  private var total = 0L

  // Define the window operator implementation logic 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)
  }
```



```
  // Create a snapshot for the user-defined status.
  override def snapshotState(l: Long, l1: Long): util.List[UDFStateScala] = {
    val udfList: util.ArrayList[UDFStateScala] = new util.ArrayList[UDFStateScala]
    val udfState = new UDFStateScala
    udfState.setState(total)
    udfList.add(udfState)
    udfList
  }

  // Restore the status from the user-defined snapshot.
  override def restoreState(list: util.List[UDFStateScala]): Unit = {
    val udfState = list.get(0)
    total = udfState.getState
  }
}
```

- Application code

  This code snippet is about the definition of StreamGraph and detailed service implementation process. The event time is used as time to trigger the window.

```
import org.apache.flink.runtime.state.filesystem.FsStateBackend
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

object FlinkEventTimeAPIChkMain {
  def main(args: Array[String]): Unit ={
    val chkPath = ParameterTool.fromArgs(args).get("chkPath", "hdfs://hacluster/flink/checkpoint/
checkpoint/")
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    env.setStateBackend(new FsStateBackend(chkPath))
    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] {
        // Set a watermark.
        override def getCurrentWatermark: Watermark = {
          new Watermark(System.currentTimeMillis())
        }
       // Add a timestamp to 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()
  }
}
```

5.3.4 Stream SQL Join Application

5.3.4.1 Development Plan of Flink Stream SQL Join

Assume that a Flink service 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, a Flink application is developed to achieve the following function: uses the username recorded in the message received by service 2 as a keyword to jointly query service data.

Data Planning

- The data of service 1 is stored in the Kafka component. Data is sent to and received from Kafka (user with Kafka permission required). For details about Kafka configuration, see **Data Planning**.
- Service 2 receives messages using the socket. You can run the **netcat** command to input the analog data source.
  - Run the Linux command **netcat -l -p <port>** 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 Guidelines

1. Start the Flink Kafka Producer application to send data to Kafka.
2. Start the Flink Kafka Consumer application to receive data from Kafka and create Table1. Ensure that topics of Kafka Consumer are consistent with that of Kafka Producer.
3. Read data from the socket and create Table2.
4. Use Flink SQL to jointly query Table1 and Table2 and print the result.

5.3.4.2 Flink Stream SQL Join Java Sample Code

Function Description

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

If you need to interconnect with Kafka in security mode before application development, **kafka-client-xx.x.x.jar** of MRS is required. You can obtain the JAR file in the MRS client directory.

Sample Code

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**.

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

```
// Kafka Producer code
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.functions.source.SourceFunction;
```



```
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer;

import java.util.Random;

public class WriteIntoKafka4SQLJoin {

    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.flink.example.sqljoin.WriteIntoKafka4SQLJoin" +
                " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21005");
        System.out.println("./bin/flink run --class
com.huawei.flink.example.sqljoin.WriteIntoKafka4SQLJoin" +
                " /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 parallelism.
        env.setParallelism(1);
        // Parse the running parameters.
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        // Construct a StreamGraph and write the data generated from self-defined sources to Kafka.
        DataStream<String> messageStream = env.addSource(new SimpleStringGenerator());
        FlinkKafkaProducer producer = new FlinkKafkaProducer<>(paraTool.get("topic"), new
SimpleStringSchema(), paraTool.getProperties());
        messageStream.addSink(producer);
        // Invoke execute to trigger the execution.
        env.execute();
    }

    // Customize the sources and generate a message every other 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;
        }
    }
}
```

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

```
import org.apache.calcite.interpreter.Row;
import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.api.java.tuple.Tuple3;
```



```
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.streaming.api.TimeCharacteristic;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;
import org.apache.flink.table.api.Table;
import org.apache.flink.table.api.TableEnvironment;
import org.apache.flink.table.api.java.StreamTableEnvironment;

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.flink.example.sqljoin.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.flink.example.sqljoin.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;
      }

      StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

      StreamTableEnvironment tableEnv = TableEnvironment.getTableEnvironment(env);

     // Process data 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 joint 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 a stream and print the output.
        tableEnv.toAppendStream(result, Row.class).print();

        env.execute();
    }
}
```

5.4 Commissioning a Flink Application

5.4.1 Compiling and Running a Flink Application

After application code development is complete, you are advised to upload it to the Linux client to run applications. The procedures for running applications developed using Scala or Java are the same on the Flink client.

NOTE

Flink applications of a YARN cluster can run only on Linux, but not on Windows.

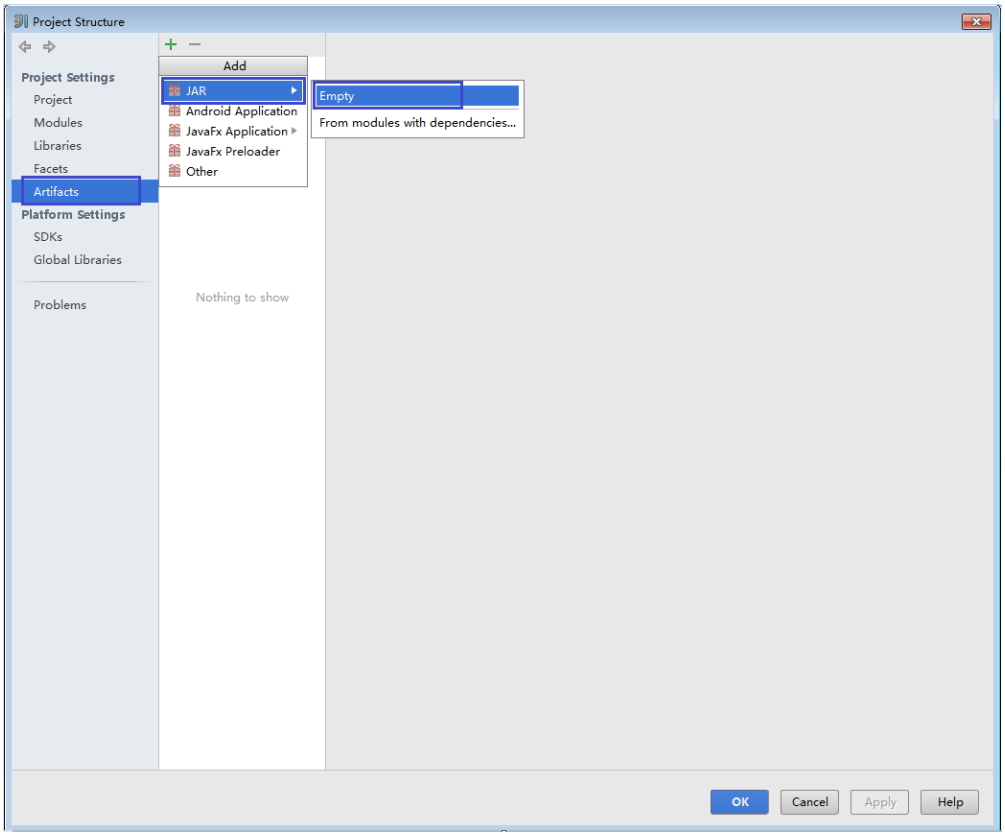
Procedure

Step 1 In IntelliJ IDEA, configure **Artifacts** of the project before generating a JAR file.

1. On the IDEA home page, choose **File** > **Project Structures...** to go to the **Project Structure** page.

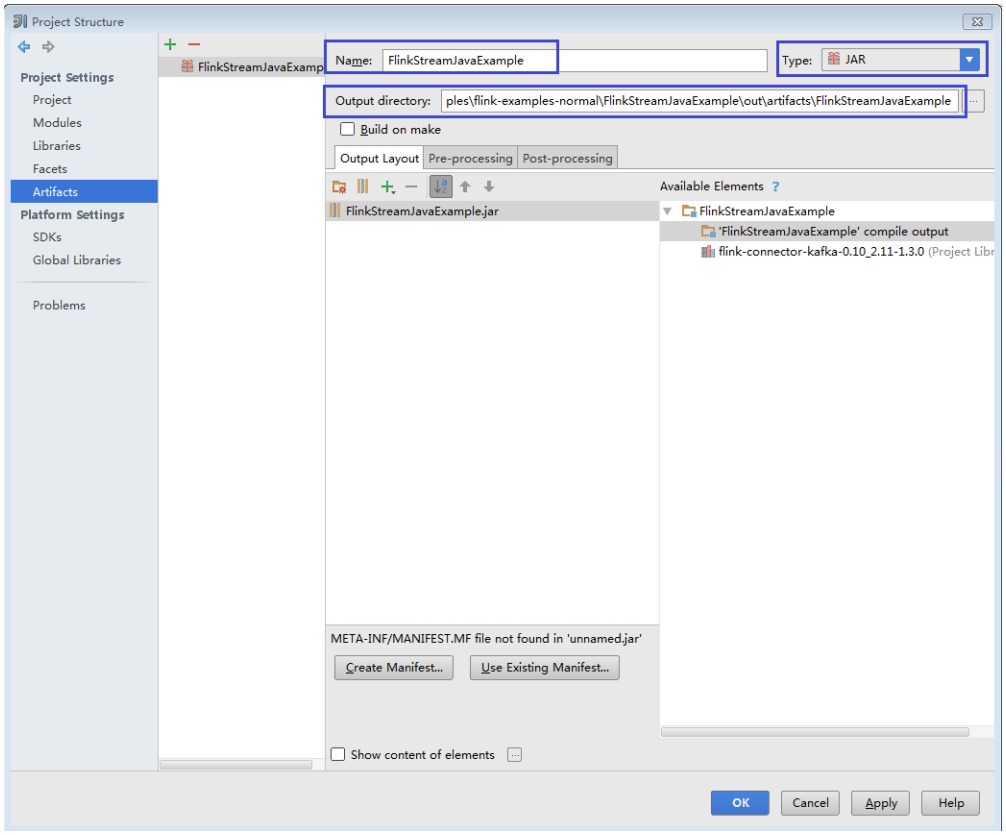
2. On the **Project Structure** page, select **Artifacts**, click **+**, and choose **JAR** > **Empty**.

Figure 5-32 Adding Artifacts



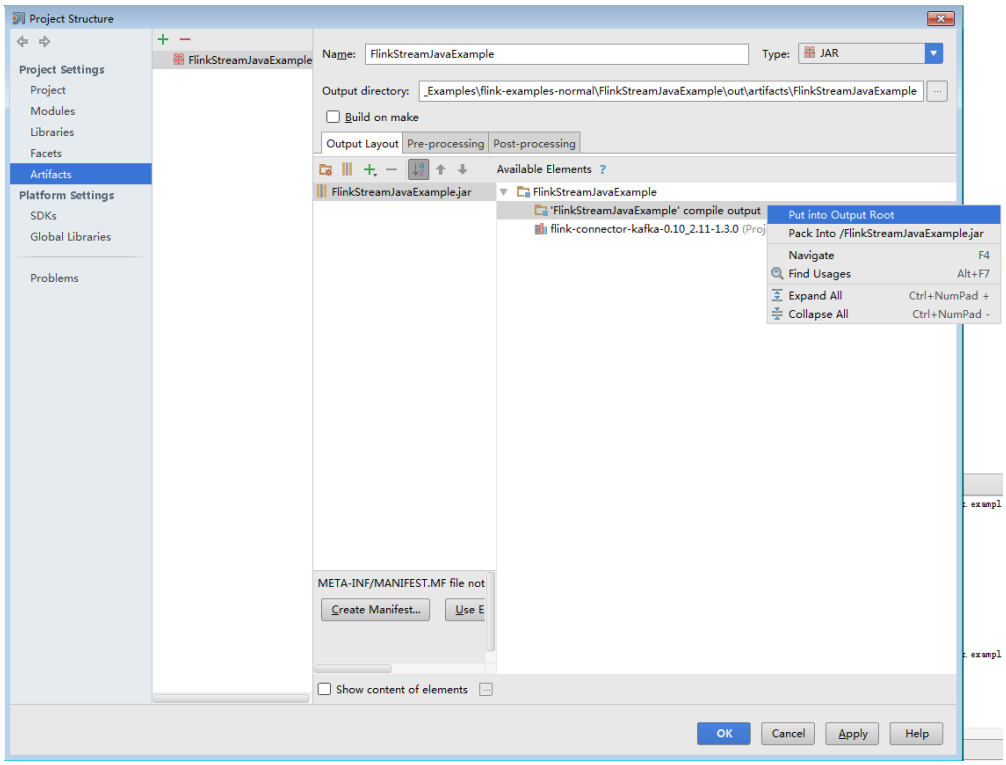
3. You can set the name, type, and output path of the JAR file based on the site requirements.

Figure 5-33 Setting basic information



4. Right-click **FlinkStreamJavaExample' compile output**, and choose **Put into Output Root**. Click **Apply**.

Figure 5-34 Put into Output Root

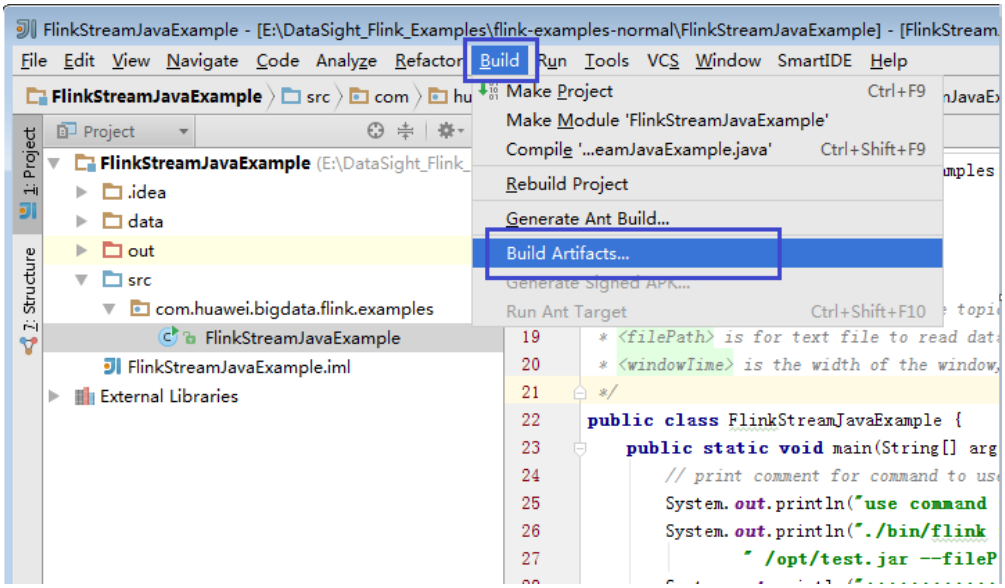


5. Click **OK** to complete the configuration.

Step 2 Generate a JAR file.

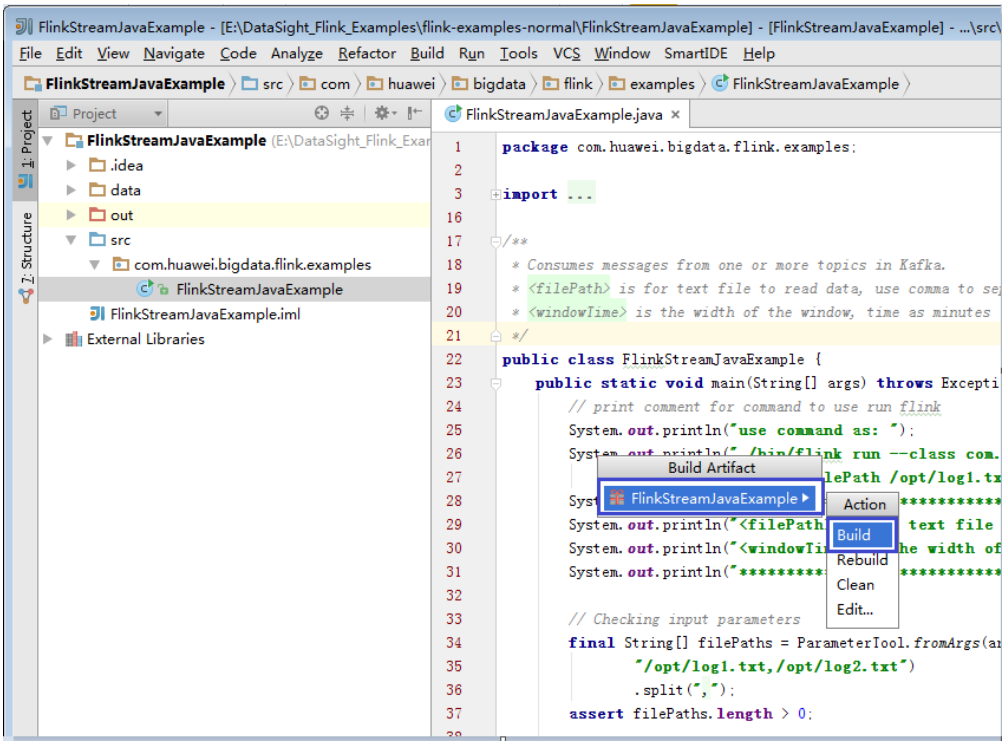
1. On the IDEA home page, choose **Build** > **Build Artifacts...**.

Figure 5-35 Build Artifacts



2. In the displayed menu, choose **FlinkStreamJavaExample** > **Build** to generate the JAR file.

Figure 5-36 Build



3. If information similar to the following is displayed in the event log, the JAR file has been successfully generated. You can obtain the JAR file from the path configured in **Step 1.3**.

```
21:25:43 Compilation completed successfully in 36 sec
```

Step 3 Copy the JAR file generated in **Step 2** (for example, **FlinkStreamJavaExample.jar**) to the Flink operating environment on Linux (that is, the Flink client), for example, **/opt/Flink_test**. Run the Flink application.

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 -n 3 -jm 1024 -tm 1024
```

NOTE

Do not restart the HDFS service or all DataNode instances during Flink job running. Otherwise, the job may fail and some temporary application data cannot be cleared.

- Running the DataStream sample application (Scala and Java)

  Open another window on the terminal. Go to the Flink client directory and invoke the **bin/flink run** script to run code. The following is an example.

```
bin/flink run --class com.huawei.flink.example.stream.FlinkStreamJavaExample /opt/Flink_test/flink-examples-1.0.jar --filePath /opt/log1.txt,/opt/log2.txt --windowTime 2

bin/flink run --class com.huawei.flink.example.stream.FlinkStreamScalaExample /opt/Flink_test/flink-examples-1.0.jar --filePath /opt/log1.txt,/opt/log2.txt --windowTime 2
```

  Table 5-8 Parameters

Parameter	Description
<filePath>	File path in the local file system. The **/opt/log1.txt** and **/opt/log2.txt** files must be stored on each node. Run the **chmod 755** *File name* command to grant the READ, WRITE, and EXECUTE permissions to users. The owner group user and other users have only the READ and EXECUTE permissions. The default value can be retained or changed.
<windowTime>	Duration of a time window. The unit is minute. The default value can be retained or changed.

- Running the sample application for producing and consuming data in Kafka (in Java or Scala)

  Run the following command to start the application to generate data:

```
bin/flink run --class com.huawei.flink.example.kafka.WriteIntoKafka /opt/Flink_test/flink-examples-1.0.jar <topic> <bootstrap.servers> [security.protocol] [sasl.kerberos.service.name] [kerberos.domain.name] [ssl.truststore.location] [ssl.truststore.password]
```

  Run the following commands to start the application to consume data. There can be security risks if a command contains the authentication password. You are advised to disable the command recording function (history) before running the command.

```
bin/flink run --class com.huawei.flink.example.kafka.ReadFromKafka /opt/Flink_test/flink-examples-1.0.jar <topic> <bootstrap.servers> [security.protocol] [sasl.kerberos.service.name] [kerberos.domain.name] [ssl.truststore.location] [ssl.truststore.password]
```


Table 5-9 Parameters

Parameter	Description	Mandatory
topic	Name of a Kafka topic	Yes
bootstrap.server	List of IP addresses or ports of broker clusters	Yes
security.protocol	The parameter can be set to protocols PLAINTEXT (optional), SASL_PLAINTEXT, SSL, and SASL_SSL, corresponding to ports 21005, 21007, 21008, and 21009 of the MRS Kafka cluster, respectively. – If the SASL is configured, the value of **sasl.kerberos.service.name** must be set to **kafka** and the configuration items related to **security.kerberos.login** in **conf/flink-conf.yaml** must be set. – 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, Kafka is in non-security mode.
kerberos.domain.name	Kafka domain name	No **NOTE** This parameter is mandatory when **security.protocol** is set to **SASL**.

The following examples use **ReadFromKafka** to show four types of commands:

```
bin/flink run --class com.huawei.flink.example.kafka.ReadFromKafka /opt/Flink_test/flink-examples-1.0.jar --topic topic1 --bootstrap.servers 10.96.101.32:21005
bin/flink run --class com.huawei.flink.example.kafka.ReadFromKafka /opt/Flink_test/flink-examples-1.0.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
bin/flink run --class com.huawei.flink.example.kafka.ReadFromKafka /opt/Flink_test/flink-examples-1.0.jar --topic topic1 --bootstrap.servers 10.96.101.32:21008 --security.protocol SSL --ssl.truststore.location /home/truststore.jks --ssl.truststore.password xxx
bin/flink run --class com.huawei.flink.example.kafka.ReadFromKafka /opt/Flink_test/flink-examples-1.0.jar --topic topic1 --bootstrap.servers 10.96.101.32:21009 --security.protocol SASL_SSL --sasl.kerberos.service.name kafka --kerberos.domain.name hadoop.hadoop.com --ssl.truststore.location /home/truststore.jks --ssl.truststore.password xxx
```

- Running the sample application of the asynchronous checkpoint mechanism (in Scala or Java)

  To diversify sample code, the processing time is used as a timestamp for DataStream in Java, and the event time is used as a timestamp for DataStream in Scala. The command reference is as follows:

  - Saving checkpoint snapshot information to HDFS
    - Java
      ```
      bin/flink run --class
      com.huawei.flink.example.checkpoint.FlinkProcessingTimeAPIChkMain /opt/Flink_test/flink-examples-1.0.jar --chkPath hdfs://hacluster/flink-checkpoint/
      ```
    - Scala
      ```
      bin/flink run --class com.huawei.flink.example.checkpoint.FlinkEventTimeAPIChkMain /opt/Flink_test/flink-examples-1.0.jar --chkPath hdfs://hacluster/flink-checkpoint/
      ```
  - Saving checkpoint snapshot information to a local file
    - Java
      ```
      bin/flink run --class
      com.huawei.flink.example.checkpoint.FlinkProcessingTimeAPIChkMain /opt/Flink_test/flink-examples-1.0.jar --chkPath file:///home/zzz/flink-checkpoint/
      ```
    - Scala
      ```
      bin/flink run --class com.huawei.flink.example.checkpoint.FlinkEventTimeAPIChkMain /opt/Flink_test/flink-examples-1.0.jar --chkPath file:///home/zzz/flink-checkpoint/
      ```

    NOTE

    - Path of the checkpoint source file: **flink/checkpoint/checkpoint/fd5f5b3d08628d83038a30302b611/chk-X/4f854bf4-ea54-4595-a9d9-9b9080779ffe**

      In the path, **flink/checkpoint/checkpoint** indicates the specified **root** directory.

      fd5f5b3d08628d83038a30302b611 indicates a second-level directory named after jobID.

      In **chk-X**, **X** indicates a checkpoint number, which is a third-level directory.

      4f854bf4-ea54-4595-a9d9-9b9080779ffe indicates a checkpoint source file.
    - In cluster mode, Flink checkpoint saves files to HDFS.

- Running the Stream SQL Join sample application

  a. Start the application to produce data in Kafka. For details about how to configure Kafka, see **Running the sample application for producing and consuming data in Kafka (in Java or Scala)**.
  ```
  bin/flink run --class com.huawei.flink.example.sqljoin.WriteIntoKafka4SQLJoin /opt/Flink_test/flink-examples-1.0.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.xxx:21005
  ```
  b. Run the **netcat** command on any node in the cluster to wait for an application connection.
  ```
  netcat -l -p 9000
  ```
  c. Start the application to receive socket data and perform a joint query.
  ```
  bin/flink run --class com.huawei.flink.example.sqljoin.SqlJoinWithSocket /opt/Flink_test/flink-examples-1.0.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.xxx:21005 --hostname xxx.xxx.xxx.xxx --port 9000
  ```

----End

5.4.2 Viewing the Running Result of a Flink Application

After a Flink application is run, you can view the running result or view the application running status on the Flink web UI.

Procedure

- **Viewing the running result of the Flink application**

  To view the running result, you need to view the **Stdout** log of the TaskManager on the Flink web UI.

  If the running result is exported to a file or another path specified by the Flink application, you can obtain the running result data from the specified file or path. The following uses checkpoints, pipelines, and the JOIN between configuration tables and streams as examples.

- **Viewing checkpoint results and files**
  - The results are stored in the **taskmanager.out** file of Flink. To view the results, log in to the Flink web UI, and click the **out** button under the **task manager** tag.
  - Two methods to view checkpoint files
    - If the checkpoint snapshot information is stored in HDFS, run the ***hdfs dfs -ls hdfs://hacluster/flink-checkpoint/*** command to view the checkpoint files.
    - If the checkpoint snapshot information is stored in a local file, log in to each node to view the checkpoint files.
- **Viewing the Stream SQL Join results**

  The results are stored in the **taskmanager.out** file of Flink. To view the results, log in to the Flink web UI, and click the **out** button under the **task manager** tag.

- **Viewing the running status of the Flink application on the Flink web UI**

  The Flink web UI contains the **Overview**, **Running Jobs**, **Completed Jobs**, **Task Managers**, **Job Manager**, and **Logout** parts.

  On the web UI of YARN, locate the Flink application. Click **ApplicationMaster** in the last column of the application information. The Flink web UI is displayed.

  To view the results printed during application execution, locate the TaskManager and view the corresponding **Stdout** tag log information.

- **Viewing Flink logs to check application running status**

  You can obtain Flink logs from the logs on either the Flink or YARN web UI.
  - On the Flink web UI, you can view logs of the TaskManagers and JobManager.
  - On the YARN web UI, you can view logs of the JobManager and GC.

    On the YARN web UI, locate the Flink application. Click the ID in the first column of the application information. On the displayed page, click **Logs** in the **Logs** column.

5.5 FAQs About Flink Application Development

5.5.1 Flink Savepoints CLI

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
// Stateful source (e.g. Kafka) with ID
.addSource(new StatefulSource())
.uid("source-id") // ID for the source operator
.shuffle()
 // Stateful mapper with ID
.map(new StateFulMapper())
.uid("mapper-id") // ID for the mapper
// Stateless printing sink
.print(); //Auto-generated ID
```

Resuming from Savepoints

If you do not specify the IDs manually, the system will automatically assign one ID to each operator. You can resume from a savepoint as long as the ID of the operator is not changed. ID generation depends on the user's application code and is sensitive to the application code structure. Therefore, it is highly recommended to specify an ID for every operator manually. Data generated by savepoints will be saved in the configured file system, for example, **FsStateBackend** or **RocksDBStateBackend**.

1. Trigger a savepoint.
   ```
   $ bin/flink savepoint <jobId> [targetDirectory]
   ```
   The command will trigger a savepoint for the job with ID: **jobId**. Furthermore, you can specify **targetDirectory** to store the savepoint. The directory must be accessible by JobManager. The **targetDirectory** parameter is optional. If you do not configure **targetDirectory**, the configured **state.savepoints.dir** in the configuration file is used to store the savepoint.

   You can set a default savepoint path using **state.savepoints.dir** in **flink-conf.yaml**.

   ```
   # Default savepoint target directory
   ```

   NOTE

   You are advised to set **targetDirectory** to an HDFS path. The following is an example.

   ```
   bin/flink savepoint  405af8c02cf6dc069a0f9b7a1f7be088 hdfs://savepoint
   ```

2. Cancel a job with a savepoint.
   ```
   $ bin/flink cancel -s [targetDirectory] jobId
   ```
   The command will atomically trigger a savepoint for the job with ID: jobid and cancel the job. Furthermore, you can specify **targetDirectory** to store the savepoint. The directory must be accessible by JobManager.

3. Restore jobs using the following methods.
   - Restoring jobs from savepoints
     ```
     $ bin/flink run -s savepointPath [runArgs]
     ```
     The command submits a job and sets the initial state of the job to the state specified by **savepointPath**.

NOTE

runArgs is a user-defined parameter, whose format and name vary depending on users.

- Allowing 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 are allowed to skip state that cannot be mapped to the new program via **--allowNonRestoredState (short: -n)** option.

4. Dispose savepoints.
   ```
   $ bin/flink savepoint -d savepointPath
   ```
   The 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 assign IDs to **b** and **c**, you need to manually define task chains using the **disableChaining()** API. The following provides an 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.

5.5.2 Flink Client CLI

For details about how to use the Flink CLI, visit **https://ci.apache.org/projects/flink/flink-docs-release-1.7/ops/cli.html**.

Common CLIs

Common Flink CLIs are as follows:

1. yarn-session.sh
   - You can run **yarn-session.sh** to start a resident 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 -n 3
     ```
   - Run the following command to obtain other parameters of **yarn-session.sh**:
     ```
     bin/yarn-session.sh -help
     ```
2. Flink
   - You can run the **flink** command to submit a Flink job to a resident Flink cluster or to run the job in single-node mode.
     - The following example command is used to submit a Flink job to a resident Flink cluster:
       ```
       bin/flink run examples/streaming/WindowJoin.jar
       ```


NOTE

Before running this command to submit a job, you need to use **yarn-session** to start the Flink cluster.

- The following example command is used to run a job in single-node mode:

```
bin/flink run -m yarn-cluster -yn 2 examples/streaming/WindowJoin.jar
```

NOTE

The **-m yarn-cluster** parameter is used for running the job in single-node mode. **-yn** indicates the number of TaskManagers.

– Run the following command to obtain other parameters of the **flink** command:

```
bin/flink --help
```

Precautions

- If **yarn-session.sh** uses **-z** to configure the specified ZooKeeper namespace, you need to use **-yid** to specify **applicationID** and use **-yz** to specify the ZooKeeper namespace when using **flink run**. The namespaces must the same.

  Example:

```
bin/yarn-session.sh -n 3 -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 -n 3
bin/flink run examples/streaming/WindowJoin.jar
```

- You can use **-yz** to specify a ZooKeeper namespace when running **flink run -m yarn-cluster** to start a cluster.
- You cannot start two or more clusters at the same time to share one namespace.
- 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 a savepoint.

5.5.3 Flink Performance Tuning Suggestions

Memory Configuration

Flink depends on in-memory computing. If memory is insufficient during computing, the Flink execution efficiency will be adversely affected. You can determine whether memory becomes a performance bottleneck by monitoring garbage collection (GC) and evaluating memory usage, and take performance optimization measures.

If full GC frequently occurs in the YARN container GC logs of monitoring node processes, optimize GC.

NOTE

GC configuration: Add the following parameters to the **env.java.opts** configuration item in the **conf/flink-conf.yaml** file on the client: "-Xloggc:<LOG_DIR>/gc.log -XX:+PrintGCDetails -XX:-OmitStackTraceInFastThrow -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=20 -XX:GCLogFileSize=20M". GC logs are added by default.

- GC optimization

  To optimize GC, adjust the ratio of the old generation to the young generation. In the **conf/flink-conf.yaml** configuration file on the client, add the **-XX:NewRatio** parameter to the **env.java.opts** configuration item. For example, **-XX:NewRatio=2** indicates that the ratio of the old generation to the young generation 2:1, the new generation occupies 1/3 of the entire heap space, and the old generation occupies 2/3.

- When developing Flink applications, optimize the data partitioning or grouping of DataStream.
  - If partitioning causes data skew, partitioning needs to be optimized.
  - Avoid unparallel operations, because some operations on DataStream, for example, WindowAll, cause parallelism failure.
  - Do not use a string for **keyBy**.

Configuring DOP

The degree of parallelism (DOP) indicates the number of tasks to be executed concurrently. It determines the number of data blocks after splitting. Adjust the DOP to maximize the number of tasks, the volume of data processed in each task, and the data processing capabilities the machines.

Query CPU and memory usage. If data and tasks are not evenly distributed among nodes, increase the DOP. Increasing the DOP makes full use of computing capabilities of machines in the cluster.

You can specify and adjust the DOP at one of the following levels (the priorities of which are in descending order) based on the actual memory, CPU, data, and application logic conditions.

- Operator

  Invoke the **setParallelism()** method to specify the DOP of an operator, data source, and data sink. Example:

```
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
DataStream<String> text = [...]
DataStream<Tuple2<String, Integer>> wordCounts = text
    .flatMap(new LineSplitter())
    .keyBy(0)
    .timeWindow(Time.seconds(5))
    .sum(1).setParallelism(5);
wordCounts.print();
env.execute("Word Count Example");
```

- Execution environment

  A Flink program runs in the execution environment. In the execution environment, a default DOP is defined for all executed operators, data sources, and data sinks.

  You can specify the default DOP by invoking the **setParallelism()** method. Example:


```
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.setParallelism(3);
DataStream<String> text = [...]
DataStream<Tuple2<String, Integer>> wordCounts = [...]
wordCounts.print();
env.execute("Word Count Example");
```

- Client

  You can specify the DOP when submitting jobs to Flink on the client. For a CLI client, you can specify the DOP using the **-p** parameter. Example:

  ```
  ./bin/flink run -p 10 ../examples/*WordCount-java*.jar
  ```

- System

  At the system level, you can modify **parallelism.default** in the **flink-conf.yaml** file in the **conf** directory of the Flink client to specify the default DOP for all execution environments.

Configuring Process Parameters

In Flink on YARN mode, there are two processes: JobManager and TaskManager. JobManagers and TaskManagers shoulder major responsibilities during task scheduling and running.

Parameter configurations of JobManagers and TaskManagers significantly affect the execution performance of Flink applications. You can perform the following operations to optimize Flink cluster performance.

Step 1 Configure JobManager memory.

JobManagers are responsible for task scheduling and message communications between the TaskManager and ResourceManager. JobManager memory needs to be added as tasks and the DOP increase.

You can set proper JobManager memory based on the number of tasks.

- When running the **yarn-session** command, add the **-jm MEM** parameter to configure memory.
- When running the **yarn-cluster** command, add the **-yjm MEM** parameter to configure memory.

Step 2 Configure the number of TaskManagers.

Each core of a TaskManager can process a task at the same time. Increasing the number of TaskManager has the same effect as increasing the DOP. Therefore, you can increase the number of TaskManagers to improve efficiency when there are sufficient resources.

- When running the **yarn-session** command, add the **-n NUM** parameter to configure the number of TaskManagers.
- When running the **yarn-cluster** command, add the **-yn NUM** parameter to configure the number of TaskManagers.

Step 3 Configure the number of TaskManager slots.

Multiple cores of a TaskManager can process multiple tasks at the same time. This has the same effect as increasing the DOP. However, the number of cores and the memory must be balanced, because all cores share the memory of the TaskManager.

- When running the **yarn-session** command, add the **-s NUM** parameter to configure the number of slots.
- When running the **yarn-cluster** command, add the **-ys NUM** parameter to configure the number of slots.

Step 4 Configure TaskManager memory.

The memory of a TaskManager is used for task execution and communications. A large-size task requires more resources. In this case, you can increase memory.

- When running the **yarn-session** command, add the **-tm MEM** parameter to configure memory.
- When running the **yarn-cluster** command, add the **-ytm MEM** parameter to configure memory.

----End

Partitioning Design Methods

A proper partitioning design can optimize task splitting. Ensure even partitioning during programming to prevent data skew in tasks. Otherwise, long-time execution of a task will delay the whole task.

Partitioning methods are as follows:

- **Random partitioning**: Partitions elements randomly.
  ```
  dataStream.shuffle();
  ```
- **Rebalancing (Round-robin partitioning)**: Partitions elements round-robin, creating equal load per partition. This is useful for performance optimization in the presence of data skew.
  ```
  dataStream.rebalance();
  ```
- **Rescaling**: Partitions elements, round-robin, to a subset of downstream operations. This is useful if you want to have pipelines where you, for example, fan out from each parallel instance of a source to a subset of several mappers to distribute load but don't want the full rebalance that **rebalance()** would incur.
  ```
  dataStream.rescale();
  ```
- **Broadcasting**: Broadcasts elements to every partition.
  ```
  dataStream.broadcast();
  ```
- **Custom partitioning**: Uses a user-defined Partitioner to select the target task for each element. Custom partitioning allows users to partition data based on a certain feature to optimize task execution.

  The following is an example:

  ```
  // Use fromElements to construct a simple Tuple2 flow.
  DataStream<Tuple2<String, Integer>> dataStream = env.fromElements(Tuple2.of("hello",1),
  Tuple2.of("test",2), Tuple2.of("world",100));
  // Define a key value used for partitioning. The return value is the partition to which the key belongs. The value plus 1 is the ID of the corresponding subtask.
  Partitioner<Tuple2<String, Integer>> strPartitioner = new Partitioner<Tuple2<String, Integer>>() {
      @Override
      public int partition(Tuple2<String, Integer> key, int numPartitions) {
          return (key.f0.length() + key.f1) % numPartitions;
      }
  };
  // Indicates the key value for partitioning using Tuple2.
  dataStream.partitionCustom(strPartitioner, new KeySelector<Tuple2<String, Integer>, Tuple2<String, Integer>>() {
  ```



```
    @Override
    public Tuple2<String, Integer> getKey(Tuple2<String, Integer> value) throws Exception {
        return value;
    }
}).print();
```

Configuring the Netty Network Communication

Flink communications depend on a Netty network. Netty network configuration is critical to Flink application execution, because the network performance determines data exchange speed and task execution efficiency.

The following parameters allow for advanced tuning in the **conf/flink-conf.yaml** configuration file on the client. The default values are sufficient. Exercise caution when changing the default values, preventing performance deterioration.

- **taskmanager.network.netty.num-arenas**: Number of Netty arenas. The default value is the value of **taskmanager.numberOfTaskSlots**.
- **taskmanager.network.netty.server.numThreads** and **taskmanager.network.netty.client.numThreads**: Number of Netty server and client threads, respectively. The default values are the value of **taskmanager.numberOfTaskSlots**.
- **taskmanager.network.netty.client.connectTimeoutSec**: Netty client connection timeout. The default value is **120s**.
- **taskmanager.network.netty.sendReceiveBufferSize**: Netty send and receive buffer size. This defaults to the system buffer size (**cat /proc/sys/net/ipv4/tcp_[rw]mem**) and is 4 MB in modern Linux.
- **taskmanager.network.netty.transport**: Netty transport type, either **nio** or **epoll**. The default value is **nio**.

Experience Summary

Avoiding Data Skew

If data skew occurs (certain data volume is extremely large), the execution time of tasks is inconsistent even though no GC is performed.

- Redefine the keys. Use keys of smaller granularity to optimize task sizes.
- Modify the DOP.
- Call the rebalance operation to balance data partitions.

Setting Buffer Timeout

- During the execution of tasks, data is exchanged through network. You can configure the **setBufferTimeout** parameter to specify a buffer timeout interval for data exchanging among different servers.
- If **setBufferTimeout** is set to **-1**, the refreshing operation is performed when the buffer is full, maximizing the throughput. If **setBufferTimeout** is set to **0**, the refreshing operation is performed each time data is received, minimizing the delay. If **setBufferTimeout** is set to a value greater than **0**, the refreshing operation is performed after the butter times out.

  The following is an example.

```
env.setBufferTimeout(timeoutMillis);
env.generateSequence(1,10).map(new MyMapper()).setBufferTimeout(timeoutMillis);
```


5.5.4 Savepoints FAQs

1. Should I assign IDs to all operators in my job?

   As a rule of thumb, yes. Strictly speaking, it is sufficient to only assign IDs via the **uid** method to the stateful operators in your job. The savepoint only contains state for these operators and stateless operators are not part of the savepoint.

   In practice, it is advised to assign IDs to all operators, because some of Flink's built-in operators like the Window operator are also stateful and it is not obvious which built-in operators are actually stateful and which are not. If you are absolutely certain that an operator is stateless, you can skip the **uid** method.

2. What happens if I add a new operator that requires state to my job?

   When you add a new operator to your job, it will be initialized without any state. Therefore, it is stateless and starts running from 0.

3. What happens if I delete an operator that has state from my job?

   By default, a savepoint restore will try to match all state back to the restored job. If you restore from a savepoint that contains state for an operator that has been deleted, this will therefore fail.

   You can allow non restored state by setting the **--allowNonRestoredState (short: -n)** with the run command:

   ```
   $ bin/flink run -s savepointPath -n [runArgs]
   ```

4. What happens if I reorder stateful operators in my job?
   - If you assigned IDs to these operators, they will be restored as usual.
   - If you did not assign IDs, the auto generated IDs of the stateful operators will most likely change after the reordering. This would result in you not being able to restore from a previous savepoint.

5. What happens if I add or delete or reorder operators that have no state in my job?
   - If you assigned IDs to your stateful operators, the stateless operators will not influence the savepoint restore.
   - If you did not assign IDs, the auto generated IDs of the stateful operators will most likely change after the reordering. This would result in you not being able to restore from a previous savepoint.

6. What happens when I change the parallelism of my program when restoring?

   If the savepoint was triggered with Flink 1.2.0 or later and using no deprecated state API like **checkpointed**, you can simply restore the program from a savepoint and specify a new parallelism. Otherwise, the restore will fail.

5.5.5 What Should I Do If Running a Checkpoint Is Slow When RocksDBStateBackend is Set for the Checkpoint and a Large Amount of Data Exists?

Issue

What should I do if running a checkpoint is slow when RocksDBStateBackend is set for the checkpoint and a large amount of data exists?

Possible Causes

Customized windows are used and the window state is **ListState**. There are many values under the same key. The **merge()** operation of **RocksDB** is used every time when a new value is added. When calculation is triggered, all values under the key are read.

- The RocksDB mode is **merge()->merge()....->merge()->read()**, which is time-consuming during data reading, as shown in **Figure 5-37**.
- When a source operator sends a large amount of data in an instant, the key values of all data are the same, which slows down window operator processing. As a result, the barriers are accumulated in the buffer and the completion of snapshot creation is delayed. The window operator fails to report a snapshot creation success to **CheckpointCoordinator** on time so that **CheckpointCoordinator** considers that the snapshot fails to be created. **Figure 5-38** shows a data flow.

Figure 5-37 Time monitoring information

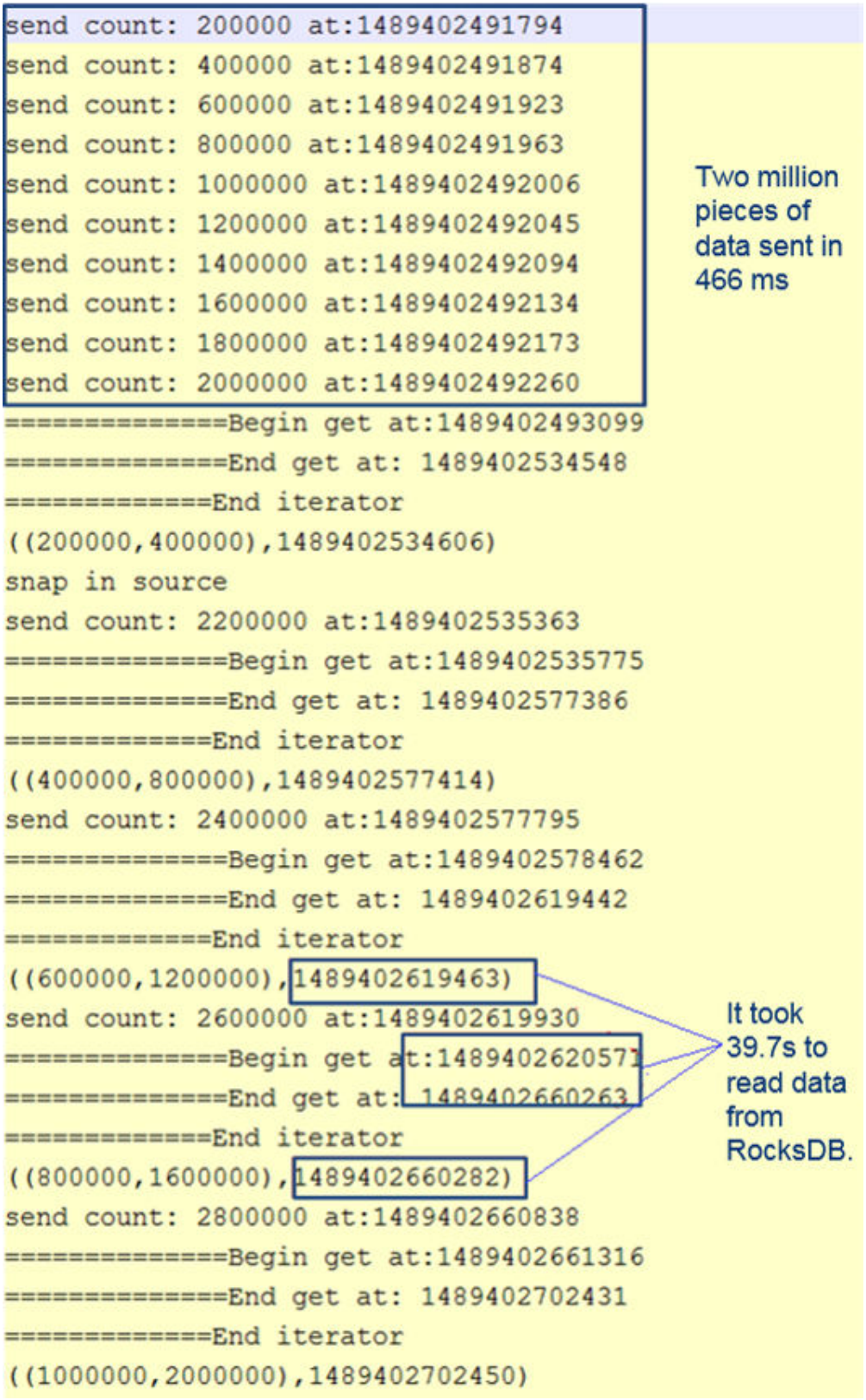
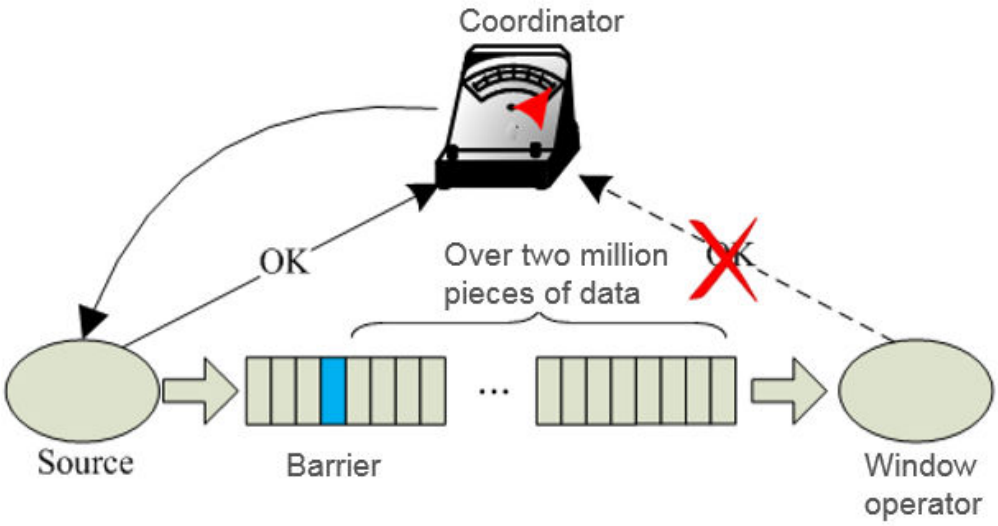


Figure 5-38 Data flow



Answer

Flink introduces the third-party software package RocksDB, whose defect causes the problem. You are advised to set **checkpoint** to **FsStateBackend**.

The following provides an example to show how to set **checkpoint** to **FsStateBackend** in the application code. The following provides an example:

```
env.setStateBackend(new FsStateBackend("hdfs://hacluster/flink-checkpoint/checkpoint/"));
```

5.5.6 What Should I Do If yarn-session Failed to Be Started When blob.storage.directory Is Set to /home?

Issue

When **blob.storage.directory** is set to **/home**, a user has no permission to create the **blobStore-UUID** file in **/home**, causing yarn-session start failure.

Answer

1. It is recommended that **blob.storage.directory** be set to **/tmp** or **/opt/Bigdata/tmp**.
2. If you set **blob.storage.directory** to a customized directory, you need to manually assign the owner permission on the directory to the user. The following uses user **admin** of MRS as an example.
   a. Modify the Flink client configuration file **conf/flink-conf.yaml** and configure **blob.storage.directory** to **/home/testdir/testdirdir/xxx**.
   b. Create the **/home/testdir** directory (level-1 directory is enough) and assign the directory to user **admin**.

```
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 in **/home/testdir/** is automatically created on each node when the Flink cluster is started.

c. Go to the client path and run **./bin/yarn-session.sh -n 3 -jm 2048 -tm 3072** to check whether **yarn-session** is normally started and the directory is 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
```

5.5.7 Why Does Non-static KafkaPartitioner Class Object Fail to Construct FlinkKafkaProducer010?

Issue

After the Flink kernel is upgraded to 1.3.0, an error is reported when Kafka calls the FlinkKafkaProducer010 that contains the non-static KafkaPartitioner class object as the parameter to construct a function.

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

The FlinkKafkaDelegatePartitioner class has been added to Flink 1.3.0 so that Flink can be compatible with APIs that use KafkaPartitioner, for example, FlinkKafkaProducer010 that contains the KafkaPartitioner class object, to construct functions.

The FlinkKafkaDelegatePartitioner class defines the member variable kafkaPartitioner.

```
private final KafkaPartitioner<T> kafkaPartitioner;
```

When Flink transfers KafkaPartitioner as a parameter to construct FlinkKafkaProducer010, the following stack is invoked:

```
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)
```

Use the KafkaPartitioner object to construct a FlinkKafkaDelegatePartitioner object, and then check whether the object is serializable. The ClosureCleaner::clean function is a static function. If the KafkaPartitioner object is non-static, the ClosureCleaner::clean function cannot access the non-static member variable kafkaPartitioner in the KafkaDelegatePartitioner class. As a result, an error is reported.

Either of the following methods can be used to solve the problem:

- Change the KafkaPartitioner class into a static class.
- Use the FlinkKafkaProducer010 that contains FlinkKafkaPartitioner as a parameter to construct functions. In this case, FlinkKafkaDelegatePartitioner will not be constructed and an error related to a member variable can be avoided.

5.5.8 When I Use the Newly-Created Flink User to Submit Tasks, Why Does the Task Submission Fail and a Message Indicating Insufficient Permission on ZooKeeper Directory Is Displayed?

Issue

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/application_1499222480199_0013
```

Answer

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** 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:

1. Check the configuration file **conf/flink-conf.yaml** on the client.
2. Modify the parameter **high-availability.zookeeper.path.root** to the corresponding ZooKeeper directory, for example, **/flink2**.
3. Submit the task again.

5.5.9 Why Can't I Access the Flink Web Page?

Issue

The Flink web page cannot be accessed through **http://JobManager IP:Port of the JobManager**.

Answer

The IP address of the computer you used has not been added to the whitelist of the Flink web page. To solve the problem, perform the following operation:

1. Check the configuration file **conf/flink-conf.yaml** on the client.
2. Check whether the value of the **jobmanager.web.ssl.enabled** parameter is **false**.
   - If the value is not **false**, change the value to **false**.

- If the value is **false**, perform the next step.

3. Check whether the IP address of the computer you used has been added to the **jobmanager.web.access-control-allow-origin** and **jobmanager.web.allow-access-address** parameters. If the IP address has not been added, add it to these two parameters. Example:

```
jobmanager.web.access-control-allow-origin: 192.168.252.35,192.168.24.216
jobmanager.web.allow-access-address: 192.168.252.35,192.168.24.216
```


6 HBase Development Guide

6.1 HBase Application Development Overview

6.1.1 Introduction to HBase Application Development

HBase

HBase is a column-based distributed storage system that features high reliability, performance, and scalability. HBase is designed to break through limitations of a relational database to process massive amounts of data.

Application scenarios of HBase have the following features:

- Massive data processing (higher than the TB or PB level)
- High throughput
- Highly efficient random read of massive data
- Excellent scalability
- Capable of concurrently processing structured and unstructured data
- Not all Atomicity, Consistency, Isolation, Durability (ACID) features supported by traditional relational databases are required.
- HBase tables have the following features:
  - Large: One table contains hundreds of millions of rows and millions of columns.
  - Column-based: 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 can be sparse.

API Types

You are advised to use Java to develop HBase applications, because HBase is developed based on Java and Java is concise, universal, and easy-to-understand.

HBase adopts the same Application Programming Interfaces (APIs) as those of Apache HBase. For details about the APIs, visit **http://hbase.apache.org/apidocs/index.html**.

Table 6-1 describes the functions that HBase can provide by invoking APIs.

Table 6-1 Functions provided by HBase APIs

Function	Description
Data CRUD function	Data creation, retrieve, update, and deletion
Advanced feature	Filter and coprocessor
Management function	Table and cluster management

6.1.2 Common Concepts of HBase Application Development

- **Filter**

  Filters provide powerful features to help users improve the table data processing efficiency of HBase. Users can use the filter predefined in HBase and customize a filter.

- **Coprocessor**

  Coprocessors enable users to perform region-level operations and provide functions similar to those of triggers in a relational database management system (RDBMS).

- **Client**

  Users can access a server from a client through Java APIs or HBase shell to read and write HBase tables. The HBase client in this document refers to the HBase client installation package downloaded from MRS Manager where the HBase service is installed. The HBase client installation package includes the sample code for accessing HBase through Java APIs.

6.1.3 HBase Application Development Process

This section describes how to use Java APIs to develop HBase applications.

Figure 6-1 and **Table 6-2** describe the phases in the development process.

Figure 6-1 HBase application development process

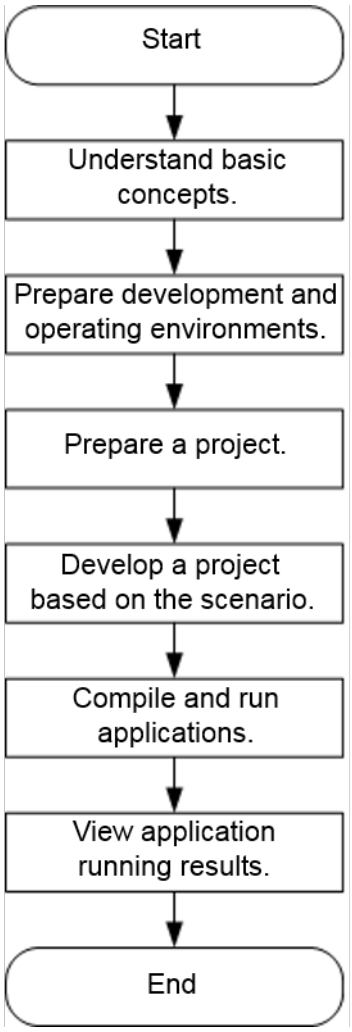


Table 6-2 HBase application development process details

Phase	Description	Reference
Understand basic concepts.	Before application development, learn basic concepts of HBase, understand the scenario requirements, and design tables.	**Common Concepts of HBase Application Development**
Prepare development and operating environments.	The Java language is recommended for HBase application development. You can use the Eclipse tool. The HBase operating environment is an HBase client. Install and configure the client according to the guide.	**Preparing a Local Application Development Environment**

Phase	Description	Reference
Prepare a project.	HBase provides sample projects for different scenarios. You can import a sample project to learn the application. You can also create an HBase project according to the guide.	**Importing and Configuring HBase Sample Projects**
Develop a project based on the scenario.	A Java sample project is provided, including creating a table, writing data into the table, and deleting the table.	**HBase Development Plan**
Compile and run applications.	You can compile the developed application and submit it for running.	**Application Commissioning**
View application running results.	Application running results are exported to a path you specify. You can also view the application running status on the UI.	**Viewing the HBase Application Commissioning Result**

6.2 Preparing an HBase Application Development Environment

6.2.1 Preparing a Local Application Development Environment

Table 6-3 describes the environment required for secondary development. You also need to prepare a Linux environment for verifying whether the application is running properly.

Table 6-3 Development environment

Item	Description
Operating system (OS)	Windows OS. Windows 7 or later is recommended.
JDK installation	Basic configurations of the development environment. JDK 1.8 or later is required.
Eclipse installation and configuration	It is a tool used to develop HBase applications.

Item	Description
Installing Apache Maven	It is a tool used to compile the sample project.
Network	The client must be interconnected with the HBase server on the network.

- Install Eclipse and JDK in the Windows development environment.

  You have installed JDK1.8 or later. You have used Eclipse supporting JDK 1.8 or later, with the JUnit plug-in installed.

  NOTE

  - If you use IBM JDK, ensure that the JDK configured in Eclipse is IBM JDK.
  - If you use Oracle JDK, ensure that the JDK configured in Eclipse is Oracle JDK.
  - Do not use the same workspace and the sample project in the same path for different Eclipse programs.
- Prepare a Linux environment for testing application running status.

Preparing a Running and Commissioning Environment

Step 1 On the ECS management console, apply for a new ECS for application development, running, and commissioning.

- The security group of the ECS must be the same as that of the master node in an MRS cluster.
- The ECS and the MRS cluster must be in the same VPC.
- The ECS network interface controller (NIC) and the MRS cluster must be in the same network segment.

Step 2 Apply for an EIP, bind it to the IP address of a new ECS, and configure an inbound or outbound rule for the security group.

Step 3 Download the client program. For details, see **Downloading an MRS Client**.

Step 4 Log in to the node where the downloaded client is located, and then install the client.

1. Run the following command to decompress the client package:

   cd /opt

   tar -xvf /opt/MRS_Services_Client.tar
2. Run the following command to verify the installation file package:

   sha256sum -c /opt/MRS_Services_ClientConfig.tar.sha256

   ```
   MRS_Services_ClientConfig.tar:OK
   ```
3. Run the following command to decompress the installation file package:

   tar -xvf /opt/MRS_Services_ClientConfig.tar
4. Run the following command to install the client to a specified directory (absolute path), for example, **/opt/client**. The directory is automatically created.


```
cd /opt/MRS_Services_ClientConfig
sh install.sh /opt/client
Components client installation is complete.
```

----End

6.2.2 Preparing an HBase Application Development User

The development user is used to run the sample project. The user must have HBase permissions to run the HBase sample project.

Prerequisites

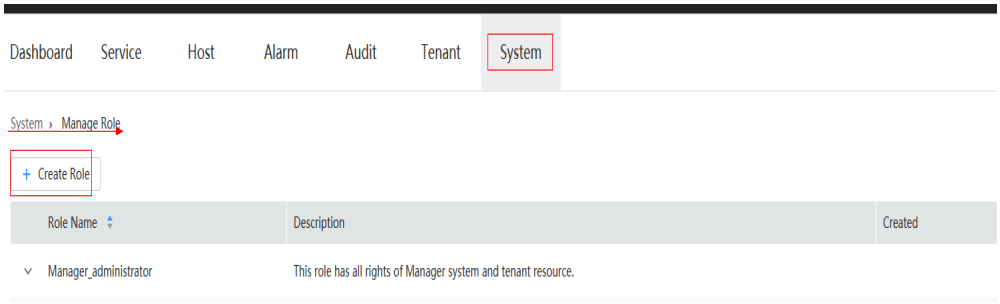
Kerberos authentication has been enabled for the MRS cluster. Skip this step if Kerberos authentication is not enabled for the cluster.

Procedure

Step 1 Log in to MRS Manager. For details, see **Logging in to MRS Manager**.

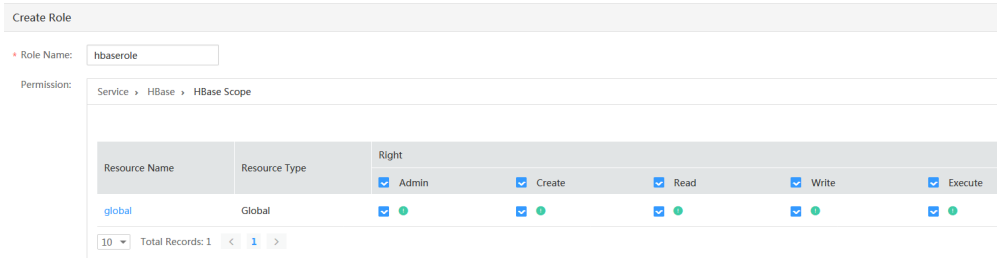
Step 2 Log in to MRS Manager and choose **System** > **Manage Role** > **Create Role**, as shown in **Figure 6-2**.

Figure 6-2 Creating a role



1. Enter a role name, for example, *hbaserole*.
2. Edit a role. Choose **HBase> HBase Scope** in **Right**. Select **Admin**, **Create**, **Read**, **Write**, and **Execute**, and click **OK**, as shown in **Figure 6-3**.

Figure 6-3 Editing a role

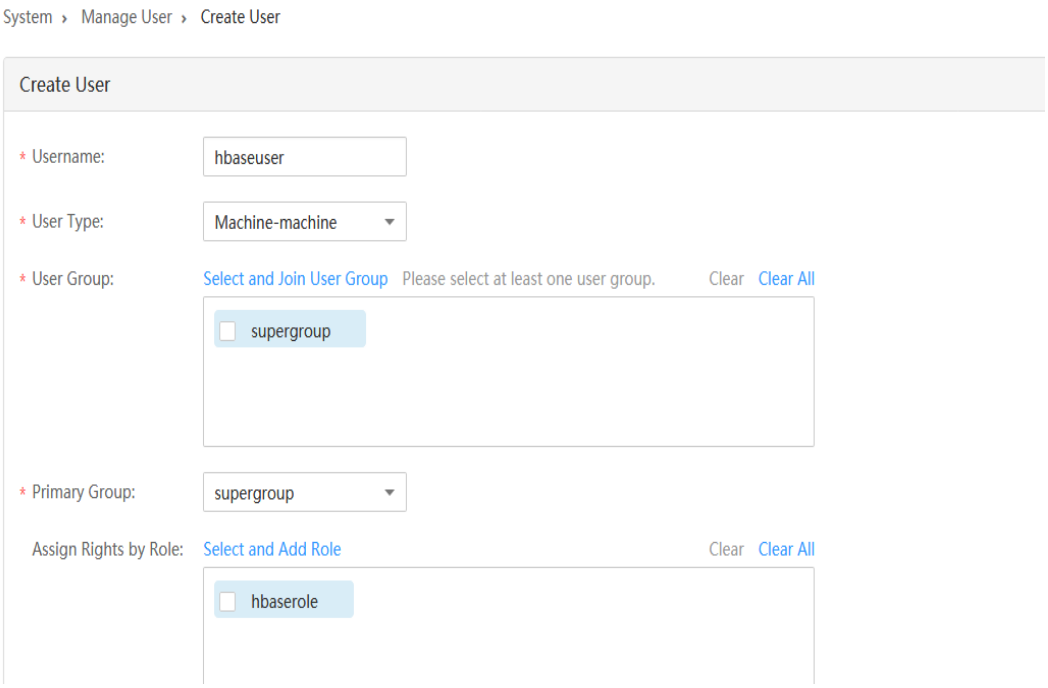


Step 3 Choose **System** > **Manage User** > **Create User** to create a user for the sample project.

Step 4 Enter a username, for example, *hbaseuser*. Set **User Type** to **Machine-machine**, and select **supergroup** in **User Group**. Set **Primary Group** to **supergroup**, select

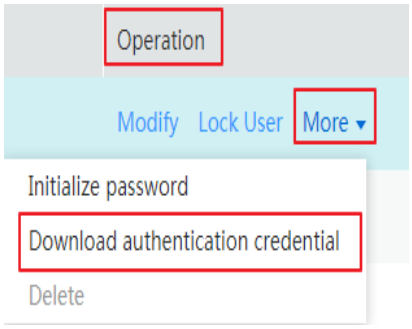
hbaserole in **Assign Rights by Role**, and click **OK**. **Figure 6-4** shows the parameter settings.

Figure 6-4 Creating a user



Step 5 On MRS Manager, choose **System > User Management**. On the displayed page, select **hbaseuser** from the Username drop-down list. In the **Operation** column on the right, choose **More > Download authentication credential**. Save the downloaded package and decompress it to obtain the **user.keytab** and **krb5.conf** files for security authentication in the sample project, as shown in **Figure 6-5**.

Figure 6-5 Downloading the authentication credential



----End

Related Information

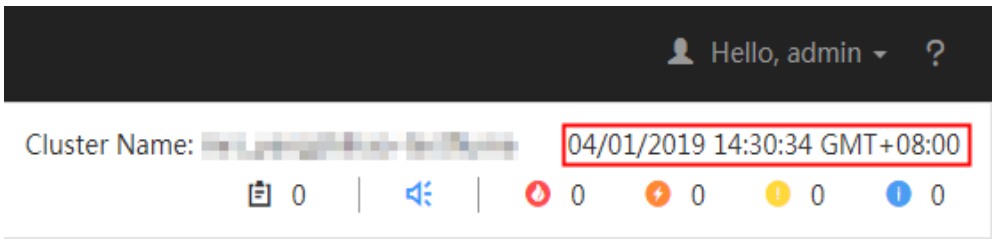
If you modify component parameter configurations, you need to download the client configuration file again and update the client in the running and commissioning environment.

6.2.3 Importing and Configuring HBase Sample Projects

Prerequisites

Ensure that the time difference between a local computer and the MRS cluster is less than 5 minutes. Time of the MRS cluster can be viewed in the upper right corner on the MRS Manager page.

Figure 6-6 Time of the MRS cluster



Procedure

Step 1 Obtain the HBase sample project by referring to **Obtaining the MRS Application Development Sample Project**.

Step 2 In the root directory of the HBase sample project, that is, the **pom.xml** directory of the HBase sample project, open the CLI and run the **mvn install** command to compile the project.

Step 3 In the cmd window opened in **Step 2**, run the **mvn eclipse:eclipse** command to create an Eclipse project.

Step 4 Configure the Eclipse development environment.

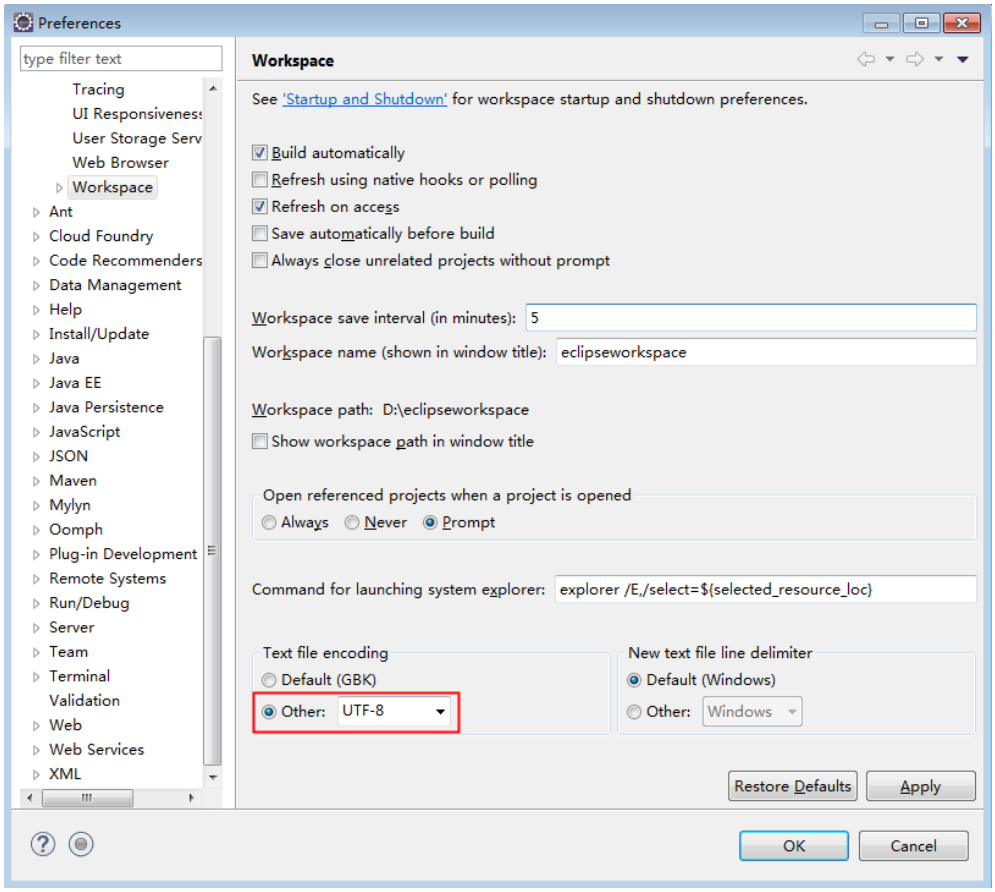
1. On the Eclipse menu bar, choose **Window** > **Preferences**.

The **Preferences** window is displayed.

2. Choose **General** > **Workspace** from the navigation tree. In the **Text file encoding** area, select **Other**, set the value to UTF-8, click **Apply**.

For details, see **Figure 6-7**.

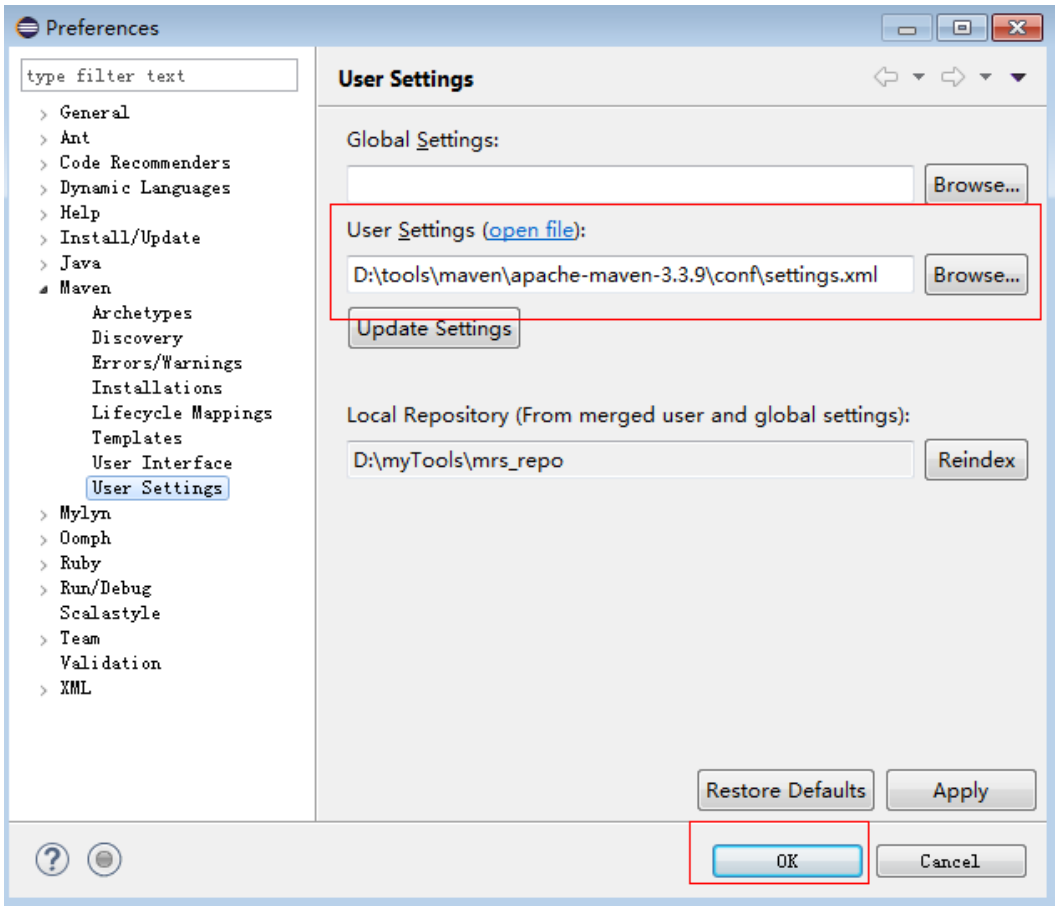
Figure 6-7 Setting the Eclipse encoding format



3. In the navigation pane on the left, choose **Maven** > **User Settings**. On the **User Settings** page, click **Browse** to import the **settings.xml** file of Maven, click **Apply**, and then click **OK**.

See **Figure 6-8**.

Figure 6-8 Setting the Maven Development Environment of the Eclipse



Step 5 In the application development environment, import the sample project to the Eclipse development environment.

1. Choose **File** > **Import** > **General** > **Existing Projects into Workspace** > **Next** > **Browse**.

   The **Browse Folder** dialog box is displayed.

2. Select the sample project folder, and click **Finish**.

----End

6.3 Developing an HBase Application

6.3.1 HBase Development Plan

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

Scenario Description

Develop an application to manage information about users who use service A in an enterprise. **Table 6-4** provides the user information. Procedures are as follows:

- Create a user information table.
- Add users' educational backgrounds and titles to the table.
- Query user names and addresses by user ID.
- Query information by user name.
- Query information about users whose age ranges from 20 to 29.
- Collect the number of users and their maximum, minimum, and average age.
- Deregister users and delete user data from the user information table.
- Delete the user information table after service A ends.

Table 6-4 User information

ID	Name	Gender	Age	Address
12005000201	Zhang San	Male	19	Shenzhen City, Guangdong Province
12005000202	Li Wanting	Female	23	Hangzhou City, Zhejiang Province
12005000203	Wang Ming	Male	26	Ningbo City, Zhejiang Province
12005000204	Li Gang	Male	18	Xiangyang City, Hubei Province
12005000205	Zhao Enru	Female	21	Shangrao City, Jiangxi Province
12005000206	Chen Long	Male	32	Zhuzhou City, Hunan Province
12005000207	Zhou Wei	Female	29	Nanyang City, Henan Province
12005000208	Yang Yiwen	Female	30	Wenzhou City, Zhejiang Province
12005000209	Xu Bing	Male	26	Weinan City, Shaanxi Province
12005000210	Xiao Kai	Male	25	Dalian City, Liaoning Province

Data Planning

Proper design of a table structure, RowKeys, and column names enable you to make full use of HBase advantages. In the sample project, a unique ID is used as a RowKey, and columns are stored in the **info** column family.

> **CAUTION**
>
> HBase tables are stored in the *Namespace*:*Table name* format. If the namespace is not specified when you create a table, the table will be stored in **default** by default. The HBase namespace is a system table namespace. Do not create service tables or read and write data in it.

Function Description

Determine functions to be developed based on the preceding scenario. **Table 6-5** describes functions to be developed.

Table 6-5 Functions to be developed in HBase

No.	Step	Code Implementation
1	Create a table based on the information in **Table 6-4**.	For details, see **Creating an HBase Table**.
2	Import user data.	For details, see **Inserting HBase Data**.
3	Add an educational background column family, and add educational backgrounds and titles to the user information table.	For details, see **Modifying an HBase Table**.
4	Query user names and addresses by user ID.	For details, see **Reading HBase Data Using the GET Command**.
5	Query information by user name.	For details, see **Using an HBase Filter**.
6	Deregister users and delete user data from the user information table.	For details, see **Deleting HBase Data**.
7	Delete the user information table after service A ends.	For details, see **Deleting an HBase Table**.

Key Design Principles

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.

6.3.2 Creating the Configuration Object

Function Description

HBase obtains configuration items by loading a configuration file, including user login information configuration items.

Sample Code

The following code snippets are in the **com.huawei.bigdata.hbase.examples** packet.

Invoke the **init ()** method under the **TestMain** class to initialize the Configuration object.

```
private static void init() throws IOException {
    // load hbase client info
    if(clientInfo == null) {
        clientInfo = new ClientInfo(CONF_DIR + HBASE_CLIENT_PROPERTIES);
        restServerInfo = clientInfo.getRestServerInfo();
    }
    // Default load from conf directory
    conf = HBaseConfiguration.create();

    conf.addResource(CONF_DIR + "core-site.xml");
    conf.addResource(CONF_DIR + "hdfs-site.xml");
    conf.addResource(CONF_DIR + "hbase-site.xml");

}
```

6.3.3 Creating a Connection Object

Function Description

HBase creates a Connection object using the **ConnectionFactory.createConnection(configuration)** method. The transferred parameter is the Configuration created in the previous 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 who can release resources by invoking **close()**.

Sample Code

The following code snippet is an example of creating a Connection:

```
private TableName tableName = null;
private Configuration conf = null;
private Connection conn = null;
public static final String TABLE_NAME = "hbase_sample_table";

public HBaseExample(Configuration conf) throws IOException {
  this.conf = conf;
  this.tableName = TableName.valueOf(TABLE_NAME);
  this.conn = ConnectionFactory.createConnection(conf);
}
```


NOTE

1. Example code involves many operations, such as creating, querying, and deleting tables. Only **testCreateTable** and **dropTable** are used as an example in this section. For details, see examples in the specific section.
2. The Admin object required by table creation is obtained from the Connection object.
3. Avoid invoking login code repeatedly.

6.3.4 Creating an HBase Table

Function Description

HBase allows you to create a table using the **createTable** method of the **org.apache.hadoop.hbase.client.Admin** object. You need to specify a table name and a column family name. You can create a table by using either of the following methods, but the latter one is recommended.

- Quickly create a table. A newly created table contains only one region, which will be automatically split into multiple new regions as data increases.
- Create a table using pre-assigned regions. You can pre-assign 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 a table consist of letters, digits, and underscores (_) but cannot contain any special characters.

Sample Code

The following code snippets are in the **testCreateTable** method in the **HBaseExample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
public void testCreateTable() {
    LOG.info("Entering testCreateTable: " + tableName);

    // Set the column family name to info.
    byte [] fam = Bytes.toBytes("info");
    ColumnFamilyDescriptor familyDescriptor = ColumnFamilyDescriptorBuilder.newBuilder(fam)
            // Set data encoding methods. HBase provides DIFF,FAST_DIFF,PREFIX
            // HBase 2.0 removed `PREFIX_TREE` Data Block Encoding from column families.
            .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 SANPPY
            .setCompressionType(Compression.Algorithm.SNAPPY)
            .build();
    TableDescriptor htd =
TableDescriptorBuilder.newBuilder(tableName).setColumnFamily(familyDescriptor).build();

    Admin admin = null;
    try {
        // Instantiate an Admin object.
        admin = conn.getAdmin();
        if (!admin.tableExists(tableName)) {
            LOG.info("Creating table...");
            admin.createTable(htd);
```



```
            LOG.info(admin.getClusterMetrics());
            LOG.info(admin.listNamespaceDescriptors());
            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.");
}
```

6.3.5 Deleting an HBase Table

Function Description

HBase allows you to delete a table using the **deleteTable** method of **org.apache.hadoop.hbase.client.Admin**.

Sample Code

The following code snippets are in the **dropTable** method in the **HBaseExample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
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);//Note [1]

            // Delete table.
            admin.deleteTable(tableName);
        }
        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

Note [1] A table can be deleted only when the table is disabled. Therefore, **deleteTable** is used together with **disableTable**, **enableTable**, **tableExists**, **isTableEnabled**, and **isTableDisabled**.

6.3.6 Modifying an HBase Table

Function Description

HBase allows you to modify table information using the **modifyTable** method of **org.apache.hadoop.hbase.client.Admin**.

Sample Code

The following code snippets are in the **testModifyTable** method in the **HBaseExample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
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 = TableDescriptorBuilder.newBuilder(tableName).build();

        // Check whether the column family is specified before modification.
        if (!htd.hasColumnFamily(familyName)) {
            // Create the column descriptor.
         ColumnFamilyDescriptor cfd = ColumnFamilyDescriptorBuilder.of(familyName);
         TableDescriptor td =
TableDescriptorBuilder.newBuilder(admin.getDescriptor(tableName)).setColumnFamily(cfd).build();
            // Disable the table to get the table offline before modifying
            // the table.
            admin.disableTable(tableName);//Note [1]
            // Submit a modifyTable request.
            admin.modifyTable(td);
            // 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.");
}
```


NOTE

Note [1] **modifyTable** takes effect only when a table is disabled.

6.3.7 Inserting HBase Data

Function Description

HBase is a column-based 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. HBase allows you to insert data (a row of data or data sets) using the **put** method of HTable.

Sample Code

The following code snippets are in the **testPut** method in the **HBaseExample** class of the **com.huawei.bigdata.hbase.examples** packet. The **com.huawei.bigdata.hbase.examples** package can be obtained from the sample project of the corresponding MRS version downloaded from **Obtaining the MRS Application Development Sample Project**.

```
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 = new Put(Bytes.toBytes("012005000201"));
        put.addColumn(familyName, qualifiers[0], Bytes.toBytes("Zhang San"));
        put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
        put.addColumn(familyName, qualifiers[2], Bytes.toBytes("19"));
        put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Shenzhen, Guangdong"));
        puts.add(put);

        put = new Put(Bytes.toBytes("012005000202"));
        put.addColumn(familyName, qualifiers[0], Bytes.toBytes("Li Wanting"));
        put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
        put.addColumn(familyName, qualifiers[2], Bytes.toBytes("23"));
        put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Shijiazhuang, Hebei"));
        puts.add(put);

        put = new Put(Bytes.toBytes("012005000203"));
        put.addColumn(familyName, qualifiers[0], Bytes.toBytes("Wang Ming"));
        put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
        put.addColumn(familyName, qualifiers[2], Bytes.toBytes("26"));
        put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Ningbo, Zhejiang"));
        puts.add(put);

        put = new Put(Bytes.toBytes("012005000204"));
        put.addColumn(familyName, qualifiers[0], Bytes.toBytes("Li Gang"));
        put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
        put.addColumn(familyName, qualifiers[2], Bytes.toBytes("18"));
        put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Xiangyang, Hubei"));
```



```
        puts.add(put);

        put = new Put(Bytes.toBytes("012005000205"));
        put.addColumn(familyName, qualifiers[0], Bytes.toBytes("Zhao Enru"));
        put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
        put.addColumn(familyName, qualifiers[2], Bytes.toBytes("21"));
        put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Shangrao, Jiangxi"));
        puts.add(put);

        put = new Put(Bytes.toBytes("012005000206"));
        put.addColumn(familyName, qualifiers[0], Bytes.toBytes("Chen Long"));
        put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
        put.addColumn(familyName, qualifiers[2], Bytes.toBytes("32"));
        put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Zhuzhou, Hunan"));
        puts.add(put);

        put = new Put(Bytes.toBytes("012005000207"));
        put.addColumn(familyName, qualifiers[0], Bytes.toBytes("Zhou Wei"));
        put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
        put.addColumn(familyName, qualifiers[2], Bytes.toBytes("29"));
        put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Nanyang, Henan"));
        puts.add(put);

        put = new Put(Bytes.toBytes("012005000208"));
        put.addColumn(familyName, qualifiers[0], Bytes.toBytes("Yang Yiwen"));
        put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
        put.addColumn(familyName, qualifiers[2], Bytes.toBytes("30"));
        put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Kaixian, Chongqing"));
        puts.add(put);

        put = new Put(Bytes.toBytes("012005000209"));
        put.addColumn(familyName, qualifiers[0], Bytes.toBytes("Xu Bing"));
        put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
        put.addColumn(familyName, qualifiers[2], Bytes.toBytes("26"));
        put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Weinan, Shaanxi"));
        puts.add(put);

        put = new Put(Bytes.toBytes("012005000210"));
        put.addColumn(familyName, qualifiers[0], Bytes.toBytes("Xiao Kai"));
        put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
        put.addColumn(familyName, qualifiers[2], Bytes.toBytes("25"));
        put.addColumn(familyName, qualifiers[3], Bytes.toBytes("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, exceptions will occur.

6.3.8 Deleting HBase Data

Function Description

HBase allows you to delete data (a row of data or data sets) using the **delete** method of a Table instance.

Sample Code

The following code snippets are in the **testDelete** method in the **HBaseExample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
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.");
}
```

6.3.9 Reading HBase Data Using the GET Command

Function Description

Before reading data from a table, create a table instance and 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.

Sample Code

The following code snippets are in the **testGet** method in the **HBaseExample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
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 Configuration 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.");
}
```

6.3.10 Reading HBase Data Using the Scan Command

Function Description

Before reading data from a table, instantiate the Table instance of the table, and then 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.

Sample Code

The following code snippets are in the **testScanData** method in the **HBaseExample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
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 Caching size.
        scan.setCaching(1000);//Note [1]

        // 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 can set **Batch** and **Caching**.
   - Batch

     Batch indicates the maximum number of records returned each time when the **next** API is invoked using Scan. This parameter is related to the number of columns read each time.
   - Caching

     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.

6.3.11 Using an HBase Filter

Function Description

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.

Sample Code

The following code snippets are in the **testFilterList** method in the **HBaseExample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
public void testFilterList() {
    LOG.info("Entering testFilterList.");

    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"));

        // Instantiate a FilterList object in which filters have "and"
        // relationship with each other.
        FilterList list = new FilterList(Operator.MUST_PASS_ALL);
        // Obtain data with age of greater than or equal to 20.
        list.addFilter(new SingleColumnValueFilter(Bytes.toBytes("info"), Bytes.toBytes("age"),
                CompareOp.GREATER_OR_EQUAL, Bytes.toBytes(new Long(20))));
        // Obtain data with age of less than or equal to 29.
        list.addFilter(new SingleColumnValueFilter(Bytes.toBytes("info"), Bytes.toBytes("age"),
                CompareOp.LESS_OR_EQUAL, Bytes.toBytes(new Long(29))));

        scan.setFilter(list);

        // 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("Filter list successfully.");
    } catch (IOException e) {
        LOG.error("Filter list 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 testFilterList.");
}
```


6.3.12 Adding an HBase Secondary Index

Function Description

You can use the methods provided by **org.apache.hadoop.hbase.hindex.client.HIndexAdmin** to manage HIndexes. This class provides methods of adding an index to an existing table.

You can add an index to a table by using either of the following methods based on whether you want to build index data when adding an index:

- addIndicesWithData()
- addIndices()

Sample Code

The following code snippets are in the **addIndicesExample** method in the **HIndexExample** class of the **com.huawei.bigdata.hbase.examples** packet.

addIndices (): Add an index to a table without data.

```
public void addIndicesExample() {
  LOG.info("Entering Adding a Hindex.");
  // Create index instance
  TableIndices tableIndices = new TableIndices();
  HIndexSpecification spec = new HIndexSpecification(indexNameToAdd);
  spec.addIndexColumn(new HColumnDescriptor("info"), "name", ValueType.STRING);
  tableIndices.addIndex(spec);
  Admin admin = null;
  HIndexAdmin iAdmin = null;
  try {
    admin = conn.getAdmin();
    iAdmin = HIndexClient.newHIndexAdmin(admin);
    // add index to the table
    iAdmin.addIndices(tableName, tableIndices);
    // Alternately, add the specified indices with data
    // iAdmin.addIndicesWithData(tableName, tableIndices);
    LOG.info("Successfully added indices to the table " + tableName);
  } catch (IOException e) {
    LOG.error("Add Indices failed for table " + tableName + "." + e);
  } finally {
    if (iAdmin != null) {
      try {
        // Close the HIndexAdmin object.
        iAdmin.close();
      } catch (IOException e) {
        LOG.error("Failed to close HIndexAdmin ", e);
      }
    }
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Failed to close admin ", e);
      }
    }
  }
  LOG.info("Exiting Adding a Hindex.");
}
```

The following code snippets are in the **addIndicesExampleWithData** method in the **HIndexExample** class of the **com.huawei.bigdata.hbase.examples** packet.

addIndicesWithData (): Add an index to a table with a large amount of data.

```
public void addIndicesExampleWithData() {
  LOG.info("Entering Adding a Hindex With Data.");
  // Create index instance
  TableIndices tableIndices = new TableIndices();
  HIndexSpecification spec = new HIndexSpecification(indexNameToAdd);
  spec.addIndexColumn(new HColumnDescriptor("info"), "age", ValueType.STRING);
  tableIndices.addIndex(spec);
  Admin admin = null;
  HIndexAdmin iAdmin = null;
  try {
    admin = conn.getAdmin();
    iAdmin = HIndexClient.newHIndexAdmin(admin);
    // add index to the table
    iAdmin.addIndicesWithData(tableName, tableIndices);
    // Alternately, add the specified indices with data
    // iAdmin.addIndicesWithData(tableName, tableIndices);
    LOG.info("Successfully added indices to the table " + tableName);
  } catch (IOException e) {
    LOG.error("Add Indices failed for table " + tableName + "." + e);
  } finally {
    if (iAdmin != null) {
      try {
        // Close the HIndexAdmin object.
        iAdmin.close();
      } catch (IOException e) {
        LOG.error("Failed to close HIndexAdmin ", e);
      }
    }
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Failed to close admin ", e);
      }
    }
  }
  LOG.info("Exiting Adding a Hindex With Data.");
}
```

6.3.13 Enabling or Disabling an HBase Secondary Index

Function Description

You can use the methods provided by **org.apache.hadoop.hbase.hindex.client.HIndexAdmin** to manage HIndexes. This class provides methods of enabling/disabling an existing index.

The HIndexAdmin provides the following APIs based on whether you want to enable or disable a table.

- disableIndices ()
- enableIndices ()

Sample Code

The following code snippets are in the **enableIndicesExample** method in the **HIndexExample** class of the **com.huawei.bigdata.hbase.examples** packet.

enableIndices (): Enable a specified index (the index status changes from INACTIVE to ACTIVE). This API can also be used to scan indexes.


```
public void enableIndicesExample() {
  LOG.info("Entering Enabling a Hindex.");
  List<String> indexNameList = new ArrayList<String>();
  indexNameList.add(indexNameToAdd);
  Admin admin = null;
  HIndexAdmin iAdmin = null;
  try {
    admin = conn.getAdmin();
    iAdmin = HIndexClient.newHIndexAdmin(admin);
    // Disable the specified indices
    iAdmin.enableIndices(tableName, indexNameList);
    // Alternately, disable the specified indices
    // iAdmin.disableIndices(tableName, indexNameList)
    LOG.info("Successfully enable indices " + indexNameList + " of the table " + tableName);
  } catch (IOException e) {
    LOG.error("Failed to enable indices " + indexNameList + " of the table " + tableName);
  } finally {
    if (iAdmin != null) {
      try {
        // Close the HIndexAdmin object.
        iAdmin.close();
      } catch (IOException e) {
        LOG.error("Failed to close HIndexAdmin ", e);
      }
    }
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Failed to close admin ", e);
      }
    }
  }
  LOG.info("Exiting Enabling a Hindex.");
}
```

The following code snippets are in the **disableIndicesExample** method in the **HIndexExample** class of the **com.huawei.bigdata.hbase.examples** packet.

disableIndices (): Disable a specified index (the index status changes from ACTIVE to INACTIVE). Therefore, index scanning becomes unavailable.

```
public void disableIndicesExample() {
  LOG.info("Entering Disabling a Hindex.");
  List<String> indexNameList = new ArrayList<~>();
  indexNameList.add(indexNameToAdd);
  Admin admin = null;
  HIndexAdmin iAdmin = null;
  try {
    admin = conn.getAdmin();
    iAdmin = HIndexClient.newHIndexAdmin(admin);
    // Disable the specified indices
    iAdmin.disableIndices(tableName, indexNameList);
    // Alternately, enable the specified indices
    // iAdmin.enableIndices(tableName, indexNameList);
    LOG.info("Successfully disabled indices " + indexNameList + " of the table " + tableName);
  } catch (IOException e) {
    LOG.error("Failed to disable indices " + indexNameList + " of the table " + tableName);
  } finally {
    if (iAdmin != null) {
      try {
        // Close the HIndexAdmin object.
        iAdmin.close();
      } catch (IOException e) {
        LOG.error("Failed to close HIndexAdmin ", e);
      }
    }
```



```
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Failed to close admin ", e);
      }
    }
  }
  LOG.info("Exiting Disabling a Hindex.");
}
```

6.3.14 Querying the HBase Secondary Index List

Function Description

You can use the methods provided by **org.apache.hadoop.hbase.hindex.client.HIndexAdmin** to manage HIndexes. This class provides methods of listing all indexes of a table.

HIndexAdmin provides the following API for listing indexes in a specified table:

- listIndices (): This API can be used to list all indexes of a specified table.

Sample Code

The following code snippets are in the **listIndicesIntable** method in the **HIndexExample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
public void listIndicesIntable() {
  LOG.info("Entering Listing Hindex.");
  Admin admin = null;
  HIndexAdmin iAdmin = null;
  try {
    admin = conn.getAdmin();
    iAdmin = HIndexClient.newHIndexAdmin(admin);
    // Retreive the list of indices and print it
    List<Pair<HIndexSpecification, IndexState>> indicesList = iAdmin.listIndices(tableName);
    LOG.info("indicesList:" + indicesList);
    LOG.info("Successfully listed indices for table " + tableName + ".");
  } catch (IOException e) {
    LOG.error("Failed to list indices for table " + tableName + "." + e);
  } finally {
    if (iAdmin != null) {
      try {
        // Close the HIndexAdmin object.
        iAdmin.close();
      } catch (IOException e) {
        LOG.error("Failed to close HIndexAdmin ", e);
      }
    }
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Failed to close admin ", e);
      }
    }
  }
  LOG.info("Exiting Listing Hindex.");
}
```


6.3.15 Using an HBase Secondary Index to Read Data

Function Description

In a user table with HIndexes, HBase uses a filter to query data.

Sample Code

The following code snippets are in the **scanDataByHIndex** method in the **HIndexExample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
public void scanDataByHIndex() {
  LOG.info("Entering HIndex-based Query.");
  Table table = null;
  ResultScanner rScanner = null;
  try {
    table = conn.getTable(tableName);
    // Create a filter for indexed column.
    SingleColumnValueFilter filter = new SingleColumnValueFilter(Bytes.toBytes("info"),
Bytes.toBytes("age"),
        CompareOp.GREATER_OR_EQUAL, Bytes.toBytes("26"));
    filter.setFilterIfMissing(true);

    Scan scan = new Scan();
    scan.setFilter(filter);
    rScanner = table.getScanner(scan);

    // Scan the data
    LOG.info("Scan data using indices..");
    for (Result result : rScanner) {
      LOG.info("Scanned row is:");
      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("Successfully scanned data using indices for table " + tableName + ".");
  } catch (IOException e) {
    LOG.error("Failed to scan data using indices for table " + tableName + "." + e);
  } finally {
    if (rScanner != null) {
      rScanner.close();
    }
    if (table != null) {
      try {
        table.close();
      } catch (IOException e) {
        LOG.error("failed to close table, ", e);
      }
    }
  }
  LOG.info("Entering HIndex-based Query.");
}
```

6.3.16 Deleting an HBase Secondary Index

Function Description

You can use the methods provided by **org.apache.hadoop.hbase.hindex.client.HIndexAdmin** to manage HIndexes. This class provides methods of deleting all indexes from a table.

Based on whether the user wants to delete index data and indexes, this class provide two APIs to delete indexes.

- dropIndices()
- dropIndicesWithData()

Sample Code

The following code snippets are in the **dropIndicesExample** method in the **HIndexExample** class of the **com.huawei.bigdata.hbase.examples** packet.

dropIndices (): Delete the specified index from the specified table, excluding index data.

```
public void dropIndicesExample() {
  LOG.info("Entering Deleting a Hindex.");
  List<String> indexNameList = new ArrayList<String>();
  indexNameList.add(indexNameToAdd);
  Admin admin = null;
  HIndexAdmin iAdmin = null;
  try {
    admin = conn.getAdmin();
    iAdmin = HIndexClient.newHIndexAdmin(admin);
    // Drop the specified indices without dropping index data
    iAdmin.dropIndices(tableName, indexNameList);
    // Alternately, drop the specified indices with data
    // iAdmin.dropIndicesWithData(tableName, indexNameList);
    LOG.info("Successfully dropped indices " + indexNameList + " from the table " + tableName);
  } catch (IOException e) {
    LOG.error("Failed to drop indices " + indexNameList + " from the table " + tableName);
  } finally {
    if (iAdmin != null) {
      try {
        // Close the HIndexAdmin object.
        iAdmin.close();
      } catch (IOException e) {
        LOG.error("Failed to close HIndexAdmin ", e);
      }
    }
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Failed to close admin ", e);
      }
    }
  }
  LOG.info("Exiting Deleting a Hindex.");
}
```

The following code snippets are in the **dropIndicesExampleWithData** method in the **HIndexExample** class of the **com.huawei.bigdata.hbase.examples** packet.

dropIndicesWithData (): Delete a specified index from a specified table, including all related index data, from the user table.

```
public void dropIndicesExampleWithData() {
  LOG.info("Entering Deleting a Hindex With Data.");
  List<String> indexNameList = new ArrayList<String>();
  indexNameList.add(indexNameToAdd);
  Admin admin = null;
  HIndexAdmin iAdmin = null;
  try {
    admin = conn.getAdmin();
    iAdmin = HIndexClient.newHIndexAdmin(admin);
    // Drop the specified indices without dropping index data
    iAdmin.dropIndicesWithData(tableName, indexNameList);
```



```
    // Alternately, drop the specified indices with data
    // iAdmin.dropIndicesWithData(tableName, indexNameList);
    LOG.info("Successfully dropped indices " + indexNameList + " from the table " + tableName);
  } catch (IOException e) {
    LOG.error("Failed to drop indices " + indexNameList + " from the table " + tableName);
  } finally {
    if (iAdmin != null) {
      try {
        // Close the HIndexAdmin object.
        iAdmin.close();
      } catch (IOException e) {
        LOG.error("Failed to close HIndexAdmin ", e);
      }
    }
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Failed to close admin ", e);
      }
    }
  }
  LOG.info("Exiting Deleting a Hindex With Data.");
}
```

6.3.17 HBase Multi-Point Region Splitting

Function Description

You can perform multi-point splitting by using **org.apache.hadoop.hbase.client.HBaseAdmin**.

NOTE

The splitting operation takes effect on empty regions only.

You can pre-partition a table when you create the table or directly split some regions.

In this example, the multi-point splitting is performed on an HBase table by using multiSplit. The table will be split into four regions: "A~D", "D~F", "F~H", and "H~Z".

Sample Code

The following code snippets are in the **testMultiSplit** method in the **HBaseExample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
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 (HRegionInfo regionInfo : regionSet) {
                ((HBaseAdmin) admin).multiSplit(regionInfo.getRegionName(), sk);
            }
            LOG.info("MultiSplit successfully.");
        } catch (Exception e) {
            LOG.error("MultiSplit failed ", e);
        } 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.");
    }
```

6.3.18 Configuring HBase ACL Security Policies

Function Description

Access rights control is mature in relational databases. HBase provides a simple access rights control feature. This feature is simply implemented in read (R), write (W), creation (C), execution (X), and administration (A) operations. In common mode, this feature is supported only when HBase permission management is enabled.

The Access Control List (ACL) method is defined in the **org.apache.hadoop.hbase.security.access.AccessControlClient** tool class.

Sample Code

The following code snippets are in the **grantACL** method in the **HBaseExample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
public void grantACL() {
    LOG.info("Entering grantACL.");

    String user = "usertest";
    String permissions = "RW";

    String familyName = "info";
    String qualifierName = "name";

    Table mt = null;
    Admin hAdmin = null;
    try {
        // Create ACL Instance
        mt = conn.getTable(AccessControlLists.ACL_TABLE_NAME);

        Permission perm = new Permission(Bytes.toBytes(permissions));
```



```
            hAdmin = conn.getAdmin();
            HTableDescriptor ht = hAdmin.getTableDescriptor(tableName);

            // Judge whether the table exists
            if (hAdmin.tableExists(mt.getName())) {
                // Judge whether ColumnFamily exists
                if (ht.hasFamily(Bytes.toBytes(familyName))) {
                    // grant permission
                    AccessControlClient.grant(conn, tableName, user, Bytes.toBytes(familyName),
                            (qualifierName == null ? null : Bytes.toBytes(qualifierName)), perm.getActions());
                } else {
                    // grant permission
                    AccessControlClient.grant(conn, tableName, user, null, null, perm.getActions());
                }
            }
            LOG.info("Grant ACL successfully.");
        } catch (Throwable e) {
            LOG.error("Grant ACL failed ", e);
        } finally {
            if (mt != null) {
                try {
                    // Close
                    mt.close();
                } catch (IOException e) {
                    LOG.error("Close table failed ", e);
                }
            }

            if (hAdmin != null) {
                try {
                    // Close Admin Object
                    hAdmin.close();
                } catch (IOException e) {
                    LOG.error("Close admin failed ", e);
                }
            }
        }
        LOG.info("Exiting grantACL.");
    }
```

Shell command format:

```
Command line interface
# Grant permissions.
grant <user> <permissions>[ <table>[ <column family>[ <column qualifier> ] ] ]

# Cancel permission granting.
revoke <user> <permissions> [ <table> [ <column family> [ <column qualifier> ] ] ]

# Set a table owner.
alter <table> {owner => <user>}

# Display a permission list.
user_permission <table>  # displays existing permissions
```

Example:

```
grant 'user1', 'RWC'
grant 'user2', 'RW', 'tableA'
user_permission 'tableA'
```

6.4 Commissioning an HBase Application

6.4.1 Commissioning an HBase Application on Windows

6.4.1.1 Compiling and Running an HBase Application

You can run applications in the Windows development environment after application code is developed.

Procedure

Step 1 When you use REST APIs to operate HBase clusters on Windows, the JDK version must be jdk1.8.0_60 or later. Obtain the **cacerts** file of JDK from the cluster environment and copy the **/opt/Bigdata/jdk/jre/lib/security/cacerts** file to **C:\Program Files\Java\jdk1.8.0_60\jre\lib\security** in the JDK environment on Windows. (Skip this step if you do not use REST APIs to operate HBase clusters.)

Step 2 Configure a mapping between the cluster IP address and host name on Windows. Log in to the cluster background, run the **cat /etc/hosts** command, and copy the mapping between IP addresses and host names in the **hosts** file to **C:\Windows\System32\drivers\etc\hosts**. The host name is subject to the query result.

```
192.168.0.90 node-master1BedB.089d8c43-12d5-410c-b980-c2728a305be3.com
192.168.0.129 node-ana-corezLaR.089d8c43-12d5-410c-b980-c2728a305be3.com
```

NOTE

You can use either of the following method to access an MRS cluster to operate HBase on Windows.

- Apply for a Windows ECS to access the MRS cluster to operate HBase. Run the sample code after the development environment is installed. To apply for ECS to access the MRS cluster, perform the following steps:
  1. On the **Active Clusters** page, click the name of an existing cluster.

     On the cluster details page, record the **AZ**, **VPC**, **Cluster Manager IP Address** of the cluster, and **Default Security Group** of the Master node.
  2. On the ECS management console, create an ECS.

     The **AZ**, **VPC**, and **security group** of ECS must be the same as those of the cluster to be accessed.

     Select a Windows public image.

     For details about other configuration parameters, see **Elastic Cloud Server** > **Quick Start** > **Purchasing and Logging In to a Windows ECS**.
- Use the local host to access the MRS cluster to operate HBase. Bind an EIP to all HBase nodes in the MRS cluster. When configuring the mapping between the cluster IP address and host name on the local host (Windows host), replace the IP address with the EIP corresponding to the host name, change the IP addresses of the **kdc**, **admin_server**, **kpasswd_server**, **kdc_listen**, **kadmind_listen**, and **kpasswd_listen** parameters in the **krb5.conf** file. (Skip this step if the cluster with a single master does not have the last three parameters.) Ensure that the cluster corresponds to the EIP in **KrbServer**. (Skip this step if the common cluster does not have the Kerberos function enabled.) Then run the sample code. To bind an EIP, perform the following steps:
  1. On the VPC management console, apply for an EIP and bind it to ECS.

     For details, see **Virtual Private Cloud** > **User Guide** > **Elastic IP Address** > **Assigning an EIP and Binding It to an ECS**.
  2. Open security group rules for the MRS cluster.

     Add security group rules to the security groups of the Master and Core nodes in the cluster to enable the ECS to access the cluster. If the cluster is a security cluster, add UDP ports 21731 and 21732, TCP ports 21730, 21731, and 21732, RPC ports of HBase HMaster and RegionServer instances, and ZooKeeper service ports to the inbound rule of the security group. For details, see **Virtual Private Cloud** > **User Guide** > **Security** > **Security Group** > **Adding a Security Group Rule**.

Step 3 Modify running environment configuration.

Modify the following parameters in the **hbase-site.xml** file built in sample code based on actual requirements:

- **hbase.zookeeper.quorum**: hostname of the ZooKeeper instance. Use commas (,) to separate hostnames of multiple instances. This parameter is mandatory.
- **hbase.regionserver.kerberos.principal**: principal of the RegionServer, which must be the same as the principal of the Master. This parameter is mandatory only for a cluster with the Kerberos function enabled.
- **hbase.master.kerberos.principal**: principal of the HMaster, which must be the same as the principal of the RegionServer. This parameter is mandatory only for a cluster with the Kerberos function enabled.
- **hadoop.security.authentication**: Hadoop authentication mode. Set this parameter to **kerberos** only for a cluster with the Kerberos function enabled. Otherwise, this parameter does not need to be set.
- **hbase.security.authentication**: HBase authentication mode. Set this parameter to **kerberos** only for a cluster with the Kerberos function enabled. Otherwise, this parameter does not need to be set.

NOTE

You can log in to any Master node and obtain the values of the preceding parameters from the HBase client configuration (**/opt/client/HBase/hbase/conf**). For example, to obtain the value of **hbase.zookeeper.quorum**, log in to any Master node and run the following command:

grep "hbase.zookeeper.quorum" /opt/client/HBase/hbase/conf/* -R -A1

```
[root@node-master1BedB ~]# grep "hbase.zookeeper.quorum" /opt/client/HBase/hbase/conf/* -R -A1
/opt/client/HBase/hbase/conf/hbase-site.xml:<name>hbase.zookeeper.quorum</name>
/opt/client/HBase/hbase/conf/hbase-site.xml-<value>node-master1bedb.089d8c43-12d5-410c-b980-c2728a305be3.com</value>
```

Step 4 Modify the sample code.

1. Currently, there are three types of HBase APIs in the sample code: common API, HFS API (no longer supported in MRS 1.9.*x*), and RESTful API. When commissioning different APIs to operate HBase, you can comment out the invoking of other APIs. In this example, common APIs are used to operate HBase and the **main** method contains only the following code snippet.

```
public static void main(String[] args) {
  try {
    init();
    login();
  } catch (IOException e) {
      LOG.error("Failed to login because ", e);
      return;
  }
  // getDefaultConfiguration();
  conf = HBaseConfiguration.create();
  // test hbase API
  HBaseExample oneSample;
  try {
    oneSample = new HBaseExample(conf);
    oneSample.test();
  } catch (Exception e) {
    LOG.error("Failed to test HBase because ", e);
  }
  LOG.info("-----------finish HBase-------------------");
}
```

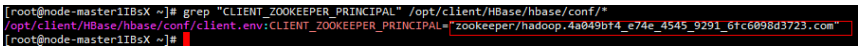

2. Change the value of **ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL** in the **com.huawei.bigdata.hbase.examples.TestMain** class of the sample project. Skip this step for a cluster with Kerberos authentication disabled.

```
private static final String ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL = "zookeeper/
hadoop.4a049bf4_e74e_4545_9291_6fc6098d3723.com";
```

NOTE

- During ZooKeeper authentication in a security cluster, a four-letter command is used to obtain the principal of the ZooKeeper server. If the ZooKeeper instance bears heavy loads or is unstable, the authentication fails because the principal of the ZooKeeper server fails to be obtained using the four-letter command. In this case, you need to use the client to transfer the value to the environment to avoid authentication failures.
- You can log in to any Master node and run the following command to obtain the value:

  grep "CLIENT_ZOOKEEPER_PRINCIPAL" /opt/client/HBase/hbase/conf/*



3. Modify the **hbaseclient.properties** file in the **src/main/resources** directory of the sample project. **userKeytab.path** and **krb5.conf.path** indicate the file addresses obtained in **Preparing an HBase Application Development User**. Skip this step for a cluster with Kerberos authentication disabled.

```
user.name=hbaseuser
userKeytabName=userKeytab.path
krb5ConfName=krb5.conf.path
```

4. If you need to modify **rest.server.info** in **hbaseclient.properties** when using REST APIs, make it correspond to **ip:port** (default port: 21309) of the rest server.

```
rest.server.info=10.10.10.10:21309
```

NOTE

1. Obtain the **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml** files from the Master node of the cluster and save the files to the **resources** directory of the sample project, that is, **src/main/resources**. The path for obtaining the files is **/opt/client/HBase/hbase/conf**. For MRS 1.9.2 or later, you only need to modify the configurations in the built-in **hbase-site.xml** file.
2. If you use ECS to access HBase, you can use the IP address of RESTServer directly. If you use the local host to access HBase, use the EIP bound to RESTServer as the IP address of RESTServer.
3. The HIndexExample sample project is supported only in MRS 1.9.2 or later. Pay attention to the current cluster version.
4. The HFSSample sample project is removed from MRS 1.9.*x*. Pay attention to the current cluster version.

Step 5 Run the sample project.

In the development environment (for example, Eclipse), right-click **TestMain.java** and choose **Run as** > **Java Application** from the shortcut menu to run the corresponding application project.

----End

6.4.1.2 Viewing the HBase Application Commissioning Result

Scenario

After HBase application running is complete, you can obtain the running status by viewing the running result or HBase logs.

Procedure

- If the application running is successful, the following information is displayed.

```
...
2020-01-09 10:43:49,338 INFO  [main] examples.HBaseExample: Entering dropTable.
2020-01-09 10:43:49,341 INFO  [main] client.HBaseAdmin: Started disable of hbase_sample_table
2020-01-09 10:43:50,080 INFO  [main] client.HBaseAdmin: Operation: DISABLE, Table Name:
default:hbase_sample_table, procId: 41 completed
2020-01-09 10:43:50,550 INFO  [main] client.HBaseAdmin: Operation: DELETE, Table Name:
default:hbase_sample_table, procId: 43 completed
2020-01-09 10:43:50,550 INFO  [main] examples.HBaseExample: Drop table successfully.
2020-01-09 10:43:50,550 INFO  [main] examples.HBaseExample: Exiting dropTable.
2020-01-09 10:43:50,550 INFO  [main] client.ConnectionImplementation: Closing master protocol:
MasterService
2020-01-09 10:43:50,556 INFO  [main] examples.TestMain: ----------finish to test HBase
API----------
```

  NOTE

  The following exception may occur when the sample code is running in the Windows OS, but it will 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 you can view more detailed information by changing the log level, such as DEBUG, INFO, WARN, ERROR, and FATAL. 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
```

6.4.2 Commissioning an HBase Application on Linux

6.4.2.1 Compiling and Running an HBase Application When a Client Is Installed

HBase applications can run in a Linux environment where an HBase client is installed. After application code development is complete, you can upload a JAR file to the Linux environment to run applications.

Prerequisites

- You have installed an HBase client.
- You have installed a JDK in the Linux environment. The version of the JDK must be consistent with that of the JDK used by Eclipse to export the JAR file.

- 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 locates. The host names and IP addresses must be mapped one by one.

Procedure

Step 1 Modify the sample code.

1. Currently, there are three types of HBase APIs in the sample code: common API, HFS API (no longer supported in MRS 1.9.*x*), and REST API. When commissioning different APIs to operate HBase, you can comment out the invoking of other APIs. In this example, common APIs are used to operate HBase and the **main** method contains only the following code snippet.

```
public static void main(String[] args) {
  try {
    init();
    login();
  } catch (IOException e) {
      LOG.error("Failed to login because ", e);
      return;
  }
  // getDefaultConfiguration();
  conf = HBaseConfiguration.create();
  // test hbase API
  HBaseExample oneSample;
  try {
    oneSample = new HBaseExample(conf);
    oneSample.test();
  } catch (Exception e) {
    LOG.error("Failed to test HBase because ", e);
  }
  LOG.info("-----------finish HBase--------------------");
}
```

2. When you call HFS APIs (no longer supported in MRS 1.9.*x*) and RESTful APIs, you need to copy the **inputfile.txt** and **hbaseclient.properties** files in the **src\main\resources** directory in the sample project to the **HBase/hbase/conf** directory of the client (for example, the client directory is **/opt/client**), and modify the **hbaseclient.properties** file. **userKeytabName** and **krb5ConfName** indicate the file addresses obtained in **Step 2**. If you need to modify **rest.server.info** when using REST APIs, make it correspond to **ip:port** (default port: 21309) of the rest server.

```
rest.server.info=10.10.10.10:21309
user.name=hbaseuser
userKeytabName=user.keytab
krb5ConfName=krb5.conf
```

NOTE

The HFSSample sample project is removed from MRS 1.9.*x*. Pay attention to the current cluster version.

Step 2 Run the **mvn package** command to generate a JAR file, for example, **hbase-examples-mrs-2.0.jar**. Obtain the JAR file from the target directory in the project directory, and upload it to the **/opt/client/HBase/hbase/lib** directory.

Step 3 Run the JAR file.

1. Before running the JAR file on the Linux client, run the following command to switch to the client directory as the user that is used for installation:

   cd $BIGDATA_CLIENT_HOME/HBase/hbase

NOTE

$BIGDATA_CLIENT_HOME indicates the client installation directory.

2. Run the following command:

   source $BIGDATA_CLIENT_HOME/bigdata_env

   NOTE

   After the multi-instance function is enabled, you also need to run the following command to switch to the client of the specified service instance before developing applications for the HBase service instance, for example, HBase2: **source /opt/client/HBase2/component_env**.

3. Copy the JAR package generated in **Step 2** and the **krb5.conf** and **user.keytab** files obtained in **Preparing an HBase Application Development User** to the **HBase/hbase/conf** directory in the client running environment, for example, **/opt/client/HBase/hbase/conf**. Create the **hbaseclient.properties** file in the **/opt/client/HBase/hbase/conf** directory if the file does not exist. In the file, **user.name** corresponds to the new user **hbaseuser**, and the values of **userKeytabName** and **krb5ConfName** correspond to the authentication-related file names obtained in **Preparing an HBase Application Development User** (skip this step if Kerberos authentication is not enabled for the cluster).

   ```
   user.name=hbaseuser
   userKeytabName=user.keytab
   krb5ConfName=krb5.conf
   ```

4. Run the following command to execute the JAR package:

   hbase com.huawei.bigdata.hbase.examples.TestMain /opt/client/HBase/hbase/conf

   com.huawei.bigdata.hbase.examples.TestMain is used as an example. Use the actual code instead.

   /opt/client/HBase/hbase/conf corresponds to the path of files such as **user.keytab** and **krb5.conf** mentioned above.

   NOTE

   For MRS 1.9.2 or later, run the **hbase com.huawei.bigdata.hbase.examples.TestMain /opt/client/HBase/hbase/conf** command.

----End

6.4.2.2 Compiling and Running an HBase Application When No Client Is Installed

HBase applications can run in a Linux environment where an HBase client is not installed. After application code development is complete, you can upload a JAR file to the Linux environment to run applications.

Prerequisites

- You have installed a JDK in the Linux environment. The version of the JDK must be consistent with that of the JDK used by Eclipse to export the JAR file.
- If the host where the Linux environment resides 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 Linux environment resides. The host names and IP addresses must be mapped one by one.

Procedure

Step 1 Modify the sample by following instructions in **Compiling and Running an HBase Application When a Client Is Installed**.

Step 2 Run the **mvn package** command to generate a JAR file, for example, **hbase-examples-2.0.jar**, and obtain it from the **target** directory in the project directory.

Step 3 Prepare the dependency JAR file and configuration file.

1. In the Linux environment, create a directory, for example, **/opt/test**, and create subdirectories **lib** and **conf**. Upload the JAR packages in the **/opt/client/HBase/hbase/lib** directory on any master node in the cluster and the JAR packages exported in **Step 2** to the **lib** directory in the new **/opt/test** directory in the Linux environment. Copy the **hbase-site.xml**, **hdfs-site.xml**, and **core-site.xml** files in the **/opt/client/HBase/hbase/conf** directory of any master node in the cluster to the **conf** directory in **/opt/test**.
2. Copy the **krb5.conf** and **user.keytab** files obtained in **Preparing an HBase Application Development User** to the **/opt/test/conf** directory, and create the **hbaseclient.properties** file. In the file, **user.name** corresponds to the new user **hbaseuser**, the **userKeytabName** and **krb5ConfName** paths correspond to the names of the authentication-related files obtained in **Preparing an HBase Application Development User** (skip this step if Kerberos authentication is not enabled for the cluster).

   ```
   user.name=hbaseuser
   userKeytabName=user.keytab
   krb5ConfName=krb5.conf
   ```

3. In the **/opt/test** root directory, create the **run.sh** script, modify the following content, and save the file.

   com.huawei.bigdata.hbase.examples.TestMain is used as an example. Use the actual code instead.

   ```
   #!/bin/sh
   BASEDIR=`pwd`
   cd ${BASEDIR}
   for file in ${BASEDIR}/lib/*.jar
   do
   i_cp=$i_cp:$file
   echo "$file"
   done
   if [ -d ${BASEDIR}/lib/client-facing-thirdparty ]; then
   for file in ${BASEDIR}/lib/client-facing-thirdparty/*.jar
   do
   i_cp=$i_cp:$file
   done
   fi
   java -cp ${BASEDIR}/conf:${i_cp} com.huawei.bigdata.hbase.examples.TestMain
   ```

Step 4 Go to **/opt/test** and run the following command to run the JAR file:

sh run.sh

----End

6.4.2.3 Viewing the HBase Application Commissioning Result

After HBase application running is complete, you can obtain the running status by viewing the running result or HBase logs.

If the application running is successful, the following information is displayed.

```
2018-01-17 19:44:28,068 INFO  [main] examples.HBaseExample: Entering dropTable.
2018-01-17 19:44:28,074 INFO  [main] client.HBaseAdmin: Started disable of hbase_sample_table
2018-01-17 19:44:30,310 INFO  [main] client.HBaseAdmin: Disabled hbase_sample_table
2018-01-17 19:44:31,727 INFO  [main] client.HBaseAdmin: Deleted hbase_sample_table
2018-01-17 19:44:31,727 INFO  [main] examples.HBaseExample: Drop table successfully.
2018-01-17 19:44:31,727 INFO  [main] examples.HBaseExample: Exiting dropTable.
2018-01-17 19:44:31,727 INFO  [main] client.ConnectionManager$HConnectionImplementation: Closing
master protocol: MasterService
2018-01-17 19:44:31,733 INFO  [main] client.ConnectionManager$HConnectionImplementation: Closing
zookeeper sessionid=0x13002d37b3933708
2018-01-17 19:44:31,736 INFO  [main-EventThread] zookeeper.ClientCnxn: EventThread shut down for
session: 0x13002d37b3933708
2018-01-17 19:44:31,737 INFO  [main] zookeeper.ZooKeeper: Session: 0x13002d37b3933708 closed
2018-01-17 19:44:31,750 INFO  [main] examples.TestMain: ----------finish HBase ----------
```

6.4.3 Commissioning the HBase Phoenix Sample Program

HBase allows users to access HBase services by invoking JDBC interfaces through Phoenix. Commission the HBase Phoenix sample program. By default, the HBase has been connected to the Phoenix service in the cluster. For details about the connection procedure, see **Configuring Phoenix for HBase**.

Running and Commissioning Applications on Windows

Step 1 For details about how to set up the Windows development environment and modify the public configuration of the sample program, see **Step 1** to **Step 3**.

Step 2 Modify the sample project.

1. Modify the **jaas.conf** file in the **src/main/resources** directory of the sample project, where the **keyTab** and **principal** parameter correspond to the path for storing user authentication credentials and the username, respectively. Skip this step for a cluster with Kerberos authentication disabled.

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="D:\\sample_project\\src\\hbase-examples\\hbase-java-examples\\src\\main\\resources\
\user.keytab"
  principal="hbaseuser"
  useTicketCache=false
  storeKey=true
  debug=true;
};
```

2. Modify the **hbaseclient.properties** file in the **src/main/resources** directory of the sample project. **user.name**, **userKeytabName**, and **krb5ConfName** indicate the username created in and the file names obtained in **Preparing an HBase Application Development User**. Skip this step for a cluster with Kerberos authentication disabled.

```
user.name=hbaseuser
userKeytabName=user.keytab
krb5ConfName=krb5.conf
#for phoenix
#confiugration for security cluster.
jaasConfName=jaas.conf
```


Step 3 In a development environment (for example, Eclipse), right-click **PhoenixExample** and choose **Run** > **PhoenixExample.main()** to run the application project.

NOTE

If error message "Message stream modified (41)" is displayed, the JDK version may be incorrect. Change the JDK version in the sample code to a version earlier than 8u_242 or delete the **renew_lifetime = 0m** configuration item from the **krb5.conf** configuration file.

Step 4 After the HBase application is run, you can check the application running status by viewing the running result.

If the following information is displayed, the application runs successfully.

```
2020-03-13 14:54:13,369 INFO  [main] client.HBaseAdmin: Operation: CREATE, Table Name: default:TEST, procId: 60 completed
2020-03-13 14:54:14,269 INFO  [main] examples.PhoenixExample: 1
2020-03-13 14:54:14,270 INFO  [main] examples.PhoenixExample: John
2020-03-13 14:54:14,270 INFO  [main] examples.PhoenixExample: 100000
2020-03-13 14:54:14,271 INFO  [main] examples.PhoenixExample: 1980-01-01
2020-03-13 14:54:14,464 INFO  [main] client.HBaseAdmin: Started disable of TEST
2020-03-13 14:54:15,199 INFO  [main] client.HBaseAdmin: Operation: DISABLE, Table Name: default:TEST, procId: 62 completed
2020-03-13 14:54:15,521 INFO  [main] client.HBaseAdmin: Operation: DELETE, Table Name: default:TEST, procId: 64 completed
```

----End

Commissioning Phoenix on Linux

To commission Phoenix in a Linux environment, an ECS that can communicate with the cluster network must be available. For details, see **Preparing a Local Application Development Environment**.

Step 1 Modify the sample. Change the value of **enablePhoenix** in the sample code **TestMain** to **true** to enable the Phoenix sample program interface.

```
/**
 * Phoenix Example
 * if you would like to operate hbase by SQL, please enable it,
 * and you can refrence the url ("https://support.huaweicloud.com/devg-mrs/mrs_06_0041.html").
 * step:
 * 1.login
 * 2.operate hbase by phoenix.
 */
boolean enablePhoenix = false;
if (enablePhoenix) {
  PhoenixExample phoenixExample;
  try {
    phoenixExample = new PhoenixExample(conf);
    phoenixExample.testSQL();
  } catch (Exception e) {
    LOG.error("Failed to run Phoenix Example, because ", e);
  }
}
```

Step 2 Run the **mvn package** command to generate a JAR file, for example, **hbase-examples-mrs-2.0.jar**. Obtain the JAR file from the target directory in the project directory, and upload it to the **/opt/client/Hbase/hbase/lib** directory.

Step 3 Run the JAR file.

1. Before running the JAR file on the Linux client, run the following command to switch to the client directory as the user that is used for installation:

   cd $BIGDATA_CLIENT_HOME/HBase/hbase

NOTE

$BIGDATA_CLIENT_HOME indicates the client installation directory.

2. Run the following command:

   source $BIGDATA_CLIENT_HOME/bigdata_env

3. Copy the **phoenix-hbase**, **phoenix-core**, and **htrace-core-3.1.0-incubating.jar** packages obtained after decompressing **HBase Phoenix APIs** to the **/opt/client/HBase/hbase/lib** directory.

4. Copy the JAR package generated in **Step 2** and the **krb5.conf** and **user.keytab** files obtained in **Preparing an HBase Application Development User** to the **HBase/hbase/conf** directory in the client running environment, for example, **/opt/client/HBase/hbase/conf**. Create the **hbaseclient.properties** file in the **/opt/client/HBase/hbase/conf** directory. In the file, **user.name** corresponds to the new user **hbaseuser**, and the values of **userKeytabName** and **krb5ConfName** correspond to the authentication-related file names obtained in **Preparing an HBase Application Development User** (skip this step if Kerberos authentication is not enabled for the cluster).

   ```
   user.name=hbaseuser
   userKeytabName=user.keytab
   krb5ConfName=krb5.conf
   ```

Step 4 Execute the JAR file program.

hbase com.huawei.bigdata.hbase.examples.TestMain /opt/client/HBase/hbase/conf

com.huawei.bigdata.hbase.examples.TestMain is used as an example. Use the actual code instead.

/opt/client/HBase/hbase/conf corresponds to the path of files such as **user.keytab** and **krb5.conf** mentioned above.

NOTE

If error message "Message stream modified (41)" is displayed, the JDK version may be incorrect. Change the JDK version in the sample code to a version earlier than 8u_242 or delete the **renew_lifetime = 0m** configuration item from the **krb5.conf** configuration file.

Step 5 After the Phoenix application is run, you can check the application running status by viewing the running result.

```
2020-03-14 16:20:40,192 INFO  [main] client.HBaseAdmin: Operation: CREATE, Table Name: default:TEST, procId: 923 completed
2020-03-14 16:20:40,806 INFO  [main] examples.PhoenixExample: 1
2020-03-14 16:20:40,807 INFO  [main] examples.PhoenixExample: John
2020-03-14 16:20:40,807 INFO  [main] examples.PhoenixExample: 100000
2020-03-14 16:20:40,807 INFO  [main] examples.PhoenixExample: 1980-01-01
2020-03-14 16:20:40,830 INFO  [main] client.HBaseAdmin: Started disable of TEST
2020-03-14 16:20:41,574 INFO  [main] client.HBaseAdmin: Operation: DISABLE, Table Name: default:TEST, procId: 925 completed
2020-03-14 16:20:41,831 INFO  [main] client.HBaseAdmin: Operation: DELETE, Table Name: default:TEST, procId: 927 completed
```

----End

6.4.4 Commissioning the HBase Python Sample Program

Only MRS 1.9.*x* and earlier versions support this function.

HBase allows users to use the ThriftServer2 service to access HBase using Python. The Python sample program can run only in the Linux environment. To commission the HBase Python sample program, an ECS that can communicate with the cluster environment must be available. For details, see **Preparing a Local Application Development Environment**. In addition, the Python environment must be installed. For details about how to download the installation package, visit **https://www.python.org/**. The following describes how to run the sample on the master node of the cluster.

Step 1 Set up the sample running environment.

Obtain the Python dependency for running the sample program, search for and download the **decorator**, **gssapi**, **kerberos**, **krbcontext**, **pure-sasl**, and **thrift** packages from **https://pypi.org/**. (If Kerberos authentication is not enabled for a common cluster, only the **thrift** package needs to be installed.) Upload the package to the master node. For example, create the **/opt/hbase-examples/python** directory and upload it to the directory.

```
decorator-4.3.2.tar.gz
gssapi-1.5.1.tar.gz
kerberos-1.3.0.tar.gz
krbcontext-0.8.tar.gz
pure-sasl-0.6.1.tar.gz
thrift-0.11.0.tar.gz
```

Step 2 Upload the **hbase-python-example** folder in the sample project to the **/opt/hbase-examples** directory on the master node of the cluster, and upload the authentication file obtained from **Preparing an HBase Application Development User** to the directory.

Step 3 Create the **hbasepython.properties** file in **/opt/hbase-examples** and modify the configuration as follows:

```
clientHome=/opt/client
exampleCodeDir=/opt/hbase-examples/hbase-python-example
pythonLib=/opt/hbase-examples/python
keyTabFile=/opt/hbase-examples/user.keytab
userName=hbaseuser
thriftIp=xxx.xxx.xx.xxx
```

NOTE

- **clientHome**: path of the cluster client
- **exampleCodeDir**: path of the **hbase-python-example** file
- **pythonLib**: path for storing python dependency files in **Step 1**
- **keyTabFile**: user authentication credential **user.keytab** obtained from **Preparing an HBase Application Development User**
- **userName**: developer username in **Preparing an HBase Application Development User**
- **thriftIp**: IP address of the node where the **thriftserver2** is installed

Step 4 Run the following command to create an HBase table named **example**:

source /opt/client/bigdata_env

kinit *Username*

echo "create 'example','family1','family2'" | hbase shell

Step 5 Install the Python environment and run the program.

```
cd /opt/hbase-examples/hbase-python-example
sh initEnvAndRunDemo.sh /opt/hbase-examples/hbasepython.properties
```


NOTE

- Format the **initEnvAndRunDemo.sh** script before running the program. For example, run the following command: **dos2unix /opt/hbase-examples/hbasepython.properties**
- Before executing the script, ensure that the **example** table contains the column family **'family1''family2'** and already exists in the cluster.
- To run the program again, go to the **/opt/hbase-examples/hbase-python-example** directory and run the following command to execute the program commissioning example: **python DemoClient.py**

Step 6 After the HBase Python application is run, you can check the application running status by viewing the running result.

Figure 6-9 Application running success

```
[root@node-master1Dzgc hbase-python-example]# python DemoClient.py
Thrift2 Demo
Please check "README.txt" before Running the sample code.
This demo assumes you have a table called "example" with a column family called "family1" and "family2"
('Putting:', TPut(timestamp=None, writeToWal=True, columnValues=[TColumnValue(value='value1', qualifier='qualifier1', family='family1', times
tamp=None)], row='row2009'))
('Putting:', TPut(timestamp=None, writeToWal=True, columnValues=[TColumnValue(value='1value2', qualifier='bb', family='family2', timestamp=No
ne)], row='row2009'))
('Getting:', TGet(filterString=None, timestamp=None, maxVersions=None, timeRange=None, columns=None, row='row2009'))
('result for get:', 'row2009', 'family1', 'qualifier1', 'value1')

('result for get:', 'row2009', 'family2', 'bb', '1value2')

('putlist: ', [TPut(timestamp=None, writeToWal=True, columnValues=[TColumnValue(value='0value1', qualifier='aa', family='family1', timestamp=
None)], row='row0'), TPut(timestamp=None, writeToWal=True, columnValues=[TColumnValue(value='0value2', qualifier='bb', family='family2', time
stamp=None)], row='row0'), TPut(timestamp=None, writeToWal=True, columnValues=[TColumnValue(value='1value1', qualifier='aa', family='family1'
, timestamp=None)], row='row1'), TPut(timestamp=None, writeToWal=True, columnValues=[TColumnValue(value='1value2', qualifier='bb', family='fa
mily2', timestamp=None)], row='row1'), TPut(timestamp=None, writeToWal=True, columnValues=[TColumnValue(value='2value1', qualifier='aa', fami
ly='family1', timestamp=None)], row='row2'), TPut(timestamp=None, writeToWal=True, columnValues=[TColumnValue(value='2value2', qualifier='bb'
, family='family2', timestamp=None)], row='row2'), TPut(timestamp=None, writeToWal=True, columnValues=[TColumnValue(value='3value1', qualifie
r='aa', family='family1', timestamp=None)], row='row3'), TPut(timestamp=None, writeToWal=True, columnValues=[TColumnValue(value='3value2', qu
alifier='bb', family='family2', timestamp=None)], row='row3'), TPut(timestamp=None, writeToWal=True, columnValues=[TColumnValue(value='4value
1', qualifier='aa', family='family1', timestamp=None)], row='row4'), TPut(timestamp=None, writeToWal=True, columnValues=[TColumnValue(value='
4value2', qualifier='bb', family='family2', timestamp=None)], row='row4')])

('scan with startRow="row", stopRow="row2", and special column(family1:aa): ', 0)
('result for scan:', 'row0', 'family1', 'aa', '0value1')
('result for scan:', 'row1', 'family1', 'aa', '1value1')

('scan with filter: family2:bb=1value2', 1)
('result for scan:', 'row1', 'family1', 'aa', '1value1')
('result for scan:', 'row1', 'family2', 'bb', '1value2')
('result for scan:', 'row2009', 'family1', 'qualifier1', 'value1')
('result for scan:', 'row2009', 'family2', 'bb', '1value2')

scanner with filter family2:bb between 1value2 and 3value2
('result for scan:', 'row2', 'family1', 'aa', '2value1')
('result for scan:', 'row2', 'family2', 'bb', '2value2')
```

----End

6.5 FAQs About HBase Application Development

6.5.1 HBase APIs

6.5.1.1 HBase Shell APIs

You can directly perform operations on HBase using shell on the server. HBase shell APIs are consistent with those in an open source community. For details, see **http://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/**.

Methods of running shell commands:

Step 1 Go to any directory of the HBase client.

Step 2 Run the following command to initialize environment variables:

source /opt/client/bigdata_env

Step 3 If the Kerberos authentication is enabled for the current cluster, run the following command to authenticate the user. If the Kerberos authentication is disabled for the current cluster, skip this step. The current user is the development user added in **Preparing an HBase Application Development User**.

Human-machine user: *kinit MRS cluster user*

For example, **kinit hbaseuser**.

Machine-machine user: *kinit -kt Authentication credential path MRS cluster user*

For example, **kinit -kt /opt/user.keytab hbaseuser**.

Step 4 Run the **hbase shell** command.

Access the running mode of the HBase command line interface (also called CLI client connection).

```
hbase(main):001:0>
```

Run the ***help*** command to obtain help information about the HBase command parameters.

----End

Commands to Obtain HBase Replication Metrics

Run the **status** shell command to obtain all required metrics.

- Run the following command to view the replication source metric:

  hbase(main):019:0> status 'replication', 'source'

  The command output is as follows: (The actual node output is used.)

  ```
  version 1.0.2
  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
  ```

- Run the following command to view the replication sink metric:

  hbase(main):020:0> status 'replication', 'sink'

  The command output is as follows: (The actual node output is used.)

  ```
  version 1.0.2
  1 live servers
  BLR1000006595:
  SINK : AppliedBatches=0, AppliedOps=0, AppliedHFiles=0, AgeOfLastAppliedOp=0,
  TimeStampsOfLastAppliedOp=Wed May 25 17:55:21 CST 2016
  ```

- Run the following command to view both replication source and replication sink metrics at the same time:

  hbase(main):018:0> status 'replication'

  The command output is as follows: (The actual node output is used.)

  ```
  version 1.0.2
  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
```

6.5.1.2 HBase Java APIs

HBase adopts the same APIs as those of Apache HBase. For details, visit **http://hbase.apache.org/apidocs/index.html**.

Newly Added or Modified APIs

- **org.apache.hadoop.hbase.Cell** of HBase 0.98.3 rather than **org.apache.hadoop.hbase.KeyValue** of HBase 0.94 is recommended as the key-value data object.
- It is recommended that **HConnection connection = HConnectionManager.createConnection(conf)** be used to create a connection pool in HBase 0.98.3. The HTablePool is abandoned.
- For details about the new EndPoint API, visit **http://hbase.apache.org/book/cp.html**.
- The **isReversed()** and **setReversed(boolean reversed)** reversed scan methods are added to **org.apache.hadoop.hbase.client.Scan**.
- For details about API changes from HBase 0.98 to HBase 1.0, visit **https://issues.apache.org/jira/browse/hbase-10602**.
- **org.apache.hadoop.hbase.mapreduce** rather than **org.apache.hadoop.hbase.mapred** is recommended for HBase 1.0.
- For details about the version, visit **https://blogs.apache.org/hbase/entry/start_of_a_new_era**.
- New APIs added to obtain HBase replication metrics

Table 6-6 org.apache.hadoop.hbase.client.replication.ReplicationAdmin

Method	Description
getSourceMetricsSummary(String id)	Parameter type: String The source metric summary of the peer ID needs to be obtained. Return type: Map<String, String> Returned: A map, where the key is the RegionServer name and the value is the source cluster metric summary of the specified peer ID. Summary metrics are **sizeOfLogToReplicate** and **timeForLogToReplicate**.

Method	Description
getSourceMetrics(String id)	Parameter type: String The source metric summary of the peer ID needs to be obtained. Return type: Map<String, String> Returned: A map, where the key is the RegionServer name and the value is the source cluster metric of the specified peer ID.
getSinkMetrics()	Return type: Map<String, String> Returned: A map, where the key is the RegionServer name and the value is the source cluster sink metric of the specified peer ID.
getPeerSinkMetrics(String id)	Parameter type: String The source metric summary of the peer ID needs to be obtained. Return type: Map<String, String> Returned: A map, where the key is the RegionServer name and the value is the source cluster sink metric of the specified peer ID.

NOTE

All methods return a Map, where the key is "RegionServer name (IP/Host)" and the value is the string containing all the metrics in format of 'Metric Name'='Metric Value' [, 'Metric Name'= 'Metric Value']*.

Example: SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0

Table 6-7 org.apache.hadoop.hbase.replication.ReplicationLoadSource

Method	Description
getPeerID()	Return type: String Returned: peer cluster ID
getAgeOfLastShippedOp()	Return type: long Returned: milliseconds that the last successful replication request lasts
getSizeOfLogQueue()	Return type: long Returned: write-ahead logs (WALs) waiting for replication in the queue

Method	Description
getTimeStampOfLastShippedOp()	Return type: long Returned: timestamp of the last successful replication request
getReplicationLag()	Return type: long Returned: interval between current time and the time of the last successful replication request
getShippedOps()	Return type: long Returned: total number of data ops transferred
getShippedBytes()	Return type: long Returned: total number of data bytes transferred
getShippedBatches()	Return type: long Returned: total number of data batches transferred
getLogReadInBytes()	Return type: long Returned: total number of bytes read from WAL
getLogEditsRead()	Return type: long Returned: total number of edits read from WAL
getSizeOfLogToReplicate()	Return type: long Returned: total size of WALs waiting to be replicated in the queue
getTimeForLogToReplicate()	Return type: long Returned: seconds spent in replicating WALs in the queue
getShippedHFiles()	Return type: long Returned: total number of HFiles transferred
getSizeOfHFileRefsQueue()	Return type: long Returned: total number of HFiles waiting to be replicated
getLogEditsFiltered()	Return type: long Returned: total number of WAL edits filtered

Method	Description
getFailedReplicationAttempts()	Return type: long Returned: times failed to replicate data for a single request

Table 6-8 org.apache.hadoop.hbase.replication.ReplicationLoadSink

Method	Description
getAgeOfLastAppliedOp()	Return type: long Returned: milliseconds that the last successful applied WAL edits last
getTimeStampsOfLastAppliedOp()	Return type: long Returned: timestamp of the last successful applied WAL edit
getAppliedBatches()	Return type: long Returned: total number of data batches applied
getAppliedOps()	Return type: long Returned: total number of data ops applied
getAppliedHFiles()	Return type: long Returned: total number of HFiles applied

NOTE

The new API OF Replication Admin obtains the metric values from HMaster. Each RegionServer reports status to HMaster at every heartbeat interval, which is 3 seconds by default. Therefore, this API reports the latest metric value at the last heartbeat by using the RegionServer.

If you need the latest metric value, use the JMX API provided by the RegionServer.

- **1.3.1 (MRS 1.9.2) API Changes**
  - Added HIndex API

Table 6-9 org.apache.hadoop.hbase.hindex.client.HIndexAdmin

Method	Description
addIndices(TableName tablename,TableIndices tableIndices)	Parameter: **TableName** Name of the table to which the user wants to add a specified index. Parameter: **TableIndices** Table index to be added to the table Return type: **void**
addIndicesWithData(TableName tablename,TableIndices tableIndices)	Parameter: **TableName** Name of the table to which the user wants to add a specified index Parameter: **TableIndices** Table index to be added to the table Return type: void
dropIndices(TableName tableName,List <String> list)	Parameter: **TableName** Name of the table from which the user wants to delete an index Parameter: **List<String>** Contains the list of indexes to be deleted. Return type: **void**
dropIndicesWithData(TableName tableName,List <String> list)	Parameter: **TableName** Name of the table from which the user wants to delete a specified index Parameter: **List<String>** Contains the list of indexes to be deleted. Return type: **void**
disableIndices(TableName tableName,List <String> list)	Parameter: **TableName** Name of the table for which the user wants to disable a specified index Parameter: **List<String>** Contains the list of indexes to be disabled. Return type: **void**

Method	Description
enableIndices(TableName tableName,List <String> list)	Parameter: **TableName** Name of the table for which the user wants to enable a specified index Parameter: **List<String>** Contains the list of indexes to be enabled. Return type: **void**
listIndices(TableName tableName)	Parameter: **TableName** Name of the table for which the user wants to list all indexes Return type: List <Pair <HIndexSpecification,IndexState >> Return: A secondary index list is returned. The first element is the index specification, and the second element is the current state of the index.

6.5.1.3 HBase HFS Java APIs

Prerequisites

The cluster version is earlier than MRS 3.*x*.

API Description

This section describes major classes.

Common APIs of **org.apache.hadoop.hbase.filestream.client.FSTableInterface**:

Method	Description
void put(FSPut fsPut)	Inserts data into HFS tables.
void put(List<FSPut> fsPuts)	Inserts data into HFS tables in batches.
FSResult get(FSGet fsGet)	Reads data from HFS tables.
FSResult[] get(List<FSGet> fsGets)	Reads multiple lines of data from HFS tables.
void delete(FSDelete fsDelete)	Deletes data from HFS tables.
void delete(List<FSDelete> fsDeletes)	Deletes multiple lines of data from HFS tables.
void close()	Closes a table object.

org.apache.hadoop.hbase.filestream.client.FSTable is the implementation class of **org.apache.hadoop.hbase.filestream.client.FSTableInterface**.

org.apache.hadoop.hbase.filestream.client.FSHColumnDescriptor inherits from **org.apache.hadoop.hbase.HColumnDescriptor**. The following APIs are added:

Method	Description
public void setFileColumn()	Sets the column family of stored files to this column family.
public void setFileThreshold(int fileThreshold)	Sets the threshold for the size of stored files.

org.apache.hadoop.hbase.filestream.client.FSTableDescriptor inherits from **org.apache.hadoop.hbase.HTableDescriptor** without added APIs. This class is required when Java APIs are used to create HFS tables for storing files.

org.apache.hadoop.hbase.filestream.client.FSPut inherits from **org.apache.hadoop.hbase.Put**. The following APIs are added:

Method	Description
public FSPut(byte[] row)	Constructor. It constructs an object using Rowkeys.
public FSPut(byte[] row, long timestamp)	Constructor. It constructs an object using Rowkeys and timestamps.
public void addFile(String name, byte[] value)	Inserts a file into the column family of the stored files in the HFS table, with **name** as the column name and **value** as the file content.
public void addFile(String name, byte[] value, long ts)	Inserts a file into the column family of the stored files in the HFS table, with **name** as the column name, **value** as the file content, and **ts** as the specified timestamp.
public void addFile(String name, InputStream inputStream)	Inserts a file into the column family of the stored files in the HFS table, with **name** as the column name and **inputStream** as the input stream object of the file. The input stream object needs to be closed by the invoker.

Method	Description
public void addFile(String name, InputStream inputStream, long ts)	Inserts a file into the column family of the stored files in the HFS table, with **name** as the column name, **inputStream** as the input stream object of the file, and **ts** as the specified timestamp. The input stream object needs to be closed by the invoker.

org.apache.hadoop.hbase.filestream.client.FSGet inherits from **org.apache.hadoop.hbase.Get**. The following APIs are added:

Method	Description
public FSGet(byte[] row)	Constructor. It constructs an object using Rowkeys.
public void addFile(String fileName)	Specifies the file to be returned.
public void addFiles(List<String> fileNames)	Specifies the files to be returned.

org.apache.hadoop.hbase.filestream.client.FSResult inherits from **org.apache.hadoop.hbase.Result**. The following APIs are added:

Method	Description
public FSFile getFile(String fileName)	Returns the FSFile object whose file name is specified from the query results.

org.apache.hadoop.hbase.filestream.client.FSFile API:

Method	Description
public InputStream createInputStream()	Obtains the input stream object from the FSFile object.

6.5.1.4 HBase Phoenix APIs

Version Mapping

If you want to use Phoenix, download the Phoenix version corresponding to the current MRS cluster. For details, see **https://phoenix.apache.org**. **Table 6-10** lists the version mapping between MRS and Phoenix.

Table 6-10 Version mapping between MRS and Phoenix

MRS Version	Phoenix Version	Remarks
MRS 1.9.2	x.xx.x-HBase-1.3	Example: 4.14.1-HBase-1.3

Configuration Method

For versions earlier than MRS 3.*x*, download the third-party Phoenix package from the official website and perform the following configurations. MRS 3.*x* or later supports Phoenix so you can directly use Phoenix on the node where the HBase client is installed. For details about operations on clusters with Kerberos authentication enabled, see **Phoenix Command Line**. For details about operations on clusters with Kerberos authentication disabled, see **Phoenix Command Line**:

1. Download the Phoenix binary package from the official website (**https://phoenix.apache.org/download.html**), and upload it to any Master node in the cluster. Decompress the package, modify the permission, and switch to user **omm** (for example, **apache-phoenix-4.14.1-HBase-1.3-bin.tar.gz**).

   ```
   tar -xvf apache-phoenix-4.14.1-HBase-1.3-bin.tar.gz
   chown omm:wheel apache-phoenix-4.14.1-HBase-1.3-bin -R
   su - omm
   ```

2. Go to the **apache-phoenix-4.14.1-HBase-1.3-bin** directory and edit the following script. For example, if the script name is **installPhoenixJar.sh**, run the following command: **sh installPhoenixJar.sh <PHOENIX_HBASE_VERSION> <MRS_VERSION> <IPs>** (IP indicates the IP address of the node where HBase is installed, that is, the IP addresses of all Master and Core nodes. Use the actual IP address of the current cluster.) For example, the script is as follows:

   ```
   #!/bin/bash

   PHOENIX_HBASE_VERSION=$1
   shift
   MRS_VERSION=$1
   shift
   IPs=$1
   shift
   check_cmd_result() {
       echo "executing command: $*"
       str="$@"
       if [ ${#str} -eq 7 ]; then
           echo "please check input args, such as, sh installPhoenixJar.sh 5.0.0-HBase-2.0 2.0.1
   xx.xx.xx.xx,xx.xx.xx.xx,xx.xx.xx.xx"
           exit 1
       fi
       if ! eval $*
       then
           echo "Failed to execute: $*"
   ```



```
        exit 1
    fi
}

check_cmd_result [ -n "$PHOENIX_HBASE_VERSION" ]
check_cmd_result [ -n "$MRS_VERSION" ]
check_cmd_result [ -n "$IPs" ]

if [ ${MRS_VERSION}X = "1.8.5"X ]; then
    MRS_VERSION="1.8.3"
fi
if [[ ${MRS_VERSION} =~ "1.6" ]]; then
    WORKDIR="FusionInsight"
elif [[ ${MRS_VERSION} =~ "1.7" ]]; then
    WORKDIR="MRS"
else
    WORKDIR="MRS_${MRS_VERSION}/install"
fi

check_cmd_result HBASE_LIBDIR=$(ls -d /opt/Bigdata/${WORKDIR}/FusionInsight-HBase-*/hbase/lib)
# copy jars to local node.
check_cmd_result cp phoenix-${PHOENIX_HBASE_VERSION}-server.jar ${HBASE_LIBDIR}
check_cmd_result cp phoenix-core-${PHOENIX_HBASE_VERSION}.jar ${HBASE_LIBDIR}

check_cmd_result chmod 700 ${HBASE_LIBDIR}/phoenix-${PHOENIX_HBASE_VERSION}-server.jar
check_cmd_result chmod 700 ${HBASE_LIBDIR}/phoenix-core-${PHOENIX_HBASE_VERSION}.jar

check_cmd_result chown omm:wheel ${HBASE_LIBDIR}/phoenix-${PHOENIX_HBASE_VERSION}-
server.jar
check_cmd_result chown omm:wheel ${HBASE_LIBDIR}/phoenix-core-$
{PHOENIX_HBASE_VERSION}.jar

if [[ "$MRS_VERSION" =~ "2." ]]; then
    check_cmd_result rm -rf ${HBASE_LIBDIR}/htrace-core-3.1.0-incubating.jar
    check_cmd_result rm -rf /opt/client/HBase/hbase/lib/joda-time-2.1.jar
    check_cmd_result ln -s /opt/share/htrace-core-3.1.0-incubating/htrace-core-3.1.0-incubating.jar \
${HBASE_LIBDIR}/htrace-core-3.1.0-incubating.jar
    check_cmd_result ln -s /opt/share/joda-time-2.1/joda-time-2.1.jar /opt/client/HBase/hbase/lib/joda-
time-2.1.jar
fi

# copy jars to other nodes.
localIp=$(hostname -i)
ipArr=($(echo "$IPs" | sed "s|\,|\ |g"))
for ip in ${ipArr[@]}
do
    if [ "$ip"X = "$localIp"X ]; then
        echo "skip copying jar to local node."
        continue
    fi
    check_cmd_result scp ${HBASE_LIBDIR}/phoenix-${PHOENIX_HBASE_VERSION}-server.jar ${ip}:$
{HBASE_LIBDIR} 2>/dev/null
    check_cmd_result scp ${HBASE_LIBDIR}/phoenix-core-${PHOENIX_HBASE_VERSION}.jar ${ip}:$
{HBASE_LIBDIR} 2>/dev/null
    if [[ "$MRS_VERSION" =~ "2." ]]; then
        check_cmd_result ssh $ip "rm -rf ${HBASE_LIBDIR}/htrace-core-3.1.0-incubating.jar" 2>/dev/null
        check_cmd_result ssh $ip "rm -rf /opt/client/HBase/hbase/lib/joda-time-2.1.jar" 2>/dev/null
        check_cmd_result ssh $ip "ln -s /opt/share/htrace-core-3.1.0-incubating/htrace-core-3.1.0-
incubating.jar \
${HBASE_LIBDIR}/htrace-core-3.1.0-incubating.jar" 2>/dev/null
        check_cmd_result ssh $ip "ln -s /opt/share/joda-time-2.1/joda-time-2.1.jar /opt/client/HBase/
hbase/lib/joda-time-2.1.jar" 2>/dev/null
    fi
done
echo "installing phoenix jars to hbase successfully..."
```


NOTE

- Copy and import the preceding scripts in **.txt** format to avoid format errors.
- **<PHOENIX_HBASE_VERSION>**: Current Phoenix version. For example, versions earlier than MRS 3.*x* support Phoenix **4.14.1-HBase-1.3**.
- **<MRS_VERSION>**: Current MRS version.
- **<IPs>**: IP addresses of the nodes where HBase is installed, that is, the IP addresses of the Master and Core nodes in the current cluster. The IP addresses are separated by comma (,).
- If the message "**installing phoenix jars to hbase successfully...**" is displayed after the script is executed, Phoenix has been successfully installed.

3. Log in to **MRS Manager** and restart the HBase service.
4. Configure the Phoenix client parameters. You can skip this step for a cluster with Kerberos authentication disabled.
   a. Configure authentication information for a Phoenix connection. Go to **$PHOENIX_HOME/bin** and edit the **hbase-site.xml** file. Set the parameters listed in **Table 6-11**.

Table 6-11 Phoenix parameters

Parameter	Description	Default Value
hbase.regionserver.kerberos.principal	Principal of RegionServer of the current cluster	Not set
hbase.master.kerberos.principal	Principal of HMaster of the current cluster	Not set
hbase.security.authentication	Authentication mode used for initializing the Phoenix connection.	kerberos

You can configure the parameters as follows:

```
<property>
<name>hbase.regionserver.kerberos.principal</name>
<value>hbase/hadoop.hadoop.com@HADOOP.COM</value>
</property>
<property>
<name>hbase.master.kerberos.principal</name>
<value>hbase/hadoop.hadoop.com@HADOOP.COM</value>
</property>
<property>
<name>hbase.security.authentication</name>
<value>kerberos</value>
</property>
```


NOTE

The **hbase.master.kerberos.principal** and **hbase.regionserver.kerberos.principal** parameters are the Kerberos users of HBase in the security cluster with Kerberos authentication enabled. You can search the **hbase-site.xml** file on the client to obtain the parameter values. For example, if the client is installed in the **/opt/client** directory of the Master node, you can run the **grep "kerberos.principal" /opt/client/HBase/hbase/conf/hbase-site.xml -A1** command to obtain the principal of HBase, as shown in **Figure 6-10**.

Figure 6-10 Obtaining the principal of HBase.

```
[root@xxxxx-xxxxx-5 opt]# grep "kerberos.principal" /opt/client/HBase/hbase/conf/hbase-site.xml -A1
<name>hbase.regionserver.kerberos.principal</name>
<value>hbase/hadoop.hadoop.com@HADOOP.COM</value>
--
<name>hbase.master.kerberos.principal</name>
<value>hbase/hadoop.hadoop.com@HADOOP.COM</value>
--
```

b. Modify the **sqlline.py** script (for example, **apache-phoenix-4.14.1-HBase-1.3-bin/bin/sqlline.py**) in the **bin** directory of the **Phoenix** path and add the dependency information of the HBase client, as shown in **Figure 6-11**.

Figure 6-11 Phoenix dependencies and ZooKeeper authentication

```
106     colorSetting = "false"
107
108 java_cmd = java + ' $PHOENIX_OPTS $HBASE_OPTS' + \
109     ' -cp "$HBASE_HOME/lib/*:"' + hbase_config_path + os.pathsep + phoenix_utils.hbase_conf_dir + os.pathsep + phoenix_utils.phoenix_client_jar + \
110     os.pathsep + phoenix_utils.hadoop_common_jar + os.pathsep + phoenix_utils.hadoop_hdfs_jar + \
111     os.pathsep + phoenix_utils.hadoop_conf + os.pathsep + phoenix_utils.hadoop_classpath + '" -Dlog4j.configuration=file:' + \
112     os.path.join(phoenix_utils.current_dir, "log4j.properties") + \
113     " sqlline.SqlLine -d org.apache.phoenix.jdbc.PhoenixDriver" + \
114     " -u jdbc:phoenix:" + phoenix_utils.shell_quote([zookeeper]) + \
115     " -n none -p none --color=" + colorSetting + " --fastConnect=" + args.fastconnect + \
116     " --verbose=" + args.verbose + " --incremental=false --isolation=TRANSACTION_READ_COMMITTED " + sqlfile
```

The configuration details are as follows:

Add the **lib** package (for example, **$HBASE_HOME/lib/*:**) of the HBase client.
Add related authentication information (for example, **$HBASE_OPTS**).

Usage

Phoenix enables you to operate HBase using SQL statements. The following describes how to use SQL statements to create tables, insert data, query data, and delete tables. Phoenix also allows you to operate HBase using JDBC. For details, see **HBase SQL Query Sample Code**.

1. Connect to Phoenix.

   source /opt/client/bigdata_env
   kinit *MRS cluster user* (The MRS cluster user can be the built-in user **hbase** or another user that has been added to the **hbase** group. Skip this command for a cluster with Kerberos authentication disabled.)
   cd $PHOENIX_HOME
   bin/sqlline.py zookeerIp:2181

NOTE

1. For versions earlier than MRS 1.9.2, the ZooKeeper port number is 24002. For details, see the ZooKeeper cluster configurations on MRS Manager.

2. If the Phoenix index function is used, add the following configurations to the HBase server (including HMaster and RegionServer). For details, see **https://phoenix.apache.org/secondary_indexing.html**.

<property>

<name>hbase.regionserver.wal.codec</name>

<value>org.apache.hadoop.hbase.regionserver.wal.IndexedWALEditCodec</value>

</property>

2. Create a table.
```
CREATE TABLE TEST (id VARCHAR PRIMARY KEY, name VARCHAR);
```
3. Insert data.
```
UPSERT INTO TEST(id,name) VALUES ('1','jamee');
```
4. Query data.
```
SELECT * FROM TEST;
```
5. Delete a table.
```
DROP TABLE TEST;
```

6.5.1.5 HBase REST APIs

MRS1.6 and later versions allow you to perform service operations on HBase using REST APIs, which support the **curl** command and Java client. The use method of the **curl** commands is the same as that of Apache HBase. Visit **https://hbase.apache.org/book.html#_rest** for more information.

NOTE

Currently, the default SSL protocols are TLSv1.1 and TLSv1.2. Therefore, you need to check whether the current environment supports the SSL protocols when you run the **curl** command to invoke a REST API.

Running the curl Command

- For clusters with Kerberos authentication disabled

  Before running the **curl** command in a cluster with Kerberos authentication disabled, add the parameters as follows:

  ```
  curl -vi -k POST -H "Accept: text/xml" -H "Content-Type: text/xml" -d '<?xml version="1.0" encoding="UTF-8"?> <TableSchema name="users"><ColumnSchema name="cf" /> </TableSchema>' "https://<IP address of the HBase node where the RESTServer service is installed>:21309/users/schema"
  ```

- For clusters with Kerberos authentication enabled

  When you run the **curl** command in a cluster with Kerberos authentication enabled, you need to perform the following steps:

  a. Perform Kerberos authentication as follows:

  Human-machine user: *kinit MRS cluster user*

  For example, **kinit hbaseuser**.

  Machine-machine user: *kinit -kt Authentication credential path MRS cluster user*

  For example, **kinit -kt /opt/user.keytab hbaseuser**.

b. In the **curl** command, add the **--negotiate -u** parameter before the request type as follows:

```
curl -vi -k --negotiate -u: POST -H "Accept: text/xml"   -H "Content-Type: text/xml" -d '<?xml version="1.0" encoding="UTF-8"?> <TableSchema name="users"><ColumnSchema name="cf" /> </TableSchema>' "https://<IP address of the HBase node where the RESTServer service is installed>:21309/users/schema"
```

Using the Java Client

Perform the following operations to use Java to call REST APIs. (You can refer to some code of **RestExample** in the sample code.)

1. Perform Kerberos authentication. You can skip this step for a cluster with Kerberos authentication disabled.
2. Create a cluster object of the **org.apache.hadoop.hbase.rest.client.Cluster** class, and add a cluster by invoking the **add** method of the cluster class and the cluster IP address and port of the REST server.

```
Cluster cluster = new Cluster();
cluster.add("10.10.10.10:21309");
```

3. Use the client object of the cluster initialization class **org.apache.hadoop.hbase.rest.client.Client** added in step 2 to invoke **doAs** to operate HBase.

```
Client client = new Client(cluster, true);
UserGroupInformation.getLoginUser().doAs(new PrivilegedAction() {
 public Object run() {

  // Rest client code

  /* Sample code to list all the tables
     client.get("/")
  */

  return null;
 }
});
```

4. You can use the following methods to call different REST APIs.

- **Using plain text to obtain a namesapce**

1. Taking a path including namespace as a parameter, use the client to invoke the Get method to obtain a namespace. The response will be captured by an object of the **org.apache.hadoop.hbase.rest.client.Response** class. The following is an example.

```
Response response;
String namespacePath = "/namespaces/" + "nameSpaceName";
response = client.get(namespacePath);
System.out.println(Bytes.toString(response.getBody()));
```

- **Creating or modifying a namespace**

1. When creating or modifying a namespace, you need to use **NamespacesInstanceModel** to create a model and use the **buildTestModel()** method to build the mode. The model you create must contain the properties of the namespace to be created.

```
Map<String, String> NAMESPACE1_PROPS = new HashMap<String, String>();
NAMESPACE1_PROPS.put("key1", "value1");

NamespacesInstanceModel model = buildTestModel(NAMESPACE1,NAMESPACE1_PROPS);

private static NamespacesInstanceModel buildTestModel(String namespace, Map<String, String> properties) {
NamespacesInstanceModel model = new NamespacesInstanceModel();
```



```
for (String key : properties.keySet()) {
model.addProperty(key, properties.get(key));
}
return model;
}
```

NOTE

When you send a POST or PUT request to create or modify a table, **TableSchemaModel** is used to create a model class.

2. You can use the following methods to create and modify namespaces.

- **Creating a namespace using XML**

1. After you use **NamespacesInstanceModel** to create a model, use the client to invoke the Post method to create a namespace. The parameters include the namespace path, content type, and content. For the content type, the invoked class is **org.apache.hadoop.hbase.rest.Constants**, and the invoked parameter here is **Constants.MIMETYPE_XML**. For the content, the following example uses the **toXML()** method to convert the content to the XML format. The response will be captured by an object of the **org.apache.hadoop.hbase.rest.client.Response** class. The following is an example.

```
Response response;
String namespacePath = "/namespaces/" + "nameSpaceName";
response = client.post(namespacePath, Constants.MIMETYPE_XML, toXML(model));

private static byte[] toXML(NamespacesInstanceModel model) throws JAXBException {
StringWriter writer = new StringWriter();
context.createMarshaller().marshal(model, writer);
return Bytes.toBytes(writer.toString());
}
```

2. When sending a Get request using XML, you can use the **fromXML()** method to obtain the model from the response and find the name of the created namespace from the model.

```
private static <T> T fromXML(byte[] content) throws JAXBException {
return (T) context.createUnmarshaller().unmarshal(new ByteArrayInputStream(content));
}
```

- **Modifying a namespace using JSON**

1. After you use **NamespacesInstanceModel** to create a model, invoke the Put method of the client class to create a namespace. The parameters include the namespace path, content type, and content. For the content type, the invoked class is **org.apache.hadoop.hbase.rest.Constants**, and the invoked parameter here is **Constants.MIMETYPE_JSON**. For the content, the following example converts the content to the JSON format and uses **jsonMapper** as a parameter. The response will be captured by an object of the **org.apache.hadoop.hbase.rest.client.Response** class. The following is an example.

```
ObjectMapper jsonMapper = new JacksonProvider().locateMapper(NamespacesInstanceModel.class,
MediaType.APPLICATION_JSON_TYPE);

Response response;
String namespacePath = "/namespaces/" + "nameSpaceName";
String jsonString = jsonMapper.writeValueAsString(model);

response = client.put(namespacePath, Constants.MIMETYPE_JSON, Bytes.toBytes(jsonString));
```

2. When sending a Get request using JSON, you can use the **readValue()** method of **jsonMapper** to obtain the model from the response and find the name of the created namespace from the model.


```
jsonMapper.readValue(response.getBody(), NamespacesInstanceModel.class);

/*Here second argument should be according to API, if its     **related to table it should be
TableSchemaModel.class*/
```

- **Modifying a namespace using Protobuf**

1. After you use **NamespacesInstanceModel** to create a model, invoke the Put method of the client class to create a namespace. The parameters include the namespace path, content type, and content. For the content type, the invoked class is **org.apache.hadoop.hbase.rest.Constants**, and the invoked parameter here is **Constants.MIMETYPE_PROTOBUF**. For the content, the following example converts the content as follows and uses **createProtobufOutput** to create Protobuf. The response will be captured by an object of the **org.apache.hadoop.hbase.rest.client.Response** class. The following is an example.

```
Response response;
String namespacePath = "/namespaces/" + "nameSpaceName";

response = client.put(namespacePath, Constants.MIMETYPE_PROTOBUF,
model.createProtobufOutput());
model.getObjectFromMessage(response.getBody());
```

2. When sending a Get request using Protobuf, you can use the **getObjectFromMessage** method to obtain the model from the response and find the name of the created namespace from the model.

```
model.getObjectFromMessage(response.getBody());
```

6.5.2 HBase SQL Query Sample Code

Function Description

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 more HBase scans, and compiles and executes the scan tasks to generate a standard JDBC result set.

Sample Code

- A temporary directory for storing intermediate query results is configured in **hbase-example/conf/hbase-site.xml** on the client. If a client program executes the temporary directory on Linux, configure a Linux path. If a client program executes the temporary directory on Windows, configure a Windows path.

```
<property>
    <name>phoenix.spool.directory</name>
    <value>[1] Temporary directory for storing intermediate query results</value>
</property>
```

- Java example: Using 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);
            // 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
            }
        }
    }
}
```

Precautions

- A temporary directory for storing intermediate query results must be configured in **hbase-site.xml**. The size of the query result set is restricted by the directory size.
- Phoenix implements most **java.sql** interfaces. SQL follows the ANSI SQL standard.
- For versions later than MRS 1.9.2, download and configure the open source phoenix package by referring to **HBase Phoenix APIs**.

6.5.3 How Do I Configure HBase File Storage?

Prerequisites

The cluster version is earlier than MRS 3.*x*.

Scenario

HBase FileStream (HFS) is an independent HBase file storage module. It is used in MRS upper-layer applications by encapsulating HBase and HDFS interfaces to provide these upper-layer applications with functions such as file storage, read, and deletion.

In the Hadoop ecosystem, both HDFS and HBase face tough problems in massive file storage in some scenarios:

- If a large number of small files are stored in HDFS, the NameNode will be under great pressure.
- Some large files cannot be directly stored on HBase due to HBase APIs and internal mechanisms.

HFS is developed for the mixed storage of massive small files and some large files in Hadoop. In a word, both massive amounts of small files (smaller than 10 MB) and some large files (greater than 10 MB) need to be stored in HBase tables.

For such a scenario, HFS provides unified operation APIs similar to HBase function APIs. You must add **org.apache.hadoop.hbase.filestream.coprocessor.FileStreamMasterObserver** to the **hbase.coprocessor.master.classes** HBase configuration parameter.

NOTICE

- If only small files are stored, HBase original APIs are recommended.
- HFS APIs need to perform operations on both HBase and HDFS at the same time. Therefore, client users must have operation permissions of both components.
- When directly storing large files in HDFS, HFS will add some metadata information. Therefore, the stored files are not the original ones. When you use these files, use HFS APIs to read them instead of directly moving them out of HDFS.
- Backup and disaster recovery are not supported for data stored in HDFS by using HFS APIs.

Procedure

Step 1 Log in to **MRS Manager**.

Step 2 Choose **Service** > **HBase** > **Service Configuration**, and set **Type** to **All**. Choose **HMaster** > **System** on the left.

Step 3 In the **hbase.coprocessor.master.classes** configuration item, add **org.apache.hadoop.hbase.filestream.coprocessor.FileStreamMasterObserver**.

Step 4 Click **Save Configuration**. In the window that is displayed, select **Restart the affected services or instances** and click **Yes** to restart the HBase service.

----End

6.5.4 What Do I Do When There Is an HBase Application Running Exception?

The prompt message contains the solution of **org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory**.

Step 1 Check whether the **hbase-site.xml** configuration file of the application development project contains the **hbase.rpc.controllerfactory.class** configuration item.

```
<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, import the **phoenix-core-4.4.0-HBase-1.0.jar** JAR file. You can obtain the JAR file from **HBase/hbase/lib** in the HBase client installation directory.

Step 3 If you do not want to import this JAR file, you need to delete **hbase.rpc.controllerfactory.class** from the **hbase-site.xml** configuration file.

----End

6.5.5 Application Scenarios of HBase BulkLoad and Put

Both the BulkLoad and Put methods can be used to load data to HBase. Though BulkLoad loads data faster than Put, BulkLoad has disadvantages. The following describes the application scenarios of these two data loading methods.

BulkLoad starts MapReduce tasks to generate HFile files, and then registers HFile files with HBase. Incorrect use of BulkLoad will consume more cluster memory and CPU resources due to started MapReduce tasks. A large number of the generated small HFile files may frequently trigger Compaction, decreasing query speed dramatically.

Incorrect use of the Put method may cause slow data loading. If the memory allocated to RegionServer is insufficient, the process may exit due to the RegionServer memory overflow.

The application scenarios of the BulkLoad and Put methods are as follows:

- BulkLoad:
  - Large amounts of data needs to be loaded to HBase in the one-off manner.
  - When data is loaded to HBase, requirements on reliability are not high and WAL files do not need to be generated.
  - When the Put method is used to load large amounts of data to HBase, data loading and query will be slow.
  - The size of an HFile generated after data loading is similar to the size of HDFS blocks.
- Put:
  - The size of the data loaded to one Region at a time is smaller than half the size of HDFS blocks.
  - Data needs to be loaded to HBase in real time.
  - The query speed must not decrease dramatically during data loading.

7 HDFS Development Guide

7.1 HDFS Application Development Overview

7.1.1 Introduction to HDFS Application Development

HDFS

Hadoop distribute file system (HDFS) is a distributed file system with high fault tolerance running on universal hardware. HDFS supports data access with high throughput and applies to processing of large-scale 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 excellent scalability

HDFS APIs

HDFS applications can be developed using Java. For details about APIs, see **HDFS Java APIs**.

7.1.2 Common Concepts of HDFS Application Development

DataNode

A DataNode is used to store data blocks of each file and periodically report the DataNode data 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.

- Active NameNode: An active NameNode manages the namespace, directory structure, and metadata of file systems, and records the mapping relationships between data blocks and files to which the data blocks belong.
- Standby NameNode: A standby NameNode synchronizes data with the active NameNode and takes over services from the active NameNode if the active NameNode becomes abnormal.

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 writes status information to ZooKeeper. ZKFC also has permission to select an active NameNode.

Colocation

Colocation is to store associated data or data on which associated operations are performed 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

HDFS clients include Java API, C API, shell, HTTP REST API, and web UI.

- Java API

  Provides application APIs for HDFS. You can follow instructions in **HDFS Java APIs** to use Java APIs to develop HDFS applications.
- C API

  Provides application APIs for HDFS. You can follow instructions in **HDFS C APIs** to use C language to develop applications.
- Shell

  Provides shell commands to perform operations on HDFS. For details, see **HDFS Shell Commands**.
- HTTP REST API

  Provides APIs except shell, Java APIs, and C APIs to monitor HDFS status. For details, see **HDFS HTTP REST APIs**.
- Web UI

  Provides a visualized management web page.

Keytab file

The keytab file is a key file that stores user information. Applications use the keytab file to perform API authentication on the MRS Hadoop component.

7.1.3 HDFS Application Development Process

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

Figure 7-1 HDFS application development process

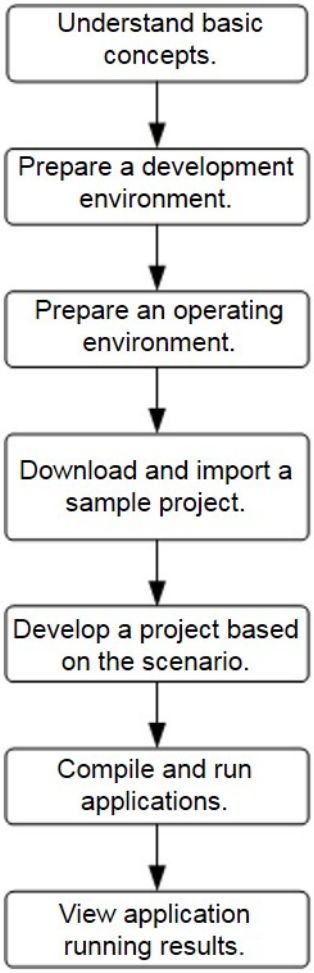


Table 7-1 Description of the HDFS development process

Phase	Description	Reference
Understand basic concepts.	Before application development, learn basic concepts of HDFS.	Common Concepts of HDFS Application Development
Prepare a development environment.	Use the Eclipse tool to configure the development environment according to the guide.	Preparing the Eclipse and JDK

Phase	Description	Reference
Prepare an operating environment.	The HDFS operating environment is an HDFS client. Install and configure the client according to the guide.	**Preparing an HDFS Application Running Environment**
Download and import a sample project.	HDFS provides sample projects for different scenarios. You can import a sample project to learn the application.	**Importing and Configuring HDFS Sample Projects**
Develop a project based on the scenario.	HDFS provides sample projects to help you quickly understand APIs of HDFS components.	**HDFS Development Plan**
Compile and run applications.	You can compile the developed application and submit it for running.	Linux: **Commissioning an HDFS Application on Linux**
View application running results.	Application running results are exported to a path you specify. You can also view the application running status on the UI.	Linux: **Viewing the HDFS Application Commissioning Result**

7.2 Preparing an HDFS Application Development Environment

7.2.1 Preparing a Local Application Development Environment

Table 7-2 describes the environment required for application development.

Table 7-2 Development environment

Item	Description
Eclipse	Basic configurations of the development environment. Eclipse 4.2 or later is required.

Item	Description
JDK	JDK 1.7 or 1.8 **NOTE** For security purpose, MRS cluster servers support only TLS 1.1 and TLS 1.2 encryption protocols. IBM JDK supports only 1.0 by default. If you use IBM JDK, set **com.ibm.jsse2.overrideDefaultTLS** to **true**. After the parameter setting, TLS1.0/1.1/1.2 can be supported at the same time. For details, visit **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**.

7.2.2 Preparing an HDFS Application Development User

Prerequisites

Kerberos authentication has been enabled for the MRS cluster. Skip this step if Kerberos authentication is not enabled for the cluster.

Scenario

The development user is used to run the sample project. The user must have HDFS permissions to run the HDFS sample project.

Procedure

Step 1 Log in to **MRS Manager** and choose **System** > **Manage Role** > **Create Role**.

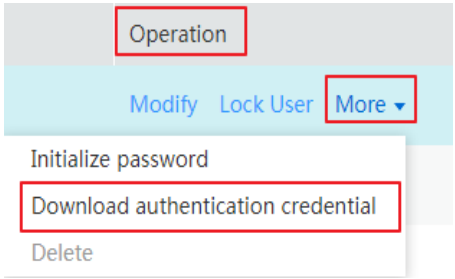
1. Enter a role name, for example, *hdfsrole*.
2. Edit the role. In **Permission**, choose **HDFS** > **File System** > **hdfs://hacluster/**, select **Read**, **Write**, and **Execute**, and click **OK**.

Step 2 Choose **System** > **Manage User** > **Create User** to create a user for the sample project.

Step 3 Enter a username, for example, *hdfsuser*. Set **User Type** to **Machine-machine**, and select **supergroup** in **User Group**. Set **Primary Group** to **supergroup**, select **hdfsrole** in **Assign Rights by Role**, and click **OK**.

Step 4 On MRS Manager, choose **System** > **User Management**. On the displayed page, select **hdfsuser** from the **Username** drop-down list. In the **Operation** column on the right, choose **More** > **Download authentication credential**. Save the downloaded package and decompress it to obtain the **user.keytab** and **krb5.conf** files for security authentication in the sample project, as shown in **Figure 7-2**.

Figure 7-2 Downloading the authentication credential



----End

7.2.3 Preparing the Eclipse and JDK

Prerequisites

You have enabled Kerberos authentication for the MRS cluster.

Scenario

In the Windows environment, you need to install the Eclipse and JDK.

Procedure

Step 1 Install the Eclipse program in the development environment. The Eclipse version must be 4.2 or later.

Step 2 Install the JDK program in the development environment. The JDK version must be 1.7 or 1.8.

NOTE

- If you use IBM JDK, ensure that the JDK configured in Eclipse is IBM JDK.
- If you use Oracle JDK, ensure that the JDK configured in Eclipse is Oracle JDK.
- Do not use the same workspace and the sample project in the same path for different Eclipse programs.

----End

7.2.4 Preparing an HDFS Application Running Environment

Prerequisites

1. You have installed HDFS on the server and confirmed that HDFS is running properly.

2. You have installed JDK 1.7 or 1.8 on the client operating environment.

3. You have obtained the **MRS_Services_Client.tar** client installation package.

Scenario

Install the client on Linux.

Procedure

Step 1 Ensure that the time difference between the client and the Hadoop cluster is less than 5 minutes. You may need to manually modify the client or Hadoop cluster time.

You can query the MRS cluster time by logging in to the active management node that corresponds to the cluster management IP address to run the **date** command.

Step 2 Download the MapReduce client program to the local computer.

1. Log in to **MRS Manager**.
2. Choose **Service** > **Download Client** to download the client program to the local PC.

Step 3 Decompress the **MRS_Services_Client.tar** client program package. Because the installation package is in **.tar** format, run the following commands to decompress the package twice:

tar -xvf MRS_Services_Client.tar

tar -xvf MRS_Service_ClientConfig.tar

Step 4 Set environment variables for the operating environment. Assume that the installation package is decompressed in **MRS_Services_ClientConfig/**.

Go to the decompressed folder and run the following command to install the client:

sh *install.sh* {client_install_home}

Step 5 Go the client installation directory and run the following command to initialize the environment variables:

source *bigdata_env*

Step 6 Copy the following files from the server to the **conf** directory in the same directory as the directory of the JAR file. For details about the JAR file exported by the sample project, see **Commissioning an HDFS Application on Linux**.

Table 7-3 Configuration files

File Name	Function	How to Obtain
core-site.xml	Configures HDFS parameters.	${HADOOP_HOME}/etc/hadoop/core-site.xml
hdfs-site.xml	Configures HDFS parameters.	${HADOOP_HOME}/etc/hadoop/hdfs-site.xml
user.keytab	Provides HDFS user information for Kerberos security authentication.	If the cluster is in security mode, contact the administrator to obtain the **keytab** and **krb5** files corresponding to the account.
krb5.conf	Contains Kerberos server configuration information.	

NOTE

- In Table 7-3, **${HADOOP_HOME}** indicates the Hadoop installation directory on the server.
- Keytab authentication is valid for 24 hours. Re-authentication is required 24 hours later.
- In the sample code, the username of **PRNCIPAL_NAME** must be the same as the account name of the **keytab** and **krb5** files.
- The **user.keytab** and **krb5.conf** of different clusters cannot be shared.
- In the sample code, the **keytab** file used in **System.getProperty("user.dir") + File.separator + "conf" + File.separator + "user.keytab"** must be the same as the user's keytab file.
- The **log4j.properties** file in the **conf** directory is configured based on the customer requirements.

----End

7.2.5 Importing and Configuring HDFS Sample Projects

Scenario

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

The following procedure describes how to import HDFS sample code.

Procedure

Step 1 Download the sample project to the local computer by referring to **Obtaining the MRS Application Development Sample Project**.

Step 2 Import the sample project to the Eclipse development environment.

1. Method 1: Open Eclipse and choose **File** > **New** > **Java Project**.
2. Deselect **Use default location** and click **Browse**.

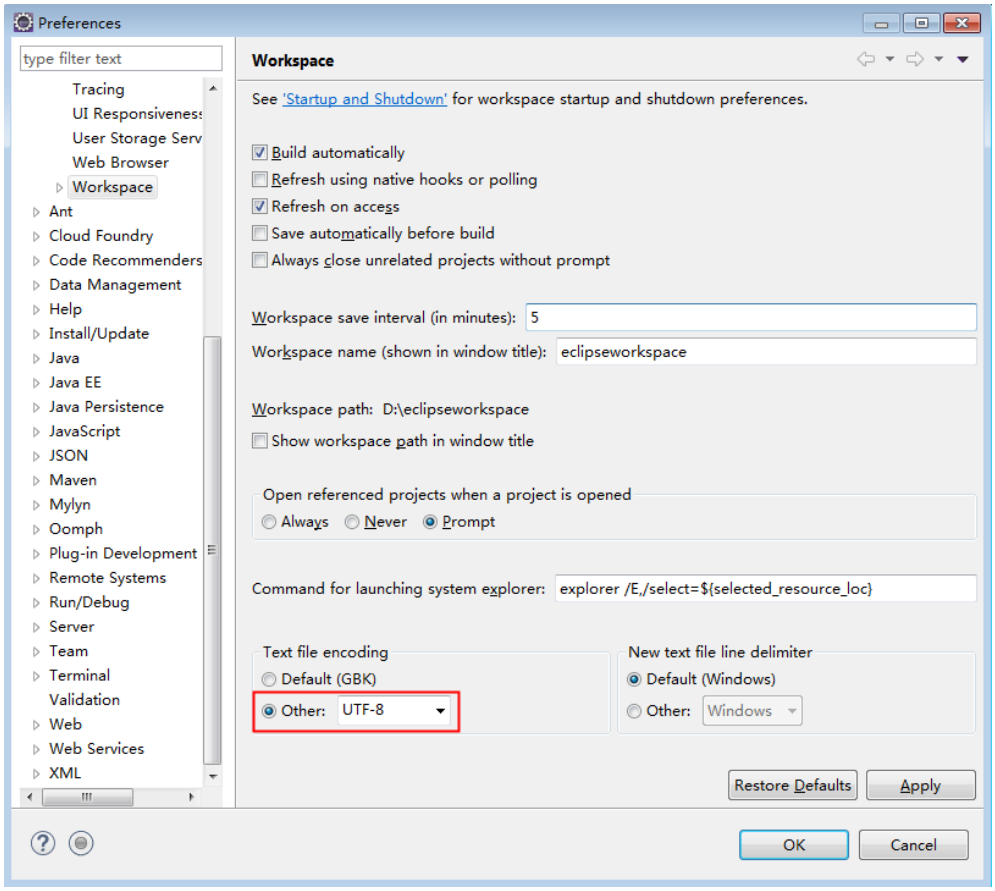
   The **Browse Folder** dialog box is displayed.
3. Select the **hdfs-examples** sample project folder, and click **OK**.
4. In the **New Java Project** window, click **Finish**.
5. Method 2: Open Eclipse, choose **File** > **Import...** > **Existing maven Projects into Workspace** > **Next**, and click **Browse** on the next page. The **Browse Folder** dialog box is displayed.
6. Select the **hdfs-examples** sample project folder, and click **OK**.
7. In the **Import** window, click **Finish**.

Step 3 Set an Eclipse text file encoding format to prevent garbled characters.

1. On the Eclipse menu bar, choose **Window** > **Preferences**.

   The **Preferences** window is displayed.
2. In the navigation pane on the left, choose **General** > **Workspace**. In the **Text file encoding** area, select **Other** and set the value to **UTF-8**. Click **Apply** and then **OK**. Figure 7-3 shows the settings.

Figure 7-3 Setting the Eclipse encoding format



----End

7.3 Developing an HDFS Application

7.3.1 HDFS Development Plan

Scenario Description

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

Service operation objects of HDFS are files. File operations covered by sample codes include creating a folder, writing data to a file, appending file content, reading a file, and deleting a file or folder. You can learn how to perform other operations on the HDFS, such as setting file access permissions, based on sample codes.

Sample codes are described in the following sequence:

1. Initialize HDFS. For details, see **Initializing HDFS**.
2. Write data to a file. For details, see **Writing Data to an HDFS File**.
3. Append file content. For details, see **Appending HDFS File Content**.

4. Read a file. For details, see **Reading an HDFS File**.
5. Delete a file. For details, see **Deleting an HDFS File**.
6. Colocation **HDFS Colocation**
7. Set storage policies. For details, see **Setting HDFS Storage Policies**.
8. Access OBS. For details, see **Using HDFS to Access OBS**.

Development Guidelines

Determine functions to be developed based on the preceding scenario description. The following example describes how to upload, query, append, and delete information about a new employee in seven parts.

1. Pass the Kerberos authentication.
2. Call the mkdir API in fileSystem to create a directory.
3. Call the dowrite API of HdfsWriter to write information.
4. Call the open API in fileSystem to read the file.
5. Call the doAppend API of HdfsWriter to append information.
6. Call the deleteOnExit API in fileSystem to delete the file.
7. Call the delete API in fileSystem to delete the folder.

7.3.2 Initializing HDFS

Function Description

Before using APIs provided by HDFS, you need to initialize HDFS. The HDFS initialization process is as follows:

1. Load HDFS service configuration files and perform Kerberos authentication.
2. Instantiate the Filesystem after the authentication succeeds.
3. Use HDFS APIs.

NOTE

Obtain the keytab file for Kerberos authentication in advance.

Configuration Files

Table 7-4 lists the configuration files used for logging in to HDFS. These files have been imported to the **conf** directory of the **hdfs-example** project.

Table 7-4 Configuration files

File Name	Function	How to Obtain
core-site.xml	Configures HDFS parameters.	MRS_Services_ClientConfig\HDFS \config\core-site.xml
hdfs-site.xml	Configures HDFS parameters.	MRS_Services_ClientConfig\HDFS \config\hdfs-site.xml

File Name	Function	How to Obtain
user.keytab	Provides HDFS user information for Kerberos security authentication.	If the cluster is in security mode, contact the administrator to obtain the **keytab** and **krb5** files corresponding to the account.
krb5.conf	Contains Kerberos server configuration information.	

NOTE

- The **user.keytab** and **krb5.conf** of different clusters cannot be shared.
- The **log4j.properties** file in the **conf** directory is configured based on the customer requirements.

Sample Code

The following provides code snippets. For complete codes, see the **HdfsMain** class in **com.huawei.bigdata.hdfs.examples**.

Run the initialization code of the application on the Linux client. The code example is as follows:

```
/**
  * Initialization. Obtain a FileSystem instance.
  *
  * @throws IOException
  */
 private void init() throws IOException {
   confLoad();
   authentication();
   instanceBuild();
 }

 /**
  *
  * If the application runs on Linux, the paths of core-site.xml and hdfs-site.xml must be
  * modified to the absolute path of the client file on Linux.
  *
  */
 private 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));
  }

 /**
  * kerberos security authentication
  * If the application runs on Linux, the paths of krb5.conf and keytab must be
  * modified to the absolute path of the client file on Linux. In addition, the keytab and principal files in
the sample code must be
  * modified to the current user's keytab file name and username.
  *
  */
 private 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);
   }
```



```
}


/**
 * build HDFS instance
 */
private void instanceBuild() throws IOException {
  // get filesystem
  fSystem = FileSystem.get(conf);
}
```

On Linux, the login sample code is required for the first login. For details about the code, see the **LoginUtil** class in **com.huawei.bigdata.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. Check whether login is required.
  if (checkNeedLogin(userPrincipal)) {

  // 5. Log in to Hadoop and perform a check.
    loginHadoop(userPrincipal, userKeytabFile.getAbsolutePath());
  }

  // 6. Check and log in again.
  checkAuthenticateOverKrb();
  System.out.println("Login success!!!!!!!!!!!!!!!");
}
```

7.3.3 Writing Data to an HDFS File

Function Description

The process of writing data to a file is as follows:

1. Instantiate a FileSystem.
2. Use the FileSystem instance to obtain various types of resources for writing data to files.
3. Write the data to a specified file in HDFS.

NOTE

Close all requested resources after writing data to the file.

Sample Code

The following provides code snippets for writing data to a file. For complete codes, see the **HdfsMain** and **HdfsWriter** classes in **com.huawei.bigdata.hdfs.examples**.

```
/**
 * Create a file and write data to the file.
 *
 * @throws IOException
 * @throws ParameterException
 */
private void write() throws IOException, ParameterException {
  final String content = "hi, I am bigdata. It is successful if you can see me.";
  InputStream in = (InputStream) new ByteArrayInputStream(
content.getBytes());
  try {
    HdfsWriter writer = new HdfsWriter(fSystem, DEST_PATH
        + File.separator + FILE_NAME);
    writer.doWrite(in);
   System.out.println("success to write.");
  } finally {
    // Stream resources must be closed.
    close(in);
  }
}
```

7.3.4 Appending HDFS File Content

Function Description

Append specific content to a specified file in HDFS. The process is as follows:

1. Instantiate a FileSystem.
2. Use the FileSystem instance to obtain various related resources.
3. Add the content to the specified file in HDFS.

NOTE

Close all requested resources after appending the file content.

Sample Code

The following provides code snippets. For complete codes, see the **HdfsMain** and **HdfsWriter** classes in **com.huawei.bigdata.hdfs.examples**.

```
/**
* Append file content.
*
* @throws IOException
*/
private void append() throws Exception {
    final String content = "I append this content.";
    InputStream in = (InputStream) new ByteArrayInputStream(
    content.getBytes());
  try {
    HdfsWriter writer = new HdfsWriter(fSystem, DEST_PATH
       + File.separator + FILE_NAME);
    writer.doAppend(in);
   System.out.println("success to append.");
   } finally {
// Stream resources must be closed.
   close(in);
   }
}
```

7.3.5 Reading an HDFS File

Function Description

Read data from a specified file in HDFS.

NOTE

Close all requested resources after reading a file.

Sample Code

The following provides code snippets for reading a file. 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 data to a file.
    while ((sTempOneLine = reader.readLine()) != null) {
      strBuffer.append(sTempOneLine);
    }

    System.out.println("result is : " + strBuffer.toString());
    System.out.println("success to read.");

  } finally {
    // Resources must be closed.
    close(reader);
    close(in);
  }
}
```

7.3.6 Deleting an HDFS File

Function Description

Delete a specified file or folder from HDFS.

NOTE

The deleted file or folder is stored in the **.Trash/Current** folder in the current user directory. If the file is deleted by mistake, you can restore it from this folder.

Sample Code

The following provides code snippets for deleting a file. 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);
  fSystem.deleteOnExit(beDeletedPath);
  System.out.println("succee to delete the file " + DEST_PATH
      + File.separator + FILE_NAME);
}
```

7.3.7 HDFS 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 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.

Before using the Colocation function, you are advised to be familiar with the internal mechanisms of Colocation, including:

- **Colocation node allocation principles**

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 use times. Based on the number of use times, Colocation sorts the data nodes. The data nodes that are rarely used are placed at the beginning and selected first. The count increase by 1 each time after a node is selected. The nodes are sorted again, and the subsequent node will be selected.

- **Capacity expansion and Colocation allocation**

  After cluster capacity expansion, you can select one of the following two policies shown in **Table 7-5** 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 7-5 Allocation policies

No.	Policy	Description
1	Delete the original locators and create new locators for all data nodes in the cluster.	1. The original locators before the capacity expansion evenly use 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 7-6** summarizes the two usage principles to ensure even data node capacity.

Table 7-6 Usage principles

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.	-

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 times, 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 MRS 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.

Sample Code

For complete sample codes, see **com.huawei.bigdata.hdfs.examples.ColocationExample**.

NOTE

When running the Colocation project, you need to bind the HDFS user to the **supergroup** user group.

1. Initialization

   Kerberos security authentication is required before using Colocation.

   ```
   private static void init() throws IOException {
       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 you perform operations, such as creating a group.

   ```
   public static void main(String[] args) throws IOException {
       init();
       dfsAdmin = new DFSColocationAdmin(conf);
       dfs = new DFSColocationClient();
   ```



```
  dfs.initialize(URI.create(conf.get("fs.defaultFS")), conf);
  createGroup();
  put();
  delete();
  deleteGroup();
  dfs.close();
  dfsAdmin.close();
 }
```

3. Create a group.

   Example: Create the **gid01** group, which contains three locators.

```
 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.

```
 private static void put() throws IOException {
  FSDataOutputStream out = dfs.create(new Path("/testfile.txt"), true,
      COLOCATION_GROUP_GROUP01, "lid01");
  // Data to be written to 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.

```
 public static void delete() throws IOException {
  dfs.delete(new Path("/testfile.txt"));
 }
```

6. Delete a group.

   Example: Delete **gid01**.

```
 private static void deleteGroup() throws IOException {
  dfsAdmin.deleteColocationGroup(COLOCATION_GROUP_GROUP01);
 }
```

7.3.8 Setting HDFS Storage Policies

Function Description

Specify storage policies for a file or folder in HDFS.

Sample Code

1. Set the following parameters in the **Hdfs-site.xml** file in **${HADOOP_HOME}/etc/hadoop/**.

```
<name>dfs.storage.policy.enabled</name>
<value>true</value>
```

2. Restart HDFS, as shown in **Figure 7-4**.

   Figure 7-4 Restarting HDFS

3. Log in to MRS Manager. Choose **Service** > **HDFS** > **Service Configuration**, and set **Type** to **All**.

4. Check whether the value of **dfs.storage.policy.enabled** is **true**. If it is not, modify the value to **true**, click **Save Configuration**, and restart HDFS.
5. View the code.

   The following provides code snippets. For complete codes, see the **HdfsMain** class in **com.huawei.bigdata.hdfs.examples**.

```
/**
* Set storage policies.
* @param policyName
* The 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;
      }
      System.out.println("StoragePolicy:" + bs.getName());
    }
    if (!flag) {
      policyName = storage[0].getName();
    }
    dfs.setStoragePolicy(destPath, policyName);
    System.out.println("success to set Storage Policy path " + DEST_PATH);
    rmdir();
  }
  else{
    System.out.println("Storage Policy is only supported in HDFS.");
  }
}
```

7.3.9 Using HDFS to Access OBS

Function Description

The OBS access process is as follows:

1. Set **fs.obs.access.key** and **fs.obs.secret.key**.
2. Use the FileSystem instance to read, add, and delete various resources.

NOTE

The appending operation is not supported.

Prerequisites

Before interconnecting with OBS, create related directories in OBS, and ensure you have the permission to access and operate the the directories.

Sample Code

The following provides code snippets for instantiating the FileSystem. For complete codes, see the **HdfsMain** class in **com.huawei.bigdata.hdfs.examples**.

```
/**
  *
  * Add configuration file if the application run on the linux ,then need make
  * the path of the core-site.xml and hdfs-site.xml to in the linux client file
  *
  */
 private 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.set("fs.obs.access.key","*** Provide your Access Key ***");
   conf.set("fs.obs.secret.key","*** Provide your Secret Key ***");
   }

 /**
  * build HDFS instance
  */
 private void instanceBuild() throws IOException {
   // get filesystem
// fSystem = FileSystem.get(conf);
   fSystem =  FileSystem.get(URI.create("obs://[BuketName]"),conf);
 }
```

7.4 Commissioning an HDFS Application

7.4.1 Commissioning an HDFS Application on Linux

Scenario

HDFS applications can run in a Linux environment where an HDFS client is installed. After application code development is complete, you can upload a JAR file to the Linux client to run applications.

NOTE

HDFS applications can run only on Linux, but not on Windows.

Prerequisites

- You have installed an HDFS 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 locates. The host names and IP addresses must be mapped one by one.

Procedure

Step 1 Run the **mvn package** command to generate a JAR file, for example, **hdfs-examples-1.0.jar**, and obtain it from the **target** directory in the project directory.

Step 2 Upload the exported JAR file to any directory in the Linux client operating environment, for example, **/opt/client**. Create the **conf** directory in the directory,

and copy the **user.keytab** and **krb5.conf** files to the **conf** directory. For details, see **Step 6**.

Step 3 Run the following command to set the environment variables:

source /opt/client/bigdata_env

Step 4 Run the following commands to run the JAR file:

hadoop jar hdfs-examples-1.0.jar com.huawei.bigdata.hdfs.examples.HdfsMain

NOTE

Before you run the commands, ensure that the Kerberos information is consistent between **Yarn/config/hdfs-site.xml** and **HDFS/hadoop/etc/hadoop/hdfs-site.xml**. Modify **mapred** in **hdfs-site.xml** to **hdfs**. **Figure 7-5** lists points that need to be modified.

Figure 7-5 hdfs-site.xml

```
<name>dfs.datanode.kerberos.https.principal</name>
<value>mapred/hadoop.hadoop.com@HADOOP.COM</value>
<name>dfs.namenode.kerberos.https.principal</name>
<value>mapred/hadoop.hadoop.com@HADOOP.COM</value>
<name>dfs.datanode.kerberos.principal</name>
<value>mapred/hadoop.hadoop.com@HADOOP.COM</value>
<name>dfs.namenode.kerberos.principal</name>
<value>mapred/hadoop.hadoop.com@HADOOP.COM</value>
```

----End

7.4.2 Viewing the HDFS Application Commissioning Result

Scenario

After HDFS application running is complete, you can obtain the running status by viewing the running result or HDFS logs.

Procedure

1. **Viewing running results to learn application running status**
   - The following provides the running result of the HdfsMain Linux sample application in the security cluster:

```
[root@node-master1dekG client]# hadoop jar hdfs-examples-1.0.jar
com.huawei.bigdata.hdfs.examples.HdfsMain
WARNING: Use "yarn jar" to launch YARN applications.
20/03/25 16:29:45 INFO security.UserGroupInformation: Login successful for user hdfsuser using
keytab file user.keytab
20/03/25 16:29:45 INFO security.LoginUtil: Login success!!!!!!!!!!!!!!
success to create path /user/hdfs-examples
success to write.
result is : hi, I am bigdata. It is successful if you can see me.
success to read.
success to delete the file /user/hdfs-examples/test.txt
success to delete path /user/hdfs-examples
success to create path /user/hdfs-examples
StoragePolicy:FROZEN
StoragePolicy:COLD
StoragePolicy:WARM
```



```
StoragePolicy:HOT
StoragePolicy:ONE_SSD
StoragePolicy:ALL_SSD
StoragePolicy:LAZY_PERSIST
succee to set Storage Policy path /user/hdfs-examples
success to delete path /user/hdfs-examples
```

- The following provides the running result of the HdfsMain Linux sample application in the normal cluster:

```
[root@node-master2VknR client]# hadoop jar hdfs-examples-1.0.jar
com.huawei.bigdata.hdfs.examples.HdfsMain
WARNING: Use "yarn jar" to launch YARN applications.
success to create path /user/hdfs-examplessuccess to write.
result is : hi, I am bigdata. It is successful if you can see me.
success to read.
success to delete the file /user/hdfs-examples/test.txt
success to delete path /user/hdfs-example
ssuccess to create path /user/hdfs-examples
StoragePolicy:FROZEN
StoragePolicy:COLD
StoragePolicy:WARM
StoragePolicy:HOT
StoragePolicy:ONE_SSD
StoragePolicy:ALL_SSD
StoragePolicy:LAZY_PERSIST
succee to set Storage Policy path /user/hdfs-examples
success to delete path /user/hdfs-examples
```

2. **Viewing HDFS logs to learn application running status**

   View HDFS namenode logs to learn application running status, and adjust applications based on log information.

7.5 FAQs About HDFS Application Development

7.5.1 HDFS Java APIs

For details about HDFS APIs, see **http://hadoop.apache.org/docs/r2.7.2/api/index.html**.

Common HDFS APIs

Common HDFS Java classes are as follows:

- FileSystem: is the core class of client applications. For details about its common APIs, see **Table 7-7**.
- FileStatus: records the status of files and directories. For details about its common APIs, see **Table 7-8**.
- DFSColocationAdmin: used to manage Colocation group information. For details about its common APIs, see **Table 7-9**.
- DFSColocationClient: used to operate Colocation files. For details about its common APIs, see **Table 7-10**.

NOTE

- The system reserves only the mapping between nodes and locator IDs, but does not to reserve the mapping between files and locator IDs. When a file is created by using a Colocation API, the file is created on the node that corresponds to a locator ID. File creation and writing must use Colocation APIs.
- After the file is written, subsequent operations on the file can use other open source APIs in addition to Colocation APIs.
- The DFSColocationClient class inherits from the open source DistributedFileSystem class, including common APIs. You are advised to use DFSColocationClient to perform operations related to Colocation files.

Table 7-7 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.
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 FileStatus object records status information of the file or folder, including the modification time and file directory.
public BlockLocation[] getFileBlockLocations(FileStatus file, long start, long len)	This API is used to query the block location of a specified file in an HDFS cluster. **file** indicates a complete file path, and **start** and **len** specify the block scope.

API	Description
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 7-8 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 7-9 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.

API	Description
public List<String> listColocationGroups()	This API is used to return all group information of Colocation. The returned group ID arrays are sorted by 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 7-10 Common DFSColocationClient APIs

API	Description
public FSDataOutputStream create(Path f, boolean overwrite, String groupId,String locatorId)	This API is used to create an FSDataOutputStream in Colocation mode to allow users to write files to the **f** path. **f** is the HDFS path. **overwrite** indicates whether an existing file can be overwritten. **groupId** and **locatorId** of the file specified by a user must exist.
public FSDataOutputStream create(final Path f, final FsPermission permission, final EnumSet<CreateFlag> cflags, final int bufferSize, final short replication, final long blockSize, final Progressable progress, final ChecksumOpt checksumOpt, final String groupId, final String locatorId)	The function of this API is the same as that of **FSDataOutputStream create(Path f, boolean overwrite, String groupId, String locatorId)** except that users are allowed to customize checksum.
public void close()	This API is used to close the connection.

Table 7-11 HDFS client WebHdfsFileSystem API

API	Description
public RemoteIterator<FileStatus> listStatusIterator(final Path)	This API will help fetch the subfiles and folders information through multiple requests using remote iterator, thereby avoiding the user interface from becoming slow when there are plenty of child information to be fetched.

Using API-based Glob Path Mode to Obtain LocatedFileStatus and Open Files from FileStatus

The following APIs are added to DistributedFileSystem to obtain the FileStatus with a block location and open the file from the FileStatus object. These APIs reduce the number of RPC calls from clients to the NameNode.

Table 7-12 FileSystem APIs

API	Description
public LocatedFileStatus[] globLocatedStatus(Path, PathFilter, boolean) throws IOException	A LocatedFileStatus object array is returned. The corresponding file path complies with the path filtering rule.
public FSDataInputStream open(FileStatus stat) throws IOException	If the **stat** object is an instance of LocatedFileStatusHdfs and the instance has location information, InputStream is directly created without contacting the NameNode.

7.5.2 HDFS C APIs

Function Description

In the C language application development sample code, file operations include creating, reading, writing, appending, and deleting files. For details about related APIs, visit **http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/LibHdfs.html**.

Sample Code

The following code snippets are used as an example. For complete code, see the HDFS C sample code in the hdfs_test.c**MRS_Services_ClientConfig/HDFS/hdfs-c-example/hdfs_test.c** file.

1. Configure the HDFS NameNode parameters and create a connection to HDFS.

```
hdfsFS fs = hdfsConnect("default", 0);
fprintf(stderr, "hdfsConnect- SUCCESS!\n");
```


2. Create an HDFS directory.

```
const char* dir = "/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 data to a file.

```
const char* file = "/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 a file.

```
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. Specify a location to start to read a 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 a file.

```
const char* destfile = "/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 a file.

```
const char* mvfile = "/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 a file.

```
const char* renamefile = "/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 a file.

```
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 file replicas.

```
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. Configure a user and a user group.

```
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. Obtain 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 API

```
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);
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);
free(buffer);
```

17. Disconnect HDFS.

```
hdfsDisconnect(fs);
```

Preparing an Operating Environment

Install the client on the node, for example, to the **/opt/client** directory. For details about how to install the client, see **Client Management** in the *MapReduce Service User Guide*.

1. You have installed HDFS on the server and confirmed that HDFS is running properly.
2. The JDK 1.7 or 1.8 has been installed on the client.
3. Obtain the **MRS_HDFS_Client.tar** installation package. Run the following commands to decompress the package:

   tar -xvf MRS_HDFS_Client.tar

   tar -xvf MRS_HDFS_ClientConfig.tar

   > NOTE
   >
   > You are advised to install a client of the same version as the cluster on the server to avoid version incompatibility.

4. Go to the **MRS_HDFS_ClientConfig** decompressed folder and run the following command to install the client:

   sh *install.sh* /opt/client

   In the preceding command, **/opt/client** is an example user-defined path.

5. Go to the **/opt/client** client installation directory and run the following command to initialize the environment variables:

   source *bigdata_env*

Compiling and Running Applications on Linux

1. Go to the Linux client directory and run the following commands to import public environment variables:

   cd */opt/client*

   source *bigdata_env*

2. Run the following command as user **hdfs** to perform command line authentication:

   kinit *hdfs*

> **NOTE**
>
> The validity period of once **kinit** is 24 hours. Run the **kinit** command again when you run the sample application 24 hours later.

3. Go to the **/opt/client/HDFS/hadoop/hdfs-c-example** directory and run the following commands to import the environment variables of client C:

   cd */opt/client/HDFS/hadoop/hdfs-c-example*

   source *component_env_C_example*

4. Run the following command to clear the generated target files and executable files:

   make clean

   The command output is 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 generate a new target and an executable file:

   make (or ***make all***)

   The command output is as follows:

```
[root@10-120-85-2 hdfs-c-example]# make all
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 run the file to create, read, write, append, and delete the file:

   make run

   The command output is as follows:

```
[root@10-120-85-2 hdfs-c-example]# make run
./hdfs_test
hdfsConnect- SUCCESS!
hdfsCreateDirectory- SUCCESS! : /nativeTest
hdfsOpenFile- SUCCESS! for write : /nativeTest/testfile.txt
hdfsWrite- SUCCESS! : /nativeTest/testfile.txt
Flushing file data ....
hdfsCloseFile- SUCCESS! : /nativeTest/testfile.txt
hdfsOpenFile- SUCCESS! for read : /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 fi|2* rite!
hdfsCopy- SUCCESS!, File copied, src : /nativeTest/testfile.txt, des : /nativeTest/testfile1.txt
hdfsMove- SUCCESS!, File moved, src : /nativeTest/testfile1.txt, des : /nativeTest/testfile2.txt
hdfsRename- SUCCESS!, File renamed, Old name : /nativeTest/testfile2.txt, New name : /nativeTest/
testfile3.txt
hdfsDelete- SUCCESS!, File deleted : /nativeTest/testfile3.txt
hdfsSetReplication- SUCCESS!, Set replication 10 for /nativeTest/testfile.txt
hdfsChown- SUCCESS!, Chown success for /nativeTest/testfile.txt
hdfsChmod- SUCCESS!, Chmod success for /nativeTest/testfile.txt
hdfsUtime- SUCCESS!, Set time success for /nativeTest/testfile.txt
Name: hdfs://hacluster/nativeTest/testfile.txt, Type: F, Replication: 10, BlockSize: 134217728, Size: 31,
LastMod: 1480589792, Owner: root, Group: root, Permissions: 511 (rwxrwxrwx)
hdfsGetPathInfo - SUCCESS!
Name: hdfs://hacluster/nativeTest/testfile.txt, Type: F, Replication: 10, BlockSize: 134217728, Size: 31,
LastMod: 1480589792, Owner: root, Group: root, Permissions: 511 (rwxrwxrwx)
hdfsListDirectory- SUCCESS!, /nativeTest
hdfsTruncateFile- SUCCESS!, /nativeTest/testfile.txt
Block Size : 134217728
hdfsGetDefaultBlockSize- SUCCESS!
Block Size : 134217728 for file /nativeTest/testfile.txt
hdfsGetDefaultBlockSizeAtPath- SUCCESS!
HDFS Capacity : 1569475438758
hdfsGetCapacity- SUCCESS!
```



```
HDFS Used : 1122248
hdfsGetCapacity- SUCCESS!
hdfsExists- SUCCESS! /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 : 31,
totalShortCircuitBytesRead : 0, totalZeroCopyBytesRead : 0
[root@10-120-85-2 hdfs-c-example]# make run
./hdfs_test
hdfsConnect- SUCCESS!
hdfsCreateDirectory- SUCCESS! : /nativeTest
hdfsOpenFile- SUCCESS! for write : /nativeTest/testfile.txt
hdfsWrite- SUCCESS! : /nativeTest/testfile.txt
Flushing file data ....
hdfsCloseFile- SUCCESS! : /nativeTest/testfile.txt
hdfsOpenFile- SUCCESS! for read : /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 : /nativeTest/testfile.txt, des : /nativeTest/testfile1.txt
hdfsMove- SUCCESS!, File moved, src : /nativeTest/testfile1.txt, des : /nativeTest/testfile2.txt
hdfsRename- SUCCESS!, File renamed, Old name : /nativeTest/testfile2.txt, New name : /nativeTest/
testfile3.txt
hdfsDelete- SUCCESS!, File deleted : /nativeTest/testfile3.txt
hdfsSetReplication- SUCCESS!, Set replication 10 for /nativeTest/testfile.txt
hdfsChown- SUCCESS!, Chown success for /nativeTest/testfile.txt
hdfsChmod- SUCCESS!, Chmod success for /nativeTest/testfile.txt
hdfsUtime- SUCCESS!, Set time success for /nativeTest/testfile.txt

Name: hdfs://hacluster/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/nativeTest/testfile.txt, Type: F, Replication: 10, BlockSize: 134217728, Size: 31,
LastMod: 1500345260, Owner: root, Group: root, Permissions: 511 (rwxrwxrwx)
hdfsListDirectory- SUCCESS!, /nativeTest
hdfsTruncateFile- SUCCESS!, /nativeTest/testfile.txt
Block Size : 134217728
hdfsGetDefaultBlockSize- SUCCESS!
Block Size : 134217728 for file /nativeTest/testfile.txt
hdfsGetDefaultBlockSizeAtPath- SUCCESS!
HDFS Capacity : 102726873909
hdfsGetCapacity- SUCCESS!
HDFS Used : 4767076324
hdfsGetCapacity- SUCCESS!
hdfsExists- SUCCESS! /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. (Optional) Enter the debug mode.

   make gdb

   Before running this command, you need to install the GDB. For details, see **Installing GDB**.

   The command output is 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/hadoop-client/HDFS/hadoop/hdfs-c-example/hdfs_test...done.
(gdb)
[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)
```

Installing GDB

1. Download the source code of the termcap package on which GDB depends.

   wget https://ftp.gnu.org/gnu/termcap/termcap-1.3.1.tar.gz
2. Decompress the termcap source code.

   tar -zxvf termcap-1.3.1.tar.gz
3. Compile and install termcap.

   cd termcap-1.3.1/

   ./configure && make && make install
4. Download the GDB source code.

   cd ~

   wget https://ftp.gnu.org/gnu/gdb/gdb-7.6.1.tar.gz
5. Decompress the GDB source code.

   tar -zxvf gdb-7.6.1.tar.gz
6. Compile and install the GDB.

   cd gdb-7.6.1/

   ./configure && make && make install
7. Check whether the GDB installation is successful.

   gdb --version

   If the GDB version information is displayed, the installation is successful.

7.5.3 HDFS HTTP REST APIs

Function Description

In the REST application development sample code, file operations include creating, reading, writing, appending and deleting files. For details about related APIs, visit **http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/WebHDFS.html**.

Preparing an Operating Environment

Step 1 Install a client. Install the client on the node, for example, to the **/opt/client** directory. For details about how to install the client, see **Client Management** in the *MapReduce Service User Guide*.

1. You have installed HDFS on the server and confirmed that HDFS is running properly.
2. The JDK 1.7 or 1.8 has been installed on the client.
3. Obtain the **MRS_HDFS_Client.tar** installation package, and run the following commands to decompress the package:

   tar -xvf MRS_HDFS_Client.tar

   tar -xvf MRS_HDFS_ClientConfig.tar

   NOTE

   You are advised to install a client of the same version as the cluster on the server to avoid version incompatibility.
4. Go to the **MRS_HDFS_ClientConfig** decompressed folder and run the following command to install the client:

   sh *install.sh* /opt/client

   In the preceding command, **/opt/client** is an example user-defined path.
5. Go to the **/opt/client** client installation directory and run the following command to initialize the environment variables:

   source *bigdata_env*
6. Run the following command to perform user authentication. The following uses user **hdfs** as an example. You can change the username based on the site requirements (skip this step for a normal cluster).

   kinit *hdfs*

   NOTE

   The validity period of once **kinit** is 24 hours. Run the **kinit** command again when you run the sample application 24 hours later.
7. Run the following commands to prepare the **testFile** and **testFileAppend** files in the client directory. Their contents are "Hello, webhdfs user!" and "Welcome back to webhdfs!", respectively.

   touch *testFile*

   vi testFile

   Write "Hello, webhdfs user!". Save the file and exit.

   touch *testFileAppend*

   vi *testFileAppend*

   Write "Welcome back to webhdfs!". Save the file and exit.

Step 2 MRS clusters support only HTTPS-based access by default. If the HTTPS service is used, go to **Step 3**. If the HTTP service is used (supported only by security clusters), go to **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** supports only the TLSv1 protocol, perform the following steps:

Log in to MRS Manager. Choose **Service** > **HDFS** > **Service Configuration**, and set **Type** to **All**. Search for **hadoop.ssl.enabled.protocols** in the search box, and check whether the parameter value contains **TLSv1**. If it does not contain **TLSv1**, add **TLSv1** to the **hadoop.ssl.enabled.protocols** configuration item. Clear the value of **ssl.server.exclude.cipher.list**. Otherwise, HDFS cannot be accessed using HTTPS. Click **Save Configuration** and select **Restart the affected services or instances**. Click **Yes** and restart the HDFS service.

NOTE

TLSv1 has security vulnerabilities. Exercise caution when using it.

Step 4 Log in to **MRS Manager**. Choose **Service** > **HDFS** > **Service Configuration**, and set **Type** to **All**. Search for **dfs.http.policy** in the search box, and select **HTTP_AND_HTTPS**. Click **Save Configuration** and select **Restart the affected services or instances**. Click **Yes** and restart the HDFS service.

----End

Procedure

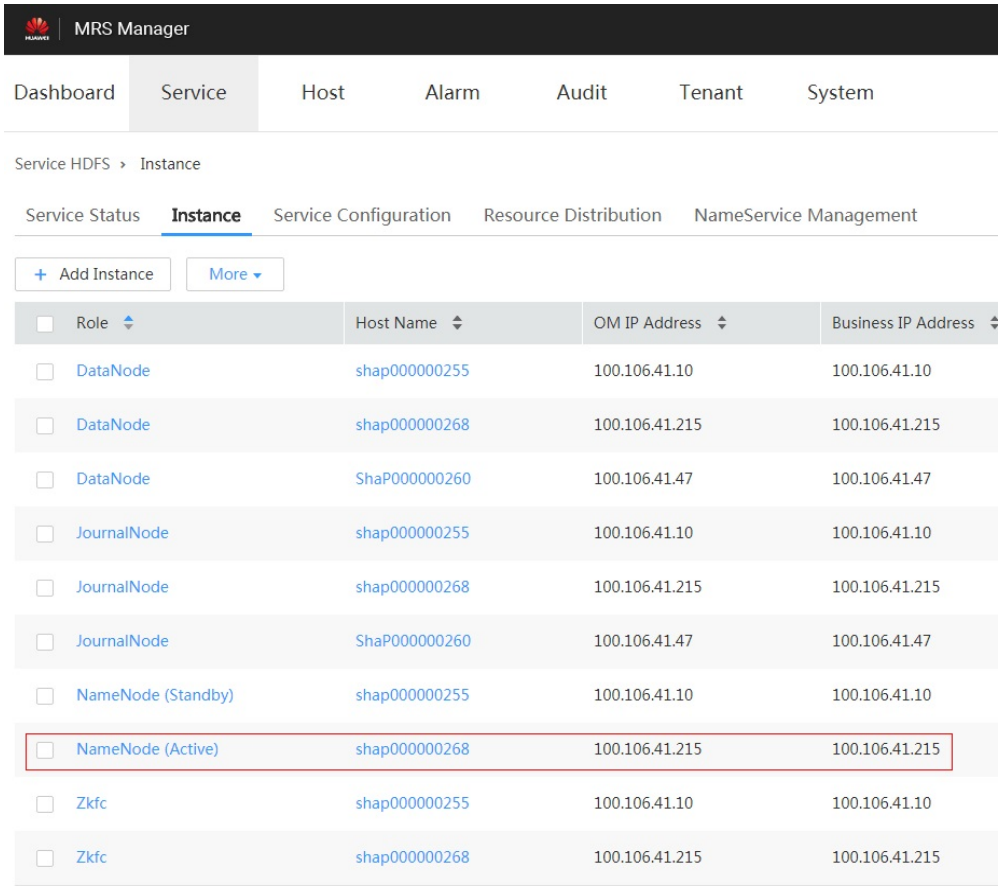
Step 1 Log in to **MRS Manager**, and click **Services**. Select **HDFS** and click it to access the HDFS service status page.

NOTE

Because **webhdfs** is accessed through HTTP and HTTPs, you need to obtain the IP address and HTTP and HTTPs ports of the active NameNode.

1. Click **Instance** to access the page displayed in **Figure 7-6**. Find the host name and IP address of **NameNode(hacluster,active)**.

Figure 7-6 HDFS instance

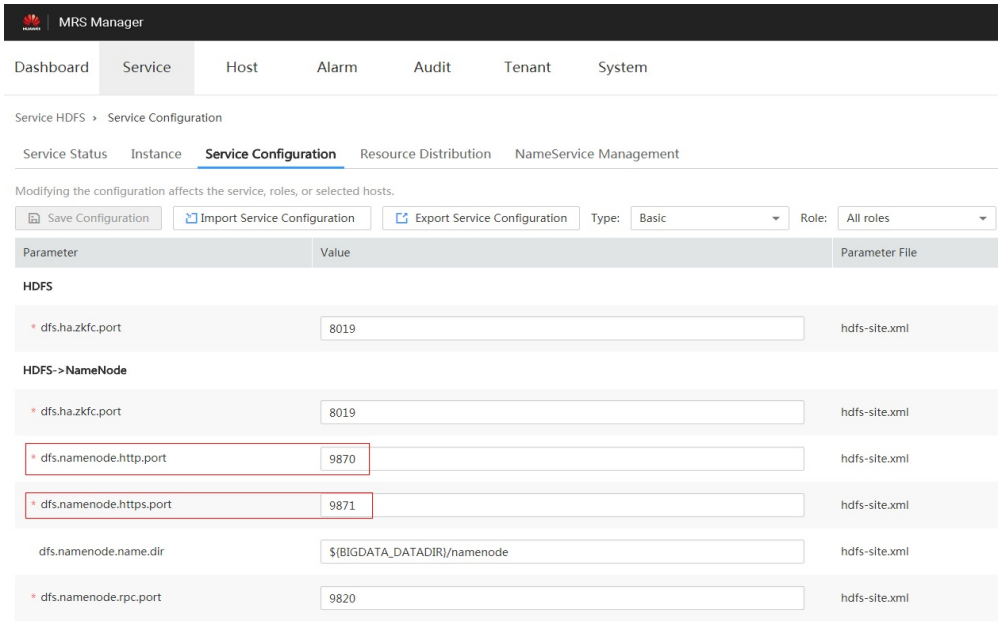


2. Click **Service Configuration** to access the page displayed in **Figure 7-7**. Find **namenode.http.port** (9870) and **namenode.https.port** (9871).

   NOTE

   For versions earlier than MRS 1.9.2, the default values of the preceding ports are **25002** and **25003**. For details, see the related port information in *MapReduce Service User Guide*.

Figure 7-7 HDFS service configuration

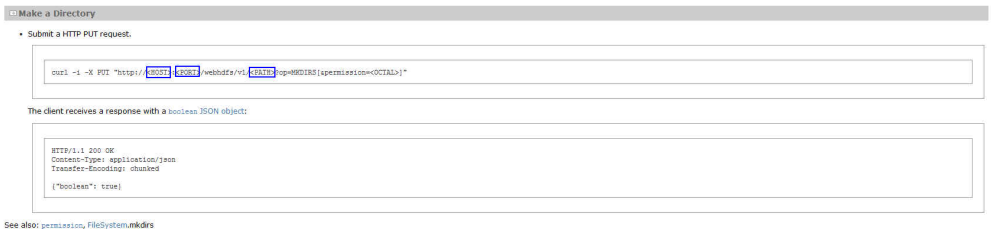


Step 2 Create a directory by referring to the following link.

http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/WebHDFS.html#Make_a_Directory

Click the link. The page is displayed, as shown in **Figure 7-8**.

Figure 7-8 Example command for creating a directory



Go to the installation directory of the client, for example, **/opt/client**, and create a directory named **huawei**.

1. Run the following command to check whether the directory named **huawei** exists:

   hdfs dfs -ls /

   The command output is 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 7-8** to create a directory named **huawei**. Replace <HOST> and <PORT> in the command with the host name or IP address and port number obtained in **Step 1**, and enter the **huawei** directory to be created in <PATH>.

   NOTE

   Replace <HOST> with the host name or IP address. Note that the HTTP and HTTPS ports are different.

   - 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 preceding information, **linux1** indicates <HOST> and **9870** indicates <PORT>.

   - Command output

     ```
     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@HADOOP.COM&t=kerberos&e=1462453809395&s=wiRF4rdTWpm
     3tDST+a/Sy0lwgA4="; Path=/; Expires=Thu, 05-May-2016 13:10:09 GMT; HttpOnly
     Transfer-Encoding: chunked
     {"boolean":true}linux1:/opt/client #
     ```

     The return value **{"boolean" :true}** indicates that the **huawei** directory is successfully created.

   - Run the following command to access HTTPS:

     ```
     linux1:/opt/client # curl -i -k -X PUT --negotiate -u: "https://linux1:9871/webhdfs/v1/huawei?op=MKDIRS"
     ```

     In the preceding information, **linux1** indicates <HOST> and **9871** indicates <PORT>.

   - Command output

     ```
     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@HADOOP.COM&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 #
```

The return value **{"boolean" :true}** indicates that 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 an upload request to obtain the location where the DataNode 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"
```

In the preceding information, **linux1** indicates <HOST> and **9870** indicates <PORT>.

- Command output

```
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@HADOOP.COM&t=kerberos&e=1462464588403&s=qry87vAyYzSn9VsS6
Rm6vKLhKeU="; Path=/; Expires=Thu, 05-May-2016 16:09:48 GMT; HttpOnly
```



```
Location: http://linux1:25010/webhdfs/v1/huawei/testHdfs?
op=CREATE&delegation=HgAFYWRtaW4FYWRtaW4AigFUf4lZdIoBVKOV3XQOCBSyXvFAp92alcRs4j-
KNulnN6wUoBJXRUJIREZTIGRlbGVnYXRpb24UMTAuMTIwLjE3Mi4xMDk6MjUwMDA&namenoderpcaddress=hacluster&overwrite=false
Content-Length: 0
```

- Run the following command to access HTTPS:

```
linux1:/opt/client # curl -i -k -X PUT --negotiate -u: "https://linux1:9871/webhdfs/v1/huawei/testHdfs?op=CREATE"
```

  In the preceding information, **linux1** indicates <HOST> and **9871** indicates <PORT>.

- Command output

```
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@HADOOP.COM&t=kerberos&e=1462455978166&s=F4rXUwEevHZze3PR
8TxkzcV7RQQ="; 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&namenoderpcaddress=hacluster&overwrite=false
Content-Length: 0
```

Step 4 Based on the obtained location information, you can create the **/huawei/testHdfs** file in HDFS and upload the content in the local **testFile** file to 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"
```

  In the preceding information, **linux1** indicates <HOST> and **9864** indicates <PORT>.

- Command output

```
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=HgAFYWRtaW4FYWRtaW4AigFUfwX3t4oBVKMSe7cCCBSFJTi9j7X64QwnSz59T
```



```
GFPKFf7GhNTV0VCSERGUyBkZWxlZ2F0aW9uFDEwLjEyMC4xNzIuMTA5OjI1MDAw&namenoderpcaddr
ess=hacluster&overwrite=false"
```

- Command output

```
HTTP/1.1 100 Continue
HTTP/1.1 201 Created
Location: hdfs://hacluster/huawei/testHdfs
Content-Length: 0
Connection: close
```

Step 5 Open the **/huawei/testHdfs** file and read content in the file.

- Run the following command to access HTTP:

```
linux1:/opt/client # curl -L --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs?op=OPEN"
```

  In the preceding information, **linux1** indicates <HOST> and **9870** indicates <PORT>.

- Command output

```
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"
```

  In the preceding information, **linux1** indicates <HOST> and **9871** indicates <PORT>.

- Command output

```
Hello, webhdfs user!
```

Step 6 Create a request to append a file to obtain the location where the DataNode address of **/huawei/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"
```

  In the preceding information, **linux1** indicates <HOST> and **9870** indicates <PORT>.

- Command output

```
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@HADOOP.COM&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"
  ```

  In the preceding information, **linux1** indicates <HOST> and **9871** indicates <PORT>.

- Command output

  ```
  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@HADOOP.COM&t=kerberos&e=1462461641713&s=tGwwOH9scmnNtxP
  jlnu28SFtex0="; 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 Based on the obtained location information, you can append content in the **testFileAppend** file to the **/huawei/testHdfs** file in 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"
  ```

  In the preceding information, **linux1** indicates <HOST> and **9864** indicates <PORT>.

- Command output

  ```
  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_EwtFKKlODK
  QSM2t32CjhNTV0VCSERGUyBkZWxlZ2F0aW9uFDEwLjEyMC4xNzIuMTA5OjI1MDAw&namenoderpcadd
  ress=hacluster"
  ```


In the preceding information, **linux1** indicates <HOST> and **9865** indicates <PORT>.

- Command output

```
HTTP/1.1 100 Continue
HTTP/1.1 200 OK
Content-Length: 0
Connection: close
```

Step 8 Open the **/huawei/testHdfs** file and read all content in the file.

- Run the following command to access HTTP:

```
linux1:/opt/client # curl -L --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs?op=OPEN"
```

In the preceding information, **linux1** indicates <HOST> and **9870** indicates <PORT>.

- Command output

```
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"
```

In the preceding information, **linux1** indicates <HOST> and **9871** indicates <PORT>.

- Command output

```
Hello, webhdfs user!
Welcome back to webhdfs!
```

Step 9 List the detailed information about all directories and files in the **huawei** directory in HDFS.

LISTSTATUS returns information about all subfiles and folders in a request.

- Run the following command to access HTTP:

```
linux1:/opt/client # curl --negotiate -u: "http://linux1:9870/webhdfs/v1/huawei/testHdfs?
op=LISTSTATUS"
```

In the preceding information, **linux1** indicates <HOST> and **9870** indicates <PORT>.

- Command output

```
{"FileStatuses":{"FileStatus":[
{"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergr
oup","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"
```

In the preceding information, **linux1** indicates <HOST> and **9871** indicates <PORT>.

- Command output

```
{"FileStatuses":{"FileStatus":[
{"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergr
oup","length":70,"modificationTime":1462426678379,"owner":"hdfs","pathSuffix":"","permission":"755"
,"replication":3,"storagePolicy":0,"type":"FILE"}
]}}
```

LISTSTATUS along with the size and **startafter** parameters will help fetch the subfiles and folders information through multiple requests, thereby avoiding the user interface from becoming slow when there are plenty 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"
  ```
  In the preceding information, **linux1** indicates <HOST> and **9870** indicates <PORT>.
- Command output
  ```
  {"FileStatuses":{"FileStatus":[
  {"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergroup","length":70,"modificationTime":1462426678379,"owner":"hdfs","pathSuffix":"testHdfs","permission":"755","replication":3,"storagePolicy":0,"type":"FILE"}
  ]}}
  ```
- Run the following command to access HTTPS:
  ```
  linux1:/opt/client # curl -k --negotiate -u: "https://linux1:9871/webhdfs/v1/huawei/?op=LISTSTATUS&startafter=sparkJobHistory&size=1"
  ```
  In the preceding information, **linux1** indicates <HOST> and **9871** indicates <PORT>.
- Command output
  ```
  {"FileStatuses":{"FileStatus":[
  {"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergroup","length":70,"modificationTime":1462426678379,"owner":"hdfs","pathSuffix":"testHdfs","permission":"755","replication":3,"storagePolicy":0,"type":"FILE"}
  ]}}
  ```

Step 10 Delete the **/huawei/testHdfs** file from 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"
  ```
  In the preceding information, **linux1** indicates <HOST> and **9870** indicates <PORT>.
- Command output
  ```
  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@HADOOP.COM&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"
  ```
  In the preceding information, **linux1** indicates <HOST> and **9871** indicates <PORT>.

- Command output

```
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@HADOOP.COM&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 system provides the key management service through HTTP REST APIs. For details about the APIs, visit the following website:

http://hadoop.apache.org/docs/r2.7.2/hadoop-kms/index.html

NOTE

Security hardening has been performed for REST APIs to prevent script injection attacks. The REST APIs cannot be used to create directories and file names that contain the keywords **<script**, **<iframe**, **<frame**, and **javascript:**.

7.5.4 HDFS Shell Commands

HDFS Shell

You can use HDFS shell commands to perform operations on HDFS, such as reading and writing files.

The following describes how to run the HDFS shell commands.

Step 1 Run the following command to initialize environment variables:

source /opt/client/bigdata_env

Step 2 If the Kerberos authentication is enabled for the current cluster, run the following command to authenticate the user. If the Kerberos authentication is disabled for the current cluster, skip this step. The current user is the development user added in **Preparing an HDFS Application Development User**.

Human-machine user: *kinit MRS cluster user*

For example, **kinit hdfsuser**.

Machine-machine user: *kinit -kt Authentication credential path MRS cluster user*

For example: **kinit -kt /opt/user.keytab hdfsuser**.

Step 3 Run the HDFS shell command.

The format of the HDFS shell operation is as follows: **hadoop fs <args>**

Enter the command. Examples:

- Viewing the content in the directory: **hadoop fs -ls /tmp/input/**
- Creating a directory: **hadoop fs -mkdir /tmp/input/new_dir**
- Viewing file content: **hadoop fs -cat /tmp/input/file1**
- Clearing files: **hadoop fs -rm -r /tmp/input/file1**
- Querying help information about the HDFS commands: **hadoop fs --help**

For details about the HDFS commands, visit the following website:

https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FileSystemShell.html

----End

Table 7-13 Commands related to transparent encryption

Scenario	Operation	Command	Description
Hadoop shell command management key	Create a key.	***hadoop key create*** *<keyname> [-cipher <cipher>] [-size <size>] [-description <description>] [-attr <attribute=value>] [-provider <provider>] [-help]*	The **create** subcommand creates a new key for the **name** specified by **<keyname>** in **provider**. The **provider** is specified by the **-provider** parameter. You can use the **-cipher** parameter to define a password. The default password is **AES/CTR/NoPadding**. The default key length is 128. You can use the **-size** parameter to define a key length. Any attribute of the attribute=value type can be defined by the **-attr** parameter. **-attr** can be defined many times for each attribute.
	Perform a rollback.	***hadoop key roll*** *<keyname> [-provider <provider>] [-help]*	The **roll** subcommand creates a new version for the **key** specified in **provider**. The **provider** is specified by the **-provider** parameter.

Scenario	Operation	Command	Description
	Delete a key.	***hadoop key delete*** *<keyname> [-provider <provider>] [-f] [-help]*	The **delete** subcommand deletes all versions of a key. The key is specified by the **<keyname>** parameter in **provider**, and the **provider** is specified by the **-provider** parameter. Unless **-f** is specified, the command needs to be confirmed by the user.
	View a key.	***hadoop key list*** *[-provider <provider>] [-metadata] [-help]*	The **list** subcommand displays all key names in **provider**. The **provider** is configured by the user in the **core-site.xml** or specified by the **-provider** parameter. The -**metadata** parameter displays metadata.

7.5.5 Logging in to MRS Manager

MRS Manager monitors, configures, and manages MRS clusters. You can open the MRS Manager page from the MRS management console.

This section describes how to open MRS Manager.

Log in to MRS Manager.

Step 1 Log in to the MRS management console.

Step 2 On the **Active Clusters** page, click the target cluster name on the right to access the cluster details page.

Step 3 Click **Manage** next to **MRS Manager**. The **Access MRS Manager** page is displayed.

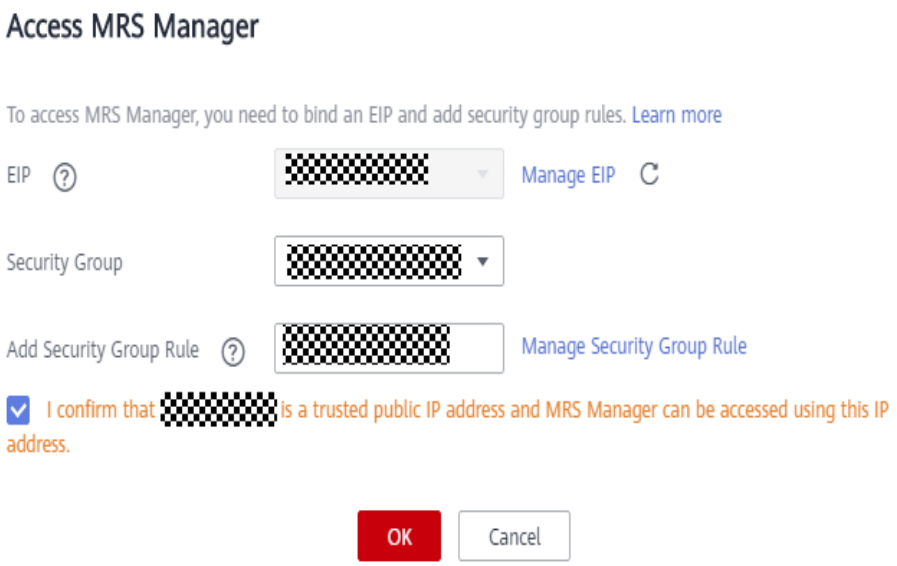
- If you have bound an EIP when creating the cluster, perform the following operations:
  a. Select the security group to which the security group rule to be added belongs. The security group is configured when the cluster is created.
  b. Add a security group rule. By default, your public IP address used for accessing port 9022 is filled in the rule. To enable multiple IP address segments to access MRS Manager, see **Adding a Security Group Rule**. If you want to view, modify, or delete a security group rule, click **Manage Security Group Rule**.

  NOTE

  - It is normal that the automatically generated public IP address is different from the local IP address.
  - If port 9022 is a Knox port, you need to enable the permission of port 9022 to access Knox for accessing MRS Manager.

c. Select the checkbox stating that **I confirm that xx.xx.xx.xx is a trusted public IP address and MRS Manager can be accessed using this IP address**.

Figure 7-9 Adding a security group rule for accessing MRS Manager



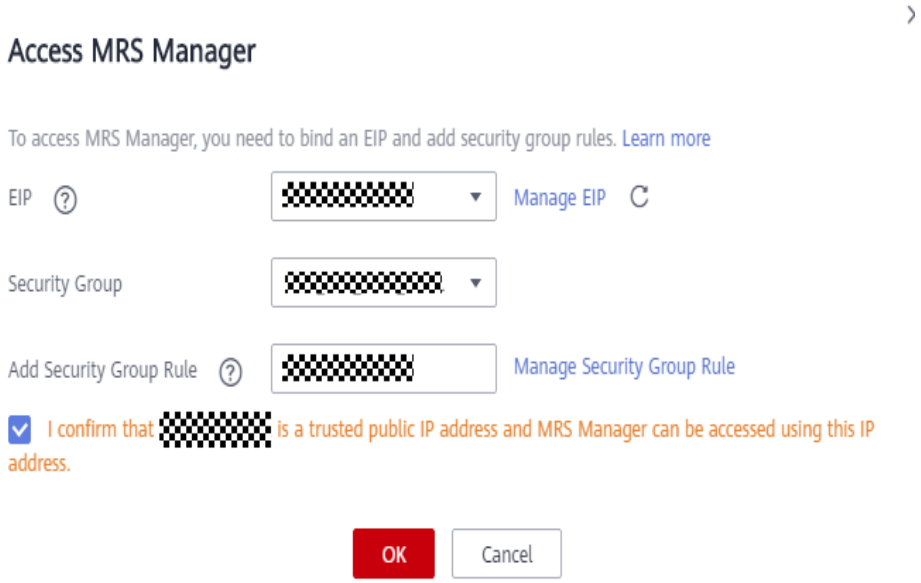
- If you have not bound an EIP when creating the cluster, perform the following operations:
  a. Select an available EIP from the drop-down list or click **Manage EIP** to create one.
  b. Select the security group to which the security group rule to be added belongs. The security group is configured when the cluster is created.
  c. Add a security group rule. By default, your public IP address used for accessing port 9022 is filled in the rule. To enable multiple IP address segments to access MRS Manager, see **Adding a Security Group Rule**. To view, modify, or delete a security group rule, click **Manage Security Group Rule**.

     NOTE

     - It is normal that the automatically generated public IP address is different from the local IP address.
     - If port 9022 is a Knox port, you need to enable the permission of port 9022 to access Knox for accessing MRS Manager.

  d. Select the checkbox stating that **I confirm that xx.xx.xx.xx is a trusted public IP address and MRS Manager can be accessed using this IP address**.

Figure 7-10 Binding an EIP



Step 4 Click **OK**. The MRS Manager login page is displayed.

Step 5 Enter the default username **admin** and password you set when creating the cluster, and click **Log In**. The **MRS Manager** page is displayed.

----End

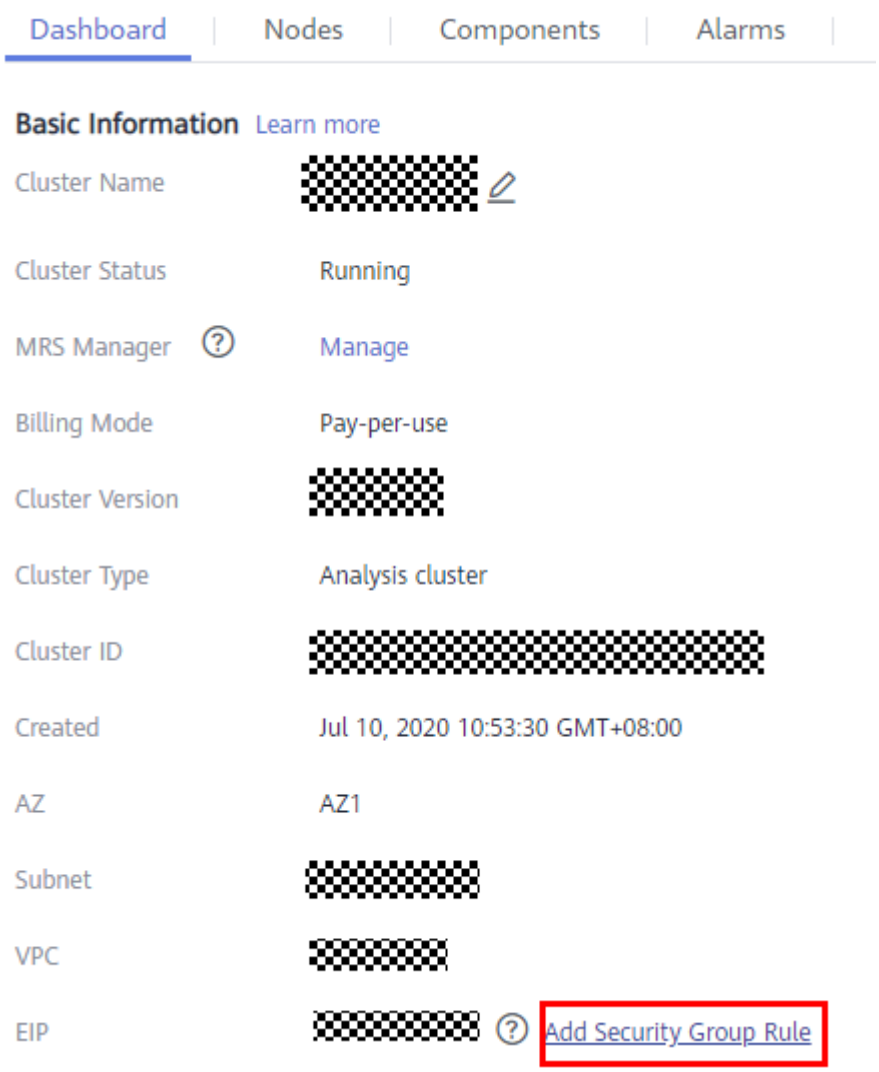
Adding a Security Group Rule

To assign MRS Manager access permissions to other users, follow instructions in this section to add the users' public IP addresses to the trusted range.

Step 1 On the MRS management console, choose **Clusters** > **Active Clusters**. Click the target cluster name to access the cluster details page.

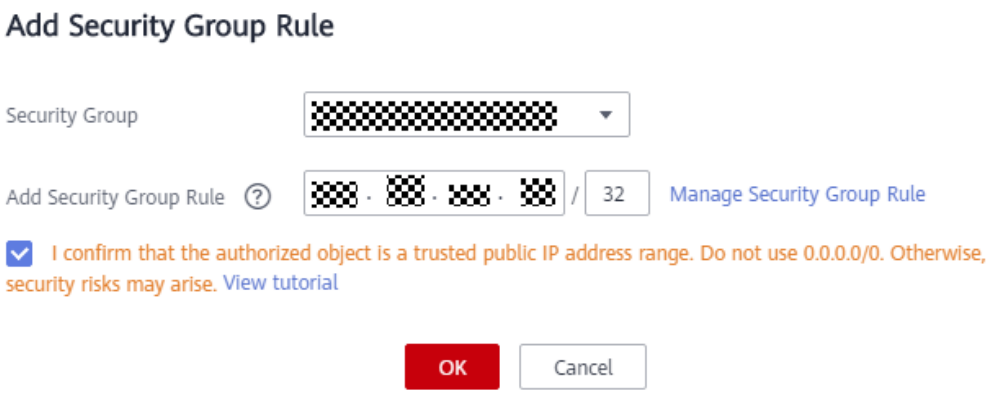
Step 2 Click **Add Security Group Rule** next to **EIP**.

Figure 7-11 Cluster details



Step 3 On the **Add Security Group Rule** page, add the IP address segment for users to access the public network and select **I confirm that the authorized object is a trusted public IP address range. Do not use 0.0.0.0/0**. Otherwise, security risks may arise.

Figure 7-12 Adding a security group rule



By default, the IP address used for accessing the public network is filled. You can change the IP address segment as required. To enable multiple IP address segments, repeat steps 6 to 9. To view, modify, or delete a security group rule, click **Manage Security Group Rule**.

Step 4 Click **OK**.

----End

7.5.6 Downloading an MRS Client

Step 1 Log in to MRS Manager. For details, see **Logging in to MRS Manager**.

Step 2 Choose **Services**.

Step 3 Click **Download Client**.

Step 4 In **Client Type**, select **All client files**.

Step 5 In **Download To**, select **Remote host**.

Step 6 Set **Host IP Address** to the IP address of the newly applied ECS, **Host Port** to **22**, and **Save Path** to **/tmp**.

- If the default port **22** for logging in to an ECS through SSH has been changed, set **Host Port** to a new port.
- The value of **Save Path** contains a maximum of 256 characters.

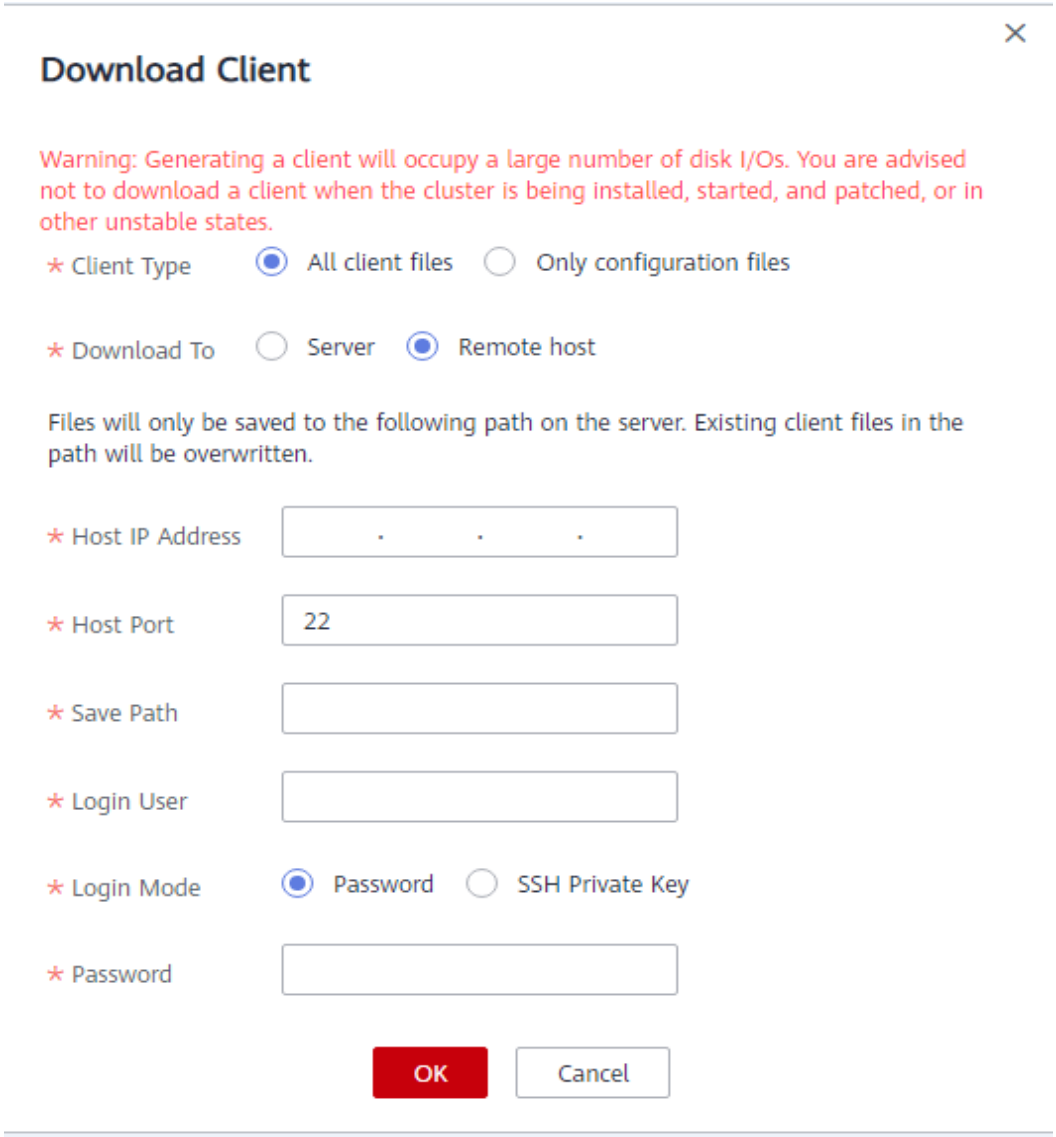
Step 7 Set **Login User** to **root**.

If another user is used, ensure that the user has permissions to read, write, and execute the save path.

Step 8 Select **Password** or **SSH Private Key** for **Login Mode**.

- **Password**: Enter the password of user **root** set during cluster creation.
- **SSH Private Key**: Select and upload the key file used for creating the cluster.

Figure 7-13 Downloading a client



Step 9 Click **OK** to generate a client file.

- If the following information is displayed, the client package is saved. Click **Close**. Obtain the client file from the **Save Path** of the remote host that is set when the client is downloaded.
  Client files downloaded to the remote host successfully.
- If the following information is displayed, check the username, password, and security group configurations of the remote host. Ensure that the username and password are correct and an inbound rule of the SSH (22) port has been added to the security group of the remote host. And then, go to **Step 2** to download the client again.
  Failed to connect to the server. Please check the network connection or parameter settings.

NOTE

Generating a client will occupy a large number of disk I/Os. You are advised not to download a client when the cluster is being installed, started, and patched, or in other unstable states.

----End

8 Hive Development Guide

8.1 Hive Application Development Overview

8.1.1 Introduction to Hive Application Development

Hive

Hive is an open-source data warehouse framework built on Hadoop. It stores structured data and provides basic data analysis services using the Hive query language (HiveQL), a language like the structured query language (SQL). Hive converts HiveQL statements to MapReduce or Spark jobs to query and analyze massive amounts of data stored in Hadoop clusters.

Hive provides the following functions:

- Extracts, transforms, and loads (ETL) data using HiveQL.
- Analyzes massive amounts of structured data using HiveQL.
- Supports various data storage formats, such as JSON, CSV, TEXTFILE, RCFILE, ORCFILE, and SEQUENCEFILE, and custom extensions.
- Provides multiple client connection modes and supports JDBC APIs.

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.

8.1.2 Common Concepts of Hive Application Development

- **Client**

  Users can access the server from the client through Java APIs and Thrift APIs to perform Hive-related operations. The Hive client in this document refers to the Hive client installation directory, which contains sample codes for Hive access using Java APIs.

- **HiveQL**

  Similar to SQL

- **HCatalog**

  HCatalog is a table information management layer created based on Hive metadata and incorporates DDL commands of Hive. HCatalog provides read/write APIs for MapReduce and offers a Hive command line interface (CLI) for defining data and querying metadata. The Hive HCatalog function enables Hive and MapReduce development personnel to share metadata, preventing intermediate conversion and adjustment and improving the data processing efficiency.

- **WebHCat**

  WebHCat running users use REST APIs to run Hive DDL commands, submit MapReduce tasks, query MapReduce task execution results, and perform other operations.

8.1.3 Hive Application Development Process

Figure 8-1 and **Table 8-1** describe the phases in the development process.

Figure 8-1 Hive application development process

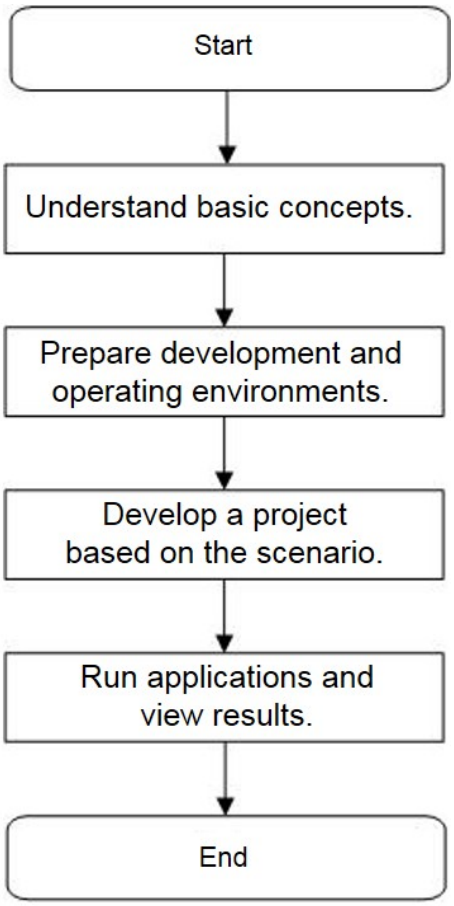


Table 8-1 Hive application development process details

Phase	Description	Reference
Understand basic concepts.	Before application development, understand basic concepts about Hive.	**Common Concepts of Hive Application Development**
Prepare development and operating environments.	You can use Java and Python to develop Hive applications. You are advised to use the Eclipse tool to configure the development environment based on the language.	**Hive Application Development Environment**
Develop a project based on the scenario.	Hive provides Java and Python sample projects, covering table creation, data load, and data queries.	**Hive Development Plan**
Run applications and view results.	This phase provides guidance for users to submit a developed application for running and view the result.	• **Commissioning a Hive JDBC Application on Linux** • **Commissioning a Hive HCatalog Application on Linux**

8.2 Preparing a Hive Application Development Environment

8.2.1 Hive Application Development Environment

Table 8-2 describes the local environment required for application development. You also need to prepare a Linux environment for verifying whether the application is running properly.

Table 8-2 Development environment

Item	Description
OS	• Development environment: Windows 7 or later version is recommended. • Operating environment: Linux system

Item	Description
JDK installation	Basic configurations of the development and operating environments. The version requirements are as follows: The server and client of an MRS cluster support only built-in Oracle JDK 1.8, which cannot be replaced. If users' applications need to reference the JAR files of the SDK class in the user application processes, Oracle JDK and IBM JDK are supported. ● Oracle JDK versions: 1.7 and 1.8 ● IBM JDK versions: 1.7.8.10, 1.7.9.40, and 1.8.3.0 Note: For security purpose in the HCatalog development environment, the MRS server supports only TLS 1.1 and TLS 1.2 encryption protocols. IBM JDK supports only TLS 1.0 by default. If you use IBM JDK, set **com.ibm.jsse2.overrideDefaultTLS** to **true**. After the parameter setting, TLS1.0/1.1/1.2 can be supported at the same time. For details, visit **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**.

Item	Description
Eclipse installation and configuration	It is a tool used to develop Hive applications. The version requirements are as follows: • JDK 1.7 and Eclipse 3.7.1 or later are supported. JDK 1.8 and Eclipse 4.3.2 or later are supported. Description If you use IBM JDK, ensure that the JDK configured in Eclipse is IBM JDK. If you use Oracle JDK, ensure that the JDK configured in Eclipse is Oracle JDK. Do not use the same workspace and the sample project in the same path for different Eclipse programs.
Network	The client must be interconnected with the Hive server on the network.

8.2.2 Preparing a Local Application Development Environment

- Install Eclipse and JDK in the Windows development environment.

  The JDK version is 1.8, and the Eclipse version is 4.3.2 or later.

  NOTE

  - If you use IBM JDK, ensure that the JDK configured in Eclipse is IBM JDK.
  - If you use Oracle JDK, ensure that the JDK configured in Eclipse is Oracle JDK.
  - If you use ODBC for secondary development, ensure that JDK 1.8 or later is used.
  - Do not use the same workspace and the sample project in the same path for different Eclipse programs.
- Prepare a Linux environment for testing application running status.

Preparing a Running and Commissioning Environment

Step 1 On the ECS management console, apply for a new ECS for user application development, running, and commissioning.

- The security group of the ECS must be the same as that of the master node in an MRS cluster.
- The ECS and the MRS cluster must be in the same VPC.
- The ECS network interface controller (NIC) and the MRS cluster must be in the same network segment.

Step 2 Apply for an EIP, bind it, and configure an inbound or outbound rule for the security group.

Step 3 Download the client program. For details, see **Downloading an MRS Client**.

Step 4 Install a cluster client as user **root**.

1. Run the following command to decompress the client package:
   tar -xvf /opt/MRS_Services_Client.tar
2. Run the following command to verify the installation file package:
   sha256sum -c /opt/MRS_Services_ClientConfig.tar.sha256
   MRS_Services_ClientConfig.tar:OK
3. Run the following command to decompress the installation file package:
   tar -xvf /opt/MRS_Services_ClientConfig.tar
4. Run the following command to install the client to a specified directory (absolute path), for example, **/opt/client**. The directory is automatically created.
   cd /opt/MRS_Services_ClientConfig
   sh install.sh /opt/client
   ```
   Components client installation is complete.
   ```

----End

8.2.3 Preparing a Hive Application Development User

The development user is used to run the sample project. The user must have Hive permissions to run the Hive sample project.

Prerequisites

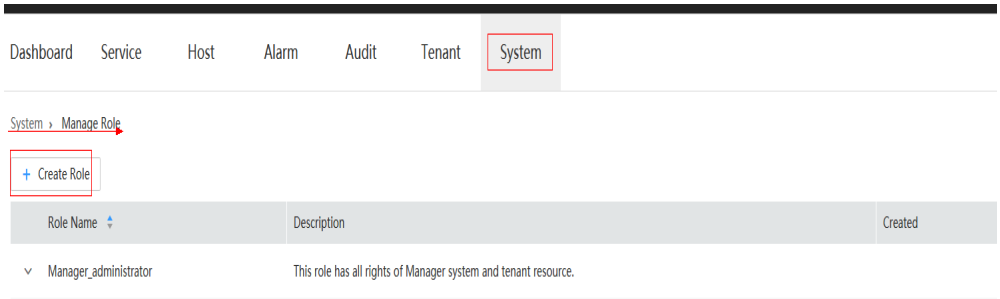
Kerberos authentication has been enabled for the MRS cluster. Skip this step if Kerberos authentication is not enabled for the cluster.

Procedure

Step 1 Log in to MRS Manager. For details, see **Logging in to MRS Manager**.

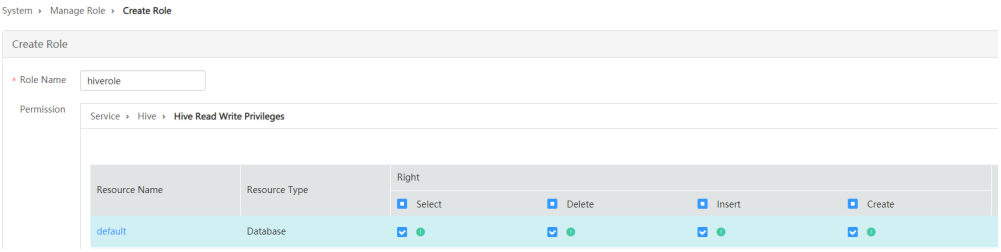
Step 2 On MRS Manager, choose **System** > **Manage Role** > **Create Role**, as shown in **Figure 8-2**.

Figure 8-2 Creating a Hive role



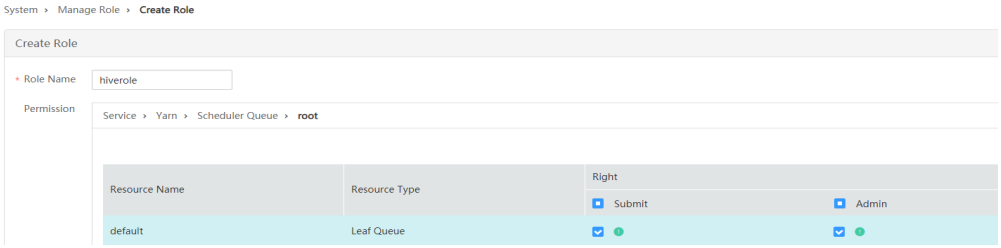
1. Enter a role name, for example, *hiverole*.
2. In the **Permissions** table, choose **Hive> Hive Read Write Privileges** and select **Select**, **Delete**, **Insert**, and **Create**, as shown in **Figure 8-3**.

Figure 8-3 Hive permission assignment



3. In the **Permissions** table, choose **Yarn > Scheduler Queue > root**, and select **Submit** and **Admin** for default, as shown in **Figure 8-4**.

Figure 8-4 Yarn permission assignment

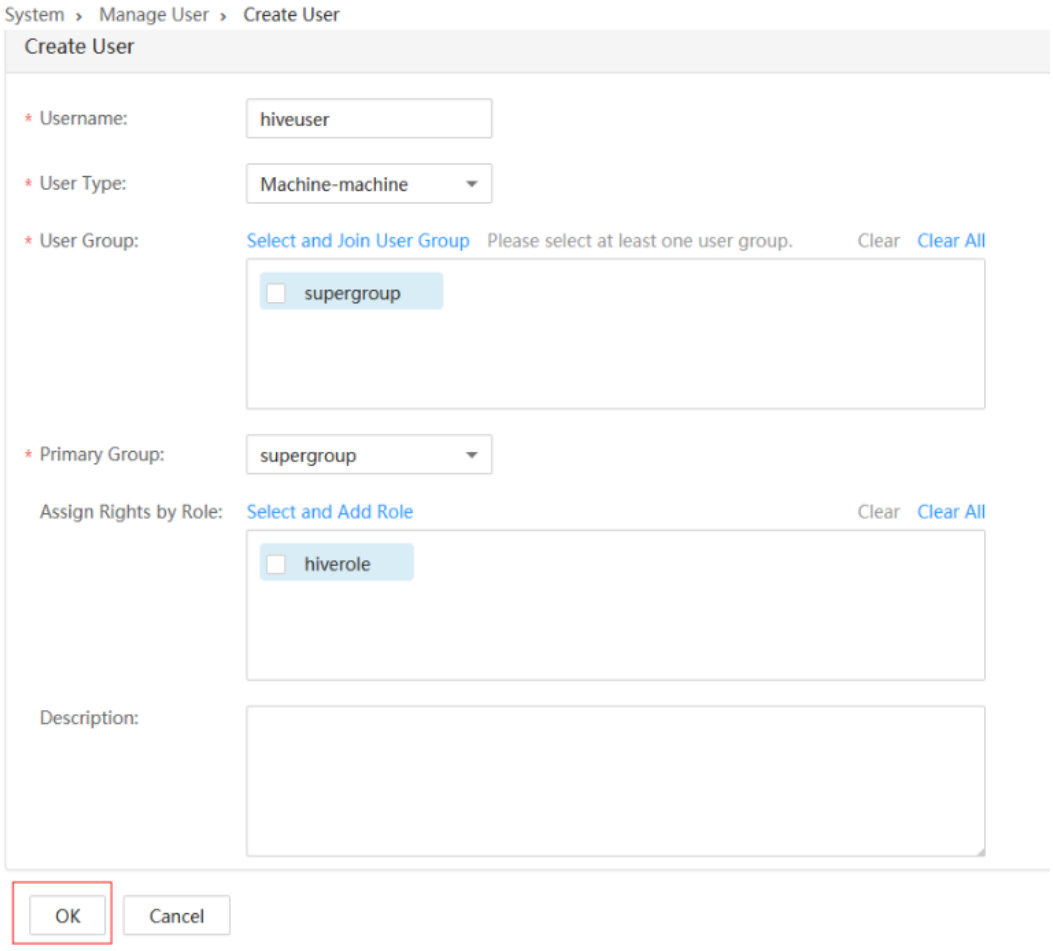


4. Click **OK**.

Step 3 Choose **System** > **Manage User** > **Create User** to create a user for the sample project.

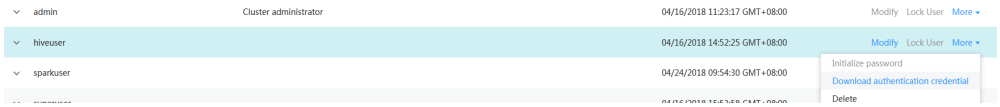
Step 4 Enter a username, for example, *hiveuser*. Set **User Type** to **Machine-machine**, and select **supergroup** in **User Group**. Set **Primary Group** to **supergroup**, select **hiverole** in **Assign Rights by Role**, and click **OK**. **Figure 8-5** shows the parameter settings.

Figure 8-5 Creating a Hive user



Step 5 On MRS Manager, choose **System** > **Manage User** and select **hiveuser**. In the **Operation** column on the right, choose **More** > **Download authentication credential**. See **Figure 8-6**. Save the file and decompress it to obtain the **user.keytab** and **krb5.conf** files. The two files are used for security authentication in the sample project.

Figure 8-6 Downloading the authentication credential



----End

Related Information

If you modify component parameter configurations, you need to download the client configuration file again and update the client in the running and commissioning environment.

8.2.4 Preparing a Hive JDBC Development Environment

To run the JDBC API sample code of Hive, you need to perform the following operations.

NOTE

The following example develops an application that uses JDBC to connect to Hive in the Windows environment.

Procedure

Step 1 Obtain the Hive sample project by referring to **Obtaining the MRS Application Development Sample Project**.

Step 2 In the root directory of the Hive sample project, run the **mvn install** command to perform compilation.

Step 3 In the root directory of the Hive sample project, run the **mvn eclipse:eclipse** command to create an Eclipse project.

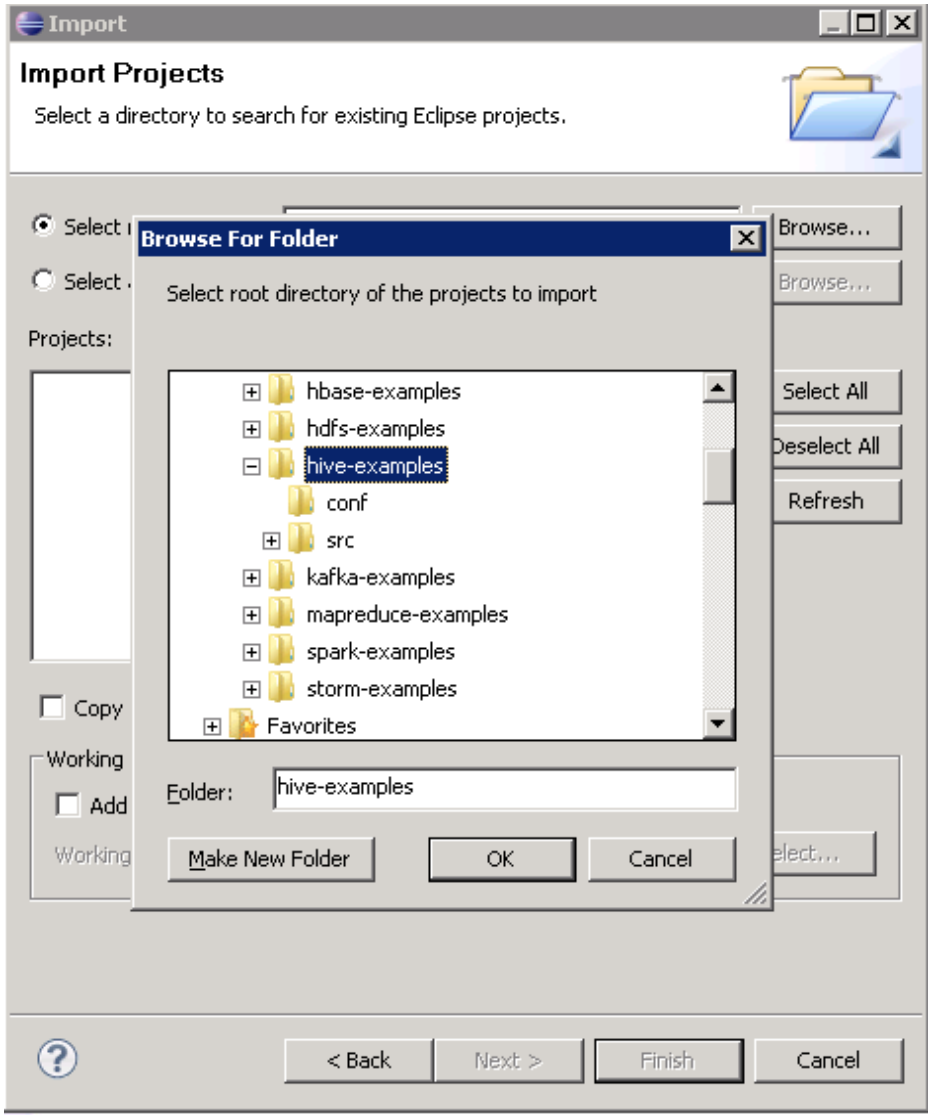
Step 4 In the application development environment, import the sample project to the Eclipse development environment.

1. Choose **File** > **Import** > **General** > **Existing Projects into Workspace** > **Next** > **Browse**.

   The **Browse Folder** dialog box is displayed.

2. Select the **hive-examples** folder, as shown in **Figure 8-7**. On Windows, the folder path cannot contain any space.

Figure 8-7 Importing a sample project to Eclipse



Click **Finish**.

After successful import, the **JDBCExample** class in **com.huawei.bigdata.hive.example** is the JDBC API sample code.

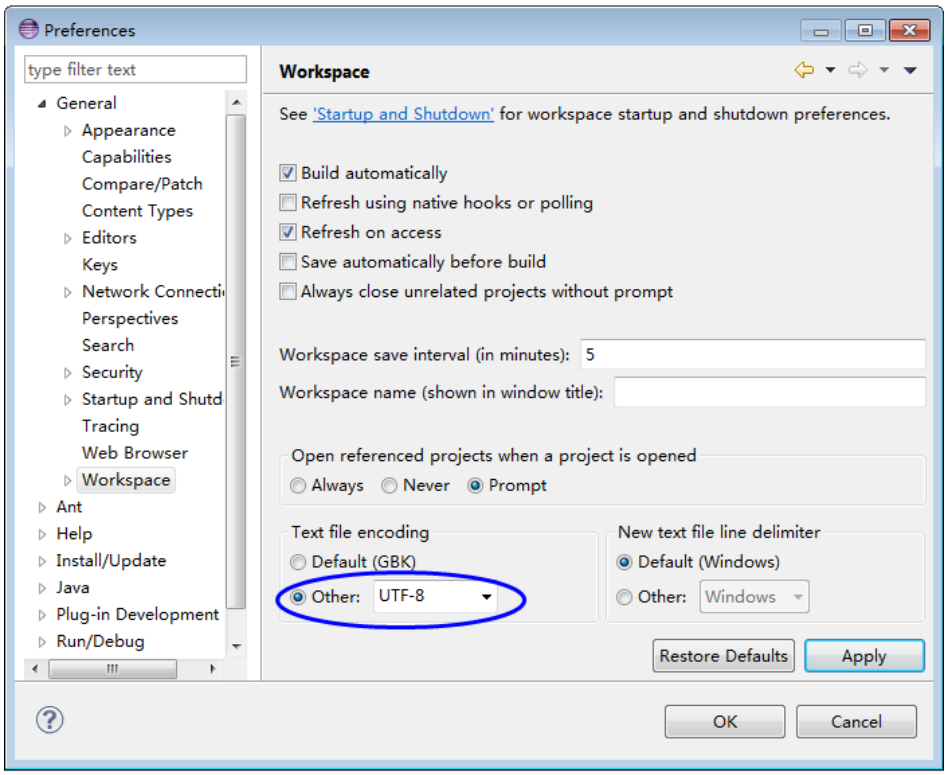
Step 5 Set an Eclipse text file encoding format to prevent garbled characters.

1. On the Eclipse menu bar, choose **Window** > **Preferences**.

   The **Preferences** window is displayed.

2. In the navigation tree, choose **General** > **Workspace**. In the **Text file encoding** area, select **Other** and set the value to **UTF-8**. Click **Apply** and then **OK**. **Figure 8-8** shows the settings.

Figure 8-8 Setting the Eclipse encoding format



Step 6 Modify the sample. You can skip this step for a cluster with Kerberos authentication disabled.

After you obtain the **krb5.conf** and **user.keytab** files of the new development user in **Step 5**, change the value of **userName** in **ExampleMain.java** to the new username, for example, **hiveuser**.

```
/**
 * Other way to set conf for zk. If use this way,
 * can ignore the way in the 'login' method
 */
if (isSecurityMode) {
    userName = "hiveuser";
    userKeytabFile = CONF_DIR + "user.keytab";
    krb5File = CONF_DIR + "krb5.conf";
    conf.set(HADOOP_SECURITY_AUTHENTICATION, "kerberos");
    conf.set(HADOOP_SECURITY_AUTHORIZATION, "true");
```

----End

8.2.5 Preparing a Hive HCatalog Development Environment

To run the HCatalog API sample code of Hive, you need to perform the following operations.

NOTE

The following example develops an application that uses HCatalog to connect to Hive in the Windows environment.

Procedure

Step 1 Obtain the Hive sample project by referring to **Obtaining the MRS Application Development Sample Project**.

Step 2 In the root directory of the Hive sample project, run the **mvn install** command to perform compilation.

Step 3 In the root directory of the Hive sample project, run the **mvn eclipse:eclipse** command to create an Eclipse project.

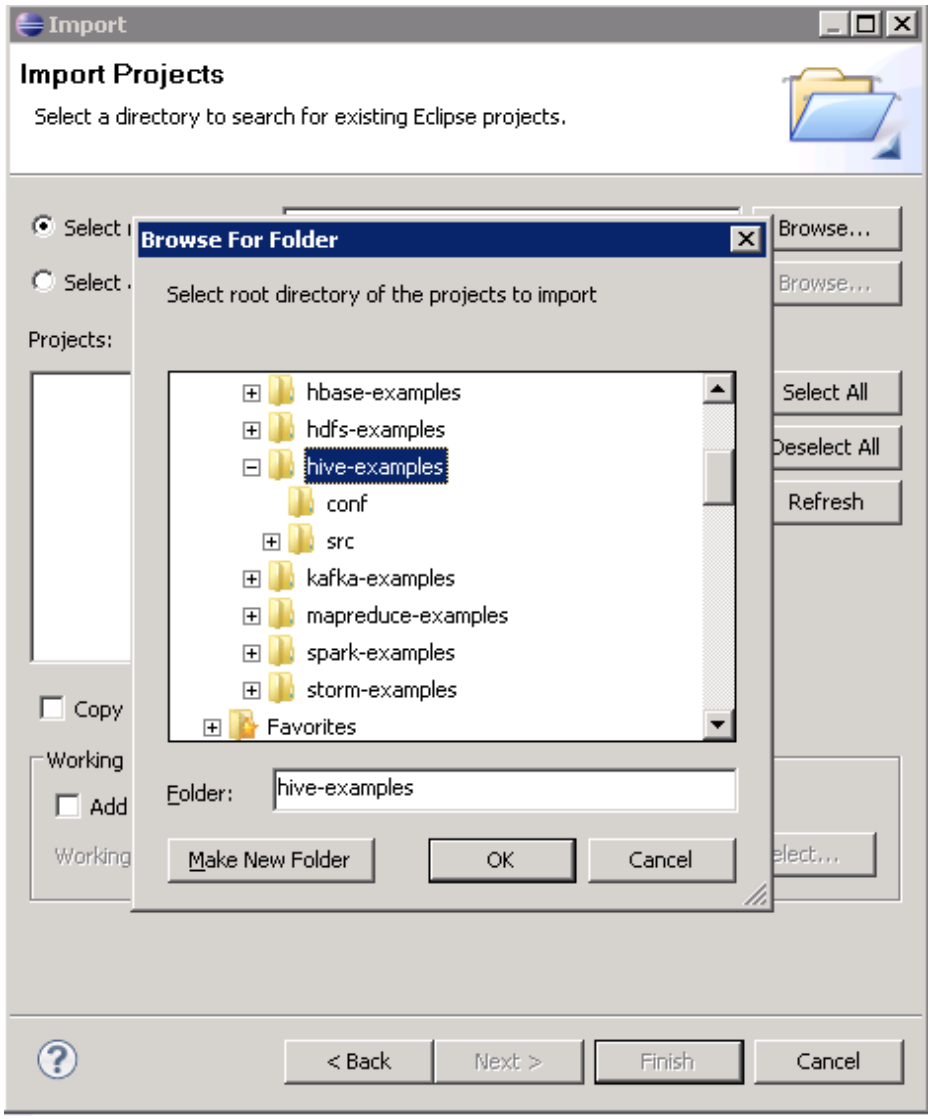
Step 4 In the application development environment, import the sample project to the Eclipse development environment.

1. Choose **File** > **Import** > **General** > **Existing Projects into Workspace** > **Next** > **Browse**.

   The **Browse Folder** dialog box is displayed.

2. After downloading the project, select the **hive-examples** folder, as shown in **Figure 8-9**. On Windows, the folder path cannot contain any space.

Figure 8-9 Importing a sample project to Eclipse



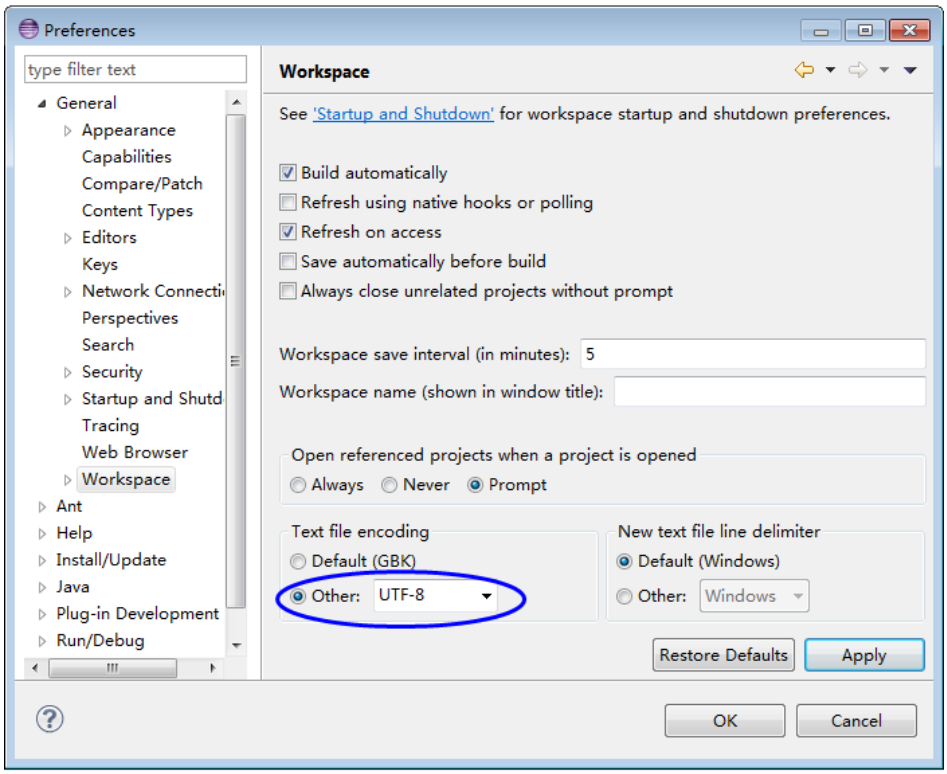
Click **Finish**.

After successful import, the **HCatalogExample** class in **com.huawei.bigdata.hive.example** is the HCatalog API sample code.

Step 5 Set an Eclipse text file encoding format to prevent garbled characters.

1. On the Eclipse menu bar, choose **Window** > **Preferences**. The **Preferences** window is displayed.
2. In the navigation tree, choose **General** > **Workspace**. In the **Text file encoding** area, select **Other** and set the value to **UTF-8**. Click **Apply** and then **OK**. **Figure 8-10** shows the settings.

Figure 8-10 Setting the Eclipse encoding format



----End

8.3 Developing a Hive Application

8.3.1 Hive Development Plan

Scenario Description

Assume that you need to develop a Hive data analysis application to manage the employee information of an enterprise. **Table 8-3** and **Table 8-4** provide employee information.

Development Guidelines

Step 1 Prepare data.

1. Create three tables: employee information table **employees_info**, employee contact table **employees_contact**, and extended employee information table **employees_info_extended**.
   - Employee information table **employees_info** contains fields such as employee ID, name, salary currency, salary, tax category, work place, and hire date. In salary currency, R indicates RMB and D indicates USD.
   - Fields in the **employees_contact** table include the employee ID, phone number, and email address.
   - Fields in the **employees_info_extended** table include the employee ID, name, mobile phone number, e-mail address, salary currency, salary, tax category, and work place. The partition field is the hire date.

     For details about table creation codes, see **Creating a Hive Table**.
2. Load employee information to **employees_info**.

   For details about data loading codes, see **Loading Hive Data**.

   Table 8-3 provides employee information.

   Table 8-3 Employee information

Employee ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hire Date
1	Wang	R	8000.01	personal income tax&0.05	China:Shenzhen	2014
3	Tom	D	12000.02	personal income tax&0.09	America:NewYork	2014
4	Jack	D	24000.03	personal income tax&0.09	America:Manhattan	2014
6	Linda	D	36000.04	personal income tax&0.09	America:NewYork	2014
8	Zhang	R	9000.05	personal income tax&0.05	China:Shanghai	2014

3. Load employee contact information to **employees_contact**.

   Table 8-4 provides employee contact information.

Table 8-4 Employee contact information

Employee ID	Phone Number	E-mail
1	135 XXXX XXXX	xxxx@xx.com
3	159 XXXX XXXX	xxxxx@xx.com.cn
4	186 XXXX XXXX	xxxx@xx.org
6	189 XXXX XXXX	xxxx@xxx.cn
8	134 XXXX XXXX	xxxx@xxxx.cn

Step 2 Analyze data.

For details about data analysis codes, see **Querying Hive Data**.

- Query contact information of employees whose salaries are paid in USD.
- Query the IDs and names of employees who were hired in 2014, and load the query results to the partition with the hire date of 2014 in the **employees_info_extended** table.
- Collect the number of records in the **employees_info** table.
- Query information about employees whose email addresses end with "cn".

Step 3 Submit a data analysis task to collect the number of records in the **employees_info** table. For details, see **Analyzing Hive Data**.

----End

8.3.2 Creating a Hive Table

Function Description

This section describes how to use HiveQL to create internal and external tables. You can create a table by using any of the following methods:

- Customize the table structure, and use the key word **EXTERNAL** to differentiate between internal and external tables.
  - If all data is to be processed by Hive, create an internal table. When an internal table is deleted, the metadata and data in the table are deleted together.
  - If data is to be processed by multiple tools, create an external table. When an external table is deleted, only metadata is deleted.
- Create a table based on existing tables. Use CREATE LIKE to fully copy the original table structure, including the storage format.
- Create a table based on query results using CREATE AS SELECT.

  This method is flexible. By using this method, you can specify fields (except for the storage format) that you want to copy to the new table when copying the structure of the existing table.

Sample Code

```
-- Create the external table employees_info.
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info
(
id INT,
name STRING,
usd_flag STRING,
salary DOUBLE,
deductions MAP<STRING, DOUBLE>,
address STRING,
entrytime STRING
)
-- Specify the field delimiter.
-- "delimited fields terminated by" indicates that the column delimiter is ',' and "MAP KEYS TERMINATED BY" indicates that the delimiter of key values in the MAP is '&'.
ROW FORMAT delimited fields terminated by ',' MAP KEYS TERMINATED BY '&'
-- Specify the table storage format to TEXTFILE.
STORED AS TEXTFILE;

-- Use CREATE Like to create a table.
CREATE TABLE employees_like LIKE employees_info;

-- Run the DESCRIBE command to check the structures of the employees_info, employees_like, and employees_as_select tables.
DESCRIBE employees_info;
DESCRIBE employees_like;
```

Extended Applications

- Create a partition table.

  A table may have one or more partitions. Each partition is saved as an independent folder in the table directory. Partitions help minimize the query scope, accelerate data query, and allow users to manage data based on certain criteria.

  A partition is defined using the PARTITIONED BY clause during table creation.

  ```
  CREATE EXTERNAL TABLE IF NOT EXISTS employees_info_extended
  (
  id INT,
  name STRING,
  usd_flag STRING,
  salary DOUBLE,
  deductions MAP<STRING, DOUBLE>,
  address STRING
  )
  -- Use "PARTITIONED BY" to specify the column name and data type of the partition.
  PARTITIONED BY (entrytime STRING)
  STORED AS TEXTFILE;
  ```

- Update the table structure.

  After a table is created, you can use ALTER TABLE to add or delete fields, modify table attributes, and add partitions.

  ```
  -- Add the tel_phone and email fields to the employees_info_extended table.
  ALTER TABLE employees_info_extended ADD COLUMNS (tel_phone STRING, email STRING);
  ```

- 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;
  ```


- Enable Hive to use OBS.

  You need to set the specified parameters in beeline. You can log in to OBS console and access the **My Credential** page to obtain the AK/SK.

  ```
  set fs.obs.access.key=AK;
  set fs.obs.secret.key=SK;
  set metaconf:fs.obs.access.key=AK;
  set metaconf:fs.obs.secret.key=SK;
  ```

  Set the storage type of the new table to obs.

  ```
  create table obs(c1 string, c2 string) stored as orc location 'obs://obs-lmm/hive/orctest'
  tblproperties('orc.compress'='SNAPPY');
  ```

  NOTE

  When Hive uses OBS to store data, partition and table storage locations in the same table cannot be stored in different buckets.

  For example, create a partition table and set its storage location to the folder in OBS bucket 1. In this case, modifying the storage location of the table partition does not take effect. When data is inserted, the storage location of the table is used.

  1. Create a partition table and specify the path for storing the table.
     ```
     create table table_name(id int,name string,company string) partitioned by(dt date) row format delimited fields terminated by ',' stored as textfile location "obs://OBS bucket 1/Folder in the bucket";
     ```
  2. Modifying the storage location of the table partition to another bucket does not take effect.
     ```
     alter table table_name partition(dt date) set location "obs://OBS bucket 2/Folder in the bucket";
     ```

8.3.3 Loading Hive Data

Function Description

This section describes how to use HiveQL to load data to the existing **employees_info** table. You can learn how to load data from a cluster.

Sample Code

```
--Load the employee_info.txt file from the /opt/hive_examples_data/ directory of the local file system to the employees_info table.
LOAD DATA LOCAL INPATH '/opt/hive_examples_data/employee_info.txt' OVERWRITE INTO TABLE employees_info;

-- Load /user/hive_examples_data/employee_info.txt from HDFS to the employees_info table.
LOAD DATA INPATH '/user/hive_examples_data/employee_info.txt' OVERWRITE INTO TABLE employees_info;
```

NOTE

The essence of loading data is to copy the data to the specified table directory in HDFS.

The **LOAD DATA LOCAL INPATH** command can be used to load files from a local file system to Hive. If **LOCAL** is specified, the path refers to the path of the local file system of the currently connected HiveServer. However, HiveServers are deployed in a cluster, and the client is randomly connected to one of all HiveServers. Therefore, you need to check whether files to be loaded exist in the local file system of the connected HiveServer. If you cannot determine which HiveServer is connected, you are advised to save the corresponding files in all HiveServer paths, and check whether the file permissions are correct.

8.3.4 Querying Hive Data

Function Description

This section describes how to use HiveQL to query and analyze data. You can query and analyze data using the following methods:

- Use common features of a SELECT query, such as JOIN.
- Load data to a specified partition.
- Use built-in functions of Hive.
- Query and analyze data using user-defined functions. For details about how to create and define functions, see **Developing User-Defined Hive Functions**.

Sample Code

```
-- Query contact information of employees whose salaries are paid in USD.
SELECT
a.name,
b.tel_phone,
b.email
FROM employees_info a JOIN employees_contact b  ON(a.id = b.id) WHERE usd_flag='D';

-- Query the IDs and names of employees who were hired in 2014, and load the query results to the partition with the hire date of 2014 in the employees_info_extended table.
INSERT OVERWRITE TABLE employees_info_extended PARTITION (entrytime = '2014')
SELECT
a.id,
a.name,
a.usd_flag,
a.salary,
a.deductions,
a.address,
b.tel_phone,
b.email
FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE a.entrytime = '2014';

-- Use the existing function COUNT() in Hive to calculate the number of records in the employees_info table.
SELECT COUNT(*) FROM employees_info;

-- Query information about employees whose email addresses end with "cn".
SELECT a.name, b.tel_phone FROM  employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE b.email like '%cn';
```

Extended Application

- 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 user-defined functions, see **Developing User-Defined Hive Functions**.

8.3.5 Analyzing Hive Data

Function Description

This section describes how to use a sample program to complete an analysis task. The sample program provides the following methods:

- Submitting a data analysis task by using JDBC APIs
- Submitting a data analysis task by using HCatalog APIs

Sample Code

- If you submit a data analysis task using Hive JDBC APIs, refer to **JDBCExample.java** in the sample program.

  a. Define HiveQL. HiveQL must be a single statement and cannot contain ";".

  ```
  // Define HiveQL, which cannot contain the semicolon (;).
  String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT,name STRING)",
          "SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};
  ```

  b. Build JDBC URL.

  ```
  // Build JDBC URL.
  StringBuilder sBuilder = new StringBuilder(
    "jdbc:hive2://").append(clientInfo.getZkQuorum()).append("/");

  if (isSecurityMode) {
      // Security mode
      // ZooKeeper login authentication
      sBuilder.append(";serviceDiscoveryMode=")
              .append(clientInfo.getServiceDiscoveryMode())
              .append(";zooKeeperNamespace=")
              .append(clientInfo.getZooKeeperNamespace())
              .append(";sasl.qop=")
              .append(clientInfo.getSaslQop())
              .append(";auth=")
              .append(clientInfo.getAuth())
              .append(";principal=")
              .append(clientInfo.getPrincipal())
              .append(";");
  } else {
      // Normal mode
      sBuilder.append(";serviceDiscoveryMode=")
              .append(clientInfo.getServiceDiscoveryMode())
              .append(";zooKeeperNamespace=")
              .append(clientInfo.getZooKeeperNamespace())
              .append(";auth=none");
  }
  String url = sBuilder.toString();
  ```

  The preceding operations are performed to access Hive through ZooKeeper. If you want to access Hive by directly connecting to HiveServer, perform the following operations to combine the JDBC URL and change the port of **zk.quorum** in the **hiveclient.properties** file to **10000**.

  ```
  // Build JDBC URL.
  StringBuilder sBuilder = new StringBuilder(
    "jdbc:hive2://").append(clientInfo.getZkQuorum()).append("/");

  if (isSecurityMode) {
      // Security mode
      // ZooKeeper login authentication
      sBuilder.append(";sasl.qop=")
              .append(clientInfo.getSaslQop())
  ```



```
            .append(";auth=")
            .append(clientInfo.getAuth())
            .append(";principal=")
            .append(clientInfo.getPrincipal())
            .append(";");
} else {
    // Normal mode
    sBuilder.append(";auth=none");
}
String url = sBuilder.toString();
```

Note: When the HiveServer is directly connected, if the connected HiveServer is faulty, Hive access fails. If the ZooKeeper is used to access Hive, any available HiveServer instance can provide services properly. Therefore, you are advised to use ZooKeeper to access Hive when using JDBC.

c. Load the Hive JDBC driver.

```
// Load the Hive JDBC driver.
Class.forName(HIVE_DRIVER);
```

d. Enter a correct username, obtain the JDBC connection, confirm the HiveQL type (DDL/DML), call APIs to run HiveQL, return the queried column name and result to the console, and close the JDBC connection.

```
Connection connection = null;
  try {
    // Obtain the JDBC connection.
    // If you set the second parameter to an incorrect username, the anonymous user will be used for login.
    connection = DriverManager.getConnection(url, "userName", "");

    // Create a table.
    // To import data to a table after the table is created, you can use the LOAD statement. For example, import data from HDFS to the table.
    //load data inpath '/tmp/employees.txt' overwrite into table employees_info;
    execDDL(connection,sqls[0]);
    System.out.println("Create table success!");

    // Query
    execDML(connection,sqls[1]);

    // Delete the table.
    execDDL(connection,sqls[2]);
    System.out.println("Delete table success!");
  }
  finally {
    // Close the JDBC connection.
    if (null != connection) {
      connection.close();
    }

public static void execDDL(Connection connection, String sql)
  throws SQLException {
    PreparedStatement statement = null;
    try {
      statement = connection.prepareStatement(sql);
      statement.execute();
    }
    finally {
      if (null != statement) {
        statement.close();
      }
    }
  }


  public static void execDML(Connection connection, String sql) throws SQLException {
    PreparedStatement statement = null;
```



```
    ResultSet resultSet = null;
    ResultSetMetaData resultMetaData = null;

    try {
      // Execute HiveQL.
      statement = connection.prepareStatement(sql);
      resultSet = statement.executeQuery();

      // Output the queried column name to the console.
      resultMetaData = resultSet.getMetaData();
      int columnCount = resultMetaData.getColumnCount();
      for (int i = 1; i <= columnCount; i++) {
        System.out.print(resultMetaData.getColumnLabel(i) + '\t');
      }
      System.out.println();

      // Output the query result to the console.
      while (resultSet.next()) {
        for (int i = 1; i <= columnCount; i++) {
          System.out.print(resultSet.getString(i) + '\t');
        }
        System.out.println();
      }
    }
    finally {
      if (null != resultSet) {
        resultSet.close();
      }

      if (null != statement) {
        statement.close();
      }
    }
  }
```

- If you submit a data analysis task using HCatalog APIs, refer to **HCatalogExample.java** in the sample program.

  a. Compile the Map class to obtain data from a Hive table.

  ```
  public static class Map extends
          Mapper<LongWritable, HCatRecord, IntWritable, IntWritable> {
      int age;
      @Override
      protected void map(
              LongWritable key,
              HCatRecord value,
              Context context)
              throws IOException, InterruptedException {
          age = (Integer) value.get(0);
          context.write(new IntWritable(age), new IntWritable(1));
      }
  }
  ```

  b. Compile the Reduce class to collect statistics on data read from the Hive table.

  ```
  public static class Reduce extends Reducer<IntWritable, IntWritable,
  IntWritable, HCatRecord> {
    @Override
    protected void reduce(
            IntWritable key,
            Iterable<IntWritable> values,
            Context context)
            throws IOException, InterruptedException {
        int sum = 0;
        Iterator<IntWritable> iter = values.iterator();
        while (iter.hasNext()) {
            sum++;
            iter.next();
        }
        HCatRecord record = new DefaultHCatRecord(2);
  ```



```
            record.set(0, key.get());
            record.set(1, sum);

            context.write(null, record);
        }
    }
```

c. After configuring the job in the **run()** method, execute the **main()** method to submit a task.

```
    public int run(String[] args) throws Exception {

        HiveConf.setLoadMetastoreConfig(true);
        Configuration conf = getConf();
        String[] otherArgs = args;

        String inputTableName = otherArgs[0];
        String outputTableName = otherArgs[1];
        String dbName = "default";

        @SuppressWarnings("deprecation")
        Job job = new Job(conf, "GroupByDemo");

        HCatInputFormat.setInput(job, dbName, inputTableName);
        job.setInputFormatClass(HCatInputFormat.class);
        job.setJarByClass(HCatalogExample.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(WritableComparable.class);
        job.setOutputValueClass(DefaultHCatRecord.class);

        OutputJobInfo outputjobInfo = OutputJobInfo.create(dbName,outputTableName, null);
        HCatOutputFormat.setOutput(job, outputjobInfo);
        HCatSchema schema = outputjobInfo.getOutputSchema();
        HCatOutputFormat.setSchema(job, schema);
        job.setOutputFormatClass(HCatOutputFormat.class);

        return (job.waitForCompletion(true) ? 0 : 1);
    }
    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new HCatalogExample(), args);
        System.exit(exitCode);
    }
```

8.3.6 Developing User-Defined Hive Functions

When built-in 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 use methods, UDFs are classified as follows:

- Temporary functions: used only in the current session and must be recreated after a session restarts.
- Permanent functions: used in multiple sessions. You do not need to create them every time a session restarts.

The following uses AddDoublesUDF as an example to describe how to compile and use UDFs.

Function Description

The AddDoublesUDF is used to add two or more floating point numbers. The following example describes how to compile and use UDFs.

NOTE

- A common UDF must be inherited from **org.apache.hadoop.hive.ql.exec.UDF**.
- A common UDF must implement at least one **evaluate()**. The evaluate function supports overloading.

Sample Code

The following is a UDF code 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 Log in to MRS Manager and configure the Hive administrator permission for the Hive service user who uses UDFs.

1. Log in to MRS Manager, choose **System** > **Manage Role** > **Create Role**, and create a role with the **Hive Admin Privilege** permission.
2. On MRS Manager, choose **System** > **Manage User**.
3. In the **Operation** column of the user, click **Modify**.
4. Bind a role with the **Hive Admin Privilege** permission to the user and click **OK**.

Step 2 Create a UDF package in the example directory of the project, compile the **AddDoublesUDF** class, package the project (for example, **AddDoublesUDF.jar**), and upload the package to a specified HDFS directory (for example, **/user/hive_examples_jars/**). Grant the read permission on the file to the user who creates the function and who uses the function. The following are example statements.

hdfs dfs -put AddDoublesUDF.jar /user/hive_examples_jars

hdfs dfs -chmod 777 /user/hive_examples_jars

Step 3 If the Kerberos authentication is enabled for the current cluster, run the following command to authenticate the user. If the Kerberos authentication is disabled for

the current cluster, skip this step. The current user is the development user added in **Preparing a Hive Application Development User**.

kinit *Hive service user*

For example, **kinit -kt '/opt/conf/user.keytab' hiveuser** (set the **user.keytab** path based on the site requirements).

Step 4 Run the **set role admin;** command to grant the administrator permission to the user.

Step 5 Run the following command:

beeline -n *Hive service user*

Step 6 Define the function in HiveServer. 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 HiveServer.

Step 7 Run the following SQL statement to use the function on the HiveServer:

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 8 Run the following SQL statement to delete the function from the HiveServer:

DROP FUNCTION addDoubles;

----End

8.4 Commissioning a Hive Application

8.4.1 Commissioning a Hive JDBC Application on Windows

Running the JDBC Client Using CLI

Step 1 Run the sample project.

After you import and modify the sample project according to **Preparing a Hive JDBC Development Environment**, download the **hiveclient.properties** file from the **/opt/client/Hive/config/hiveclient.properties** directory of any master node in the cluster, and save the file to the **conf** directory of the sample project, that is, **hive-examples/conf**. Then, you can right-click **JDBCExample.java** in the development environment, for example, **Eclipse**, and choose **Run as** > **Java Application** to run the corresponding application project.

NOTE

You can use either of the following method to access an MRS cluster to operate Hive on Windows.

- Apply for a Windows ECS to access the MRS cluster to operate Hive. This method provides high availability because it obtains dynamic HiveServer address by connecting to ZooKeeper to operate Hive.
- Use the local host to access the MRS cluster to operate Hive. Because the network between the local host and MRS cluster is disconnected, Hive can be operated only by directly connecting to HiveServer.

Method 1: Apply for a Windows ECS to access the MRS cluster to operate Hive. Run the sample code after the development environment is installed. To apply for ECS to access the MRS cluster, perform the following steps:

1. On the **Active Clusters** page, click the name of an existing cluster.

   On the cluster details page, record the **AZ**, **VPC**, **Cluster Manager IP Address** of the cluster, and **Default Security Group** of the Master node.

2. On the ECS management console, create a new ECS.

   The **AZ**, **VPC**, and **security group** of ECS must be the same as those of the cluster to be accessed.

   Select a Windows public image.

   For details about other configuration parameters, see **Elastic Cloud Server** > **Quick Start** > **Purchasing and Logging In to a Windows ECS**.

Method 2: Use the local host to access the MRS cluster to operate Hive. After installing the development environment and completing the following steps, run the sample code.

1. Bind an EIP to the HiveServer or ZooKeeper node that will use the Hive service in the MRS cluster. To bind an EIP, perform the following steps:
   1. On the VPC management console, apply for an EIP and bind it to ECS.

      For details, see **Virtual Private Cloud** > **User Guide** > **Elastic IP Address** > **Assigning an EIP and Binding It to an ECS**.

   2. Open security group rules for the MRS cluster.

      Add security group rules to a security group of the Master and Core nodes in the cluster to enable the ECS to access the cluster. If the cluster is a security cluster, you need to enable UDP ports 21731 and 21732 and TCP ports 21730, 21731, and 21732. Add the HiveServer instance port and ZooKeeper service port of Hive to the inbound rule of the security group. For details, see **Virtual Private Cloud** > **User Guide** > **Security** > **Security Group** > **Adding a Security Group Rule**.
2. Modify the imported **hiveclient.properties** file to make the **zk.quorum** parameter correspond to the EIP bound to HiveServer and the ZooKeeper port. Modify the URL combination for connecting to the JDBC in the sample code. For details about how to directly connect to HiveServer, see **Analyzing Hive Data**.
3. Change the IP addresses of the **kdc**, **admin_server**, **kpasswd_server**, **kdc_listen**, **kadmind_listen**, and **kpasswd_listen** parameters in the **krb5.conf** file of the import example (if the cluster has only one master node, you do not need to modify the last three parameters). Map it to the EIP of the **KrbServer** service (skip this step if the Kerberos function is not enabled for the common cluster). Save the modified **krb5.conf** and **user.keytab** file to the **conf** directory of the sample project.
4. If you access Hive using ZooKeeper, add the mapping between the EIPs bound to the nodes in **Step 1.1** and host names in the local **hosts** file.
5. If error message "Message stream modified (41)" is displayed, the JDK version may be incorrect. Change the JDK version in the sample code to a version earlier than 8u_242 or delete the **renew_lifetime = 0m** configuration item from the **krb5.conf** configuration file.

Step 2 View the execution result.

View the HiveQL query results in the sample code. If the following information is displayed, the execution is successful.

Result:

```
Create table success!
_c0
0
Delete table success!
```

----End

8.4.2 Commissioning a Hive JDBC Application on Linux

Step 1 Run the **mvn package** command to generate a JAR file, for example, **hive-examples-1.0.jar**, and obtain it from the **target** directory in the project directory.

Step 2 Create a directory as the running directory in the running and commissioning environment, for example, **/opt/hive_examples** (Linux), and create the **conf** subdirectory in the directory.

Copy the **hive-examples-1.0.jar** file exported in **Step 1** to **/opt/hive_examples**.

Copy the configuration file from the client to the **conf** directory. For a security cluster with Kerberos authentication enabled, copy the **user.keytab** and **krb5.conf** files obtained in **Step 5** to the **/opt/hive_examples/conf** directory. For a cluster with Kerberos authentication disabled, you do not need to copy the **user.keytab** and **krb5.conf** files. Copy the **${HIVE_HOME}/../config/hiveclient.properties** file to the **/opt/hive_examples/conf** directory.

```
cd /opt/hive_examples/conf
cp /opt/client/Hive/config/hiveclient.properties .
```

Step 3 Prepare the JAR packages related to the sample program.

Create a directory (for example, **/opt/hive_examples/lib**) in the commissioning environment to store dependency JAR packages. Copy all packages in **${HIVE_HOME}/lib/** to the directory, delete the **derby-10.10.2.0.jar** package. (The JAR package version number varies according to the site requirements.)

```
mkdir /opt/hive_examples/lib
cp ${HIVE_HOME}/lib/* /opt/hive_examples/lib
rm -f /opt/hive_examples/lib/derby-10.10.2.0.jar
```

Step 4 In Linux, run the following command to run the sample program:

```
chmod +x /opt/hive_examples -R
cd /opt/hive_examples
source /opt/client/bigdata_env
java -cp .:hive-examples-1.0.jar:/opt/hive_examples/conf:/opt/hive_examples/lib/*:/opt/client/HDFS/hadoop/lib/* com.huawei.bigdata.hive.example.ExampleMain
```

Step 5 In the CLI, view the HiveQL query results in the example code.

If the following information is displayed, the sample project execution is successful on Linux.

```
Create table success!
_c0
0
Delete table success!
```

----End

8.4.3 Commissioning a Hive HCatalog Application on Linux

Step 1 Run the **mvn package** command to generate a JAR file, for example, **hive-examples-1.0.jar**, and obtain it from the **target** directory in the project directory.

Step 2 Upload the **hive-examples-1.0.jar** package generated in the previous step to a specified path on Linux, for example, **/opt/hive_examples**, marked as $HCAT_CLIENT, and ensure that the client has been installed.

```
export HCAT_CLIENT=/opt/hive_examples/
```

Step 3 Run the following command to configure environment variables (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-1.3.0.jar,$HCAT_HOME/lib/hive-metastore-1.3.0.jar,$HIVE_HOME/lib/hive-exec-1.3.0.jar,$HCAT_HOME/lib/libfb303-0.9.3.jar,$HCAT_HOME/lib/slf4j-api-1.7.5.jar,$HCAT_HOME/lib/antlr-2.7.7.jar,$HCAT_HOME/lib/jdo-api-3.0.1.jar,$HCAT_HOME/lib/antlr-runtime-3.4.jar,$HCAT_HOME/lib/datanucleus-api-jdo-3.2.6.jar,$HCAT_HOME/lib/datanucleus-core-3.2.10.jar,$HCAT_HOME/lib/datanucleus-rdbms-3.2.9.jar
export HADOOP_CLASSPATH=$HCAT_HOME/lib/hive-hcatalog-core-1.3.0.jar:$HCAT_HOME/lib/hive-metastore-1.3.0.jar:$HIVE_HOME/lib/hive-exec-1.3.0.jar:$HCAT_HOME/lib/libfb303-0.9.3.jar:$HADOOP_HOME/etc/hadoop:$HCAT_HOME/conf:$HCAT_HOME/lib/slf4j-api-1.7.5.jar:$HCAT_HOME/lib/antlr-2.7.7.jar:$HCAT_HOME/lib/jdo-api-3.0.1.jar:$HCAT_HOME/lib/antlr-runtime-3.4.jar:$HCAT_HOME/lib/datanucleus-api-jdo-3.2.6.jar:$HCAT_HOME/lib/datanucleus-core-3.2.10.jar:$HCAT_HOME/lib/datanucleus-rdbms-3.2.9.jar
```

NOTE

Before importing the preceding environment variables, check whether the current JAR file exists. You can obtain the version number from the **lib** directory of Hive on the client.

Step 4 Prepare for the client running.

1. If the Kerberos authentication is enabled for the current cluster, run the following command to authenticate the user. If the Kerberos authentication is disabled for the current cluster, skip this step. The current user is the development user added in **Preparing a Hive Application Development User**.

   Human-machine user: **kinit *MRS* cluster user**

   For example, **kinit hiveuser**.

   Machine-machine user: kinit -kt <user.keytab path> <MRS cluster user>

   For example, **kinit -kt /opt/hive_examples/conf/user.keytab hiveuser**

   NOTE

   When connecting to a security cluster, add the following parameters to the HCatalog configuration file (for example, **/opt/client/Hive/HCatalog/conf/hive-site.xml**) on the Hive client:

   ```
   <property>
   <name>hive.metastore.sasl.enabled</name>
   <value>true</value>
   </property>
   ```

2. Use the Hive client to create source table **t1** in beeline: **create table t1(col1 int);**

   Run the **insert into t1(col1) values(X);** command to insert the following data into **t1**. In the command, **X** indicates the data value to be inserted.


```
+----------+--+
| t1.col1  |
+----------+--+
| 1        |
| 1        |
| 1        |
| 2        |
| 2        |
| 3        |
```

3. Create destination table **t2**: **create table t2(col1 int,col2 int);**

Step 5 Use the YARN client to submit a task.

yarn --config $HADOOP_HOME/etc/hadoop jar $HCAT_CLIENT/hive-examples-1.0.jar com.huawei.bigdata.hive.example.HCatalogExample -libjars $LIB_JARS t1 t2

Step 6 View the running result. The data in **t2** is as follows:

```
0: jdbc:hive2://192.168.1.18:24002,192.168.1.> select * from t2;
+----------+----------+--+
| t2.col1  | t2.col2  |
+----------+----------+--+
| 1        | 3        |
| 2        | 2        |
| 3        | 1        |
+----------+----------+--+
```

----End

8.5 FAQs About Hive Application Development

8.5.1 Hive JDBC APIs

Hive JDBC APIs comply with the Java JDBC driver standard. For details, see **JDK1.7 API**.

NOTE

As a database of the data warehouse type, Hive does not support all JDBC APIs. For example, if transactional operations are performed, such as rollback and setAutoCommit, SQL exceptions like **Method not supported** will occur.

8.5.2 HiveQL APIs

HiveQL supports all features of MRS Hive and the corresponding open-source Hive version. For details, see **https://cwiki.apache.org/confluence/display/hive/languagemanual**. **Table 8-5** lists the mapping between MRS Hive versions and open-source Hive versions.

Table 8-5 Mapping between MRS Hive and open-source versions

MRS Version	Open-source Hive version
MRS 1.9.x	2.3.3

8.5.3 Hive WebHCat APIs

NOTE

- The following uses the service IP address of WebHCat node and the WebHCat HTTP port configured during the installation as an example.
- The **user.name** parameter must be added for APIs except **:version**, **status**, **version**, **version/hive**, and **version/hadoop**.

:version(GET)

- Description

  Query a list of response types supported by WebHCat.

- URL

  http://www.myserver.com/templeton/:version

- Parameter

Parameter	Description
:version	WebHCat version number. Currently, the version number must be v1.

- Return result

Parameter	Description
responseTypes	List of response types supported by WebHCat

- Example

```
curl -ik -u : --negotiate 'http://10.64.35.144:9111/templeton/v1'
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

status (GET)

- Description

  Obtain the status of the current server.

- URL

  http://www.myserver.com/templeton/v1/status

- Parameter

  None

- Return result

Parameter	Description
status	If the WebChat connection is normal, **OK** is returned.
version	Character string, including the version number, for example, v1

- Example
  ```
  curl -ik -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/status'
  ```

  NOTE

  - The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
  - In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
  - In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

version (GET)

- Description

  Obtain the WebHCat version of the server.
- URL

  http://www.myserver.com/templeton/v1/version
- Parameter

  None
- Return result

Parameter	Description
supportedVersions	All supported versions
version	WebHCat version of the server

- Example
  ```
  curl -ik -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/version'
  ```

  NOTE

  - The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
  - In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
  - In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

version/hive (GET)

- Description

  Obtain the Hive version of the server.

- URL

  http://www.myserver.com/templeton/v1/version/hive

- Parameter

  None

- Return result

Parameter	Description
module	hive
version	Hive version

- Example

```
curl -ik -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/version/hive'
```

  NOTE

  - The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
  - In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
  - In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

version/hadoop (GET)

- Description

  Obtain the Hadoop version of the server.

- URL

  http://www.myserver.com/templeton/v1/version/hadoop

- Parameter

  None

- Return result

Parameter	Description
module	hadoop
version	Hadoop version

- Example

```
curl -ik -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/version/hadoop'
```


NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

ddl (POST)

- Description

  Execute a DDL statement.

- URL

  http://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**.

- Return 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 -ik -u : --negotiate -d exec="show tables" 'http://10.64.35.144:9111/templeton/v1/ddl?user.name=user1'
```


NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

ddl/database (GET)

- Description

  List all databases.

- URL

  http://www.myserver.com/templeton/v1/ddl/database

- Parameter

Parameter	Description
like	Regular expression used to match the database name

- Return result

Parameter	Description
databases	Database name

- Example

```
curl -ik -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/ddl/database?user.name=user1'
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

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	Database name

- Return result

Parameter	Description
location	Database location
comment	Database remarks. If there are no database remarks, the value does not exist.
database	Database name
owner	Database owner
ownertype	Type of the database owner

- Example

```
curl -ik -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/ddl/database/default?user.name=user1'
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

ddl/database/:db (PUT)

- Description

  Create a database.

- URL

  http://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

Parameter	Description
comment	Database remarks, for example, description
properties	Database properties

- Return result

Parameter	Description
database	Name of the newly created database

- Example

```
curl -ik -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'
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

ddl/database/:db (DELETE)

- Description

  Delete a database.

- URL

  http://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.

- Return result

Parameter	Description
database	Name of the deleted database

- Example

```
curl -ik -u : --negotiate -X DELETE 'http://10.64.35.144:9111/templeton/v1/ddl/database/db3?ifExists=true&user.name=user1'
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

ddl/database/:db/table (GET)

- Description

  List all tables in the database.

- URL

  http://www.myserver.com/templeton/v1/ddl/database/:db/table

- Parameter

Parameter	Description
:db	Database name
like	Regular expression used to match a table name

- Return result

Parameter	Description
database	Database name
tables	List of tables in the database

- Example

```
curl -ik -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/ddl/database/default/table?user.name=user1'
```


NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

ddl/database/:db/table/:table (GET)

- Description

  Obtain details about a table.

- URL

  http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table

  http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table?format=extended

- Parameter

Parameter	Description
:db	Database name
:table	Table name
format	Set "format=extended" to view more information about the table. (The function is equivalent to "show table extended like tableName" of HiveQL.)

- Return result

Parameter	Description
columns	Column name and type
database	Database name
table	Table name
partitioned	Indicates whether a table is a partition table. This parameter is available only when the table format is **extended**.
location	Table location. This parameter is available only when the table format is **extended**.

Parameter	Description
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 -ik -u : --negotiate 'http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1?format=extended&user.name=user1'
```

  NOTE

  - The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
  - In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
  - In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

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	Database name
:table	New table name
group	User group used for creating the table
permissions	Permission used for creating the table

Parameter	Description
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
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 the **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	Path in HDFS
tableProperties	Table property names and values (name-value pairs)

- Return result

Parameter	Description
database	Database name
table	Table name

- Example

```
curl -ik -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'
```


NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

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	Database name
:table	Existing table name
rename	New table name

- Return result

Parameter	Description
database	Database name
table	New table name

- Example

```
curl -ik -u : --negotiate -d rename=table1 'http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/tbl1?user.name=user1'
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

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	Database name
:table	Table name
ifExists	If this parameter is set to **true**, no error is reported.

- Return result

Parameter	Description
database	Database name
table	Table name

- Example

```
curl -ik -u : --negotiate -X DELETE 'http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/table2?ifExists=true&user.name=user1'
```

  NOTE

  - The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
  - In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
  - In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

ddl/database/:db/table/:existingtable/like/:newtable (PUT)

- Description

  Create a table that is the same as an existing table.

- URL

  http://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

Parameter	Description
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	Path in HDFS

- Return result

Parameter	Description
database	Database name
table	Table name

- Example

```
curl -ik -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'
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

ddl/database/:db/table/:table/partition(GET)

- Description

  List information about all partitions of a table.

- URL

  http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition

- Parameter

Parameter	Description
:db	Database name

Parameter	Description
:table	Table name

- Return result

Parameter	Description
database	Database name
table	Table name
partitions	List of partition attribute values and partition names

- Example

```
curl -ik -u : --negotiate http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/x1/partition?user.name=user1
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

ddl/database/:db/table/:table/partition/:partition(GET)

- Description

  List information about a specific partition of a table.

- URL

  http://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**.

- Return result

Parameter	Description
database	Database name
table	Table name
partition	Partition name
partitioned	If this parameter is set to **true**, the table is a partition table.
location	Storage path of the table
outputFormat	Output format
columns	Column name, type, and remarks
owner	Owner
partitionColumns	Partition column
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 -ik -u : --negotiate http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/x1/partition/dt=1?user.name=user1
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

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	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.

- Return result

Parameter	Description
database	Database name
table	Table name
partitions	Partition name

- Example

```
curl -ik -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
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

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	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**.

- Return result

Parameter	Description
database	Database name
table	Table name
partitions	Partition name

- Example

```
curl -ik -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
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

ddl/database/:db/table/:table/column(GET)

- Description

  Obtain a column list of a table.

- URL

  http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/column

- Parameter

Parameter	Description
:db	Database name

Parameter	Description
:table	Table name

- Return result

Parameter	Description
database	Database name
table	Table name
columns	List of column names and types

- Example

```
curl -ik -u : --negotiate http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/column?user.name=user1
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

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	Database name
:table	Table name
:column	Column Name

- Return result

Parameter	Description
database	Database name
table	Table name

Parameter	Description
column	Column name and type

- Example

```
curl -ik -u : --negotiate http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/column/id?user.name=user1
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

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	Database name
:table	Table name
:column	Column Name
type	Column type, for example, string and int.
comment	Column remarks, for example, description.

- Return result

Parameter	Description
database	Database name
table	Table name
column	Column name

- Example


```
curl -ik -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
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

ddl/database/:db/table/:table/property(GET)

- Description

  Obtain properties of a table.

- URL

  http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/property

- Parameter

Parameter	Description
:db	Database name
:table	Table name

- Return result

Parameter	Description
database	Database name
table	Table name
properties	List of properties

- Example

```
curl -ik -u : --negotiate http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/property?user.name=user1
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

ddl/database/:db/table/:table/property/:property(GET)

- Description

  Obtain a specific property value of a table.

- URL

  http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/property/:property

- Parameter

Parameter	Description
:db	Database name
:table	Table name
:property	Property

- Return result

Parameter	Description
database	Database name
table	Table name
property	Property

- Example

```
curl -ik -u : --negotiate http://10.64.35.144:9111/templeton/v1/ddl/database/default/table/t1/property/last_modified_by?user.name=user1
```

  NOTE

  - The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
  - In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
  - In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

ddl/database/:db/table/:table/property/:property(PUT)

- Description

  Add a property value to a table.

- URL

  http://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

- Return result

Parameter	Description
database	Database name
table	Table name
property	Property name

- Example

```
curl -ik -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
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

mapreduce/jar(POST)

- Description

  Execute a MapReduce job. Before executing a MapReduce job, upload the JAR file of the MapReduce job to HDFS.

- URL

  http://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 (,)

Parameter	Description
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.
statusdir	WebHCat writes the status of the MapReduce job 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.

- Return result

Parameter	Description
id	Job ID, similar to **job_201110132141_0001**

- Example

```
curl -ik -u : --negotiate -d jar="/tmp/word.count-0.0.1-SNAPSHOT.jar" -d
class=com.huawei.word.count.WD -d statusdir="/output" "http://10.64.35.144:9111/templeton/v1/
mapreduce/jar?user.name=user1"
```


NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

mapreduce/streaming(POST)

- Description

  Submit a MapReduce job in Streaming mode.

- URL

  http://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, WebChat 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 **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 job to **statusdir**. If this parameter is set, you need to manually delete it.

Parameter	Description
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.

- Return result

Parameter	Description
id	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://10.64.35.144:50111/templeton/v1/mapreduce/streaming?user.name=user1'
```

NOTE

- Prerequisites are required for using this API. For details, see **Hive Application Development Rules**.
- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

/hive(POST)

- Description

  Run Hive commands.

- URL

http://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 **argument**.
define	This parameter is used to configure Hadoop in the **define=key=value** format.
statusdir	WebHCat writes the status of the MapReduce job 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.

- Return result

Parameter	Description
id	Job ID, similar to **job_201110132141_0001**

- Example

```
curl -ik -u : --negotiate -d execute="select count(*) from t1" -d statusdir="/output" "http://10.64.35.144:9111/templeton/v1/hive?user.name=user1"
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

jobs(GET)

- Description

  Obtain all job IDs.

- URL

  http://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**.

Parameter	Description
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 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. The default value is **false**.

- Return result

Parameter	Description
id	Job id
detail	If the value of **showall** is **true**, details are displayed. Otherwise, the value is null.

- Example

```
curl -ik -u : --negotiate  "http://10.64.35.144:9111/templeton/v1/jobs?user.name=user1"
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

jobs/:jobid(GET)

- Description

  Obtain information about a specified job.

- URL

  http://www.myserver.com/templeton/v1/jobs/:jobid

- Parameter

Parameter	Description
jobid	Job ID, received after a job is created

- Return 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 -ik -u : --negotiate  "http://10.64.35.144:9111/templeton/v1/jobs/job_1440386556001_0255?user.name=user1"
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In MRS 1.9.2 or later, the default port number is **9111**. For details, see **templeton.port** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

jobs/:jobid(DELETE)

- Description

  Kill a job.

- URL

  http://www.myserver.com/templeton/v1/jobs/:jobid

- Parameter

Parameter	Description
:jobid	ID of the job to be deleted

- Return 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 -ik -u : --negotiate -X DELETE  "http://10.64.35.143:9111/templeton/v1/jobs/
job_1440386556001_0265?user.name=user1"
```

NOTE

- The example uses the service IP address of WebHCat node and the WebHCat port configured during the installation as an example.
- In this example, **http** is used for a common cluster and **https** is used for a security cluster. For details, see **templeton.protocol.type** in **Services** > **Hive** > **Service Configuration** on the MRS Manager management page.

9 Impala Development Guide

9.1 Impala Application Development Overview

9.1.1 Introduction to Impala Application Development

Introduction to Impala

Impala provides fast, interactive SQL queries directly on your Apache Hadoop data stored in HDFS, HBase, or Object Storage Service (OBS). In addition to using the same unified storage platform, Impala also uses the same metadata, SQL syntax (Hive SQL), ODBC driver, and user interface (Impala query UI in Hue) as Apache Hive. This provides a familiar and unified platform for real-time or batch-oriented queries. Impala is an addition to tools available for querying big data. It does not replace the batch processing frameworks built on MapReduce such as Hive. Hive and other frameworks built on MapReduce are best suited for long running batch jobs.

Impala provides the following features:

- Most common SQL-92 features of Hive Query Language (HiveQL) including SELECT, JOIN, and aggregate functions
- HDFS, HBase, and OBS storage, including:
  - HDFS file formats: delimited text files, Parquet, Avro, SequenceFile, and RCFile
  - Compression codecs: Snappy, GZIP, Deflate, BZIP
- Common data access interfaces, including:
  - JDBC driver
  - ODBC driver
  - Hue Beeswax and the Impala query UI
- **impala-shell** command line interface
- Kerberos authentication

Impala applies to offline analysis (such as log and cluster status analysis) of real-time data queries, large-scale data mining (such as user behavior analysis, interest region analysis, and region display), and other scenarios.

9.1.2 Common Concepts of Impala Application Development

- **Client**

  Users can access the server from the client through Java APIs and Thrift APIs to perform Impala-related operations. The Impala client in this document refers to the ImpalaImpala client installation directory, which contains sample codes for Impala access using Java APIs.

- **HiveQL**

  Hive Query Language (SQL-like statement, similar to Hive)

- **Statestore**

  The StateStore checks on the health of all Impalad instance in an Impala cluster, and continuously relays its findings to each of those Impalad instances. If an Impalad instance goes offline due to node failure, network error, or other reasons, the StateStore informs all the other Impalad instances so that future queries can avoid making requests to the unreachable Impalad instances.

- **Catalog**

  The Catalog Service relays the metadata changes from each Impalad instance to other Impalad instances in a cluster. The catalog service avoids the need to issue the REFRESH statement on other instances when the metadata changes on an Impalad instance. When you create or modify a table in Hive, you need to issue REFRESH or INVALIDATE METADATA.

9.1.3 Impala Application Development Process

Figure 9-1 and **Table 9-1** describe the phases in the development process.

Figure 9-1 Impala application development process

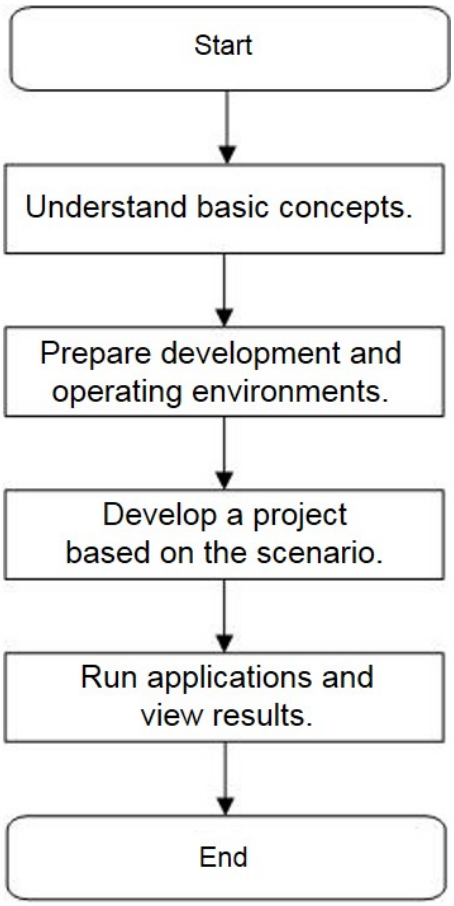


Table 9-1 Description of the Impala application development process

Phase	Description	Reference
Understand basic concepts.	Before application development, learn basic concepts of Impala.	**Common Concepts of Impala Application Development**
Prepare development and operating environments.	You can use Java and Python to develop Impala applications. You are advised to use the Eclipse tool to configure the development environment based on the language.	**Impala Application Development Environment**
Develop a project based on the scenario.	Impala provides Java and Python sample projects, covering table creation, data load, and data queries.	**Impala Development Plan**

Phase	Description	Reference
Run applications and view results.	This phase provides guidance for users to submit a developed application for running and view the result.	**Commissioning an Impala JDBC Application on Linux**

9.2 Preparing an Impala Application Development Environment

9.2.1 Impala Application Development Environment

Table 9-2 describes the local environment required for application development. You also need to prepare an environment for verifying whether the application is running properly.

Table 9-2 Development environment

Item	Description
OS	• Development environment: Windows 7 or later version is recommended. • Operating environment: Linux system
JDK installation	Basic configurations of the development and operating environments. The version requirements are as follows: The server and client of an MRS cluster support only built-in Oracle JDK 1.8, which cannot be replaced. If users' applications need to reference the JAR files of the SDK class in the user application processes, Oracle JDK and IBM JDK are supported. • Oracle JDK versions: 1.7 and 1.8 • IBM JDK versions: 1.7.8.10, 1.7.9.40, and 1.8.3.0

Item	Description
Eclipse installation and configuration	It is a tool used to develop Impalad applications. The version requirements are as follows: • The JDK version is 1.7, and the Eclipse version is 3.7.1 or later. • The JDK version is 1.8, and the Eclipse version is 4.3.2 or later. Note: If you use IBM JDK, ensure that the JDK configured in Eclipse is IBM JDK. If you use Oracle JDK, ensure that the JDK configured in Eclipse is Oracle JDK. Do not use the same workspace and the sample project in the same path for different Eclipse programs.
Network	The client must be interconnected with the Impala server on the network.

9.2.2 Preparing a Local Application Development Environment

- Install Eclipse and JDK in the Windows development environment.

  The JDK version is 1.8, and the Eclipse version is 4.3.2 or later.

  NOTE

  - If you use IBM JDK, ensure that the JDK configured in Eclipse is IBM JDK.
  - If you use Oracle JDK, ensure that the JDK configured in Eclipse is Oracle JDK.
  - If you use ODBC for secondary development, ensure that JDK 1.8 or later is used.
  - Do not use the same workspace and the sample project in the same path for different Eclipse programs.
- Prepare a Linux environment for testing application running status.

Preparing a Running and Commissioning Environment

Step 1 On the ECS management console, apply for a new ECS for user application development, running, and commissioning.

- The security group of the ECS must be the same as that of the Master node in an MRS cluster.
- The ECS and the MRS cluster must be in the same VPC.
- The ECS NIC and the MRS cluster must be in the same network segment.

Step 2 Apply for an EIP, bind it, and configure an inbound or outbound rule for the security group.

Step 3 Download a client program.

1. Log in to **MRS Manager**.
2. Choose **Services** > **Download Client** to download the complete client to the remote host, that is, download the client program to the newly applied ECS.

Step 4 Install a cluster client as user **root**.

1. Run the following command to decompress the client package:

   tar -xvf /opt/MRS_Services_Client.tar

2. Run the following command to verify the installation file package:

   sha256sum -c /opt/MRS_Services_ClientConfig.tar.sha256

   ```
   MRS_Services_ClientConfig.tar:OK
   ```

3. Run the following command to decompress the installation file package:

   tar -xvf /opt/MRS_Services_ClientConfig.tar

4. Run the following command to install the client to a specified directory (absolute path), for example, **/opt/client**. The directory is automatically created.

   cd /opt/MRS_Services_ClientConfig

   sh install.sh /opt/client

   ```
   Components client installation is complete.
   ```

Step 5 Run the following command to update client configurations:

sh /opt/client/refreshConfig.sh *Client installation directory Full path of the client configuration file package*

Example:

sh /opt/client/refreshConfig.sh /opt/client /opt/MRS_Services_Client.tar

NOTE

If you modify component parameter configurations, you need to download the client configuration file again and update the client in the running and commissioning environment.

----End

9.2.3 Preparing an Impala Application Development User

The development user is used to run the sample project. The user must have Impala permissions to run the Impala sample project.

Prerequisites

Kerberos authentication has been enabled for the MRS cluster. Skip this step if Kerberos authentication is not enabled for the cluster.

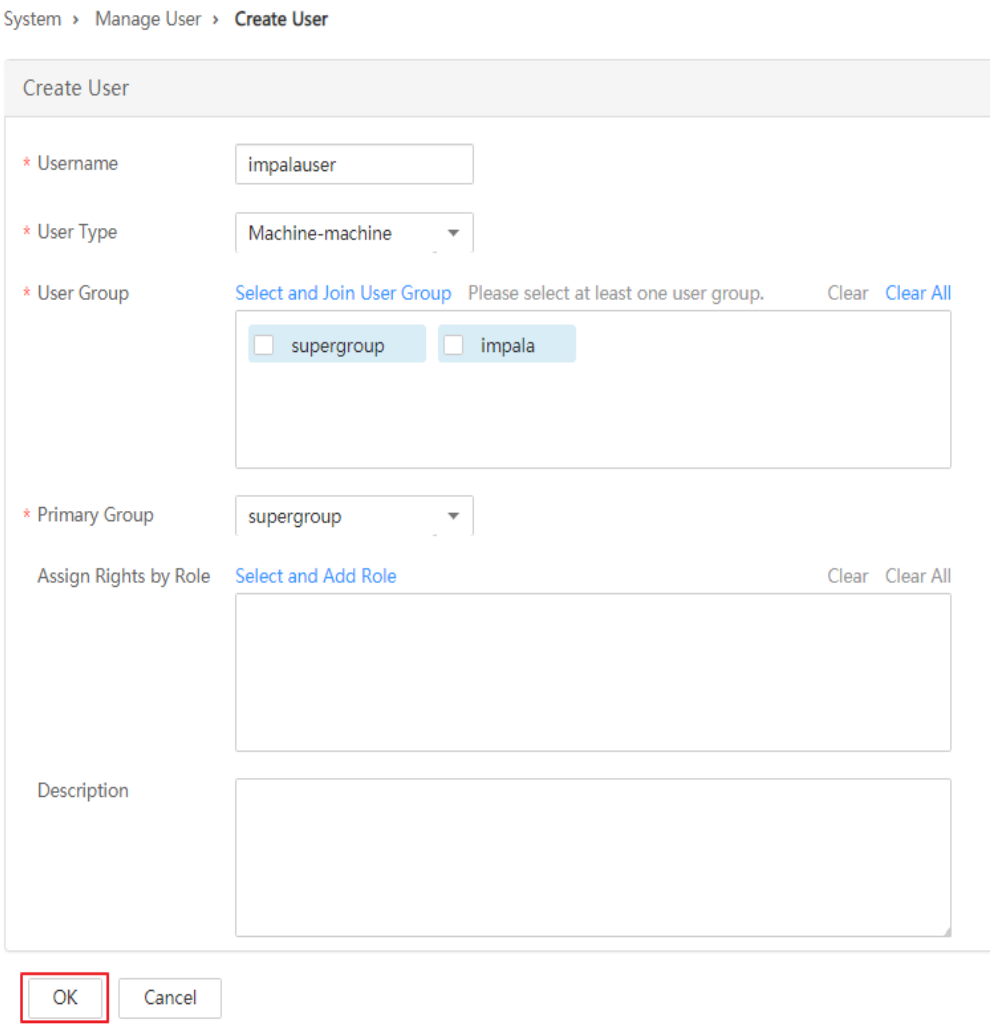
Procedure

Step 1 Log in to **MRS Manager**.

Step 2 Choose **System** > **Manage User** > **Create User** to create a user for the sample project.

Step 3 Enter a username, for example, *impalauser*. Set **User Type** to **Machine-machine**, and select **impala** and **supergroup** in **User Group**. Set **Primary Group** to **supergroup** and click **OK**. Figure 9-2 shows the parameter settings.

Figure 9-2 Creating a user



Step 4 On MRS Manager, choose **System** > **Manage User**. In the **Operation** column corresponding to username **impalauser**, choose **More** > **Download authentication credential**. Save the file and decompress it to obtain the **user.keytab** and **krb5.conf** files. The two files are used for security authentication in the sample project.

----End

Related Information

If you modify component parameter configurations, you need to download the client configuration file again and update the client in the running and commissioning environment.

9.2.4 Preparing the Impala JDBC Client

To run the JDBC API sample code of Impala, you need to perform the following operations.

NOTE

The following example develops an application that uses JDBC to connect to Impala in the Windows environment.

Procedure

Step 1 Obtain the Impala sample project by referring to **Obtaining the MRS Application Development Sample Project**.

Step 2 In the root directory of the Impala sample project, run the **mvn install** command to perform compilation.

Step 3 In the root directory of the Impala sample project, run the **mvn eclipse:eclipse** command to create an Eclipse project.

Step 4 In the application development environment, import the sample project to the Eclipse development environment.

1. Choose **File** > **Import** > **General** > **Existing Projects into Workspace** > **Next** > **Browse**.

   The **Browse Folder** dialog box is displayed.

2. Select the **impala-examples** folder. On Windows, the folder path cannot contain any space.

   Click **Finish**.

   After successful import, the **JDBCExample** class in **com.huawei.bigdata.impala.example** is the JDBC API sample code.

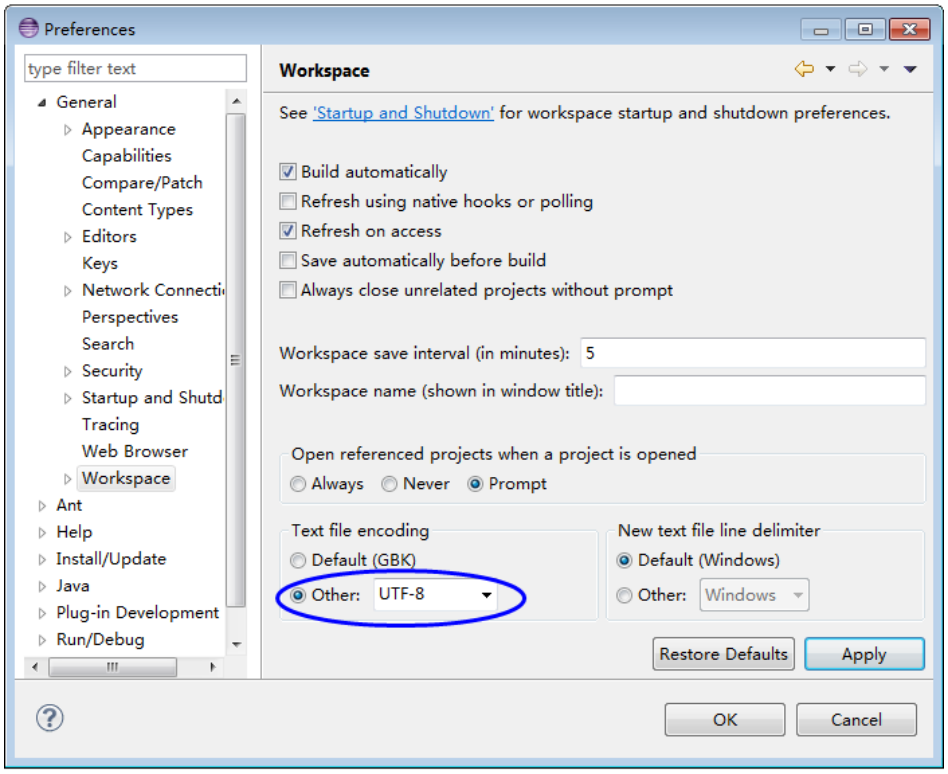
Step 5 Set an Eclipse text file encoding format to prevent garbled characters.

1. On the Eclipse menu bar, choose **Window** > **Preferences**.

   The **Preferences** window is displayed.

2. In the navigation pane on the left, choose **General** > **Workspace**. In the **Text file encoding** area, select **Other** and set the value to **UTF-8**. Click **Apply** and then **OK**. **Figure 9-3** shows the settings.

Figure 9-3 Setting the Eclipse encoding format



Step 6 Modify the sample. You can skip this step for a cluster with Kerberos authentication disabled.

After you obtain the **krb5.conf** and **user.keytab** files of the new development user in **Step 4**, change the value of **userName** in **ExampleMain.java** to the new username, for example, **impalauser**.

```
/**
 * Other way to set conf for zk. If use this way,
 * can ignore the way in the 'login' method
 */
if (isSecurityMode) {
    userName = "impalauser";
    userKeytabFile = CONF_DIR + "user.keytab";
    krb5File = CONF_DIR + "krb5.conf";
    conf.set(HADOOP_SECURITY_AUTHENTICATION, "kerberos");
    conf.set(HADOOP_SECURITY_AUTHORIZATION, "true");
```

----End

9.3 Developing Impala Applications

9.3.1 Impala Development Plan

Scenario Description

Assume that you need to develop an Impala data analysis application to manage the employee information of an enterprise. **Table 9-3** and **Table 9-4** provide employee information.

Development Guidelines

Step 1 Prepare data.

1. Create three tables: employee information table **employees_info**, employee contact table **employees_contact**, and extended employee information table **employees_info_extended**.
   - Employee information table **employees_info** contains fields such as employee ID, name, salary currency, salary, tax category, work place, and hire date. In salary currency, R indicates RMB and D indicates USD.
   - Fields in the **employees_contact** table include the employee ID, phone number, and email address.
   - Fields in the **employees_info_extended** table include the employee ID, name, mobile phone number, e-mail address, salary currency, salary, tax category, and work place. The partition field is the hire date.

     For details about table creation codes, see **Creating an Impala Table**.
2. Load employee information to **employees_info**.

   For details about data loading codes, see **Loading Impala Data**.

   Table 9-3 provides employee information.

   Table 9-3 Employee information

Employee ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hire Date
1	Wang	R	8000.01	personal income tax&0.05	China:Shenzhen	2014
3	Tom	D	12000.02	personal income tax&0.09	America:NewYork	2014
4	Jack	D	24000.03	personal income tax&0.09	America:Manhattan	2014
6	Linda	D	36000.04	personal income tax&0.09	America:NewYork	2014
8	Zhang	R	9000.05	personal income tax&0.05	China:Shanghai	2014

3. Load employee contact information to **employees_contact**.

   Table 9-4 provides employee contact information.

Table 9-4 Employee contact information

Employee ID	Phone Number	Email
1	135 XXXX XXXX	xxxx@xx.com
3	159 XXXX XXXX	xxxxx@xx.com.cn
4	186 XXXX XXXX	xxxx@xx.org
6	189 XXXX XXXX	xxxx@xxx.cn
8	134 XXXX XXXX	xxxx@xxxx.cn

Step 2 Analyze data.

For details about data analysis codes, see **Querying Impala Data**.

- Query contact information of employees whose salaries are paid in USD.
- Query the IDs and names of employees who were hired in 2014, and load the query results to the partition with the hire date of 2014 in the **employees_info_extended** table.
- Collect the number of records in the **employees_info** table.
- Query information about employees whose email addresses end with "cn".

Step 3 Submit a data analysis task to collect the number of records in the **employees_info** table. For details, see **Analyzing Impala Data**.

----End

9.3.2 Creating an Impala Table

Function Description

This section describes how to use Impala SQL to create internal and external tables. You can create a table by using any of the following methods:

- Customize the table structure, and use the key word **EXTERNAL** to differentiate between internal and external tables.
  - If all data is to be processed by Impala, create an internal table. When an internal table is deleted, the metadata and data in the table are deleted together.
  - If data is to be processed by multiple tools (such as Pig), create an external table. When an external table is deleted, only metadata is deleted.
- Create a table based on existing tables. Use CREATE LIKE to fully copy the original table structure, including the storage format.
- Create a table based on query results using CREATE AS SELECT.

  This method is flexible. By using this method, you can specify fields (except for the storage format) that you want to copy to the new table when copying the structure of the existing table.

Sample Code

```
-- Create the external table employees_info.
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info
(
id INT,
name STRING,
usd_flag STRING,
salary DOUBLE,
deductions MAP<STRING, DOUBLE>,
address STRING,
entrytime STRING
)
-- Specify the field delimiter.
-- "delimited fields terminated by" indicates that a column delimiter is a comma (,).
ROW FORMAT delimited fields terminated by ','
-- Specify the table storage format to TEXTFILE.
STORED AS TEXTFILE;

-- Use CREATE Like to create a table.
CREATE TABLE employees_like LIKE employees_info;

-- Run the DESCRIBE command to check the structures of the employees_info and employees_like tables.
DESCRIBE employees_info;
DESCRIBE employees_like;

-- Create the internal table employees_info.
CREATE TABLE IF NOT EXISTS employees_info
(
id INT,
name STRING,
usd_flag STRING,
salary DOUBLE,
deductions MAP<STRING, DOUBLE>,
address STRING,
entrytime STRING
)
-- Specify the field delimiter.
-- "delimited fields terminated by" indicates that a column delimiter is a comma (,).
ROW FORMAT delimited fields terminated by ','
-- Specify the table storage format to TEXTFILE.
STORED AS TEXTFILE;
```

Extended Applications

- Create a partition table.

  A table may have one or more partitions. Each partition is saved as an independent folder in the table directory. Partitions help minimize the query scope, accelerate data query, and allow users to manage data based on certain criteria.

  A partition is defined using the PARTITIONED BY clause during table creation.

  ```
  CREATE EXTERNAL TABLE IF NOT EXISTS employees_info_extended
  (
  id INT,
  name STRING,
  usd_flag STRING,
  salary DOUBLE,
  deductions MAP<STRING, DOUBLE>,
  address STRING
  )
  -- Use "PARTITIONED BY" to specify the column name and data type of the partition.
  PARTITIONED BY (entrytime STRING)
  STORED AS TEXTFILE;
  ```

- Update the table structure.

After a table is created, you can use ALTER TABLE to add or delete fields, modify table attributes, and add partitions.

```
-- Add the tel_phone and email fields to the employees_info_extended table.
 ALTER TABLE employees_info_extended ADD COLUMNS (tel_phone STRING, email STRING);
```

- Enable Impala to use OBS.

  You need to set the specified parameters for **core-site.xml** on the cluster management page of MRS Manager. You can log in to OBS console and access the **My Credential** page to obtain the AK/SK.

  ```
  fs.obs.access.key=AK;
  fs.obs.secret.key=SK;
  fs.obs.endpoint=endpoint;
  ```

  Set the storage type of the new table to obs.

  ```
  create table obs(c1 string, c2 string) stored as parquet location 'obs://obs-lmm/hive/orctest'
  tblproperties('orc.compress'='SNAPPY');
  ```

  NOTE

  When Impala uses OBS to store data, partition and table storage locations in the same table cannot be stored in different buckets.

  For example, create a partition table and set its storage location to the folder in OBS bucket 1. In this case, modifying the storage location of the table partition does not take effect. When data is inserted, the storage location of the table is used.

  1. Create a partition table and specify the path for storing the table.
     ```
     create table table_name(id int,name string,company string) partitioned by(dt date) row format delimited fields terminated by ',' stored as textfile location "obs://OBS bucket 1/Folder in the bucket";
     ```
  2. Modifying the storage location of the table partition to another bucket does not take effect.
     ```
     alter table table_name partition(dt date) set location "obs://OBS bucket 2/Folder in the bucket";
     ```

9.3.3 Loading Impala Data

Function Description

This section describes how to use Impala SQL to load data to the existing **employees_info** table. You can learn how to load data from a cluster.

Sample Code

```
--Load the employee_info.txt file from the /opt/impala_examples_data/ directory of the local file system to the employees_info table.
LOAD DATA LOCAL INPATH '/opt/impala_examples_data/employee_info.txt' OVERWRITE INTO TABLE employees_info;

-- Load /user/impala_examples_data/employee_info.txt from HDFS to the employees_info table.
LOAD DATA INPATH '/user/impala_examples_data/employee_info.txt' OVERWRITE INTO TABLE employees_info;
```

NOTE

The essence of loading data is to copy the data to the specified table directory in HDFS.

The **LOAD DATA LOCAL INPATH** command can be used to load files from a local file system to Impala. If **LOCAL** is specified, the path refers to the path of the local file system of the currently connected **Impalad**.

9.3.4 Querying Impala Data

Function Description

This section describes how to use Impala SQL to query and analyze data. You can query and analyze data using the following methods:

- Use common features of a SELECT query, such as JOIN.
- Load data to a specified partition.
- Use built-in functions of Impala.
- Query and analyze data using user-defined functions. For details about how to create and define functions, see **Developing User-Defined Impala Functions**.

Sample Code

```
-- Query contact information of employees whose salaries are paid in USD.
SELECT
a.name,
b.tel_phone,
b.email
FROM employees_info a JOIN employees_contact b  ON(a.id = b.id) WHERE usd_flag='D';

-- Query the IDs and names of employees who were hired in 2014, and load the query results to the partition with the hire date of 2014 in the employees_info_extended table.
INSERT OVERWRITE TABLE employees_info_extended PARTITION (entrytime = '2014')
SELECT
a.id,
a.name,
a.usd_flag,
a.salary,
a.deductions,
a.address,
b.tel_phone,
b.email
FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE a.entrytime = '2014';

-- Use the existing function COUNT() in Impala to calculate the number of records in the employees_info table.
SELECT COUNT(*) FROM employees_info;

-- Query information about employees whose email addresses end with "cn".
SELECT a.name, b.tel_phone FROM  employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE b.email like '%cn';
```

Extended Applications

For details about user-defined functions, see **Developing User-Defined Impala Functions**.

9.3.5 Analyzing Impala Data

Function Description

This section describes how to use a sample program to complete an analysis task. This section uses JDBC APIs as an example to describe how to submit a data analysis task.

Sample Code

If you submit a data analysis task using Impala JDBC APIs, refer to **JDBCExample.java** in the sample program.

1. Change the value of the following variable to **false**, specifying that the authentication mode for the connected clusters is normal mode.

```
// Indicates whether the authentication mode of the connected cluster is security mode.
  boolean isSecureVer = false;
```

2. Define Impala SQL. Impala SQL must be a single statement and cannot contain ";".

```
  // Define HQL, which cannot contain the semicolon (;).
  String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT,name STRING)",
          "SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};
```

3. Build JDBC URL.

```
// Build JDBC URL.
StringBuilder sBuilder = new StringBuilder(
  "jdbc:hive2://").append("impalad_ip").append("/");

if (isSecurityMode) {
    // Security mode
    sBuilder.append(";auth=")
            .append(clientInfo.getAuth())
            .append(";principal=")
            .append(clientInfo.getPrincipal())
            .append(";");
} else {
    // Normal mode
    sBuilder.append(";auth=noSasl");
}
String url = sBuilder.toString();
```

NOTE

If an Impalad instance is directly connected, an Impalad instance fault will cause Impala access failure.

4. Load the Hive JDBC driver.

```
  // Load the Hive JDBC driver.
  Class.forName(HIVE_DRIVER);
```

5. Enter a correct username, obtain the JDBC connection, confirm the Impala SQL type (DDL/DML), call APIs to run Impala SQL, return the queried column name and result to the console, and close the JDBC connection.

```
  Connection connection = null;
    try {
      // Obtain the JDBC connection.
      // If you set the second parameter to an incorrect username, the anonymous user will be used for login.
      connection = DriverManager.getConnection(url, "userName", "");

      // Create a table.
      // To import data to a table after the table is created, you can use the LOAD statement. For example, import data from HDFS to the table.
      //load data inpath '/tmp/employees.txt' overwrite into table employees_info;
      execDDL(connection,sqls[0]);
      System.out.println("Create table success!");

      // Query
      execDML(connection,sqls[1]);

      // Delete the table.
      execDDL(connection,sqls[2]);
      System.out.println("Delete table success!");
    }
```



```
    finally {
      // Close the JDBC connection.
      if (null != connection) {
        connection.close();
      }

 public static void execDDL(Connection connection, String sql)
   throws SQLException {
     PreparedStatement statement = null;
     try {
       statement = connection.prepareStatement(sql);
       statement.execute();
     }
     finally {
       if (null != statement) {
         statement.close();
       }
     }
   }

   public static void execDML(Connection connection, String sql) throws SQLException {
     PreparedStatement statement = null;
     ResultSet resultSet = null;
     ResultSetMetaData resultMetaData = null;

     try {
// Run the Impala SQL statement.
       statement = connection.prepareStatement(sql);
       resultSet = statement.executeQuery();

       // Output the queried column name to the console.
       resultMetaData = resultSet.getMetaData();
       int columnCount = resultMetaData.getColumnCount();
       for (int i = 1; i <= columnCount; i++) {
         System.out.print(resultMetaData.getColumnLabel(i) + '\t');
       }
       System.out.println();

       // Output the query result to the console.
       while (resultSet.next()) {
         for (int i = 1; i <= columnCount; i++) {
           System.out.print(resultSet.getString(i) + '\t');
         }
         System.out.println();
       }
     }
     finally {
       if (null != resultSet) {
         resultSet.close();
       }

       if (null != statement) {
         statement.close();
       }
     }
   }
```

9.3.6 Developing User-Defined Impala Functions

When built-in functions of Impala cannot meet requirements, you can compile user-defined functions (UDFs) and use them for query.

According to implementation methods, UDFs are classified as follows:

- Common UDFs: used to perform operations on a single data row and export a single data row.

- User-defined aggregating functions (UDAFs): used to input multiple data rows and export a single data row.
- User-defined table-generating functions (UDTFs): used to perform operations on a single data row and export multiple data rows. **Impala does not support this type of UDFs.**

According to use methods, UDFs are classified as follows:

- Temporary functions: used only in the current session and must be recreated after a session restarts.
- Permanent functions: used in multiple sessions. You do not need to create them every time a session restarts.

Impala supports development of Java UDFs and reuse of UDFs developed by Hive. The prerequisite is that the data types supported by Impala are used. For details about the data types supported by Impala, visit **http://impala.apache.org/docs/build3x/html/topics/impala_datatypes.html**.

In addition, Impala supports UDFs written in C++, which provides better performance than Java UDFs.

Example

The following is an example of reusing the **lower()** function:

```
[localhost:21000] > create database udfs;
[localhost:21000] > use udfs;
[localhost:21000] > create function my_lower(string)
returns string location '/user/hive/udfs/hive.jar'
symbol='org.apache.hadoop.hive.ql.udf.UDFLower';
[localhost:21000] > select my_lower('Some String NOT ALREADY LOWERCASE');
+----------------------------------------------------+
| udfs.my_lower('some string not already lowercase') |
+----------------------------------------------------+
| some string not already lowercase |
+----------------------------------------------------+
Returned 1 row(s) in 0.11s
[localhost:21000] > create table t2 (s string);
[localhost:21000] > insert into t2 values ('lower'),('UPPER'),('Init cap'),('CamelCase');
Inserted 4 rows in 2.28s
[localhost:21000] > select * from t2;
+-----------+
| s |
+-----------+
| lower |
| UPPER |
| Init cap |
| CamelCase |
+-----------+
Returned 4 row(s) in 0.47s
[localhost:21000] > select my_lower(s) from t2;
+------------------+
| udfs.my_lower(s) |
+------------------+
| lower |
| upper |
| init cap |
| camelcase |
+------------------+
Returned 4 row(s) in 0.54s
```


9.4 Commissioning an Impala Application

9.4.1 Commissioning an Impala JDBC Application on Windows

Running the JDBC Client Using CLI

Step 1 Run the sample project.

After importing and modifying the sample project according to **Preparing the Impala JDBC Client**, save the **keytab** file obtained from MRS Manager of the cluster to the **conf** directory of the sample project, that is, **impala-examples/conf**. You can skip this step for a common cluster. In the development environment (for example, Eclipse), right-click **JDBCExample.java** and choose **Run as** > **Java Application** from the shortcut menu to run the corresponding application project.

NOTE

You can use either of the following method to access an MRS cluster to operate Impala on Windows.

- Apply for a Windows ECS to access the MRS cluster to operate Impala.
- Use the local host to access the MRS cluster to operate Impala.

Method 1: Apply for a Windows ECS to access the MRS cluster to operate Impala. Run the sample code after the development environment is installed. To apply for ECS to access the MRS cluster, perform the following steps:

1. On the **Active Clusters** page, click the name of an existing cluster.

   On the cluster details page, record the **AZ**, **VPC**, and **Default Security Group** of the Master node.

2. On the ECS management console, create a new ECS.

   The **AZ**, **VPC**, and **security group** of ECS must be the same as those of the cluster to be accessed.

   Select a Windows public image.

   For details about other configuration parameters, see **Elastic Cloud Server** > **Quick Start** > **Purchasing and Logging In to a Windows ECS**.

Method 2: Use the local host to access the MRS cluster to operate Impala. After installing the development environment and completing the following steps, run the sample code.

1. Bind an EIP address to any Core node. Then, configure the IP address in the **impala-server** configuration item in the **client.properties** file of the development sample to access the Impala service and submit SQL statements. To bind an EIP, perform the following steps:
   1. On the VPC management console, apply for an EIP and bind it to ECS.

      For details, see **Virtual Private Cloud** > **User Guide** > **Elastic IP Address** > **Assigning an EIP and Binding It to an ECS**.

   2. Open security group rules for the MRS cluster.

      Add security group rules to the security groups of Master and Core nodes in the cluster so that ECS can access the cluster. For details, see **Virtual Private Cloud** > **User Guide** > **Security** > **Security Group** > **Adding a Security Group Rule**.

2. Modify the IP addresses of the **kdc**, **admin_server**, and **kpasswd_server** parameters in the **krb5.conf** file of the imported sample to the EIPs corresponding to KrbServer. (The Kerberos service is deployed on the master node by default. Therefore, the public IP address of the master node is used.) (Skip this step for common clusters with the Kerberos function disabled.)

The **client.properties** configuration in the sample is as follows:

```
auth = KERBEROS ## Kerberos mode
principal = impala/node-ana-corexphm@10530B19_8446_4846_97BD_87880A2535DF.COM  ## Principal used by the Impalad instance to be connected
impala-server = XX.XX.XX.XX:21050 ## Specifies the service address bound to the Core node where the Impalad instance to be connected is located. If method 2 is used, enter the EIP bound in step 1.
```

Step 2 View the execution result.

View the Impala SQL query results in the sample code. If the following information is displayed, the execution is successful.

Result:

```
Create table success!
_c0
0
Delete table success!
```

----End

9.4.2 Commissioning an Impala JDBC Application on Linux

Step 1 Create a directory as the running directory in the running and commissioning environment, for example, **/opt/impala_examples** (Linux), and create the **conf** subdirectory in the directory.

Step 2 Run the **mvn package** command to obtain a JAR package, for example, **impala-examples-mrs-2.1-jar-with-dependencies.jar** obtained from the **target** directory of the project and copy the package to the **/opt/impala_examples** directory.

Step 3 For a security cluster with Kerberos authentication enabled, copy the **user.keytab** and **krb5.conf** files obtained in **Step 4** to the **/opt/impala_examples/conf** directory. Skip this step for a common cluster.

Step 4 In Linux, run the following command to run the sample program:

```
chmod +x /opt/impala_examples -R
cd /opt/impala_examples
java -cp impala-examples-mrs-2.1-jar-with-dependencies.jar
com.huawei.bigdata.impala.example.ExampleMain
```

Step 5 In the CLI, view the Impala SQL query results in the example code.

If the following information is displayed, the sample project execution is successful on Linux.

```
Create table success!
_c0
0
Delete table success!
```

----End

9.5 FAQs About Impala Application Development

9.5.1 Impala JDBC APIs

Impala uses Hive JDBC APIs that comply with the Java JDBC driver standard. For details, see JDK 1.7 API.

NOTE

Impala does not support all standard Hive JDBC APIs. SQLException "**Method not supported**" is generated when some operations are performed.

9.5.2 Impala SQL APIs

Impala SQL is highly compatible with HiveQL. For details, visit **https://impala.apache.org/docs/build/html/topics/impala_langref.html**.

10 Kafka Development Guide

10.1 Kafka Application Development Overview

10.1.1 Introduction to Kafka Application Development

Introduction to Kafka

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 has the following features:

- High throughput
- Message persistence to disks
- Scalable distributed system
- High fault tolerance
- Support for online and offline scenarios

API Types

APIs provided by Kafka can be divided into Producer APIs and Consumer APIs. Both types support Java APIs. For details, see **Kafka Java APIs**.

10.1.2 Common Concepts of Kafka Application Development

- Topic

  Messages of the same type maintained by the Kafka are called a topic.
- Partition

  Each topic can be divided into multiple partitions. Each partition corresponds to an appendant log file whose sequence is fixed.

- Producer

  The role that sends messages to a Kafka topic is called Producer.
- Consumer

  The role that obtains messages from Kafka topics is called Consumer.
- Broker

  Each node server in the Kafka cluster is called a Broker.

10.1.3 Kafka Application Development Process

Kafka client roles include Producer and Consumer, which share the same application development process.

Figure 10-1 and **Table 10-1** show each phase of the development process.

Figure 10-1 Kafka client application development process

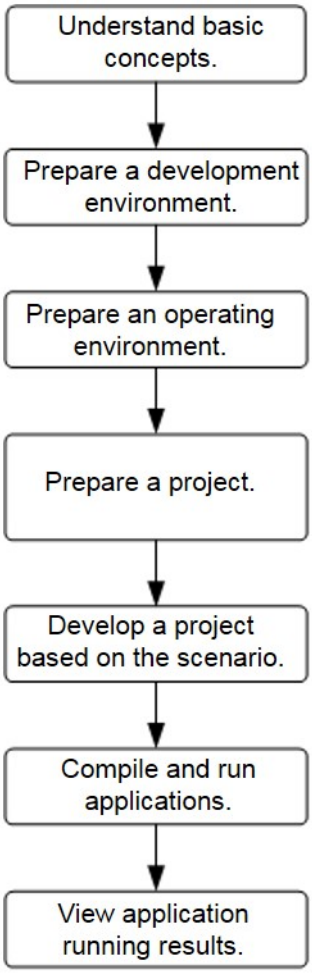


Table 10-1 Kafka client application development process description

Phase	Description	Reference
Understand basic concepts.	Before developing an application, learn basic concepts of Kafka, and determine whether the desired role is Producer or Consumer based on the actual scenario.	**Common Concepts of Kafka Application Development**
Prepare the development environment.	The Java language is recommended for the development of Kafka client applications, and Maven is recommended for constructing projects.	**Preparing the Maven and JDK**
Prepare the operating environment.	The running environment of the Kafka sample application consists of nodes of the VPC cluster where the MRS service is deployed.	-
Prepare a project.	Kafka provides sample projects for different scenarios. You can download a sample project for learning. You can also create a Kafka project according to the guide.	**Importing and Configuring Kafka Sample Projects**
Develop a project based on the scenario.	Producer and Consumer API usage samples are provided and cover old APIs, new APIs, and multi-thread usage scenarios, helping you quickly know Kafka APIs well.	**Kafka Development Plan**
Compile and run an application.	Compile and Compress a developed application. Then upload it to a Linux node in the VPC cluster for running.	**Commissioning a Kafka Application**

Phase	Description	Reference
View application running results.	Application running results can be output to the Linux CLI page. You can also use a Linux client to consume the topic data to check whether the application running results are successfully written.	**Commissioning a Kafka Application**

10.2 Preparing a Kafka Application Development Environment

10.2.1 Kafka Application Development Environment

The following table lists the development environment required for Kafka development.

Table 10-2 Development environment

Item	Description
OS	Windows OS. Windows 7 or later is recommended.
Installation of JDK and Maven	Basic configurations of the development environment. Oracle JDK versions: 1.7 and 1.8. Apache Maven versions: 3.3.0 and later
Installation and configuration of Eclipse or IntelliJ IDEA	Tool used for developing Kafka applications
Network	Ensure that at least one node of the VPC where the Kafka service is deployed is connected to your local host.
Security authentication for accessing the ECS	You can log in to the Linux ECS using a key or password.

10.2.2 Preparing the Maven and JDK

Scenario

The development environment is set up on Windows OS.

Procedure

1. The requirement of Eclipse installation in the development environment is as follows:
   - The Eclipse version is 3.0 or later.
   - The IntelliJ IDEA version is 15.0 or later.
2. The requirement of JDK installation in the development environment is as follows:

   The JDK version is 1.7 or 1.8. IBM JDK and Oracle JDK are supported.

   NOTE

   - If the IBM JDK is used, ensure that the JDK configured in Eclipse or IntelliJ IDEA is the IBM JDK.
   - If the Oracle JDK is used, ensure that the JDK configured in Eclipse or IntelliJ IDEA is the Oracle JDK.
   - Do not use the same workspace and the sample project in the same path for different Eclipse programs.
3. If the Maven environment is to be installed, use version 3.0.0 or later.

10.2.3 Importing and Configuring Kafka Sample Projects

Procedure

Step 1 Download the sample project to the local computer by referring to **Obtaining the MRS Application Development Sample Project**.

Step 2 Decompress the sample project and locate the **kafka-examples** directory.

Step 3 Import the sample project to the Eclipse development environment.

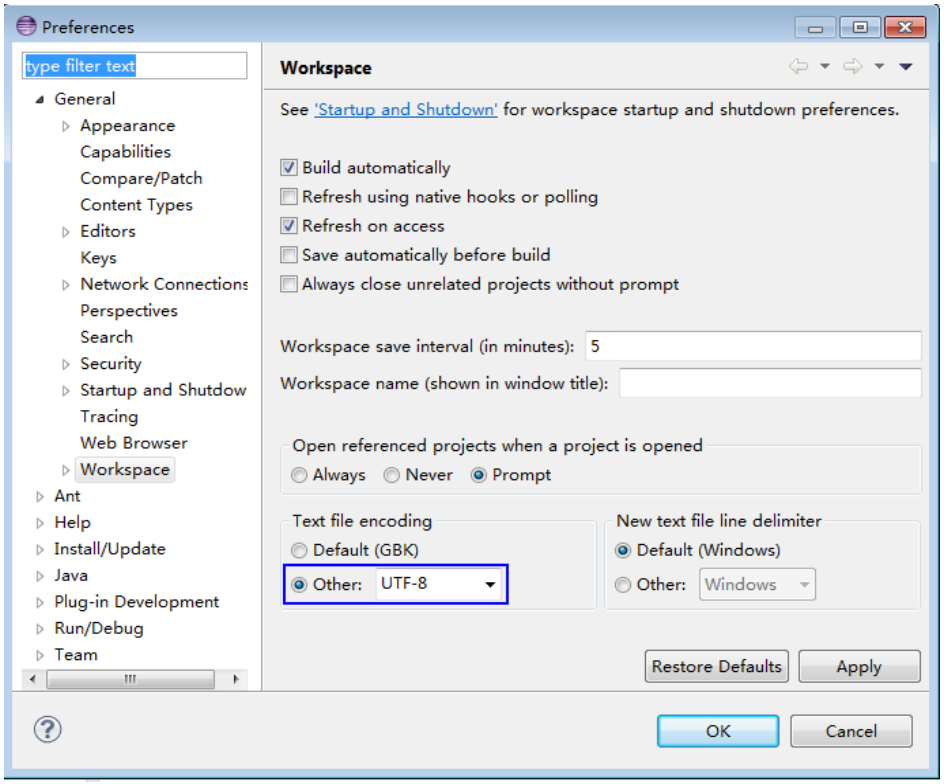
1. Start Eclipse and choose **File** > **Import**. In the **Import** dialog box, select **Existing Maven Projects**, and click **next**.
2. Click **Browse** in the **Import Maven Projects** window. The **Select Root Folder** dialog box is displayed.
3. Select the **kafka-examples** sample project folder, and click **OK**.
4. Click **Finish** in the **Import Maven Projects** window.

Step 4 Set an Eclipse text file encoding format to prevent garbled characters.

1. On the Eclipse menu bar, choose **Window** > **Preferences**.

   The **Preferences** window is displayed.
2. In the navigation tree, choose **General** > **Workspace**. In the **Text file encoding** area, select **Other** and set the value to **UTF-8**. Click **Apply** and then **OK**. **Figure 10-2** shows the settings.

Figure 10-2 Setting the Eclipse encoding format



----End

10.2.4 Preparing the Kafka Application Security Authentication

Prerequisites

Kerberos authentication has been enabled for the MRS cluster. Skip this step if Kerberos authentication is not enabled for the cluster.

Preparing the Authentication Mechanism Code

In the environment with Kerberos authentication enabled, the components must be mutually authenticated before communicating with each other, in order 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 set related environment variables accordingly. LoginUtil related interfaces are provided to complete the configuration. In the following sample code, only the account applied by a user and the keytab file name need to be configured. The keytab file of a human-machine account becomes invalid when the user password expires. Therefore, you are advised to use a machine-machine account for configuration.

Code sample:

Configure the keytab authentication file module.

```
/**
 * keytab file name of the account that a user applies for
```



```
 */
private static final String USER_KEYTAB_FILE = "keytab file name of the account that a user applies for";

/**
* Account that a user applies for
*/
private static final String USER_PRINCIPAL = "Account that a user applies for";
```

Kerberos authentication module of MRS. If the Kerberos authentication is not enabled for the service, this logic does not need to be executed.

```
public static void securityPrepare() throws IOException
{
    String filePath = System.getProperty("user.dir") + File.separator + "conf" + 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.hadoop.com");
    LoginUtil.setJaasFile(USER_PRINCIPAL, userKeyTableFile);
}
```

NOTE

If you change the Kerberos domain name of a cluster, you need to add **kerberos.domain.name** to the code and configure a correct domain name following the **hadoop.expr=toLowerCase(%{default_realm}%{KerberosServer})** rule. For example: If the domain name is changed to **HUAWEI.COM**, set this parameter to **hadoop.huawei.com**.

Obtaining the Keytab File

1. Access MRS Manager with Kerberos enabled. For details, see section "Accessing MRS Manager" in *MapReduce Service User Guide*.
2. Choose **System** > **Manage User**. Locate the specified user, click **More** > **Download authentication credential**.
3. Decompress the downloaded **.zip** file to obtain the **krb5.conf** file and the keytab file of the user.
4. Copy the **krb5.conf** file and the keytab file of the user to the **conf** directory of the sample project.

10.3 Developing a Kafka Application

10.3.1 Kafka Development Plan

Scenario Description

Kafka is a distributed message system, in which messages can be published or subscribed. A Producer is developed to send a message to a topic of a Kafka cluster every second, and a Consumer is implemented to ensure that the topic is subscribed and that messages of the topic are consumed in real time.

Development Guidelines

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.

10.3.2 Kafka Old Producer API Usage Sample

Function Description

Playing as a message producer role, Producer publicize messages on Kafka Broker.

The following code snippet belongs to the run method in the **com.huawei.bigdata.kafka.example.Old_Producer** class. It is used to send one message to a specific topic per second. (Note: The old Producer APIs support only access to topics without ACL restrictions through ports that have not enabled Kerberos authentication. For details, see **Kafka Security APIs**.)

Sample Code

Logic in the run method of old Producer APIs

```
/*
 * Start producer to send a message per second.
 */
public void run()
{
    LOG.info("Old Producer: start.");
    int messageNo = 1;

    while (true)
    {
        String messageStr = new String("Message_" + messageNo);

        // Specify the message sequence number as the key value.
        String key = String.valueOf(messageNo);
        producer.send(new KeyedMessage<String, String>(topic, key, messageStr));
        LOG.info("Producer: send " + messageStr + " to " + topic);
        messageNo++;

        // Send a message every other second.
        try
        {
            Thread.sleep(1000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

10.3.3 Kafka Old Consumer API Usage Sample

Function Description

Each Consumer instance belongs to a Consumer group, and one message is consumed by one Consumer instance in a same Consumer group. Multiple Consumer groups can consume a same message at the same time.

The following code snippet belongs to the run method in the **com.huawei.bigdata.kafka.example.Old_Consumer** class. It is used to subscribe to messages of a specific topic. (Note: The old Consumer APIs support only access to topics without ACL restrictions. For details, see **Kafka Security APIs**.)

Sample Code

Consumption logic in the run method of the old Consumer API threads

```
/**  *Run Consumer to subscribe to specified topic messages on Kafka.  */
 public void run()
 {
   LOG.info("Consumer: start.");

   Map<String, Integer> topicCountMap = new HashMap<String, Integer>();
   topicCountMap.put(topic, new Integer(1));
   Map<String, List<KafkaStream<byte[], byte[]>>> consumerMap =
consumer.createMessageStreams(topicCountMap);
   List<KafkaStream<byte[], byte[]>> streams = consumerMap.get(topic);

   LOG.info("Consumerstreams size is : " + streams.size());

   for (KafkaStream<byte[], byte[]> stream : streams)
   {
     ConsumerIterator<byte[], byte[]> it = stream.iterator();

     while (it.hasNext())
     {
       LOG.info("Consumer: receive " + new String(it.next().message()) + " from " + topic);
     }
   }

   LOG.info("Consumer End.");
 }
```

10.3.4 Kafka Producer API Usage Sample

Function Description

The following code snippet belongs to the **com.huawei.bigdata.kafka.example.Producer** class. It is used by the new Producer APIs to produce messages for the security topic.

Sample Code

Consumption logic in the run method of the Producer threads

```
   public void run()
   {
       LOG.info("New Producer: start.");
       int messageNo = 1;
       // Specify the number of messages to be sent before the thread sleeps for one second.
       int intervalMessages=10;

       while (messageNo <= messageNumToSend)
       {
           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 asynchronously.
            producer.send(record, new DemoCallBack(startTime, messageNo, messageStr));
        }
        else
        {
            try
            {
                // Send synchronously.
                producer.send(record).get();
            }
            catch (InterruptedException ie)
            {
                LOG.info("The InterruptedException occured : {}.", ie);
            }
            catch (ExecutionException ee)
            {
                LOG.info("The ExecutionException occured : {}.", ee);
            }
        }
        messageNo++;

        if (messageNo % intervalMessages == 0)
        {
            // Send the number of messages specified for intervalMessage before the thread sleeps for one
second.
            try
            {
                Thread.sleep(1000);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
            LOG.info("The Producer have send {} messages.", messageNo);
        }
    }

}
```

10.3.5 Kafka Consumer API Usage Sample

Function Description

The following code snippet belongs to the run method in the **com.huawei.bigdata.kafka.example.Consumer** class. It is used to consume topic messages that are subscribed to.

Sample Code

DoWork method logic of the consumer thread. This method is the rewrite of the run method.

```
/**
 * Message processing function for subscribing to topics
 */
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("[NewConsumerExample], Received message: (" + record.key() + ", " + record.value()
                + ") at offset " + record.offset());
        }
    }
```

10.3.6 Kafka Multi-Thread Producer API Usage Sample

Function Description

The multi-thread producer function is implemented based on the code sample described in **Kafka 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 snippets belong to the **com.huawei.bigdata.kafka.example.ProducerMultThread** class. They are used to enable multiple threads to produce data.

Sample Code

Run method logic of the producer thread class

```
    /**
     * The Producer thread executes a function to send messages periodically.
     */
    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.
            Integer key = new Integer(sendThreadId);

            long startTime = System.currentTimeMillis();

            // Construct a message record.
            ProducerRecord<Integer, String> record = new ProducerRecord<Integer, String>(topic, key,
messageStr);

            if (isAsync)
            {
                // Send asynchronously.
                producer.send(record, new DemoCallBack(startTime, key, messageStr));
            }
            else
            {
                try
                {
                    // Send synchronously.
                    producer.send(record).get();
                }
                catch (InterruptedException ie)
                {
                    LOG.info("The InterruptedException occured : {}.", ie);
                }
```



```
                catch (ExecutionException ee)
                {
                    LOG.info("The ExecutionException occured : {}.", ee);
                }
            }
            LOG.info("Producer: send " + messageStr + " to " + topic + " with key: " + key);
            messageCount++;

            // Send a message every other second.
            try
            {
                Thread.sleep(1000);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
```

Thread startup logic of the **ProducerMultThread** main class

```
/**
 * Start multiple threads for sending.
 */
public void run()
{
    // Specify whether to use the asynchronous sending mode.
    final boolean asyncEnable = false;

    // Specify the thread number, which is the unique identifier of a thread.
    for (int threadNum = 0; threadNum < PRODUCER_THREAD_COUNT; threadNum++)
    {
        ProducerThread producerThread = new ProducerThread(topic, asyncEnable, threadNum);
        producerThread.start();
    }

}
```

10.3.7 Kafka Multi-Thread Consumer API Usage Sample

Function Description

The multi-thread consumer function is implemented based on the sample codes described in **Kafka 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 snippets belong to the **com.huawei.bigdata.kafka.example.ConsumerMultThread** class. They are used to implement concurrent consumption of messages in a specified topic.

Kafka does not support seamless integration of the SpringBoot project.

Sample Code

DoWork() method logic of a single consumer thread (rewrite of the run method)

```
    /**
     * Message processing function for subscribing to topics
     */
    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(receivedThreadId+"Received message: (" + record.key() + ", " + record.value()
                    + ") at offset " + record.offset());
        }
    }
```

Thread startup logic of the **ConsumerMultThread** main class

```
public void run()
{
    LOG.info("Consumer: start.");

    for (int threadNum = 0; threadNum < CONCURRENCY_THREAD_NUM; threadNum++)
    {
        Consumer consumerThread = new Consumer(KafkaProperties.TOPIC,threadNum);
        consumerThread.start();
    }
}
```

10.3.8 Kafka SimpleConsumer API Usage Sample

Function Description

The following code snippet belongs to the **com.huawei.bigdata.kafka.example.SimpleConsumerDemo** class. It is used to enable the new SimpleConsumer APIs to subscribe a topic and consume messages. (Note: The SimpleConsumer APIs support only access to topics without ACL restrictions. For details, see **Kafka Security APIs**.)

SimpleConsumer APIs belong to low-level Consumer APIs, which are not recommended for accessing ZooKeeper metadata and managing the offset of the consumption topic queue.

Sample Code

The main method of SimpleConsumer APIs requires three parameters: Maximum consumption amount, consumption topic, and consumption topic partition.

```
public static void main(String args[])
    {
        // Maximum number of messages that can be read
        long maxReads = 0;

        try
        {
            maxReads = Long.valueOf(args[0]);
        }
        catch (Exception e)
        {
            log.error("args[0] should be a number for maxReads.\n" +
                "args[1] should be a string for topic. \n" +
                "args[2] should be a number for partition.");
            return;
        }

        if (null == args[1])
        {
            log.error("args[0] should be a number for maxReads.\n" +
                "args[1] should be a string for topic. \n" +
                "args[2] should be a number for partition.");
            return;
        }
```



```
    // Topic of messages that are consumed
    // String topic = KafkaProperties.TOPIC;
    String topic = args[1];

    // Partition of messages that are consumed
    int partition = 0;
    try
    {
        partition = Integer.parseInt(args[2]);
    }
    catch (Exception e)
    {
        log.error("args[0] should be a number for maxReads.\n" +
            "args[1] should be a string for topic. \n" +
            "args[2] should be a number for partition.");
    }

    // Broker List
    String bkList = KafkaProperties.getInstance().getValues("metadata.broker.list", "localhost:9092");

    Map<String, Integer> ipPort = getIpPortMap(bkList);

    SimpleConsumerDemo example = new SimpleConsumerDemo();
    try
    {
        example.run(maxReads, topic, partition, ipPort);
    }
    catch (Exception e)
    {
        log.info("Oops:" + e);
        e.printStackTrace();
    }
}
```

10.3.9 Kafka Configuration File

Descriptions about configuration files and key parameters in the **Conf** directory

- Producer API configuration items

Table 10-3 Configuration items in the **producer.properties** file

Parameter	Description	Remarks
security.protocol	Security protocol type	The Producer uses the security protocol of the type specified by this parameter. When Kerberos authentication is enabled, only the SASL protocol is supported, and this parameter must be set to **SASL_PLAINTEXT**. Set this parameter to **PLAINTEXT** when Kerberos authentication is disabled.

Parameter	Description	Remarks
kerberos.domain.name	Domain name	Kerberos domain name of the MRS cluster. This parameter is not required for clusters with Kerberos authentication disabled.
sasl.kerberos.service.name	Service name	The service name indicates the Kerberos username used by Kafka clusters for running. This parameter must be configured as **kafka**. You do not need to configure this parameter for clusters with Kerberos authentication disabled.

- Consumer API configuration items

Table 10-4 Configuration items in the **consumer.properties** file

Parameter	Description	Remarks
security.protocol	Security protocol type	The Consumer uses the security protocol of the type specified by this parameter. When Kerberos authentication is enabled, only the SASL protocol is supported, and this parameter must be set to **SASL_PLAINTEXT**. Set this parameter to **PLAINTEXT** when Kerberos authentication is disabled.
kerberos.domain.name	Domain name	Kerberos domain name of the MRS cluster. This parameter is not required for clusters with Kerberos authentication disabled.
group.id	Consumer group ID	-

Parameter	Description	Remarks
auto.commit.interval.ms	Indicates whether to automatically submit the offset.	Boolean value. The default value is **true**.
sasl.kerberos.service.name	Service name	The service name indicates the Kerberos username used by Kafka clusters for running. This parameter must be configured as **kafka**. You do not need to configure this parameter for clusters with Kerberos authentication disabled.

- Configuration items of the client information

Table 10-5 Configuration items in the **client.properties** file

Parameter	Description	Remarks
metadata.broker.list	Metadata Broker address list	This parameter is used to create a connection to the metadata broker. This parameter is required for APIs that are used to access metadata directly. The access port does not support Kerberos authentication. For details about the port description, see **Kafka Security APIs**.
kafka.client.zookeeper.principal	Authentication and domain name used by the Kafka cluster to access ZooKeeper	-
bootstrap.servers	Broker address list	Connections with the Broker are created based on this parameter. For details about the port configuration items, see **Kafka Security APIs**.

Parameter	Description	Remarks
zookeeper.connect	ZooKeeper address list	This parameter is used to access the ZooKeeper. The Kafka service name **kafka** must be added at the end.

- Indicates whether to enable Kerberos authentication for MRS.

Table 10-6 Configuration items in the **kafkaSecurityMode** file

Parameter	Description	Remarks
kafka.client.security.mode	Whether to enable Kerberos authentication for the MRS cluster where Kafka is located.	If Kerberos authentication is enabled, set this parameter to **yes**. Otherwise, set this parameter to **no**.

- Configuration items in the **log4j.properties** file

  Configuration file of the Log4j framework. By default, no run log of the sample project is entered.

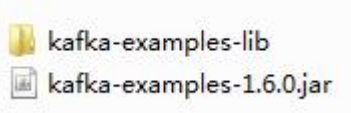
10.4 Commissioning a Kafka Application

Prerequisites

- The client can log in to the Elastic Cloud Server (ECS) of MRS. For details about how to log in to the ECS, see **Getting Started** > **Logging In to an ECS Using SSH** in the *ECS User Guide*.
- The sample project has been compiled using Maven.

Example: Packing the Maven Project to Run Examples on Linux

1. Run the **mvn package** command to generate a JAR file, for example, **kafka-examples-1.6.0.jar**, and obtain it from the **target** directory in the project directory.
2. Run the **mvn dependency:copy-dependencies -DoutputDirectory=kafka-examples-lib -DincludeScope=compile** command to export the JAR files that the Kafka sample project depends on, for example, to the **kafka-examples-lib** directory.
3. A JAR file and a **lib** directory are generated in the specified path.



4. Copy the generated dependency library folder (**kafka-examples-lib**) to any directory in a Linux environment of the MRS service, for example: **/opt/example**, and then copy the generated JAR file to the **/opt/example/kafka-examples-lib** directory.
5. Copy the **conf** directory of the sample project to the directory where the dependent library folder is located, that is, **/opt/example**, and create the **logs** directory for storing the run logs of the JAR file.
6. Switch to the **root** user and change the directories **conf**, **kafka-examples-lib**, and **logs** to **omm**, which is owned by the **omm:wheel** user group. Run the following command to switch to the user group:

   sudo su - root

   chown -R omm:wheel /opt/example/*

   ```
   drwxr-xr-x. 2 omm wheel 4096 Jan 16 16:48 conf
   drwxr-xr-x. 2 omm wheel 4096 Jan 16 16:50 kafka-examples-lib
   drwxr-xr-x. 2 omm wheel 4096 Jan 16 16:50 logs
   ```

7. Switch to user **omm** and go to the **/opt/example** directory. Ensure that all files in the **conf** directory and the dependent library file directory can be read by the current user. Ensure that the JDK has been installed and Java environment variables have been set. Then run the **java -cp .:/opt/example/conf:/opt/example/kafka-examples-lib/* com.huawei.bigdata.kafka.example.Producer** command to run the sample project.

   su - omm

   chmod 750 /opt/example

   cd /opt/example

   java -cp .:/opt/example/conf:/opt/example/kafka-examples-lib/* com.huawei.bigdata.kafka.example.Producer

Observing the Running Result

The running result of the JAR file of the sample program project can be viewed in the **client.log** file in the **logs** directory. The **log4j.properties** file does not output the running status by default. To view the running information of the program, you need to configure the **log4j.properties** file as follows:

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

kafka.logs.dir=logs

log4j.rootLogger=INFO, stdout, kafkaAppender

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
```



```
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.logger.kafka=INFO, kafkaAppender

log4j.appender.kafkaAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appender.kafkaAppender.DatePattern='.'yyyy-MM-dd-HH
log4j.appender.kafkaAppender.File=${kafka.logs.dir}/client.log
log4j.appender.kafkaAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.kafkaAppender.layout.ConversionPattern=[%d] %p %m (%c)%n

# Turn on all our debugging info
#log4j.logger.kafka.producer.async.DefaultEventHandler=DEBUG, kafkaAppender
#log4j.logger.kafka.client.ClientUtils=DEBUG, kafkaAppender
#log4j.logger.kafka.perf=DEBUG, kafkaAppender
#log4j.logger.kafka.perf.ProducerPerformance$ProducerThread=DEBUG, kafkaAppender
#log4j.logger.org.I0Itec.zkclient.ZkClient=DEBUG
```

Add **kafkaAppender** to **rootLogger** and change the log level to the level that you want to observe.

10.5 FAQs About Kafka Application Development

10.5.1 Kafka APIs

10.5.1.1 Kafka Shell Commands

Prerequisites

The Linux client of Kafka has been installed. For details, see **Installing a Client**.

Shell Command Guide

Methods of running shell commands:

Step 1 Go to any directory of the Kafka client.

Step 2 Run the following command to initialize environment variables:

source /opt/client/bigdata_env

Step 3 If Kerberos authentication has been enabled for the current cluster, run the following command to authenticate the current user (the user must be added to the **kafkaadmin** user group and have the Kafka administrator permission): If the Kerberos authentication is disabled for the current cluster, skip this step.

kinit *MRS cluster user*

For example, **kinit admin**

Step 4 Go to *Client installation directory*/**Kafka/kafka/bin** and run the Kafka shell command.

----End

The common commands are as follows:

- Query the list of topics in the current cluster.

  sh kafka-topics.sh --list --zookeeper <*ZooKeeper cluster IP address***:2181/kafka>**
- Query the details of a topic.

  sh kafka-topics.sh --describe --zookeeper <*ZooKeeper cluster IP address***:2181/kafka> --topic <***Topic name***>**
- Delete a topic (performed by an administrator).

  sh kafka-topics.sh --delete --zookeeper <*ZooKeeper cluster IP address***:2181/kafka> --topic <***Topic name***>**
- Create a topic (performed by an administrator).

  sh kafka-topics.sh --create --zookeeper <*ZooKeeper cluster IP address***:2181/kafka> --partitions 6 --replication-factor 2 --topic <***Topic name***>**
- Produce data using the old Producer APIs.

  sh kafka-console-producer.sh --broker-list <*Kafka cluster IP address***:9092> --topic <***Topic name***> --old-producer -sync**
- Consume data using the old Consumer APIs.

  sh kafka-console-consumer.sh --zookeeper <*ZooKeeper cluster IP address***:2181/kafka> --topic <***Topic name***> --from-beginning**
- Enable the Producer APIs to produce messages (this operation requires the producer permission of the desired topic).

  sh kafka-console-producer.sh --broker-list <*Kafka cluster IP address***:21007> --topic <***Topic name***> --producer.config config/producer.properties**
- Enable the Consumer APIs to consume data (this operation requires the consumer permission of the desired topic).

  sh kafka-console-consumer.sh --topic <*Topic name***> --bootstrap-server <***Kafka cluster IP address***:21007> --new-consumer --consumer.config config/consumer.properties**

NOTE

- The default port of the Kafka cluster with Kerberos authentication disabled is **9092**, and the default port of the Kafka cluster with Kerberos authentication enabled is **21007**.
- Log in to **MRS Manager**, choose **Services** > **ZooKeeper** > **Instance**, and get the **OM IP Address** of the **quorumpeer** instance.
- Log in to **MRS Manager**, choose **Services** > **Kafka** > **Instance**, and obtain the **OM IP Address** of the Kafka broker instance.

10.5.1.2 Kafka Java APIs

The Kafka APIs are the same as those in the open source community. For details, see **http://kafka.apache.org/documentation.html#api**.

10.5.1.3 Kafka Security APIs

1. If Kerberos authentication is enabled, the port for accessing the Kafka cluster is **21007** by default. If Kerberos is not enable, the port for accessing the Kafka cluster is **21005** by default.
2. The old APIs can access only the **9092** port. The new APIs can access the **9092** port with Kerberos authentication disabled and the **21007** port with Kerberos authentication enabled.

10.5.2 What Do I Do if Metadata Fails to Be Obtained by Running the Producer.java Sample?

Troubleshooting Procedure

1. Find **bootstrap.servers** in **client.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** (port with Kerberos authentication enabled), change it to **9092** (port with Kerberos authentication disabled).
2. Check whether network connections are correct to ensure that the current device can access the Kafka cluster normally.

11 MapReduce Development Guide

11.1 MapReduce Application Development Overview

11.1.1 Introduction to MapReduce Application Development

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 concurrently process TB-level data sets in fault tolerance mode.

A MapReduce job (application or job) splits an input data set into several independent data blocks, which 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 HDFS. The framework schedules and monitors tasks as well as re-executes failed tasks.

MapReduce has the following characteristics:

- Large-scale parallel computing
- Large data set processing
- High fault tolerance and reliability
- Proper resource scheduling

11.1.2 Common Concepts of MapReduce Application Development

- **Hadoop shell commands**

  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 API. In addition, to facilitate sorting operations, key classes must support the WritableComparable API.

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)

- **Core of Jobs**

  Typically, 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 Web UI**

  MapReduce web UIs allow users to monitor running or historical MapReduce jobs, view logs, and implement fine-grained job development, configuration, and optimization.

- **Archiving**

  Archiving ensures that all mapped key-value pairs share one key group.

- **Shuffle**

  Shuffle is a process of outputting data from a Map task to a Reduce task.

- **Mapping**

  Mapping is used to map a group of key-value pairs into a new group of key-value pairs.

11.1.3 MapReduce Application Development Process

Figure 11-1 and **Table 11-1** describe the phases in the development process.

Figure 11-1 MapReduce application development process

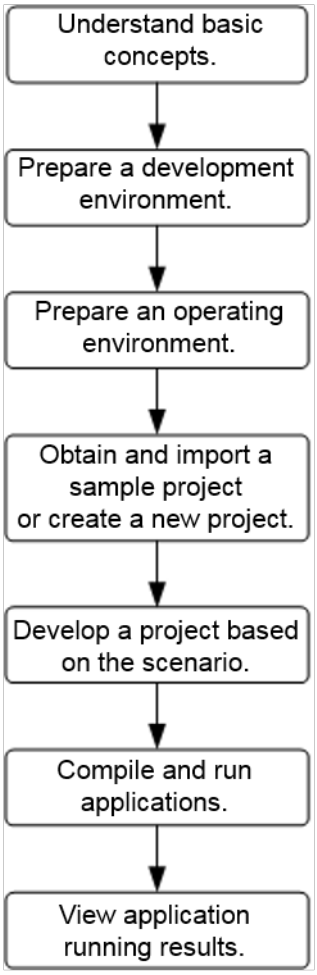


Table 11-1 Description of the MapReduce development process

Phase	Description	Reference
Understand basic concepts.	Before application development, learn basic concepts of MapReduce.	**Common Concepts of MapReduce Application Development**
Prepare a development environment.	Use the Eclipse tool to configure the development environment according to the guide.	**Preparing the Eclipse and JDK**
Prepare an operating environment.	The MapReduce operating environment is a MapReduce client. Install and configure the client according to the guide.	**Preparing a MapReduce Application Running Environment**

Phase	Description	Reference
Obtain and import a sample project or create a new project.	MapReduce provides sample projects for different scenarios. You can import a sample project to learn the application. You can also create a MapReduce project according to the guide.	**Importing and Configuring MapReduce Sample Projects**
Develop a project based on the scenario.	MRS provides you with a sample project to help you quickly learn APIs of all MapReduce components.	• **MapReduce Development Plan** • **Development Plan of Accessing a Multi-Component Program**
Compile and run applications.	You can compile the developed application and submit it for running.	**Compiling and Running Applications**
View application running results.	Application running results are exported to a path you specify. You can also view the application running status on the UI.	**Viewing the MapReduce Application Commissioning Result**

11.2 Preparing a MapReduce Application Development Environment

11.2.1 MapReduce Application Development Environment

Table 11-2 describes the environment required for application development. You also need to prepare a Linux environment for verifying whether the application is running properly.

Table 11-2 Development environment

Item	Description
Eclipse installation	Basic configurations of the development environment. Eclipse 4.2 is required.
JDK installation	JDK 1.8 is required.

11.2.2 Preparing a MapReduce Application Development User

The development user is used to run the sample project. The user must have component permissions to run the sample project.

Prerequisites

Kerberos authentication has been enabled for the MRS cluster. Skip this step if Kerberos authentication is not enabled for the cluster.

Procedure

Step 1 Log in to **MRS Manager** and choose **System** > **Manage Role** > **Create Role**.

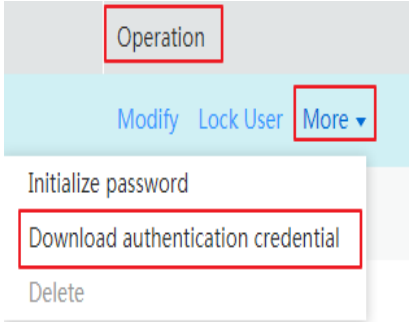
1. Enter a role name, for example, *mrrole*.
2. Edit the role. In **Permission**, choose **Yarn > Scheduler Queue > root**, select **Submit** and **Admin**.
3. In **Permission**, choose **HBase** > **HBase Scope**, select **Create**, **Read**, **Write**, and **Execute** for **global**.
4. In **Permission**, choose **HDFS** > **File System** > **hdfs://hacluster/** and select **Read**, **Write**, and **Execute**.
5. In **Permission**, choose **Hive** > **Hive Read Write Privileges** and select **Create**, **Select**, **Delete**, and **Insert** for **default**.
6. Click **OK**.

Step 2 Choose **System** > **Manage User** > **Create User** to create a user for the sample project.

Step 3 Enter a username, for example, *test*. Set **User Type** to **Machine-machine**, and select **supergroup** in **User Group**. Set **Primary Group** to **supergroup**, select **mrrole** in **Assign Rights by Role**, and click **OK**.

Step 4 On MRS Manager, choose **System > User Management**. On the displayed page, select **test** from the **Username** drop-down list. In the **Operation** column on the right, choose **More > Download authentication credential**. Save the downloaded package and decompress it to obtain the **user.keytab** and **krb5.conf** files for security authentication in the sample project, as shown in **Configuring Security Authentication for MapReduce Applications**.

Figure 11-2 Downloading the authentication credential



----End

11.2.3 Preparing the Eclipse and JDK

Install Eclipse and JDK in the Windows development environment.

Step 1 Install the Eclipse program in the development environment. The Eclipse version must be 4.2 or later.

Step 2 Install the JDK program in the development environment. The JDK version must be 1.8.

NOTE

- If you use IBM JDK, ensure that the JDK configured in Eclipse is IBM JDK.
- If you use Oracle JDK, ensure that the JDK configured in Eclipse is Oracle JDK.
- Do not use the same workspace and the sample project in the same path for different Eclipse programs.

----End

11.2.4 Preparing a MapReduce Application Running Environment

The operating environment of MapReduce can be deployed in Linux. Perform the following operations to prepare the operating environment.

Procedure

Step 1 Ensure that the YARN and MapReduce components on the server have been installed and are running properly.

Step 2 The JDK 1.7 or 1.8 has been installed on the client.

Step 3 Ensure that the time difference between the client and the Hadoop cluster is less than 5 minutes.

You can query the MRS cluster time by logging in to the active management node that corresponds to the cluster management IP address to run the **date** command.

Step 4 Download the MapReduce client program to the local computer.

1. Log in to **MRS Manager**.

   Enter the address in the address box of your browser. The address format is **https://***Floating IP address of WebService of MRS Manager***:8080/web**. For example, enter **https://10.10.10.172:8080/web**.

2. Choose **Service** > **Download Client** to download the client program to the local PC.

Step 5 Decompress the **MRS_Services_Client.tar** client program package. Because the installation package is in **.tar** format, run the following commands to decompress the package twice:

tar -xvf /opt/MRS_Services_Client.tar

tar -xvf /opt/MRS_Service_ClientConfig.tar

Step 6 Set environment variables for the operating environment. Assume that the installation package is decompressed in **MRS_Services_ClientConfig/**.

Go to the decompressed folder and run the following command to install the client:

sh *install.sh* {client_install_home}

Step 7 Go the client installation directory and run the following command to initialize the environment variables:

source *bigdata_env*

Step 8 Copy the **user.keytab** and **krb5.conf** files downloaded in **Preparing a MapReduce Application Development User** to the Linux environment, for example, **/opt/conf**. For details, see **Compiling and Running Applications**.

NOTE

During the secondary development, the username used by PRINCIPAL must contain the domain name. For example, if you create a user named **test** and the domain name is **HADOOP.COM**, the username of PRINCIPAL is **test@HADOOP.COM**. The following is sample code.

```
conf.set(PRINCIPAL, "test@HADOOP.COM");
```

Step 9 Run the **kinit -kt /opt/conf/user.keytab test** command.

NOTE

The path of the **user.keytab** file is the path for storing the configuration file on the Linux host. The **test** username can be changed to the name of the new user created in **Preparing a MapReduce Application Development User**.

----End

11.2.5 Importing and Configuring MapReduce Sample Projects

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

The following procedure describes how to import MapReduce sample code.

Procedure

Step 1 Download the sample project to the local computer by referring to **Obtaining the MRS Application Development Sample Project**.

Step 2 Import the sample project to the Eclipse development environment.

1. Start Eclipse and choose **File** > **Import**. In the **Import** dialog box, select **Existing Maven Projects** and click **next**.
2. Click **Browse** in the **Import Maven Projects** window. The **Select Root Folder** dialog box is displayed.
3. Select the **mapreduce-examples** sample project folder, and click **OK**.
4. Click **Finish** in the **Import Maven Projects** window.

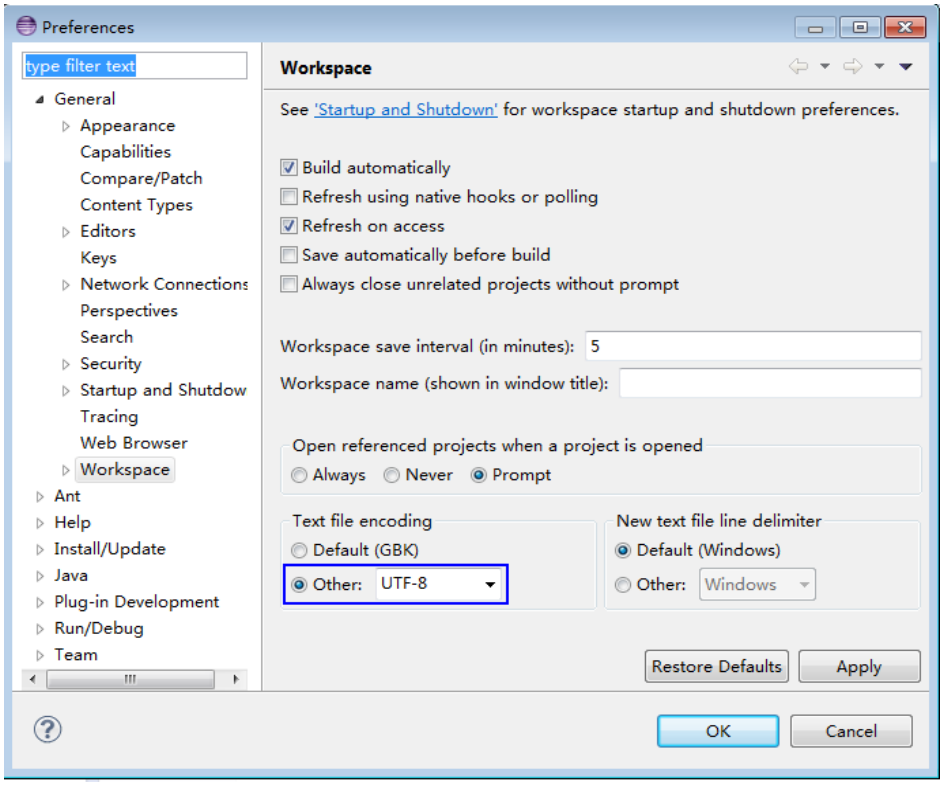
Step 3 Set an Eclipse text file encoding format to prevent garbled characters.

1. On the Eclipse menu bar, choose **Window** > **Preferences**.

   The **Preferences** window is displayed.

2. In the navigation tree, choose **General** > **Workspace**. In the **Text file encoding** area, select **Other** and set the value to **UTF-8**. Click **Apply** and then **OK**. Figure 11-3 shows the settings.

Figure 11-3 Setting the Eclipse encoding format



----End

11.2.6 Configuring Security Authentication for MapReduce Applications

Scenario Description

In cluster environment with Kerberos authentication enabled, the components must be mutually authenticated before communicating with each other to ensure communication security.

When submitting MapReduce applications, users need to communicate with Yarn and HDFS. Code for security authentication needs to be written into the MapReduce application to be submitted to ensure that MapReduce can work properly.

Two security authentication modes are available.

- CLI authentication

  Before submitting and running the MapReduce application, run the following command on the MapReduce client to obtain authentication:

  kinit *Component service user*

- Code authentication

  Obtain the principal and keytab files of the client for authentication.

Security Authentication Code

Currently, the LoginUtil class is invoked for security authentication in a unified manner.

In the MapReduce sample project code, **test@HADOOP.COM**, **user.keytab**, and **krb5.conf** are examples. In actual operations, contact the administrator to obtain the **keytab** and **krb5.conf** files corresponding to the account and the permission. Save the **keytab** and **krb5.conf** files to the **conf** directory of the sample code. The code for security login is as follows:

NOTE

Modify the authentication information based on the site requirements.

```
public static final String PRINCIPAL= "test@HADOOP.COM";
public static final String KEYTAB =
FemaleInfoCollector.class.getClassLoader().getResource("user.keytab").getPath();
public static final String KRB =
FemaleInfoCollector.class.getClassLoader().getResource("krb5.conf").getPath();
// Check whether the security mode is used.
   if("kerberos".equalsIgnoreCase(conf.get("hadoop.security.authentication"))){
      // Security login
      System.setProperty("java.security.krb5.conf", KRB);
      LoginUtil.login(PRINCIPAL, KEYTAB, KRB, conf);
   }
```

11.3 Developing a MapReduce Application

11.3.1 MapReduce Development Plan

Scenario Description

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 on the 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 columns are separated by comma (,).

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 Planning

Save the original log files in the HDFS.

1. Create two text files **input_data1.txt** and **input_data2.txt** on a local computer, and copy **log1.txt** to **input_data1.txt** and **log2.txt** to **input_data2.txt**.
2. Create the **/tmp/input** folder in the HDFS, and run the following commands to upload **input_data1.txt** and **input_data2.txt** to the **/tmp/input** directory:
   a. On the Linux HDFS client, run the ***hdfs dfs -mkdir /tmp/input***.
   b. On the Linux HDFS client, run the **hdfs dfs -put** *local_filepath /tmp/input* command.

Development Guidelines

Collect statistics on female netizens who dwell on online shopping for more than 2 hours on the weekend.

To achieve the objective, the process is as follows:

- Read original file data.
- Filter data information of the time that female netizens spend online.
- Summarize the total time that each female netizen spends online.
- Filter the information of female netizens who spend more than 2 hours online.

Function Description

Collect statistics on female netizens who dwell on online shopping for more than 2 hours on the weekend.

The operation is performed in three steps.

- Filter the dwell duration of female netizens in original files using the CollectionMapper class inherited from the Mapper abstract class.
- Summarize the dwell duration 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.

- Use the main method to create a MapReduce job and then submit the MapReduce job to the Hadoop cluster.

Sample Code

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;
// Gender screening
String sexFilter;

// Name information
private Text nameInfo = new Text();

// Output key-values must be serialized.
private IntWritable timeInfo = new IntWritable(1);

 /**
 * Distributed computing
 *
 * @param key Object: Location offset of the original file.
 * @param value Text: A line of character data in the original 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 line of character string data 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);
      }
   }

   /**
  * Invoke map to perform some initial operations.
  *
  * @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>
{

    // Statistics results
        private IntWritable result = new IntWritable();

        // Total time threshold
        private int timeThreshold;

     /**
      * @param key Text: Key after Mapper
      * @param values Iterable: all statistical results of 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 outputted if the time is less than the threshold.
            if (sum < timeThreshold)
       {
                return;
            }
            result.set(sum);

            // In the reduce output, key indicates netizen information, and value indicates the total dwell
duration of the netizen.
            context.write(key, result);
        }

        /**
         * The setup() method is called only once before the map() method of a map task or the reduce()
method of a reduce task is called.
     *
     * @param context Context
     * @throws IOException , InterruptedException
     */
        public void setup(Context context) throws IOException,
                InterruptedException
    {

            // Obtain configuration information using Context.
            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);
    }

    // Check whether the security mode is used.
    if("kerberos".equalsIgnoreCase(conf.get("hadoop.security.authentication"))){
       //security mode
       System.setProperty("java.security.krb5.conf", KRB);
       LoginUtil.login(PRINCIPAL, KEYTAB, KRB, conf);
    }

    // Initialize the job object.
     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);
}
```



```
}
```

11.3.2 Development Plan of Accessing a Multi-Component Program

Scenario Description

The following example illustrates how to compile MapReduce jobs to visit multiple service components in HDFS, HBase, and Hive, helping users to understand key actions such as authentication and configuration loading.

The logic process of the example is as follows:

Use an HDFS text file as input data.

log1.txt: Input file

```
YuanJing,male,10
GuoYijun,male,5
```

Map phase

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.

Reduce phase

1. Obtain the last piece of data from the Map output.
2. Export 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 the **/tmp/examples/multi-components/mapreduce/input/** folder in HDFS, and upload **data.txt** to it:
      i. On the Linux-based HDFS client, run the **hdfs dfs -mkdir -p** /tmp/examples/multi-components/mapreduce/input/ command.
      ii. On the Linux-based HDFS client, run the **hdfs dfs -put** data.txt /tmp/examples/multi-components/mapreduce/input/ command.
2. Create an HBase table and insert data.
   a. Run the **hbase shell** command on the Linux-based HBase client.
   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.

3. Create a Hive table and insert data.

a. Run the **beeline** command on the 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 fields **name**, **gender**, and **stayTime**.

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 the **!q** command to exit.

4. The HDFS data directory will be cleared when data is loaded to Hive. Therefore, you need to perform step 1 again.

Function Description

The example is divided into three parts:

- Collect the name information from HDFS original 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 it to HBase and HDFS, using the MultiComponentReducer class inherited from the Reducer abstract class.
- Use the **main** method to create a MapReduce job and then submit the MapReduce job to the Hadoop cluster.

Sample Code

The following code snippets are used as an example. For complete codes, see the **com.huawei.bigdata.mapreduce.examples.MultiComponentExample** class.

Example 1: The MultiComponentMapper class defines the **map** method of 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 {

     String name = "";
     String line = value.toString();

     // Load the configuration file.
       conf = context.getConfiguration();

       setJaasInfo("krb5.conf", "jaas.conf");
       LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, "test", KEYTAB);
       LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY,
ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);

       // Prepare for a Hive query.
       // Load parameters.
```



```
Properties clientInfo = null;
InputStream fileInputStream = null;
try {
    clientInfo = new Properties();
    File propertiesFile = new File(hiveClientProperties);
    fileInputStream = new FileInputStream(propertiesFile);
    clientInfo.load(fileInputStream);
} catch (Exception e) {
} finally {
    if (fileInputStream != null) {
        fileInputStream.close();
    }
}

String zkQuorum = clientInfo.getProperty("zk.quorum");
String zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
String serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");

// Create Hive authentication information.
// 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 = ? 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();

try {
    Class.forName(driver);
    hiveConn = DriverManager.getConnection(url, "", "");
    statement = hiveConn.prepareStatement(sql);
} catch (Exception e) {
    LOG.error("Init jdbc driver failed.", e);
}

// Create an HBase connection.
try {
    // Create a HBase connection
    hbaseConn = ConnectionFactory.createConnection(conf);
    // get table
    table = hbaseConn.getTable(TableName.valueOf(HBASE_TABLE_NAME));
} catch (IOException e) {
    LOG.error("Exception occur when connect to HBase", e);
    throw e;
}

if (line.contains("male")) {
  name = line.substring(0, line.indexOf(","));
}
// 1. Read HBase data.
String hbaseData = readHBase();

// 2. Read Hive data.
String hiveData = readHive(name);

// Map outputs a key-value pair, which is a character string combining HBase and Hive data.
```



```
    context.write(new Text(name), new Text("hbase:" + hbaseData + ", hive:" + hiveData));
  }
```

Example 2: Use the readHBase method to read HBase data.

```
  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 an HBase Get request instance.
      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 (hbaseConn != null) {
        try {
          hbaseConn.close();
        } catch (Exception e1) {
          LOG.error("Failed to close the connection ", e1);
        }
      }
    }

    return "";
  }
```

Example 3: Use the readHive method to read Hive data.

```
  private int readHive(String name) {

    ResultSet resultSet = null;
    try {
      statement.setString(1, name);
      resultSet = statement.executeQuery();

      if (resultSet.next()) {
        return resultSet.getInt("stayTime");
      }
    } 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 != hiveConn) {
        try {
          hiveConn.close();
```



```
        } catch (SQLException e) {
          // handle exception
        }
      }
    }

    return 0;
  }
```

Example 4: The MultiComponentReducer class defines the reduce method of the Reducer abstract class.

```
  public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {

    Text finalValue = new Text("");

    setJaasInfo("krb5.conf", "jaas.conf");
    LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, "test", KEYTAB);
    LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY,
ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);

    conf = context.getConfiguration();
    try {
        // Create an HBase connection.
        conn = ConnectionFactory.createConnection(conf);
        // Obtain a table.
        table = conn.getTable(TableName.valueOf(HBASE_TABLE_NAME));
    } catch (IOException e) {
        LOG.error("Exception occur when connect to HBase", e);
        throw e;
    }

    for (Text value : values) {
      finalValue = value;
    }

    // Export the result to HBase.
    writeHBase(key.toString(), finalValue.toString());

    // Save the result to HDFS.
    context.write(key, finalValue);
  }
```

Example 5: Use the writeHBase method to output data to HBase.

```
  private void writeHBase(String rowKey, String data) {

    try {
      // Create an HBase Put request instance.
      List<Put> list = new ArrayList<Put>();
      byte[] row = Bytes.toBytes("1");
      Put put = new Put(row);
      byte[] family = Bytes.toBytes("cf");
      byte[] qualifier = Bytes.toBytes("value");
      byte[] value = Bytes.toBytes(data);
      put.addColumn(family, qualifier, value);
      list.add(put);
      // Execute the Put request.
      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 6: Use the main () method to create a job, configure dependencies and authentication information, and submit the job to the Hadoop cluster.

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

    // Clear required directories.
    MultiComponentExample.cleanupBeforeRun();

    // Query the Hive dependency JAR file.
    Class hiveDriverClass = Class.forName("org.apache.hive.jdbc.HiveDriver");
    Class thriftClass = Class.forName("org.apache.thrift.TException");
    Class thriftCLIClass = Class.forName("org.apache.hive.service.cli.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.thrift.HadoopThriftAuthBridge23");

    // Add a Hive running dependency to the job.
    JarFinderUtil
        .addDependencyJars(config, hiveDriverClass, thriftCLIClass, thriftClass, hiveConfClass, hiveTransClass,
            hiveMetaClass, hiveShimClass);

    // Log in to a cluster with Kerberos authentication enabled.
    if("kerberos".equalsIgnoreCase(config.get("hadoop.security.authentication"))){
    //security mode
        LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, PRINCIPAL, KEYTAB);
        LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY,
ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);
     System.setProperty("java.security.krb5.conf", KRB);
  LoginUtil.login(PRINCIPAL, KEYTAB, KRB, config);
  }
    // Add a Hive configuration file.
    config.addResource("hive-site.xml");
    // Add an HBase configuration file.
    Configuration conf = HBaseConfiguration.create(config);

    // Instantiate the job.
    Job job = Job.getInstance(conf);
    job.setJarByClass(MultiComponentExample.class);

    // Set the mapper and reducer classes.
    job.setMapperClass(MultiComponentMapper.class);
    job.setReducerClass(MultiComponentReducer.class);

    // Set the input and output paths of the job.
    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 key-value type.
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    // HBase provides a tool class to add the HBase running dependency to the job.
    TableMapReduceUtil.addDependencyJars(job);

    // This operation must be performed in security mode.
    // HBase adds authentication information to the job. The map or reduce task uses the authentication
information.
    TableMapReduceUtil.initCredentials(job);

    // Create Hive authentication information.
    Properties clientInfo = null;
    InputStream fileInputStream = null;
    try {
        clientInfo = new Properties();
        File propertiesFile = new File(hiveClientProperties);
```



```
            fileInputStream = new FileInputStream(propertiesFile);
            clientInfo.load(fileInputStream);
        } catch (Exception e) {
        } finally {
            if (fileInputStream != null) {
                fileInputStream.close();
            }
        }
        String zkQuorum = clientInfo.getProperty("zk.quorum");// List of ZooKeeper node IP addresses and ports
        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");
        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 authentication information to a job.
        job.getCredentials().addToken(new Text("hive.server2.delegation.token"), hive2Token);
        job.getCredentials().addToken(new Text(HiveAuthFactory.HS2_CLIENT_TOKEN), hive2Token);

        // Submit the job.
        System.exit(job.waitForCompletion(true) ? 0 : 1);

    }
```

NOTE

Replace all zkQuorum objects in the examples with the actual ZooKeeper cluster node information.

11.4 Commissioning a MapReduce Application

11.4.1 Compiling and Running Applications

Run an application on Linux after application code development is complete.

NOTE

MapReduce applications can run only on Linux, but not on Windows.

Procedure

Step 1 Generate an executable MapReduce application package.

Run the **mvn package** command to generate a JAR file, for example, **mapreduce-examples-1.0.jar**, and obtain it from the **target** directory in the project directory.

Step 2 Upload the generated **mapreduce-examples-1.0.jar** application package to the Linux client, for example, **/opt**.

Step 3 If Kerberos authentication is enabled in the cluster, create a folder (for example, **/opt/conf**) in the Linux environment to save the **user.keytab** and **krb5.conf** files obtained by referring to **Preparing a MapReduce Application Development User**. Obtain the **core-site.xml** and **hdfs-site.xml** files from the client directory in the Linux environment and save them to the preceding folder.

Step 4 If the sample application specifies OBS as the target file system (for example, **obs://<BucketName>/input/**) for input and output, you need to configure parameters as follows:

Add AK configuration item **fs.obs.access.key** and SK configuration item **fs.obs.secret.key** to **$YARN_CONF_DIR/core-site.xml**. You can obtain the AK and SK by logging in to the OBS console and go to the **My Credentials** page.

```
<property>
<name>fs.obs.access.key</name>
<value>xxxxxxxxxxxxxxx</value>
</property>
<property>
<name>fs.obs.secret.key</name>
<value>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</value>
</property>
```

Step 5 Execute the sample project on Linux.

- For the MapReduce statistics sample project, run the following command to configure parameters and submit jobs.

  a. If Kerberos authentication is enabled in the cluster, add **classpath** required for running the sample project in the Linux environment.

     export YARN_USER_CLASSPATH=/opt/conf/

  b. Run the following command:

     ```
     cd /opt
     yarn jar mapreduce-examples-1.0.jar
     com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector <inputPath>
     <outputPath>
     ```

This command is used to set parameters and submit jobs. *<inputPath>* indicates the input path in HDFS and *<outputPath>* indicates the output path in HDFS.

NOTE

- Before running the **yarn jar mapreduce-examples-1.0.jar com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector** *<inputPath>* *<outputPath>* command, upload the **log1.txt** and **log2.txt** files to the *<inputPath>* directory of HDFS.
- Before running the **yarn jar mapreduce-examples-1.0.jar com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector** *<inputPath>* *<outputPath>* command, ensure that the *<outputPath>* directory does not exist. Otherwise, an error will be reported.
- **mapreduce-examples-1.0.jar** is applicable to MRS 1.*x*. For MRS 2.*x*, use **mapreduce-examples-2.0.jar**.
- Do not restart the HDFS service during the running of MapReduce jobs. Otherwise, the jobs may fail.
- Before running the sample project, you need to modify the authentication information based on site environments.
- For a security cluster with Kerberos authentication enabled, modify **principal** in the code based on site environments, for example, **test@FAA12CC3_0996_432F_9D6F_E18F6F9D7F43.COM**.

- For the sample application about multi-component access from MapReduce, perform the following steps.

  a. Obtain the **hbase-site.xml**, **hiveclient.properties**, **hive-site.xml**, and **mapred-site.xml** files. If the cluster is in security mode, you need to additionally obtain the **user.keytab** and **krb5.conf** files and create a folder in the Linux environment to save the configuration file, for example, **/opt/conf**.

  NOTE

  Contact the administrator to obtain the **user.keytab** and **krb5.conf** files of the corresponding account. Additionally, you can obtain the **hbase-site.xml** file from the HBase client, for example, **/opt/client/HBase/hbase/conf**, the **hiveclient.properties** and **hive-site.xml** files from the Hive client, for example, **/opt/client/Hive/config**, and the **mapred-site.xml** file from the Yarn client, for example, **/opt/client/Yarn/config**.

  b. For a cluster in security mode, create the **jaas_mr.conf** file in the new folder. The file content is as follows:

```
Client {
com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true
keyTab="user.keytab"
principal="test@FAA12CC3_0996_432F_9D6F_E18F6F9D7F43.COM"
useTicketCache=false
storeKey=true
debug=true;
};
```

  NOTE

  - In the preceding file content, **test@HADOOP.COM** is an example. Change it based on the site requirements.
  - Modify **principal** in the **jaas_mr.conf** file and code based on the site requirements, for example, **test@FAA12CC3_0996_432F_9D6F_E18F6F9D7F43.COM**.
  - Skip this step for a cluster with Kerberos authentication disabled.

c. In the Linux environment, add the **classpath** required for running the sample project. For example, if the client installation path is **/opt/conf**, run the following command:

export YARN_USER_CLASSPATH=/opt/conf/:/opt/client/HBase/hbase/lib/*:/opt/client/Hive/Beeline/lib/*

NOTE

- For an MRS 1.9.*x* cluster, you need to run the **mv /opt/client/Hive/Beeline/lib/derby-10.10.2.0.jar derby-10.10.2.0.jar.bak** command before or after running the preceding command.
- The JAR package used in the command must be changed based on the actual version in the corresponding path in the cluster.

d. Submit the MapReduce job and run the following command to run the sample project. Before running the sample project, you need to modify the authentication information based on site environments.

yarn jar mapreduce-examples-1.0.jar com.huawei.bigdata.mapreduce.examples.MultiComponentExample

----End

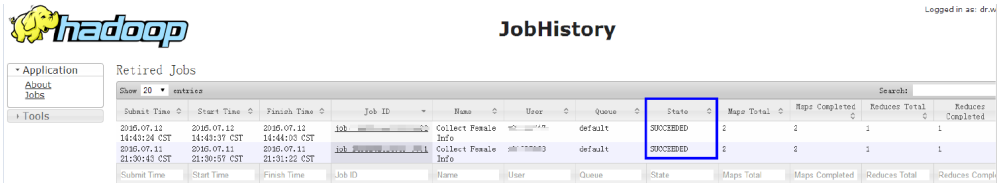
11.4.2 Viewing the MapReduce Application Commissioning Result

After a MapReduce application is run, you can view the running result by using WebUI or MapReduce logs.

- **Viewing job execution status by using the MapReduce Web UI**

  Log in to MRS Manager, choose **Service** > **MapReduce** > **JobHistoryServer**, and view the job execution status on the web UI.

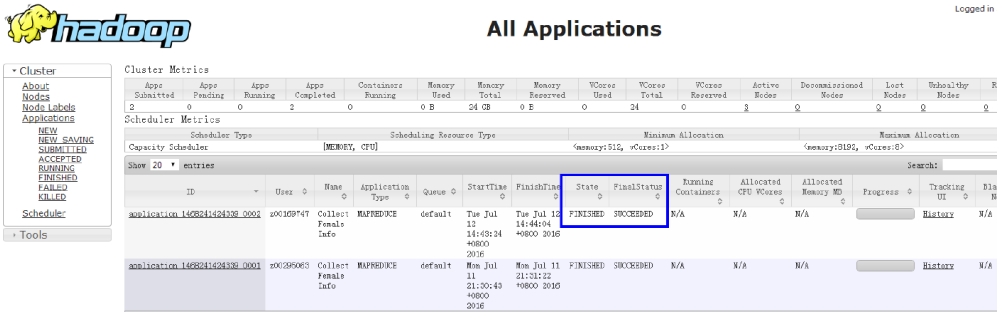
  Figure 11-4 JobHistory web UI



- **Viewing job execution status by using Yarn web UI**

  Log in to MRS Manager, choose **Service** > **Yarn** > **ResourceManager(Active)**, and view the job execution status on the web UI.

  Figure 11-5 ResourceManager web UI



- **Viewing the running result of a MapReduce application**
  - After you run the ***yarn jar mapreduce-example.jar*** command in the Linux environment, view application running status in the command output. Example:

```
yarn jar mapreduce-example.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://10-120-180-170:26000/
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
```

  - After you run the ***yarn application -status*** *<ApplicationId>* command in the Linux environment, view the application running status in the command output. 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://10-120-180-170:26012/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>
```

- **Viewing MapReduce logs to learn application running status**

  View MapReduce logs to learn application running status, and adjust applications based on log information.

11.5 FAQs About MapReduce Application Development

11.5.1 MapReduce APIs

11.5.1.1 MapReduce Java APIs

Common MapReduce APIs

Common classes in MapReduce are as follows:

- **org.apache.hadoop.mapreduce.Job**: API for users to submit MapReduce jobs. It is used to set job parameters, submit jobs, control job execution, and query job status.
- **org.apache.hadoop.mapred.JobConf**: configuration class of MapReduce jobs and a major configuration API for users to submit jobs to Hadoop.

Table 11-3 Common APIs of **org.apache.hadoop.mapreduce.Job**

Function	Description
Job(Configuration conf, String jobName), Job(Configuration conf)	Creates a MapReduce client for configuring job attributes and submitting a job.
setMapperClass(Class<extends Mapper> cls)	A core API 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 API 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 API 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 API 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 API used to specify the local location of the JAR file of a class. Java uses the class file to find the JAR file, which is uploaded to HDFS.
setJar(String jar)	Specifies the local location of the JAR file of a class. You can directly set the location of a JAR file, which is uploaded to 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 API 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**, and specify the data format for the output. In the default TextOutputFormat, each key and value are recorded in text. OutputFormat is not specified usually.

Function	Description
setOutputKeyClass(Class< > theClass)	A core API 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 API 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 a Reduce class. HashPartitioner is used by default, and 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 intermediate data for transmission to lighten network pressure 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 11-4 Common APIs of **org.apache.hadoop.mapred.JobConf**

Method	Description
setNumMapTasks(int n)	A core API 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 allows the number of Map tasks to be set on the client.
setNumReduceTasks(int n)	A core API 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**.

11.5.2 What Should I Do if the Client Has No Response after a MapReduce Job is Submitted?

Question

What should I do if the client has no response for a long time after the MapReduce job is submitted to the Yarn server?

Answer

ResourceManager provides diagnosis information about key steps of MapReduce job execution on the web UI. For the MapReduce job that has been submitted to the Yarn server, you can obtain the current task status and the reason from the diagnosis information.

Procedures: On the management console of the public cloud, choose **Basic Information** > **Yarn Monitoring** to enter the web UI. Click the submitted MapReduce job, and view diagnosis information. Take measures based on the diagnosis information.

View MapReduce logs to learn application running status, and adjust applications based on log information.

12 OpenTSDB Development Guide

12.1 OpenTSDB Application Development Overview

12.1.1 Introduction to OpenTSDB Application Development

Introduction to OpenTSDB

OpenTSDB is a distributed, scalable time series database based on HBase. OpenTSDB is designed to collect monitoring information of a large-scale cluster and implement second-level data query, eliminating the limitations of querying and storing massive amounts of monitoring data in common databases.

Application scenarios of OpenTSDB have the following features:

- The collected metrics have a unique value at a time point and do not have a complex structure or relationship.
- Monitoring metrics change with time.
- Like HBase, OpenTSDB features high throughput and good scalability.

API Types

OpenTSDB provides an HTTP based application programming interface to enable integration with external systems. Almost all OpenTSDB features are accessible via the API such as querying time series data, managing metadata, and storing data points. For details, visit **http://opentsdb.net/docs/build/html/api_http/index.html**.

12.1.2 Common Concepts of OpenTSDB Application Development

Basic Concepts

- **data point**: A time series data point consists of a metric, a timestamp, a value, and a set of tags. The data point indicates the value of a metric at a specific time point.

- **metric**: Metrics include CPU usage, memory, and I/Os in system monitoring.
- **timestamp**: A UNIX timestamp (seconds or milliseconds since Epoch), that is, the time when the value is generated.
- **value**: The value of a metric is a JSON formatted event or a histogram/digest.
- **tag**: A tag is a key-value pair consisting of Tagk and Tagv. It describes the time series the point belongs to.

Tags allow you to separate similar data points from different sources or related entities, so you can easily graph them individually or in groups. One common use case for tags consists in annotating a data point with the name of the machine that produced it as well as name of the cluster or pool the machine belongs to. This allows you to easily make dashboards that show the state of your service on a per-server basis as well as dashboards that show an aggregated state across logical pools of servers.

Introduction to an OpenTSDB System Table

OpenTSDB stores time series data based on HBase. After OpenTSDB is enabled in a cluster, the system will create four HBase tables in the cluster. **Table 12-1** describes the OpenTSDB system tables.

NOTE

Do not modify the four HBase tables manually, because this may cause unavailable OpenTSDB.

Table 12-1 OpenTSDB system table

Table Name	Description
tsdb	It stores data points. All OpenTSDB data is stored in this table.
tsdb-meta	It stores time series indexes and metadata.
tsdb-tree	It stores metric structure information.
tsdb-uid	It stores unique identifier (UID) mappings. Each metric in a data point is mapped to a UID, and each tag in a data point is mapped to a UID. At the same time, each UID is reversely mapped to the metric or tag. These mappings are stored in this table.

12.1.3 OpenTSDB Application Development Process

Figure 12-1 and **Table 12-2** describe the phases in the development process.

Figure 12-1 OpenTSDB application development process

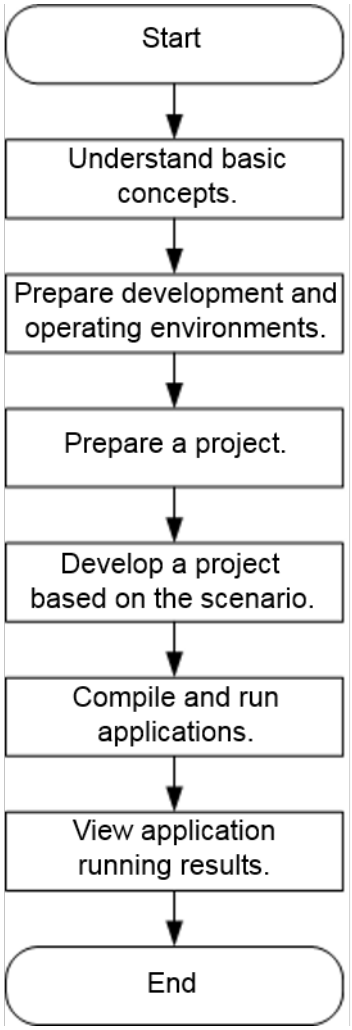


Table 12-2 Description of the OpenTSDB application development process

Phase	Description	Reference
Understand basic concepts.	Before application development, learn basic concepts of OpenTSDB, understand the scenario requirements, and design tables.	**Common Concepts of OpenTSDB Application Development**

Phase	Description	Reference
Prepare development and operating environments.	The Java language is recommended for OpenTSDB application development. You can use the Eclipse tool. The OpenTSDB operating environment is an OpenTSDB client. Install and configure the client according to the guide.	**OpenTSDB Application Development Environment**
Prepare a project.	OpenTSDB provides sample projects for different scenarios. You can import a sample project to learn the application. You can also create an OpenTSDB project according to the guide.	**Importing and Configuring an OpenTSDB Sample Project**
Develop a project based on the scenario.	A Java sample project is provided, including creating a metric, writing data, and querying data.	**OpenTSDB Development Plan**
Compile and run an application.	You can compile the developed application and submit it for running.	**Commissioning an OpenTSDB Application**
View application running results.	Application running results are exported to a path you specify. You can also view the status of the imported data on the UI.	**Viewing the OpenTSDB Application Commissioning Result**

12.2 Preparing an OpenTSDB Application Development Environment

12.2.1 OpenTSDB Application Development Environment

Table 12-3 describes the environment required for secondary development. You also need to prepare a Linux environment for verifying whether the application is running properly.

Table 12-3 Development environment

Item	Description
OS	Windows OS. Windows 7 or later is recommended.
JDK installation	Basic configurations of the development environment. JDK 1.8 or later is required.
Eclipse installation and configuration	Tool used for developing OpenTSDB applications.
Network	The client must be interconnected with the OpenTSDB server on the network.

12.2.2 Preparing an OpenTSDB Application Development Environment

- Install Eclipse and JDK in the Windows development environment.

  Install JDK 1.8 or later. Use Eclipse supporting JDK 1.8 or later, with the JUnit plug-in installed.

  NOTE

  - If you use IBM JDK, ensure that the JDK configured in Eclipse is IBM JDK.
  - If you use Oracle JDK, ensure that the JDK configured in Eclipse is Oracle JDK.
  - Do not use the same workspace and the sample project in the same path for different Eclipse programs.
- Prepare an environment for testing application running status.

Preparing a Running and Commissioning Environment

Step 1 On the ECS management console, apply for a new ECS for application running and commissioning.

- The security group of the ECS must be the same as that of the Master node in an MRS cluster.
- The ECS and the MRS cluster must be in the same VPC.
- The ECS NIC and the MRS cluster must be in the same network segment.

Step 2 Apply for an EIP, bind it to a new ECS, and configure an inbound or outbound rule for the security group.

Step 3 Download a client program.

1. Log in to **MRS Manager**.
2. Choose **Services** > **Download Client** to download the complete client to the remote host, that is, download the client program to the newly applied ECS.

Step 4 Log in to the node where the downloaded client is located, and then install the client.

1. Run the following command to decompress the client package:

   cd /opt

   tar -xvf /opt/MRS_Services_Client.tar

2. Run the following command to verify the installation file package:

   sha256sum -c /opt/MRS_Services_ClientConfig.tar.sha256

   ```
   MRS_Services_ClientConfig.tar:OK
   ```

3. Run the following command to decompress the installation file package:

   tar -xvf /opt/MRS_Services_ClientConfig.tar

4. Run the following command to install the client to a specified directory (absolute path), for example, **/opt/client**. The directory is automatically created.

   cd /opt/MRS_Services_ClientConfig

   sh install.sh /opt/client

   ```
   Components client installation is complete.
   ```

----End

12.2.3 Preparing an OpenTSDB Application Development User

The development user is used to run the sample project. The user must have HBase permissions to run the OpenTSDB sample project.

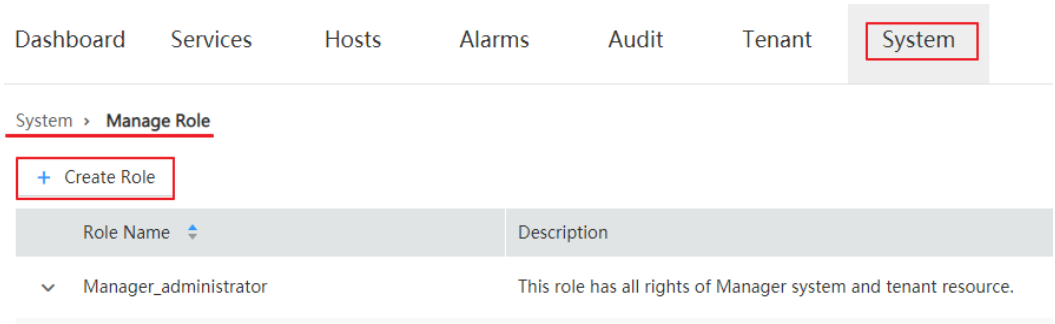
Prerequisites

Kerberos authentication has been enabled for the MRS cluster. Skip this step if Kerberos authentication is not enabled for the cluster.

Procedure

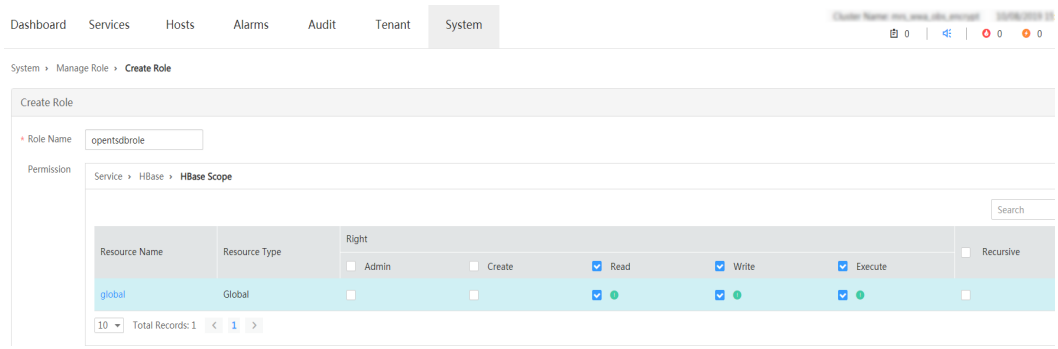
Step 1 Log in to **MRS Manager** and choose **System** > **Manage Role** > **Create Role**.

Figure 12-2 Creating a role



1. Enter a role name, for example, *opentsdbrole*.
2. Edit a role. Choose **HBase> HBase Scope > global** in **Permission**. Select **Read**, **Write**, and **Execute**, and click **OK**, as shown in **Figure 12-3**.

Figure 12-3 Editing a role

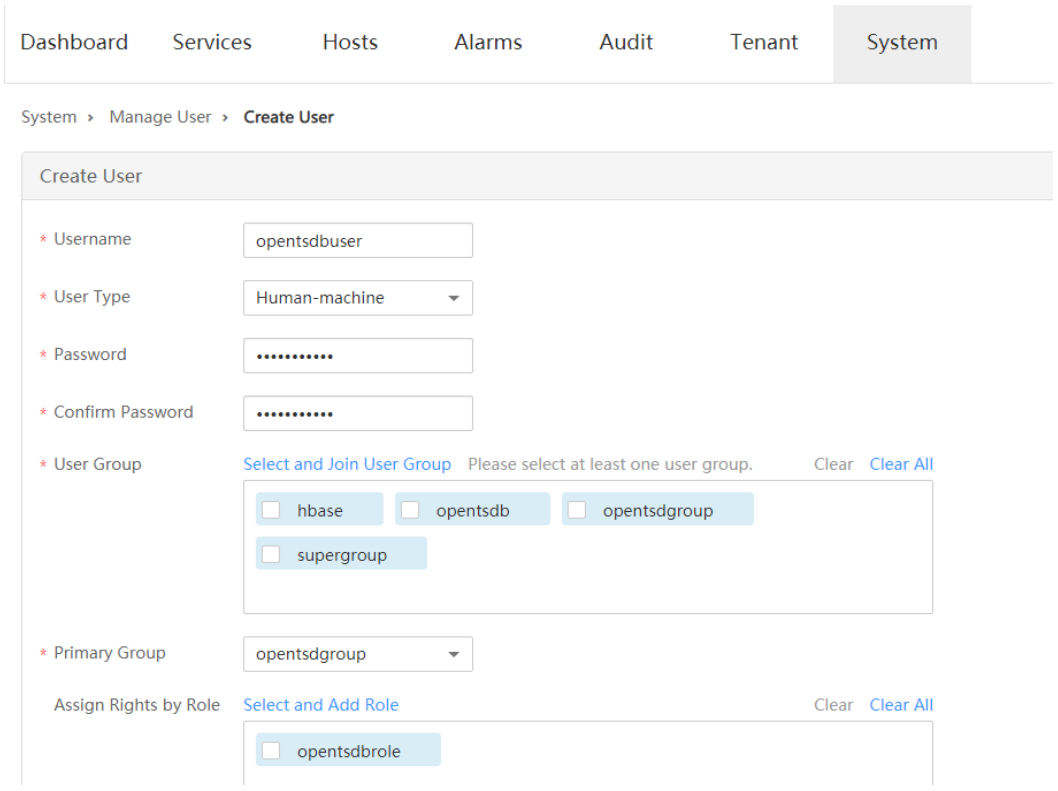


Step 2 Choose **System** > **Manage User Group** > **Create User Group** to create a user group for the sample project, for example, *opentsdgroup*.

Step 3 Choose **System** > **Manage User** > **Create User** to create a user for the sample project.

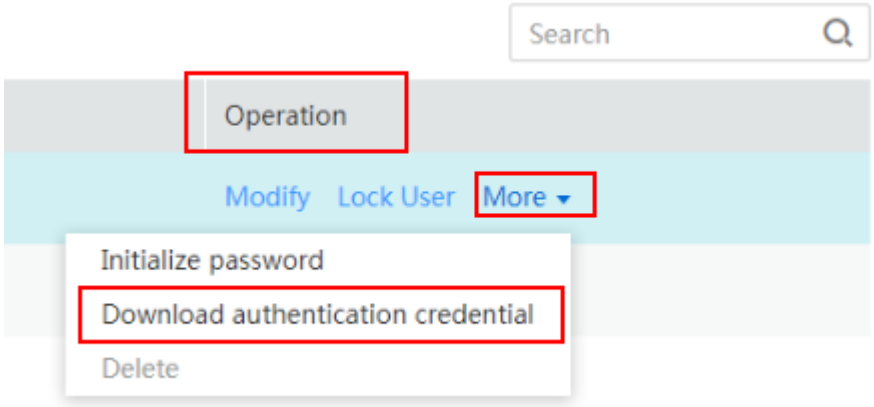
Step 4 Enter a username, for example, *opentsdbuser*. Set **User Type** to **Human-machine**, and select **opentsdb**, **hbase**, **opentsdbgroup**, and **supergroup** in **User Group**. Set **Primary Group** to **opentsdbgroup**, select **opentsdbrole** in **Assign Rights by Role**, and click **OK**. **Figure 12-4** shows the parameter settings.

Figure 12-4 Creating a user



Step 5 On MRS Manager, choose **System > Manage User**, select **opentsdbuser**, and modify the password. Choose **More > Download authentication credential** from the **Operation** column on the right. Save the file and decompress it to obtain the **user.keytab** and **krb5.conf** files. The two files are used for security authentication in the sample project, as shown in **Figure 12-5**.

Figure 12-5 Downloading the authentication credential



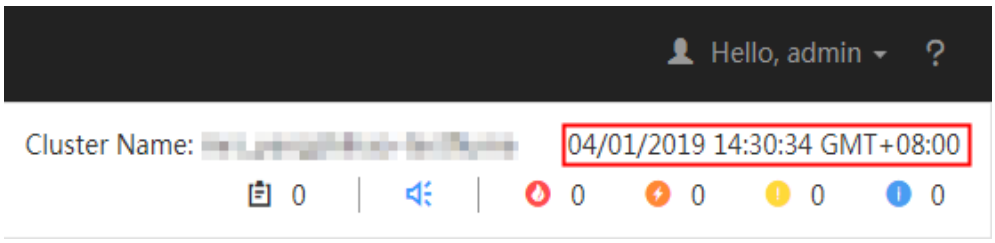
----End

12.2.4 Importing and Configuring an OpenTSDB Sample Project

Prerequisites

Ensure that the time difference between a local computer and the MRS cluster is less than 5 minutes. Time of the MRS cluster can be viewed in the upper right corner on the MRS Manager page. See **Figure 12-6**.

Figure 12-6 Time of the MRS cluster



Procedure

Step 1 Obtain the OpenTSDB sample project by referring to **Obtaining the MRS Application Development Sample Project**.

Step 2 In the root directory of the OpenTSDB sample project, run the **mvn install** command to perform compilation.

Step 3 In the root directory of the OpenTSDB sample project, run the **mvn eclipse:eclipse** command to create an Eclipse project.

Step 4 In the application development environment, import the sample project to the Eclipse development environment.

1. Choose **File** > **Import** > **General** > **Existing Projects into Workspace** > **Next** > **Browse**.

   The **Browse Folder** dialog box is displayed.

2. Select the sample project folder, and click **Finish**.

Step 5 Set an Eclipse text file encoding format to prevent garbled characters.

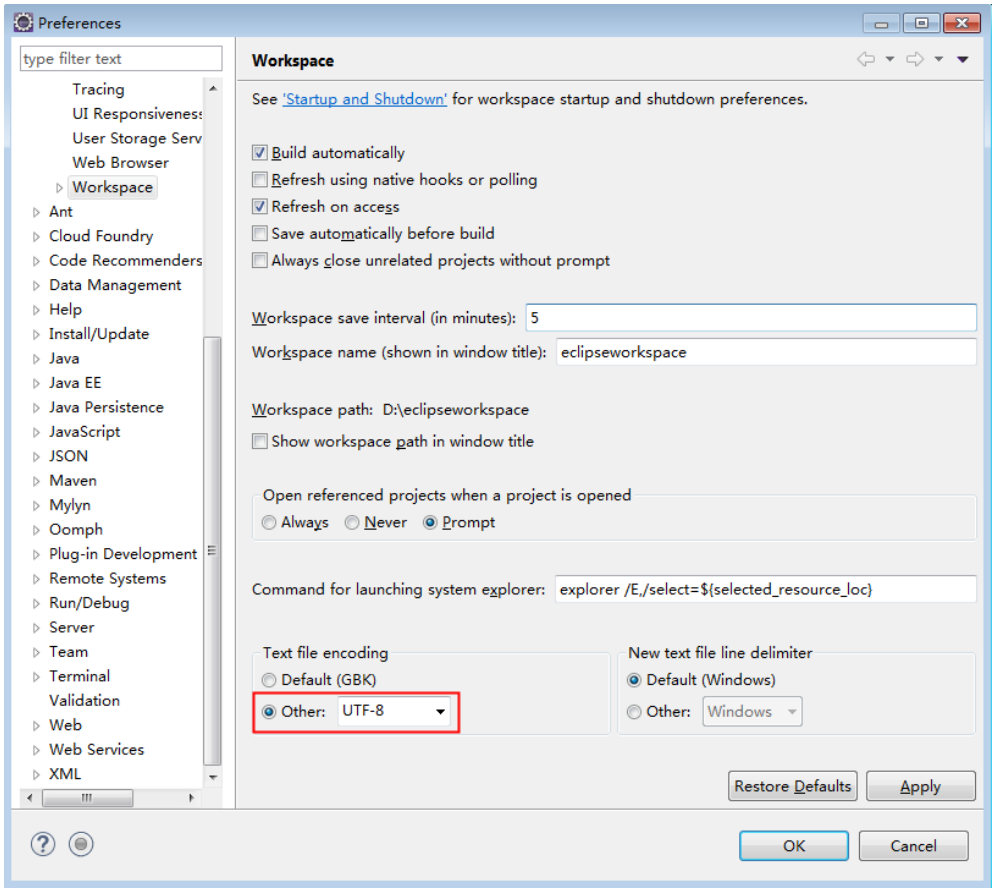
1. On the Eclipse menu bar, choose **Window** > **Preferences**.

   The **Preferences** window is displayed.

2. In the navigation tree, choose **General** > **Workspace**. In the **Text file encoding** area, select **Other** and set the value to **UTF-8**. Click **Apply** and then **OK**. Figure 12-7 shows the settings.

   Setting the Eclipse encoding format

Figure 12-7 Setting the Eclipse encoding format



----End

12.3 Developing an OpenTSDB Application

12.3.1 OpenTSDB Development Plan

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

Scenario Description

Assume that a user develops an application to record and query the weather information of a city. **Table 12-4**, **Table 12-5**, and **Table 12-6** describe the recorded data.

Table 12-4 Raw data

City	District	Time	Temperature	Humidity
Shenzhen	Longgang	2017/7/1 00:00:00	28	54
Shenzhen	Longgang	2017/7/1 01:00:00	27	53
Shenzhen	Longgang	2017/7/1 02:00:00	27	52
Shenzhen	Longgang	2017/7/1 03:00:00	27	51
Shenzhen	Longgang	2017/7/1 04:00:00	27	50
Shenzhen	Longgang	2017/7/1 05:00:00	27	49
Shenzhen	Longgang	2017/7/1 06:00:00	27	48
Shenzhen	Longgang	2017/7/1 07:00:00	27	46
Shenzhen	Longgang	2017/7/1 08:00:00	29	46
Shenzhen	Longgang	2017/7/1 09:00:00	30	48
Shenzhen	Longgang	2017/7/1 10:00:00	32	48
Shenzhen	Longgang	2017/7/1 11:00:00	32	49
Shenzhen	Longgang	2017/7/1 12:00:00	33	49
Shenzhen	Longgang	2017/7/1 13:00:00	33	50
Shenzhen	Longgang	2017/7/1 14:00:00	32	50
Shenzhen	Longgang	2017/7/1 15:00:00	32	50

City	District	Time	Temperature	Humidity
Shenzhen	Longgang	2017/7/1 16:00:00	31	51
Shenzhen	Longgang	2017/7/1 17:00:00	30	51
Shenzhen	Longgang	2017/7/1 18:00:00	30	51
Shenzhen	Longgang	2017/7/1 19:00:00	29	51
Shenzhen	Longgang	2017/7/1 20:00:00	29	52
Shenzhen	Longgang	2017/7/1 21:00:00	29	53
Shenzhen	Longgang	2017/7/1 22:00:00	28	54
Shenzhen	Longgang	2017/7/1 23:00:00	28	54

In this scenario, the temperature and humidity data of the Longgang district, Shenzhen, is recorded at 00:00 on July 1, 2017. OpenTSDB uses two groups of data points for modeling.

Table 12-5 Metric data point 1

Metric	City	District	Unix timestamp	Value
city.temp	Shenzhen	Longgang	1498838400	28
city.temp	Shenzhen	Longgang	1498842000	27
city.temp	Shenzhen	Longgang	1498845600	27
city.temp	Shenzhen	Longgang	1498849200	27
city.temp	Shenzhen	Longgang	1498852800	27
city.temp	Shenzhen	Longgang	1498856400	27
city.temp	Shenzhen	Longgang	1498860000	27
city.temp	Shenzhen	Longgang	1498863600	27
city.temp	Shenzhen	Longgang	1498867200	29
city.temp	Shenzhen	Longgang	1498870800	30
city.temp	Shenzhen	Longgang	1498874400	32

Metric	City	District	Unix timestamp	Value
city.temp	Shenzhen	Longgang	1498878000	32
city.temp	Shenzhen	Longgang	1498881600	33
city.temp	Shenzhen	Longgang	1498885200	33
city.temp	Shenzhen	Longgang	1498888800	32
city.temp	Shenzhen	Longgang	1498892400	32
city.temp	Shenzhen	Longgang	1498896000	31
city.temp	Shenzhen	Longgang	1498899600	30
city.temp	Shenzhen	Longgang	1498903200	30
city.temp	Shenzhen	Longgang	1498906800	29
city.temp	Shenzhen	Longgang	1498910400	29
city.temp	Shenzhen	Longgang	1498914000	29
city.temp	Shenzhen	Longgang	1498917600	28
city.temp	Shenzhen	Longgang	1498921200	28

Table 12-6 Metric data point 2

Metric	City	District	Unix timestamp	Value
city.hum	Shenzhen	Longgang	1498838400	54
city.hum	Shenzhen	Longgang	1498842000	53
city.hum	Shenzhen	Longgang	1498845600	52
city.hum	Shenzhen	Longgang	1498849200	51
city.hum	Shenzhen	Longgang	1498852800	50
city.hum	Shenzhen	Longgang	1498856400	49
city.hum	Shenzhen	Longgang	1498860000	48
city.hum	Shenzhen	Longgang	1498863600	46
city.hum	Shenzhen	Longgang	1498867200	46
city.hum	Shenzhen	Longgang	1498870800	48
city.hum	Shenzhen	Longgang	1498874400	48
city.hum	Shenzhen	Longgang	1498878000	49

Metric	City	District	Unix timestamp	Value
city.hum	Shenzhen	Longgang	1498881600	49
city.hum	Shenzhen	Longgang	1498885200	50
city.hum	Shenzhen	Longgang	1498888800	50
city.hum	Shenzhen	Longgang	1498892400	50
city.hum	Shenzhen	Longgang	1498896000	51
city.hum	Shenzhen	Longgang	1498899600	51
city.hum	Shenzhen	Longgang	1498903200	51
city.hum	Shenzhen	Longgang	1498906800	51
city.hum	Shenzhen	Longgang	1498910400	52
city.hum	Shenzhen	Longgang	1498914000	53
city.hum	Shenzhen	Longgang	1498917600	54
city.hum	Shenzhen	Longgang	1498921200	54

Each group of metric data points has two tags:

- Tags: **City** and **District**
- Tag values: ShenZhen and Longgang

You can perform the following operations on data:

- Obtain the daily monitored data and write data points of the two groups to the database through the **put** API of OpenTSDB.
- Use the query API of OpenTSDB to query and analyze the existing data.

Function Description

Develop functions based on the preceding scenario. **Table 12-7** describes functions to be developed.

Table 12-7 Functions to be developed in OpenTSDB

No.	Step	Code Implementation
1	Build a data model based on the typical scenario description.	For details, see **Configuring OpenTSDB Parameters**.
2	Write metric data.	For details, see **Writing Data into OpenTSDB**.

No.	Step	Code Implementation
3	Query data based on metrics.	For details, see **Querying OpenTSDB Data**.
4	Delete data in a specified range.	For details, see **Deleting OpenTSDB Data**.

12.3.2 Configuring OpenTSDB Parameters

Step 1 Before executing the sample code, you must change the values of the following parameters in **opentsdb.properties** in the **resources** directory of the sample code project.

```
tsd_hostname = node-ana-coreYQnTx
tsd_port = 4242
tsd_protocol = https
```

- **tsd_hostname**: Modify this parameter to the host name or IP address of the TSD instance that connects to the OpenTSDB service.

  NOTE

  - If the current running environment and the OpenTSDB installation environment are in the same VPC, you can use either the IP address or host name of the connected TSD instance.
  - If the current running environment and the OpenTSDB installation environment are in different VPCs, you can only use the host name to access the environment. In addition, you need to bind an EIP to the connected TSD instance, and configure the EIP and the host name of the TSD instance in the **hosts** file. The file path is **/etc/hosts** on Linux and **C:\Windows\System32\drivers\etc\hosts** on Windows.

  For example, if the host name of the TSD instance is **node-ana-corexxqm** and the corresponding EIP is **100.94.10.10**, enter the following information:

  ```
  100.94.10.10 node-ana-coreYQnTx
  ```

- **tsd_port**: TSD port. The default value is **4242**.
- **tsd_protocol**: request protocol. The default value is **https**.

Step 2 (Optional) If the sample project is not used, add the following dependencies to the **pom.xml** file of your project:

- guava

  ```
  <!-- https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient -->
  <dependency>
      <groupId>com.google.guava</groupId>
      <artifactId>guava</artifactId>
      <version>18.0</version>
  </dependency>
  ```

- gson

  ```
  <!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
  <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.2.4</version>
  </dependency>
  ```

- httpcore

  ```
  <!-- https://mvnrepository.com/artifact/org.apache.httpcomponents/httpcore -->
  <dependency>
      <groupId>org.apache.httpcomponents</groupId>
  ```



```
    <artifactId>httpcore</artifactId>
    <version>4.4.4</version>
</dependency>
```

- httpclient

```
<!-- https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient -->
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.2</version>
</dependency>
```

Step 3 Set a timeout interval for each HTTP request as follows:

```
public static void addTimeout(HttpRequestBase req) {
  RequestConfig requestConfig = RequestConfig.custom().setConnectTimeout(5000)
      .setConnectionRequestTimeout(10000).setSocketTimeout(60000).build();
  req.setConfig(requestConfig);
}
```

----End

12.3.3 Writing Data into OpenTSDB

Function Description

You can use the OpenTSDB API (/api/put) to write data.

Function **genWeatherData ()** simulates the generated weather data. The function **putData()** sends weather data to the OpenTSDB server.

Sample Code

The following code snippets are in the **putData** method of the **OpentsdbExample** class in the **com.huawei.bigdata.opentsdb.examples** packet.

```
private void putData(String tmpURL) {
  PUT_URL = BASE_URL + tmpURL;
  LOG.info("start to put data in opentsdb, the url is " + PUT_URL);
  try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
    HttpPost httpPost = new HttpPost(PUT_URL);//A timeout interval must be set for the request.
    addTimeout(httpPost);
    String weatherData = genWeatherData();
    StringEntity entity = new StringEntity(weatherData, "ISO-8859-1");
    entity.setContentType("application/json");
    httpPost.setEntity(entity);
    HttpResponse response = httpClient.execute(httpPost);
    int statusCode = response.getStatusLine().getStatusCode();
    LOG.info("Status Code : " + statusCode);
    if (statusCode != HttpStatus.SC_NO_CONTENT) {
      LOG.info("Request failed! " + response.getStatusLine());
    }
    LOG.info("put data to opentsdb successfully.");
  } catch (IOException e) {
    LOG.error("Failed to put data.", e);
  }
}

static class DataPoint {
  public String metric;
  public Long timestamp;
  public Double value;
  public Map<String, String> tags;
  public DataPoint(String metric, Long timestamp, Double value, Map<String, String> tags) {
    this.metric = metric;
    this.timestamp = timestamp;
    this.value = value;
```



```
    this.tags = tags;
  }
}

private String genWeatherData() {
  List<DataPoint> dataPoints = new ArrayList<DataPoint>();
  Map<String, String> tags = ImmutableMap.of("city", "Shenzhen", "region", "Longgang");

  // Data of air temperature
  dataPoints.add(new DataPoint("city.temp", 1498838400L, 28.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498842000L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498845600L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498849200L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498852800L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498856400L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498860000L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498863600L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498867200L, 29.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498870800L, 30.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498874400L, 32.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498878000L, 32.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498881600L, 33.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498885200L, 33.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498888800L, 32.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498892400L, 32.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498896000L, 31.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498899600L, 30.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498903200L, 30.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498906800L, 29.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498910400L, 29.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498914000L, 29.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498917600L, 28.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498921200L, 28.0, tags));

  // Data of humidity
  dataPoints.add(new DataPoint("city.hum", 1498838400L, 54.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498842000L, 53.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498845600L, 52.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498849200L, 51.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498852800L, 50.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498856400L, 49.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498860000L, 48.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498863600L, 46.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498867200L, 46.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498870800L, 48.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498874400L, 48.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498878000L, 49.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498881600L, 49.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498885200L, 50.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498888800L, 50.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498892400L, 50.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498896000L, 51.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498899600L, 51.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498903200L, 51.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498906800L, 51.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498910400L, 52.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498914000L, 53.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498917600L, 54.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498921200L, 54.0, tags));

  Gson gson = new Gson();
  return gson.toJson(dataPoints);
}
```


NOTE

The **sync** parameter is added to **PUT_URL**, indicating that data can be returned only after data is written to HBase. This parameter is strongly recommended. If **sync** is not used, data is asynchronously written to HBase, which may cause data loss. For details, see **FAQs About OpenTSDB Application Development**.

12.3.4 Querying OpenTSDB Data

Function Description

You can use the OpenTSDB query API (/api/query) to read data.

Function **genQueryReq ()** generates a query request, and function **queryData()** sends the query request to the OpenTSDB server.

Sample Code

The following code snippets are in the **queryData** method of the **OpentsdbExample** class in the **com.huawei.bigdata.opentsdb.examples** packet.

```
private void queryData(String dataPoint) {
  QUERY_URL = BASE_URL + dataPoint;
  LOG.info("start to query data in opentsdb, the url is " + QUERY_URL);
  try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
    HttpPost httpPost = new HttpPost(QUERY_URL);//A timeout interval must be set for the request.
    addTimeout(httpPost);
    String queryRequest = genQueryReq();
    StringEntity entity = new StringEntity(queryRequest, "utf-8");
    entity.setContentType("application/json");
    httpPost.setEntity(entity);
    HttpResponse response = httpClient.execute(httpPost);
    int statusCode = response.getStatusLine().getStatusCode();
    LOG.info("Status Code : " + statusCode);
    if (statusCode != HttpStatus.SC_OK) {
      LOG.info("Request failed! " + response.getStatusLine());
    }
    String body = EntityUtils.toString(response.getEntity(), "utf-8");
    LOG.info("Response content : " + body);
    LOG.info("query data to opentsdb successfully.");
  } catch (IOException e) {
    LOG.error("Failed to query data.", e);
  }
}

static class Query {
  public Long start;
  public Long end;
  public boolean delete = false;
  public List<SubQuery> queries;
}

static class SubQuery {
  public String metric;
  public String aggregator;
  public SubQuery(String metric, String aggregator) {
    this.metric = metric;
    this.aggregator = aggregator;
  }
}

String genQueryReq() {
  Query query = new Query();
  query.start = 1498838400L;
```



```
  query.end = 1498921200L;
  query.queries = ImmutableList.of(new SubQuery("city.temp", "sum"), new SubQuery("city.hum", "sum"));
  Gson gson = new Gson();
  return gson.toJson(query);
}
```

12.3.5 Deleting OpenTSDB Data

Function Description

Add the **delete** parameter to the query API of OpenTSDB and set it to **true**. Function **genQueryReq ()** generates a deletion request, and function **deleteData()** sends the deletion request to the OpenTSDB server.

Sample Code

The following code snippets are in the **deleteData** method of the **OpentsdbExample** class in the **com.huawei.bigdata.opentsdb.examples** packet.

```
public void deleteData(String dataPoint) {
  QUERY_URL = BASE_URL + dataPoint;
  try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
    HttpPost httpPost = new HttpPost(QUERY_URL);addTimeout(httpPost);
    String deleteRequest = genDeleteReq();
    StringEntity entity = new StringEntity(deleteRequest, "utf-8");
    entity.setContentType("application/json");
    httpPost.setEntity(entity);
    HttpResponse response = httpClient.execute(httpPost);
    int statusCode = response.getStatusLine().getStatusCode();
    LOG.info("Status Code : " + statusCode);
    if (statusCode != HttpStatus.SC_OK) {
      LOG.info("Request failed! " + response.getStatusLine());
    }
  } catch (IOException e) {
    LOG.error("Failed to delete data.", e);
  }
}

static class Query {
  public Long start;
  public Long end;
  public boolean delete = false;
  public List<SubQuery> queries;
}

static class SubQuery {
  public String metric;
  public String aggregator;
  public SubQuery(String metric, String aggregator) {
    this.metric = metric;
    this.aggregator = aggregator;
  }
}

String genQueryReq() {
  Query query = new Query();
  query.start = 1498838400L;
  query.end = 1498921200L;
  query.queries = ImmutableList.of(new SubQuery("city.temp", "sum"), new SubQuery("city.hum", "sum"));
  Gson gson = new Gson();
  return gson.toJson(query);
}

String genDeleteReq() {
  Query query = new Query();
```



```
  query.start = 1498838400L;
  query.end = 1498921200L;
  query.queries = ImmutableList.of(new SubQuery("city.temp", "sum"), new SubQuery("city.hum", "sum"));
  query.delete = true;

  Gson gson = new Gson();
  return gson.toJson(query);
}
```

12.4 Commissioning an OpenTSDB Application

12.4.1 Commissioning Applications on Windows

12.4.1.1 Commissioning an OpenTSDB Application

Scenario

You can run applications in the Windows development environment after application code is developed.

Procedure

Step 1 Configure a mapping between the cluster IP address and host name on Windows. Log in to the cluster background, run the **cat /etc/hosts** command, and copy the mapping between IP addresses and host names in the **hosts** file to **C:\Windows\System32\drivers\etc\hosts**.

```
xx.xx.xx.xx node-ana-corejnWt
xx.xx.xx.xx node-ana-coreddlI
```


NOTE

You can use either of the following method to access an MRS cluster to operate OpenTSDB on Windows.

- Apply for a Windows ECS to access the MRS cluster to operate OpenTSDB. Run the sample code after the development environment is installed. To apply for ECS to access the MRS cluster, perform the following steps:
  1. On the **Active Clusters** page, click the name of an existing cluster.

     On the cluster details page, record the **AZ**, **VPC**, **Cluster Manager IP Address** of the cluster, and **Default Security Group** of the Master node.
  2. On the ECS management console, create a new ECS.

     The **AZ**, **VPC**, and **security group** of ECS must be the same as those of the cluster to be accessed.

     Select a Windows public image.

     For details about other configuration parameters, see **Elastic Cloud Server** > **Quick Start** > **Purchasing and Logging In to a Windows ECS**.
- Use the local host to access the MRS cluster to operate OpenTSDB. Bind an EIP to the TSD instance that OpenTSDB in the MRS cluster accesses. When configuring the mapping between the IP address and host name of the cluster on the local Windows host, replace the IP address with the EIP corresponding to the host name. Run the sample code. To bind an EIP, perform the following steps:
  1. On the VPC management console, apply for an EIP and bind it to ECS.

     For details, see **Virtual Private Cloud** > **User Guide** > **Elastic IP Address** > **Assigning an EIP and Binding It to an ECS**.
  2. Open security group rules for the MRS cluster.

     Add security group rules to the security groups of Master and Core nodes in the cluster so that ECS can access the cluster. For details, see **Virtual Private Cloud** > **User Guide** > **Security** > **Security Group** > **Adding a Security Group Rule**.

Step 2 Copy the **/opt/Bigdata/jdk1.8.0_212/jre/lib/security/cacerts** file on the master node in the cluster to the JDK directory on Windows, for example, **C:\Program Files\Java\jdk1.8.0_73\jre\lib\security**.

Step 3 Modify related configurations. Modify the **opentsdb.properties** file in the **resources** directory of the sample project and configure the OpenTSDB properties.

```
tsd_hostname = node-ana-corejnWt
tsd_port = 4242
tsd_protocol = https
```

- **tsd_hostname**: name of the connected TSD instance host for accessing OpenTSDB
- **tsd_port**: port for accessing OpenTSDB. The default port number is **4242**.
- **tsd_protocol**: request protocol for accessing OpenTSDB. The default value is **https**.

Step 4 Run the sample project.

In the development environment (for example, Eclipse), right-click **OpentsdbExample.java** and choose **Run as** > **Java Application** from the shortcut menu to run the corresponding application project.

----End

12.4.1.2 Viewing the OpenTSDB Application Commissioning Result

Scenario

After OpenTSDB sample project running is complete, you can obtain the running status by viewing the running result or logs.

Procedure

- If the application running is successful, the following information is displayed.

```
2019-06-27 14:05:20,713 INFO  [main] examples.OpentsdbExample: start to put data in opentsdb, the url is https://node-ana-corejnWt:4242/api/put/?sync&sync_timeout=60000
2019-06-27 14:05:23,680 INFO  [main] examples.OpentsdbExample: Status Code : 204
2019-06-27 14:05:23,680 INFO  [main] examples.OpentsdbExample: put data to opentsdb successfully.
2019-06-27 14:05:23,681 INFO  [main] examples.OpentsdbExample: start to query data in opentsdb, the url is https://node-ana-corejnWt:4242/api/query
2019-06-27 14:05:23,895 INFO  [main] examples.OpentsdbExample: Status Code : 200
2019-06-27 14:05:23,897 INFO  [main] examples.OpentsdbExample: Response content :
[{"metric":"city.temp","tags":{"region":"Longgang","city":"Shenzhen"},"aggregateTags":[],"dps":
{"1498838400":28.0,"1498842000":27.0,"1498845600":27.0,"1498849200":27.0,"1498852800":27.0,"1498856400":27.0,"1498860000":27.0,"1498863600":27.0,"1498867200":29.0,"1498870800":30.0,"1498874400":32.0,"1498878000":32.0,"1498881600":33.0,"1498885200":33.0,"1498888800":32.0,"1498892400":32.0,"1498896000":31.0,"1498899600":30.0,"1498903200":30.0,"1498906800":29.0,"1498910400":29.0,"1498914000":29.0,"1498917600":28.0,"1498921200":28.0}},{"metric":"city.hum","tags":
{"region":"Longgang","city":"Shenzhen"},"aggregateTags":[],"dps":
{"1498838400":54.0,"1498842000":53.0,"1498845600":52.0,"1498849200":51.0,"1498852800":50.0,"1498856400":49.0,"1498860000":48.0,"1498863600":46.0,"1498867200":46.0,"1498870800":48.0,"1498874400":48.0,"1498878000":49.0,"1498881600":49.0,"1498885200":50.0,"1498888800":50.0,"1498892400":50.0,"1498896000":51.0,"1498899600":51.0,"1498903200":51.0,"1498906800":51.0,"1498910400":52.0,"1498914000":53.0,"1498917600":54.0,"1498921200":54.0}}]
2019-06-27 14:05:23,897 INFO  [main] examples.OpentsdbExample: query data to opentsdb successfully.
2019-06-27 14:05:23,897 INFO  [main] examples.OpentsdbExample: start to delete data in opentsdb, the url is https://node-ana-corejnWt:4242/api/query
2019-06-27 14:05:24,112 INFO  [main] examples.OpentsdbExample: Status Code : 200
2019-06-27 14:05:24,112 INFO  [main] examples.OpentsdbExample: delete data to opentsdb successfully.
```

- Log description

The log level is INFO by default and you can view more detailed information by changing the log level, such as DEBUG, INFO, WARN, ERROR, and FATAL. You can modify the **log4j.properties** file to change log levels, for example:

```
# Define some default values that can be overridden by system properties
opentsdb.root.logger=INFO,console
# Define the root logger to the system property "opentsdb.root.logger".
log4j.rootLogger=${opentsdb.root.logger}
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{ISO8601} %-5p [%t] %c{2}: %m%n
```

12.4.2 Commissioning Applications on Linux

12.4.2.1 Commissioning an OpenTSDB Application

OpenTSDB applications can run in a Linux environment where an OpenTSDB client is installed. After application code development is complete, you can upload a JAR file to the Linux environment to run applications.

Prerequisites

- You have installed a JDK in the Linux environment. The version of the JDK must be consistent with that of the JDK used by Eclipse to export the JAR file.
- 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 locates. The host names and IP addresses must be mapped one by one.

Procedure

Step 1 Log in to the Linux environment, create a directory for running the OpenTSDB sample project, for example, **/opt/opentsdb-example**, and a directory for storing configuration files, for example, **/opt/opentsdb-example/conf**. Then, edit the **/opt/opentsdb-example/conf/opentsdb.properties** configuration file based on actual requirements.

```
mkdir -p /opt/opentsdb-example/conf
[root@node-master1rLqO ~]# cat /opt/opentsdb-example/conf/opentsdb.properties
tsd_hostname = node-ana-corejnWt
tsd_port = 4242
tsd_protocol = https
```

Step 2 Run the **mvn package** command to generate a JAR file, for example, **opentsdb-examples-mrs-xxx-jar-with-dependencies.jar**, in which **mrs-xxx** indicates an MRS version. Obtain the JAR file from the target directory in the project directory, and upload it to the **/opt/opentsdb-example** directory.

Step 3 Run the JAR file.

Load the environment variables.

```
source /opt/client/bigdata_env
```

Authenticate the cluster user. (Skip this step for a cluster with Kerberos disabled.)

Human-machine user: *kinit kerberos user*

Machine-machine user: *kinit -kt authentication file path kerberos user*

Run the OpenTSDB sample program.

```
java -cp /opt/opentsdb-example/conf:/opt/opentsdb-example/opentsdb-examples-mrs-xxx-jar-with-dependencies.jar com.huawei.bigdata.opentsdb.examples.OpentsdbExample
```

----End

12.4.2.2 Viewing the OpenTSDB Application Commissioning Result

Scenario

After OpenTSDB sample project running is complete, you can obtain the running status by viewing the running result or logs.

Procedure

- If the application running is successful, the following information is displayed.

```
2019-06-27 14:05:20,713 INFO  [main] examples.OpentsdbExample: start to put data in opentsdb, the url is https://node-ana-corejnWt:4242/api/put/?sync&sync_timeout=60000
2019-06-27 14:05:23,680 INFO  [main] examples.OpentsdbExample: Status Code : 204
```



```
2019-06-27 14:05:23,680 INFO  [main] examples.OpentsdbExample: put data to opentsdb successfully.
2019-06-27 14:05:23,681 INFO  [main] examples.OpentsdbExample: start to query data in opentsdb, the url is https://node-ana-corejnWt:4242/api/query
2019-06-27 14:05:23,895 INFO  [main] examples.OpentsdbExample: Status Code : 200
2019-06-27 14:05:23,897 INFO  [main] examples.OpentsdbExample: Response content :
[{"metric":"city.temp","tags":{"region":"Longgang","city":"Shenzhen"},"aggregateTags":[],"dps":
{"1498838400":28.0,"1498842000":27.0,"1498845600":27.0,"1498849200":27.0,"1498852800":27.0,"1498856400":27.0,"1498860000":27.0,"1498863600":27.0,"1498867200":29.0,"1498870800":30.0,"1498874400":32.0,"1498878000":32.0,"1498881600":33.0,"1498885200":33.0,"1498888800":32.0,"1498892400":32.0,"1498896000":31.0,"1498899600":30.0,"1498903200":30.0,"1498906800":29.0,"1498910400":29.0,"1498914000":29.0,"1498917600":28.0,"1498921200":28.0}},{"metric":"city.hum","tags":
{"region":"Longgang","city":"Shenzhen"},"aggregateTags":[],"dps":
{"1498838400":54.0,"1498842000":53.0,"1498845600":52.0,"1498849200":51.0,"1498852800":50.0,"1498856400":49.0,"1498860000":48.0,"1498863600":46.0,"1498867200":46.0,"1498870800":48.0,"1498874400":48.0,"1498878000":49.0,"1498881600":49.0,"1498885200":50.0,"1498888800":50.0,"1498892400":50.0,"1498896000":51.0,"1498899600":51.0,"1498903200":51.0,"1498906800":51.0,"1498910400":52.0,"1498914000":53.0,"1498917600":54.0,"1498921200":54.0}}]
2019-06-27 14:05:23,897 INFO  [main] examples.OpentsdbExample: query data to opentsdb successfully.
2019-06-27 14:05:23,897 INFO  [main] examples.OpentsdbExample: start to delete data in opentsdb, the url is https://node-ana-corejnWt:4242/api/query
2019-06-27 14:05:24,112 INFO  [main] examples.OpentsdbExample: Status Code : 200
2019-06-27 14:05:24,112 INFO  [main] examples.OpentsdbExample: delete data to opentsdb successfully.
```

- Log description

The log level is INFO by default and you can view more detailed information by changing the log level, such as DEBUG, INFO, WARN, ERROR, and FATAL. You can modify the **log4j.properties** file to change log levels, for example:

```
# Define some default values that can be overridden by system properties
opentsdb.root.logger=INFO,console
# Define the root logger to the system property "opentsdb.root.logger".
log4j.rootLogger=${opentsdb.root.logger}
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{ISO8601} %-5p [%t] %c{2}: %m%n
```

12.5 FAQs About OpenTSDB Application Development

12.5.1 OpenTSDB CLI Tools

OpenTSDB provides client tools that can directly invoke commands to operate OpenTSDB. The version of a client tool is the same as that of the open source community. For details, visit **https://opentsdb.net/docs/build/html/user_guide/cli/index.html**.

Usage of the client tools:

Step 1 Log in to any Master node.

Step 2 Run the following command to initialize environment variables:

source /opt/client/bigdata_env

Step 3 If the Kerberos authentication is enabled for the current cluster, run the following command to authenticate the user. If the Kerberos authentication is disabled for the current cluster, skip this step.

kinit *MRS cluster user*

For example, **kinit opentsdbuser**

Step 4 Run the **tsdb** command. For example, you can run the **tsdb** command to print all commands supported by OpenTSDB, such as, **fsck**, **import**, **mkmetric**, **query**, **tsd**, **scan**, **search**, **uid**, and **version**.

```
tsdb: error: unknown command ''
usage: tsdb <command> [args]
Valid commands: fsck, import, mkmetric, query, tsd, scan, search, uid, version
```

----End

Creating an OpenTSDB Metric

Create a metric that can be saved to OpenTSDB. For example, run the **tsdb mkmetric sys.cpu.user** command to create **sys.cpu.user**.

```
Start run net.opentsdb.tools.UidManager, args: assign metrics sys.cpu.user
metrics sys.cpu.user: [0, 0, 6]
```

Querying the OpenTSDB Metric

You can run the **tsdb** command to obtain the metric saved in OpenTSDB, for example, **tsdb uid metrics sys.cpu.user**.

```
Start run net.opentsdb.tools.UidManager, args: metrics sys.cpu.user
metrics sys.cpu.user: [0, 0, 6]
```

To obtain more information, run the **tsdb uid** command.

```
Start run net.opentsdb.tools.UidManager, args:
Not enough arguments
Usage: uid <subcommand> args
Sub commands:
  grep [kind] <RE>: Finds matching IDs.
  assign <kind> <name> [names]: Assign an ID for the given name(s).
  rename <kind> <name> <newname>: Renames this UID.
  delete <kind> <name>: Deletes this UID.
  fsck: [fix] [delete_unknown] Checks the consistency of UIDs.
        fix            - Fix errors. By default errors are logged.
        delete_unknown - Remove columns with unknown qualifiers.
                         The "fix" flag must be supplied as well.

  [kind] <name>: Lookup the ID of this name.
  [kind] <ID>: Lookup the name of this ID.
  metasync: Generates missing TSUID and UID meta entries, updates
            created timestamps
  metapurge: Removes meta data entries from the UID table
  treesync: Process all timeseries meta objects through tree rules
  treepurge <id> [definition]: Purge a tree and/or the branches
            from storage. Provide an integer Tree ID and optionally
            add "true" to delete the tree definition

Example values for [kind]: metrics, tagk (tag name), tagv (tag value).
  --config=PATH    Path to a configuration file (default: Searches for file see docs).
  --idwidth=N      Number of bytes on which the UniqueId is encoded.
  --ignore-case    Ignore case distinctions when matching a regexp.
  --table=TABLE    Name of the HBase table where to store the time series (default: tsdb).
  --uidtable=TABLE Name of the HBase table to use for Unique IDs (default: tsdb-uid).
  --verbose        Print more logging messages and not just errors.
  --zkbasedir=PATH Path under which is the znode for the -ROOT- region (default: /hbase).
  --zkquorum=SPEC  Specification of the ZooKeeper quorum to use (default: localhost).
  -i               Short for --ignore-case.
  -v               Short for --verbose.
```

Importing Data to the OpenTSDB Metric

You can run the **tsdb import** command to import metric data in batches.

- Prepare metric data, for example, the **importData.txt** file that contains the following content.

```
sys.cpu.user 1356998400 41 host=web01 cpu=0
sys.cpu.user 1356998401 42 host=web01 cpu=0
sys.cpu.user 1356998402 44 host=web01 cpu=0
sys.cpu.user 1356998403 47 host=web01 cpu=0
sys.cpu.user 1356998404 42 host=web01 cpu=0
sys.cpu.user 1356998405 42 host=web01 cpu=0
```

- Run the **tsdb import importData.txt** command to import metric data.

```
Start run net.opentsdb.tools.TextImporter, args: importData.txt
2019-06-26 15:45:22,091 INFO  [main] TextImporter: reading from file:importData.txt
2019-06-26 15:45:22,102 INFO  [main] TextImporter: Processed importData.txt in 11 ms, 6 data points (545.5 points/s)
2019-06-26 15:45:22,102 INFO  [main] TextImporter: Total: imported 6 data points in 0.012s (504.0 points/s)
```

Scanning the OpenTSDB Metric Data

You can run the **tsdb query** command to query the imported metric data in batches, for example, **tsdb query 0 1h-ago sum sys.cpu.user host=web01**.

```
Start run net.opentsdb.tools.CliQuery, args: 0 1h-ago sum sys.cpu.user host=web01
sys.cpu.user 1356998400000 41 {host=web01, cpu=0}
sys.cpu.user 1356998401000 42 {host=web01, cpu=0}
sys.cpu.user 1356998402000 44 {host=web01, cpu=0}
sys.cpu.user 1356998403000 47 {host=web01, cpu=0}
sys.cpu.user 1356998404000 42 {host=web01, cpu=0}
sys.cpu.user 1356998405000 42 {host=web01, cpu=0}
```

Deleting the Imported OpenTSDB Metric

You can run the **tsdb uid delete** command to delete the imported metric and its value, for example, **tsdb uid delete metrics sys.cpu.user**.

```
Start run net.opentsdb.tools.UidManager, args: delete metrics sys.cpu.user
```

12.5.2 OpenTSDB HTTP APIs

OpenTSDB provides HTTP-based or HTTPS-based APIs. A request method is to send a standard HTTP request containing the GET and POST methods to a path of a resource. The API is the same as that of the open source OpenTSDB. For details, visit **https://opentsdb.net/docs/build/html/api_http/index.html**.

The request and response entity type is application/JSON.

The code of the request and response entity is ISO-8859-1.

NOTE

- HTTP has security risks and HTTPS is a secure protocol. You are advised to use HTTPS for connection.
- OpenTSDB provides HTTP-based RESTful APIs that are language-independent. Any language that supports HTTP requests can interconnect to OpenTSDB.

Using Java APIs to Perform Operations on OpenTSDB

OpenTSDB provides HTTP-based or HTTPS-based APIs. You can use Java APIs to call related APIs to operate data. For details, see chapter **Application Development**.

Running the curl Command to Operate OpenTSDB

- Write data. For example, to write data of a metric named **testdata**, whose timestamp is **1524900185**, value is **true**, tag is **key** and **value**, run the following command:

  ```
  curl -ki -X POST -d '{"metric":"testdata", "timestamp":1524900185, "value":"true", "tags":{"key":"value"}}' https://<tsd_ip>:4242/api/put?sync
  ```

  <tsd_ip>: indicates the IP address of the TSD instance of OpenTSDB to which data is to be written.

  ```
  HTTP/1.1 204 No Content
  Content-Type: application/json; charset=UTF-8
  Content-Length: 0
  ```

- Query data. For example, to query summary information about the **testdata** metric in the past three years, run the following command:

  ```
  curl -ks https://<tsd_ip>:4242/api/query?start=3y-ago\&m=sum:testdata | python -m json.tool
  ```

  - **<tsd_ip>**: indicates the IP address or host name of the TSD instance of OpenTSDB that needs to be accessed.
  - **<start=3y-ago\&m=sum:testdata>**: Translates the **&** symbol, which may not be identified in the request.
  - (Optional) **<python -m json.tool>**: Converts the response request to the JSON format.

  ```
  [
    {
      "aggregateTags": [],
      "dps": {
        "1524900185": 1
      },
      "metric": "testdata",
      "tags": {
        "key": "value"
      }
    }
  ]
  ```

- Query **tsd** status. For example, to query information about the client connected to HBase, run the following command:

  ```
  curl -ks https://<tsd_ip>:4242/api/stats/region_clients | python -m json.tool
  ```

  <tsd_ip>: indicates the IP address of the TSD instance of OpenTSDB that needs to be accessed.

  ```
  [
    {
      "dead": false,
      "endpoint": "/192.168.2.187:16020",
      "inflightBreached": 0,
      "pendingBatchedRPCs": 0,
      "pendingBreached": 0,
      "pendingRPCs": 0,
      "rpcResponsesTimedout": 0,
      "rpcResponsesUnknown": 0,
      "rpcid": 78,
      "rpcsInFlight": 0,
      "rpcsSent": 79,
      "rpcsTimedout": 0,
      "writesBlocked": 0
    }
  ]
  ```


13 Presto Development Guide

13.1 Presto Application Development Overview

13.1.1 Introduction to Presto Application Development

Introduction to Presto

Presto is an open source distributed SQL query engine for running interactive analytic queries against data sources of all sizes ranging from gigabytes to petabytes.

Presto has the following characteristics:

- Multiple data sources: Presto supports multiple types of connectors, such as MySQL, Hive, and JMX.
- Support for SQL: Presto supports the ANSI SQL. You can directly run the SQL shell command for query.
- Hybrid computing: You can query multiple catalogs using JOIN.

13.1.2 Common Concepts of Presto Application Development

- **Connector**

  A connector adapts Presto to a data source such as Hive or a relational database.
- **Catalog**

  A Presto catalog contains schemas and references a data source via a connector.
- **Schema**

  Schemas are a way to organize tables.

13.1.3 Presto Application Development Process

Figure 13-1 and **Table 13-1** describe the phases in the development process.

Figure 13-1 Presto application development process

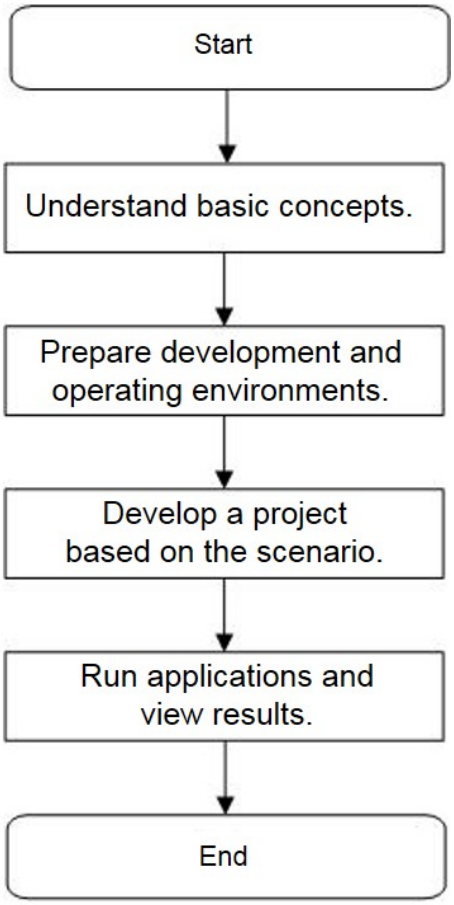


Table 13-1 Description of the Presto application development process

Phase	Description	Reference
Understand basic concepts.	Before application development, learn basic concepts of Presto.	**Common Concepts of Presto Application Development**
Prepare development and operating environments.	Presto applications can be developed in Java. Use the Eclipse tool to configure the development environment according to the guide.	**Presto Application Development Environment**
Develop a project based on the scenario.	Presto provides a Java sample project and a sample project of data query.	**Presto Development Plan**

Phase	Description	Reference
Run applications and view results.	This phase provides guidance for users to submit a developed application for running and view the result.	**Running the JDBC Client and Viewing Results**

13.2 Preparing a Presto Application Development Environment

13.2.1 Presto Application Development Environment

Table 13-2 describes the local environment required for application development. You also need to prepare a Linux environment for verifying whether the application is running properly.

Table 13-2 Development environment

Item	Description
OS	• Development environment: Windows 7 or later version is recommended. • Operating environment: Linux system

Item	Description
JDK installation	Basic configurations of the development and operating environments. The version requirements are as follows: The server and client of an MRS cluster support only built-in Oracle JDK 1.8, which cannot be replaced. If users' applications need to reference the JAR files of the SDK class in the user application processes, Oracle JDK and IBM JDK are supported. • Oracle JDK versions: 1.7 and 1.8 • IBM JDK versions: 1.7.8.10, 1.7.9.40, and 1.8.3.0 **NOTE** For security purpose, MRS cluster servers support only TLS 1.1 and TLS 1.2 encryption protocols in the Presto development environment. IBM JDK supports only TLS 1.0 by default. If you use IBM JDK, set **com.ibm.jsse2.overrideDefaultTLS** to **true**. After the parameter setting, TLS1.0/1.1/1.2 can be supported at the same time. For details, visit **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**.
Eclipse installation and configuration	Tool used for developing Presto applications. The version requirements are as follows: • The JDK version is 1.7, and the Eclipse version is 3.7.1 or later. • The JDK version is 1.8, and the Eclipse version is 4.3.2 or later. Note: • If you use IBM JDK, ensure that the JDK configured in Eclipse is IBM JDK. • If you use Oracle JDK, ensure that the JDK configured in Eclipse is Oracle JDK. • Do not use the same workspace and the sample project in the same path for different Eclipse programs.

Item	Description
Network	The client must be interconnected with the Presto server on the network.

13.2.2 Preparing a Presto Application Development Environment

- Install Eclipse and JDK in the Windows development environment.

  The recommended JDK version is 1.8, and the Eclipse version is 4.3.2 or later.

  NOTE

  - If you use IBM JDK, ensure that the JDK configured in Eclipse is IBM JDK.
  - If you use Oracle JDK, ensure that the JDK configured in Eclipse is Oracle JDK.
  - If you use ODBC for secondary development, ensure that JDK 1.8 or later is used.
  - Do not use the same workspace and the sample project in the same path for different Eclipse programs.
- Prepare a Linux environment for testing application running status.

Preparing a Running and Commissioning Environment

Step 1 On the ECS management console, apply for a new ECS for user application development, running, and commissioning.

- The security group of the ECS must be the same as that of the master node in an MRS cluster.
- The ECS and the MRS cluster must be in the same VPC.
- The ECS NIC and the MRS cluster must be in the same network segment.

Step 2 On the EIP page, apply for an EIP and bind it to the ECS. For details, see **Assigning an EIP and Binding It to an ECS**.

Step 3 Configure an inbound or outbound rule for the security group. For details, see **Configuring Security Group Rules**.

Step 4 Download a client program.

1. Log in to **MRS Manager**.
2. Choose **Services** > **Download Client** to download the complete client to the remote host, that is, download the client program to the newly applied ECS.

Step 5 Install a cluster client as user **root**.

1. Run the following command to decompress the client package:

   tar -xvf /opt/MRS_Services_Client.tar
2. Run the following command to verify the installation file package:

   sha256sum -c /opt/MRS_Services_ClientConfig.tar.sha256

   MRS_Services_ClientConfig.tar:OK
3. Run the following command to decompress the installation file package:

   tar -xvf /opt/MRS_Services_ClientConfig.tar

4. Run the following command to install the client to a specified directory (absolute path), for example, **/opt/client**. The directory is automatically created.

   cd /opt/MRS_Services_ClientConfig

   sh install.sh /opt/client

   ```
   Components client installation is complete.
   ```

----End

13.2.3 Preparing a Presto Application Development User

The development user is used to run the sample project. The user must have Presto permissions to run the Presto sample project. If Kerberos authentication is enabled for the MRS cluster, perform the following steps to prepare a development user. If Kerberos authentication is not enabled, skip the following steps.

Procedure

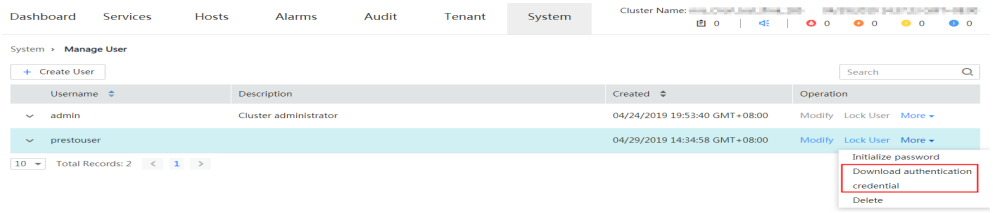
Step 1 Log in to **MRS Manager**.

Step 2 Choose **System** > **Manage User** > **Create User** to create a user for the sample project.

Step 3 Enter a username, for example, *prestouser*. Set **User Type** to **Machine-machine**, and select **presto** in **User Group**. Set **Primary Group** to **presto** and click **OK**.

Step 4 On MRS Manager, choose **System** > **Manage User**. In the **Operation** column corresponding to username *prestouser*, choose **More** > **Download authentication credential**, as shown in **Figure 13-2**. Save the file and decompress it to obtain the **user.keytab** and **krb5.conf** files. The two files are used for security authentication in the sample project.

Figure 13-2 Downloading the authentication credential



NOTE

If you modify component parameter configurations, you need to download the client configuration file again and update the client in the running and commissioning environment.

----End

13.2.4 Preparing a Presto JDBC Application Development Environment

To run the JDBC API sample code of Presto, you need to perform the following operations. The following example develops an application that uses JDBC to connect to Presto in the Windows environment.

Procedure

Step 1 Obtain the Presto sample project by referring to **Obtaining the MRS Application Development Sample Project**.

Step 2 In the root directory of the Presto sample project, run the **mvn install** command to perform compilation.

Step 3 In the root directory of the Presto sample project, run the **mvn eclipse:eclipse** command to create an Eclipse project.

Step 4 In the application development environment, import the sample project to the Eclipse development environment.

1. Choose **File** > **Import** > **General** > **Existing Projects into Workspace** > **Next** > **Browse**.

   The **Browse Folder** dialog box is displayed.

2. Select the **presto-examples** folder. On Windows, the folder path cannot contain any space.
3. Click **Finish**.

   After successful import, the **PrestoJDBCExample** class is the JDBC API sample code.

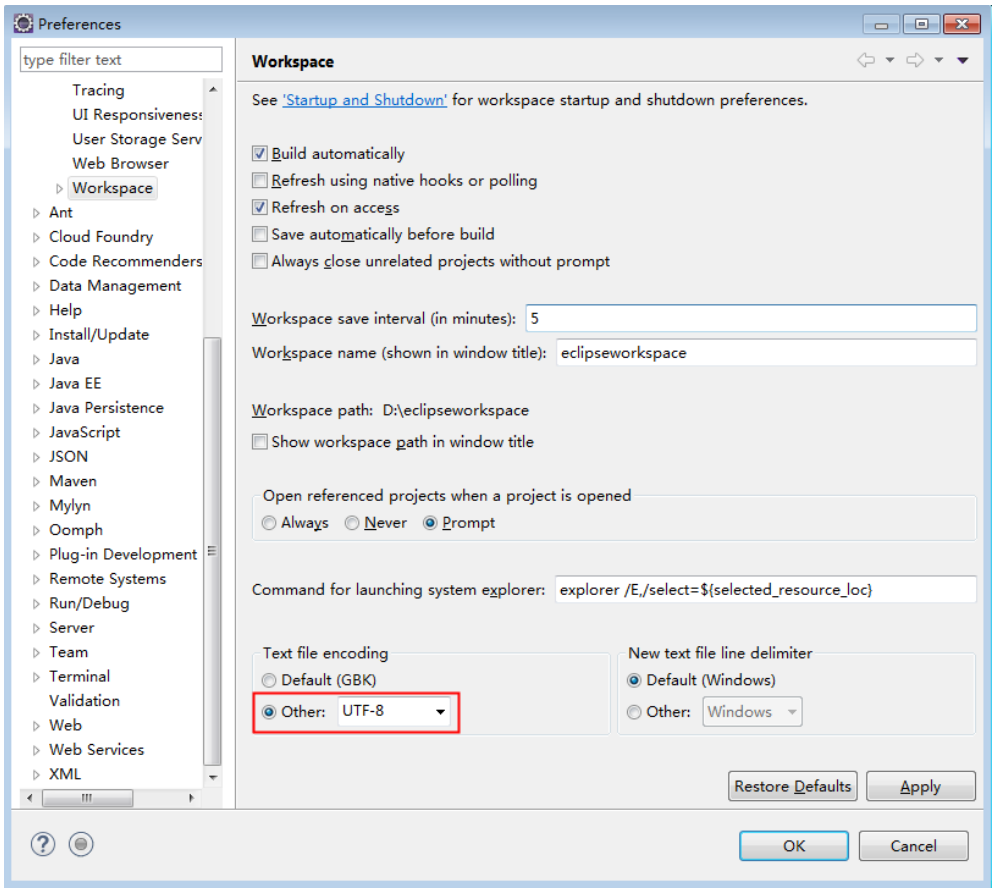
Step 5 Set an Eclipse text file encoding format to prevent garbled characters.

1. On the Eclipse menu bar, choose **Window** > **Preferences**.

   The **Preferences** window is displayed.

2. In the navigation pane on the left, choose **General** > **Workspace**. In the **Text file encoding** area, select **Other** and set the value to **UTF-8**. Click **Apply** and then **OK**. **Figure 13-3** shows the settings.

Figure 13-3 Setting the Eclipse encoding format



Step 6 Modify the sample. You can skip this step for a cluster with Kerberos authentication disabled.

After obtaining the **krb5.conf** and **user.keytab** files of the new development user in **Step 4**, modify **KerberosPrincipal** in **presto.properties** to the principal of the new user, **KerberosConfigPath** to the path of the **krb5.conf** file, and **KerberosKeytabPath** to the path of the **keytab** file.

----End

13.2.5 Preparing a Presto HCatalog Application Development Environment

To run the HCatalog API sample code of Presto, you need to perform the following operations. The following example develops an application that uses HCatalog to connect to Presto in the Windows environment.

Procedure

Step 1 Obtain the Presto sample project by referring to **Obtaining the MRS Application Development Sample Project**.

Step 2 In the root directory of the Presto sample project, run the **mvn install** command to perform compilation.

Step 3 In the root directory of the Presto sample project, run the **mvn eclipse:eclipse** command to create an Eclipse project.

Step 4 In the application development environment, import the sample project to the Eclipse development environment.

1. Choose **File** > **Import** > **General** > **Existing Projects into Workspace** > **Next** > **Browse**.

   The **Browse Folder** dialog box is displayed.

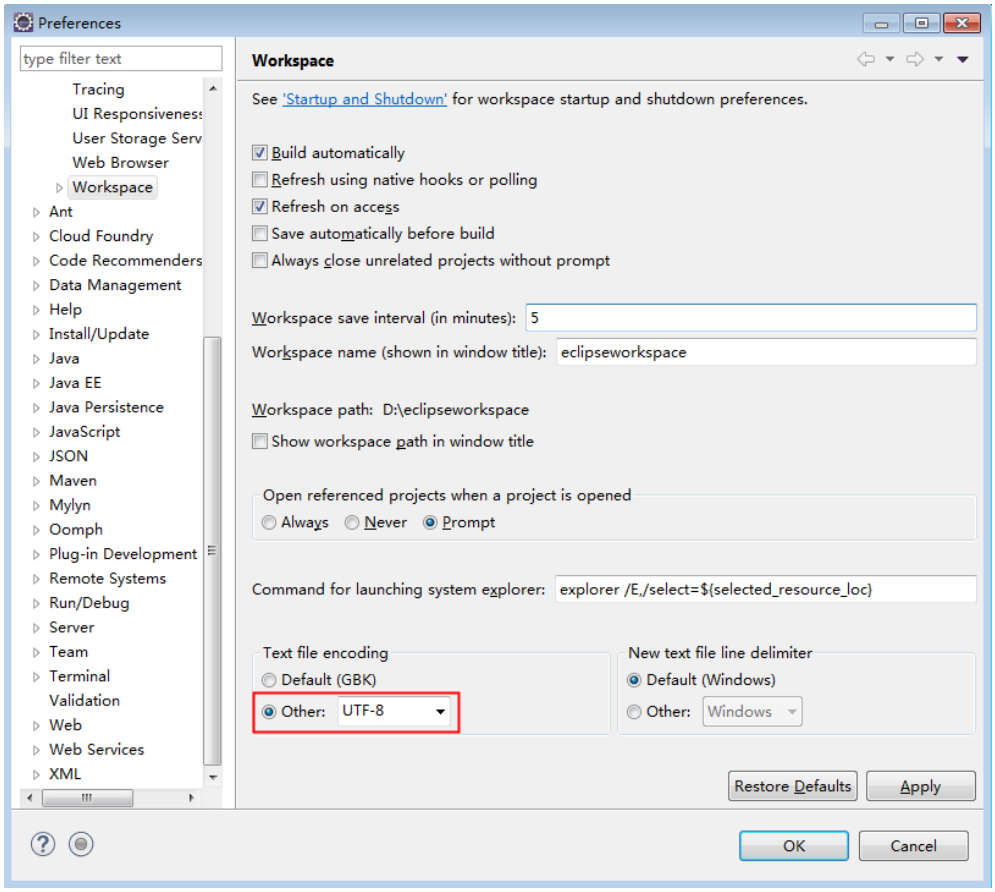
2. After downloading the project, select the **presto-examples** folder. On Windows, the folder path cannot contain any space.
3. Click **Finish**.

   After successful import, the **HCatalogExample** class in **com.huawei.bigdata.presto.example** is the HCatalog API sample code.

Step 5 Set an Eclipse text file encoding format to prevent garbled characters.

1. On the Eclipse menu bar, choose **Window** > **Preferences**. The **Preferences** window is displayed.
2. In the navigation pane on the left, choose **General** > **Workspace**. In the **Text file encoding** area, select **Other** and set the value to **UTF-8**. Click **Apply** and then **OK**. **Figure 13-4** shows the settings.

Figure 13-4 Setting the Eclipse encoding format



----End

13.3 Developing a Presto Application

13.3.1 Presto Development Plan

Scenario Description

Assume that you need to develop a Presto data analysis application to obtain the **call_center** table of TPCDS Catalog provided by Presto.

Development Guidelines

Step 1 Prepare data.

1. Create three tables: employee information table **employees_info**, employee contact table **employees_contact**, and extended employee information table **employees_info_extended**.
   - Employee information table **employees_info** contains fields such as employee ID, name, salary currency, salary, tax category, work place, and hire date. In salary currency, R indicates RMB and D indicates USD.
   - Fields in the **employees_contact** table include the employee ID, phone number, and email address.
   - Fields in the **employees_info_extended** table include the employee ID, name, mobile phone number, e-mail address, salary currency, salary, tax category, and work place. The partition field is the hire date.

     For details about table creation codes, see **Creating a Hive Table**.
2. Load employee information to **employees_info**.

   For details about data loading codes, see **Loading Hive Data**.

   Table 13-3 provides employee information.

Table 13-3 Employee information

Employee ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hire Date
1	Wang	R	8000.01	personal income tax&0.05	China:Shenzhen	2014
3	Tom	D	12000.02	personal income tax&0.09	America:NewYork	2014
4	Jack	D	24000.03	personal income tax&0.09	America:Manhattan	2014

Employee ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hire Date
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 13-4 provides employee contact information.

   Table 13-4 Employee contact information

Employee ID	Phone Number	e-mail
1	135 XXXX XXXX	xxxx@xx.com
3	159 XXXX XXXX	xxxxx@xx.com.cn
4	186 XXXX XXXX	xxxx@xx.org
6	189 XXXX XXXX	xxxx@xxx.cn
8	134 XXXX XXXX	xxxx@xxxx.cn

Step 2 Analyze data.

For details about data analysis codes, see **Querying Hive Data**.

- Query contact information of employees whose salaries are paid in USD.
- Query the IDs and names of employees who were hired in 2014, and load the query results to the partition with the hire date of 2014 in the **employees_info_extended** table.
- Collect the number of records in the **employees_info** table.
- Query information about employees whose email addresses end with "cn".

Step 3 Submit a data analysis task to collect the number of records in the **employees_info** table. For details, see **Analyzing Hive Data**.

----End

13.3.2 Presto JDBC Usage Example

Presto JDBC Usage Example

The following code snippet is in the **PrestoJDBCExample** class and is used to connect JDBC to Presto TPCDS Catalog.

```
private static Connection connection;
private static Statement statement;
```



```
/**
 * Only when Kerberos authentication enabled, configurations in presto-examples/conf/presto.properties
 * should be set. More details please refer to https://prestodb.io/docs/0.215/installation/jdbc.html.
*/
private static void initConnection(String url, boolean krbsEnabled) throws SQLException {
    if (krbsEnabled) {
        String filePath = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
        File proFile = new File(filePath + "presto.properties");if (proFile.exists()) {
            Properties props = new Properties();
            try {
                props.load(new FileInputStream(proFile));
            } catch (IOException e) {
                e.printStackTrace();
            }
            connection = DriverManager.getConnection(url, props);
        }
    } else {
          connection = DriverManager.getConnection(url, "presto", null);
    }
    statement = connection.createStatement();
}

private static void releaseConnection() throws SQLException {
    statement.close();
    connection.close();
}

public static void main(String[] args) throws SQLException {
    try {
        /**
         * Replace example_ip with your cluster presto server ip.
         * By default, Kerberos authentication disabled cluster presto service port is 7520, Kerberos
         * authentication enabled cluster presto service port is 7521
         * The postfix /tpcds/sf1 means to use tpcds catalog and sf1 schema, you can use hive catalog as well
         * If Kerberos authentication enabled, set the second param to true.
         * see PrestoJDBCExample#initConnection(java.lang.String, boolean).
         */
        initConnection("jdbc:presto://example_ip:7520/tpcds/sf1", false);
        //initConnection("jdbc:presto://example_ip:7521/tpcds/sf1", true);
        ResultSet resultSet = statement.executeQuery("select * from call_center");
        while (resultSet.next()) {
            System.out.println(resultSet.getString("cc_name") + " : " + resultSet.getString("cc_employees"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        releaseConnection();
    }
}
```

13.4 Commissioning a Presto Application

Running the JDBC Client and Viewing Results

Step 1 Run the **mvn clean compile assembly:single** command to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar**.

Step 2 Create a directory as the running directory in the running and commissioning environment, for example, **/opt/presto_examples** (Linux), and create the **conf** subdirectory in the directory.

Copy **presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar** exported in **Step 1** to **/opt/presto_examples**.

Step 3 To enable Kerberos authentication for a cluster, you need to copy the obtained **user.keytab** and **krb5.conf** files to the **/opt/presto_examples/conf** directory and modify the **presto.preperties** file in the **conf** directory of the sample code. Skip this step for a cluster with Kerberos authentication disabled.

Table 13-5 presto.preperties parameters

Parameter	Description
user	Username used for Kerberos authentication, that is, the username of the development user created in **Preparing a Presto Application Development User**.
KerberosPrincipal	Username used for authentication. That is, the username of the development user created in **Preparing a Presto Application Development User** needs to be certificated.
KerberosConfigPath	Path where **krb5.conf** is stored
KerberosKeytabPath	Path where **user.keytab** is stored

Example

```
user = prestouser
SSL = true
KerberosRemoteServiceName = HTTP
KerberosPrincipal = prestouser
KerberosConfigPath = /opt/presto_examples/conf/krb5.conf
KerberosKeytabPath = /opt/presto_examples/conf/user.keytab
```

Step 4 In Linux, run the sample program.

```
chmod +x /opt/presto_examples -R
cd /opt/presto_examples
java -jar presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
```

Step 5 In the CLI, view the query results of the example code.

If the following information is displayed, the sample project execution is successful in Linux.

```
NY Metro : 2
Mid Atlantic : 6
Mid Atlantic : 6
North Midwest : 1
North Midwest : 3
North Midwest : 7
```

----End

13.4.1 Commissioning a Presto Application on Windows

Step 1 Apply for a Windows ECS to access the MRS cluster to operate Presto. To apply for ECS to access the MRS cluster, perform the following steps:

1. On the **Active Clusters** page, click the name of an existing cluster.

   On the cluster details page, record the **AZ**, **VPC**, and **Default Security Group** of the Master node.

2. On the ECS management console, create a ECS.

   The **AZ**, **VPC**, and **security group** of ECS must be the same as those of the cluster to be accessed.

   Select a Windows public image.

   For details about other configuration parameters, see **Elastic Cloud Server** > **Quick Start** > **Purchasing and Logging In to a Windows ECS**.

Step 2 To enable Kerberos authentication for a cluster, you need to configure the mappings between cluster IP addresses and host names on Windows. Log in to the cluster background, run the **cat /etc/hosts** command, and copy the mapping between IP addresses and host names in the **hosts** file to **C:\Windows\System32\drivers\etc\hosts** of the ECS. Skip this step for a cluster with Kerberos authentication disabled.

Step 3 To operate Presto clusters on Windows, the JDK version must be **jdk1.8.0_60** or later. To enable Kerberos authentication for a cluster, you need to copy **/opt/Bigdata/om-0.0.1/packaged-distributables/client_packet/ca.crt** of the active node of the MRS cluster to the Windows ECS, open the CLI in the **jdk/bin** directory, run the following command, and modify the configuration of the **presto.properties** file in the **conf** directory in the sample code.

For jdk1.8.0_242 or later, delete **renew_lifetime = 0m from [libdefaults]** in **krb5.conf**.

Skip this step for a cluster with Kerberos authentication disabled.

keytool -import -v -trustcacerts -alias presto_trust -file <ca.crt_path> -keystore <keystore_path>/truststore.jks –keypass <password>

In the preceding command, **<ca.crt_path>** indicates the path of the copied **ca.crt** file, **<keystore_path>** indicates the path where the **truststore.jks** file is generated, and **<password>** indicates the truststore password. You can specify the password as needed. There can be security risks if a command contains the authentication password. You are advised to disable the command recording function (history) before running the command.

Table 13-6 Parameters in presto.properties

Parameter	Description
user	Username used for Kerberos authentication, that is, the username of the development user created in **Preparing a Presto Application Development User**.
KerberosPrincipal	Name used for authentication, that is, the name of the development user created in **Preparing a Presto Application Development User**.

Parameter	Description
KerberosConfigPath	Path where **krb5.conf** is stored Pay attention to escape character (\).
KerberosKeytabPath	Path where **user.keytab** is stored Pay attention to escape character (\).
SSLTrustStorePath	Path where **truststore.jks** is stored Pay attention to escape character (\).
SSLTrustStorePassword	Truststore password

Step 4 Modify and run the sample.

1. In the development environment (for example, Eclipse), modify the example_ip, port number, and krbsEnabled configurations in the sample code.
2. Right-click **PrestoJDBCExample.java**.
3. Click **Run as > Java Application** to run the corresponding application project.

Step 5 View the execution result. The following information is displayed if the execution is successful:

```
NY Metro : 2
Mid Atlantic : 6
Mid Atlantic : 6
North Midwest : 1
North Midwest : 3
North Midwest : 7
```

----End

13.4.2 Commissioning a Presto Application on Linux

Running the JDBC Client and Viewing Results

Step 1 Run the **mvn clean compile assembly:single** command to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar**.

Step 2 Create a directory as the running directory in the running and commissioning environment, for example, **/opt/presto_examples** (Linux), and create the **conf** subdirectory in the directory.

Copy **presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar** exported in **Step 1** to **/opt/presto_examples**.

Step 3 To enable Kerberos authentication for a cluster, you need to copy the obtained **user.keytab** and **krb5.conf** files in **Step 4** to the **/opt/presto_examples/conf** directory and modify the **presto.preperties** file in the **conf** directory of the sample code. Skip this step for a cluster with Kerberos authentication disabled.

Table 13-7 presto.preperties parameters

Parameter	Description
user	Username used for Kerberos authentication, that is, the username of the development user created in **Preparing a Presto Application Development User**.
KerberosPrincipal	Username used for authentication. That is, the username of the development user created in **Preparing a Presto Application Development User** needs to be certificated.
KerberosConfigPath	Path where **krb5.conf** is stored
KerberosKeytabPath	Path where **user.keytab** is stored

Example

```
user = prestouser
SSL = true
KerberosRemoteServiceName = HTTP
KerberosPrincipal = prestouser
KerberosConfigPath = /opt/presto_examples/conf/krb5.conf
KerberosKeytabPath = /opt/presto_examples/conf/user.keytab
```

Step 4 In Linux, run the sample program.

```
chmod +x /opt/presto_examples -R
cd /opt/presto_examples
java -jar presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
```

Step 5 In the CLI, view the query results of the sample code.

The following information is displayed if the sample project is successful in Linux:

```
NY Metro : 2
Mid Atlantic : 6
Mid Atlantic : 6
North Midwest : 1
North Midwest : 3
North Midwest : 7
```

----End

13.5 FAQs About Presto Application Development

13.5.1 Presto APIs

Presto JDBC APIs comply with the Java JDBC driver standard. For details, see JDK 1.7 API. For details about how to use the Presto JDBC, see **https://prestodb.io/docs/current/installation/jdbc.html**.

13.5.2 No Certificate Is Available When PrestoJDBCExample Run on a Node Outside the Cluster

Question

The **presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar** file is running properly on nodes in the cluster. However, no certificate is available when PrestoJDBCExample runs on a node outside the cluster to connect to the cluster with Kerberos authentication enabled, the following error message is displayed:

```
java.sql.SQLException: Error executing query
        at
com.facebook.presto.jdbc.PrestoStatement.internalExecute(PrestoStatement.java:274)
        at com.facebook.presto.jdbc.PrestoStatement.execute(PrestoStatement.java:227)
        at
com.facebook.presto.jdbc.PrestoStatement.executeQuery(PrestoStatement.java:76)
        at
PrestoJDBCExample.main(PrestoJDBCExample.java:65)
Caused by: java.io.UncheckedIOException:
javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException:
PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to find
valid certification path to requested target
        at
com.facebook.presto.jdbc.internal.client.JsonResponse.execute(JsonResponse.java:154)
        at
com.facebook.presto.jdbc.internal.client.StatementClientV1.<init>(StatementClientV1.java:129)
        at
com.facebook.presto.jdbc.internal.client.StatementClientFactory.newStatementClient(StatementClientFactory.
java:24)
        at
com.facebook.presto.jdbc.QueryExecutor.startQuery(QueryExecutor.java:46)
        at
com.facebook.presto.jdbc.PrestoConnection.startQuery(PrestoConnection.java:683)
        at
com.facebook.presto.jdbc.PrestoStatement.internalExecute(PrestoStatement.java:239)
        ... 3 more
Caused by: javax.net.ssl.SSLHandshakeException:
sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to find
valid certification path to requested target
        at
sun.security.ssl.Alerts.getSSLException(Alerts.java:192)
        at
sun.security.ssl.SSLSocketImpl.fatal(SSLSocketImpl.java:1959)
        at
sun.security.ssl.Handshaker.fatalSE(Handshaker.java:302)
        at
sun.security.ssl.Handshaker.fatalSE(Handshaker.java:296)
        at
sun.security.ssl.ClientHandshaker.serverCertificate(ClientHandshaker.java:1514)
        at
sun.security.ssl.ClientHandshaker.processMessage(ClientHandshaker.java:216)
        at
sun.security.ssl.Handshaker.processLoop(Handshaker.java:1026)
        at
sun.security.ssl.Handshaker.process_record(Handshaker.java:961)
        at
sun.security.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:1072)
        at
sun.security.ssl.SSLSocketImpl.performInitialHandshake(SSLSocketImpl.java:1385)
        at
sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:1413)
        at
sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:1397)
        at
```



```
com.facebook.presto.jdbc.internal.okhttp3.internal.connection.RealConnection.connectTls(RealConnection.java:318)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.connection.RealConnection.establishProtocol(RealConnection.java:282)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.connection.RealConnection.connect(RealConnection.java:167)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.connection.StreamAllocation.findConnection(StreamAllocation.java:257)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.connection.StreamAllocation.findHealthyConnection(StreamAllocation.java:135)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.connection.StreamAllocation.newStream(StreamAllocation.java:114)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.connection.ConnectInterceptor.intercept(ConnectInterceptor.java:42)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.cache.CacheInterceptor.intercept(CacheInterceptor.java:93)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.http.BridgeInterceptor.intercept(BridgeInterceptor.java:93)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.http.RetryAndFollowUpInterceptor.intercept(RetryAndFollowUpInterceptor.java:126)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
	at com.facebook.presto.jdbc.internal.client.SpnegoHandler.intercept(SpnegoHandler.java:109)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
	at com.facebook.presto.jdbc.internal.client.OkHttpUtil.lambda$userAgent$0(OkHttpUtil.java:77)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
	at com.facebook.presto.jdbc.internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
	at com.facebook.presto.jdbc.internal.okhttp3.RealCall.getResponseWithInterceptorChain(RealCall.java:200)
	at com.facebook.presto.jdbc.internal.okhttp3.RealCall.execute(RealCall.java:77)
	at
```



```
com.facebook.presto.jdbc.internal.client.JsonResponse.execute(JsonResponse.java:131)
        ... 8 more
Caused by: sun.security.validator.ValidatorException: PKIX
path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to find
valid certification path to requested target
        at
sun.security.validator.PKIXValidator.doBuild(PKIXValidator.java:397)
        at
sun.security.validator.PKIXValidator.engineValidate(PKIXValidator.java:302)
        at
sun.security.validator.Validator.validate(Validator.java:260)
        at
sun.security.ssl.X509TrustManagerImpl.validate(X509TrustManagerImpl.java:324)
        at
sun.security.ssl.X509TrustManagerImpl.checkTrusted(X509TrustManagerImpl.java:229)
        at
sun.security.ssl.X509TrustManagerImpl.checkServerTrusted(X509TrustManagerImpl.java:124)
        at
sun.security.ssl.ClientHandshaker.serverCertificate(ClientHandshaker.java:1496)
        ... 41 more
Caused by: sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target
        at
sun.security.provider.certpath.SunCertPathBuilder.build(SunCertPathBuilder.java:141)
        at
sun.security.provider.certpath.SunCertPathBuilder.engineBuild(SunCertPathBuilder.java:126)
        at
java.security.cert.CertPathBuilder.build(CertPathBuilder.java:280)
        at
sun.security.validator.PKIXValidator.doBuild(PKIXValidator.java:392)
        ... 47 more
```

Answer

When the HTTPS protocol is used to connect to the security cluster, the server certificate is not authenticated. As a result, the connection fails.

You can replace the cacerts file in the **java jdk** directory on the current node with the cacerts file (for example, **/opt/Bigdata/jdk1.8.0_232/jre/lib/security/cacerts**) in the **java jdk** directory on a node in the cluster.

13.5.3 When a Node Outside a Cluster Is Connected to a Cluster with Kerberos Authentication Enabled, HTTP Cannot Find the Corresponding Record in the Kerberos Database

Question

The **presto-examples-1.0-SNAPSHOT-jar-with-dependencies.jar** file is running properly on nodes in the cluster. However, when PrestoJDBCExample running on a node outside the cluster connect to the cluster with Kerberos authentication enabled, the following error messages is displayed:

Error 1:

```
java.sql.SQLException:
Kerberos error for [HTTP@10.33.11.138]: No valid credentials provided
(Mechanism level: No valid credentials provided (Mechanism level: Server not
found in Kerberos database (7) - UNKNOWN_SERVER))
at
io.prestosql.jdbc.PrestoStatement.internalExecute(PrestoStatement.java:281)
at
```



```
io.prestosql.jdbc.PrestoStatement.execute(PrestoStatement.java:229)
at
io.prestosql.jdbc.PrestoStatement.executeQuery(PrestoStatement.java:78)
at PrestoJDBCExample.main(PrestoJDBCExample.java:68)
Caused by:
io.prestosql.jdbc.$internal.client.ClientException: Kerberos error for
[HTTP@10.33.11.138]: No valid credentials provided (Mechanism level: No valid
credentials provided (Mechanism level: Server not found in Kerberos database
(7) - UNKNOWN_SERVER))
at
io.prestosql.jdbc.$internal.client.SpnegoHandler.generateToken(SpnegoHandler.java:174)
at
io.prestosql.jdbc.$internal.client.SpnegoHandler.authenticate(SpnegoHandler.java:140)
at
io.prestosql.jdbc.$internal.client.SpnegoHandler.authenticate(SpnegoHandler.java:128)
at
io.prestosql.jdbc.$internal.okhttp3.internal.http.RetryAndFollowUpInterceptor.followUpRequest(RetryAndFollo
wUpInterceptor.java:289)
at
io.prestosql.jdbc.$internal.okhttp3.internal.http.RetryAndFollowUpInterceptor.intercept(RetryAndFollowUpInt
erceptor.java:157)
at
io.prestosql.jdbc.$internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
at
io.prestosql.jdbc.$internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
at
io.prestosql.jdbc.$internal.client.SpnegoHandler.intercept(SpnegoHandler.java:115)
at
io.prestosql.jdbc.$internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
at
io.prestosql.jdbc.$internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
at
io.prestosql.jdbc.$internal.client.OkHttpUtil.lambda$userAgent$0(OkHttpUtil.java:64)
at
io.prestosql.jdbc.$internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147)
at
io.prestosql.jdbc.$internal.okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121)
at
io.prestosql.jdbc.$internal.okhttp3.RealCall.getResponseWithInterceptorChain(RealCall.java:200)
at
io.prestosql.jdbc.$internal.okhttp3.RealCall.execute(RealCall.java:77)
at
io.prestosql.jdbc.$internal.client.JsonResponse.execute(JsonResponse.java:131)
at
io.prestosql.jdbc.$internal.client.StatementClientV1.<init>(StatementClientV1.java:132)
at
io.prestosql.jdbc.$internal.client.StatementClientFactory.newStatementClient(StatementClientFactory.java:24)
at
io.prestosql.jdbc.QueryExecutor.startQuery(QueryExecutor.java:46)
at io.prestosql.jdbc.PrestoConnection.startQuery(PrestoConnection.java:714)
at
io.prestosql.jdbc.PrestoStatement.internalExecute(PrestoStatement.java:241)
... 3 more
Caused by: GSSException:
No valid credentials provided (Mechanism level: No valid credentials provided
(Mechanism level: Server not found in Kerberos database (7) - UNKNOWN_SERVER))
at
sun.security.jgss.spnego.SpNegoContext.initSecContext(SpNegoContext.java:454)
at
sun.security.jgss.GSSContextImpl.initSecContext(GSSContextImpl.java:248)
at sun.security.jgss.GSSContextImpl.initSecContext(GSSContextImpl.java:179)
at
io.prestosql.jdbc.$internal.client.SpnegoHandler.generateToken(SpnegoHandler.java:167)
... 23 more
Caused by: GSSException:
No valid credentials provided (Mechanism level: Server not found in Kerberos database
(7) - UNKNOWN_SERVER)
at
sun.security.jgss.krb5.Krb5Context.initSecContext(Krb5Context.java:772)
```



```
at
sun.security.jgss.GSSContextImpl.initSecContext(GSSContextImpl.java:248)
at
sun.security.jgss.GSSContextImpl.initSecContext(GSSContextImpl.java:179)
at
sun.security.jgss.spnego.SpNegoContext.GSS_initSecContext(SpNegoContext.java:882)
at
sun.security.jgss.spnego.SpNegoContext.initSecContext(SpNegoContext.java:317)
... 26 more
Caused by: KrbException:
Server not found in Kerberos database (7) - UNKNOWN_SERVER
at
sun.security.krb5.KrbTgsRep.<init>(KrbTgsRep.java:73)
at
sun.security.krb5.KrbTgsReq.getReply(KrbTgsReq.java:251)
at
sun.security.krb5.KrbTgsReq.sendAndGetCreds(KrbTgsReq.java:262)
at
sun.security.krb5.internal.CredentialsUtil.serviceCreds(CredentialsUtil.java:308)
at
sun.security.krb5.internal.CredentialsUtil.acquireServiceCreds(CredentialsUtil.java:126)
at
sun.security.krb5.Credentials.acquireServiceCreds(Credentials.java:466)
at sun.security.jgss.krb5.Krb5Context.initSecContext(Krb5Context.java:695)
... 30 more
Caused by: KrbException:
Identifier doesn't match expected value (906)
at
sun.security.krb5.internal.KDCRep.init(KDCRep.java:140)
at
sun.security.krb5.internal.TGSRep.init(TGSRep.java:65)
at
sun.security.krb5.internal.TGSRep.<init>(TGSRep.java:60)
at
sun.security.krb5.KrbTgsRep.<init>(KrbTgsRep.java:55)
... 36 more
```

Error 2:

```
java.sql.SQLException:
Authentication failed: Authentication failed for token:
...
        at
com.facebook.presto.jdbc.PrestoStatement.internalExecute(PrestoStatement.java:271)
        at
com.facebook.presto.jdbc.PrestoStatement.execute(PrestoStatement.java:227)
        at
com.facebook.presto.jdbc.PrestoStatement.executeQuery(PrestoStatement.java:76)
        at
PrestoJDBCExample.main(PrestoJDBCExample.java:65)
Caused by:
com.facebook.presto.jdbc.internal.client.ClientException: Authentication failed:
Authentication failed for token:
...
        at
com.facebook.presto.jdbc.internal.client.StatementClientV1.requestFailedException(StatementClientV1.java:4
32)
        at
com.facebook.presto.jdbc.internal.client.StatementClientV1.<init>(StatementClientV1.java:132)
        at
com.facebook.presto.jdbc.internal.client.StatementClientFactory.newStatementClient(StatementClientFactory.
java:24)
        at
com.facebook.presto.jdbc.QueryExecutor.startQuery(QueryExecutor.java:46)
        at
com.facebook.presto.jdbc.PrestoConnection.startQuery(PrestoConnection.java:683)
        at
com.facebook.presto.jdbc.PrestoStatement.internalExecute(PrestoStatement.java:239)
        ... 3 more
```


Answer

The principal of HTTP concatenated by the client is inconsistent with that in the Kerberos database (Error 1) or the obtained token cannot be connected to Presto.

Run the **cat /etc/hosts** command in the cluster to add the IP address and host name of the Presto coordinator to the **/etc/hosts** file of the current node.

14 Spark Development Guide

14.1 Spark Application Development Overview

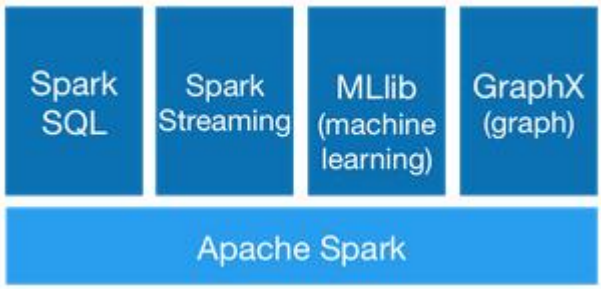
14.1.1 Introduction to Spark Application Development

Spark

Spark is a distributed batch processing framework. It provides analysis and mining and iterative memory computing capabilities and supports application development in multiple programming languages, including Scala, Java, and Python. Spark applies to the following scenarios:

- Data processing: Spark can process data quickly and has fault tolerance and scalability.
- Iterative computation: Spark supports iterative computation to meet the requirements of multi-step data processing logic.
- Data mining: 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 streaming processing with delay in seconds and supports multiple external data sources.
- Query analysis: Spark supports standard SQL query analysis, provides the DSL (DataFrame), and supports multiple external inputs.
- **Figure 14-1** shows the component architecture of Apache Spark. This section provides guidance to application development of Spark, Spark SQL and Spark Streaming. For details about MLlib and GraghX, visit the Spark official website at **http://spark.apache.org/docs/2.2.2/**.

Figure 14-1 Spark architecture



Spark APIs

Spark supports application development in multiple programming languages, including Scala, Java, and Python. Since Spark is developed in Scala and Scala is easy to read, you are advised to develop Spark applications in Scala.

Table 14-1 describes Spark APIs in different languages.

Table 14-1 Spark APIs

API	Description
Scala API	Indicates the API in Scala. Since Scala is easy to read, you are advised to use Scala APIs to develop applications.
Java API	Indicates the API in Java.
Python API	Indicates the API in Python.

Divided by different modes, Spark Core and Spark Streaming use APIs listed in the preceding table to develop applications. Spark SQL can be accessed through CLI and ThriftServer. There are two ways to access the ThriftServer: Beeline and the JDBC client code.

NOTE

For the **spark-sql**, **spark-shell**, and **spark-submit** scripts (running applications contain SQL operations), do not use the proxy user parameter to submit a task.

14.1.2 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 creation

  – An RDD can be created from the input of HDFS or other storage systems that are compatible with Hadoop.

  - A new RDD can be converted from a parent RDD.
  - An RDD can be converted from a collection of data sets through encoding.

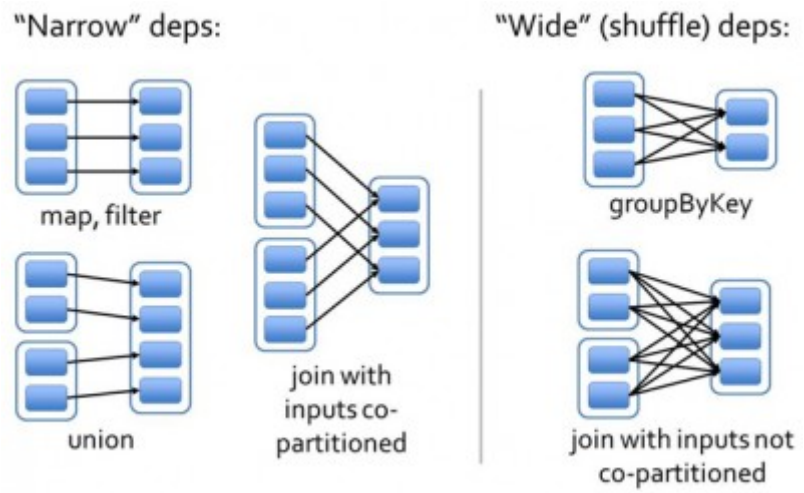
  RDD storage

  - Users can select different storage levels to store an RDD for reuse. (There are 11 storage levels to store an RDD.)
  - By default, the RDD is stored in the memory. When the memory is insufficient, the RDD overflows to the disk.

- **RDD dependency**

  The RDD dependency includes the narrow dependency and wide dependency.

  Figure 14-2 RDD dependency



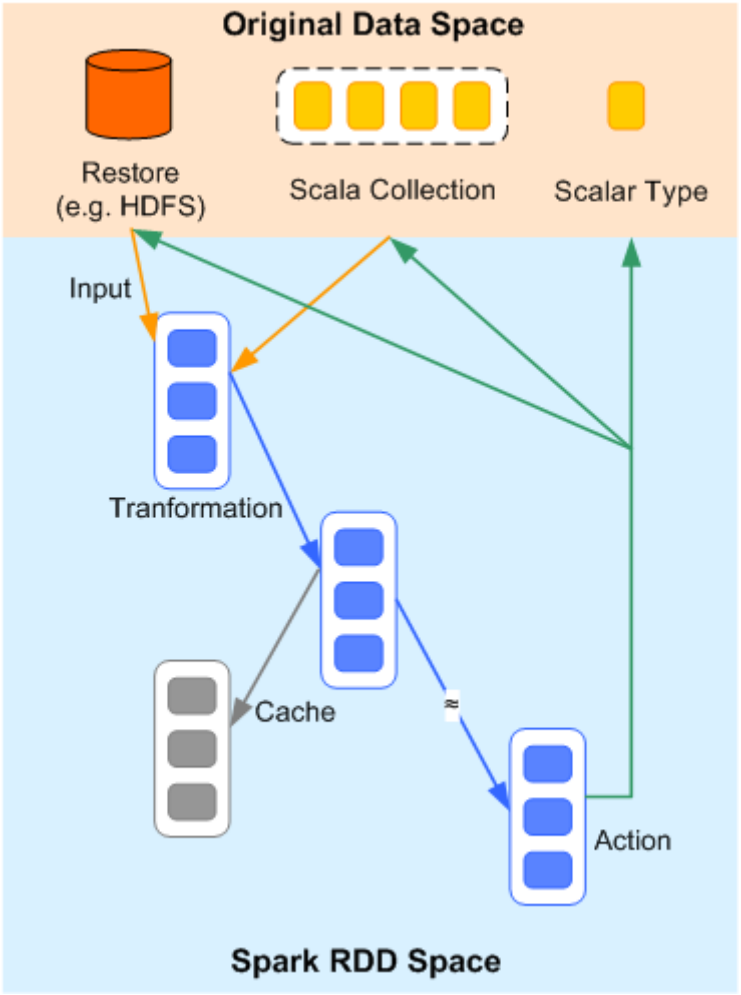
  - **Narrow dependency**: Each partition of the parent RDD is used by at most one partition of the child RDD.
  - **Wide dependency**: Partitions of the child RDD depend on all partitions of the parent RDD.

  The narrow dependency facilitates the optimization. Logically, each RDD operator is a fork/join (the join is the barrier used to synchronize multiple concurrent tasks); 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 the next RDD operator. It is uneconomical to directly translate the RDD into physical implementation. The first is that every RDD (even intermediate result) needs to be physicalized into memory or storage, which is time-consuming and occupies much space. The second is that as a global barrier, the join operation is very expensive and the entire join process will be slowed down by the slowest node. 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 implement classic fusion optimization. If the relationship in the continuous operator sequence is narrow dependency, multiple fork/join processes can be combined to reduce a large number of global barriers and eliminate the physicalization of many RDD intermediate results, which greatly improves the performance. This is called pipeline optimization in Spark.

- **Transformation and action (RDD operations)**

  Operations on RDD include transformation (the return value is an RDD) and action (the return value is not an RDD). **Figure 14-3** shows the RDD operation 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 14-3 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 has the one-to-one relationship, and the partition structure of the result RDD remains unchanged, for example, map.

  The input and output has 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 after map and then flattens to multiple elements).

  The input and output has 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 the key, such as join and cogroup.

  NOTE

  The later three operations involve sorting and are called shuffle operations.

Action includes the following types:

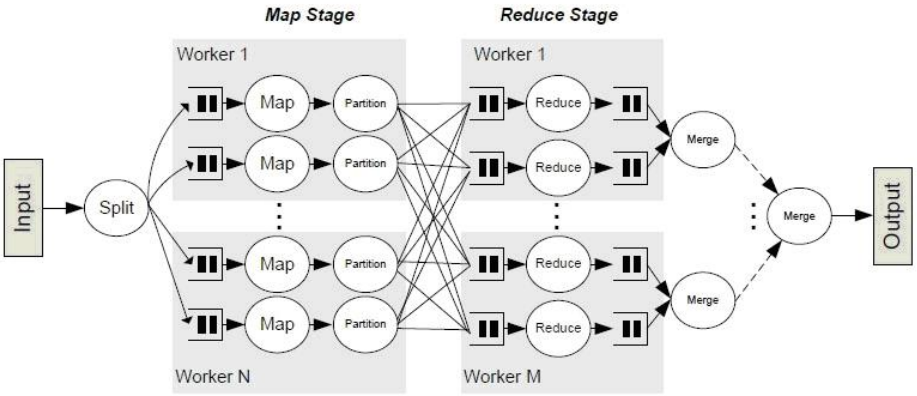
- Generate scalar configuration items, such as count (the number of elements in the returned RDD), reduce, fold/aggregate (the number of scalar configuration items that are returned), and take (the number of elements before the return).
- Generate the Scala collection, such as collect (import all elements 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, the output results must be hashed based on a key and distributed to each 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 shows the entire process of the MapReduce algorithm.

  Figure 14-4 Algorithm process

  Shuffle is a bridge to connect 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 more ShuffleMapTasks, and the last stage contains one or more ResultTasks.

- **Spark application structure**

  The Spark application structure includes the initialized SparkContext and the main program.

  – Initialized SparkContext: constructs the operating environment of the Spark application.

    Constructs the SparkContext object, for example,

    ```
    new SparkContext(master, appName, [SparkHome], [jars])
    ```

    Parameter description:

    master: indicates the link string. The link modes include local, Yarn-cluster, and Yarn-client.

    appName: indicates the application name.

    SparkHome: indicates the directory where Spark is installed in the cluster.

    jars: indicates the code and dependency package of the application.

  – Main program: processes data.

  For details about how to submit an application, see **https://spark.apache.org/docs/2.2.2/submitting-applications.html**.

- **Spark shell commands**

The basic Spark shell command supports the submission of the Spark application. The Spark shell command is as follows:

```
./bin/spark-submit \
  --class <main-class> \
  --master <master-url> \
  ... # other options
  <application-jar> \
  [application-arguments]
```

Parameter description:

--class: indicates the name of the class of the Spark application.

--master: indicates the master to which the Spark application links, such as Yarn-client and Yarn-cluster.

application-jar: indicates the path of the JAR file of the Spark application.

application-arguments: indicates the parameter required to submit the Spark application. This parameter can be left blank.

- **Spark JobHistory Server**

  It 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.

Basic Concepts of the Spark SQL

DataFrame

The DataFrame is a structured and distributed dataset consisting of multiple columns. The DataFrame is equal to a table in the relationship database or the DataFrame in the R/Python. The DataFrame is the most basic concept in the Spark SQL, which can be created by using multiple methods, such as the structured dataset, Hive table, external database or RDD.

The program entry of the Spark SQL is the SQLContext class (or its subclasses). A SparkContext object is required as a construction parameter for the creation of the SQLContext. One subclass of the SQLContext is the HiveContext. Compared with its parent class, the functions of parser, UDF and reading inventory Hive data of the HiveQL are added in the HiveContext. However, the HiveContext does not rely on the running Hive but the class library of the Hive.

By using the SQLContext and its subclasses, the basic dataset DataFrame in the Spark SQL can be easily created. The DataFrame provides various APIs for Spark SQL and is compatible with various data sources, such as the Parquet, JSON, Hive data, Database, and HBase. These data sources can be read by using the same syntax.

Basic Concepts of the Spark Streaming

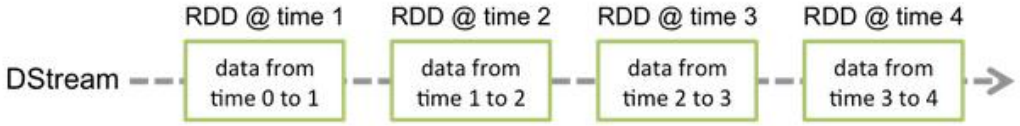
Dstream

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 input stream. In essence, a DStream is a series of continuous RDDs. The RDD is a read-only and partitionable distributed dataset.

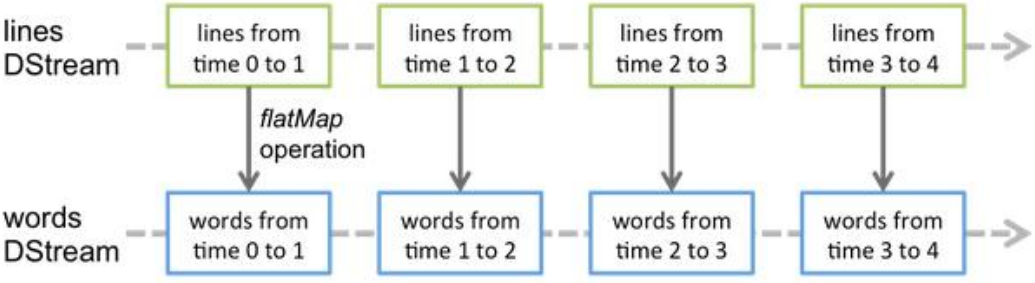
Each RDD in the DStream contains data in a range, as shown in **Figure 14-5**.

Figure 14-5 Relationship between DStream and RDD



All operators applied in the DStream are translated to the operations in the lower RDD, as shown in **Figure 14-6**. The transformation of the lower RDDs is calculated by using a Spark engine. Most operation details are concealed in the DStream operators and High-level APIs are provided for developers.

Figure 14-6 DStream operator translation



14.1.3 Spark Application Development Process

Spark includes Spark Core, Spark SQL, and Spark Streaming, whose development processes are the same.

Figure 14-7 and **Table 14-2** describe the phases in the development process.

Figure 14-7 Spark application development process

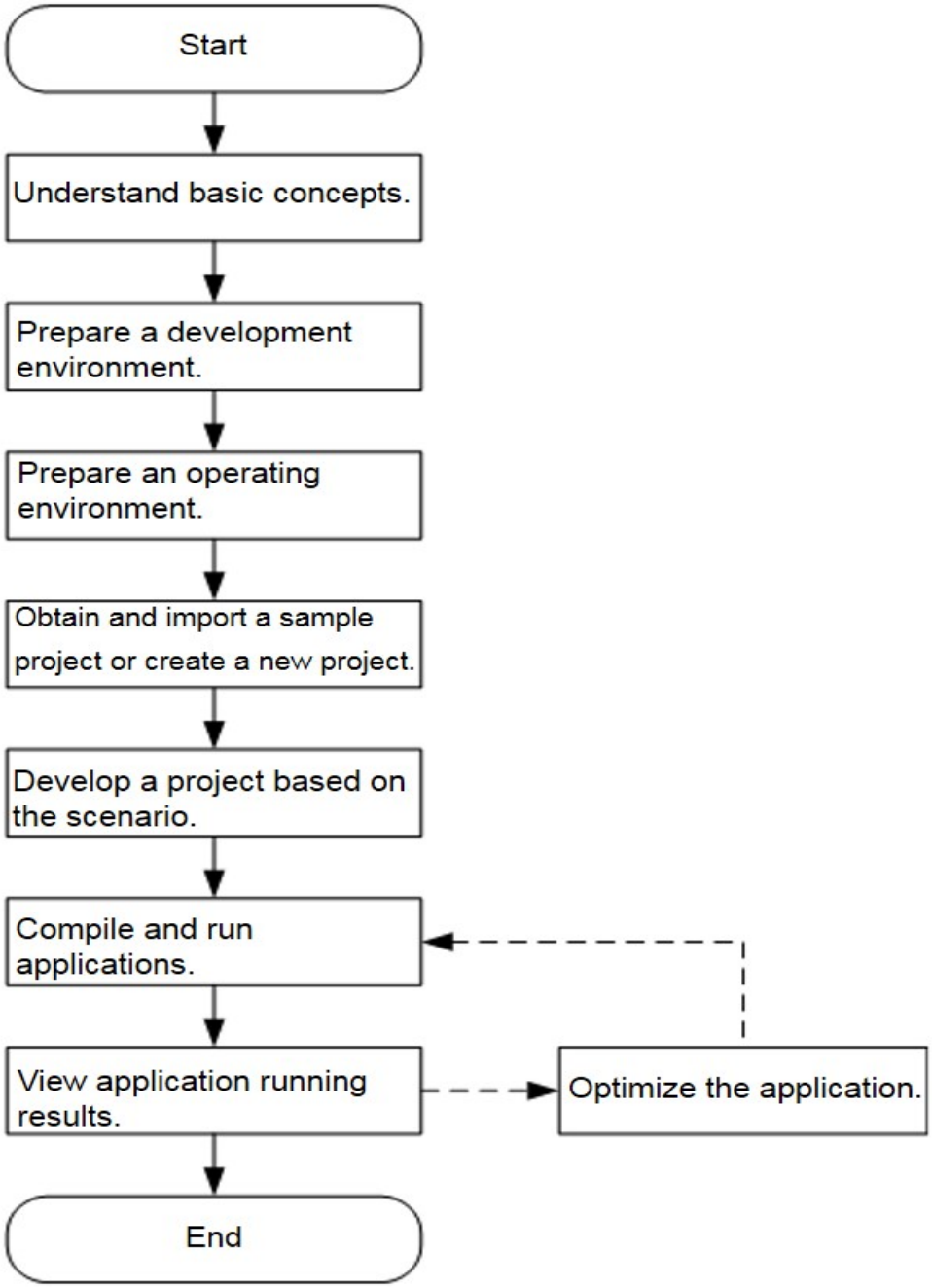


Table 14-2 Description of the Spark application development process

Phase	Description	Reference
Understand basic concepts.	Before application development, learn basic concepts of Spark. Select basic concepts of Spark Core, Spark SQL, and Spark Streaming to learn based on actual scenarios.	**Basic Concepts**

Phase	Description	Reference
Prepare a development environment.	Spark applications can be developed in Scala, Java, and Python. You are advised to use the IDEA tool to configure development environments in different languages according to the guide.	**Preparing a Java Development Environment for Spark** to **Preparing a Python Development Environment for Spark**
Prepare an operating environment.	The Spark operating environment is a Spark client. Install and configure the client according to the guide.	**Preparing a Spark Application Running Environment**
Obtain and import a sample project or create a new project.	Spark provides sample projects for different scenarios. You can import a sample project to learn the application. You can also create a Spark project according to the guide.	**Importing and Configuring Spark Sample Projects**
Develop a project based on the scenario.	Sample projects in 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 you quickly learn APIs of all Spark components.	**Scenario Description** to **Scala Sample Code**
Compile and run applications.	You can compile the developed application and submit it for running.	**Compiling and Running a Spark Application**
View application running results.	Application running results are exported to a path you specify. You can also view the application running status on the UI.	**Viewing the Spark Application Commissioning Result**
Optimize the application.	You can optimize a program based on its running status to meet the performance requirement in the current service scenario. After optimizing the application, compile and run it again.	**Data Serialization** to **Spark CBO Tuning**

14.2 Preparing a Spark Application Development Environment

14.2.1 Spark Application Development Environment

Table 14-3 describes the environment required for application development. You also need to prepare a Linux environment for verifying whether the application is running properly.

Table 14-3 Development environment

Item	Description
JDK installation	Basic configurations of the development environment. JDK 1.7 or 1.8 is required. **NOTE** For security purpose, MRS servers support only TLS 1.1 and TLS 1.2 encryption protocols. IBM JDK supports only TSL 1.0 by default. If you use IBM JDK, set **com.ibm.jsse2.overrideDefaultTLS** to **true**. After the parameter setting, TLS1.0/1.1/1.2 can be supported at the same time. For details, visit **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**.
IDEA installation and configuration	Tool used for developing Spark applications. Version 13.1.6 or later is required.
Scala installation	Basic configuration for the Scala development environment. Version 2.11.0 or later is required.
Scala plugin installation	Basic configuration for the Scala development environment. Version 0.35.683 or later is required.
Python installation	Basic configuration for the Python development environment. Python 2.7.x or later is required.

14.2.2 Preparing a Spark Application Development User

Prerequisites

Kerberos authentication has been enabled for the MRS cluster. Skip this step if Kerberos authentication is not enabled for the cluster.

Scenario

The development user is used to run the sample project. The user must have HDFS, Yarn, and Hive permissions to run the Spark sample project.

Procedure

Step 1 Log in to MRS Manager. For details, see **Logging in to MRS Manager**.

Step 2 On MRS Manager, choose **System** > **Manage Role** > **Create Role**.

1. Enter a role name, for example, *sparkrole*.
2. In **Permission**, choose **HBase** > **HBase Scope** > **global**. Select **Create** for **default**.
3. In **Permission**, choose **HBase** > **HBase Scope** > **global** > **hbase**. Select **Execute** for **hbase:meta**.
4. Modify the role. In **Permission**, choose **HDFS** > **File System**, select **Read**, **Write**, and **Execute**.
5. In **Permission**, select **HDFS** > **File System** > **hdfs://hacluster/** > **user** > **hive**, and select **Execute**.
6. In **Permission**, choose **HDFS** > **File System** > **hdfs://hacluster/** > **user** > **hive** > **warehouse**, and select **Read**, **Write**, and **Execute**.
7. In **Permission**, choose **Hive** > **Hive Read Write Privileges** and select **Create** for **default**.
8. In **Permission**, choose **Yarn** > **Scheduler Queue** > **root**, and select **Submit** for **default**.
9. Click **OK**.

Step 3 On MRS Manager, choose **System** > **Manage User** > **Create User** to create a user for the sample project. Enter a username, for example, **sparkuser**. Set **User Type** to **Machine-machine**, and select both **supergroup** and **kafkaadmin** in **User Group**. Set **Primary Group** to **supergroup**, select the **sparkrole** role to obtain permissions, and click **OK**.

NOTE

Users using the Spark Streaming program need the **kafkaadmin** group permission to operate the Kafka component.

Step 4 On MRS Manager, choose **System** > **Manage User** and select **sparkuser**. Click to download an authentication credential file. Save the file and decompress it to obtain the **keytab** and **krb5.conf** files. They are used for security authentication in the sample project. For details how to use them, see **Preparing the Authentication Mechanism Code**.

----End

14.2.3 Preparing a Java Development Environment for Spark

Scenario

The Java development environment can be set up on Windows, but the operating environment (client) can be deployed on Linux only.

Procedure

Step 1 The IDEA tool is recommended for the Java development environment. The installation requirements are as follows:

- JDK 1.7 or 1.8 is required.
- IntelliJ IDEA 13.1.6 is required.

NOTE

- Spark tasks cannot be submitted to a server in Yarn-client mode when client applications are running on IBM JDK 1.7.
- Oracle JDK requires security hardening. The operations are as follows:

1. Obtain the Java Cryptography Extension (JCE) file whose version matches that of JDK from the Oracle official website. After decompression, the JCE file contains **local_policy.jar** and **US_export_policy.jar**. Copy the JAR file to the following directory:

   Linux: JDK installation directory/jre/lib/security

   Windows: JDK installation directory\jre\lib\security

2. Copy **SMS4JA.jar** in the **Client installation directory/JDK/jdk/jre/lib/ext/** directory to the following directory:

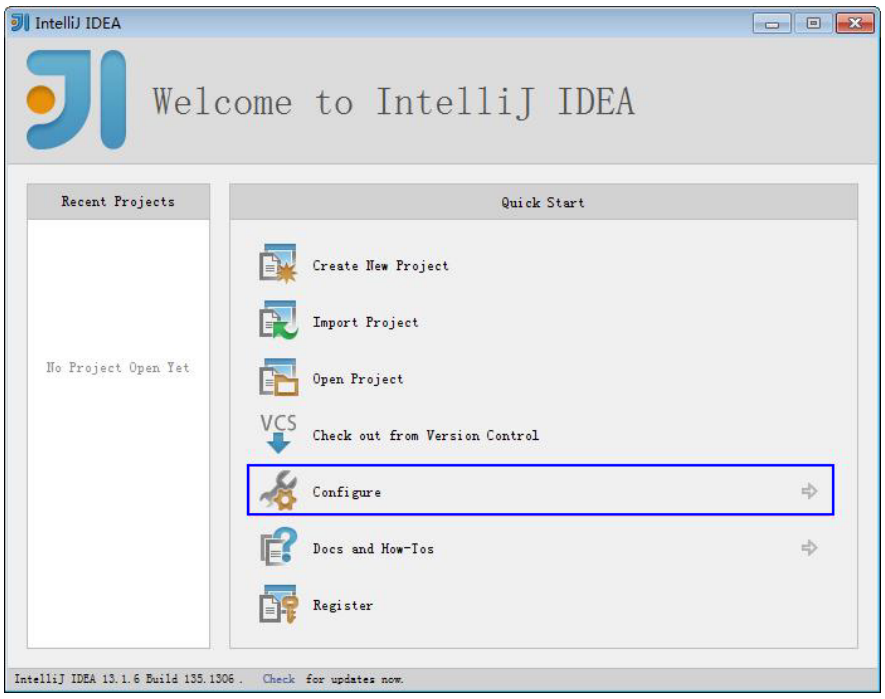
   Linux: *JDK installation directory***/jre/lib/ext/**

   Windows: *JDK installation directory***\jre\lib\ext**

Step 2 Install and configure the IntelliJ IDEA and JDK tools.

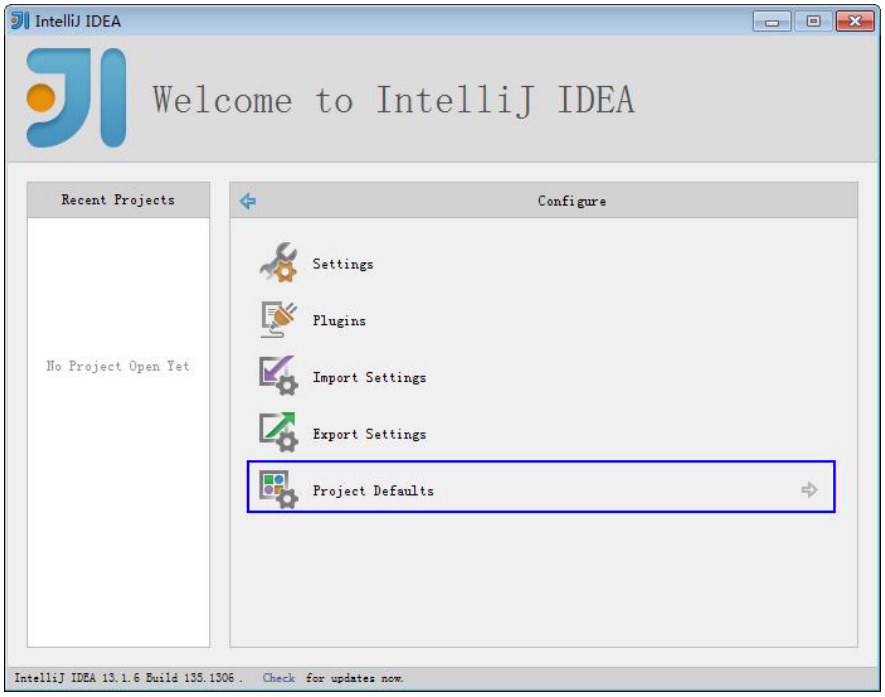
1. Install the JDK.
2. Install IntelliJ IDEA.
3. Configure the JDK in the IntelliJ IDEA.
   a. Start the IntelliJ IDEA and select **Configure**.

Figure 14-8 Quick start



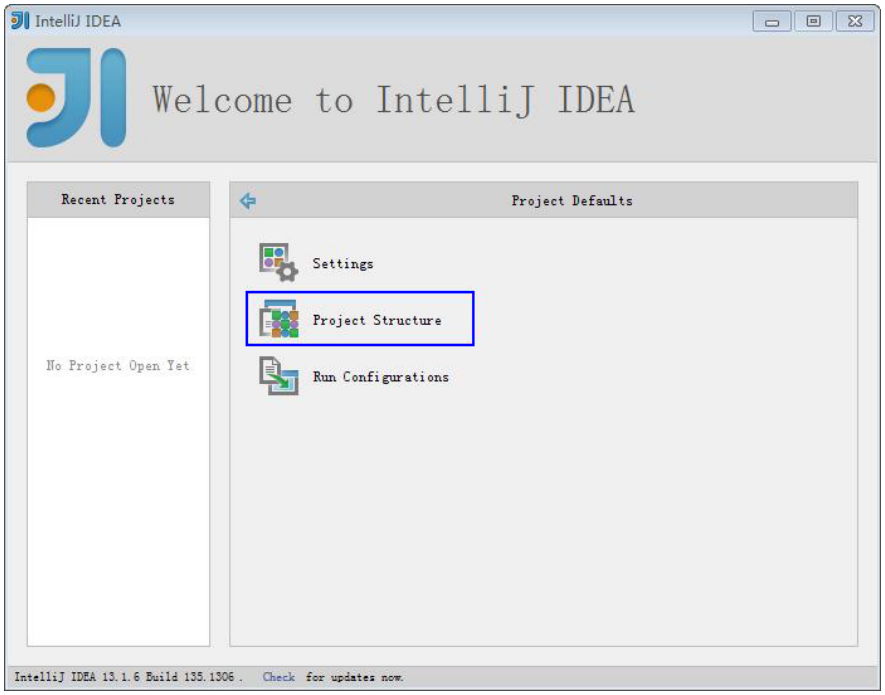
   b. Choose **Project Defaults** on the **Configure** page.

Figure 14-9 Configure page



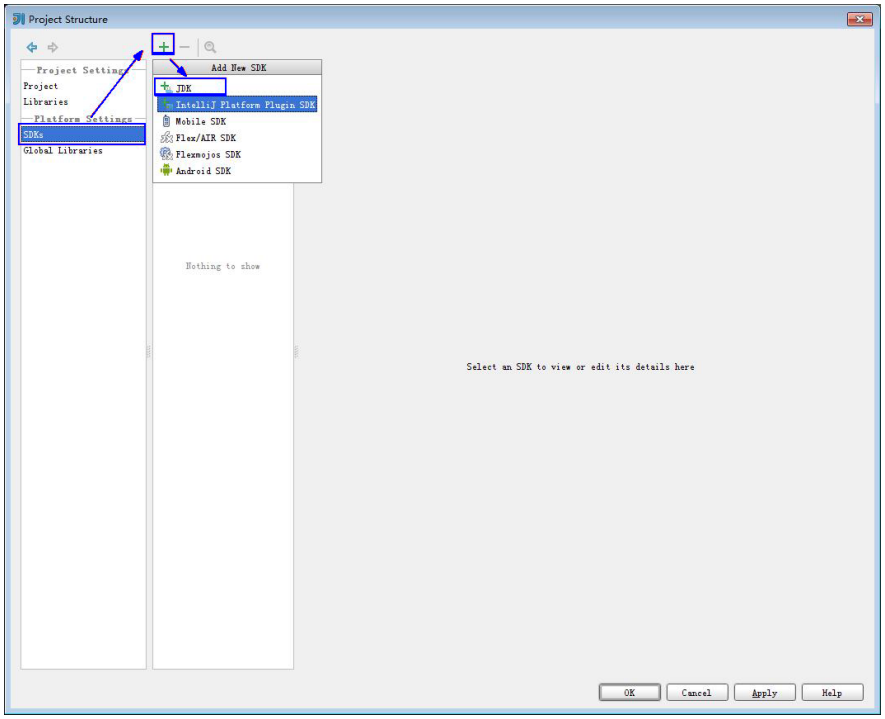
c. Choose **Project Structure** on the **Project Defaults** page.

Figure 14-10 Project Defaults page



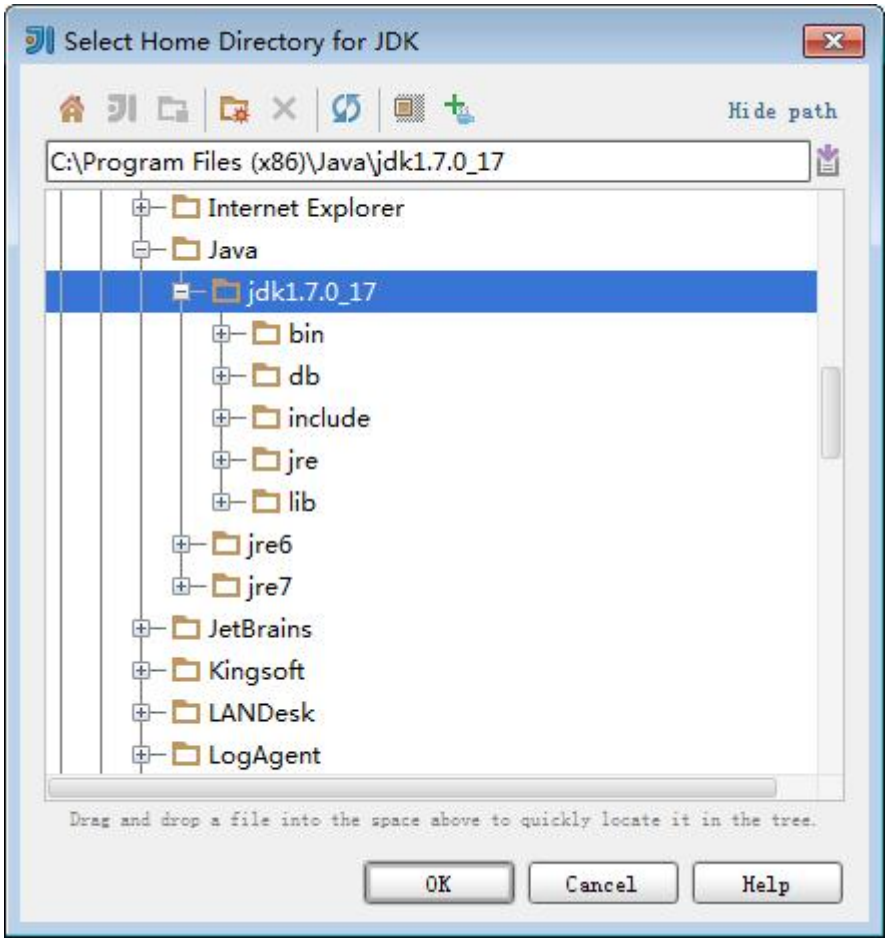
d. On the **Project Structure** page, select **SDKs** and click the green plus sign to add the JDK.

Figure 14-11 Adding the JDK



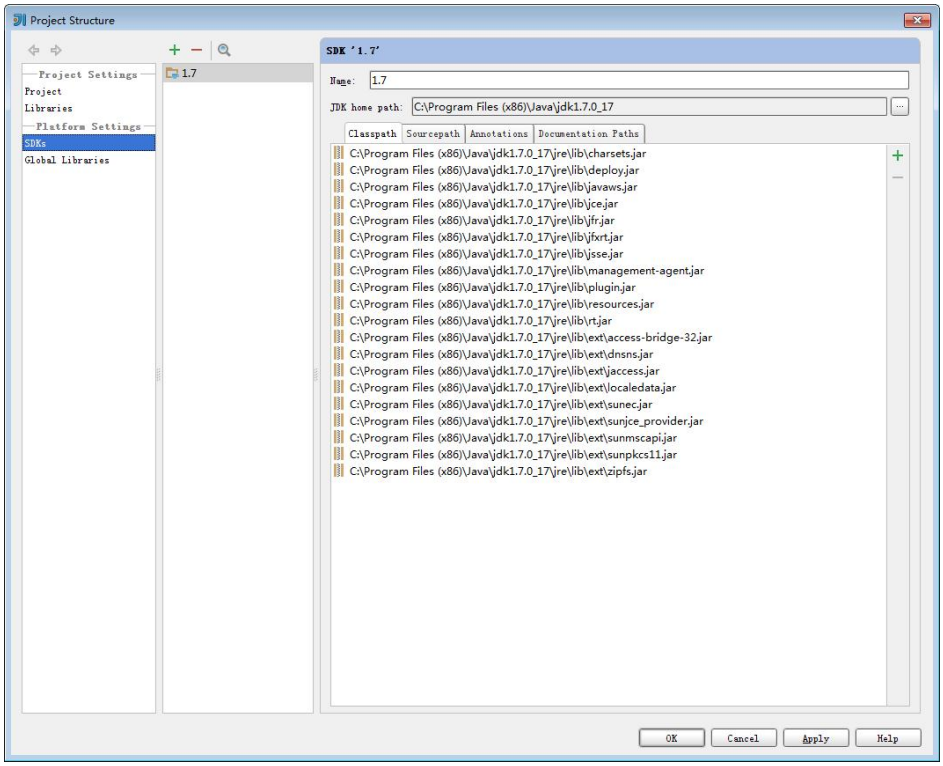
e. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 14-12 Selecting the JDK directory



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

Figure 14-13 Completing the JDK configuration



----End

14.2.4 Preparing a Scala Development Environment for Spark

Scenario

The Scala development environment can be set up on Windows, but the operating environment (client) can be deployed on Linux only.

Procedure

Step 1 The IDEA tool is recommended for the Scala development environment. The installation requirements are as follows:

- JDK 1.7 or 1.8 is required.
- IntelliJ IDEA 13.1.6 is required.
- Scala 2.11.8 is required.
- Scala plugin 0.35.683 is required.

NOTE

- Spark tasks cannot be submitted to a server in Yarn-client mode when client applications are running on IBM JDK 1.7.
- Oracle JDK requires security hardening. The operations are as follows:

1. Obtain the JCE file whose version matches that of JDK from the Oracle official website. After decompression, the JCE file contains **local_policy.jar** and **US_export_policy.jar**. Copy the JAR file to the following directory:

   Linux: *JDK installation directory*/**jre/lib/security**

   Windows: *JDK installation directory***jre\lib\security**
2. Copy **SMS4JA.jar** in the **Client installation directory/JDK/jdk/jre/lib/ext/** directory to the following directory:

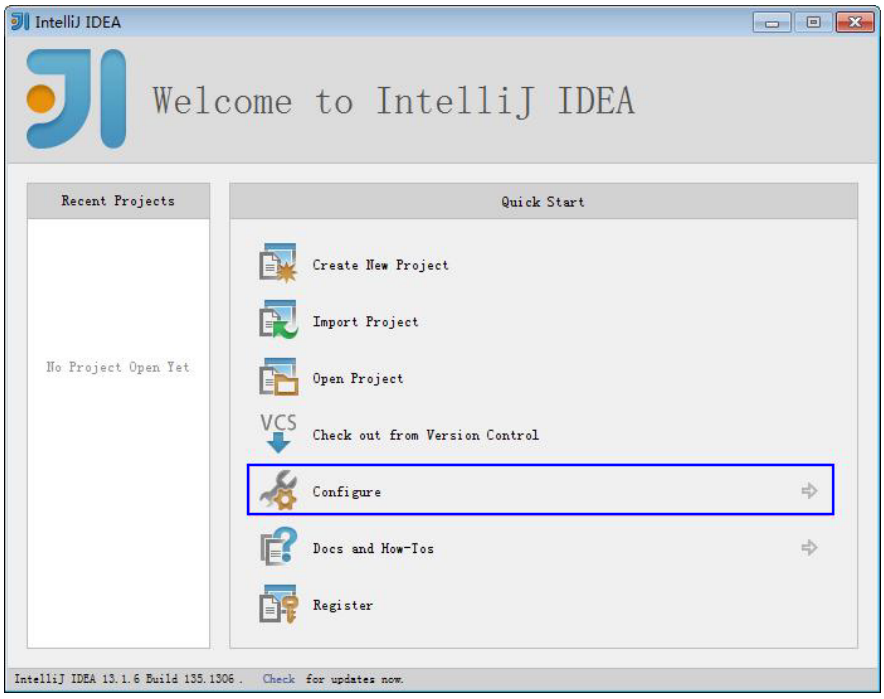
   Linux: *JDK installation directory*/**jre/lib/ext/**

   Windows: *JDK installation directory***jre\lib\ext**

Step 2 Install and configure the IntelliJ IDEA, JDK, and Scala tools.

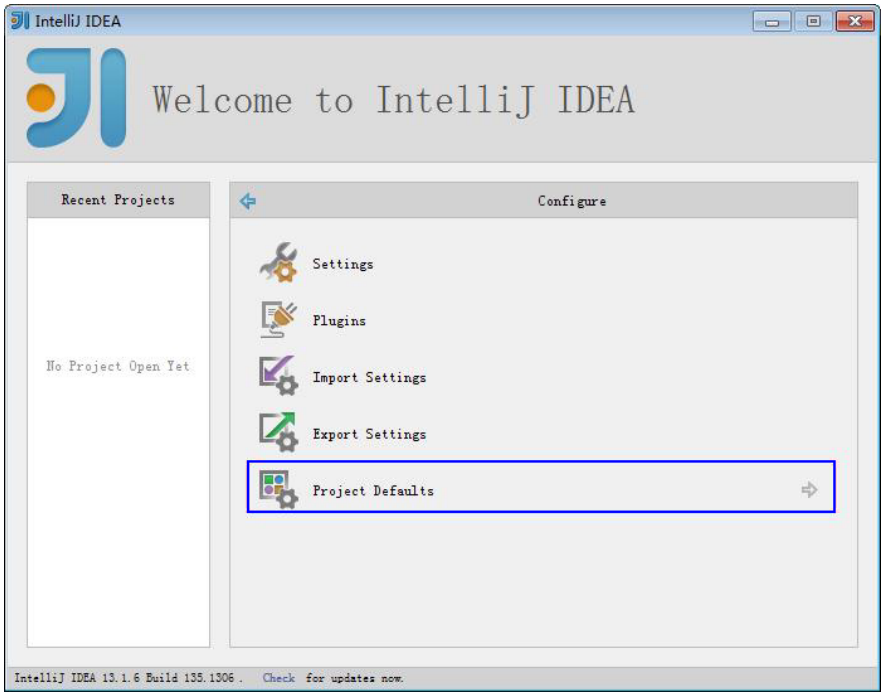
1. Install the JDK.
2. Install the IntelliJ IDEA.
3. Install the Scala.
4. Configure the JDK in the IntelliJ IDEA.
   a. Start the IntelliJ IDEA and select **Configure**.

Figure 14-14 Quick Start



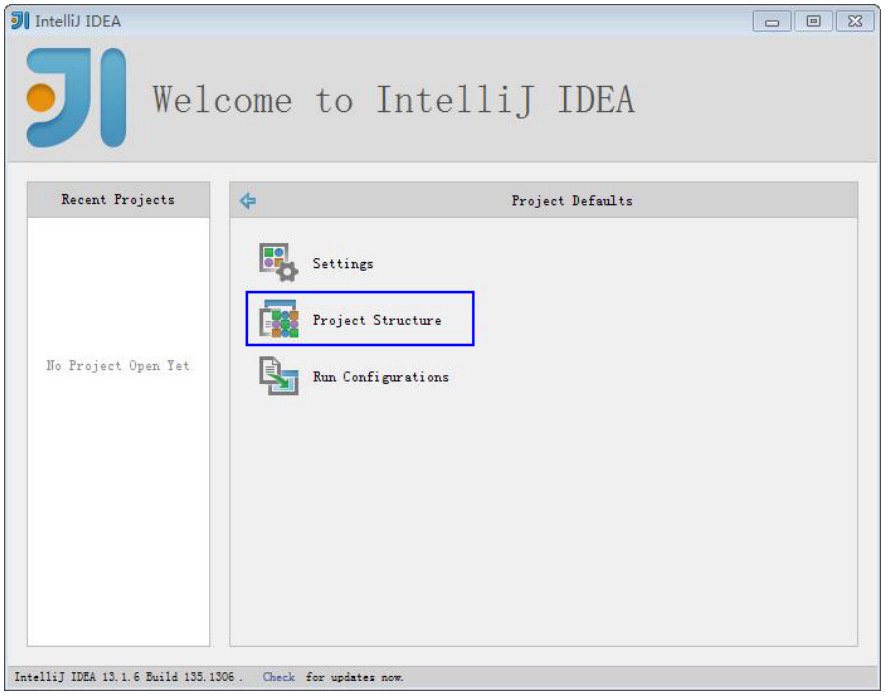
   b. Choose **Project Defaults** on the **Configure** page.

Figure 14-15 Configure



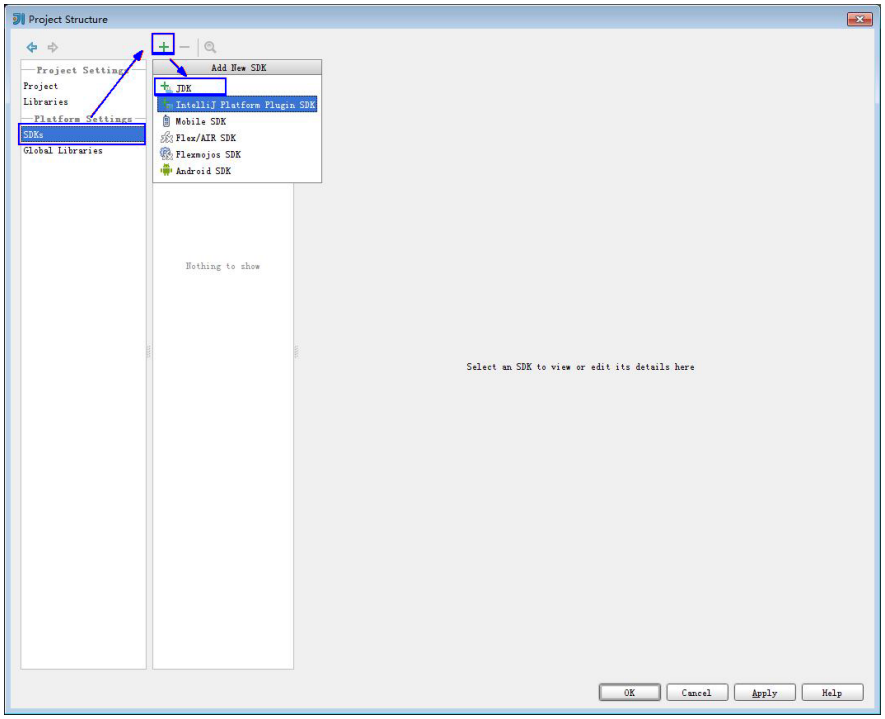
c. Choose **Project Structure** on the **Project Defaults** page.

Figure 14-16 Project Defaults



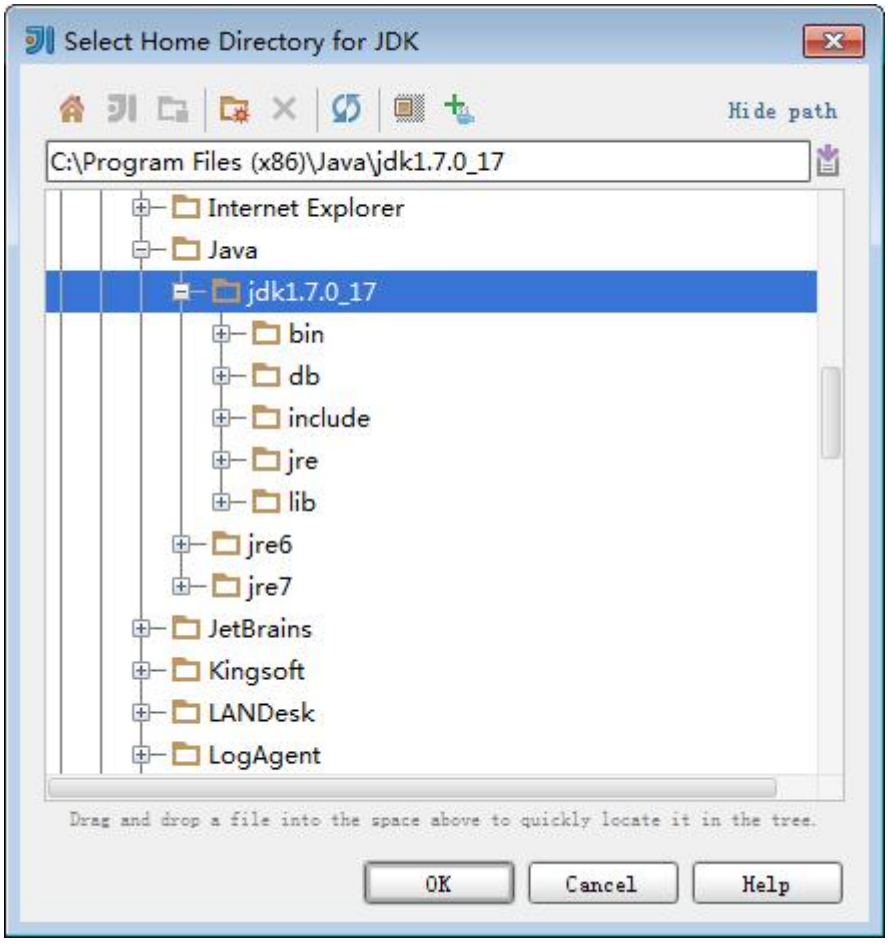
d. On the **Project Structure** page, select **SDKs** and click the green plus sign to add the JDK.

Figure 14-17 Adding the JDK



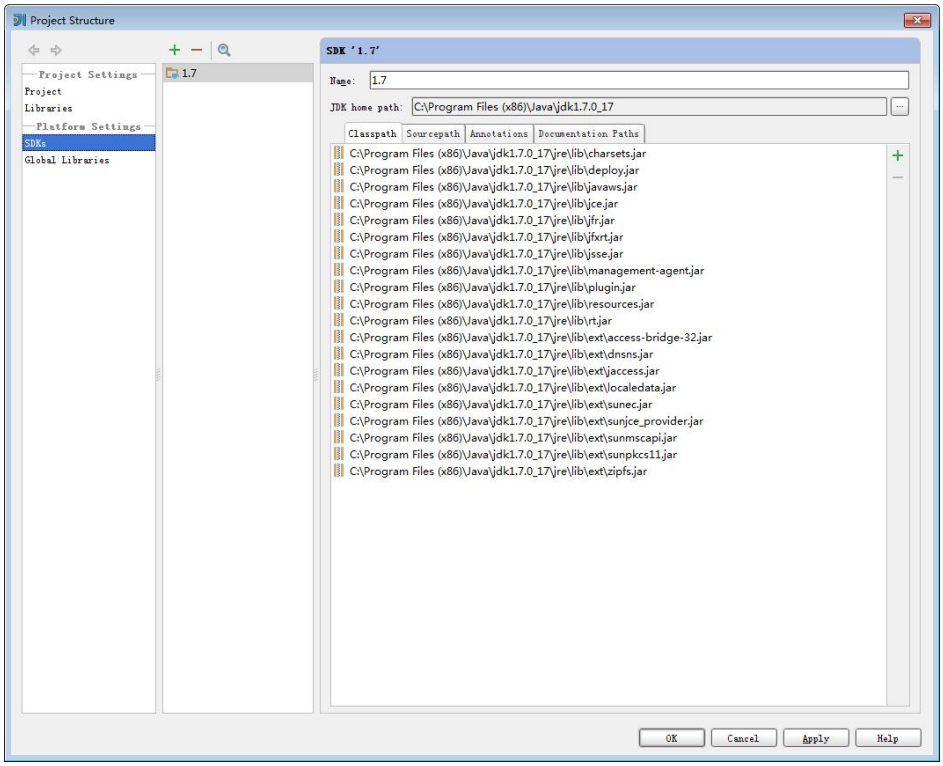
e. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 14-18 Selecting the JDK directory



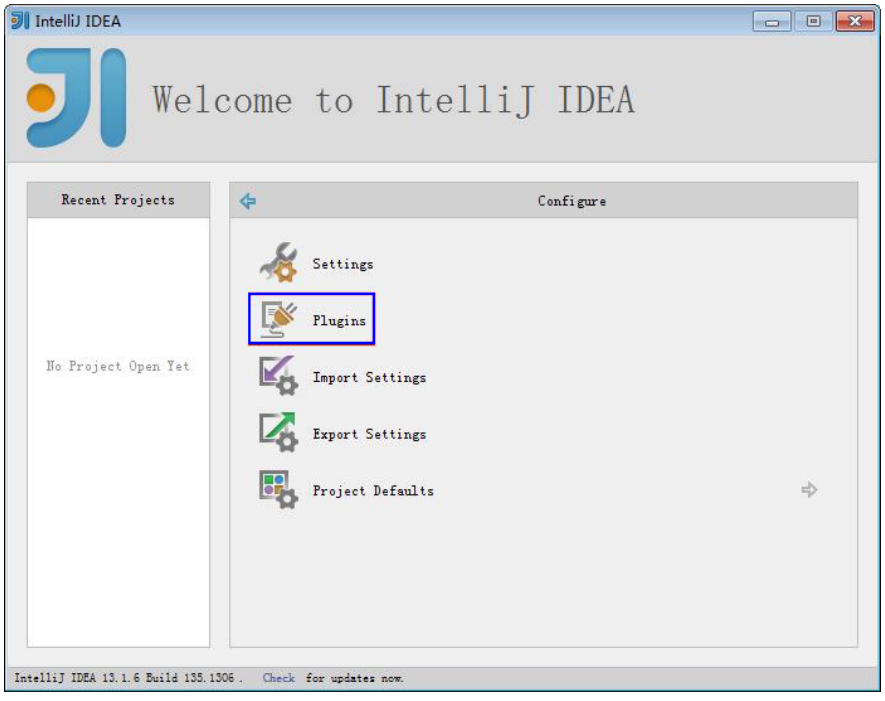
f. After selecting the JDK, click **OK** to complete the configuration.

Figure 14-19 Completing the JDK configuration



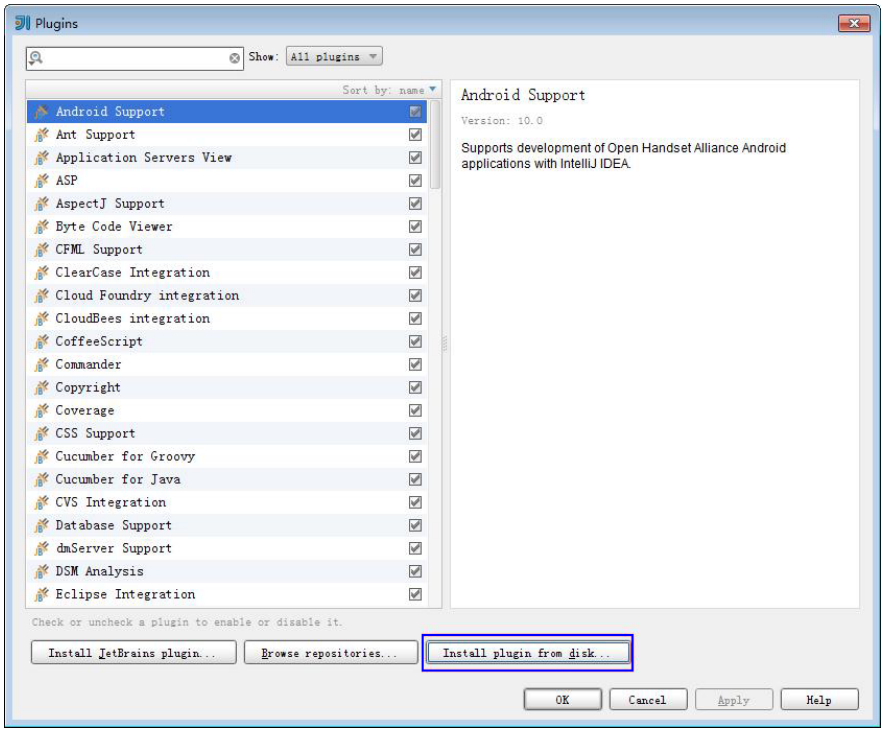
5. Install the Scala plugin in the IntelliJ IDEA.
   a. On the **Configure** page, select **Plugins**.

Figure 14-20 Plugins

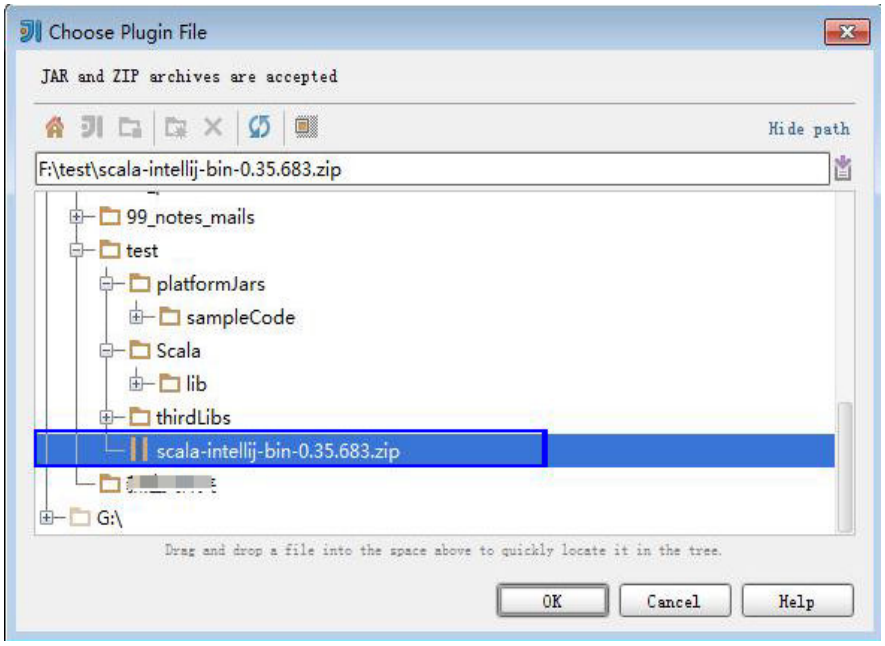


   b. On the **Plugins** page, select **Install plugin from disk**.

Figure 14-21 Install plugin from disk



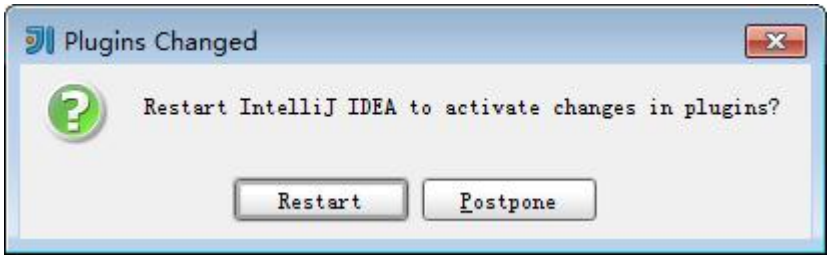
c. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.



d. On the **Plugins** page, click **Apply** to install the Scala plugin.

e. On the **Plugins Changed** page that is displayed, click **Restart** to make the configurations take effect.

Figure 14-22 Plugins Changed



----End

14.2.5 Preparing a Python Development Environment for Spark

Scenario

The Python development environment can be set up on Windows, but the operating environment (client) can be deployed on Linux only.

Procedure

Step 1 In the Python development environment, use the Editra editor or other IDEs for compiling Python applications.

Step 2 Download a client sample configuration program to the local development environment.

Use the FTP tool to download the **MRS_Service_client** client package in the commissioning environment to the local PC and decompress the package to obtain the **MRS_Services_ClientConfig** directory.

----End

14.2.6 Preparing a Spark Application Running Environment

Scenario

The operating environment (client) of Spark can be deployed on Linux only. Perform the following operations to prepare the operating environment.

Preparing a Running and Commissioning Environment

Step 1 On the ECS management console, apply for a new ECS for application running and commissioning.

- The security group of the ECS must be the same as that of the master node in an MRS cluster.
- The ECS and the MRS cluster must be in the same VPC.
- The ECS network interface controller (NIC) and the MRS cluster must be in the same network segment.

Step 2 Apply for an EIP, bind it, and configure an inbound or outbound rule for the security group.

Step 3 Download the client program. For details, see **Downloading an MRS Client**.

Step 4 Log in to the client to download the target node and install a cluster client as user **root**.

1. Run the following command to decompress the client package:

   tar -xvf /opt/MRS_Services_Client.tar

2. Run the following command to verify the installation file package:

   sha256sum -c /opt/MRS_Services_ClientConfig.tar.sha256

   ```
   MRS_Services_ClientConfig.tar:OK
   ```

3. Run the following command to decompress the installation file package:

   tar -xvf MRS_Services_ClientConfig.tar

4. Run the following command to install the client to a specified directory (absolute path), for example, **/opt/client**. The directory is automatically created.

   cd /opt/MRS_Services_ClientConfig

   sh install.sh /opt/client

   ```
   Components client installation is complete.
   ```

----End

14.2.7 Importing and Configuring Spark Sample Projects

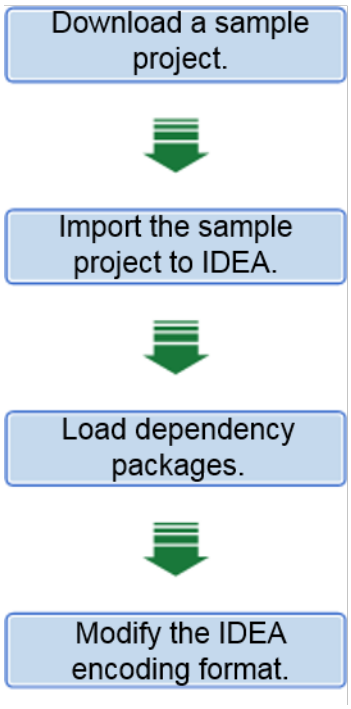
Scenario

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

Methods to import Java and Scala projects are the same. Python sample projects do not need to be imported, and you only need to open the Python file (*.py).

The following example describes how to import Java sample code. **Figure 14-23** shows the operation process.

Figure 14-23 Procedure of importing a sample project



Procedure

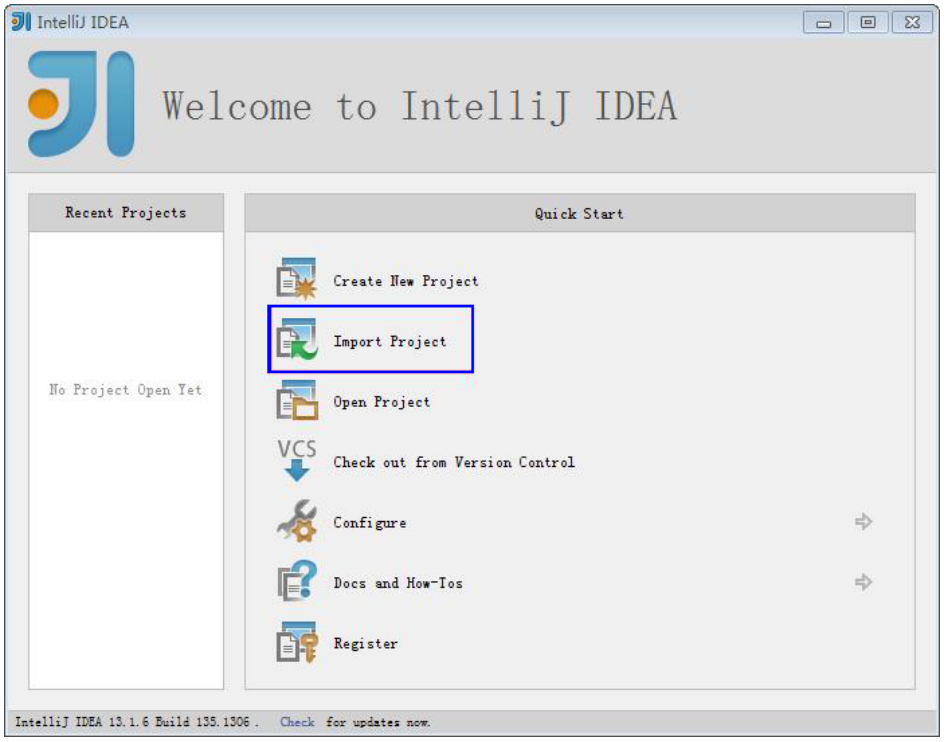
Step 1 Download the sample project to the local computer by referring to **Obtaining the MRS Application Development Sample Project**.

Step 2 Import the Java sample project to IDEA.

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

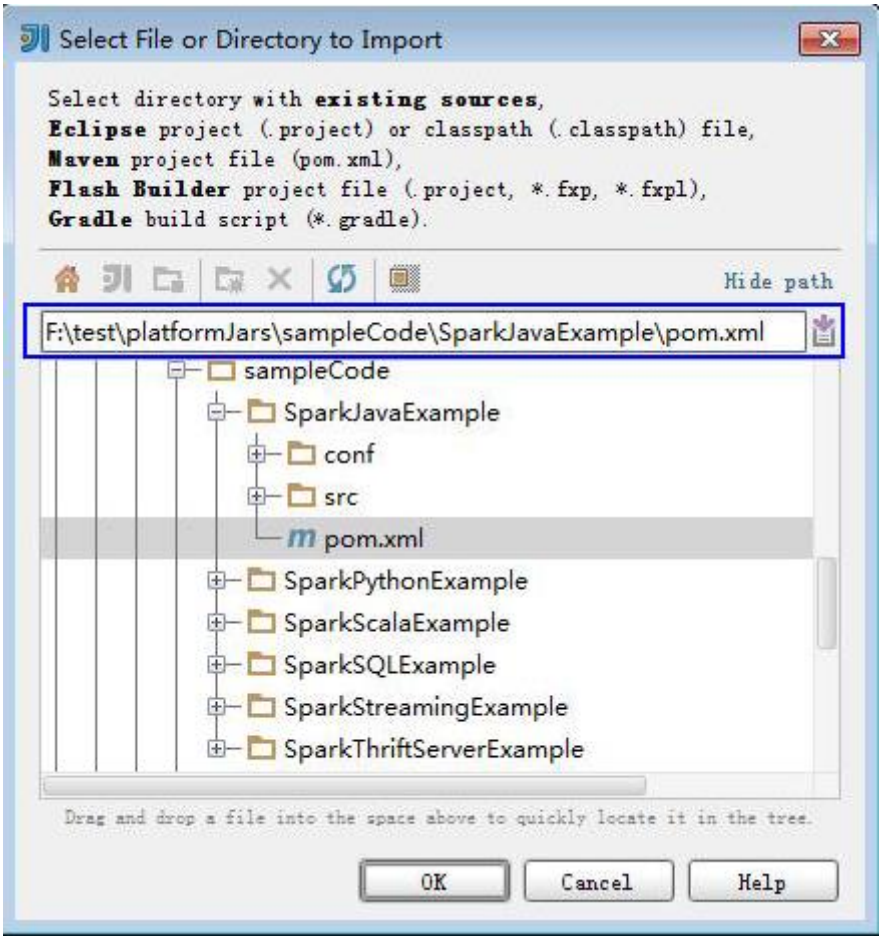
   Alternatively, for the used IDEA tool, add the project directly from the IDEA home page. Choose **File** > **Import project...** to import a project.

Figure 14-24 Importing the project (on the **Quick Start** page)



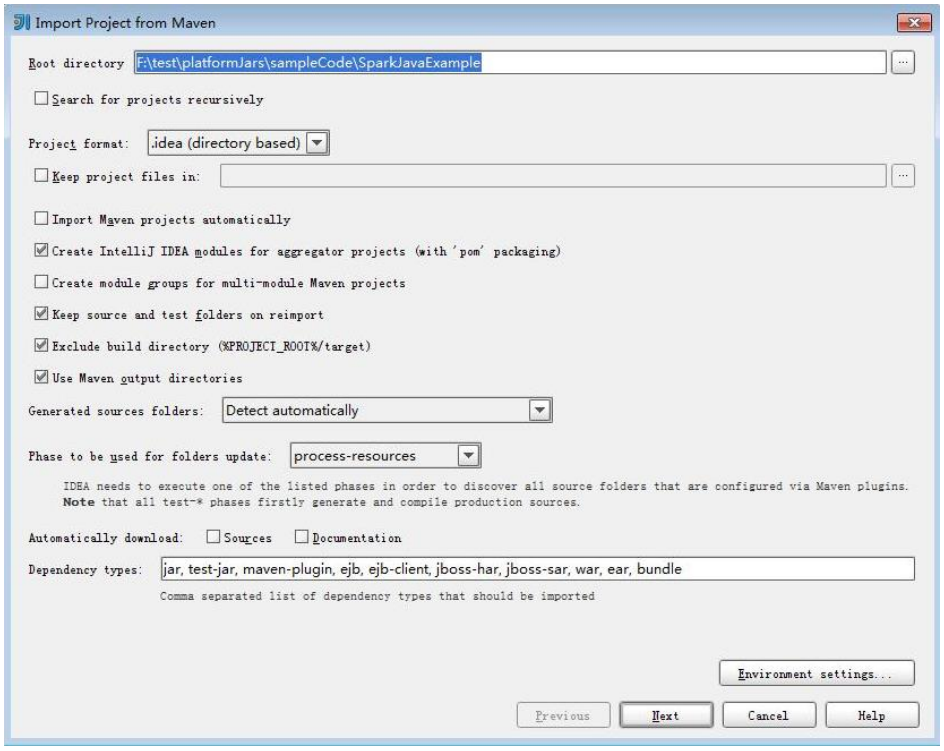
2. Select a path for storing the sample project to be imported and click **OK**.

Figure 14-25 Selecting a file or directory to import



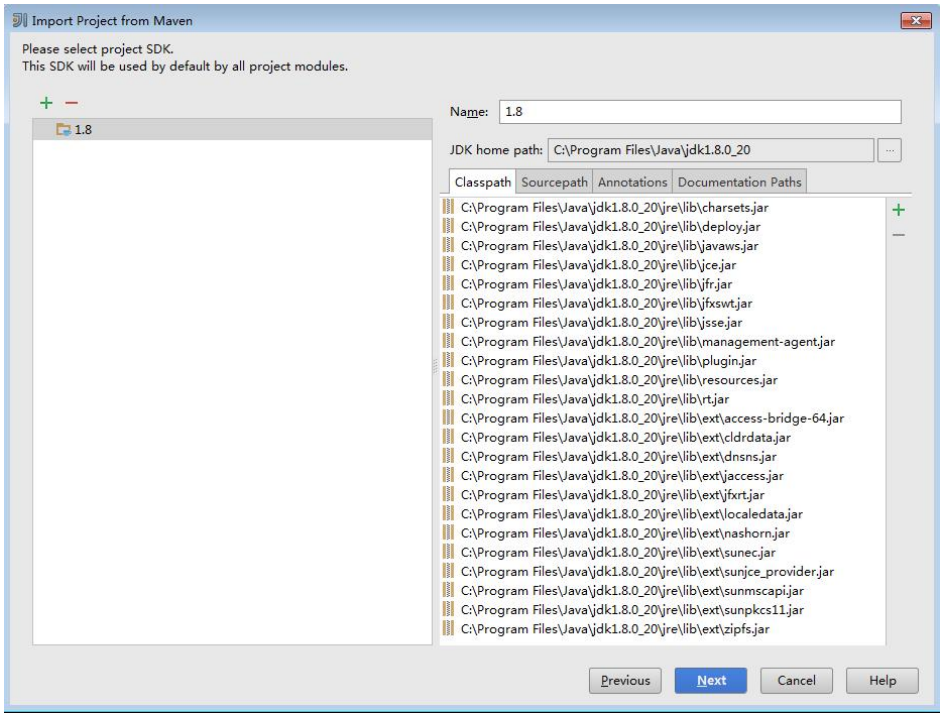
3. Confirm the path and project name, and click **Next**.

Figure 14-26 Importing a project from Maven



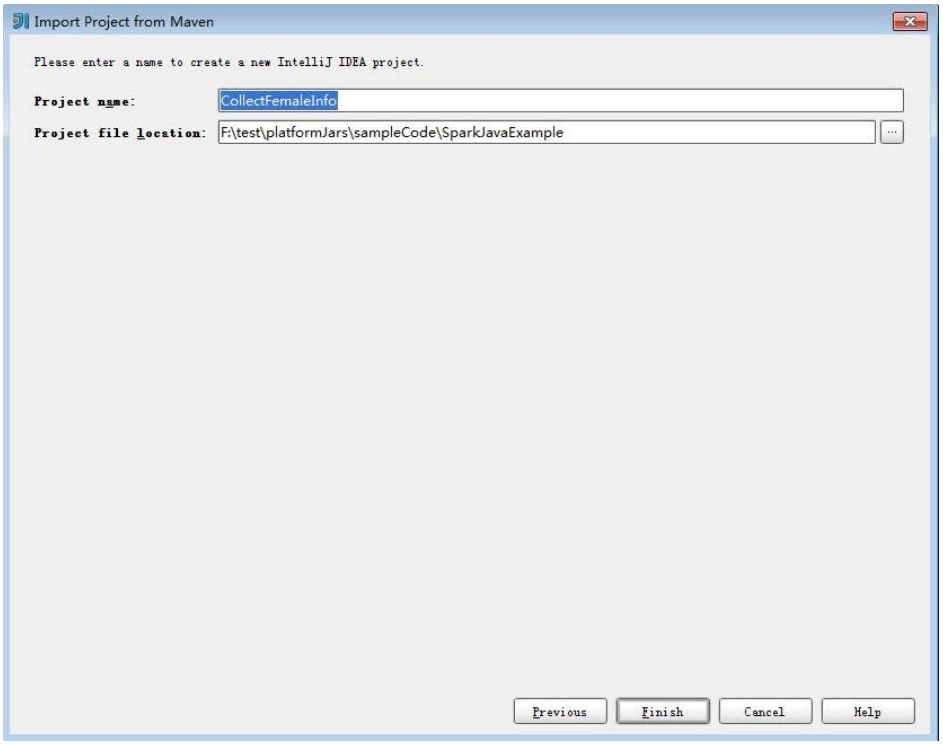
4. Select the project to import and click **Next**.
5. Confirm the JDK used by the project and click **Next**.

Figure 14-27 Selecting the SDK



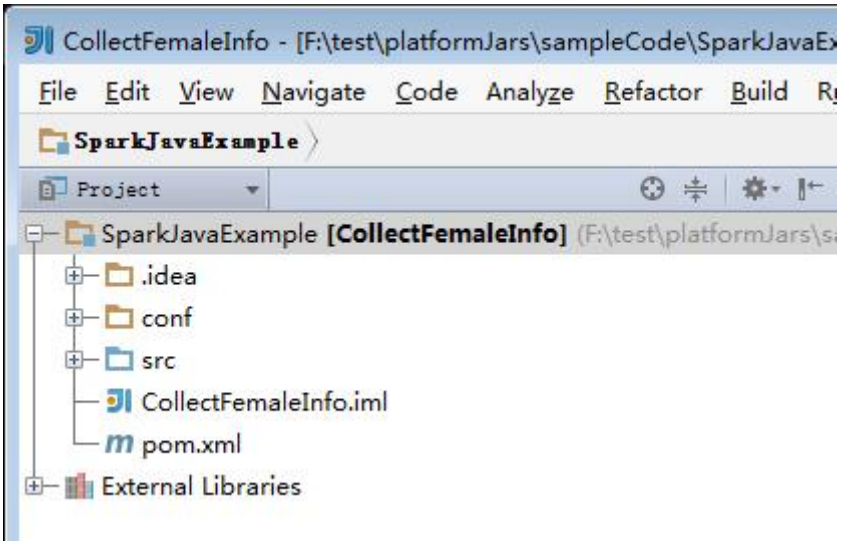
6. Confirm the project name and path and click **Finish** to complete the import.

Figure 14-28 Selecting a project to import



7. After the import, the imported project is displayed on the IDEA home page.

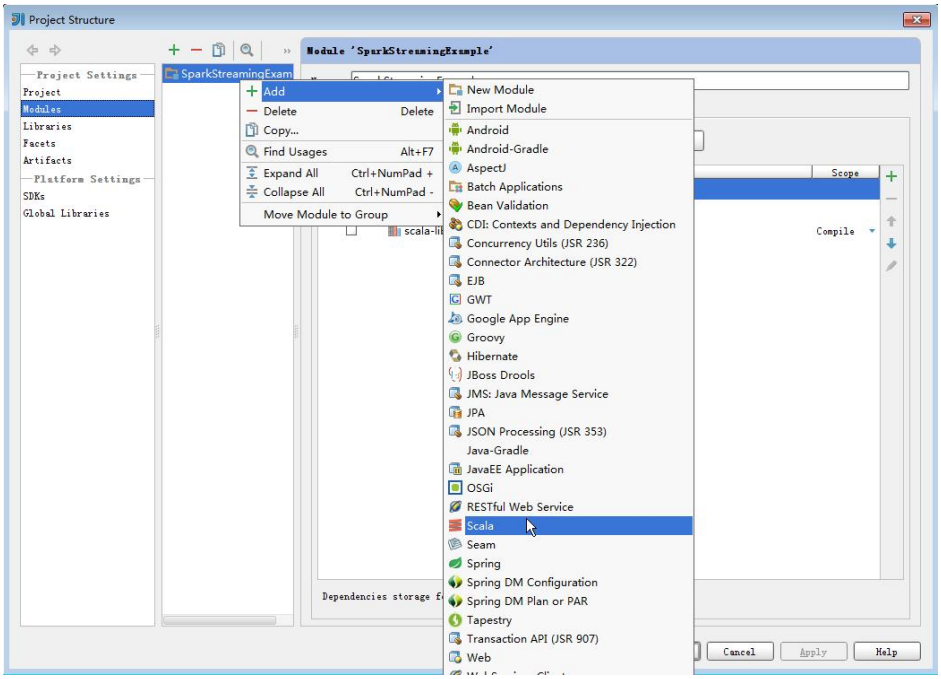
Figure 14-29 Imported project



Step 3 (Optional) If a Scala sample application is imported, configure a language for the project.

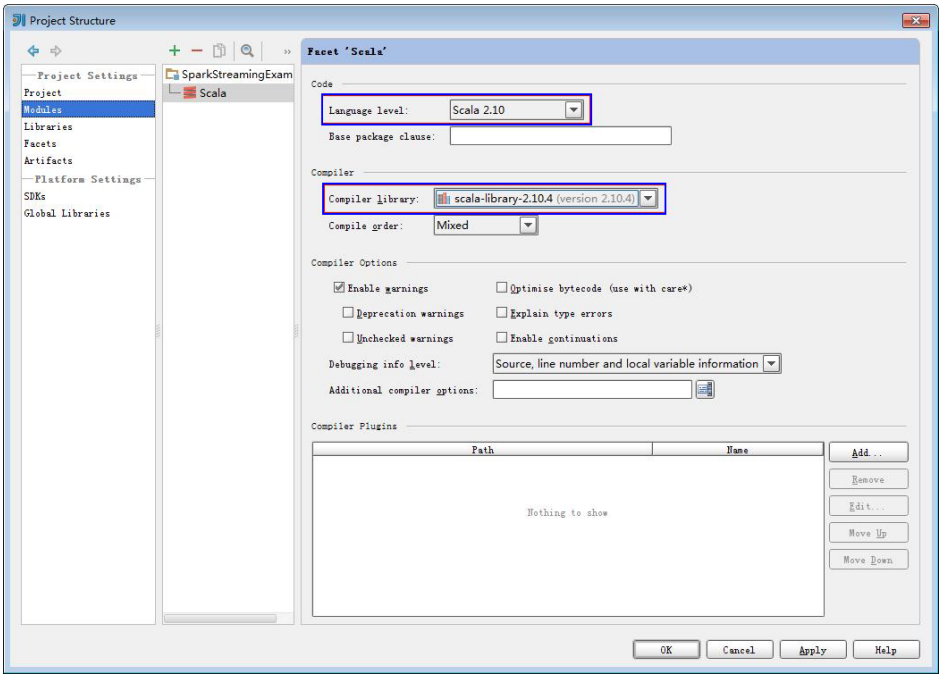
1. On the IDEA home page, choose **File** > **Project Structures...** to go to the **Project Structure** page.
2. Choose **Modules**, right-click a project name, and choose **Add** > **Scala**.

Figure 14-30 Selecting Scala



3. On the setting page, select the compiled dependency JAR file and click **Apply**.

Figure 14-31 Selecting the compiler library

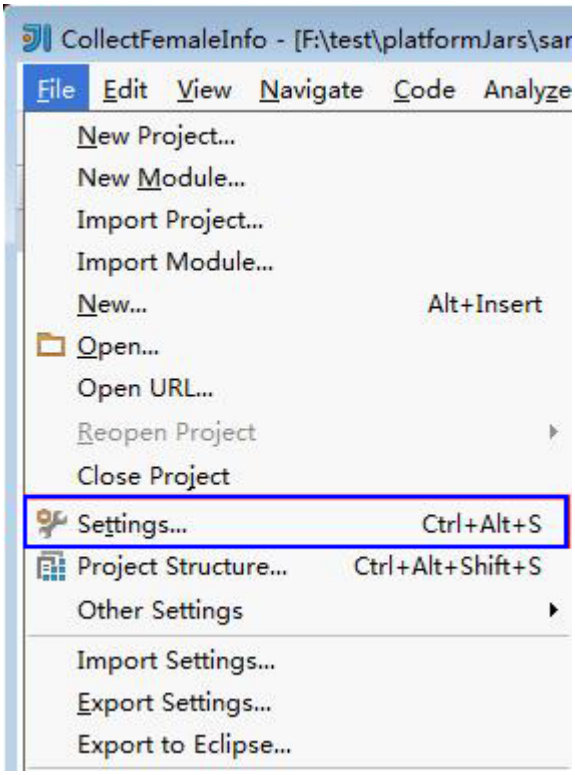


4. Click **OK** to save the settings.

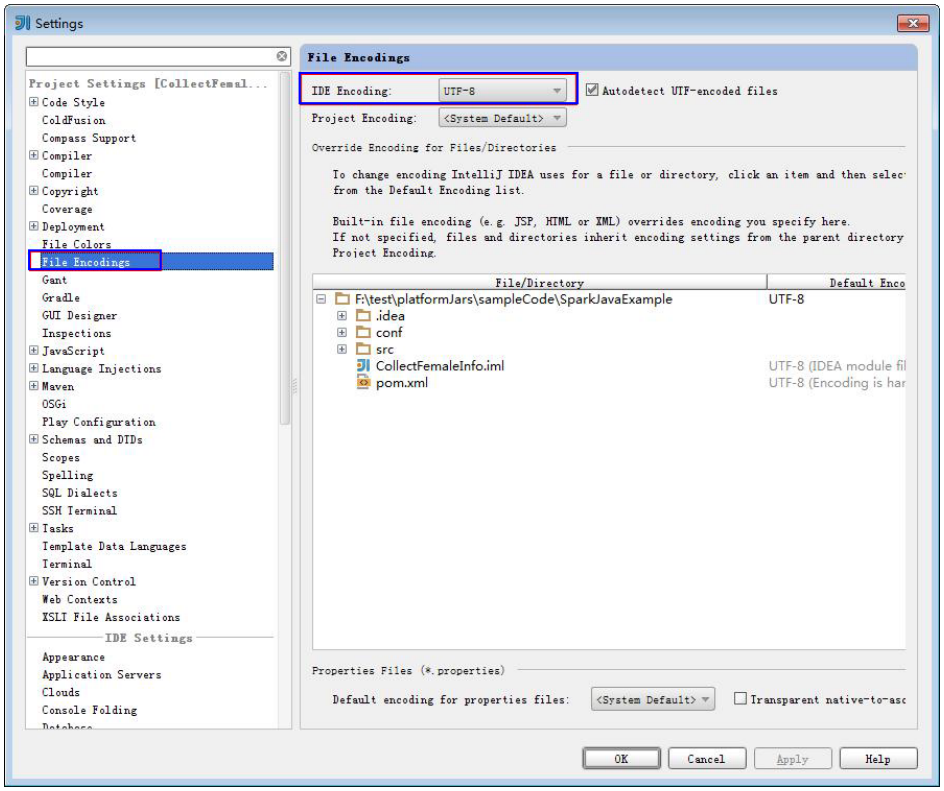
Step 4 Configure the text file encoding format of IDEA to prevent garbled characters.

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

Figure 14-32 Choosing **Settings**



2. On the **Settings** page, choose **File Encodings**. Select **UTF-8** from the **IDE Encoding** drop-down list on the right. Click **Apply**.



3. Click **OK** to complete the encoding settings.

----End

14.2.8 (Optional) Creating a Spark Application Development Project

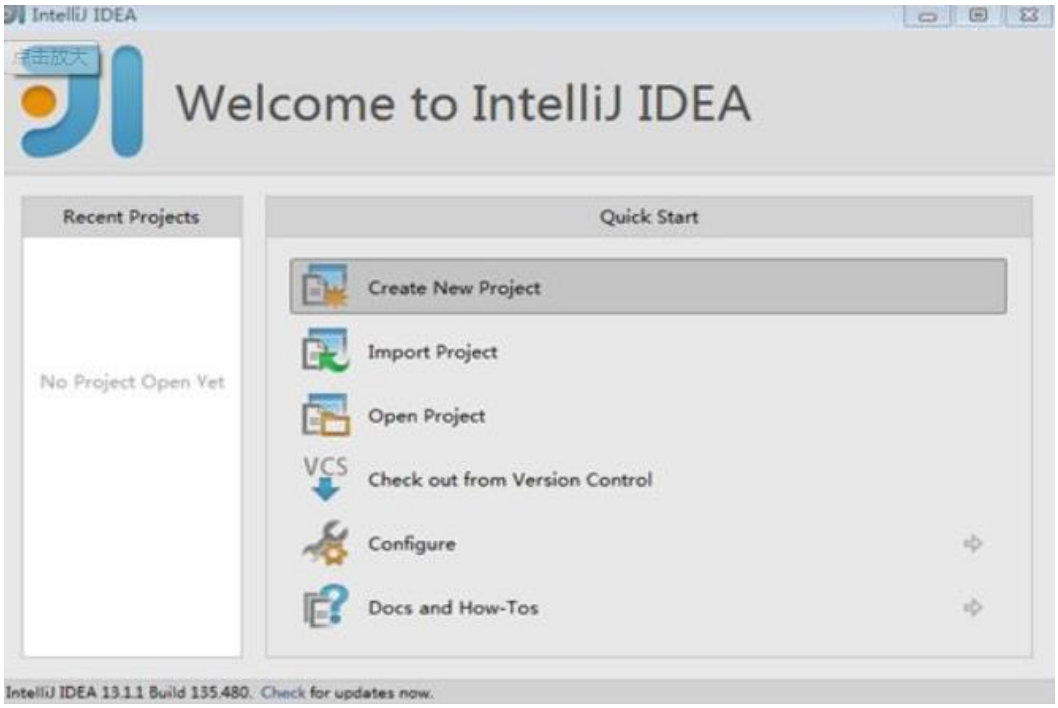
Scenario

In addition to importing Spark sample projects, you can use IDEA to create a Spark project. The following describes how to create a Scala project.

Procedure

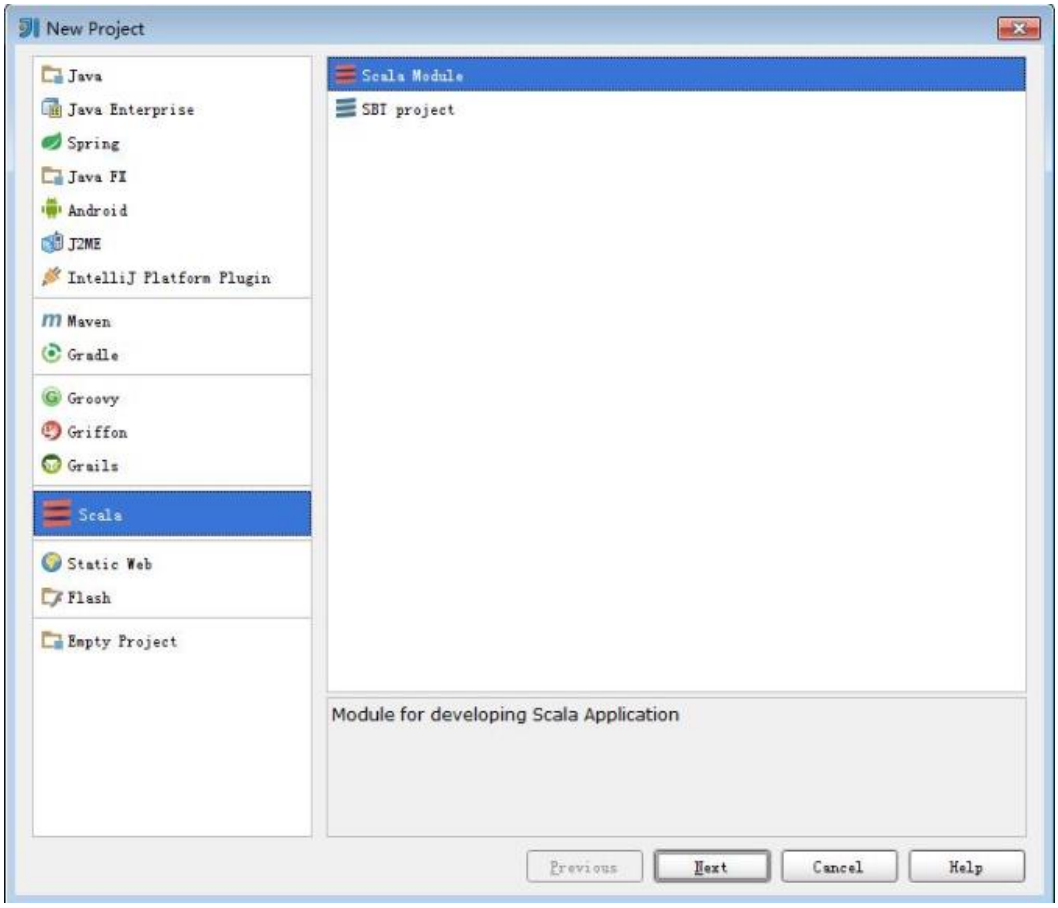
Step 1 Start the IDEA tool and choose **Create New Project**.

Figure 14-33 Creating a project



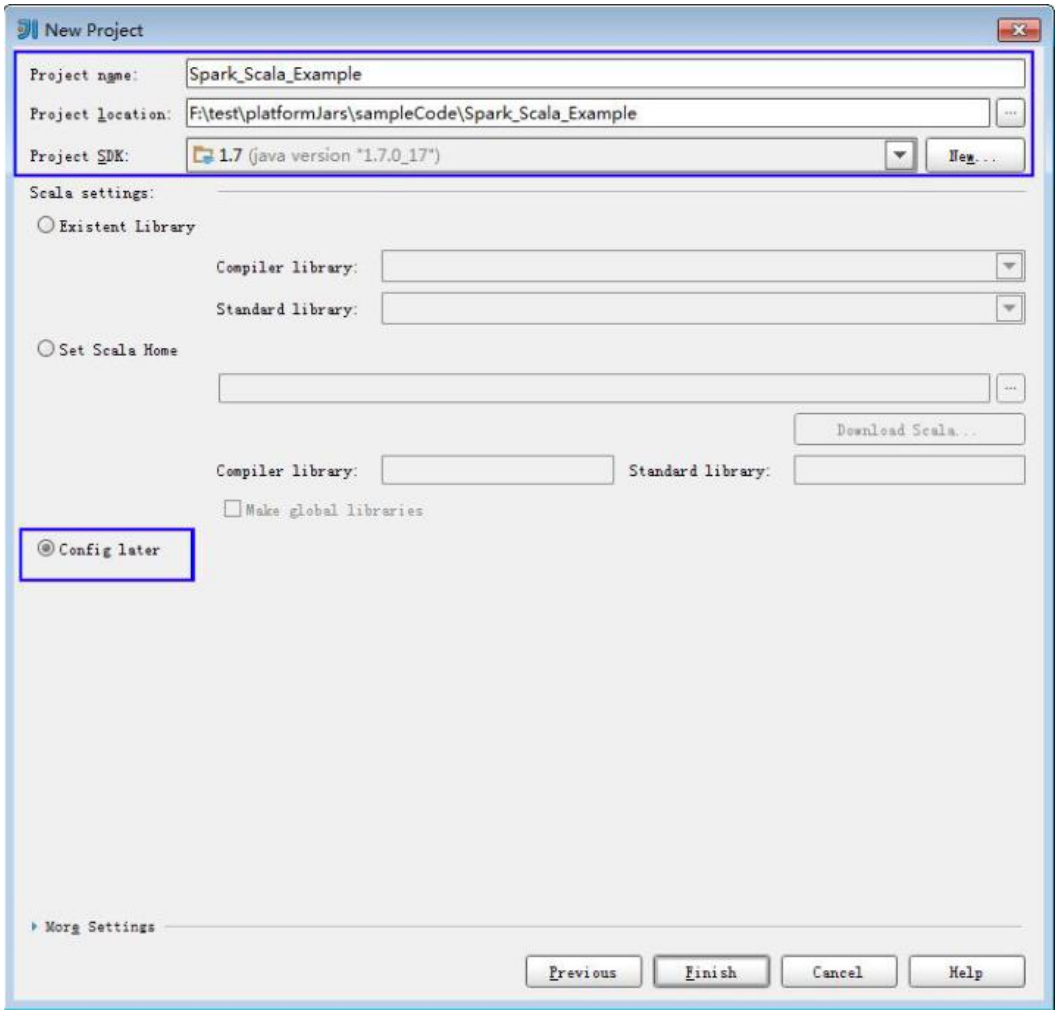
Step 2 On the **New Project** page, choose **Scala** > **Scala Module** and click **Next**. If you need to create a Java project, select the corresponding parameter.

Figure 14-34 Selecting a development environment



Step 3 On the project information page, specify **Project name**, **Project location**, and **Project JDK**, select **Config later** (indicating that the compiling file of Scala is imported after the project creation), and click **Finish** to complete the project creation

Figure 14-35 Entering the project information



----End

14.2.9 Configuring Security Authentication for Spark Applications

Prerequisites

You have enabled Kerberos authentication for the MRS cluster.

Scenario Description

In a cluster with Kerberos authentication enabled, the components must be mutually authenticated before communicating with each other to ensure communication security.

In some cases, Spark needs to communicate with Hadoop and HBase when users develop Spark applications. Codes for security authentication need to be written into the Spark applications to ensure that the Spark applications can work properly.

Three security authentication modes are available:

- Command authentication:

  Before running the Spark applications or using the CLI to connect to Spark SQL, run the following command on the Spark client for authentication:

  kinit *Component service user*

- Configuration authentication:

  You can specify security authentication information in any of the following ways:

  a. In the **spark-default.conf** configuration file of the client, set **spark.yarn.keytab** and **spark.yarn.principal** to specify the authentication information.

  b. Add the following parameters to the **bin/spark-submit** command to specify authentication information.

     --conf spark.yarn.keytab=<*keytab file path*> **--conf spark.yarn.principal**=<*Principal account*>

  c. Add the following parameters to the **bin/spark-submit** command to specify authentication information.

     --keytab <*keytab file path*> **--principal** <*Principal account*>

- Code authentication:

  Obtain the **principal** and **keytab** files of the client for authentication.

  The following table lists the authentication method used by the sample code in the cluster with Kerberos authentication enabled.

Table 14-4 Security authentication method

Sample Code	Mode	Security Authentication Method
spark-examples-normal	yarn-client	Command authentication, configuration authentication, or code authentication
	yarn-cluster	Either command authentication or configuration authentication
spark-examples-security (including security authentication code)	yarn-client	Code authentication
	yarn-cluster	Not supported

NOTE

- In the preceding table, the yarn-cluster mode does not support security authentication in the Spark project code, because authentication must be completed before the application is started.
- The security authentication code of the Python sample project is not provided. You are advised to set security authentication parameters in the command for running applications.

Security Authentication Code (Java)

Currently, the sample code invokes the LoginUtil class for security authentication in a unified manner.

In the Spark sample project code, different sample projects use different authentication codes. Basic security authentication or ZooKeeper authentication is used. The following table describes the example authentication parameters used in the sample project. Change the parameter values based on the site requirements.

Table 14-5 Parameters

Parameter	Example Value	Description
userPrincipal	sparkuser	Principal account used for authentication. You can obtain the account from the administrator.
userKeytabPath	/opt/FIclient/user.keytab	Keytab file used for authentication. You can obtain the file from the administrator.
krb5ConfPath	/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf	Path and name of the **krb5.conf** file
ZKServerPrincipal	zookeeper/hadoop.hadoop.com	Principal of the ZooKeeper server. Contact the administrator to obtain the account.

- Basic security authentication:

  Spark Core and Spark SQL applications do not need to access HBase or ZooKeeper. They need only the basic authentication code. Add the following code to the applications and set security authentication parameters 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);
  ```

- ZooKeeper authentication:

  The sample projects of Spark Streaming, accessing Spark SQL applications through JDBC, and Spark on HBase do not only require basic security

authentication, but also need to add the principal of the ZooKeeper server to complete security authentication. Add the following code to the applications and set security authentication parameters 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.hadoop.com";
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);
```

Security Authentication Code (Scala)

Currently, the sample code invokes the LoginUtil class for security authentication in a unified manner.

In the Spark sample project code, different sample projects use different authentication codes. Basic security authentication or ZooKeeper authentication is used. The following table describes the example authentication parameters used in the sample project. Change the parameter values based on the site requirements.

Table 14-6 Parameters

Parameter	Example Value	Description
userPrincipal	sparkuser	Principal account used for authentication. You can obtain the account from the administrator.
userKeytabPath	/opt/FIclient/user.keytab	Keytab file used for authentication. You can obtain the file from the administrator.
krb5ConfPath	/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf	Path and name of the **krb5.conf** file
ZKServerPrincipal	zookeeper/hadoop.hadoop.com	Principal of the ZooKeeper server. Contact the administrator to obtain the account.

- Basic security authentication:

  Spark Core and Spark SQL applications do not need to access HBase or ZooKeeper. They need only the basic authentication code. Add the following code to the applications and set security authentication parameters 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);
```

- ZooKeeper authentication:

  The sample projects of Spark Streaming, accessing Spark SQL applications through JDBC, and Spark on HBase do not only require basic security authentication, but also need to add the principal of the ZooKeeper server to complete security authentication. Add the following code to the applications and set security authentication parameters 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.hadoop.com"
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);
```

14.3 Developing a Spark Application

14.3.1 Spark Core Application

14.3.1.1 Scenario Description

Scenario Description

Develop a Spark application to perform the following operations on logs about netizens' dwell time for online shopping on a weekend.

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours on the 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 columns are separated by comma (,).

  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 Planning

Save the original log files in the HDFS.

1. Create two text files **input_data1.txt** and **input_data2.txt** on a local computer, and copy **log1.txt** to **input_data1.txt** and **log2.txt** to **input_data2.txt**.
2. Create the **/tmp/input** folder in the HDFS, and run the following commands to upload **input_data1.txt** and **input_data2.txt** to the **/tmp/input** directory:
   a. On the HDFS client, run the following commands for authentication:

      cd /opt/client

      kinit -kt '/opt/client/Spark/spark/conf/user.keytab' <Service user for authentication>

      NOTE

      Specify the path of the **user.keytab** file based on the site requirements.

   b. On the HDFS client running the Linux OS, run the ***hadoop fs -mkdir /tmp/input*** command (or the **hdfs dfs** command) to create a directory.
   c. On the HDFS client running the Linux OS, run the ***hadoop fs -put*** *input_xxx.txt /tmp/input* command to upload the data file.

Development Guidelines

Collect statistics on female netizens who dwell on online shopping for more than 2 hours on the weekend.

To achieve the objective, the process is as follows:

- Read original file data.
- Filter data information of the time that female netizens spend online.
- Summarize the total time that each female netizen spends online.
- Filter the information of female netizens who spend more than 2 hours online.

14.3.1.2 Java Sample Code

Function Description

Collect statistics on female netizens who dwell on online shopping for more than 2 hours on the weekend.

Sample Code

The following code snippets are used as an example. For complete codes, see the **com.huawei.bigdata.spark.examples.FemaleInfoCollection** class.

```
// Create a configuration class SparkConf, and then create a SparkContext.
SparkConf conf = new SparkConf().setAppName("CollectFemaleInfo");
JavaSparkContext jsc = new JavaSparkContext(conf);

// Read the original file data, and transfer each row of records to an element of the RDD.
JavaRDD<String> data = jsc.textFile(args[0]);

// Split each column of each record to generate a Tuple.
JavaRDD<Tuple3<String,String,Integer>> person = data.map(new
Function<String,Tuple3<String,String,Integer>>()
{
    private static final long serialVersionUID = -2381522520231963249L;

    @Override
    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;

    @Override
    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;
    }
});

// Summarize 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;

    @Override
    public Tuple2<String, Integer> call(Tuple3<String, String, Integer> female) throws Exception
    {
        // Extract the two columns representing the name and dwell duration for the sum of dwell duration
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;

    @Override
    public Integer call(Integer integer, Integer integer2) throws Exception
    {
```



```
            // Sum the two dwell durations of the same female netizen.
            return (integer + integer2);
        }
    });

    // Filter the information of 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;

        @Override
        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());
    }
```

14.3.1.3 Scala Sample Code

Function Description

Collect statistics on female netizens who dwell on online shopping for more than 2 hours on the weekend.

Sample Code

The following code snippets are used as an example. For complete codes, see the **com.huawei.bigdata.spark.examples.FemaleInfoCollection** class.

```
// Configure the Spark application name.
val conf = new SparkConf().setAppName("CollectFemaleInfo")

// Submit a Spark job.
val sc = new SparkContext(conf)
// Reads data. The input parameter args(0) specifies the data path.
val text = sc.textFile(args(0))
// Filter data information of the time that female netizens spend online.
val data = text.filter(_.contains("female"))
// Summarize the total 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 of female netizens who spend more than 2 hours online and output the result.
val result = femaleData.filter(line => line._2 > 120)
result.foreach(println)
```


14.3.1.4 Python Sample Code

Function Description

Collect statistics on female netizens who dwell on online shopping for more than 2 hours on the weekend.

Sample Code

The following code snippets are used as an example. For complete codes, 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)

  # Create SparkContext and set AppName.
  sc = SparkContext(appName = "CollectFemaleInfo")?

  """
The following programs are used to implement the following functions:
  //1. Read data. The input parameter argv[1] specifies the data path. - textFile
  2. Filter data information of the time that female netizens spend online. - filter
  3. Summarize the total time that each female netizen spends online. -map/map/reduceByKey.
  4. Filter the information of female netizens who spend more than 2 hours online. - filter
  """
  inputPath = sys.argv[1]
  result = sc.textFile(name = inputPath, use_unicode = False) \
    .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.
  sc.stop()
```

14.3.2 Spark SQL Application

14.3.2.1 Scenario Description

Scenario Description

Develop a Spark application to perform the following operations on logs about netizens' dwell time for online shopping on a weekend.

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours on the 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 columns are separated by comma (,).

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 Planning

Save the original log files in the HDFS.

1. Create two text files **input_data1.txt** and **input_data2.txt** on a local computer, and copy **log1.txt** to **input_data1.txt** and **log2.txt** to **input_data2.txt**.
2. Create the **/tmp/input** folder in the HDFS, and run the following commands to upload **input_data1.txt** and **input_data2.txt** to the **/tmp/input** directory:
   a. On the HDFS client, run the following commands for authentication:

      cd /opt/client

      kinit -kt '/opt/client/Spark/spark/conf/user.keytab' <Service user for authentication>

      NOTE

      Specify the path of the **user.keytab** file based on the site requirements.

   b. On the HDFS client running the Linux OS, run the ***hadoop fs -mkdir /tmp/input*** command (or the **hdfs dfs** command) to create a directory.
   c. On the HDFS client running the Linux OS, run the ***hadoop fs -put*** *input_xxx.txt /tmp/input* command to upload the data file.

Development Guidelines

Collect statistics on female netizens who dwell on online shopping for more than 2 hours on the weekend.

To achieve the objective, the process is as follows:

- Create a table and import the log files into the table.
- Filter data information of the time that female netizens spend online.
- Summarize the total time that each female netizen spends online.
- Filter the information of female netizens who spend more than 2 hours online.

14.3.2.2 Java Sample Code

Function Description

Collect statistics on female netizens who dwell on online shopping for more than 2 hours on the weekend.

Sample Code

The following code snippets are used as an example. For complete codes, see the **com.huawei.bigdata.spark.examples.FemaleInfoCollection** class.

```
SparkConf conf = new SparkConf().setAppName("CollectFemaleInfo");
JavaSparkContext jsc = new JavaSparkContext(conf);
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(jsc);

// Convert RDD to DataFrame through the implicit conversion.
JavaRDD<FemaleInfo> femaleInfoJavaRDD = jsc.textFile(args[0]).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 a table.
DataFrame schemaFemaleInfo = sqlContext.createDataFrame(femaleInfoJavaRDD,FemaleInfo.class);
schemaFemaleInfo.registerTempTable("FemaleInfoTable");

// Execute an SQL query.
DataFrame femaleTimeInfo = sqlContext.sql("select * from " +
        "(select name,sum(stayTime) as totalStayTime from FemaleInfoTable " +
        "where gender = 'female' group by name )" +
        " tmp where totalStayTime >120");

// Display 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);
jsc.stop();
```

For details about the code of other Spark SQL features, visit **http://spark.apache.org/docs/latest/sql-programming-guide.html#running-sql-queries-programmatically**.

14.3.2.3 Scala Sample Code

Function Description

Collect statistics on female netizens who dwell on online shopping for more than 2 hours on the weekend.

Sample Code

The following code snippets are used as an example. For complete codes, see the **com.huawei.bigdata.spark.examples.FemaleInfoCollection** class.

```
object CollectFemaleInfo {
  // Table structure, used for mapping the text data to df
  case class FemaleInfo(name: String, gender: String, stayTime: Int)
  def main(args: Array[String]) {
    // Configure the Spark application name.
    val sparkConf = new SparkConf().setAppName("FemaleInfo")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new org.apache.spark.sql.SQLContext(sc)
    import sqlContext.implicits._
   // Convert RDD to DataFrame through the implicit conversion, then register a table.
    sc.textFile(args(0)).map(_.split(","))
      .map(p => FemaleInfo(p(0), p(1), p(2).trim.toInt))
      .toDF.registerTempTable("FemaleInfoTable")
    // Use SQL statements to filter female netizens' dwell duration data and aggregate data of the same
name.
    val femaleTimeInfo = sqlContext.sql("select name,sum(stayTime) as stayTime from FemaleInfoTable
where
?gender = 'female' group by name")
    // Filter the information of female netizens who spend more than 2 hours online and output the result.
    val c = femaleTimeInfo.filter("stayTime >= 120").collect()
    c.foreach(println)
    sc.stop()
  }
}
```

For details about the code of other Spark SQL features, visit **http://spark.apache.org/docs/latest/sql-programming-guide.html#running-sql-queries-programmatically**.

14.3.3 Spark Streaming Application

14.3.3.1 Scenario Description

Scenario Description

Develop a Spark application to perform the following operations on logs about netizens' dwell time for online shopping on a weekend.

- Collect statistics on female netizens who continuously dwell on online shopping for more than half an hour in real time.
- 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 columns are separated by comma (,).

  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 Planning

The data of the Spark Streaming sample project is stored in the Kafka component. A user with the Kafka permission is required.

1. Create two text files **input_data1.txt** and **input_data2.txt** on a local computer, and copy **log1.txt** to **input_data1.txt** and **log2.txt** to **input_data2.txt**.
2. Create the **/home/data** directory on the client installation node. Upload the preceding two files to the **/home/data** directory.
3. Set **allow.everyone.if.no.acl.found** of Kafka Broker to **true**. (This parameter does not need to be set for the normal cluster.)
4. Start the Producer of the sample code to send data to Kafka.

   java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/streamingClient/*:{JAR_PATH} com.huawei.bigdata.spark.examples.StreamingExampleProducer {BrokerList} {Topic}

   NOTE

   - **JAR_PATH** indicates the path of the JAR package.
   - The format of **brokerlist** is **brokerIp:9092**.

Development Guidelines

Collect statistics on female netizens who dwell on online shopping for more than half an hour on the weekend.

To achieve the objective, the process is as follows:

- Receive data from Kafka and generate the corresponding DStream.
- Filter data information of the time that female netizens spend online.

- Summarize the total time that each female netizen spends online within a time window.
- Filter data about netizens whose consecutive online duration exceeds the threshold, and obtain the results.

14.3.3.2 Java Sample Code

Function Description

Collect statistics on female netizens who continuously dwell on online shopping for more than half an hour in real time. Print statistics directly or output statistics and write them to Kafka.

Spark Streaming Write To Print Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.FemaleInfoCollectionPrint**.

```
   // Parameter description:
   // <batchTime>: Interval for Streaming processing in batches.
   // <windowTime> is the time span of the statistics data. The unit is second.
   // <topics>: Topics subscribed in the Kafka. Multiple topics are separated by commas (,).
   // <brokers> is the Kafka address for obtaining metadata.
public class FemaleInfoCollectionPrint {
   public static void main(String[] args) throws Exception {

      String batchTime = args[0];
      final String windowTime = args[1];
      String topics = args[2];
      String brokers = args[3];

      Duration batchDuration = Durations.seconds(Integer.parseInt(batchTime));
      Duration windowDuration = Durations.seconds(Integer.parseInt(windowTime));


      SparkConf conf = new SparkConf().setAppName("DataSightStreamingExample");
      JavaStreamingContext jssc = new JavaStreamingContext(conf, batchDuration);

      // Set the CheckPoint directory of Streaming. This parameter is mandatory because the window
concept exists.
      jssc.checkpoint("checkpoint");

      // Assemble a Kafka topic list.
      HashSet<String> topicsSet = new HashSet<String>(Arrays.asList(topics.split(",")));
      HashMap<String, String> kafkaParams = new HashMap<String, String>();
      kafkaParams.put("metadata.broker.list", brokers);

      // Create a kafka stream by using brokers and topics.
      // 1. Receive data from Kafka and generate the corresponding DStream.
      JavaDStream<String> lines = KafkaUtils.createDirectStream(jssc,String.class,String.class,
            StringDecoder.class,  StringDecoder.class, kafkaParams, topicsSet).map(
            new Function<Tuple2<String, String>, String>() {
               @Override
               public String call(Tuple2<String, String> tuple2) {
                  return tuple2._2();
               }
            }
      );

      // 2. Obtain the field attribute of each row.
      JavaDStream<Tuple3<String, String, Integer>> records = lines.map(
            new Function<String, Tuple3<String, String, Integer>>() {
               @Override
               public Tuple3<String, String, Integer> call(String line) throws Exception {
```



```
                String[] elems = line.split(",");
                return new Tuple3<String, String, Integer>(elems[0], elems[1], Integer.parseInt(elems[2]));
            }
        }
    );

    // 3. Filter data information of the time that female netizens spend online.
    JavaDStream<Tuple2<String, Integer>> femaleRecords = records.filter(new Function<Tuple3<String,
String, Integer>, Boolean>() {
        public Boolean call(Tuple3<String, String, Integer> line) throws Exception {
            if (line._2().equals("female")) {
                return true;
            } else {
                return false;
            }
        }
    }).map(new Function<Tuple3<String, String, Integer>, Tuple2<String, Integer>>() {
        public Tuple2<String, Integer> call(Tuple3<String, String, Integer> stringStringIntegerTuple3) throws
Exception {
            return new Tuple2<String, Integer>(stringStringIntegerTuple3._1(),
stringStringIntegerTuple3._3());
        }
    });

    // 4. Summarize the total time that each female netizen spends online within a time window.
    JavaPairDStream<String, Integer> aggregateRecords =
JavaPairDStream.fromJavaDStream(femaleRecords)
            .reduceByKeyAndWindow(new Function2<Integer, Integer, Integer>() {
                public Integer call(Integer integer, Integer integer2) throws Exception {
                    return integer + integer2;
                }
            }, new Function2<Integer, Integer, Integer>() {
                public Integer call(Integer integer, Integer integer2) throws Exception {
                    return integer - integer2;
                }
            }, windowDuration, batchDuration);


    JavaPairDStream<String, Integer> upTimeUser = aggregateRecords.filter(new Function<Tuple2<String,
Integer>, Boolean>() {
        public Boolean call(Tuple2<String, Integer> stringIntegerTuple2) throws Exception {
            if (stringIntegerTuple2._2() > 0.9 * Integer.parseInt(windowTime)) {
                return true;
            } else {
                return false;
            }
        }
    });

    // 5. Filter data about netizens whose consecutive online duration exceeds the threshold, and obtain
the results.
    upTimeUser.print();

    // 6. Start Streaming.
    jssc.start();
    jssc.awaitTermination();

  }
```

Spark Streaming Write To Kafka Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.JavaDstreamKafkaWriter**.

NOTE

- After the Spark is upgraded, the new API createDirectStream is recommended. The old API createStream still exists, but the performance and stability are poor. You are advised not to use the old API to develop applications.
- The sample code exists only in **mrs-sample-project-1.6.0.zip**.

```
    // Parameter description:
  //<groupId> Consumer's group.id
  //<brokers> IP address and port number of the broker
   //<topic> Topic of Kafka
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");

  // Configure Kafka.
  Properties kafkaParams = new Properties();
  kafkaParams.put("metadata.broker.list", brokers);
  kafkaParams.put("group.id", groupId);
  kafkaParams.put("auto.offset.reset", "smallest");

  // Create Java streaming context.
  JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.milliseconds(500));

  // Send data 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 an RDD queue.
  Queue<JavaRDD<String>> sent = new LinkedList();
  sent.add(ssc.sparkContext().parallelize(sentData));

 // Use the written data to create Dstream.
  JavaDStream wStream = ssc.queueStream(sent);

  // Write data to Kafka.
  JavaDStreamKafkaWriterFactory.fromJavaDStream(wStream).writeToKafka(kafkaParams,
      new Function<String, KeyedMessage<String, byte[]>>() {
          public KeyedMessage<String, byte[]> call(String s) {
            return new KeyedMessage(topic, s.getBytes());
          }
  });

  ssc.start();
  ssc.awaitTermination();
 }
}
```


14.3.3.3 Scala Sample Code

Function Description

Collect statistics on female netizens who continuously dwell on online shopping for more than half an hour in real time. Print statistics directly or output statistics and write them to Kafka.

Spark Streaming Write To Print Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.FemaleInfoCollectionPrint**.

```
  // Parameter description:
  // <batchTime>: Interval for Streaming processing in batches.
  // <windowTime> is the time span of the statistics data. The unit is second.
  // <topics>: Topics subscribed in the Kafka. Multiple topics are separated by commas (,).
  // <brokers> is the Kafka address for obtaining metadata.
  val Array(batchTime, windowTime, topics, brokers) = args
  val batchDuration = Seconds(batchTime.toInt)
  val windowDuration = Seconds(windowTime.toInt)

  // Set up a Streaming startup environment.
  val sparkConf = new SparkConf()
  sparkConf.setAppName("DataSightStreamingExample")
  val ssc = new StreamingContext(sparkConf, batchDuration)

  // Set the CheckPoint directory of Streaming. This parameter is mandatory because the window concept exists.
  ssc.checkpoint("checkpoint")

  // Assemble a Kafka topic list.
   val topicsSet = topics.split(",").toSet

  // Create a kafka stream by using brokers and topics.
  // 1. Receive data from Kafka and generate the corresponding DStream.
  val kafkaParams = Map[String, String]("metadata.broker.list" -> brokers)
  val lines = KafkaUtils.createDirectStream[String, String, StringDecoder, StringDecoder](
    ssc, kafkaParams, topicsSet).map(_._2)

  // 2. Obtain the field attribute of each row.
  val records = lines.map(getRecord)

  // 3. Filter data information of the time that female netizens spend online.
  val femaleRecords = records.filter(_._2 == "female")
    .map(x => (x._1, x._3))

  // 4. Summarize the total time that each female netizen spends online within a time window.
  val aggregateRecords = femaleRecords
    .reduceByKeyAndWindow(_ + _, _ - _, windowDuration)

  // 5. Filter data about netizens whose consecutive online duration exceeds the threshold, and obtain the results.
  aggregateRecords.filter(_._2 > 0.9 * windowTime.toInt).print()

  // 6. Start Streaming.
  ssc.start()
  ssc.awaitTermination()
```

The preceding code cites the following functions:

```
 // Obtain field functions.
 def getRecord(line: String): (String, String, Int) = {
   val elems = line.split(",")
   val name = elems(0)
   val sexy = elems(1)
```



```
  val time = elems(2).toInt
  (name, sexy, time)
}
```

Spark Streaming Write To Kafka Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.DstreamKafkaWriter**.

NOTE

- After the Spark is upgraded, the new API createDirectStream is recommended. The old API createStream still exists, but the performance and stability are poor. You are advised not to use the old API to develop applications.
- The sample code exists only in **mrs-sample-project-1.6.0.zip**.

```
// Parameter description:
//<groupId> Consumer's group.id
//<brokers> IP address and port number of the broker
 //<topic> Topic of Kafka
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")

// Configure Kafka.
val kafkaParams = new Properties()
kafkaParams.put("metadata.broker.list", brokers)
kafkaParams.put("group.id", groupId)
kafkaParams.put("auto.offset.reset", "smallest")

// Create Java streaming context.
val ssc = new StreamingContext(sparkConf, Milliseconds(500))

// Send data to Kafka.
val sentData = Seq("kafka_writer_test_msg_01", "kafka_writer_test_msg_02",
  "kafka_writer_test_msg_03")

// Create an RDD queue.
val sent = new mutable.Queue[RDD[String]]()
sent.enqueue(ssc.sparkContext.makeRDD(sentData))

// Use the written data to create Dstream.
val wStream = ssc.queueStream(sent)

// Write data to Kafka.
wStream.writeToKafka(kafkaParams,
  (x: String) => new KeyedMessage[String, Array[Byte]](topic, x.getBytes))

ssc.start()
ssc.awaitTermination()
```

14.3.4 Application for Accessing Spark SQL Through JDBC

14.3.4.1 Scenario Description

Scenario Description

Users can customize JDBCServer clients and use JDBC connections to create, load data to, query, and delete data tables.

Data Planning

Step 1 Ensure that the JDBCServer service is started in HA mode and at least one instance provides services for external systems. Create the **/home/data** directory on HDFS, add the files that contain the following content, and upload them to the **/home/data** directory on the HDFS.

```
Miranda,32
Karlie,23
Candice,27
```

Step 2 Ensure that the user who starts JDBCServer has permissions to read and write the file.

Step 3 Ensure that the **hive-site.xml** file exists in **$SPARK_HOME/conf**, and set related parameters based on the actual cluster conditions.

Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration>
   <property>
      <name>spark.thriftserver.ha.enabled</name>
      <value>true</value>
   </property>
</configuration>
```

Step 4 Change the value of **principal** in the **ThriftServerQueriesTest** class to the value of **spark.beeline.principal** in the **$SPARK_HOME/conf/spark-default.conf** configuration file of the cluster.

----End

Development Guidelines

1. Create the **child** table in the **default** database.
2. Load data in **/home/data** to the **child** table.
3. Query data in the **child** table.
4. Delete the **child** table.

14.3.4.2 Java Sample Code

Function Description

The JDBC API of the user-defined client is used to submit a data analysis task and return the results.

Sample Code

Step 1 Define an SQL statement. SQL must be a single statement and cannot contain ";". 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 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 in the **home** directory of the HDFS.
- Ensure that the user and user group of the **data** file are the same as those of the created table.

Step 2 Build JDBC URL.

NOTE

In HA mode, the host and port of the URL must be **ha-cluster**.

For a normal cluster, you need to change the 67th and 68th lines (as shown in the following) of the **com.huawei.bigdata.spark.examples.ThriftServerQueriesTest** class in the sample code from

StringBuilder sb = new StringBuilder("jdbc:hive2://ha-cluster/default"

+ securityConfig);

to **StringBuilder sb = new StringBuilder("jdbc:hive2://ha-cluster/default")**.

```
String HA_CLUSTER_URL = "ha-cluster";
StringBuilder sb = new StringBuilder("jdbc:hive2://" + HA_CLUSTER_URL + "/default;");
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 HiveQL statement, return the queried column name and results to the console, and close the JDBC connection.

In network congestion, configure a timeout interval for a connection between the client and JDBCServer to avoid a client suspension due to timeless wait of the return result from the server. The configuration method is as follows:

Before using the **DriverManager.getConnection** method to obtain the JDBC connection, add the **DriverManager.setLoginTimeout(n)** method to configure a timeout interval. **n** indicates the timeout interval for waiting for the return result from the server. The unit is second, the type is **Int**, and the default value is **0** (indicating never timing out).

```
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

14.3.4.3 Scala Sample Code

Function Description

The JDBC API of the user-defined client is used to submit a data analysis task and return the results.

Sample Code

Step 1 Define an SQL statement. SQL must be a single statement and cannot contain ";". Example:

```
val sqlList = new ArrayBuffer[String]
sqlList += "CREATE TABLE CHILD (NAME STRING, AGE INT) " +
"ROW FORMAT DELIMITED FIELDS TERMINATED BY ','"
sqlList += "LOAD DATA 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 in the **home** directory of the host where the JDBCServer is located.
- Ensure that the user and user group of the local **data** file are the same as those of the created table.

Step 2 Build JDBC URL.

NOTE

In HA mode, the host and port of the URL must be **ha-cluster**.

For a normal cluster, you need to change the 61st and 62nd lines (as shown in the following) of **com.huawei.bigdata.spark.examples.ThriftServerQueriesTest.scala** in the sample code from

val sb = new StringBuilder("jdbc:hive2://ha-cluster/default"

+ securityConfig)

to **val sb = new StringBuilder("jdbc:hive2://ha-cluster/default")**.

```
val HA_CLUSTER_URL = "ha-cluster"
val sb = new StringBuilder(s"jdbc:hive2://$HA_CLUSTER_URL/default;")
val 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 HiveQL statement, return the queried column name and results to the console, and close the JDBC connection.

In network congestion, configure a timeout interval for a connection between the client and JDBCServer to avoid a client suspension due to timeless wait of the return result from the server. The configuration method is as follows:

Before using the **DriverManager.getConnection** method to obtain the JDBC connection, add the **DriverManager.setLoginTimeout(n)** method to configure a timeout interval. **n** indicates the timeout interval for waiting for the return result from the server. The unit is second, the type is **Int**, and the default value is **0** (indicating never timing out).

```
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

14.3.4.4 Python Sample Code

Function Description

The IP address and port number of the current active JDBCServer can be obtained by connecting the znode on ZooKeeper and the JDBCServer is connected through PyHive. Thereby, after an active/standby switchover, the new active JDBCServer service can be directly accessed without code modification in the JDBCServer-HA mode.

This function applies only to common clusters (clusters with Kerberos authentication disabled).

Environment Preparation

1. Install the support environment. (For details, see **Spark Application Development Environment**.)

   Run the following commands to install the compilation tools:

   yum install cyrus-sasl-devel -y

   yum install gcc-c++ -y

2. Install the Python modules, including

   SASL, Thrift, Thrift-SASL, and PyHive.

   pip install sasl

   pip install thrift

   pip install thrift-sasl

   pip install PyHive

3. Install the Python tool for connecting to ZooKeeper.

   pip install kazoo

4. Obtain related parameters from the MRS cluster.
   - To obtain the IP address and port number of the ZooKeeper:

     View the configuration item **spark.deploy.zookeeper.url** in the configuration file **/opt/client/Spark/spark/conf/hive-site.xml**.
   - To obtain the IP address and port number of the active JDBCServer node stored in the ZooKeeper:

     View the configuration item **spark.thriftserver.zookeeper.dir** (**/thriftserver** by default) in the configuration file **/opt/client/Spark/spark/conf/hive-site.xml**. The IP address and port number of the active JDBCServer node are stored on the znode subnode (**active_thriftserver**).

Sample Code

```
from kazoo.client import KazooClient
zk = KazooClient(hosts='ZookeeperHost')
zk.start()
result=zk.get("/thriftserver/active_thriftserver")
result=result[0].decode('utf-8')
JDBCServerHost=result[0].split(":")[0]
JDBCServerPort=result[0].split(":")[1]
from pyhive import hive
conn = hive.Connection(host=JDBCServerHost, port=JDBCServerPort,database='default')
cursor=conn.cursor()
cursor.execute("select * from test")
for result in cursor.fetchall():
    print result
```

Replace **ZookeeperHost** with the ZooKeeper IP address and port number obtained in **4**.

14.3.5 Spark on HBase Application

14.3.5.1 Scenario Description

Scenario Description

Users can use Spark to call HBase APIs to operate HBase tables. In the Spark applications, users can use HBase APIs to create a table, read the table, and insert data into the table.

Data Planning

Save the original data files in HDFS.

1. Create the **input_data1.txt** text file on the local PC and copy the following content to the **input_data1.txt** file.
   ```
   20,30,40,xxx
   ```
2. Create the **/tmp/input** folder in the HDFS, and run the following commands to upload **input_data1.txt** to the **/tmp/input** directory:
   a. On the HDFS client, run the following commands for authentication:

      cd /opt/client

      kinit -kt '/opt/client/Spark/spark/conf/user.keytab' <Service user for authentication>

      NOTE

      Specify the path of the **user.keytab** file based on the site requirements.

   b. On the HDFS client running the Linux OS, run the ***hadoop fs -mkdir /tmp/input*** command (or the **hdfs dfs** command) to create a directory.
   c. On the HDFS client running the Linux OS, run the ***hadoop fs -put*** *input_xxx.txt /tmp/input* command to upload the data file.

NOTE

If Kerberos authentication is enabled, set **spark.yarn.security.credentials.hbase.enabled** in the client configuration file **spark-defaults.conf** to **true**.

Development Guidelines

1. Create an HBase table.
2. Insert data to the HBase table.
3. Use Spark Application to read data from the HBase table.

14.3.5.2 Java Sample Code

Function Description

In the Spark applications, users can use HBase APIs to create a table, read the table, and insert data into the table.

Sample Code

The following code snippets are used as an example. For complete codes, see **SparkOnHbaseJavaExample**.

Example: Creating an HBase table

```
public class TableCreation {
   public static void main (String[] args) throws IOException {

      SparkConf conf = new SparkConf().setAppName("CollectFemaleInfo");
      JavaSparkContext jsc = new JavaSparkContext(conf);
      Configuration hbConf = HBaseConfiguration.create(jsc.hadoopConfiguration());

      // Create a connection channel to connect to HBase.
      Connection connection = ConnectionFactory.createConnection(hbConf);

      // Declare table description.
      TableName userTable  = TableName.valueOf("shb1");
      HTableDescriptor tableDescr = new HTableDescriptor(userTable);
      tableDescr.addFamily(new HColumnDescriptor("info".getBytes()));

      //Create a table.
      System.out.println("Creating table shb1. ");
      Admin admin = connection.getAdmin();
      if (admin.tableExists(userTable)) {
         admin.disableTable(userTable);
         admin.deleteTable(userTable);
      }
      admin.createTable(tableDescr);

      connection.close();
      jsc.stop();
      System.out.println("Done!");

   }
}
```

Example: Inserting data into the HBase table

```
public class TableInputData {
   public static void main (String[] args) throws IOException {

      // Create a configuration parameter to connect to HBase and ensure that hbase-site.xml is in
classpath.
      SparkConf conf = new SparkConf().setAppName("CollectFemaleInfo");
      JavaSparkContext jsc = new JavaSparkContext(conf);
      Configuration hbConf = HBaseConfiguration.create(jsc.hadoopConfiguration());

      // Declare table information.
      Table table = null;
      String tableName = "shb1";
      byte[] familyName = Bytes.toBytes("info");
      Connection connection = null;

      try {
         // Obtain the HBase connection.
         connection = ConnectionFactory.createConnection(hbConf);
         // Obtain the table object.
         table = connection.getTable(TableName.valueOf(tableName));
         List<Tuple4<String, String, String, String>> data = jsc.textFile(args[0]).map(
               new Function<String, Tuple4<String, String, String, String>>() {
                  @Override
                  public Tuple4<String, String, String, String> call(String s) throws Exception {
                     String[] tokens = s.split(",");

                     return new Tuple4<String, String, String, String>(tokens[0], tokens[1], tokens[2],
tokens[3]);
                  }
```



```
                }).collect();

            Integer i = 0;
            for(Tuple4<String, String, String, String> line:  data) {
                Put put = new Put(Bytes.toBytes("row" + i));
                put.addColumn(familyName, Bytes.toBytes("c11"), Bytes.toBytes(line._1()));
                put.addColumn(familyName, Bytes.toBytes("c12"), Bytes.toBytes(line._2()));
                put.addColumn(familyName, Bytes.toBytes("c13"), Bytes.toBytes(line._3()));
                put.addColumn(familyName, Bytes.toBytes("c14"), Bytes.toBytes(line._4()));
                i += 1;
                table.put(put);
            }

        } 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();
                 }
             }
             jsc.stop();
        }
    }
}
```

Example: Reading HBase table data

```
public class TableOutputData {
    public static void main(String[] args) throws IOException {

        System.setProperty("spark.serializer", "org.apache.spark.serializer.KryoSerializer");
        System.setProperty("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrator");

        // Create a configuration parameter to connect to HBase and ensure that hbase-site.xml is in
classpath.
        SparkConf conf = new SparkConf().setAppName("CollectFemaleInfo");
        JavaSparkContext jsc = new JavaSparkContext(conf);
        Configuration hbConf = HBaseConfiguration.create(jsc.hadoopConfiguration());

        //Declare information about the table to be queried.
        Scan scan = new org.apache.hadoop.hbase.client.Scan();
        scan.addFamily(Bytes.toBytes("info"));
        org.apache.hadoop.hbase.protobuf.generated.ClientProtos.Scan proto = ProtobufUtil.toScan(scan);
        String scanToString = Base64.encodeBytes(proto.toByteArray());
        hbConf.set(TableInputFormat.INPUT_TABLE, "shb1");
        hbConf.set(TableInputFormat.SCAN, scanToString);

        // Use the Spark API to obtain table data.
        JavaPairRDD rdd = jsc.newAPIHadoopRDD(hbConf, TableInputFormat.class,
ImmutableBytesWritable.class, Result.class);

        // Traverse every row in the HBase table and print the results.
        List<Tuple2<ImmutableBytesWritable, Result>> rddList = rdd.collect();
        for (int i = 0; i < rddList.size(); i++) {
            Tuple2<ImmutableBytesWritable, Result> t2 = rddList.get(i);
            ImmutableBytesWritable key = t2._1();
            Iterator<Cell> it = t2._2().listCells().iterator();
            while (it.hasNext()) {
                Cell c = it.next();
```



```
            String family = Bytes.toString(CellUtil.cloneFamily(c));
            String qualifier = Bytes.toString(CellUtil.cloneQualifier(c));
            String value = Bytes.toString(CellUtil.cloneValue(c));
            Long tm = c.getTimestamp();
            System.out.println(" Family=" + family + " Qualifier=" + qualifier + " Value=" + value + "
TimeStamp=" + tm);
          }
        }
        jsc.stop();
    }
}
```

14.3.5.3 Scala Sample Code

Function Description

In the Spark applications, users can use HBase APIs to create a table, read the table, and insert data into the table.

Sample Code

The following code snippets are used as an example. For complete codes, see **SparkOnHbaseScalaExample**.

Example: Creating an HBase table

```
// Create a configuration parameter to connect to HBase and ensure that hbase-site.xml is in classpath.
val conf: SparkConf = new SparkConf
val sc: SparkContext = new SparkContext(conf)
val hbConf: Configuration = HBaseConfiguration.create(sc.hadoopConfiguration)
// Create a connection channel to connect to HBase.
val connection: Connection = ConnectionFactory.createConnection(hbConf)

// Declare table description.
val userTable = TableName.valueOf("shb1")
val tableDescr = new HTableDescriptor(userTable)
tableDescr.addFamily(new HColumnDescriptor("info".getBytes))

// Create a table.
println("Creating table shb1. ")
val admin = connection.getAdmin
if (admin.tableExists(userTable)) {
  admin.disableTable(userTable)
  admin.deleteTable(userTable)
}
admin.createTable(tableDescr)

connection.close()
sc.stop()
println("Done!")
```

Example: Inserting data into the HBase table

```
// Create a configuration parameter to connect to HBase and ensure that hbase-site.xml is in classpath.
val conf = new SparkConf()
val sc = new SparkContext(conf)
val hbConf = HBaseConfiguration.create(sc.hadoopConfiguration)

// Declare table information.
val table: HTable = null
val tableName = "shb1"
val familyName = Bytes.toBytes("info");
var connection: Connection = null
try {
  // Obtain the HBase connection.
  connection = ConnectionFactory.createConnection(hbConf);
```



```
    // Obtain the table object.
    val table = connection.getTable(TableName.valueOf(tableName));
    val data = sc.textFile(args(0)).map { line =>
      val value = line.split(",")
      (value(0), value(1), value(2), value(3))
    }.collect()

    var i = 0
    for (line <- data) {
      val put = new Put(Bytes.toBytes("row" + i));
      put.addColumn(familyName, Bytes.toBytes("c11"), Bytes.toBytes(line._1))
      put.addColumn(familyName, Bytes.toBytes("c12"), Bytes.toBytes(line._2))
      put.addColumn(familyName, Bytes.toBytes("c13"), Bytes.toBytes(line._3))
      put.addColumn(familyName, Bytes.toBytes("c14"), Bytes.toBytes(line._4))
      i += 1
      table.put(put)
    }
  } catch {
    case e: IOException =>
      e.printStackTrace();
  } finally {
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch {
        case e: IOException =>
          e.printStackTrace();
      }
    }
    if (connection != null) {
      try {
        // Close the HBase connection.
        connection.close();
      } catch {
        case e: IOException =>
          e.printStackTrace();
      }
    }
     sc.stop()
  }
```

Example: Reading HBase table data

```
  // Create a configuration parameter to connect to HBase and ensure that hbase-site.xml is in classpath.
    val conf = new SparkConf()
    val sc = new SparkContext(conf)
    val hbConf = HBaseConfiguration.create(sc.hadoopConfiguration)

    // Declare information about the table to be queried.
    val scan = new Scan()
    scan.addFamily(Bytes.toBytes("info"))
    val proto = ProtobufUtil.toScan(scan)
    val scanToString = Base64.encodeBytes(proto.toByteArray)
    hbConf.set(TableInputFormat.INPUT_TABLE, "shb1")
    hbConf.set(TableInputFormat.SCAN, scanToString)

    // Use the Spark API to obtain table data.
    val rdd = sc.newAPIHadoopRDD(hbConf, classOf[TableInputFormat], classOf[ImmutableBytesWritable],
classOf[Result])

    // Traverse every row in the HBase table and print the results.
    rdd.collect().foreach(x => {
      val key = x._1.toString
      val it = x._2.listCells().iterator()
      while (it.hasNext) {
        val c = it.next()
        val family = Bytes.toString(CellUtil.cloneFamily(c))
        val qualifier = Bytes.toString(CellUtil.cloneQualifier(c))
        val value = Bytes.toString(CellUtil.cloneValue(c))
```



```
    val tm = c.getTimestamp
    println(" Family=" + family + " Qualifier=" + qualifier + " Value=" + value + " TimeStamp=" + tm)
  }
})
sc.stop()
```

14.3.6 Reading Data from HBase and Writing Data Back to HBase

14.3.6.1 Scenario Description

Scenario Description

Assume that table1 of HBase stores a user's consumption amount on the current day and table2 stores the user's history consumption amount data.

In table1, the **key=1,cf:cid=100** record indicates that user1's consumption amount on the current day is 100 CNY.

In table2, the **key=1,cf:cid=1000** record indicates that user1's history consumption amount is 1000 CNY.

Based on some service requirements, a Spark application must be developed to implement the following functions:

Calculate a user's history consumption amount based on the user name, that is, the user's total consumption amount =100 (consumption amount of the current day) + 1000 (history consumption amount).

In the preceding example, the application run result is that in table2, the total consumption amount of user1 (**key=1**) is **cf:cid=1100** CNY.

Data Planning

Use the HBase shell tool to create HBase table1 and table2 and insert data to them.

Step 1 Run the following command to create a table named **table1** through HBase:

create 'table1', 'cf'

Step 2 Run the following command to insert data through HBase:

put 'table1', '1', 'cf:cid', '100'

Step 3 Run the following command to create a table named **table2** through HBase:

create 'table2', 'cf'

Step 4 Run the following command on HBase to insert data into table2:

put 'table2', '1', 'cf:cid', '1000'

NOTE

If Kerberos authentication is enabled, set **spark.yarn.security.credentials.hbase.enabled** in the client configuration file **spark-defaults.conf** and on the sparkJDBC server to **true**.

----End

Development Guidelines

1. Query data in table1.
2. Query data in table2 based on the key value in table1.
3. Sum the data records obtained in the previous two steps.
4. Write the result of the previous step to table2.

14.3.6.2 Java Sample Code

Function Description

Users can use Spark to call an HBase API to operate HBase table1 and write the data analysis result of table1 to HBase table2.

Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.SparkHbasetoHbase**.

```
/**
 * Read data from table1, and obtain the corresponding record from table2 based on the key value. Sum the obtained two data records and update the sum result to  table2.
*/
public class SparkHbasetoHbase {

  public static void main(String[] args) throws Exception {
    if (args.length < 1) {
      printUsage();
    }

    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 a configuration parameter to connect to HBase and ensure that hbase-site.xml is in classpath.
    Configuration hbConf = HBaseConfiguration.create(jsc.hadoopConfiguration());

    // Declare table information.
    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);

    // Use the Spark API to obtain table data.
    JavaPairRDD rdd = jsc.newAPIHadoopRDD(hbConf, TableInputFormat.class,
ImmutableBytesWritable.class, Result.class);

    // Traverse every partition in HBase table1 and update data to HBase table2.
    // If the number of data records is small, you can use the rdd.foreach() method.
    final  String zkQuorum = args[0];
    rdd.foreachPartition(
      new VoidFunction<Iterator<Tuple2<ImmutableBytesWritable, Result>>>() {
        public void call(Iterator<Tuple2<ImmutableBytesWritable, Result>> iterator) throws Exception {
          hBaseWriter(iterator, zkQuorum);
        }
      }
    );

    jsc.stop();
  }
```



```
/**
  * Update records in table2 on the executor.
  *
  * @param iterator Partition data in table1.
  */
 private static void hBaseWriter(Iterator<Tuple2<ImmutableBytesWritable, Result>> iterator, String
zkQuorum) throws IOException {
   // Prepare for reading HBase.
   String tableName = "table2";
   String columnFamily = "cf";
   String qualifier = "cid";
   Configuration conf = HBaseConfiguration.create();
   conf.set("hbase.zookeeper.property.clientPort", "24002");
   conf.set("hbase.zookeeper.quorum", zkQuorum);
   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);
     }
     // Obtain the records in table2.
     Result[] resultDataBuffer = table.get(rowList);
     // Modify records in table2.
     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();
           // Check whether the values of cf and qualifile are the same.
           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 the result.
         int resultValue = Integer.parseInt(hbase1Value) + Integer.parseInt(hbase2Value);
         // Set the result to the Put object.
         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();
      }
    }
  }
 }
 private static void printUsage() {
   System.out.println("Usage: {zkQuorum}");
   System.exit(1);
 }
}
```

14.3.6.3 Scala Sample Code

Function Description

Users can use Spark to call an HBase API to operate HBase table1 and write the data analysis result of table1 to HBase table2.

Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.SparkHbasetoHbase**.

```
/**
  * Read data from table1, and obtain the corresponding record from table2 based on the key value. Sum
the obtained two data records and update the sum result to  table2.
  */
object SparkHbasetoHbase {

  case class FemaleInfo(name: String, gender: String, stayTime: Int)

  def main(args: Array[String]) {
    if (args.length < 1) {
      printUsage
    }

    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 a configuration parameter to connect to HBase and ensure that hbase-site.xml is in classpath.
    val hbConf = HBaseConfiguration.create(sc.hadoopConfiguration)

    // Declare table information.
    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)

    // Use the Spark API to obtain table data.
    val rdd = sc.newAPIHadoopRDD(hbConf, classOf[TableInputFormat], classOf[ImmutableBytesWritable],
classOf[Result])

    // Traverse every partition in HBase table1 and update data to HBase table2.
    // If the number of data records is small, you can use the rdd.foreach() method.
    rdd.foreachPartition(x => hBaseWriter(x, args(0)))
```



```
    sc.stop()
  }
  /**
    * Update records in table2 on the executor.
    *
    * @param iterator Partition data in table1.
    */
  def hBaseWriter(iterator: Iterator[(ImmutableBytesWritable, Result)], zkQuorum: String): Unit = {
    // Prepare for reading HBase.
    val tableName = "table2"
    val columnFamily = "cf"
    val qualifier = "cid"
    val conf = HBaseConfiguration.create()
    conf.set("hbase.zookeeper.property.clientPort", "24002")
    conf.set("hbase.zookeeper.quorum", zkQuorum)
    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)
      }
      // Obtain the records in table2.
      val resultDataBuffer = table.get(rowList)
      // Modify records in table2.
      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()
            // Check whether the values of cf and qualifile are the same.
            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 the result.
          val resultValue = hbase1Value.toInt + hbase2Value.toInt
          // Set the result to the Put object.
          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()
        }
      }
    }
  }
  private def printUsage {
    System.out.println("Usage: {zkQuorum}")
    System.exit(1)
  }
}


/**
  * Sequential auxiliary 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])
  }
}
```

14.3.7 Reading Data from Hive and Write Data to HBase

14.3.7.1 Scenario Description

Scenario Description

Assume that **person** table of Hive stores a user's consumption amount on the current day and HBase table2 stores the user's history consumption amount data.

In the **person** table, the **name=1,account=100** record indicates that user1's consumption amount on the current day is 100 CNY.

In table2, the **key=1,cf:cid=1000** record indicates that user1's history consumption amount is 1000 CNY.

Based on some service requirements, a Spark application must be developed to implement the following functions:

Calculate a user's history consumption amount based on the user name, that is, the user's total consumption amount =100 (consumption amount of the current day) + 1000 (history consumption amount).

In the preceding example, the application run result is that in table2, the total consumption amount of user1 (**key=1**) is **cf:cid=1100** CNY.

Data Planning

Before developing the application, create a Hive table named **person** and insert data to the table. At the same time, create HBase table2 so that you can write the data analysis result to it.

Step 1 Save original log files to HDFS.

1. Create a blank **log1.txt** file on the local PC and write the following content to the file.

   ```
   1,100
   ```

2. Create the **/tmp/input** directory in HDFS and upload the **log1.txt** file to the directory.

   a. On the HDFS client, run the following commands for authentication:

   cd /opt/client

   kinit -kt '/opt/client/Spark/spark/conf/user.keytab' <Service user for authentication>

   NOTE

   Specify the path of the **user.keytab** file based on the site requirements.

   b. On the HDFS client running the Linux OS, run the ***hadoop fs -mkdir /tmp/input*** command (or the **hdfs dfs** command) to create a directory.

   c. On the HDFS client running the Linux OS, run the ***hadoop fs -put*** *log1.txt /tmp/input* command to upload the data file.

Step 2 Store the imported data to the Hive table.

Ensure that the ThriftServer is started. Use the Beeline tool to create a Hive table and insert data to the table.

1. Run the following command to create a Hive table named **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 **person** table:

   load data inpath '/tmp/input/log1.txt' into table person;

Step 3 Create an HBase table.

1. Run the following command to create a table named **table2** through HBase:

   create 'table2', 'cf'

2. Run the following command on HBase to insert data to HBase table2:

   put 'table2', '1', 'cf:cid', '1000'

NOTE

If Kerberos authentication is enabled, set **spark.yarn.security.credentials.hbase.enabled** in the client configuration file **spark-default.conf** and on the sparkJDBC server to **true**.

----End

Development Guidelines

1. Query data in the **person** Hive table.
2. Query data in table2 based on the key value in the **person** table.
3. Sum the data records obtained in the previous two steps.
4. Write the result of the previous step to table2.

14.3.7.2 Java Sample Code

Function Description

In a Spark application, users can 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.

Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.SparkHivetoHbase**.

```
/**
 * Read data from the Hive table, and obtain the corresponding record from the HBase table based on the key value. Sum the obtained two data records and update the sum result to the HBase table.
 */
public class SparkHivetoHbase {

 public static void main(String[] args) throws Exception {
  if (args.length < 1) {
   printUsage();
  }

  // Use the Spark API to obtain table data.
  SparkConf conf = new SparkConf().setAppName("SparkHivetoHbase");
  JavaSparkContext jsc = new JavaSparkContext(conf);
  HiveContext sqlContext = new org.apache.spark.sql.hive.HiveContext(jsc);
  DataFrame dataFrame = sqlContext.sql("select name, account from person");

  // Traverse every partition in the Hive table and update data to the HBase table.
  // If the number of data records is small, you can use the foreach() method.
  final String zkQuorum = args[0];
  dataFrame.toJavaRDD().foreachPartition(
   new VoidFunction<Iterator<Row>>() {
    public void call(Iterator<Row> iterator) throws Exception {
     hBaseWriter(iterator,zkQuorum);
    }
   }
  );

  jsc.stop();
 }

 /**
  * Update records in the HBase table on the executor.
  *
  * @param iterator Partition data in the Hive table.
  */
 private static void hBaseWriter(Iterator<Row> iterator, String zkQuorum) throws IOException {
  // Read the HBase table.
  String tableName = "table2";
  String columnFamily = "cf";
  Configuration conf = HBaseConfiguration.create();
  conf.set("hbase.zookeeper.property.clientPort", "24002");
  conf.set("hbase.zookeeper.quorum", zkQuorum);
  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);
      }
      // Obtain the records in the HBase table.
      Result[] resultDataBuffer = table.get(rowList);
      // Modify records in the HBase table.
      List<Put> putList = new ArrayList<Put>();
      for (int i = 0; i < resultDataBuffer.length; i++) {
        // Hive table value
        Result resultData = resultDataBuffer[i];
        if (!resultData.isEmpty()) {
          // get hiveValue
          int hiveValue = table1List.get(i).getInt(1);
          // Obtain the HBase table value based on the column family and column.
          String hbaseValue = Bytes.toString(resultData.getValue(columnFamily.getBytes(), "cid".getBytes()));
          Put put = new Put(table1List.get(i).getString(0).getBytes());
          // Calculate the result.
          int resultValue = hiveValue + Integer.valueOf(hbaseValue);
          // Set the result to the Put object.
          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();
        }
      }
    }
  }


  private static void printUsage() {
    System.out.println("Usage: {zkQuorum}");
    System.exit(1);
  }
}
```


14.3.7.3 Scala Sample Code

Function Description

In a Spark application, users can 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.

Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.SparkHivetoHbase**.

```
/**
 * Read data from the Hive table, and obtain the corresponding record from the HBase table based on the key value. Sum the obtained two data records and update the sum result to the HBase table.
 */
object SparkHivetoHbase {
 case class FemaleInfo(name: String, gender: String, stayTime: Int)
 def main(args: Array[String]) {
  if (args.length < 1) {
   printUsage
  }
  // Use the Spark API to obtain table data.
  val sparkConf = new SparkConf().setAppName("SparkHivetoHbase")
  val sc = new SparkContext(sparkConf)
  val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
  import sqlContext.implicits._
  val dataFrame = sqlContext.sql("select name, account from person")
  // Traverse every partition in the Hive table and update data to the HBase table.
  // If the number of data records is small, you can use the foreach() method.
  dataFrame.rdd.foreachPartition(x => hBaseWriter(x, args(0)))
  sc.stop()
 }
 /**
  * Update records in the HBase table on the executor.
  *
  * @param iterator Partition data in the Hive table.
  */
 def hBaseWriter(iterator: Iterator[Row], zkQuorum: String): Unit = {
  // Read the HBase table.
  val tableName = "table2"
  val columnFamily = "cf"
  val conf = HBaseConfiguration.create()
  conf.set("hbase.zookeeper.property.clientPort", "24002")
  conf.set("hbase.zookeeper.quorum", zkQuorum)
  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)
   }
   // Obtain the records in the HBase table.
   val resultDataBuffer = table.get(rowList)
   // Modify records in the HBase table.
   val putList = new util.ArrayList[Put]()
   for (i <- 0 until iteratorArray.size) {
    // hbase row
    val resultData = resultDataBuffer(i)
    if (!resultData.isEmpty) {
    // Hive table value
     var hiveValue = iteratorArray(i).getInt(1)
```



```
        // Obtain the HBase table value based on the column family and column.
        val hbaseValue = Bytes.toString(resultData.getValue(columnFamily.getBytes, "cid".getBytes))
        val put = new Put(iteratorArray(i).getString(0).getBytes)
        // Calculate the result.
        val resultValue = hiveValue + hbaseValue.toInt
        // Set the result to the Put object.
        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()
      }
    }
  }
 }

 private def printUsage {
   System.out.println("Usage: {zkQuorum}")
   System.exit(1)
 }
}
```

14.3.8 Using Streaming to Read Data from Kafka and Write Data to HBase

14.3.8.1 Scenario Description

Scenario Description

Assume that Kafka receives the consumption records of five users every 30 seconds in a service. HBase table1 stores users' history consumption amount information.

There are 10 records in **table 1**, indicating that users whose user names are **1** to **10**. All users' initial history consumption amount is 0 CNY.

Based on some service requirements, a Spark application must be developed to implement the following functions:

Calculate a user's consumption amount in real time using the following formula: Total consumption amount = Current consumption amount (Kafka data) + History

consumption amount (value in table1). Then, update the calculation result to table1.

Data Planning

Step 1 Create an HBase table and insert data.

1. Run the following command to create a table named **table1** through HBase:

   create 'table1', 'cf'

2. Run the following command on HBase to insert data into table1:

```
put 'table1', '1', 'cf:cid', '0'
put 'table1', '2', 'cf:cid', '0'
put 'table1', '3', 'cf:cid', '0'
put 'table1', '4', 'cf:cid', '0'
put 'table1', '5', 'cf:cid', '0'
put 'table1', '6', 'cf:cid', '0'
put 'table1', '7', 'cf:cid', '0'
put 'table1', '8', 'cf:cid', '0'
put 'table1', '9', 'cf:cid', '0'
put 'table1', '10', 'cf:cid', '0'
```

Step 2 Data of the Spark Streaming sample project is stored in Kafka.

1. Ensure that the clusters are installed, including HDFS, Yarn, and Spark.
2. Modify **allow.everyone.if.no.acl.found** of Kafka Broker to **true**. (This parameter does not need to be set for the normal cluster.)
3. Create a topic.

   {zkQuorum} indicates ZooKeeper cluster information in the IP:port format.

   $KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper {zkQuorum}/kafka --replication-factor 1 --partitions 3 --topic {Topic}

4. Start the Producer of the sample code to send data to Kafka.

   {ClassPath} indicates the path for storing the JAR file of the project. The path is specified by users. For details about how to export the JAR file, see **Compiling and Running a Spark Application**.

   java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/streamingClient/*:{JAR_PATH} com.huawei.bigdata.spark.examples.streaming.StreamingExampleProducer {BrokerList} {Topic}

NOTE

- If Kerberos authentication is enabled, set **spark.yarn.security.credentials.hbase.enabled** in the client configuration file **spark-default.conf** and on the sparkJDBC server to **true**.
- The format of **{zkQuorum}** is in **zkIp:2181** format.
- **JAR_PATH** indicates the path of the JAR package.
- The value of **BrokerList** is in **brokerIp:9092** format.

----End

Development Guidelines

1. Receive data from Kafka and generate the corresponding DStream.

2. Filter and analyze data.
3. Find the corresponding record in the HBase table.
4. Calculate the result and write the result to the HBase table.

14.3.8.2 Java Sample Code

Function Description

In Spark applications, use Streaming to call Kafka APIs to obtain data and write data analysis results to an HBase table.

Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.SparkOnStreamingToHbase**.

```
/**
 * Run a Streaming job. Read data from HBase table1 based on the value, sum two data records, and
update the new data in the HBase table1.
 */
public class SparkOnStreamingToHbase {
  public static void main(String[] args) throws Exception {
    if (args.length < 4) {
      printUsage();
    }

    String checkPointDir = args[0];
    String topics = args[1];
    final String brokers = args[2];
    final String zkQuorum = args[3];

    Duration batchDuration = Durations.seconds(5);
    SparkConf sparkConf = new SparkConf().setAppName("SparkOnStreamingToHbase");
    JavaStreamingContext jssc = new JavaStreamingContext(sparkConf, batchDuration);

    // Set the CheckPoint directory of Streaming.
    if (!"nocp".equals(checkPointDir)) {
      jssc.checkpoint(checkPointDir);
    }

    final String columnFamily = "cf";
    final String zkClientPort = "24002";
    HashMap<String, String> kafkaParams = new HashMap<String, String>();
    kafkaParams.put("metadata.broker.list", brokers);

    String[] topicArr = topics.split(",");
    Set<String> topicSet = new HashSet<String>(Arrays.asList(topicArr));

    // Create a kafka stream by using brokers and topics.
    // Receive data from Kafka and generate the corresponding DStream.
    JavaDStream<String> lines = KafkaUtils.createDirectStream(jssc, String.class, String.class,
      StringDecoder.class, StringDecoder.class, kafkaParams, topicSet).map(
      new Function<Tuple2<String, String>, String>() {
        public String call(Tuple2<String, String> tuple2) {
          // map(_._1) is the key of the message, and map(_._2) is the value of the message.
          return tuple2._2();
        }
      }
    );

    lines.foreachRDD(
      new Function<JavaRDD<String>, Void>() {
        public Void call(JavaRDD<String> rdd) throws Exception {
          rdd.foreachPartition(
```



```
          new VoidFunction<Iterator<String>>() {
            public void call(Iterator<String> iterator) throws Exception {
              hBaseWriter(iterator, zkClientPort, zkQuorum, columnFamily);
            }
          }
        );
        return null;
      }
    }
  );

  jssc.start();
  jssc.awaitTermination();
 }

 /**
  * Write data to the executor.
  * @param iterator Message
  * @param zkClientPort
  * @param zkQuorum
  * @param columnFamily
  */
 private static void hBaseWriter(Iterator<String> iterator, String zkClientPort, String zkQuorum, String
columnFamily) throws IOException {
   Configuration conf = HBaseConfiguration.create();
   conf.set("hbase.zookeeper.property.clientPort", zkClientPort);
   conf.set("hbase.zookeeper.quorum", zkQuorum);
   Connection connection = null;
   Table table = null;
   try {
     connection = ConnectionFactory.createConnection(conf);
     table = connection.getTable(TableName.valueOf("table1"));
     List<Get> rowList = new ArrayList<Get>();
     while (iterator.hasNext()) {
       Get get = new Get(iterator.next().getBytes());
       rowList.add(get);
     }
     // Obtain data in table1.
     Result[] resultDataBuffer = table.get(rowList);
     // Set data in table1.
     List<Put> putList = new ArrayList<Put>();
     for (int i = 0; i < resultDataBuffer.length; i++) {
       String row = new String(rowList.get(i).getRow());
       Result resultData = resultDataBuffer[i];
       if (!resultData.isEmpty()) {
         // Obtain the old value based on the column family and column.
         String aCid = Bytes.toString(resultData.getValue(columnFamily.getBytes(), "cid".getBytes()));
         Put put = new Put(Bytes.toBytes(row));
         // Calculate the result.
         int resultValue = Integer.valueOf(row) + Integer.valueOf(aCid);
         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();
      }
    }
  }
 }


 private static void printUsage() {
   System.out.println("Usage: {checkPointDir} {topic} {brokerList} {zkQuorum}");
   System.exit(1);
 }
}
```

14.3.8.3 Scala Sample Code

Function Description

In Spark applications, use Streaming to call Kafka APIs to obtain data and write data analysis results to an HBase table.

Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.SparkOnStreamingToHbase**.

```
/**
 * Run a Streaming job. Read data from HBase table1 based on the value, sum two data records, and
update the new data in the HBase table1.
 */
object SparkOnStreamingToHbase {
 def main(args: Array[String]) {
   if (args.length < 4) {
     printUsage
   }

   val Array(checkPointDir, topics, brokers, zkQuorum) = args
   val sparkConf = new SparkConf().setAppName("DirectStreamToHbase")
   val ssc = new StreamingContext(sparkConf, Seconds(5))

   // Set the CheckPoint directory of Streaming.
   if (!"nocp".equals(checkPointDir)) {
     ssc.checkpoint(checkPointDir)
   }

   val columnFamily = "cf"
   val zkClientPort = "24002"
   val kafkaParams = Map[String, String](
     "metadata.broker.list" -> brokers
   )

   val topicArr = topics.split(",")
   val topicSet = topicArr.toSet
   // map(_._1) is the key of the message, and map(_._2) is the value of the message.
   val lines = KafkaUtils.createDirectStream[String, String, StringDecoder, StringDecoder](ssc, kafkaParams,
topicSet).map(_._2)
   lines.foreachRDD(rdd => {
     // Partitions run on the executor.
     rdd.foreachPartition(iterator => hBaseWriter(iterator, zkClientPort, zkQuorum, columnFamily))
   })

   ssc.start()
   ssc.awaitTermination()
```



```
 }


 /**
  * Write data to the executor.
  * @param iterator Message
  * @param zkClientPort
  * @param zkQuorum
  * @param columnFamily
  */
 def hBaseWriter(iterator: Iterator[String], zkClientPort: String, zkQuorum: String, columnFamily: String):
Unit = {
   val conf = HBaseConfiguration.create()
   conf.set("hbase.zookeeper.property.clientPort", zkClientPort)
   conf.set("hbase.zookeeper.quorum", zkQuorum)
   var table: Table = null
   var connection: Connection = null
   try {
     connection = ConnectionFactory.createConnection(conf)
     table = connection.getTable(TableName.valueOf("table1"))
     val iteratorArray = iterator.toArray
     val rowList = new util.ArrayList[Get]()
     for (row <- iteratorArray) {
       val get = new Get(row.getBytes)
       rowList.add(get)
     }
     // Obtain data in table1.
     val resultDataBuffer = table.get(rowList)
     // Set data in table1.
     val putList = new util.ArrayList[Put]()
     for (i <- 0 until iteratorArray.size) {
       val row = iteratorArray(i)
       val resultData = resultDataBuffer(i)
       if (!resultData.isEmpty) {
         // Obtain the old value based on the column family and column.
         val aCid = Bytes.toString(resultData.getValue(columnFamily.getBytes, "cid".getBytes))
         val put = new Put(Bytes.toBytes(row))
         // Calculate the result.
         val resultValue = row.toInt + aCid.toInt
         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()
       }
     }
   }
```



```
  }


  private def printUsage {
    System.out.println("Usage: {checkPointDir} {topic} {brokerList} {zkQuorum}")
    System.exit(1)
  }
}
```

14.3.9 Application for Connecting Spark Streaming to Kafka0-10

14.3.9.1 Scenario Description

Scenario Description

Assume that Kafka receives one word record every second in a service.

Based on some service requirements, a Spark application must be developed to implement the following functions:

Calculate the total number of records of each word in real time.

The following is an example of the **log1.txt** file.

```
LiuYang
YuanJing
GuoYijun
CaiXuyu
Liyuan
FangBo
LiuYang
YuanJing
GuoYijun
CaiXuyu
FangBo
```

Data Planning

Data of the Spark Streaming sample project is stored in Kafka. Send data to Kafka (A user with the Kafka permission is required).

1. Ensure that the clusters are installed, including HDFS, Yarn, Spark, and Kafka.
2. Create the **input_data1.txt** file on the local PC and copy the content of the **log1.txt** file to **input_data1.txt**.

   Create the **/home/data** directory on the client installation node. Upload the preceding file to the **/home/data** directory.
3. Modify **allow.everyone.if.no.acl.found** of Kafka Broker to **true**. (This parameter does not need to be set for the normal cluster.)
4. Create a topic.

   {zkQuorum} indicates ZooKeeper cluster information in the IP:port format.

   $KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper {zkQuorum}/kafka --replication-factor 1 --partitions 3 --topic {Topic}
5. Start the Producer of Kafka to send data to Kafka.

   java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/streamingClient010/*:$KAFKA_HOME/libs/*:{JAR_PATH}

com.huawei.bigdata.spark.examples.StreamingExampleProducer {BrokerList} {Topic}

NOTE

- **JAR_PATH** indicates the path of the JAR package. The value of **BrokerList** is in **brokerIp:9092** format.
- You need to change the value of **kerberos.domain.name** in the **SecurityKafkaWordCount** class to the value of **kerberos.domain.name** in the **$KAFKA_HOME/config/consumer.properties** file.
- If the user needs to connect to the security Kafka, add **KafkaClient** configuration information to the **jaas.conf** file in the **conf** directory on the Spark client. The following is an example:

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true
keyTab = "./user.keytab"
principal="leoB@HADOOP.COM"
useTicketCache=false
storeKey=true
debug=true;
};
```

  In Spark on Yarn mode, **jaas.conf** and **user.keytab** are distributed to the **container** directory of Spark on Yarn through Yarn. Therefore, the path of **keyTab** in **KafkaClient** must be the same as the path of **jaas.conf**, for example, **./user.keytab**. Change **principal** to the username created by yourself and domain name of the cluster.

Development Guidelines

1. Receive data from Kafka and generate the corresponding DStream.
2. Classify word records.
3. Calculate the result and print it.

14.3.9.2 Java Sample Code

Function Description

In Spark applications, use Streaming to call Kafka APIs to obtain word records. Classify word records to obtain the number of records of each word and write the result data to Kafka0-10.

Sample Code for Streaming to Read Kafka0-10

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.SecurityKafkaWordCount**. For details about the sample code, see **Obtaining the MRS Application Development Sample Project**.

NOTE

For a normal cluster, you need to comment out the 78th line (as shown in the following) in the **com.huawei.bigdata.spark.examples.SecurityKafkaWordCount** class:

kafkaParams.put("security.protocol", "SASL_PLAINTEXT");.

```
/**
 * One or more topic messages from Kafka
 * <checkPointDir> is the Spark Streaming checkpoint directory.
```



```
 * <brokers> is used for bootstrapping. The producer only uses it to obtain metadata.
 * <topics> is a list of one or more Kafka topics to be consumed.
 * <batchTime> is the duration (in seconds) of one Spark Streaming batch.
 */
public class SecurityKafkaWordCount
{
  public static void main(String[] args) throws Exception {
    JavaStreamingContext ssc = createContext(args);

    // Start the Streaming system.
    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 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));

    // Set the CheckPoint directory of Streaming.
    // This parameter is mandatory because a window concept exists.
 ssc.checkpoint(checkPointDir);

    // Obtain a list of topics 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.hadoop.com");

    LocationStrategy locationStrategy = LocationStrategies.PreferConsistent();
    ConsumerStrategy consumerStrategy = ConsumerStrategies.Subscribe(topicSet, kafkaParams);

    // Create a direct kafka stream using brokers and topics.
    // Receive data from Kafka and generate the corresponding DStream.
    JavaInputDStream<ConsumerRecord<String, String>> messages = KafkaUtils.createDirectStream(ssc,
locationStrategy, consumerStrategy);

    // Obtain the field attribute of 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();
      }
    });

    // Sum the total time for calculating the number of words.
    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 result.
  wordCounts.print();
  return ssc;
 }
}
```

Streaming Write To Kafka 0–10 Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.DstreamKafkaWriter**.

NOTE

- You are advised to use the new API createDirectStream to develop applications instead of the old API createStream. The old API can still be used, but the new API provides better performance and stability.
- The sample code exists only in **mrs-sample-project-1.6.0.zip**.

```
/**
 * Parameter description:
 * <checkPointDir> is the checkPoint directory.
 * <topics>: Topics subscribed in the Kafka. Multiple topics are separated by commas (,).
 * <brokers> is the Kafka address for obtaining metadata.
 */
public class JavaDstreamKafkaWriter {

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

  if (args.length != 4) {
    System.err.println("Usage: DstreamKafkaWriter <checkPointDir> <brokers> <topic>");
    System.exit(1);
  }

  String checkPointDir = args[0];
  String brokers = args[1];
  String topic = args[2];

  SparkConf sparkConf = new SparkConf().setAppName("KafkaWriter");

  // Enter the properties of Kafka.
  Map kafkaParams = new HashMap<String, Object>();
  kafkaParams.put("zookeeper.connect", brokers);
  kafkaParams.put("metadata.broker.list", brokers);
  kafkaParams.put("group.id", "dstreamKafkaWriterFt08");
  kafkaParams.put("auto.offset.reset", "smallest");

  // Create a Context of the Java Spark Streaming.
  JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.milliseconds(500));
```



```
    // Enter data to be written to Kafka.
    List<String> sentData = new ArrayList<String>();
    sentData.add("kafka_writer_test_msg_01");
    sentData.add("kafka_writer_test_msg_02");
    sentData.add("kafka_writer_test_msg_03");

    // Create a Java RDD queue.
    Queue<JavaRDD<String>> sent = new LinkedList();
    sent.add(ssc.sparkContext().parallelize(sentData));

    // Create a Java DStream for writing data.
    JavaDStream wStream = ssc.queueStream(sent);
    // Write data to Kafka.

JavaDStreamKafkaWriterFactory.fromJavaDStream(wStream).writeToKafka(JavaConverters.mapAsScalaMapC
onverter(kafkaParams),
        new Function<String, ProducerRecord<String, byte[]>>() {
          public ProducerRecord<String, byte[]> call(String s) {
            return new ProducerRecord(topic, s.getBytes());
          }
        });

    ssc.start();
    ssc.awaitTermination();
  }
}
```

14.3.9.3 Scala Sample Code

Function Description

In Spark applications, use Streaming to call Kafka APIs to obtain word records. Classify word records to obtain the number of records of each word and write the result data to Kafka0-10.

Sample Code for Streaming to Read Kafka0-10

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.SecurityKafkaWordCount**.

NOTE

For a normal cluster, you need to comment out the 60th line (as shown in the following) in the **com.huawei.bigdata.spark.examples.SecurityKafkaWordCount.scala** class:

"security.protocol" -> "SASL_PLAINTEXT",

```
/**
 * One or more topic messages from Kafka
 * <checkPointDir> is the Spark Streaming checkpoint directory.
 * <brokers> is used for bootstrapping. The producer only uses it to obtain metadata.
 * <topics> is a list of one or more Kafka topics to be consumed.
 * <batchTime> is the duration (in seconds) of one Spark Streaming batch.
 */
object SecurityKafkaWordCount {

  def main(args: Array[String]) {
    val ssc = createContext(args)

    // Start the Streaming system.
    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))

  // Set the CheckPoint directory of Streaming.
  // This parameter is mandatory because a window concept exists.
  ssc.checkpoint(checkPointDir)

  // Obtain a list of topics 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.hadoop.com"
  );

  val locationStrategy = LocationStrategies.PreferConsistent
  val consumerStrategy = ConsumerStrategies.Subscribe[String, String](topicSet, kafkaParams)

  // Create a direct kafka stream using brokers and topics.
  // Receive data from Kafka and generate the corresponding DStream.
  val stream = KafkaUtils.createDirectStream[String, String](ssc, locationStrategy, consumerStrategy)

  // Obtain the field attribute of each row.
  val tf = stream.transform ( rdd =>
    rdd.map(r => (r.value, 1L))
  )

  // Sum the total time for calculating the number of words.
  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))
}
```

Streaming Write To Kafka 0–10 Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.DstreamKafkaWriter**.

NOTE

- You are advised to use the new API createDirectStream to develop applications instead of the old API createStream. The old API can still be used, but the new API provides better performance and stability.
- The sample code exists only in **mrs-sample-project-1.6.0.zip**.

```
/**
 * Parameter description:
 * <checkPointDir> is the checkPoint directory.
 * <topics>: Topics subscribed in the Kafka. Multiple topics are separated by commas (,).
 * <brokers> is the Kafka address for obtaining metadata.
 */
object DstreamKafkaWriterTest1 {

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



```
  if (args.length != 4) {
    System.err.println("Usage: DstreamKafkaWriterTest <checkPointDir> <brokers> <topic>")
    System.exit(1)
  }

  val Array(checkPointDir, brokers, topic) = args
  val sparkConf = new SparkConf().setAppName("KafkaWriter")

  // Enter the properties of Kafka.
  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" -> "dstreamKafkaWriterFt",
    "auto.offset.reset" -> "latest"
  )

  // Creates the context of Streaming.
  val ssc = new StreamingContext(sparkConf, Milliseconds(500));
  val sentData = Seq("kafka_writer_test_msg_01", "kafka_writer_test_msg_02", "kafka_writer_test_msg_03")

  // Create an RDD queue.
  val sent = new mutable.Queue[RDD[String]]()
  sent.enqueue(ssc.sparkContext.makeRDD(sentData))

  // Create a DStream for writing data.
  val wStream = ssc.queueStream(sent)

  // Use the writetokafka API to write data to Kafka.
  wStream.writeToKafka(kafkaParams,
    (x: String) => new ProducerRecord[String, Array[Byte]](topic, x.getBytes))

  // Start the context of Streaming.
  ssc.start()
  ssc.awaitTermination()
 }
}
```

14.3.10 Structured Streaming Application

14.3.10.1 Scenario Description

Scenario Description

In Spark applications, use Structured Streaming to call Kafka APIs to obtain word records. Classify word records to obtain the number of records of each word.

Data Planning

Data of the Structured Streaming sample project is stored in Kafka. Send data to Kafka (A user with the Kafka permission is required).

1. Ensure that the clusters are installed, including HDFS, Yarn, Spark, and Kafka.
2. Modify **allow.everyone.if.no.acl.found** of Kafka Broker to **true**. (This parameter does not need to be set for the normal cluster.)
3. Create a topic.

   {zkQuorum} indicates ZooKeeper cluster information in the IP:port format.

   $KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper {zkQuorum}/kafka --replication-factor 1 --partitions 1 --topic {Topic}

4. Start the Producer of Kafka to send data to Kafka.

{ClassPath} indicates the path for storing the JAR file of the project. The path is specified by users. For details, see **Compiling and Running a Spark Application**.

java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/streamingClient010/*:{JAR_PATH} com.huawei.bigdata.spark.examples.KafkaWordCountProducer {BrokerList} {Topic} {messagesPerSec} {wordsPerMessage}

NOTE

- **JAR_PATH** indicates the path of the JAR package. The value of **BrokerList** is in **brokerIp:9092** format.
- If the user needs to connect to the security Kafka, add **KafkaClient** configuration information to the **jaas.conf** file in the **conf** directory on the Spark client. The following is an example:

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true
keyTab = "./user.keytab"
principal="leoB@HADOOP.COM"
useTicketCache=false
storeKey=true
debug=true;
};
```

  In Spark on Yarn mode, **jaas.conf** and **user.keytab** are distributed to the **container** directory of Spark on Yarn through Yarn. Therefore, the path of **keyTab** in **KafkaClient** must be the same as the path of **jaas.conf**, for example, **./user.keytab**. Change **principal** to the username created by yourself and domain name of the cluster.

Development Guidelines

1. Receive data from Kafka and generate the corresponding DataStreamReader.
2. Classify word records.
3. Calculate the result and print it.

14.3.10.2 Java Sample Code

Function Description

In Spark applications, use Structured Streaming to call Kafka APIs to obtain word records. Classify word records to obtain the number of records of each word.

Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.SecurityKafkaWordCount**.

NOTE

- For a normal cluster, comment out **.option("kafka.security.protocol", protocol)** in the 61st line of the **com.huawei.bigdata.spark.examples.SecurityKafkaWordCount** class in the sample code.
- When new data is available in Streaming DataFrame/Dataset, **outputMode** is used for configuring data written to the Streaming receiver. The default value is **append**.


```
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 the running word counts.
    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();
  }
}
```

14.3.10.3 Scala Sample Code

Function Description

In Spark applications, use Structured Streaming to call Kafka APIs to obtain word records. Classify word records to obtain the number of records of each word.

Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.bigdata.spark.examples.SecurityKafkaWordCount**.

NOTE

- For a normal cluster, comment out **.option("kafka.security.protocol", protocol)** in the 49th line of the **com.huawei.bigdata.spark.examples.SecurityKafkaWordCount.scala** in the sample code.
- When new data is available in Streaming DataFrame/Dataset, **outputMode** is used for configuring data written to the Streaming receiver. The default value is **append**.

```
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 the running word counts.
    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()
  }
}
```

14.4 Commissioning a Spark Application

14.4.1 Compiling and Running a Spark Application

Scenario

After application code development is complete, you can upload the JAR file to the Linux client to run applications. The procedures for running applications developed using Scala or Java are the same on the Spark client.

NOTE

- Spark applications can run only on Linux, but not on Windows.
- A Spark application developed using Python does not need to be packed into a JAR file. You only need to copy the sample project to the compiler.

Running the Spark Core Sample Application

Step 1 Run the **mvn package** command in the project directory to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **FemaleInfoCollection.jar**.

Step 2 Copy the generated JAR file (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. In the security cluster with Kerberos authentication enabled, copy the **user.keytab** and **krb5.conf** files obtained in **Preparing a Spark Application Development User** to the **conf** directory of the Spark client, for example, **/opt/client/Spark/spark/conf**. For a cluster with Kerberos authentication disabled, you do not need to copy the **user.keytab** and **krb5.conf** files.

Step 3 Run the Spark Core sample application (Scala and Java).

NOTICE

- 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.
- When running the program, you can select the following running mode as required:
  - ***--deploy-mode client***: The driver process runs on the client, and the running result is directly output after the program running.
  - ***--deploy-mode cluster***: The driver process runs in ApplicationMaster (AM) of Yarn. The running result and logs are displayed on the Yarn web UI.

Go to the Spark client directory and invoke the **bin/spark-submit** script to run code.

<inputPath> indicates the input path in HDFS.

bin/spark-submit --class com.huawei.bigdata.spark.examples.*FemaleInfoCollection* ***--master yarn --deploy-mode client*** */opt/female/FemaleInfoCollection.jar <inputPath>*

----End

Running the Spark SQL Sample Application

Step 1 Run the **mvn package** command in the project directory to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **FemaleInfoCollection.jar**.

Step 2 Copy the generated JAR file (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. In the security cluster with Kerberos authentication enabled, copy the **user.keytab** and

krb5.conf files obtained in **Preparing a Spark Application Development User** to the **conf** directory of the Spark client, for example, **/opt/client/Spark/spark/conf**. For a cluster with Kerberos authentication disabled, you do not need to copy the **user.keytab** and **krb5.conf** files.

Step 3 Run the Spark SQL sample application (Scala and Java).

NOTICE

- 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.
- When running the program, you can select the following running mode as required:
  - ***--deploy-mode client***: The driver process runs on the client, and the running result is directly output after the program running.
  - ***--deploy-mode cluster***: The driver process runs in ApplicationMaster (AM) of Yarn. The running result and logs are displayed on the Yarn web UI.

Go to the Spark client directory and invoke the **bin/spark-submit** script to run code.

<inputPath> indicates the input path in HDFS.

bin/spark-submit --class com.huawei.bigdata.spark.examples.*FemaleInfoCollection* ***--master yarn --deploy-mode client*** */opt/female/FemaleInfoCollection.jar <inputPath>*

----End

Running the Spark Streaming Sample Application

Step 1 Run the **mvn package** command in the project directory to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **FemaleInfoCollection.jar**.

Step 2 Copy the generated JAR file (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. In the security cluster with Kerberos authentication enabled, copy the **user.keytab** and **krb5.conf** files obtained in **Preparing a Spark Application Development User** to the **conf** directory of the Spark client, for example, **/opt/client/Spark/spark/conf**. For a cluster with Kerberos authentication disabled, you do not need to copy the **user.keytab** and **krb5.conf** files.

Step 3 Run the Spark Streaming sample application (Scala and Java).

NOTICE

- 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.
- When running the program, you can select the following running mode as required:
  - ***--deploy-mode client***: The driver process runs on the client, and the running result is directly output after the program running.
  - ***--deploy-mode cluster***: The driver process runs in ApplicationMaster (AM) of Yarn. The running result and logs are displayed on the Yarn web UI.

Go to the Spark client directory and invoke the **bin/spark-submit** script to run code.

NOTE

The path of the Spark Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is **$SPARK_HOME/jars**, and the path of the Spark Streaming Kafka dependency package is **$SPARK_HOME/jars/streamingClient**. Therefore, when you run an application, you need to add a configuration item to the **spark-submit** command to specify the path of the Spark Streaming Kafka dependency package, for example, *--jars $SPARK_HOME/jars/streamingClient/kafka-clients-*.jar,$SPARK_HOME/jars/streamingClient/kafka_*.jar,$SPARK_HOME/jars/streamingClient/spark-streaming-kafka-0-8_*.jar*.

- Spark Streaming Write To Print Sample Code

  bin/spark-submit --master yarn --deploy-mode client *--jars $SPARK_HOME/jars/streamingClient/kafka-clients-*.jar,$SPARK_HOME/jars/streamingClient/kafka_*.jar,$SPARK_HOME/jars/streamingClient/spark-streaming-kafka-*.jar* ***--class*** com.huawei.bigdata.spark.examples.*FemaleInfoCollectionPrint /opt/female/FemaleInfoCollectionPrint.jar <checkPointDir> <batchTime> <topics> <brokers>*

  NOTE

  - The JAR version name in **--jars** varies depending on the cluster.
  - The value of **brokers** is in **brokerIp:9092** format.
  - **<checkPointDir>** indicates the path for backing up the application result to the HDFS. **<batchTime>** indicates the interval for Streaming to process data in batches.

- Write To Kafka Sample Code for Spark Streaming

  bin/spark-submit --master yarn --deploy-mode client *--jars $SPARK_HOME/jars/streamingClient/kafka-clients-*.jar,$SPARK_HOME/jars/streamingClient/kafka_*.jar,$SPARK_HOME/jars/streamingClient/spark-streaming-kafka-*.jar* ***--***

  class *com.huawei.bigdata.spark.examples.DstreamKafkaWriter/opt/female/SparkStreamingExample-1.0.jar <groupId> <brokers> <topic>*

----End

Running the "Accessing Spark SQL Through JDBC" Sample Application

Step 1 Run the **mvn package** command in the project directory to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **FemaleInfoCollection.jar**.

Step 2 Copy the generated JAR file (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. In the security cluster with Kerberos authentication enabled, copy the **user.keytab** and **krb5.conf** files obtained in **Preparing a Spark Application Development User** to the **conf** directory of the Spark client, for example, **/opt/client/Spark/spark/conf**. For a cluster with Kerberos authentication disabled, you do not need to copy the **user.keytab** and **krb5.conf** files.

Step 3 Run the "Accessing Spark SQL Through JDBC" sample application (Scala and Java).

NOTICE

- 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.
- When running the program, you can select the following running mode as required:
  - ***--deploy-mode client***: The driver process runs on the client, and the running result is directly output after the program running.
  - ***--deploy-mode cluster***: The driver process runs in ApplicationMaster (AM) of Yarn. The running result and logs are displayed on the Yarn web UI.

Go to the Spark client directory and run the **java -cp** command to run the code.

```
java -cp ${SPARK_HOME}/jars/*:${SPARK_HOME}/conf:/opt/female/
SparkThriftServerJavaExample-*.jar
com.huawei.bigdata.spark.examples.ThriftServerQueriesTest ${SPARK_HOME}/
conf/hive-site.xml ${SPARK_HOME}/conf/spark-defaults.conf
```

NOTE

For a normal cluster, comment out the security configuration code. For details, see **Step 2** and **Step 2**.

In the preceding command line, you can minimize the corresponding running dependency packages based on different sample projects. For details about the dependency package of the sample project, see **Step 1**.

----End

Running the Spark on HBase Sample Application

Step 1 Run the **mvn package** command in the project directory to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **FemaleInfoCollection.jar**.

Step 2 Copy the generated JAR file (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. In the security cluster with Kerberos authentication enabled, copy the **user.keytab** and **krb5.conf** files obtained in **Preparing a Spark Application Development User** to

the **conf** directory of the Spark client, for example, **/opt/client/Spark/spark/conf**. For a cluster with Kerberos authentication disabled, you do not need to copy the **user.keytab** and **krb5.conf** files.

Step 3 Run the Spark on HBase sample application (Scala and Java).

NOTICE

- 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.
- When running the program, you can select the following running mode as required:
  - ***--deploy-mode client***: The driver process runs on the client, and the running result is directly output after the program running.
  - ***--deploy-mode cluster***: The driver process runs in ApplicationMaster (AM) of Yarn. The running result and logs are displayed on the Yarn web UI.

Go to the Spark client directory and invoke the **bin/spark-submit** script to run code. The application running sequence is as follows: TableCreation, TableInputData, and TableOutputData.

When running the TableInputData sample application, you need to specify **<inputPath>**, which indicates the input path in HDFS.

bin/spark-submit --class com.huawei.bigdata.spark.examples.*TableInputData* ***--master yarn --deploy-mode client*** */opt/female/TableInputData.jar <inputPath>*

NOTE

If Kerberos authentication is enabled when Spark tasks connect to HBase to read and write data, set **spark.yarn.security.credentials.hbase.enabled** in the client configuration file **spark-default.conf** to **true**. This configuration needs to be modified for all Spark tasks connecting to HBase to read and write data.

----End

Running the Spark HBase to HBase Sample Application

Step 1 Run the **mvn package** command in the project directory to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **FemaleInfoCollection.jar**.

Step 2 Copy the generated JAR file (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. In the security cluster with Kerberos authentication enabled, copy the **user.keytab** and **krb5.conf** files obtained in **Preparing a Spark Application Development User** to the **conf** directory of the Spark client, for example, **/opt/client/Spark/spark/conf**. For a cluster with Kerberos authentication disabled, you do not need to copy the **user.keytab** and **krb5.conf** files.

Step 3 Run the Spark HBase to HBase sample application (Scala and Java).

NOTICE

- 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.
- When running the program, you can select the following running mode as required:
  - ***--deploy-mode client***: The driver process runs on the client, and the running result is directly output after the program running.
  - ***--deploy-mode cluster***: The driver process runs in ApplicationMaster (AM) of Yarn. The running result and logs are displayed on the Yarn web UI.

Go to the Spark client directory and invoke the **bin/spark-submit** script to run code.

When running the sample application, you need to specify **<zkQuorum>**, which indicates the IP address of ZooKeeper.

bin/spark-submit --class com.huawei.bigdata.spark.examples.*SparkHbasetoHbase* ***--master yarn --deploy-mode client*** */opt/female/FemaleInfoCollection.jar* *<zkQuorum>*

----End

Running the Spark Hive to HBase Sample Application

Step 1 Run the **mvn package** command in the project directory to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **FemaleInfoCollection.jar**.

Step 2 Copy the generated JAR file (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. In the security cluster with Kerberos authentication enabled, copy the **user.keytab** and **krb5.conf** files obtained in **Preparing a Spark Application Development User** to the **conf** directory of the Spark client, for example, **/opt/client/Spark/spark/conf**. For a cluster with Kerberos authentication disabled, you do not need to copy the **user.keytab** and **krb5.conf** files.

Step 3 Run the Spark Hive to HBase sample application (Scala and Java).

NOTICE

- 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.
- When running the program, you can select the following running mode as required:
  - ***--deploy-mode client***: The driver process runs on the client, and the running result is directly output after the program running.
  - ***--deploy-mode cluster***: The driver process runs in ApplicationMaster (AM) of Yarn. The running result and logs are displayed on the Yarn web UI.

Go to the Spark client directory and invoke the **bin/spark-submit** script to run code.

When running the sample application, you need to specify **<zkQuorum>**, which indicates the IP address of ZooKeeper.

bin/spark-submit --class com.huawei.bigdata.spark.examples.*SparkHivetoHbase* ***--master yarn --deploy-mode client*** */opt/female/FemaleInfoCollection.jar* *<zkQuorum>*

----End

Running the Spark Streaming Kafka to HBase Sample Application

Step 1 Run the **mvn package** command in the project directory to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **FemaleInfoCollection.jar**.

Step 2 Copy the generated JAR file (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. In the security cluster with Kerberos authentication enabled, copy the **user.keytab** and **krb5.conf** files obtained in **Preparing a Spark Application Development User** to the **conf** directory of the Spark client, for example, **/opt/client/Spark/spark/conf**. For a cluster with Kerberos authentication disabled, you do not need to copy the **user.keytab** and **krb5.conf** files.

Step 3 Run the Spark Streaming Kafka to HBase sample application (Scala and Java).

NOTICE

- 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.
- When running the program, you can select the following running mode as required:
  - ***--deploy-mode client***: The driver process runs on the client, and the running result is directly output after the program running.
  - ***--deploy-mode cluster***: The driver process runs in ApplicationMaster (AM) of Yarn. The running result and logs are displayed on the Yarn web UI.

Go to the Spark client directory and invoke the **bin/spark-submit** script to run code.

When running the sample application, you need to specify **<checkPointDir><topic><brokerList>**. **<checkPointDir>** indicates an HDFS path for storing the application result backup. **<topic>** indicates a topic name read from Kafka. **<brokerList>** indicates the IP address of the Kafka server.

NOTE

The path of the Spark Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is **$SPARK_HOME/lib**, and the path of the Spark Streaming Kafka dependency package is **$SPARK_HOME/lib/streamingClient010**. Therefore, when you run an application, you need to add a configuration item to the **spark-submit** command to specify the path of the Spark Streaming Kafka dependency package, for example, *--jars $SPARK_HOME/jars/streamingClient010/kafka-clients-*.jar,$SPARK_HOME/jars/streamingClient010/kafka_*.jar,$SPARK_HOME/jars/streamingClient010/spark-streaming-kafka-*.jar*.

Spark Streaming To HBase Sample Code

bin/spark-submit --master yarn --deploy-mode client --jars *$SPARK_HOME/jars/streamingClient010/kafka-clients-*.jar,$SPARK_HOME/jars/streamingClient010/kafka_*.jar,$SPARK_HOME/jars/streamingClient010/spark-streaming-kafka-0*.jar* ***--class*** com.huawei.bigdata.spark.examples.*streaming.SparkOnStreamingToHbase /opt/female/FemaleInfoCollectionPrint.jar <checkPointDir> <topic> <brokerList>*

NOTE

- -- The JAR file name in **--jars** varies depending on the cluster.
- The format of **brokerlist** is **brokerIp:9092**.

----End

Running the "Connecting Spark Streaming to Kafka0-10" Sample Application

Step 1 Run the **mvn package** command in the project directory to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **FemaleInfoCollection.jar**.

Step 2 Copy the generated JAR file (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. In the security cluster with Kerberos authentication enabled, copy the **user.keytab** and **krb5.conf** files obtained in **Preparing a Spark Application Development User** to the **conf** directory of the Spark client, for example, **/opt/client/Spark/spark/conf**. For a cluster with Kerberos authentication disabled, you do not need to copy the **user.keytab** and **krb5.conf** files.

Step 3 Run the sample application (Scala and Java) for connecting Spark Streaming to Kafka0-10.

NOTICE

- 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.
- When running the program, you can select the following running mode as required:
  - ***--deploy-mode client***: The driver process runs on the client, and the running result is directly output after the program running.
  - ***--deploy-mode cluster***: The driver process runs in ApplicationMaster (AM) of Yarn. The running result and logs are displayed on the Yarn web UI.

Go to the Spark client directory and invoke the **bin/spark-submit** script to run code.

When running the sample application, you need to specify **<checkpointDir> <brokers> <topic> <batchTime>**. **<checkPointDir>** indicates an HDFS path for storing the application result backup. **<brokers>** indicates the Kafka address for obtaining metadata, and the value is in **brokerIp:21007** format in the security cluster mode and **brokerIp:9092** in normal cluster mode. **<topic>** indicates a topic name read from Kafka. **<batchTime>** indicates an interval for Streaming processing in batches.

"Spark Streaming Reads Kafka 0-10" Sample Code

- Run the following commands to submit a security cluster task:

  bin/spark-submit --master yarn --deploy-mode client --files ./conf/jaas.conf,./conf/user.keytab --driver-java-options "-Djava.security.auth.login.config=./jaas.conf" --conf "spark.executor.extraJavaOptions=-Djava.security.auth.login.config=./jaas.conf" --jars $SPARK_HOME/jars/streamingClient010/kafka-clients-*.jar,$SPARK_HOME/jars/streamingClient010/kafka_*.jar,$SPARK_HOME/jars/streamingClient010/spark-streaming-kafka-*.jar --class com.huawei.bigdata.spark.examples.SecurityKafkaWordCount /opt/SparkStreamingKafka010JavaExample-*.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.
  --driver-java-options "-Djava.security.auth.login.config=./jaas.conf" // Specify the path of jaas.conf file on the driver. In yarn-client mode, use --driver-java-options "-Djava.security.auth.login.config" to specify it. In yarn-cluster mode, use --conf "spark.yarn.cluster.driver.extraJavaOptions to specify it.
  --conf "spark.executor.extraJavaOptions=-Djava.security.auth.login.config=./jaas.conf"// Specify the path of the jaas.conf file on the executor.
  ```

- Command for submitting tasks in the normal cluster:

  bin/spark-submit --master yarn --deploy-mode client --jars $SPARK_HOME/jars/streamingClient010/kafka-clients-*.jar,$SPARK_HOME/jars/streamingClient010/kafka_*.jar,$SPARK_HOME/jars/streamingClient010/spark-streaming-kafka-*.jar --class com.huawei.bigdata.spark.examples.SecurityKafkaWordCount /opt/SparkStreamingKafka010JavaExample-*.jar <checkpointDir> <brokers> <topic> <batchTime>

  Spark Streaming Write To Kafka 0-10 code example (this example exists only in **mrs-sample-project-1.6.0.zip**):

  bin/spark-submit --master yarn --deploy-mode client --jars $SPARK_HOME/jars/streamingClient010/kafka-clients-*.jar,$SPARK_HOME/jars/streamingClient010/kafka_*.jar,$SPARK_HOME/jars/streamingClient010/spark-streaming-kafka-*.jar --class com.huawei.bigdata.spark.examples.JavaDstreamKafkaWriter /opt/JavaDstreamKafkaWriter.jar <checkPointDir> <brokers> <topics>

----End

Running the Spark Structured Streaming Sample Application

Step 1 Run the **mvn package** command in the project directory to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **FemaleInfoCollection.jar**.

Step 2 Copy the generated JAR file (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. In the security cluster with Kerberos authentication enabled, copy the **user.keytab** and **krb5.conf** files obtained in **Preparing a Spark Application Development User** to the **conf** directory of the Spark client, for example, **/opt/client/Spark/spark/conf**. For a cluster with Kerberos authentication disabled, you do not need to copy the **user.keytab** and **krb5.conf** files.

Step 3 Run the Spark Structured Streaming sample application (Scala and Java).

NOTICE

- 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.
- When running the program, you can select the following running mode as required:
  - ***--deploy-mode client***: The driver process runs on the client, and the running result is directly output after the program running.
  - ***--deploy-mode cluster***: The driver process runs in ApplicationMaster (AM) of Yarn. The running result and logs are displayed on the Yarn web UI.

Go to the **conf** directory of the Spark client and invoke the **spark-submit** script to run code.

When running the sample application, you need to specify **<brokers> <subscribe-type> <topic> <protocol> <service> <domain>**. **<brokers>** indicates the Kafka address for obtaining metadata. **<subscribe-type>** indicates a Kafka subscription type (which is generally **subscribe**, indicating the specified topic that is subscribed). **<topic>** indicates a topic name read from Kafka. **<protocol>** indicates a security access protocol. **<service>** indicates a Kerberos service name. **<domain>** indicates a Kerberos domain name.

NOTE

For a normal cluster, comment out some code for configuring the Kafka security protocol. For details, see the description in **Java Sample Code** and **Scala Sample Code**.

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 **$SPARK_HOME/jars**, and the path of the Spark Structured Streaming Kafka dependency package is **$SPARK_HOME/jars/streamingClient010**. Therefore, when you run an application, you need to add a configuration item to the **spark-submit** command to specify the path of the Spark Streaming Kafka dependency package, for example, **--jars $SPARK_HOME/jars/streamingClient010/kafka-clients-*.jar,$SPARK_HOME/jars/streamingClient010/kafka_*.jar,$SPARK_HOME/jars/streamingClient010/spark-sql-kafka-*.jar**.

Sample Code for Connecting Spark Structured Streaming to Kafka

- Run the following commands to submit tasks in the security cluster:

  cd /opt/client/Spark/spark/conf

  spark-submit --master yarn --deploy-mode client --files ./jaas.conf,./user.keytab --driver-java-options "-Djava.security.auth.login.config=./jaas.conf" --conf "spark.executor.extraJavaOptions=-Djava.security.auth.login.config=./jaas.conf" --jars $SPARK_HOME/jars/streamingClient010/kafka-clients-*.jar,$SPARK_HOME/jars/streamingClient010/kafka_*.jar,$SPARK_HOME/jars/streamingClient010/spark-sql-kafka-*.jar --class com.huawei.bigdata.spark.examples.SecurityKafkaWordCount /root/jars/SparkStructuredStreamingJavaExample-*.jar <brokers> <subscribe-type> <topic> <protocol> <service> <domain>

- Command for submitting tasks in the normal cluster:

  spark-submit --master yarn --deploy-mode client --jars $SPARK_HOME/jars/streamingClient010/kafka-clients-*.jar,$SPARK_HOME/jars/streamingClient010/kafka_*.jar,$SPARK_HOME/jars/streamingClient010/spark-sql-kafka-*.jar --class com.huawei.bigdata.spark.examples.SecurityKafkaWordCount /root/jars/SparkStructuredStreamingJavaExample-*.jar <brokers> <subscribe-type> <topic> <protocol> <service> <domain>

  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.
  --driver-java-options "-Djava.security.auth.login.config=<local Path>/jaas.conf" // Specify the path of **jaas.conf** file on the driver. In yarn-client mode, use **--driver-java-options "-Djava.security.auth.login.config"** to specify it. In **yarn-cluster** mode, use **--conf "spark.yarn.cluster.driver.extraJavaOptions** to specify it. If an error is reported, indicating that you have no permission to read and write the local directory, you need to specify **spark.sql.streaming.checkpointLocation** and you must have read and write permissions on the directory specified by this parameter.
  --conf "spark.executor.extraJavaOptions=-Djava.security.auth.login.config=./jaas.conf" // Specify the path of the **jaas.conf** file on the executor.
  -- The JAR file name in **--jars** varies depending on the cluster.
  The security cluster **<brokers>** is in **brokerIp:21007** format. For the **<protocol> <service> <domain>** format, see the **$KAFKA_HOME/config/consumer.properties** file.
  For a normal cluster, the value of **<brokers>** is in **brokerIp:9092** format. For details about **<domain>**, see the **$KAFKA_HOME/config/consumer.properties** file. The value of **<protocol>** is replaced with **null**, and the value of **<service>** is **kafka**.

----End

Submitting a Python Application

Step 1 Run the **mvn package** command in the project directory to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **FemaleInfoCollection.jar**.

Step 2 Copy the generated JAR file (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. In the security cluster with Kerberos authentication enabled, copy the **user.keytab** and **krb5.conf** files obtained in **Preparing a Spark Application Development User** to the **conf** directory of the Spark client, for example, **/opt/client/Spark/spark/conf**. For a cluster with Kerberos authentication disabled, you do not need to copy the **user.keytab** and **krb5.conf** files.

Step 3 Submit a Python application.

NOTICE

- 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.
- When running the program, you can select the following running mode as required:
  - ***--deploy-mode client***: The driver process runs on the client, and the running result is directly output after the program running.
  - ***--deploy-mode cluster***: The driver process runs in ApplicationMaster (AM) of Yarn. The running result and logs are displayed on the Yarn web UI.

Go to the Spark client directory and invoke the **bin/spark-submit** script to run code.

<inputPath> indicates the input path in HDFS.

NOTE

The sample code does not provide authentication information. Therefore, you need to configure **spark.yarn.keytab** and **spark.yarn.principal** to specify authentication information.

bin/spark-submit --master yarn --deploy-mode client --conf spark.yarn.keytab=/opt/FIclient/user.keytab --conf spark.yarn.principal=sparkuser */opt/female/SparkPythonExample/collectFemaleInfo.py <inputPath>*

----End

Submitting the SparkLauncher Application

Step 1 Run the **mvn package** command in the project directory to generate a JAR file and obtain it from the **target** directory in the project directory, for example, **FemaleInfoCollection.jar**.

Step 2 Copy the generated JAR file (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. In the security cluster with Kerberos authentication enabled, copy the **user.keytab** and **krb5.conf** files obtained in **Preparing a Spark Application Development User** to the **conf** directory of the Spark client, for example, **/opt/client/Spark/spark/conf**. For a cluster with Kerberos authentication disabled, you do not need to copy the **user.keytab** and **krb5.conf** files.

Step 3 Submit the SparkLauncher application.

NOTICE

- 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.
- When running the program, you can select the following running mode as required:
  - ***--deploy-mode client***: The driver process runs on the client, and the running result is directly output after the program running.
  - ***--deploy-mode cluster***: The driver process runs in ApplicationMaster (AM) of Yarn. The running result and logs are displayed on the Yarn web UI.

java -cp $SPARK_HOME/jars/*:{JAR_PATH} com.huawei.bigdata.spark.examples.SparkLauncherExample yarn-client {TARGET_JAR_PATH} { TARGET_JAR_MAIN_CLASS} {args}

NOTE

- **JAR_PATH** indicates the path of the SparkLauncher JAR package.
- **TARGET_JAR_PATH** indicates the path of the JAR package of the Spark application to be submitted.
- **args** is the parameter of the Spark application to be submitted.

----End

Related Information

The running dependency packages of the "Accessing Spark SQL Through JDBC" sample application (Scala and Java) are as follows:

- "Accessing Spark SQL Through JDBC" Sample Projects (Scala)
  - commons-collections-<version>.jar
  - commons-configuration-<version>.jar
  - commons-io-<version>.jar
  - commons-lang-<version>.jar
  - commons-logging-<version>.jar
  - guava-<version>.jar
  - hadoop-auth-<version>.jar
  - hadoop-common-<version>.jar
  - hadoop-mapreduce-client-core-<version>.jar
  - hive-exec-<version>.spark2.jar
  - hive-jdbc-<version>.spark2.jar
  - hive-metastore-<version>.spark2.jar
  - hive-service-<version>.spark2.jar
  - httpclient-<version>.jar
  - httpcore-<version>.jar
  - libthrift-<version>.jar

  - log4j-<version>.jar
  - slf4j-api-<version>.jar
  - zookeeper-<version>.jar
  - scala-library-<version>.jar
- "Accessing Spark SQL Through JDBC" Sample Projects (Java)
  - commons-collections-<version>.jar
  - commons-configuration-<version>.jar
  - commons-io-<version>.jar
  - commons-lang-<version>.jar
  - commons-logging-<version>.jar
  - guava-<version>.jar
  - hadoop-auth-<version>.jar
  - hadoop-common-<version>.jar
  - hadoop-mapreduce-client-core-<version>.jar
  - hive-exec-<version>.spark2.jar
  - hive-jdbc-<version>.spark2.jar
  - hive-metastore-<version>.spark2.jar
  - hive-service-<version>.spark2.jar
  - httpclient-<version>.jar
  - httpcore-<version>.jar
  - libthrift-<version>.jar
  - log4j-<version>.jar
  - slf4j-api-<version>.jar
  - zookeeper-<version>.jar

14.4.2 Viewing the Spark Application Commissioning Result

Scenario

After a Spark application is run, you can view the running result or view the application running status on the Spark web UI.

Procedure

- **Viewing the running result of the Spark application**

  The storage path and format of execution results are specified by the Spark application. You can obtain the results from the specified file.
- **Viewing the running status of the Spark application**

  Spark has two web UIs.
  - Spark UI: used to display the status of running applications.

    Spark UI consists of the following parts: Jobs, Stages, Storage, Environment, Executors, and SQL. The Streaming application has one more Streaming tab.

    Entry: Log in to MRS Manager by referring to **Logging in to MRS Manager**, choose **Services > Yarn**, and click **ResourceManager**

corresponding to **ResourceManager Web UI**. On the page that is displayed, find the corresponding Spark application. Click **ApplicationMaster** in the last column of the application information. The Spark UI page is displayed.

- History Server UI: used to display the running status of Spark applications that are complete or incomplete.

  History Server UI involves the application ID, application name, start time, end time, execution time, and owner information.

  Entry: Log in to MRS Manager by referring to **Logging in to MRS Manager**, choose **Services > Spark**, and click **JobHistory** corresponding to **Spark Web UI**.

- **Viewing Spark logs to learn application running status**

  View Spark logs to learn application running status, and adjust applications based on log information.

14.5 FAQs About Spark Application Development

14.5.1 Spark APIs

14.5.1.1 Spark Java APIs

To avoid API compatibility or reliability problems, you are advised to use open source APIs of the corresponding version.

Common Spark Core APIs

Spark mainly uses the following classes:

- JavaSparkContext: external API of Spark, which is used to provide the functions of Spark for Java applications that invoke this class, for example, connecting Spark clusters and creating RDDs, accumulations, and broadcasts. Its function is equivalent to a container.
- SparkConf: Spark application configuration class, which is used to configure the application name, execution model, and executor memory.
- JavaRDD: Class for defining JavaRDD in Java applications. The function is similar to the RDD class of Scala.
- JavaPairRDD: JavaRDD class in the key-value format. This class provides methods such as groupByKey and reduceByKey.
- Broadcast: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- StorageLevel: data storage levels, including memory (MEMORY_ONLY), disk (DISK_ONLY), and memory+disk (MEMORY_AND_DISK)

JavaRDD supports two types of operations: Transformation and Action. **Table 14-7** and **Table 14-8** describe their common methods.

Table 14-7 Transformation

Method	Description
<R> JavaRDD<R> map(Function<T,R> f)	Uses Function on each element of the RDD.
JavaRDD<T> filter(Function<T,Boolean> f)	Invokes Function on all elements of the RDD and returns the element that is **true**.
<U> JavaRDD<U> flatMap(FlatMapFunction<T,U> f)	Invokes Function on all elements of the RDD and then flattens the results.
JavaRDD<T> sample(boolean withReplacement, double fraction, long seed)	Sampling
JavaRDD<T> distinct(int numPartitions)	Deletes duplicate elements.
JavaPairRDD<K,Iterable<V>> groupByKey(int numPartitions)	Returns (K,Seq[V]) and combines the values of the same key to a set.
JavaPairRDD<K,V> reduceByKey(Function2<V,V,V> func, int numPartitions)	Invokes Function on the values of the same key.
JavaPairRDD<K,V> sortByKey(boolean ascending, int numPartitions)	Sorts data by key. If **ascending** is set to **true**, data is sorted by key in ascending order.
JavaPairRDD<K,scala.Tuple2<V,W>> join(JavaPairRDD<K,W> other)	Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. **numTasks** indicates the number of concurrent tasks.
JavaPairRDD<K,scala.Tuple2<Iterable<V>,Iterable<W>>> cogroup(JavaPairRDD<K,W> other, int numPartitions)	Returns the dataset of <K,scala.Tuple2<Iterable<V>,Iterable<W>>> when the (K,V) and (K,W) datasets exist. **numTasks** indicates the number of concurrent tasks.
JavaPairRDD<T,U> cartesian(JavaRDDLike<U,?> other)	Returns the Cartesian product of the RDD and other RDDs.

Table 14-8 Action

Method	Description
T reduce(Function2<T,T, T> f)	Invokes Function2 on elements of the RDD.
java.util.List<T> collect()	Returns an array that contains all elements of the RDD.
long count()	Returns the number of elements in the dataset.
T first()	Returns the first element in the dataset.
java.util.List<T> take(int num)	Returns the first N elements.
java.util.List<T> takeSample(boolean withReplacement, int num, long seed)	Samples the dataset randomly and returns a dataset of num elements. **withReplacement** indicates whether replacement is used.
void saveAsTextFile(String path, Class<? extends org.apache.hadoop.io.compress.CompressionCodec> codec)	Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file.
java.util.Map<K,Object> countByKey()	Counts the times that each key occurs.
void foreach(VoidFunction<T> f)	Runs func on each element of the dataset.
java.util.Map<T,Long> countByValue()	Counts the times that each element of the RDD occurs.

Common Spark Streaming APIs

Spark Streaming mainly uses the following classes:

- JavaStreamingContext: main entrance of Spark Streaming. It provides methods for creating the DStream. A batch interval needs 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: API of key-value DStream. It provides operations such as reduceByKey and join.
- JavaReceiverInputDStream<T>: Defines any input stream that receives data from the network.

The common methods of Spark Streaming are similar to those of Spark Core. The following table provides some methods of Spark Streaming.

Table 14-9 Spark Streaming methods

Method	Description
JavaReceiverInputDStream<java.lang.String> socketStream(java.lang.String hostname,int port)	Creates an input stream and uses a TCP socket to receive data from the corresponding hostname and port. The received bytes are parsed to the UTF8 format. The default storage level is Memory+Disk.
JavaDStream<java.lang.String> textFileStream(java.lang.String directory)	Creates an input stream to detect new files compatible with the Hadoop file system, and read them as text files. The directory of the input parameter is an HDFS directory.
void start()	Starts the Streaming calculation.
void awaitTermination()	Terminates the await of the process, which is similar to pressing Ctrl+C.
void stop()	Stops the Streaming calculation.
<T> JavaDStream<T> transform(java.util.List<JavaDStream<?>> dstreams,Function2<java.util.List<JavaRDD<?>>,Time,JavaRDD<T>> transformFunc)	Performs the Function operation on each RDD to obtain a new DStream. In this function, the sequence of the JavaRDDs must be the same as the corresponding DStreams in the list.
<T> JavaDStream<T> union(JavaDStream<T> first,java.util.List<JavaDStream<T>> rest)	Creates a unified DStream from multiple DStreams with the same type and sliding time.

Table 14-10 Streaming enhanced feature APIs

Method	Description
JAVADStreamKafkaWriter.writeToKafka()	Writes data from DStream into Kafka in batch.
JAVADStreamKafkaWriter.writeToKafkaBySingle()	Writes data from DStream into Kafka one by one.

Common Spark SQL APIs

Spark SQL mainly uses the following classes:

- SQLContext: main entrance of Spark SQL functions and DataFrame
- DataFrame: distributed dataset organized by naming columns
- DataFrameReader: API for loading DataFrame from external storage systems
- DataFrameStatFunctions: implements the statistics function of DataFrame.
- UserDefinedFunction: user-defined functions

The following table provides common Actions methods.

Table 14-11 Spark SQL methods

Method	Description
Row[] collect()	Returns an array containing all the columns of DataFrame.
long count()	Returns the number of rows in DataFrame.
DataFrame describe(java.lang.String... cols)	Calculates the statistics, including the count, average value, standard deviation, minimum value, and maximum value.
Row first()	Returns the first row.
Row[] head(int n)	Returns the first n rows.
void show()	Displays the first 20 rows in DataFrame using a table.
Row[] take(int n)	Returns the first n rows in DataFrame.

Table 14-12 Basic DataFrame functions

Method	Description
void explain(boolean extended)	Prints the logical plan and physical plan of the SQL statement.
void printSchema()	Prints schema information to the console.
registerTempTable	Registers DataFrame as a temporary table, whose period is bound to SQLContext.
DataFrame toDF(java.lang.String... colNames)	Returns a DataFrame whose columns are renamed.
DataFrame sort(java.lang.String sortCol,java.lang.String... sortCols)	Sorts by column in ascending or descending order.
GroupedData rollup(Column... cols)	Rolls back the specified column of the current DataFrame in multiple dimensions.

14.5.1.2 Spark Scala APIs

To avoid API compatibility or reliability problems, you are advised to use open source APIs of the corresponding version.

Common Spark Core APIs

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 creating RDDs.
- SparkConf: Spark application configuration class, which is used to configure the application name, execution model, and executor memory.
- RDD: defines the RDD class in the Spark application. The class provides the data collection operation methods, such as map and filter.
- PairRDDFunctions: provides computation operations for the RDD data of the key-value pair, such as groupByKey.
- Broadcast: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- StorageLevel: data storage levels, including memory (MEMORY_ONLY), disk (DISK_ONLY), and memory+disk (MEMORY_AND_DISK)

RDD supports two types of operations: Transformation and Action. **Table 14-13** and **Table 14-14** describe their common methods.

Table 14-13 Transformation

Method	Description
map[U](f: (T) => U): RDD[U]	Uses the **f** method to generate a new RDD for each element in the RDD that invokes map.
filter(f: (T) => Boolean): RDD[T]	Invokes the f method for all RDD elements to generate a satisfied data set that is returned in the form of RDD.
flatMap[U](f: (T) => TraversableOnce[U]) (implicit arg0: ClassTag[U]): RDD[U]	Invokes the f method for all RDD elements and then flattens the results to generate a new RDD.
sample(withReplacement: Boolean, fraction: Double, seed: Long = Utils.random.nextLong): RDD[T]	Samples and returns a subset of RDD.
union(other: RDD[T]): RDD[T]	Returns a new RDD that contains a set of elements of the source RDD and the specified RDD.
distinct([numPartitions: Int]): RDD[T]	Deletes duplicate elements to generate a new RDD.

Method	Description
groupByKey(): RDD[(K, Iterable[V])]	Returns (K,Iterable[V]) and combines the values of the same key to a set.
reduceByKey(func: (V, V) => V[, numPartitions: Int]): RDD[(K, V)]	Invokes func on the values of the same key.
sortByKey(ascending: Boolean = true, numPartitions: Int = self.partitions.length): RDD[(K, V)]	Sorts by key in ascending or descending order. Ascending is of the boolean type.
join[W](other: RDD[(K, W)][, numPartitions: Int]): RDD[(K, (V, W))]	Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. **numPartitions** indicates the number of concurrent tasks.
cogroup[W](other: RDD[(K, W)], numPartitions: Int): RDD[(K, (Iterable[V], Iterable[W]))]	Returns the dataset of (K, (Iterable[V], Iterable[W])) when the (K,V) and (K,W) datasets of two key-value pairs exist. **numPartitions** indicates the number of concurrent tasks.
cartesian[U](other: RDD[U])(implicit arg0: ClassTag[U]): RDD[(T, U)]	Returns the Cartesian product of the RDD and other RDDs.

Table 14-14 Action

Method	Description
reduce(f: (T, T) => T):	Invokes f on elements of the RDD.
collect(): Array[T]	Returns an array that contains all elements of the RDD.
count(): Long	Returns the number of elements in the dataset.
first(): T	Returns the first element in the dataset.
take(num: Int): Array[T]	Returns the first N elements.
takeSample(withReplacement: Boolean, num: Int, seed: Long = Utils.random.nextLong): Array[T]	Samples the dataset randomly and returns a dataset of num elements. **withReplacement** indicates whether replacement is used.

Method	Description
saveAsTextFile(path: String): Unit	Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file.
saveAsSequenceFile(path: String, codec: Option[Class[_ <: CompressionCodec]] = None): Unit	This can be used only on the key-value pair, and then generates SequenceFile and writes is to the local or Hadoop file system.
countByKey(): Map[K, Long]	Counts the times that each key occurs.
foreach(func: (T) => Unit): Unit	Runs func on each element of the dataset.
countByValue() (implicit ord: Ordering[T] = null): Map[T, Long]	Counts the times that each element of the RDD occurs.

Common Spark Streaming APIs

Spark Streaming mainly uses the following classes:

- StreamingContext: main entrance of Spark Streaming. It provides methods for creating the DStream. A batch interval needs 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 the key-value pair. Common operations include groupByKey and reduceByKey.

  The Java APIs of the Spark Streaming are JavaSteamingContext, JavaDStream, and JavaPairDStream.

The common methods of Spark Streaming are similar to those of Spark Core. The following table provides some methods of Spark Streaming.

Table 14-15 Spark Streaming methods

Method	Description
socketTextStream(hostname: String, port: Int, storageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK_SER_2): ReceiverInputDStream[String]	Creates an input stream using the TCP protocol (source host:port).

Method	Description
start():Unit	Starts the Streaming calculation.
awaitTermination(timeout: long):Unit	Terminates the await of the process, which is similar to pressing Ctrl+C.
stop(stopSparkContext: Boolean, stopGracefully: Boolean): Unit	Stops the Streaming calculation.
transform[T] (dstreams: Seq[DStream[_]], transformFunc: (Seq[RDD[_]], Time) ? RDD[T])(implicit arg0: ClassTag[T]): DStream[T]	Performs the function operation on each RDD to obtain a new DStream.
UpdateStateByKey(func)	Updates the status of DStream. To use this method, you need to define the state and state update functions.
window(windowLength, slideInterval)	Generates a new DStream by batch calculating according to the window of the source DStream.
countByWindow(windowLength, slideInterval)	Returns the number of sliding window elements in the stream.
reduceByWindow(func, windowLength, slideInterval)	When the key-value pair of DStream is invoked, a new key-value pair of DStream is returned. The value of each key is obtained by aggregating the reduce function in batches in the sliding window.
join(otherStream, [numTasks])	Performs a join operation between different Spark Streamings.
DStreamKafkaWriter.writeToKafka()	Writes data from DStream into Kafka in batch.
DStreamKafkaWriter.writeToKafkaBySingle()	Writes data from DStream into Kafka one by one.

Table 14-16 Streaming enhanced feature APIs

Method	Description
DStreamKafkaWriter.writeToKafka()	Writes data from DStream into Kafka in batch.

Method	Description
DStreamKafkaWriter.writeToKafkaBySingle()	Writes data from DStream into Kafka one by one.

Common Spark SQL APIs

Spark SQL mainly uses the following classes:

- SQLContext: main entrance of Spark SQL functions and DataFrame
- DataFrame: distributed dataset organized by naming columns
- HiveContext: main entrance for obtaining data stored in Hive.

Table 14-17 Common Actions methods

Method	Description
collect(): Array[Row]	Returns an array containing all the columns of DataFrame.
count(): Long	Returns the number of rows in DataFrame.
describe(cols: String*): DataFrame	Calculates the statistics, including the count, average value, standard deviation, minimum value, and maximum value.
first(): Row	Returns the first row.
Head(n:Int): Row	Returns the first n rows.
show(numRows: Int, truncate: Boolean): Unit	Displays DataFrame in a table.
take(n:Int): Array[Row]	Returns the first n rows in DataFrame.

Table 14-18 Basic DataFrame functions

Method	Description
explain(): Unit	Prints the logical plan and physical plan of the SQL statement.
printSchema(): Unit	Prints schema information to the console.
registerTempTable(tableName: String): Unit	Registers DataFrame as a temporary table, whose period is bound to SQLContext.
toDF(colNames: String*): DataFrame	Returns a DataFrame whose columns are renamed.

14.5.1.3 Spark Python APIs

To avoid API compatibility or reliability problems, you are advised to use open source APIs of the corresponding version.

Common Spark Core APIs

Spark mainly uses the following classes:

- pyspark.SparkContext: external API of Spark. It provides the functions of Spark for Python applications that invoke this class, for example, connecting Spark clusters, creating RDDs, and broadcasting variables.
- pyspark.SparkConf: Spark application configuration class. It is used to set an application name, execution mode, and executor memory.
- pyspark.RDD: defines the RDD class in the Spark application. The class provides the data collection operation methods, such as map and filter.
- pyspark.Broadcast: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- pyspark.StorageLevel: data storage levels, including memory (MEMORY_ONLY), disk (DISK_ONLY), and memory+disk (MEMORY_AND_DISK)
- pyspark.sql.SQLContext: main entrance of the SparkSQL functions. It can be used to create DataFrame, register DataFrame as a table, and execute SQL on a table.
- pyspark.sql.DataFrame: distributed dataset. DataFrame is equivalent to a relationship table in SparkSQL and can be created using the method in SQLContext.
- pyspark.sql.DataFrameNaFunctions: function in DataFrame for processing data loss
- pyspark.sql.DataFrameStatFunctions: function in DataFrame for statistics. It calculates the variance between columns and sample covariance.

RDD supports two types of operations: Transformation and Action. **Table 14-19** and **Table 14-20** describe their common methods.

Table 14-19 Transformation

Method	Description
map(f, preservesPartition-ing=False)	Uses the Func method to generate a new RDD for each element in the RDD that invokes map.
filter(f)	Invokes the Func method for all RDD elements to generate a satisfied data set that is returned in the form of RDD.

Method	Description
flatMap(f, preservesPartition-ing=False)	Invokes the Func method for all RDD elements and then flattens the results to generate a new RDD.
sample(withReplacement, fraction, seed=None)	Samples and returns a subset of RDD.
union(rdds)	Returns a new RDD that contains a set of elements of the source RDD and the specified RDD.
distinct([numPartitions: Int]): RDD[T]	Deletes duplicate elements to generate a new RDD.
groupByKey(): RDD[(K, Iterable[V])]	Returns (K,Iterable[V]) and combines the values of the same key to a set.
reduceByKey(func, numPartitions=None)	Invokes Func on the values of the same key.
sortByKey(ascending=True, numPartitions=None, keyfunc=function <lambda>)	Sorts by key in ascending or descending order. Ascending is of the boolean type.
join(other, numPartitions)	Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. **numPartitions** indicates the number of concurrent tasks.
cogroup(other, numPartitions)	Returns the dataset of (K, (Iterable[V], Iterable[W])) when the (K,V) and (K,W) datasets of two key-value pairs exist. **numPartitions** indicates the number of concurrent tasks.
cartesian(other)	Returns the Cartesian product of the RDD and other RDDs.

Table 14-20 Action

Method	Description
reduce(f)	Invokes Func on elements of the RDD.
collect()	Returns an array that contains all elements of the RDD.
count()	Returns the number of elements in the dataset.
first()	Returns the first element in the dataset.
take(num)	Returns the first num elements.

Method	Description
takeSample(withReplacement, num, seed)	Samples the dataset randomly and returns a dataset of num elements. **withReplacement** indicates whether replacement is used.
saveAsTextFile(path, compressionCodecClass)	Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file.
saveAsSequenceFile(path, compressionCodecClass=None)	This can be used only on the key-value pair, and then generates SequenceFile and writes is to the local or Hadoop file system.
countByKey()	Counts the times that each key occurs.
foreach(func)	Runs the function on each element of the dataset.
countByValue()	Counts the times that each value of the RDD occurs.

Common Spark Streaming APIs

Spark Streaming mainly uses the following classes:

- pyspark.streaming.StreamingContext: main entrance of Spark Streaming. It provides methods for creating the DStream. A batch interval needs to be set in the input parameter.
- pyspark.streaming.DStream: a type of data which indicates the RDDs continuous sequence. It indicates the continuous data flow.
- dstream.PariDStreamFunctions: DStream of the key-value pair. Common operations include groupByKey and reduceByKey.

  The Java APIs of the Spark Streaming are JavaSteamingContext, JavaDStream, and JavaPairDStream.

The common methods of Spark Streaming are similar to those of Spark Core. The following table provides some methods of Spark Streaming.

Table 14-21 Common Spark Streaming APIs

Method	Description
socketTextStream(hostname, port, storageLevel)	Creates an input stream from the TCP source host:port.
start()	Starts the Streaming calculation.
awaitTermination(timeout)	Terminates the await of the process, which is similar to pressing Ctrl+C.

Method	Description
stop(stopSparkContext, stopGraceFully)	Stops the Streaming calculation. **stopSparkContext** is used to determine whether SparkContext needs to be terminated. **StopGracefully** is used to determine whether to wait for all the received data to be processed.
UpdateStateByKey(func)	Updates the status of DStream. To use this method, you need to define the state and state update functions.
window(windowLength, slideInterval)	Generates a new DStream by batch calculating according to the window of the source DStream.
countByWindow(windowLength, slideInterval)	Returns the number of sliding window elements in the stream.
reduceByWindow(func, windowLength, slideInterval)	When the key-value pair of DStream is invoked, a new key-value pair of DStream is returned. The value of each key is obtained by aggregating the reduce function in batches in the sliding window.
join(other,numPartitions)	Performs a join operation between different Spark Streamings.

Common Spark SQL APIs

Spark SQL mainly uses the following classes in Python:

- pyspark.sql.SQLContext: main entrance of Spark SQL functions and DataFrame
- pyspark.sql.DataFrame: distributed dataset organized by naming columns
- pyspark.sql.HiveContext: main entrance for obtaining data stored in Hive.
- pyspark.sql.DataFrameStatFunctions: some statistics functions
- pyspark.sql.functions: functions embedded in DataFrame
- pyspark.sql.Window: window function provided by SQL

Table 14-22 Common Actions of Spark SQL

Method	Description
collect()	Returns an array containing all the columns of DataFrame.
count()	Returns the number of rows in DataFrame.
describe()	Calculates the statistics, including the count, average value, standard deviation, minimum value, and maximum value.

Method	Description
first()	Returns the first row.
head(n)	Returns the first n rows.
show()	Displays DataFrame in a table.
take(num)	Returns the first num rows in DataFrame.

Table 14-23 Basic DataFrame functions

Method	Description
explain()	Prints the logical plan and physical plan of the SQL statement.
printSchema()	Prints schema information to the console.
registerTempTable(name)	Registers DataFrame as the **name** temporary table, whose period is bound to SQLContext.
toDF()	Returns a DataFrame whose columns are renamed.

14.5.1.4 Spark REST APIs

Function Description

Spark REST APIs display some metrics of the web UI in JSON format, provide users with a simpler method to create new display and monitoring tools, and enable users to query information about running apps and the completed apps. The open source Spark REST APIs allow users to query information about Jobs, Stages, Storage, Environment, and Executors. The REST APIs for querying the information about SQL, JDBC/ODBC Server, and Streaming are added to MRS. For details about open source RESTful APIs, visit **https://spark.apache.org/docs/2.2.2/monitoring.html#rest-api**.

Preparing an Operating Environment

Install a client in a directory on a node, for example, **/opt/client.**

1. You have installed Spark on the server and confirmed that Spark is running properly.
2. You have installed JDK 1.7 or 1.8 on the client operating environment.
3. Obtain the **MRS_Spark_Client.tar** installation package, and run the following commands to decompress the package:

   tar -xvf MRS_Spark_Client.tar

   tar -xvf MRS_Spark_ClientConfig.tar

> NOTE
>
> You are advised to install a client of the same version as the cluster on the server to avoid version incompatibility.

4. Go to the **MRS_Spark_ClientConfig** decompressed folder and run the following command to install the client:

   sh install.sh /opt/client

   In the preceding command, **/opt/client** is an example user-defined path.

5. Go to **/opt/client** (the client installation directory) and run the following command to initialize environment variables:

   source bigdata_env

REST APIs

You can run the following command to skip the REST API filter to obtain application information:

- Obtaining information about all applications in JobHistory
  - Command:

    ```
    curl https://192.168.227.16:18080/api/v1/applications?mode=monitoring --insecure
    ```

    192.168.227.16 is the service IP address of the JobHistory node, and **18080** is the port number of the JobHistory node.
  - Command output:

    ```
    [ {
    "id" : "application_1478570725074_0042",
    "name" : "Spark-JDBCServer",
    "attempts" : [ {
    "startTime" : "2016-11-09T16:57:15.237CST",
    "endTime" : "2016-11-09T17:01:22.573CST",
    "lastUpdated" : "2016-11-09T17:01:22.614CST",
    "duration" : 247336,
    "sparkUser" : "spark",
    "completed" : true
    } ]
    }, {
    "id" : "application_1478570725074_0047-part1",
    "name" : "SparkSQL::192.168.169.84",
    "attempts" : [ {
    "startTime" : "2016-11-10T11:57:36.626CST",
    "endTime" : "1969-12-31T07:59:59.999CST",
    "lastUpdated" : "2016-11-10T11:57:48.613CST",
    "duration" : 0,
    "sparkUser" : "admin",
    "completed" : false
    } ]
    }]
    ```
  - Result analysis:

    By running this command, you can query all Spark applications (including running applications and completed applications) in the current cluster. Table 14-24 provides information about the applications.

    Table 14-24 Basic application information

Parameter	Description
id	Application ID

Parameter	Description
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 an application in JobHistory
  - Command:
    ```
    curl https://192.168.227.16:18080/api/v1/applications/application_1478570725074_0042?mode=monitoring --insecure
    ```
    192.168.227.16 is the service IP address of the JobHistory node, and **18080** is the port number of the JobHistory node. **application_1478570725074_0042** is the application ID.
  - Command output:
    ```
    {
    "id" : "application_1478570725074_0042",
    "name" : "Spark-JDBCServer",
    "attempts" : [ {
    "startTime" : "2016-11-09T16:57:15.237CST",
    "endTime" : "2016-11-09T17:01:22.573CST",
    "lastUpdated" : "2016-11-09T17:01:22.614CST",
    "duration" : 247336,
    "sparkUser" : "spark",
    "completed" : true
    } ]
    }
    ```
  - Result analysis:

    By running this command, you can query information about a Spark application. **Table 14-24** provides information about the application.
- Obtaining information about the executor of a running application
  - Command for an alive executor:
    ```
    curl https://192.168.169.84:26001/proxy/application_1478570725074_0046/api/v1/applications/application_1478570725074_0046/executors?mode=monitoring --insecure
    ```
  - Commands for all alive and dead executors:
    ```
    curl https://192.168.169.84:26001/proxy/application_1478570725074_0046/api/v1/applications/application_1478570725074_0046/allexecutors?mode=monitoring --insecure
    ```
    192.168.195.232 is the service IP address of the active ResourceManager node, **26001** is the port number of ResourceManager, and **application_1478570725074_0046** is 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,
"activeTasks" : 0,
"failedTasks" : 0,
"completedTasks" : 0,
"totalTasks" : 0,
"totalDuration" : 0,
"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"
}
} ]
```


- Result analysis:

  By running this command, you can query information about all executors (including drivers of the current application). **Table 14-25** provides basic information about each executor.

  Table 14-25 Basic executor information

Parameter	Description
id	Executor ID
hostPort	IP address:port of the node of the executor
executorLogs	Executor log path

Enhanced REST APIs

- SQL commands: Obtain all SQL statements and SQL statements that have the longest execution time.
  - SparkUI command:

    ```
    curl https://192.168.195.232:26001/proxy/application_1476947670799_0053/api/v1/applications/
    Spark-JDBCServerapplication_1476947670799_0053/SQL?mode=monitoring --insecure
    ```

    192.168.195.232 is the service IP address of the active ResourceManager node, **26001** is the port number of ResourceManager. **application_1476947670799_0053** is the application ID in Yarn, and **Spark-JDBCServer** is the Spark application name.
  - JobHistory command:

    ```
    curl https://192.168.227.16:22500/api/v1/applications/application_1478570725074_0004-part1/
    SQL?mode=monitoring --insecure
    ```


192.168.227.16 is the service IP address of the JobHistory node, and **22500** is the port number of the JobHistory node. **application_1478570725074_0004-part1** is the application ID.

- Command output:

  The query results of the SparkUI and JobHistory commands are 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" : [ ]
}]
}
```

- Result analysis:

  After running the commands, you can query information about all SQL statements of the current application (that is, the **sqls** part in the result) and information about the SQL statement with the longest execution time (that is, the **longestDurationOfCompletedSQL** part in the result). The following table describes information about each SQL statement.

Table 14-26 Basic SQL statement information

Parameter	Description
id	ID of an SQL statement
status	Execution status of an SQL statement. The options are RUNNING, COMPLETED, and FAILED.
runningJobs	List of jobs that are being executed among the jobs generated by the SQL statement
successedJobs	List of jobs that are successfully executed among the jobs generated by the SQL statement
failedJobs	List of jobs that fail to be executed among the jobs generated by the SQL statement

- JDBC/ODBC Server commands: Obtain the number of connections, the number of running SQL statements, as well as information about all sessions and all SQL statements.
  - Command:

    ```
    curl https://192.168.195.232:26001/proxy/application_1476947670799_0053/api/v1/applications/application_1476947670799_0053/sqlserver?mode=monitoring --insecure
    ```

    192.168.195.232 is the service IP address of the active ResourceManager node, **26001** is the port number of ResourceManager, and **application_1476947670799_0053** is 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"
    } ]
    }
    ```
  - Result analysis:

    After running this command, you can query the number of sessions, the number of running SQL statements, as well as information about all sessions and SQL statements of the current JDBC/ODBC application. **Table 14-27** provides information about each session. **Table 14-28** provides information about each SQL statement.

    Table 14-27 Basic session information

Parameter	Description
user	User connected to the session
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	Session connection duration
totalExecute	Number of SQL statements executed on the session

Table 14-28 Basic SQL information

Parameter	Description
user	User who executes the SQL statement
jobId	List of job IDs contained in the SQL statement
groupId	ID of the group where the SQL statement resides
startTime	SQL start time
finishTime	SQL end time
duration	SQL execution duration
statement	SQL statement
detail	Logical plan and physical plan

- Streaming commands: Obtain the average input frequency, average scheduling delay, average execution duration, and average total delay.
  - Command:
    ```
    curl https://192.168.195.232:26001/proxy/application_1477722033672_0008/api/v1/applications/NetworkWordCountapplication_1477722033672_0008/streaming?mode=monitoring --insecure
    ```
    192.168.195.232 is the service IP address of the active ResourceManager node, **26001** is the port number of ResourceManager. **application_1477722033672_0008** is the application ID in Yarn, and **NetworkWordCount** is the Spark application name.
  - Command output:
    ```
    {
    "avgInputRate" : "0.00 events/sec",
    "avgSchedulingDelay" : "1 ms",
    "avgProcessingTime" : "72 ms",
    "avgTotalDelay" : "73 ms"
    }
    ```
  - Result analysis:

    After running this command, you can query the average input frequency, average scheduling delay, average execution duration, and average total delay of the current Streaming application.

14.5.1.5 Spark ThriftServer APIs

Introduction

ThriftServer is another HiveServer2 implementation in Hive. It uses the Spark structured query language (SQL) to process the SQL statements, providing higher performance than Hive.

ThriftServer is a JDBC API. Users can use JDBC to connect to Thrift Server to access SparkSQL data. When ThriftServer is started, a SparkSQL application is started, and the clients connected through the JDBC share the resources of the SparkSQL application, that is, different users can share data. When ThriftServer is started, a monitor is enabled to wait for the connections and queries submitted by the JDBC client. Therefore, when configuring ThriftServer, you must configure at least the host name and port number of ThriftServer. If you want to use Hive data, you also need to provide the URIs of Hive Metastore.

ThriftServer starts a JDBC service on port 10000 of the installation node by default. Users can connect to ThriftServer using Beeline or running the JDBC client code to run SQL statements.

For more information about ThriftServer, visit the Spark official website at **http://spark.apache.org/docs/1.5.1/sql-programming-guide.html#distributed-sql-engine**.

Beeline

For details about the Beeline connection modes provided by the open source community, visit **https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**.

JDBC Client Code

You can run the JDBC client code to connect to the ThriftServer to access SparkSQL data.

Enhanced Features

Compared with the open source community, MRS provides two enhanced features: ThriftServer HA solution and timeout interval setting for ThriftServer connections.

- In the ThriftServer HA solution, when the active ThriftServer node is faulty, the standby node can automatically switch to the active one to provide services for the cluster. The operations of using Beeline or JDBC client code for connection are the same.

  The difference between the character strings used for connecting to ThriftServer in HA or non-HA modes is that you need to replace **ip:port** with **ha-cluster** for connecting to ThriftServer in HA mode. **Table 14-29** provides other parameters that will be used.

Table 14-29 List of client parameters

Parameter	Description	Default Value
spark.thriftserver.ha.enabled	Indicates whether to enable the HA mode. The value **true** indicates that the HA mode is enabled. If the HA mode is enabled, change **host:port** to **ha-cluster** in the connection string. Otherwise, the system automatically exits the HA mode.	false
spark.thriftserver.zookeeper.dir	ThriftServer path for storing metadata on the ZooKeeper. The parameter value must be the same as those on the server. The subdirectory named **active_thriftserver** in this directory is used for storing the IP address and port number of the Hive ThriftServer.	/thriftserver
spark.deploy.zookeeper.url	ZooKeeper URL. The parameter value must be the same as those on the server.	-
spark.thriftserver.retry.times	Maximum number of attempts to connect to the server. If the value is set to a negative number or zero, the client does not attempt to connect to the server again.	5
spark.thriftserver.retry.wait.time	Interval for attempting to reconnect to the server. The unit is second.	10

The parameters in **Table 14-29** must be configured in the **hive-site.xml** file in **classpath** on the client. For example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration>
   <property>
      <name>spark.thriftserver.ha.enabled</name>
      <value>true</value>
   </property>
</configuration>
```

The **spark.deploy.zookeeper.url** parameter can also be replaced with **zk.quorum** in the connection string. For example:

```
!connect jdbc:hive2://ha-cluster/default;zk.quorum=spark25:2181,spark
26:2181,spark27:2181
```

- Set a timeout interval for a connection between the client and the ThriftServer.
  - Beeline

    In the case of network congestion, this feature prevents the beeline from being suspended due to the infinite waiting for the response of the server. The configuration method is as follows:

    When the beeline is started, append **--socketTimeOut=n**, where **n** indicates the timeout interval (unit: second) for waiting for the service response. The default value is **0**, indicating that the beeline never times

out. You are advised to set this parameter to the maximum value allowed based on the service scenario.

- JDBC client code

  In the case of network congestion, this feature prevents the client from being suspended due to the infinite waiting for the response of the server. The configuration method is as follows:

  Before using the **DriverManager.getConnection** method to obtain the JDBC connection, add the **DriverManager.setLoginTimeout(n)** method to configure a timeout interval. **n** indicates the timeout interval for waiting for the return result from the server. The unit is second, the type is **Int**, and the default value is **0** (indicating never timing out). You are advised to set this parameter to the maximum value allowed based on the service scenario.

14.5.1.6 Common Spark Commands

For details about how to use Spark commands, visit **http://spark.apache.org/docs/latest/quick-start.html**.

Common Commands

Methods of running shell commands:

Step 1 Go to the directory where the Spark client is installed.

Step 2 Run the following command to initialize environment variables:

source /opt/client/bigdata_env

Step 3 If the Kerberos authentication is enabled for the current cluster, run the following command to authenticate the user. If the Kerberos authentication is disabled for the current cluster, skip this step. The current user is the development user added in **Preparing a Spark Application Development User**.

kinit *MRS cluster user*

Example:

- If the development user is a machine-machine user, run **kinit -kt user.keytab sparkuser**.
- If the development user is a human-machine user, run **kinit sparkuser**.

Step 4 Run the Spark shell command.

----End

The common Spark commands are as follows:

- ***spark-shell***

  It provides a simple commissioning tool that supports Scala language.

  Run the ***spark-shell*** command on the shell console to enter the Scala interactive interface. Obtain data from HDFS, perform calculation by using the RDD, and output and print the result.

  Example: A line of code can be used to collect statistics on the frequency of all words in a file.

scala> sc.textFile("hdfs://hacluster/tmp/wordcount_data.txt").flatMap(line=> line.split(" ")).map(w => (w,1)).reduceByKey(_+_).collect()

- ***spark-submit***

  This command is used to submit Spark applications to an MRS cluster for running and return the running result. The **class**, **master**, JAR file, and input parameters must be specified.

  Example: Run GroupByTest in the JAR file. There are four input parameters and the cluster running mode is the yarn-client mode.

  spark-submit --class org.apache.spark.examples.GroupByTest --master yarn --deploy-mode client ${SPARK_HOME}/examples/jars/spark-examples_2.11-2.3.2-mrs-2.0.jar 6 3000 3000 3

- ***spark-sql***

  Start a Spark application and execute Spark SQL. You can specify **local(--master local)** or cluster mode **(--master yarn)**.

  Example of starting an instance:

  spark-sql --master yarn

  Example of SQL:

  – ***SELECT key FROM src GROUP BY key;***

  – ***EXPLAIN EXTENDED SHOW TABLES;***

- ***spark-beeline***

  Call Spark JDBCServer to execute Spark SQL to efficiently compute and analyze massive amounts of data. JDBCServer contains a long-term Spark task. All statements in the spark-beeline are submitted to the task for execution.

  Example of starting an instance:

  cd $SPARK_HOME/bin

  spark-beeline

  Example of SQL:

  a. ***CREATE TABLE info(id int, name string, company string);***

  b. ***INSERT INTO TABLE info values(001,'jack','huawei');***

  c. ***SELECT * FROM info;***

- ***beeline***

  Call Spark JDBCServer to execute Spark SQL to efficiently compute and analyze massive amounts of data. JDBCServer contains a long-term Spark task. All statements in the beeline are submitted to the task for execution.

  For security clusters with Kerberos authentication enabled

  cd $SPARK_HOME/bin

  ./beeline -u 'jdbc:hive2://ha-cluster/default;user.principal=spark/hadoop.COM;saslQop=auth-conf;auth=KERBEROS;principal=spark/hadoop.COM;'

NOTE

The **spark/hadoop.COM** character string is obtained from the principal character string displayed by running the **klist -kt /opt/Bigdata/MRS_*XXX*/1_20_SparkResource/etc/spark.keytay** command in the cluster. You can paste the character string to the beeline command.

For normal clusters with Kerberos authentication disabled

cd $SPARK_HOME/bin

beeline

Example of SQL:

a. ***CREATE TABLE info(id int, name string, company string);***

b. ***INSERT INTO TABLE info values(001,'jack','huawei');***

c. ***SELECT * FROM info;***

You are advised to use spark-beeline because it is encapsulated based on beeline, which can be directly executed.

- ***run-example***

  This command is used to run or debug the built-in sample code in the Spark open source community.

  Example: Running SparkPi

  run-example --master yarn --deploy-mode client SparkPi 100

14.5.2 Spark Application Tuning

14.5.2.1 Spark Core Tuning

14.5.2.1.1 Data Serialization

Scenario

Spark supports the following types of serialization:

- JavaSerializer
- KryoSerializer

Data serialization greatly affects the Spark application performance. In specific data format, KryoSerializer offers 10 times higher performance than JavaSerializer. For Int data, performance optimization can be ignored.

KryoSerializer depends on Chill of Twitter. Not all Java Serializable objects support KryoSerializer. Therefore, a class must be manually registered.

Serialization involves task serialization and data serialization. Only JavaSerializer can be used for Spark task serialization. JavaSerializer and KryoSerializer can be used for data serialization.

Procedure

When the Spark application is running, a large volume of data needs to be serialized during the shuffle and RDD cache procedures. By default, JavaSerializer

is used. You can also configure KryoSerializer as the data serializer to improve serialization performance.

When developing an application, add the following code to enable KryoSerializer as data serializer:

- Implement the class register and manually register classes.

```
package com.etl.common;

import com.esotericsoftware.kryo.Kryo;
import org.apache.spark.serializer.KryoRegistrator;

public class DemoRegistrator implements KryoRegistrator
{
   @Override
   public void registerClasses(Kryo kryo)
   {
      // The following is an example class. Please register a customized class.
      kryo.register(AggrateKey.class);
      kryo.register(AggrateValue.class);
   }
}
```

  You can configure **spark.kryo.registrationRequired** on a Spark client to determine whether registration with KryoSerializer is required.

  If the parameter is set to **true**, an exception is thrown if a project has classes that are not serialized. If the parameter is set to **false** (default value), KryoSerializer automatically writes unregistered classes to the corresponding objects. This operation affects system performance. If the parameter is set to **true**, you must manually register classes. The system does not write classes that are not serialized but throws exceptions. System performance is not affected.

- Configure KryoSerializer as the data serializer and class register.

```
val conf = new SparkConf()
conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
.set("spark.kryo.registrator", "com.etl.common.DemoRegistrator")
```

14.5.2.1.2 Memory Configuration Optimization

Scenario

Spark is an in-memory computing frame. If the memory is insufficient during computing, the Spark execution efficiency will be adversely affected. You can determine whether memory becomes a performance bottleneck by monitoring garbage collection (GC) and evaluating the resilient distributed dataset (RDD) size in the memory, and take performance optimization measures.

To monitor GC of node processes, add the **-verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps** parameter to the **spark.driver.extraJavaOptions** and **spark.executor.extraJavaOptions** configuration items in the **conf/spark-defaults.conf** configuration file of the client.

If Full GC is frequently reported, GC needs to be optimized. Cache the RDD and query the RDD size in the log. If a large value is found, change the RDD storage level.

Procedure

- To optimize GC, adjust the size and ratio of the old generation and young generation. In the **conf/spark-defaults.conf** configuration file of the client,

add the -XX:NewRatio parameter to the **spark.driver.extraJavaOptions** and **spark.executor.extraJavaOptions** configuration items. For example, if you add **-XX:NewRatio=2**, the young generation accounts for 1/3 of the heap space, and the old generation accounts for 2/3.

- Optimize the RDD data structure when developing Spark applications.
  - Use primitive arrays to replace fastutil arrays.
  - Avoid nested structure.
  - Avoid using String in keys.
- Serialize RDDs when developing Spark applications.

  By default, data is not serialized when RDDs are cached. You can set the storage level to serialize the RDDs and minimize memory usage. Example:

  ```
  testRDD.persist(StorageLevel.MEMORY_ONLY_SER)
  ```

14.5.2.1.3 Setting a Degree of Parallelism

Scenario

A degree of parallelism (DOP) specifies the number of tasks to be executed concurrently. It determines the number of data blocks after the shuffle operation. Adjust the DOP to optimize the number of tasks, the data processed by each task, and the processing capability of the machine.

Query the CPU and memory usage. If the tasks and data are not evenly distributed among nodes, increase the DOP. Generally, set the DOP to two or three times that of the total CPUs in the cluster.

Procedure

You can use any of the following methods to set the DOP and adjust the DOP parameters according to the actual memory, CPU, data, and application logic:

- Set the DOP parameters in the function of shuffle operations. This method has the highest preference.

  ```
  testRDD.groupByKey(24)
  ```

- Set the **spark.default.parallelism** parameter in the code. This method has the second highest preference.

  ```
  val conf = new SparkConf()
  conf.set("spark.default.parallelism", 24)
  ```

- Set the **spark.default.parallelism** parameter in the **$SPARK_HOME/conf/spark-defaults.conf** file. This method has the lowest preference.

  ```
  spark.default.parallelism    24
  ```

14.5.2.1.4 Using Broadcast Variables

Scenario

Broadcast distributes data sets to each node. It allows data to be obtained locally when a data set is needed during a Spark task. If broadcast is not used, data serialization will be scheduled to tasks each time when a task requires data sets. It is time-consuming and makes the task get bigger.

1. If a data set will be used by each slice of a task, broadcast the data set to each node.

2. When small and big tables need to be joined, broadcast small tables to each node. This eliminates the shuffle operation, changing the join operation into a common operation.

Procedure

When developing an application, add the following code to broadcast the testArr data to each node:

```
def main(args: Array[String]) {
  ...
  val testArr: Array[Long] = new Array[Long](200)
  val testBroadcast: Broadcast[Array[Long]] = sc.broadcast(testArr)
  val resultRdd: RDD[Long] = inpputRdd.map(input => handleData(testBroadcast, input))
  ...
}

def handleData(broadcast: Broadcast[Array[Long]], input: String) {
  val value = broadcast.value
  ...
}
```

14.5.2.1.5 Using the External Shuffle Service to Improve Performance

Scenario

When the Spark system runs applications that contain a shuffle process, an executor process also writes shuffle data and provides shuffle data for other executors in addition to running tasks. If the executor is heavily loaded and GC occurs, the executor cannot provide shuffle data for other Executors, affecting task running.

The external shuffle service is an auxiliary service in NodeManager. It captures shuffle data to reduce the load on executors. If GC occurs on an executor, tasks on other executors are not affected.

Procedure

1. Enable the external shuffle service on NodeManager.

   a. On MRS Manager (for details about how to log in to MRS Manager, see **Logging in to MRS Manager**), choose **Services** > **Yarn** > **Service Configuration** and choose **Yarn** > **Customize** to add the following configuration items to **yarn-site.xml**:

   ```
   <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>spark_shuffle</value>
   </property>
   <property>
      <name>yarn.nodemanager.aux-services.spark_shuffle.class</name>
      <value>org.apache.spark.network.yarn.YarnShuffleService</value>
   </property>
   ```

Parameter	Description
yarn.nodemanager.aux-services	A long-term auxiliary service in NodeManager for improving shuffle computing performance

Parameter	Description
yarn.nodemanager.aux-services.spark_shuffle.class	Class of an auxiliary service in NodeManager

b. Add a dependency JAR file.

Copy **${SPARK_HOME}/lib/spark-1.5.1-yarn-shuffle.jar** to the **${HADOOP_HOME}/share/hadoop/yarn/lib/** directory.

c. Restart the NodeManager process so that the external shuffle service is started.

2. Apply the external shuffle service to Spark applications.

– Add the following configuration items to the client installation directory **/Spark/spark/conf/spark-defaults.conf**:

```
spark.shuffle.service.enabled   true
spark.shuffle.service.port      7337
```

Parameter	Description
spark.shuffle.service.enabled	A long-term auxiliary service in NodeManager for improving shuffle computing performance The default value is **false**, indicating that this function is disabled.
spark.shuffle.service.port	Port for the shuffle service to monitor requests for obtaining data. This parameter is optional and its default value is **7337**.

NOTE

1. If the **yarn.nodemanager.aux-services** configuration item exists, add **spark_shuffle** to its value. Use a comma to separate this value from other values.

2. The value of **spark.shuffle.service.port** must be the same as that in the **yarn-site.xml** file.

14.5.2.1.6 Configuring Dynamic Resource Scheduling in Yarn Mode

Scenario

Resources are a key factor that affects Spark execution efficiency. If multiple executors are allocated to a long-term service (for example, JDBCServer) that has no task but resources of other applications are insufficient, these resources are wasted and improperly scheduled.

Dynamic resource scheduling can add or remove executors of applications in real time based on the task load. In this way, resources are dynamically scheduled to applications.

Procedure

1. You need to configure the external shuffle service first. For details, see **Using the External Shuffle Service to Improve Performance**.
2. In the **spark-defaults.conf** file, add the **spark.dynamicAllocation.enabled** configuration item and set its value to **true** to enable dynamic resource scheduling. This function is disabled by default.
3. **Table 14-30** lists some optional configuration items.

Table 14-30 Parameters for dynamic resource scheduling

Configuration Item	Description	Default Value
spark.dynamicAllocation.minExecutors	Minimum number of executors	0
spark.dynamicAllocation.initialExecutors	Initial number of executors	spark.dynamicAllocation.minExecutors
spark.dynamicAllocation.maxExecutors	Maximum number of executors	Integer.MAX_VALUE
spark.dynamicAllocation.schedulerBacklogTimeout	First timeout interval for scheduling	1(s)
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout	Second and later timeout interval for scheduling	spark.dynamicAllocation.schedulerBacklogTimeout
spark.dynamicAllocation.executorIdleTimeout	Idle timeout interval for common executors	60(s)
spark.dynamicAllocation.cachedExecutorIdleTimeout	Idle timeout interval for executors with cached blocks	Integer.MAX_VALUE

NOTE

- The external shuffle service must be configured before dynamic resource scheduling is enabled. If the external shuffle service is not configured, shuffle files are lost when an executor is killed.
- If **spark.executor.instances** or **--num-executors** specifies the number of Executor, the dynamic resource allocation will not take effect even if it is enabled.
- After dynamic resource scheduling is enabled, a task may be allocated to an executor to be removed, resulting in a task failure. After the same task fails for four times (can be configured by the **spark.task.maxFailures** parameter), the job fails. In practice, it is unlikely that a task is allocated to executors to be removed. In addition, the probability of job failure can be reduced by increasing the value of **spark.task.maxFailures**.

14.5.2.1.7 Configuring Process Parameters

Scenario

There are three processes in Spark on YARN mode: driver, ApplicationMaster, and executor. During task scheduling and running, the driver and executor have major responsibilities, and ApplicationMaster is responsible for starting and stopping containers.

Therefore, the parameter settings of the driver and executor greatly affect the execution of Spark applications. You can perform the following operations to optimize Spark cluster performance.

Procedure

Step 1 Configure driver memory.

The driver schedules tasks and communicates with the executor and ApplicationMaster. When the number of tasks and the task parallelism increase, the driver memory needs to be increased accordingly.

You can set a proper memory for the driver based on the actual number of tasks.

- Set **spark.driver.memory** in **spark-defaults.conf** or **SPARK_DRIVER_MEMORY** in **spark-env.sh** to a proper value.
- When you run the **spark-submit** command, add the **--driver-memory MEM** parameter to set the memory.

Step 2 Configure the number of executors.

Every core of each executor can run one task at the same time. Therefore, increasing the number of executors increases the concurrency of tasks. When resources are sufficient, you can increase the number of executors to improve running efficiency.

- Set **spark.executor.instance** in **spark-defaults.conf** or **SPARK_EXECUTOR_INSTANCES** in **spark-env.sh** to a proper value. You can also set the dynamic resource scheduling function for optimization.
- When you run the **spark-submit** command, add the **--num-executors NUM** parameter to set the number of executors.

Step 3 Configure the number of executor cores.

Multiple cores of an executor can run multiple tasks at the same time, which increases the task concurrency. However, because all cores share the memory of an executor, you need to balance the memory and the number of cores.

- Set **spark.executor.cores** in **spark-defaults.conf** or **SPARK_EXECUTOR_CORES** in **spark-env.sh** to a proper value.
- When you run the **spark-submit** command, add the **--executor-cores NUM** parameter to set the number of executor cores.

Step 4 Configure executor memory.

The executor memory is used for task execution and communication. You can increase the memory for a big task that needs more resources, and reduce the memory to increase the concurrency level for a small task that runs fast.

- Set **spark.executor.memory** in **spark-defaults.conf** or **SPARK_EXECUTOR_MEMORY** in **spark-env.sh** to a proper value.
- When you run the **spark-submit** command, add the **--executor-memory MEM** parameter to set the memory.

----End

Examples

- During the **spark wordcount** calculation, the amount of data is 1.6 TB and the number of the executors is 250.

  The execution fails under the default configuration, and the **Futures timed out** and **OOM** errors occur.

  The causes are as follows: The data volume is large and there are many tasks. Each task of wordcount is small and can be quickly completed. When the number of tasks increases, some objects on the driver side become larger. In addition, when each task is complete, the executor and driver communicate with each other. As a result, the memory is insufficient and the communication between processes is interrupted.

  When the driver memory is set to 4 GB, the application is successfully executed.
- When using Thrift Server to execute the TPC-DS test suite, many errors such as **Executor Lost** are reported under default parameter configuration. When there is 30 GB of driver memory, 2 executor cores, 125 executors, and 6 GB of executor memory, all tasks can be successfully executed.

14.5.2.1.8 Designing a Direction Acyclic Graph (DAG)

Scenario

Proper application design improves the execution efficiency. Reduce shuffle operations if possible during programming and combine narrow dependency operations.

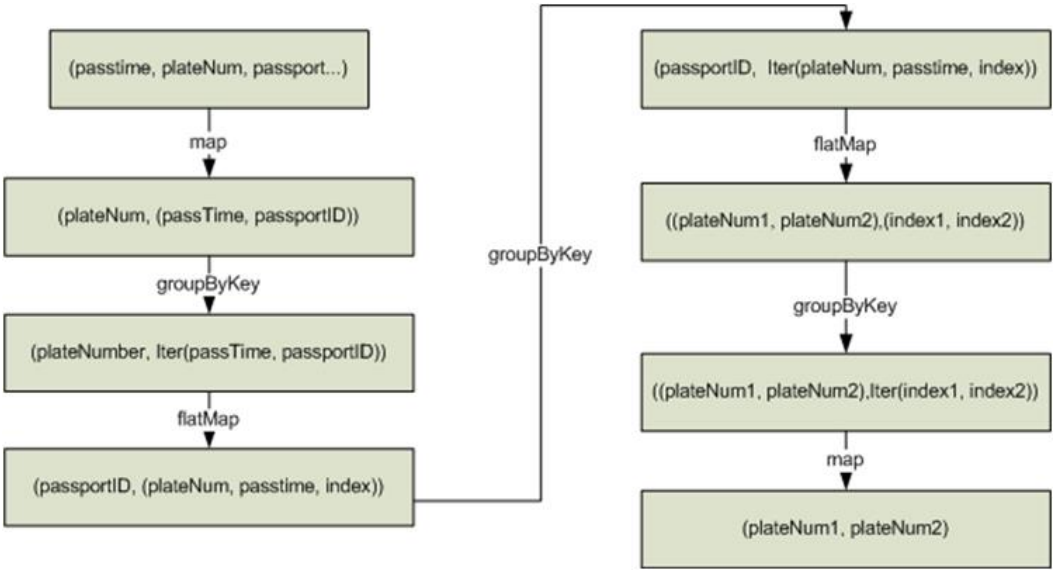
Procedure

The following example describes how to determine whether two vehicles are peers to show the DAG design principles.

- **Data format**: Time when a vehicle passes a toll station, license plate number, and toll station number...
- **Logic**: Two vehicles are considered as peers if they meet the following conditions:
  - Both vehicles pass the same toll stations in the same sequence.
  - The difference between the time that the vehicles pass the same toll station is smaller than a specified value.

There are two implementation modes for this example. **Figure 14-36** shows logic of implementation mode 1 and **Figure 14-37** shows logic of implementation mode 2.

Figure 14-36 Logic of implementation mode 1



Logic description:

1. Based on the license template number, aggregate and sequence all toll stations that a vehicle passes. The following data is obtained after processing:

   License plate No.1, [(pass time, toll station 3), (pass time, toll station 2), (pass time, toll station 4), (pass time, toll station 5)]

2. Identify the sequence number of the toll stations passed by the vehicle.

   (Toll station 3, (License plate No.1, pass time, first toll station))

   (Toll station 2, (License plate No.1, pass time, second toll station))

   (Toll station 4, (License plate No.1, pass time, third toll station))

   (Toll station 5, (License plate No.1, pass time, fourth toll station))

3. Aggregate data based on the toll stations.

   Toll station 1, [(License plate No.1, pass time, first toll station), (License plate No.2, pass time, fifth toll station), (License plate No.3, pass time, second toll station)]

4. Checks whether the difference between the time that two vehicles pass a toll station is within the range specified for determining peers. If the time difference is smaller than the specified value, collect information about the two vehicles.

   (License plate No.1, License plate No.2), (first toll station, fifth toll station)

   (License plate No.1, License plate No.3), (first toll station, second toll station)

5. Aggregate information data about two vehicles that pass the same toll stations based on their license template number.

   (License plate No.1, License plate No.2), [(first toll station, fifth toll station), (second toll station, sixth toll station), (first toll station, seventh toll station), (third toll station, eighth toll station)]

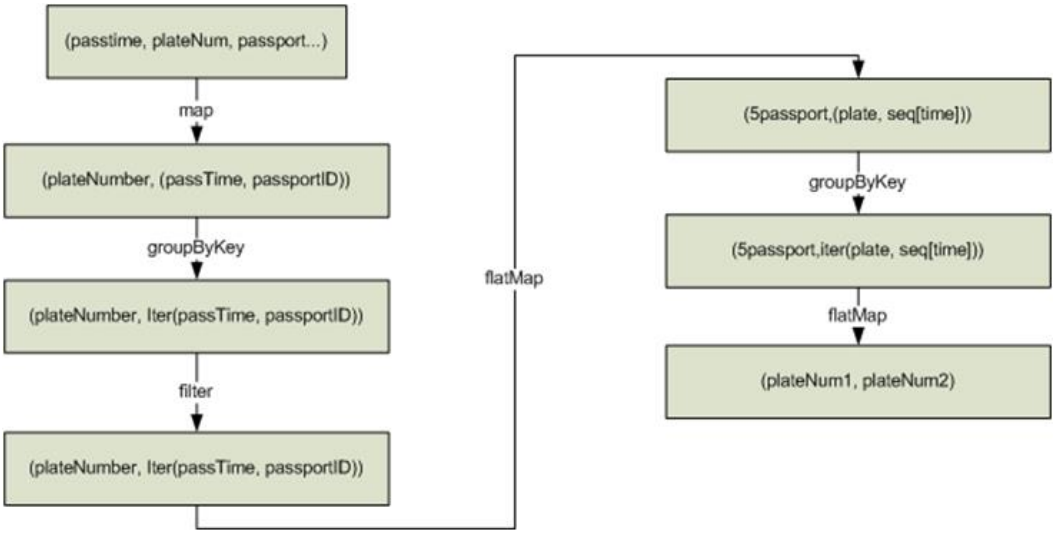
6. If the vehicles with license plate No.1 and No.2 pass the same toll stations in sequence, (for example, toll stations 3, 4, and 5 are the first, second, and third ones passed by vehicle 1, and the sixth, seventh, and eighth ones passed by

vehicle 2), and the number of passed toll stations reaches the specified value, the two vehicles are considered as peers.

Disadvantages of implementation mode 1:

- The logic is complex.
- A large number of shuffle operations are performed, deteriorating performance.

Figure 14-37 Logic of implementation mode 2



Logic description:

1. Based on the license template number, aggregate and sequence all toll stations that a vehicle passes. The following data is obtained after processing:

   License plate No.1, [(pass time, toll station 3), (pass time, toll station 2), (pass time, toll station 4), (pass time, toll station 5)]

2. Based on the number of toll stations required for determining peers (3 toll stations in this example), segment the sequence of toll stations passed by a vehicle. For example, the preceding information is segmented into the following:

   Toll station 3 > Toll station 2 > Toll station 4, (License plate No.1, [pass time at toll station 3, pass time at toll station 2, pass time at toll station 4])

   Toll station 2 > Toll station 4 > Toll station 5, (License plate No.1, [pass time at toll station 2, pass time at toll station 4, pass time at toll station 5])

3. Aggregate information about vehicles that pass the same toll stations in the same sequence.

   Toll station 3 > Toll station 2 > Toll station 4, [(License plate No.1, [pass time at toll station 3, pass time at toll station 2, pass time at toll station 4]), (License plate No.2, [pass time at toll station 3, pass time at toll station 2, pass time at toll station 4]), (License plate No.3, [pass time at toll station 3, pass time at toll station 2, pass time at toll station 4])]

4. Determine whether the time difference that these vehicles passed through the same toll station is smaller than the specified value. If yes, the vehicles are determined to be peers.

Advantages of implementation mode 2:

- The logic is simplified.
- A **groupByKey** is removed, that is, a shuffle operation is deducted. This improves performance.

14.5.2.1.9 Experience Summary

Using mapPartitions to Calculate Data by Partition

If the overhead of each record is high, for example,

```
rdd.map{x=>conn=getDBConn;conn.write(x.toString);conn.close}
```

use mapPartitions to calculate data by partition.

```
rdd.mapPartitions(records => conn.getDBConn;for(item <- records)
write(item.toString); conn.close)
```

mapPartitions can flexibly operate data. For example, to calculate the top N of a large data block, mapPartitions can be used to calculate the top N of each partition and then sort the top N of all partitions if N is a small value. Compared with calculating top N with full data, this method has a higher efficiency.

Using coalesce to Adjust the Number of Slices

Use coalesce to adjust the number of slices. The coalesce function has two parameters.

```
coalesce(numPartitions: Int, shuffle: Boolean = false)
```

If the value of shuffle is **true**, coalesce has the same function as repartition(numPartitions: Int). It recreates partitions using the shuffle. If the value of shuffle is set to **false**, partitions of a parent RDD are calculated in the same task. In this case, if the value of **numPartitions** is greater than the number of slices of the parent RDD, partitions are not recreated.

The coalesce operator can be used in the following scenarios:

- If the previous operation involves a large number of filters, use coalesce to minimize the number of zero-loaded tasks. In coalesce (numPartitions, false), the value of **numPartitions** is smaller than the number of slices of the parent RDD.
- Use coalesce when the number of slices entered is too large to execute.
- Use coalesce when the programs are suspended in the shuffle operation because of a large number of tasks or limited Linux resources. In this case, use coalesce (numPartitions, true) to recreate partitions.

Configuring localDir

During the shuffle procedure of Spark, data needs to be written into local disks. The performance bottleneck of Spark is shuffle, and the bottleneck of shuffle is the I/O. To improve the I/O performance, you can configure multiple disks to implement concurrent data writing. If multiple disks are mounted to a node, configure a Spark localDir for each disk. This can effectively distribute shuffle files in multiple locations, improving disk I/O efficiency. The performance cannot be improved if a disk is configured with multiple directories.

Using the Collect Operation for Small Data

The collect operation does not apply to a large volume of data.

When the collect operation is performed, the executor data is sent to the driver. If the driver does not have sufficient memory, **OutOfMemory** occurs on the driver. Therefore, if the data volume is unknown, perform the saveAsTextFile operation to write data into HDFS. If the data volume is known and the driver has sufficient memory, perform the collect operation.

Using reduceByKey

The reduceByKey operator implements local aggregation on the Map side, which offers a smooth shuffle procedure. The groupByKey operator, however, does not perform aggregation on the Map side. Therefore, use reduceByKey if possible to avoid implementation modes like groupByKey().map(x=>(x._1,x._2.size)).

Broadcasting Map Instead of Arrays

If a table query is required for each record of data that is broadcast from the driver side, broadcast the data in the set/map structure instead of Iterator. The query speed of the set/map structure is approximately O(1), while that of Iterator is O(n).

Avoiding Data Skew

If data skew occurs (certain data volume is extremely large), the execution time of tasks is inconsistent even if no GC is performed.

- Redefine the keys. Use keys of smaller granularity to optimize the task size.
- Modify the DOP.

Optimizing the Data Structure

- Store data by column. In this way, only the required columns are scanned when data is read.
- When using Hash Shuffle, set **spark.shuffle.consolidateFiles** to **true** to combine intermediate files of shuffle, minimize the number of shuffle files and file I/O operations, and improve performance. The number of final files is the number of reduce tasks.

14.5.2.2 SQL and DataFrame Tuning

14.5.2.2.1 Optimizing the Spark SQL Join Operation

Scenario

When two tables are joined in Spark SQL, the broadcast function (see section **Using Broadcast Variables**) can be used to broadcast small tables to each node. This minimizes shuffle operations and improves task execution efficiency.

NOTE

The join operation refers to the inner join operation only.

Procedure

The following describes how to optimize the join operation in Spark SQL. Assume that both tables A and B have the **name** column. Join tables A and B as follows:

1. Estimate the table sizes.

   Estimate the table size based on the size of data loaded each time.

   You can also check the table size in the directory of the Hive database. In the **hive-site.xml** configuration file of Spark, view the Hive database directory, which is **/user/hive/warehouse** by default.

   ```
   <property>
     <name>hive.metastore.warehouse.dir</name>
    <value>/user/hive/warehouse</value>
   </property>
   ```

   Run the **hadoop** command to check the size of the table. For example, run the following command to view the size of table A:

   ```
   hadoop fs -du -s -h ${test.warehouse.dir}/a
   ```

   NOTE

   The tables must meet the following requirements for broadcasting:

   1. At least one table is not empty.
   2. Tables must not be external tables.
   3. The storage mode of the tables must be **textfile** (default value), for example,
      ```
      create table A( name string ) stored as textfile;
      or
      create table A( name string );
      ```

2. Configure a threshold for automatic broadcast.

   The threshold for triggering broadcast for a table is 10485760 (that is, 10 MB) in Spark. If either of the table sizes is smaller than 10 MB, skip this step.

   Table 14-31 describes the parameter for configuring the automatic broadcast threshold.

   Table 14-31 Parameter

Parameter	Default Value	Description
spark.sql.autoBroadcastJoinThreshold	10485760	Specifies the maximum value for broadcast configuration when two tables are joined. If the table size is smaller than the parameter value, broadcast is performed. If the value is set to **-1**, broadcast is not performed. For details, visit **https://spark.apache.org/docs/latest/sql-programming-guide.html**.

Configure the threshold for automatic broadcast as follows:

- Set **spark.sql.autoBroadcastJoinThreshold** in the **spark-defaults.conf** configuration file of Spark. The value of **<size>** varies with scenarios and must be greater than the size of at least one table.
  ```
  spark.sql.autoBroadcastJoinThreshold = <size>
  ```
- Run the Hive command to set the threshold. Before joining the tables, run the following command:
  ```
  SET spark.sql.autoBroadcastJoinThreshold=<size>
  ```
  The value of **<size>** varies with scenarios and must be greater than the size of at least one table.

3. Join the tables.

   In this example, the size of at least one table is smaller than the threshold.

   If the sizes of both tables A and B are smaller than the threshold and the size of table A is smaller than that of table B, run the following command:

   ```
   SELECT A.name FROM B JOIN A ON A.name = B.name;
   ```

   If the size of table B is smaller than that of table A, run the following command:

   ```
   SELECT A.name FROM A JOIN B ON A.name = B.name;
   ```

14.5.2.2.2 Optimizing INSERT...SELECT Operation

Scenario

The INSERT...SELECT operation can be optimized in the following scenarios:

- Data in a large number of small files is queried.
- Data in large files is queried.
- A non-Spark user is used in beeline/thriftserver mode.

Procedure

The INSERT...SELECT operation can be optimized as follows:

- When creating a Hive table, set the storage type to Parquet to accelerate execution of the INSERT...SELECT statement.
- Use **spark-sql** or a Spark user in beeline/thriftserver mode to execute INSERT...SELECT operations. This eliminates the need for changing the file owner, which quickens INSERT...SELECT statement execution.

  NOTE

  In beeline/thriftserver mode, an executor and a driver are run by the same user. Because a driver is a part of ThriftServer and ThriftServer is run by a Spark user, the driver is also run by the Spark user. At present, the user of the beeline client cannot be transparently transmitted to the executor during operation. If a non-Spark user is used, the owner of a file must be changed to the user of the beeline client, that is, the actual user.

14.5.2.3 Spark Streaming Tuning

Scenario

Streaming is a mini-batch streaming processing framework that features second-level delay and high throughput. To optimize Streaming is to improve its throughput while maintaining second-level delay so that more data can be processed per unit time.

NOTE

This section applies to the scenario where the input data source is Kafka.

Procedure

A simple streaming processing system consists of a data source, a receiver, and a processor. The data source is Kafka, the receiver is the Kafka data source receiver of Streaming, and the processor is Streaming.

Streaming optimization is to optimize the performance of the three components.

- **Data source optimization**

  In actual application scenarios, the data source stores the data in the local disks to ensure the error tolerance of the data. However, the calculation results of the Streaming are stored in the memory, and the data source may become the largest bottleneck of the streaming system.

  Kafka can be optimized from the following aspects:

  - Use Kafka-0.8.2 or later version that allows you to use new Producer interfaces in asynchronous mode.
  - Configure multiple Broker directories, multiple I/O threads, and a proper number of partitions for a topic.

  For details, see section **Performance Tuning** in the Kafka open source documentation at **http://kafka.apache.org/documentation.html**.

- **Receiver optimization**

  Streaming has multiple data source receivers, such as Kafka, Flume, MQTT, and ZeroMQ. Kafka has the most receiver types and is the most mature receiver.

  Kafka provides three types of receiver APIs:

  - KafkaReceiver: directly receives Kafka data. If the process is abnormal, data may be lost.
  - ReliableKafkaReceiver: ZooKeeper records the received data displacement.
  - DirectKafka: reads data from each partition of Kafka through the RDD, ensuring high reliability.

  According to the implementation mechanism and test results, DirectKafka provides better performance than the other two APIs. Therefore, the DirectKafka API is recommended to implement the receiver.

  For details about the Kafka receivers and their optimization methods, see the Kafka open source documentation at **http://kafka.apache.org/documentation.html**.

- **Processor optimization**

  The bottom layer of Streaming is executed by Spark. Therefore, most optimization measures for Spark can also be applied to Streaming. For example:

  – Data serialization
  – Memory configuration
  – Setting a DOP
  – Using the external shuffle service to improve performance

  NOTE

  Higher performance of Spark Streaming indicates lower overall reliability. Example:

  If **spark.streaming.receiver.writeAheadLog.enable** is set to **false**, disk I/Os are reduced and performance is improved. However, because WAL is disabled, data is lost during fault recovery.

  Therefore, do not disable configuration items that ensure data reliability in production environments during Streaming optimization.

14.5.2.4 Spark CBO Tuning

Scenario

An SQL query compiler is responsible for converting SQL statements into execution plans, while an optimizer instructs the SQL query compiler to select the most efficient execution plan. Traditional databases (for example, Oracle) support two types of optimizers: Rule-Based Optimization (RBO) and Cost-Based Optimization (CBO).

- RBO

  Rules of RBO are formed based on experience. Execution plans for SQL statements following the RBO rules are not affected by contents or data distribution in tables.

- CBO

  Rules of CBO are determined by data distribution and organization. The cost of each execution plan is evaluated and the plan with the lowest cost is selected.

Currently, all Spark optimizers are RBO-based and have dozens of optimization rules, for example, predicate pushdown, constant folding, and projection tailoring. These rules are valid but insensitive to data. When data distribution in a table changes, RBO is not aware of the changes and the execution plan generated by RBO is not the optimal. In comparison, CBO calculates SQL statements based on actual data distribution. It generates a group of execution plans and selects the one with the lowest cost to improve performance.

Join algorithm selection is a major improvement in CBO compared with RBO. For example, when two tables are joined, if the result set of a large table is smaller than the threshold of BroadCast after the filter operation is performed, without CBO, the changes cannot be detected and the SortMergeJoin algorithm is used, which involves a large number of shuffle operations and deteriorates performance. However, with CBO, the changes can be detected and the BroadcastHashJoin algorithm is used to broadcast the small tables to every node. This involves no shuffle operation and greatly improves performance.

Procedure

Based on table and column statistics, Spark CBO calculates the sizes of intermediate result sets generated by operators and then selects the optimal execution plan according to the calculation result.

1. Configure parameters.
   - Add the **spark.sql.cbo** configuration item to the **spark-defaults.conf** configuration file and set it to **true**. The default value is **false**.
   - Run the ***set spark.sql.cbo=true*** SQL statement on the client.
2. Run commands to obtain the statistics.

   NOTE

   Perform this step once before running all SQL statements. If a data set changes (added, updated, or deleted), you must run the commands again to obtain the latest statistics and data distribution information to ensure CBO optimization effects.

   - For tables, run the ***COMPUTE STATS FOR TABLE src*** command to obtain the table statistics, including the number of records, number of files, and physical storage capacity.
   - For columns:
     - Run the ***COMPUTE STATS FOR TABLE src ON COLUMNS*** command to obtain the statistics of all columns.
     - Run the ***COMPUTE STATS FOR TABLE src ON COLUMNS*** *name,age* command to obtain the statistics of the name and age fields.

       Four types of column statistics are supported: number, date, time, and character string. The number, data, and time statistics consist of the maximum value, minimum value, number of distinct values, number of null values, and histogram (equi-width or equi-height histogram). The character string statistics consist of the maximum value, minimum value, maximum length, average length, number of distinct values, number of null values, and histogram (equi-width histogram only).
3. Optimize CBO.
   - Automatic optimization: The system determines whether the input SQL statements can be optimized and automatically selects the optimization algorithm.
   - Manual optimization: You can run the ***DESC FORMATTED src*** command to view the statistics and then manually optimize the SQL statements based on data distribution.

14.5.3 How Do I Add a Dependency Package with Customized Codes?

Question

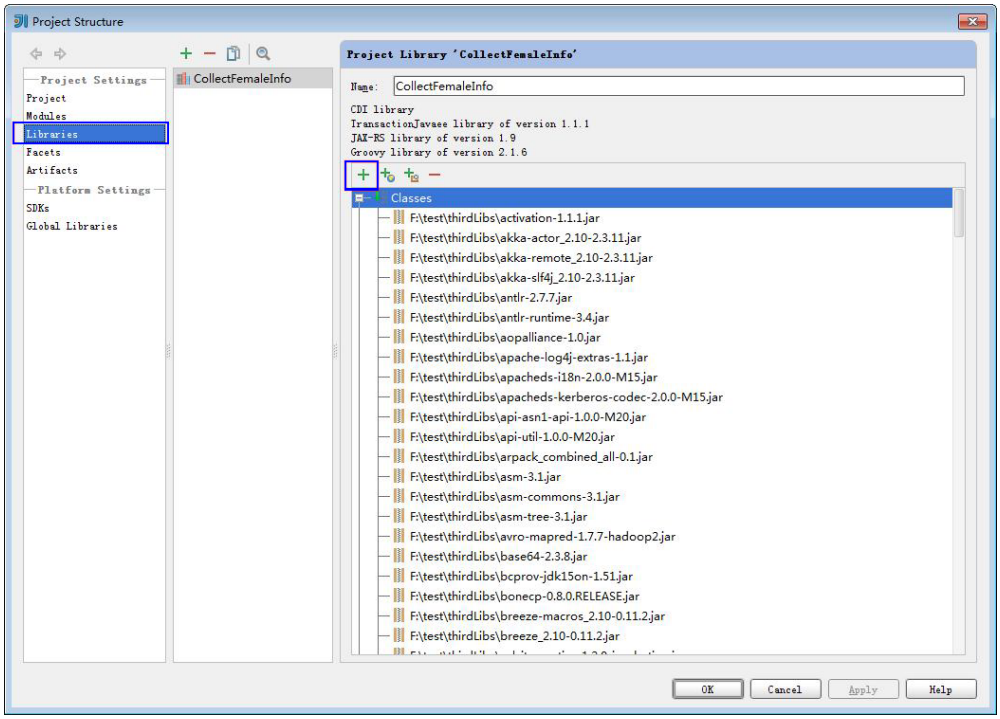
When developing Spark applications, I can add customized dependency packages except for the sample application. How do I use the IDEA to add the dependency package with customized codes to the project?

Answer

Step 1 On the IDEA homepage, choose **File** > **Project Structures...** to go to the **Project Structure** page.

Step 2 Click the **Libraries** tab. On the page that is displayed, click **+** to add a local dependency package.

Figure 14-38 Adding the dependency package.

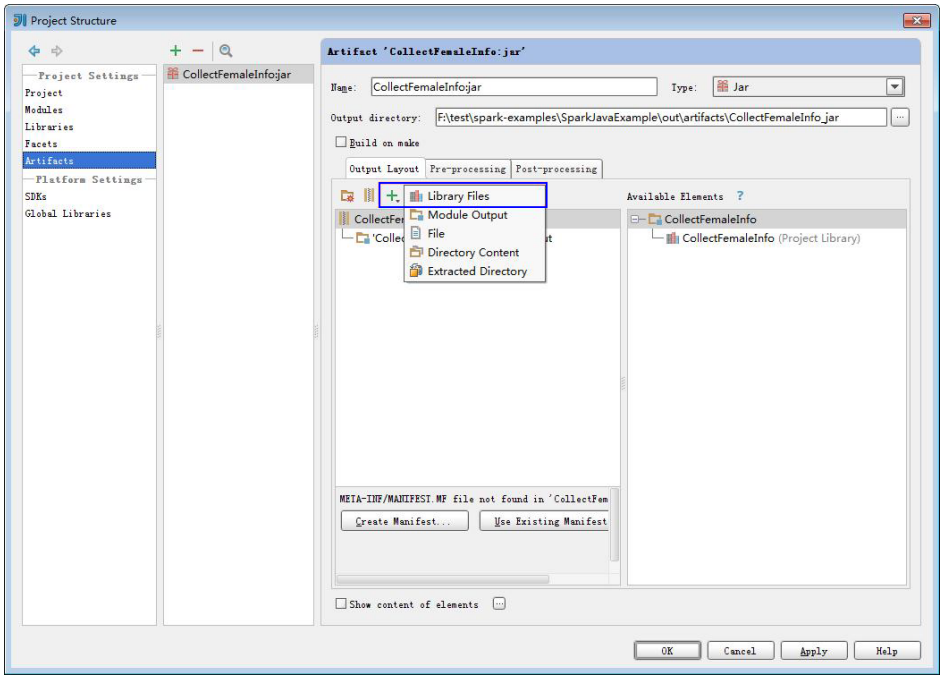


Step 3 Click **Apply** to load the dependency package and click **OK**.

Step 4 Because there is no custom dependency package in the running environment, you also need to add this dependency package when compiling the package, so that the generated JAR file contains the custom dependency package, ensuring that the Spark application can run properly.

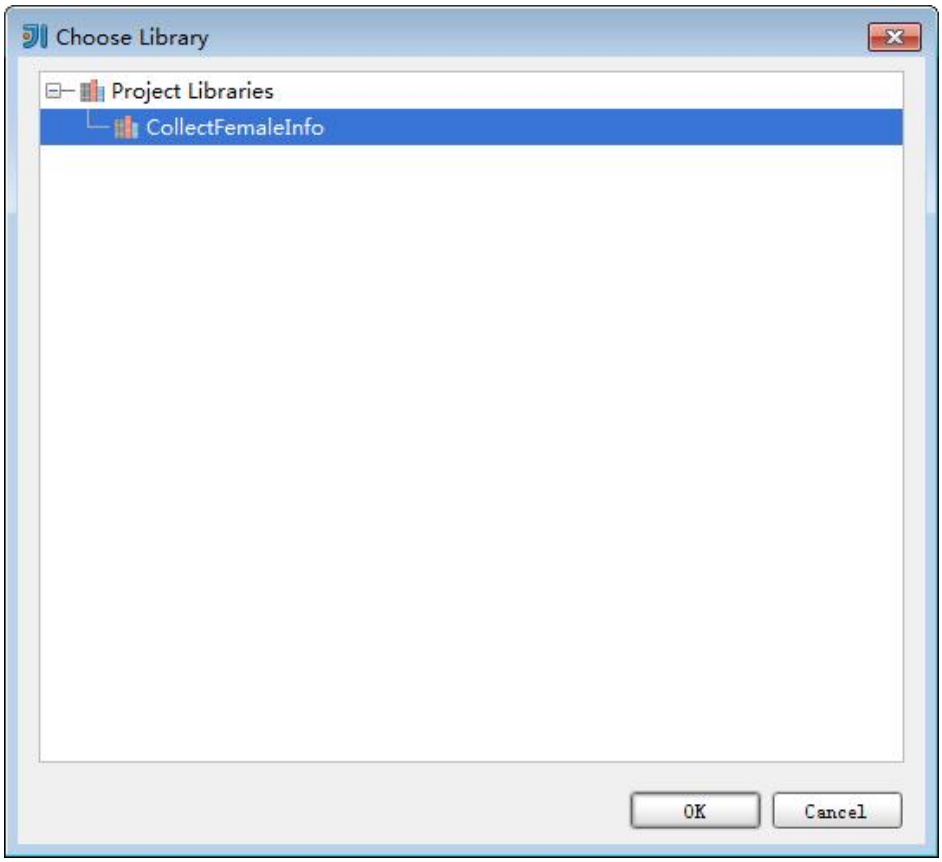
1. On the **Project Structure** page, click the **Artifacts** tab.
2. In the pane on the right, click **+** and choose **Library Files** to add a dependency package.

Figure 14-39 Adding library files



3. Select the dependency package that you want to add and click **OK**.

Figure 14-40 Choosing the library



4. Click **Apply** to load the dependency package and click **OK**.

----End

14.5.4 How Do I Handle the Dependency Package That Is Automatically Loaded?

Question

Before a project is imported using the IDEA, if the Maven has been configured in the IDEA, the tool automatically loads the dependency package specified in the Maven configuration. If the automatically loaded dependency package is not compatible with the application, the project fails to be built. How do I handle the dependency package loaded automatically?

Answer

It is recommended that you manually delete the dependency package that is automatically loaded after the project is imported. The procedure is as follows:

1. On the IDEA tool, **File** > **Project Structures...**
2. Choose **Libraries**, right-click the dependency package that is automatically loaded, and choose **Delete** from the shortcut menu.

14.5.5 Why the "Class Does not Exist" Error Is Reported While the SparkStreamingKafka Project Is Running?

Question

When 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. The KafkaWordCount sample is provided by the Spark open-source community.

Answer

When Spark is deployed, the following JAR files are saved in the **$SPARK_HOME/jars/streamingClient** directory on the client and the **/opt/Bigdata/MRS/FusionInsight-Spark-2.2.1/spark/jars/streamingClient** directory on the server.

- kafka-clients-0.8.2.1.jar
- kafka_2.10-0.8.2.1.jar
- spark-streaming-kafka_2.10-1.5.1.jar

Because **$SPARK_HOME/lib/streamingClient/*** is not added in to classpath by default, you need to configure manually.

When the application is submitted and run, add following parameters in the command:

--jars $SPARK_CLIENT_HOME/jars/streamingClient/kafka-clients-0.8.2.1.jar,$SPARK_CLIENT_HOME/jars/streamingClient/kafka_2.10-0.8.2.1.jar,$SPARK_CLIENT_HOME/jars/streamingClient/park-streaming-kafka_2.10-1.5.1.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/lib/streamingClient/***, (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 **/opt/huawei/Bigdata/FusionInsight/spark/spark/lib/streamingClient/***, to the **spark.yarn.cluster.driver.extraClassPath** parameter.
  - Delete the **spark-examples_2.10-1.5.1.jar** package from each server node.
  - In the **spark-defaults.conf** configuration file on the client, modify (or add and modify) the parameter **spark.driver.userClassPathFirst** to **true**.

14.5.6 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?

```
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 OutOfMemory (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.

Troubleshooting solution:

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:

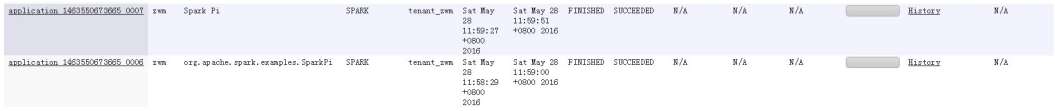
```
-XX:OnOutOfMemoryError='kill -9 %p'
```


14.5.7 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. As shown in **Figure 14-41**, 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.

Figure 14-41 Submitting the application



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.

Troubleshooting 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 lib/spark-examples*.jar 10*

14.5.8 How Do I Submit the Spark Application Using Java Commands?

Question

How do I use Java commands to submit Spark applications in addition to the **spark-submit** command?

Answer

Use the **org.apache.spark.launcher.SparkLauncher** class and run Java command to submit the Spark application. The procedure is as follows:

Step 1 Define the **org.apache.spark.launcher.SparkLauncher** class. The SparkLauncherJavaExample and SparkLauncherScalaExample are provided by

default as 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 logic and configure constant values such as the main class of the user-compiled Spark application.

If you use the normal mode, you are advised to 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 files from the SparkLauncher application and user applications, and upload the JAR files to the Spark node of the application.
   - The compilation dependency package of **SparkLauncher** is **spark-launcher_2.10-1.5.1.jar**.

  - 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 file of the application to a directory, for example, **$SPARK_HOME/lib** (the node where the application will run).

   Upload the dependency packages of the **SparkLauncher** class and the application to the lib directory on the client. The dependency package of the example code has existed in the lib directory on the client.

   NOTE

   If you want to use the **SparkLauncher** class, the node where the application runs must have the Spark client installed, and the client runs properly. The running of the **SparkLauncher** 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 using **SparkLauncher**:

   java -cp $SPARK_HOME/conf:$SPARK_HOME/lib/*:SparkLauncherExample.jar com.huawei.bigdata.spark.examples.SparkLauncherExample yarn-client /opt/female/FemaleInfoCollection.jar com.huawei.bigdata.spark.examples.FemaleInfoCollection <inputPath>

----End

14.5.9 How Does the Permission Control Mechanism Work for the UDF Function in SparkSQL?

Question

How does the permission control mechanism work for the UDF function in SparkSQL?

Answer

If the existing SQL statements cannot meet your requirements, you can use the UDF function to perform customized operations.

To ensure data security and prevent malicious codes in the UDF from damaging the system, the UDF function of SparkSQL allows only users with the **admin** permission to register. The user **admin** ensures the security of user-defined functions.

14.5.10 Why Does Kafka Fail to Receive the Data Written Back by Spark Streaming?

Question

When 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
```



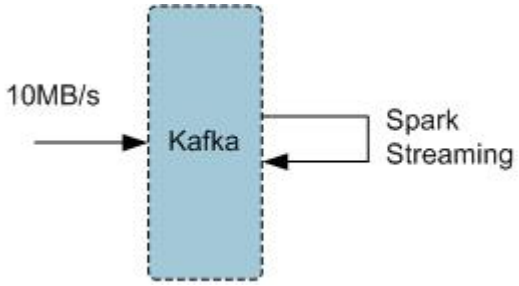
```
122371301 is not valid, it is larger than the maximum size of 104857600 bytes. | kafka.network.Processor
(Logging.scala:68)
2016-03-02 17:46:19,155 | INFO | [kafka-network-thread-21005-2] | Closing socket connection to /
10.91.8.208. | kafka.network.Processor (Logging.scala:68)
2016-03-02 17:46:19,270 | 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 a maximum of 500 MB data can be received at a time in Kafka, then the size of written-back data exceeds the threshold, triggering the error information.

Figure 14-42 Scenarios



Troubleshooting 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 MRS Manager.

14.5.11 How Do I Perform Remote Debugging Using IDEA?

Question

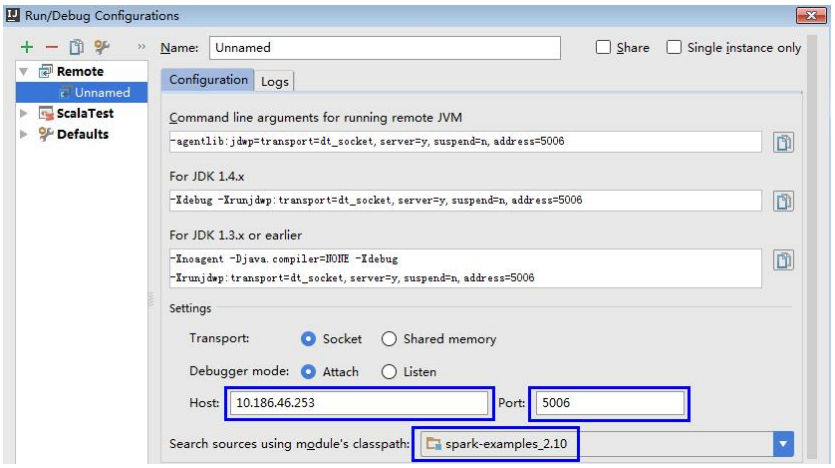
How do I perform remote debugging using IDEA during Spark secondary development?

Answer

The SparkPi application is used as an example to illustrate how to perform remote debugging using IDEA.

1. Open the project and choose **Run** > **Edit Configurations**.
2. In the displayed window, click **+** at the upper left corner. Then on the drop-down menu, choose Remote, as shown in **Figure 14-43**.

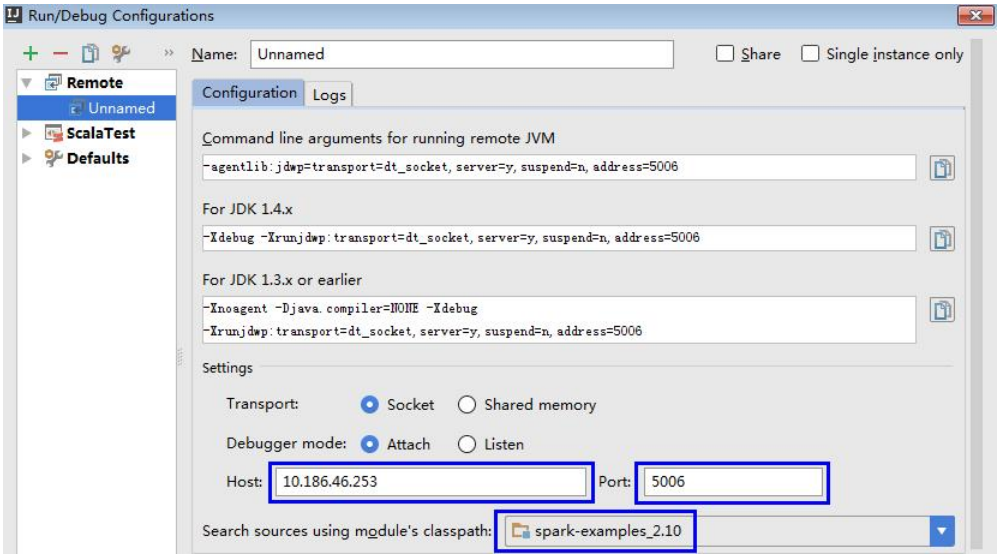
Figure 14-43 Choosing remote



3. Configure the **Host**, **Port**, and search source using module's classpath, as shown in Figure 2.

   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 14-44 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:

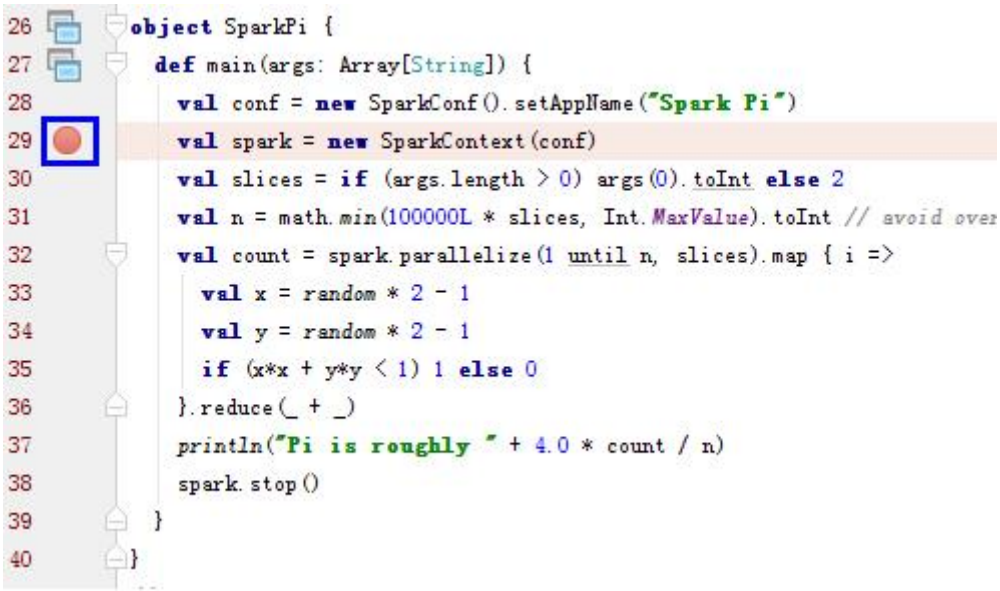
   ./spark-submit --master yarn-client --driver-java-options "-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5006" --class org.apache.spark.examples.SparkPi /opt/client/Spark/spark/examples/jars/spark-examples-<version>.jar

   - **org.apache.spark.examples.SparkPi,opt/client/Spark/spark/examples/jars/spark-examples-<version>.jar**: You need to change it to your own main class and JAR file path.
   - **-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5006**: Change the port to the commissioning port corresponding to **For JDK1.4.x** obtained in **3**.

5. Set the debugging breakpoint.

   Click the blank area on the left of the IDEA code editing window to select the breakpoint of code. Figure 4 illustrates how to select the breakpoint of the code in row 29 of **SparkPi.scala**.

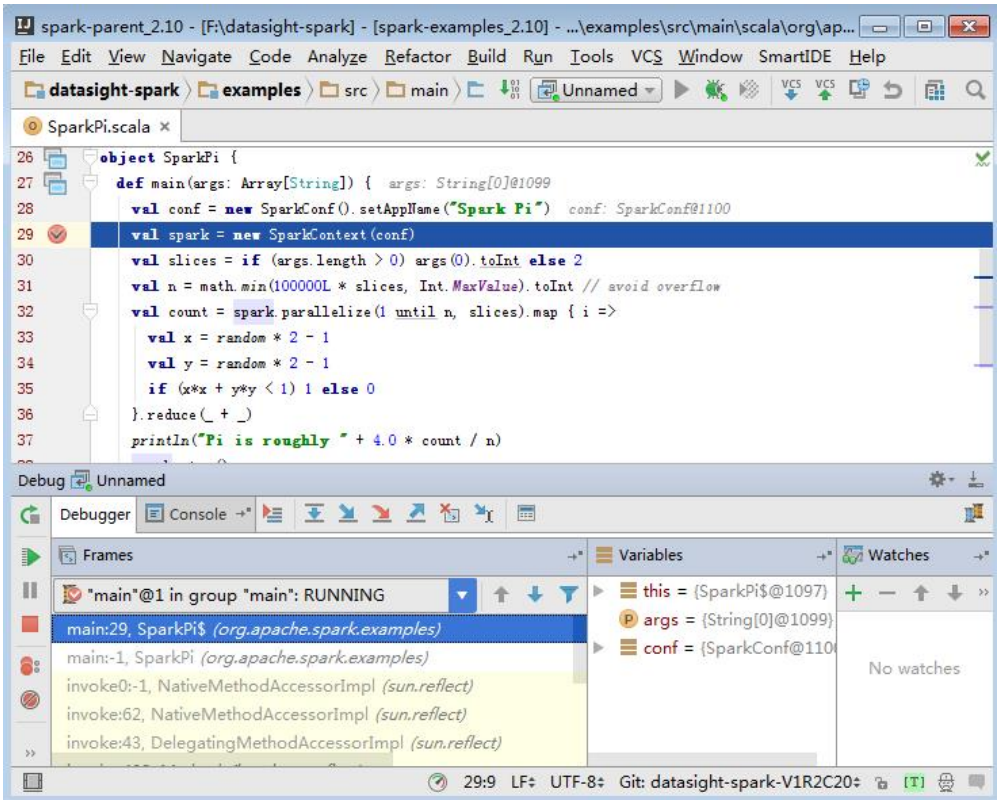
   Figure 14-45 Setting the Breakpoint



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 5.

Figure 14-46 Debugging



14.5.12 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 cause:

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, the explicit calling relogin cannot be updated.

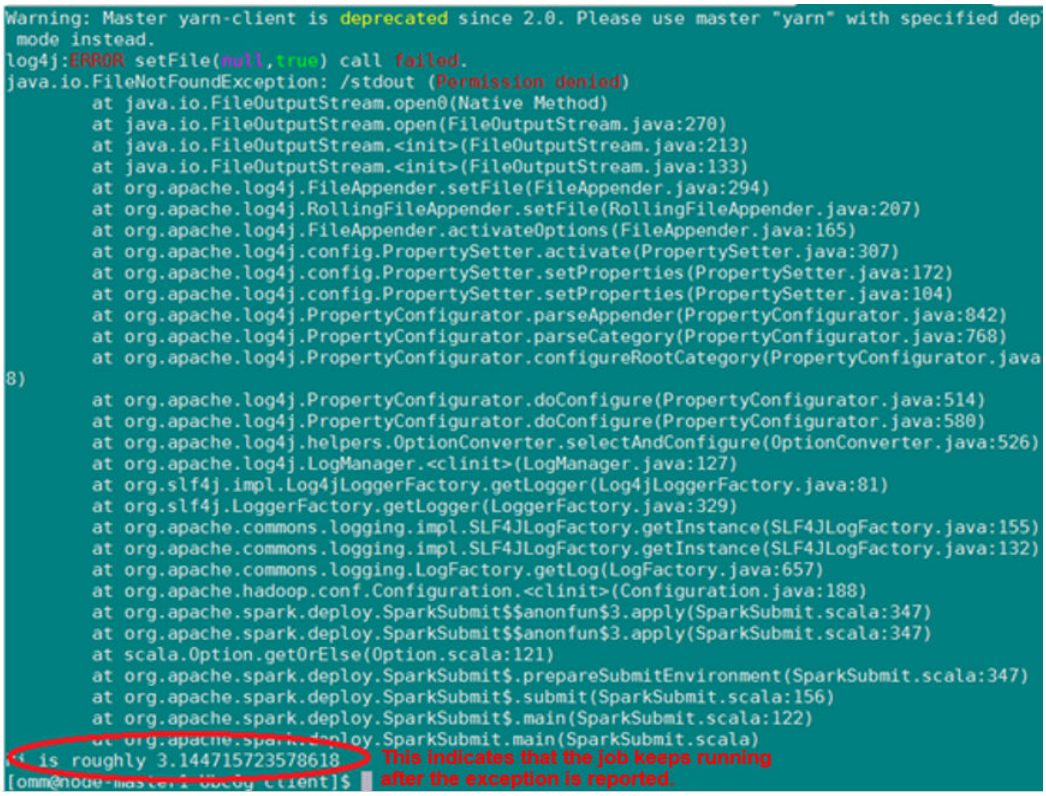
Troubleshooting solution:

When one JDBC connection fails, disable this connection, and create a new connection to perform further operations.

14.5.13 What Should I Do If FileNotFoundException Occurs When spark-submit Is Used to Submit a Job in Spark on Yarn Client Mode?

Question

When user **omm** (not user **root**) uses **spark-submit** to submit a job in yarn-client mode, the **FileNotFoundException** occurs and the job can continue running. However, the logs of the Driver program fail to be viewed. For example, after running the **spark-submit --class org.apache.spark.examples.SparkPi --master yarn-client /opt/client/Spark/spark/examples/jars/spark-examples_2.11-2.2.1-mrs-1.7.0.jar** command, the command output is shown in the following figure.



Answer

Possible Causes

When a job is executed in yarn-client mode, the Spark Driver is executed locally. The Driver log file is configured using **-Dlog4j.configuration=./log4j-executor.properties**. In the **log4j-executor.properties** configuration file, the Driver logs are outputted to the **${spark.yarn.app.container.log.dir}/stdout** file. However, when Spark Driver is executed locally, its log output directory changes to **/stdout**, because **${spark.yarn.app.container.log.dir}** is not configured. In addition, non-root users do not have the permission to create and modify **stdout** in the root directory. As a result, **FileNotFoundException** is reported. However, when a job is executed in yarn-cluster mode, the Spark Driver is executed on Application Master. When Application Master is started, a log output directory is

set using **-D${spark.yarn.app.container.log.dir}**. Therefore, FileNotFoundException is not reported when the job is executed in yarn-cluster mode.

Solution:

Note: In the following example, the default value of **$SPAKR_HOME** is **/opt/client/Spark/spark**.

Solution 1: Manually switch the log configuration file. Change the value of the **-Dlog4j.configuration=./log4j-executor.properties** configuration item (default: **./log4j-executor.properties**) of **spark.driver.extraJavaOptions** in the **$SPARK_HOME/conf/spark-defaults.conf** file. In yarn-client mode, change the value to **-Dlog4j.configuration=./log4j.properties**. In yarn-cluster mode, change the value to **-Dlog4j.configuration=./log4j-executor.properties**.

Solution 2: Modify the startup script **$SPARK_HOME/bin/spark-class**. In the **spark-class** script, add the following information below **#!/usr/bin/env bash**.

```
# Judge mode: client and cluster; Default: client
argv=`echo $@ | tr [A-Z] [a-z]`
if [[ "$argv" =~ "--master" ]];then
    mode=`echo $argv | sed -e 's/.*--master //'`
    master=`echo $mode | awk '{print $1}'`
    case $master in
    "yarn")
        deploy=`echo $mode | awk '{print $3}'`
        if [[ "$mode" =~ "--deploy-mode" ]];then
                deploy=$deploy
        else
                deploy="client"
        fi
    ;;
  "yarn-client"|"local")
        deploy="client"
    ;;
    "yarn-cluster")
        deploy="cluster"
    ;;
    esac
else
    deploy="client"
fi
# modify the spark-defaults.conf
number=`sed -n -e '/spark.driver.extraJavaOptions/=' $SPARK_HOME/conf/spark-defaults.conf`
if [ "$deploy"x = "client"x ];then
    `sed -i "${number}s/-Dlog4j.configuration=.*properties /-Dlog4j.configuration=.\/log4j.properties /g" $SPARK_HOME/conf/spark-defaults.conf`
else
    `sed -i "${number}s/-Dlog4j.configuration=.*properties /-Dlog4j.configuration=.\/log4j-executor.properties /g" $SPARK_HOME/conf/spark-defaults.conf`
fi
```

The functions of these script lines are similar to those of solution 1. You can change the value of the **-Dlog4j.configuration=./log4j-executor.properties** configuration item (default: **./log4j-executor.properties**) of **spark.driver.extraJavaOptions** in the **$SPARK_HOME/conf/spark-defaults.conf** file based on the Yarn mode.

14.5.14 What Should I Do If the "had a not serializable result" Error Is Reported When a Spark Task Reads HBase Data?

Question

What should I do if the error "Task 0.0 in stage 0.0 (TID 0) had a not serializable result: org.apache.hadoop.hbase.io.ImmutableBytesWritable" is reported when a Spark task reads HBase data?

Answer

You can resolve this exception by using either of the following methods:

- Run the following lines of code before initializing SparkConf:
  ```
  System.setProperty("spark.serializer", "org.apache.spark.serializer.KryoSerializer");
  System.setProperty("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrator");
  ```
- Use the **set** method to set the SparkConf object. The code is as follows:
  ```
  val conf = new SparkConf().setAppName("HbaseTest");
  conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer");
  conf.set("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrator");
  ```

14.5.15 How Do I Connect to Hive and HDFS of an MRS Cluster when the Spark Program Is Running on a Local Host?

Question

How do I connect to Hive and HDFS of an MRS cluster when the Spark program is running on a local host?

Answer

Step 1 Apply for and bind an elastic public IP address for each master node.

Step 2 Configure the mapping between the cluster IP addresses and host names on the local Windows host. Log in to the cluster background, run the **cat /etc/hosts** command, and copy the mapping between IP addresses and host names in the **hosts** file to **C:\Windows\System32\drivers\etc\hosts**. Host names are subject to the query result.

```
192.168.0.90 node-master1BedB.089d8c43-12d5-410c-b980-c2728a305be3.com
192.168.0.129 node-ana-corezLaR.089d8c43-12d5-410c-b980-c2728a305be3.com
```

Step 3 Log in to the background of any master node in the MRS cluster as user **root** and run the **cat /etc/hosts** command to obtain the mapping between IP addresses and host names in the **hosts** file.

Step 4 In **C:\Windows\System32\drivers\etc\hosts** of the local Windows host, configure the mapping obtained in **Step 3** and change the IP addresses of all master nodes to the EIPs bound to the nodes.

Step 5 Save **/opt/client/Hive/Beeline/conf/core-site.xml**, **/opt/client/Hive/config/hiveclient.properties**, and **/opt/client/Hive/config/hive-site.xml** of the MRS cluster to the **conf** directory of the project.

Step 6 Log in to MRS Manager and choose **System** > **Manage User**.

Step 7 Select a user who has Hive privileges, choose **More** > **Download Authentication Credential** in the **Operation** column, save the file, and decompress the package to obtain the **user.keytab** and **krb5.conf** files.

Step 8 Change the IP address of the master node in the **krb5.conf** file to the EIP bound to the node. Save the **user.keytab** and **krb5.conf** files to the **conf** directory of the project.

Step 9 Modify the security group rules of the MRS cluster and change the IP address policy of the Windows where IDEA resides to allow all IP addresses.

----End

15 Storm Development Guide

15.1 Storm Application Development Overview

15.1.1 Introduction to Storm Application Development

Intended Audience

This document is provided for users who want to implement Storm secondary development. This document is intended for development personnel who are experienced in Java development.

Introduction

Storm is a distributed, reliable, and fault-tolerant data stream processing system. Storm delegates work tasks to components of different types, and each component processes a specific simple task. Storm processes big data streams in real time and unlimited data streams in reliable mode.

Storm applies to real-time analytic, online machine learning, continuous computation, and distributed Extract, Transform, and Load (ETL). It is scalable and fault-tolerant, and is easy to set up and operate.

Storm has the following features:

- **Wide application**
- **High scalability**
- **Free from data loss**
- **High fault tolerance**
- **Language independence**
- **Easy to construction and control**

15.1.2 Common Concepts of Storm Application Development

Topology

A computing stream chart, in which each node contains processing logic and the lines between nodes specify data flow between nodes.

Spout

A component that generates source data flows in a topology. A Spout reads data from an external data source and converts the data into source data inside a topology.

Bolt

A component that receives data from a topology and then processes data. A Bolt can perform operations, such as filtering, executing functions, combination, and writing data into a database.

Tuple

Basic unit for transferring messages once.

Stream

A set of (infinite) elements, each of which belongs to the same schema. Each element is related to the logic time, that is, a Stream has the Tuple and Time attributes. Any elements can be expressed in the format of Element<Tuple,Time>, in which Tuple includes the data structure and content, and Time is the logic time of data.

Keytab file

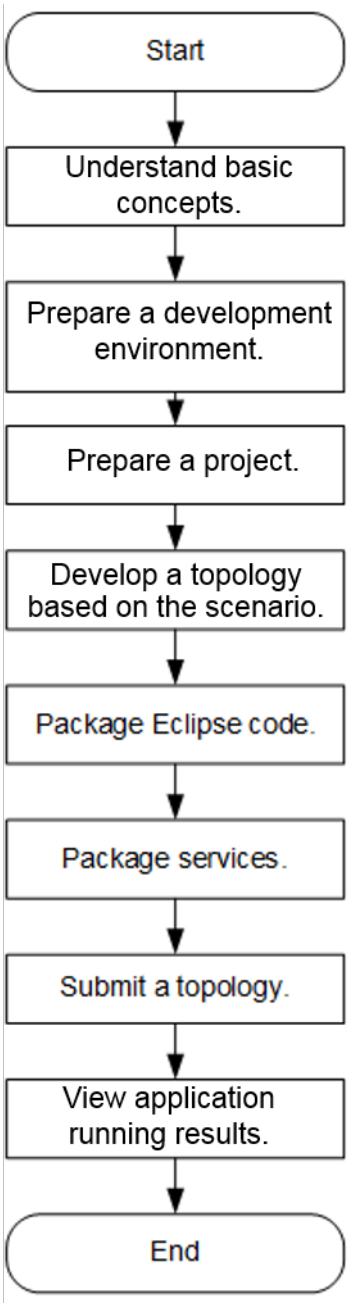
The keytab file is a key file that stores user information. Applications use the key file for API authentication on MRS.

15.1.3 Storm Application Development Process

This document describes Storm topology development based on the Java API.

Figure 15-1 shows the development process.

Figure 15-1 Topology development process



15.2 Preparing a Storm Application Development Environment

15.2.1 Storm Application Development Environment

This document describes the Eclipse sample project and common APIs of the Storm component of MRS based on open-source Storm. This helps developers quickly understand Storm development.

Prepare clients for developing and submitting applications. Generally, applications are developed in Windows and submitted in Linux.

Table 15-1 describes the environment required for secondary development.

Table 15-1 Development environment

Item	Description
OS	Windows OS. Windows 7 or later is recommended.
JDK installation	Basic configurations of the development environment. JDK 1.7 or 1.8 is required. **NOTE** For security purpose, the server supports only TLS 1.1 and TLS 1.2 encryption protocols. IBM JDK supports only TLS 1.0 by default. If you use IBM JDK, set **com.ibm.jsse2.overrideDefaultTLS** to **true**. After the parameter setting, TLS1.0/1.1/1.2 can be supported at the same time. For details, visit **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**.
Eclipse installation and configuration	Tool used for developing Storm applications.
Network	The client must be interconnected with the Storm server on the network.

15.2.2 Preparing the Eclipse and JDK

Scenario

The development environment can be set up on Windows.

Procedure

Step 1 Install the Eclipse. Install Eclipse. The Eclipse version must be 3.0 or later.

Step 2 Install the JDK. Install JDK. The JDK version must be 1.7 or 1.8, and IBM JDK and Oracle JDK are supported.

NOTE

- If you use IBM JDK, ensure that the JDK configured in Eclipse is IBM JDK.
- If you use Oracle JDK, ensure that the JDK configured in Eclipse is Oracle JDK.
- Do not use the same workspace and the sample project in the same path for different Eclipse programs.

----End

15.2.3 Preparing a Linux Client Environment

Background

Install the Linux client to submit the topology.

Prerequisites

- The Storm component has been installed and is running correctly.
- Ensure that the difference between the client time and the cluster time is less than 5 minutes.

Procedure

Step 1 Download a Storm client program.

1. Log in to **MRS Manager**.
2. Choose **Services** > **Storm** > **Download Client** and select **All client files** to download the client program to a **Remote host** (target ECS).

Step 2 Log in to the target ECS downloaded from the client.

Step 3 In the Linux OS, run the following command to decompress the client package:

tar -xvf MRS_Storm_Client.tar

tar -xvf MRS_Storm_ClientConfig.tar

Step 4 Switch to **MRS_Services_ClientConfig**. Run the **install.sh** script to install the client and run the **./install.sh /opt/Storm_Client** command to install the client to an empty folder. In the **./install.sh /opt/Storm_Client** command, **/opt/Storm_Client** indicates the Storm installation directory, which must be an empty directory and must be an absolute path.

Step 5 Initialize the environment variables of the client.

Go to the installation directory **/opt/Storm_Client**, and run the following command to import environment variables:

source bigdata_env

Step 6 In the cluster with Kerberos authentication enabled, you need to apply for a human-machine user for security login.

1. Obtain a human-machine user from the administrator for service authentication, for example, **john**.

   NOTE

   The obtained user must belong to the storm group.

2. Run the kinit command to log in to the system as the human-machine user.

   kinit *Username*

   Example:

   kinit john

   Enter the password as prompted. If no error message is displayed, Kerberos authentication is complete.

Step 7 Run the following command:

storm list

If the information about the running tasks of the storm cluster is correctly displayed, the client is installed successfully.

----End

15.2.4 Configuring and Importing a Project

Background

The installation directory of the Storm client contains a Storm development sample project. Import the sample project to Eclipse and start to learn the sample project.

Prerequisites

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

Procedure

Step 1 In the root directory of the Storm sample project, run the **mvn install** command to perform compilation.

Step 2 In the root directory of the Storm sample project, run the **mvn eclipse:eclipse** command to create an Eclipse project.

Step 3 In the application development environment, import the sample project to the Eclipse development environment.

1. Choose **File** > **Import** > **General** > **Existing Projects into Workspace** > **Next** > **Browse**.

   The **Browse Folder** dialog box is displayed.

2. Select the sample project folder, and click **Finish**.

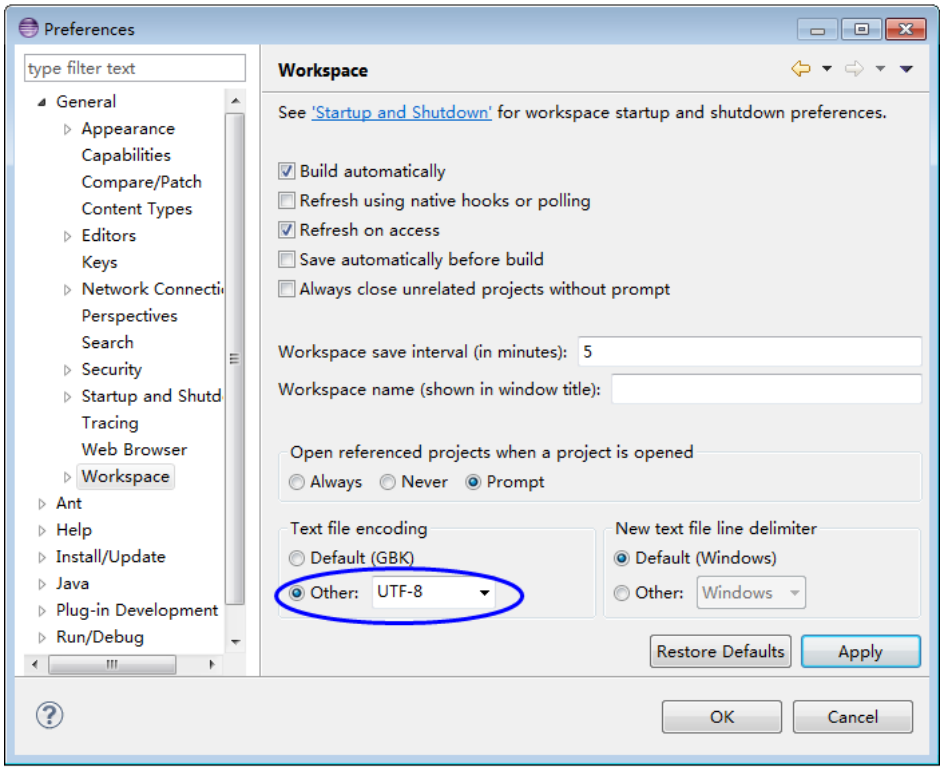
Step 4 Set an Eclipse text file encoding format to prevent garbled characters.

1. On the Eclipse menu bar, choose **Window** > **Preferences**.

   The **Preferences** window is displayed.

2. In the navigation tree, choose **General** > **Workspace**. In the **Text file encoding** area, select **Other** and set the value to **UTF-8**. Click **Apply** and then **OK**. **Figure 15-2** shows the settings.

Figure 15-2 Setting the Eclipse encoding format



Step 5 If you submit a task to a cluster in the Windows environment, you need to map the cluster IP address to the local **host** file. Take Windows 7 as an example, the path is **C:\Windows\System32\drivers\etc\hosts**.

Assume that the cluster has three nodes: 10.1.131.131, 10.1.131.132, and 10.1.131.133.

Check whether the following information is configured in the **hosts** file.

```
10.1.131.131 10-1-131-131
10.1.131.132 10-1-131-132
10.1.131.133 10-1-131-133
```

----End

15.3 Developing a Storm Application

15.3.1 Storm Development Plan

A typical scenario can help you quickly learn and master the Storm topology structure and Spout/Bolt development process.

Scenario Description

Service processing of a dynamic word counting system is described as follows. The data source is a logical unit that produces random text continuously.

- The data source continuously sends random text, such as "apple orange apple", to the text splitting logic.

- The word splitting logic splits each text entry sent by the data source by space, such as "apple", "orange", "apple", and then sends each word to the word counting logic.
- The word counting logic increases the number of times that a specific word occurs by one when receiving the word, and prints the real-time results. For example:
  - apple: 1
  - orange: 1
  - apple: 2

Function Description

Table 15-2 describes the procedure for a user to develop an application to calculate the number of times that each word appears in random text.

Table 15-2 Functions to be developed

No.	Step	Sample Code
1	Create a Spout to generate random text.	For details, see **Creating a Storm Spout**.
2	Create a Bolt to split the random text into words.	For details, see **Creating a Storm Bolt**.
3	Create a Blot to calculate the number of times that each word appears.	For details, see **Creating a Storm Bolt**.
4	Create a topology.	For details, see **Creating a Storm Topology**.

For details about certain code, see **Creating a Storm Spout**, **Creating a Storm Bolt**, and **Creating a Storm Topology**. For details about complete code, see the Storm-examples project.

15.3.2 Creating a Storm Spout

Function Description

A Spout is a message source of Storm and message producer of the topology. Generally, a message source reads data from an external source and sends messages (Tuple) to the topology.

One message source can send multiple message streams, and therefore, OutputFieldsDeclarer.declarerStream can be used to define multiple streams, and then SpoutOutputCollector emits specific streams.

Sample Code

The following code snippet belongs to the **nextTuple** method in the **RandomSentenceSpout** class of the **com.huawei.storm.example.common** package, and these code snippets are used to split strings into words.

```
/**
  * {@inheritDoc}
  */
 @Override
 public void nextTuple()
 {
     Utils.sleep(100);
     String[] sentences =
         new String[] {"the cow jumped over the moon",
                       "an apple a day keeps the doctor away",
                       "four score and seven years ago",
                       "snow white and the seven dwarfs",
                       "i am at two with nature"};
     String sentence = sentences[random.nextInt(sentences.length)];
     collector.emit(new Values(sentence));
 }
```

15.3.3 Creating a Storm Bolt

Function Description

All message processing logic is encapsulated in Bolts. Bolts provide multiple functions, such as filtering and aggregation.

If other topology operators, except for Bolts, OutputFieldsDeclarer.declareStream can be used to define streams, and OutputCollector.emit can be used to select streams to be emitted.

Sample Code

The following code snippets are in the **com.huawei.storm.example.common.SplitSentenceBolt** class, and these code snippets are used to split a statement into words and send the words.

```
/**
  * {@inheritDoc}
  */
 @Override
 public void execute(Tuple input, BasicOutputCollector collector)
 {
     String sentence = input.getString(0);
     String[] words = sentence.split(" ");
     for (String word : words)
     {
         word = word.trim();
         if (!word.isEmpty())
         {
             word = word.toLowerCase();
             collector.emit(new Values(word));
         }
     }
 }
```

The following code snippets are in the **com.huawei.storm.example.wordcount.WordCountBolt** class, and these code snippets are used to calculate the number of received words.


```
@Override
 public void execute(Tuple tuple, BasicOutputCollector collector)
 {
     String word = tuple.getString(0);
     Integer count = counts.get(word);
     if (count == null)
     {
         count = 0;
     }
     count++;
     counts.put(word, count);
     System.out.println("word: " + word + ", count: " + count);
 }
```

15.3.4 Creating a Storm Topology

Function Description

A topology is a directed acyclic graph (DAG) consisting of Spouts and Bolts.

Applications are submitted in storm jar mode. Therefore, a function for creating a topology must be invoked in the main function, and the class to which the main function belongs must be specified in storm jar parameters.

Sample Code

The following code snippets are in the **com.huawei.storm.example.wordcount.WordCountTopology** class, and these code snippets are used to create and submit applications.

```
public static void main(String[] args)
        throws Exception
    {
        TopologyBuilder builder = buildTopology();

        /*
         * Tasks can be submitted in the following three modes:
         * 1. Command line submitting. In this mode, a user must copy an application JAR package to a client and run related commands on the client.
         * 2. Remote submitting. In this mode, a user must package application JAR files and execute the main method in Eclipse.
         * 3. Local submitting. In this mode, a user must run an application for test on a local computer.
         * The command line submitting and remote submitting modes support both security and normal modes.
         * The local submitting mode supports the normal mode only.
         *
         * A user can select only one mode for submitting a task. By default, the command line submitting mode is used. To use another mode, delete code comments.
         */

        submitTopology(builder, SubmitType.CMD);

    }

    private static void submitTopology(TopologyBuilder builder, SubmitType type) throws Exception
    {
        switch (type)
        {
            case CMD:
            {
                cmdSubmit(builder, null);
                break;
            }
            case REMOTE:
            {
```



```
                remoteSubmit(builder);
                break;
            }
            case LOCAL:
            {
                localSubmit(builder);
                break;
            }
        }
    }

    /**
     * Command line submitting mode
     * The procedures are as follows:
     * 1. Package a JAR file and then submit the task in the client CLI.
     * 2. In remote submitting mode, package the JAR file of the application and other external dependency JAR files of users' applications into a big JAR file. External dependency JAR files are not provided by the sample project.
     * 3. Run the storm -jar command on the Storm client to submit the task.
     *
     * In a security environment, before submitting the task in the client CLI, run the kinit command to perform login in security mode.
     *
     * Run the following command:
     *./storm jar ../example/example.jar com.huawei.streaming.storm.example.WordCountTopology
     */
    private static void cmdSubmit(TopologyBuilder builder, Config conf)
        throws AlreadyAliveException, InvalidTopologyException, NotALeaderException, AuthorizationException
    {
        if (conf == null)
        {
            conf = new Config();
        }
        conf.setNumWorkers(1);

        StormSubmitter.submitTopologyWithProgressBar(TOPOLOGY_NAME, conf, builder.createTopology());
    }

     private static void localSubmit(TopologyBuilder builder)
         throws InterruptedException
     {
         Config conf = new Config();
         conf.setDebug(true);
         conf.setMaxTaskParallelism(3);
         LocalCluster cluster = new LocalCluster();
         cluster.submitTopology(TOPOLOGY_NAME, conf, builder.createTopology());
         Thread.sleep(10000);
         cluster.shutdown();
     }


    private static void remoteSubmit(TopologyBuilder builder)
        throws AlreadyAliveException, InvalidTopologyException, NotALeaderException, AuthorizationException,
        IOException
    {
        Config config = createConf();

        String userJarFilePath = "User JAR file address";
        System.setProperty(STORM_SUBMIT_JAR_PROPERTY, userJarFilePath);

        //Preparations to be made in security mode
        if (isSecurityModel())
        {
            securityPrepare(config);
        }
        config.setNumWorkers(1);
        StormSubmitter.submitTopologyWithProgressBar(TOPOLOGY_NAME, config,
```



```
builder.createTopology());
    }

      private static TopologyBuilder buildTopology()
    {
        TopologyBuilder builder = new TopologyBuilder();
        builder.setSpout("spout", new RandomSentenceSpout(), 5);
        builder.setBolt("split", new SplitSentenceBolt(), 8).shuffleGrouping("spout");
        builder.setBolt("count", new WordCountBolt(), 12).fieldsGrouping("split", new Fields("word"));
        return builder;
    }
```

15.4 Commissioning a Storm Application

15.4.1 Generating the JAR Package of the Storm Application

Scenario

Run related commands to generate the JAR file of the sample code.

Procedure

In the root directory of Storm sample code, run the **mvn package** command. After the command is executed successfully, the **storm-examples-1.0.jar** file is generated in the target directory.

15.4.2 Commissioning a Storm Application on Linux

Scenario

NOTE

Storm applications can run only on Linux, but not on Windows.

You can use storm commands to submit topologies in a Linux environment.

Prerequisites

- You have installed a Storm 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 locates. The host names and IP addresses must be mapped one by one.
- **Generating the JAR Package of the Storm Application** has been performed and **storm-examples-1.0.jar** has been generated and stored in **/opt/jartarget/**.

Procedure

Step 1 Perform security authentication in security mode. For details, see **Preparing a Linux Client Environment**.

Step 2 Submit a topology. (Wordcount is used as an example. For details about other topologies, see the related development guidelines.) Go to the **storm-0.10.0/bin**

directory on the Storm client, and run the **storm jar /opt/jartarget/**storm-examples-1.0.jar **com.huawei.storm.example.wordcount.WordCountTopology** command.

Step 3 Run the **storm list** command to view the submitted applications. If the word-count application can be viewed, the task is submitted successfully.

NOTE

If a service is set to the local mode and is submitted by using commands, ensure that the submitting environment is a normal one. Currently, services in local mode cannot be submitted by using commands in a security environment.

----End

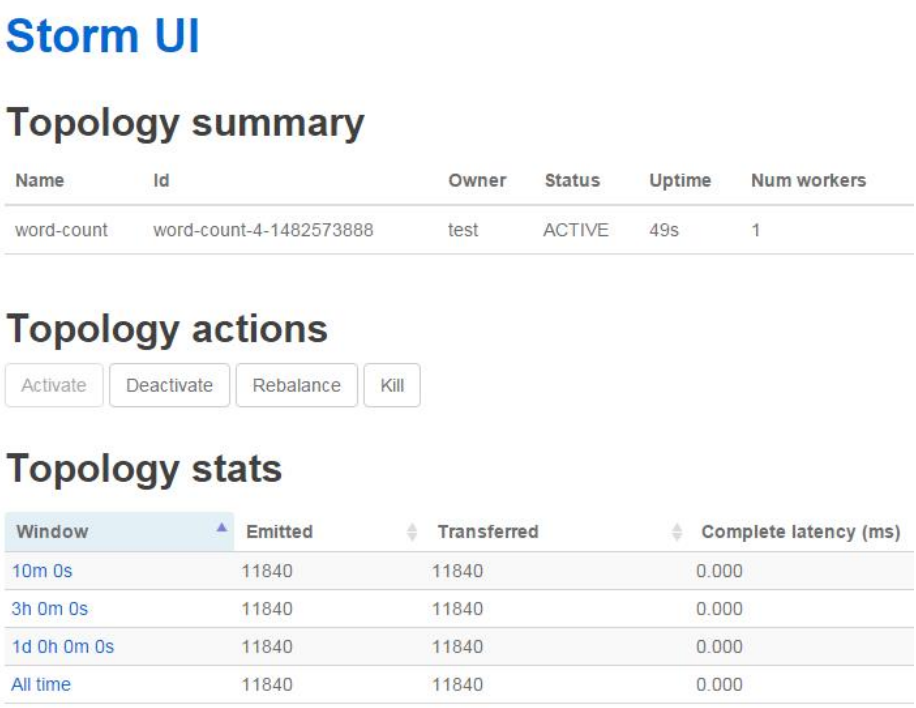
15.4.3 Viewing the Storm Application Commissioning Result

Procedure

Step 1 Access the Storm web page by referring to section "Accessing the UI of the Open Source Component" in the *MapReduce Service User Guide*.

Step 2 On the Storm UI, click the word-count application to view the application running status, as shown in **Figure 15-3**.

Figure 15-3 Storm application execution page



In **Topology stats**, the total volume of data transferred between operators in different time periods is displayed.

In **Spouts**, the total number of messages sent by the spout operator from the moment the operator is started till now is displayed. In **Bolts**, the total number of

messages sent by the Count operator and the split operator is displayed. See **Figure 15-4**.

Figure 15-4 Total volume of data sent by the Storm application operators

Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Last error
spout	5	5	20940	20940	0.000	0	0	

Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)
count	12	12	0	0	0.006	0.105	133920	0.086
split	8	8	133880	133880	0.005	0.670	20940	0.648

----End

15.5 FAQs About Storm Application Development

15.5.1 Storm APIs

Versions of APIs adopted by Storm are consistent with those in the open-source community. For details, see the following website:

http://storm.apache.org/documentation/Home.html

Versions of APIs adopted by Storm-HDFS are consistent with those in the open-source community. For details, see the following website:

https://github.com/apache/storm/tree/v0.10.0/external/storm-hdfs

Versions of APIs adopted by Storm-HBase are consistent with those in the open-source community. For details, see the following website:

https://github.com/apache/storm/tree/v0.10.0/external/storm-hbase

Versions of APIs adopted by Storm-Kafka are consistent with those in the open-source community. For details, see the following website:

https://github.com/apache/storm/tree/v0.10.0/external/storm-kafka

Versions of APIs adopted by Storm-JDBC are consistent with those in the open-source community. For details, see the following website:

https://github.com/apache/storm/tree/v0.10.0/external/storm-jdbc

15.5.2 Storm-Kafka Development Guideline

Scenario

This section describes how to use the Storm-Kafka toolkit to implement the interaction between Storm and Kafka. KafkaSpout and KafkaBolt are included. KafkaSpout enables Storm to read data from Kafka. KafkaBolt enables Storm to write data into Kafka.

The sample code uses new Kafka APIs and corresponds to com.huawei.storm.example.kafka.NewKafkaTopology.java in the Eclipse project.

This section applies only to the access between the Storm component and the Kafka component of MRS. Determine the versions of the JAR files described in this section based on the actual situation.

Procedure for Developing an Application

Step 1 Verify that the Storm and Kafka components of MRS have been installed and are running properly.

Step 2 Ensure that a Storm sample code project has been set up. Import storm-examples to the Eclipse development environment. For details, see **Configuring and Importing a Project**.

Step 3 Use WinScp to import the Storm client installation package to the Linux environment and install the client. For details, see **Preparing a Linux Client Environment**.

Step 4 If the security service is enabled on the cluster, you need to obtain a human-machine user from the administrator for authentication and obtain the keytab file of the user. Copy the obtained file to the **src/main/resources** directory of the sample project.

NOTE

- The obtained user must belong to both the storm and Kafka groups.

Step 5 Download and install the Kafka client. For details, see the *Kafka Development Guide*.

----End

Sample Code

Create a topology.

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

// Set the topology.
Config conf = new Config();

// Configure the security plug-in.
setSecurityPlugin(conf);

if (args.length >= 2) {
// If the default keytab file name has been changed, configure the new keytab file name.
conf.put(Config.TOPOLOGY_KEYTAB_FILE, args[1]);
}
```



```
// Define KafkaSpout.
KafkaSpout kafkaSpout = new KafkaSpout<String, String>(
getKafkaSpoutConfig(getKafkaSpoutStreams()));

// CountBolt
CountBolt countBolt = new CountBolt();
//SplitBolt
SplitSentenceBolt splitBolt = new SplitSentenceBolt();

// KafkaBolt configuration information
conf.put(KafkaBolt.KAFKA_BROKER_PROPERTIES, getKafkaProducerProps());
KafkaBolt<String, String> kafkaBolt = new KafkaBolt<String, String>();
kafkaBolt.withTopicSelector(new DefaultTopicSelector(OUTPUT_TOPIC))
.withTupleToKafkaMapper(
new FieldNameBasedTupleToKafkaMapper("word", "count"));

// Define the topology.
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("kafka-spout", kafkaSpout, 10);
builder.setBolt("split-bolt", splitBolt,10).shuffleGrouping("kafka-spout", STREAMS[0]);
builder.setBolt("count-bolt", countBolt, 10).fieldsGrouping(
"split-bolt", new Fields("word"));
builder.setBolt("kafka-bolt", kafkaBolt, 10).shuffleGrouping("count-bolt");

// Run the related command to submit the topology.
StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
}
```

Running the Application and Viewing Results

Step 1 Obtain the related configuration file using the following method:

- Security mode: Obtain the keytab file by referring to **Step 4**.
- Common mode: None

Step 2 In the root directory of Storm sample code, run the **mvn package** command. After the command is executed successfully, the **storm-examples-1.0.jar** file is generated in the target directory.

Step 3 Run the following commands to use the Kafka client to create the topic used by the topology:

./kafka-topics.sh --create --topic input --partitions 2 --replication-factor 2 --zookeeper {ip:port}/kafka

./kafka-topics.sh --create --topic output --partitions 2 --replication-factor 2 --zookeeper {ip:port}/kafka

NOTE

- The variable appended to **--zookeeper** specifies the ZooKeeper address. You must set the ZooKeeper address to the ZooKeeper address configured during cluster installation.
- In security mode, the Kafka administrator needs to create a topic.

Step 4 Submit the topology on a Linux OS. The submission command example is as follows (the topology name is **kafka-test**):

storm jar /opt/jartarget/storm-examples-1.0.jar **com.huawei.storm.example.kafka.NewKafkaTopology kafka-test**

NOTE

- In security mode, ensure that Kerberos security login has been performed before the **storm-examples-1.0.jar** file is submitted. In keytab mode, the login user and the user to whom the uploaded keytab file belongs must be the same user.
- In security mode, the Kafka user must have the permission to access the corresponding topic. Therefore, you need to assign permission to the user before submitting the topology.

Step 5 After the topology is successfully submitted, send data to Kafka and check whether related information is generated.

Go to the directory where the Kafka client locates in the Linux system, start the consumer in the **Kafka/kafka/bin** directory, and check whether data is generated. The command is detailed as follows:

./kafka-console-consumer.sh --bootstrap-server {ip:port} --topic output --new-consumer --consumer.config ../../../Kafka/kafka/config/consumer.properties

Go to the directory whether the Kafka client locates in the Linux system, start the producer in the **Kafka/kafka/bin** directory, and write data into Kafka. The command is detailed as follows:

./kafka-console-producer.sh --broker-list {ip:port} --topic input --producer.config ../../../Kafka/kafka/config/producer.properties

Write test data into input, and check whether related data is generated in output. If yes, the Storm-Kafka topology is executed successfully.

----End

15.5.3 Storm-JDBC Development Guideline

Scenario

This section describes how to use the open-source Storm-JDBC toolkit to implement the interaction between Storm and JDBC. JdbcInsertBolt and JdbcLookupBolt are included. JdbcLookupBolt is used to query data from the database. JdbcInsertBolt is used to store data to the database. In addition, JdbcLookupBolt and JdbcInsertBolt can be used to process data using the data processing logic.

This section applies only to the access between the Storm component and the JDBC component of MRS. Determine the versions of the JAR files described in this section based on the actual situation.

Procedure for Developing an Application

Step 1 Verify that the Storm component has been installed and is running correctly.

Step 2 Download the Storm client and import the Storm sample project to the Eclipse development environment. For details, see **Configuring and Importing a Project**.

Step 3 Use WinScp to import the Storm client to the Linux environment and install the client. For details, see **Preparing a Linux Client Environment**.

----End

Configuring the Database – Configuring the Derby Database

Step 1 Download a database. Select the best suitable database based on the actual scenario.

In this section, the Derby database is used as an example. The Derby database is a Java-based small-sized open-source database that is easy to use and suitable to most applications.

Step 2 Obtain the Derby database. Download the Derby database package of the latest version (10.14.1.0 is used in this example) from the official website, use WinSCP to upload the database package to the Linux client, and decompress the package.

Step 3 In the Derby installation directory, go to the **bin** directory, and run the following commands:

export DERBY_INSTALL=/opt/db-derby-10.14.1.0**-bin**

export CLASSPATH=$DERBY_INSTALL/lib/derbytools.jar:$DERBY_INSTALL\lib\derbynet.jar:.

export DERBY_HOME=/opt/db-derby-10.14.1.0**-bin**

. setNetworkServerCP

./startNetworkServer -h *Host name*

Step 4 Run the ***./ij*** command and enter ***connect 'jdbc:derby://***Host name***:1527/example;create=true';*** to create the connection.

NOTE

- Before running the **./ij** command, ensure that **java_home** has been configured. You can run the **which java** command to check whether **java_home** has been configured.

Step 5 After the database is connected, run SQL statements to create table ORIGINAL and table GOAL and insert a group of data into table ORIGINAL. The statement examples are as follows (the table names can be customized):

CREATE TABLE GOAL(WORD VARCHAR(12),COUNT INT);

CREATE TABLE ORIGINAL(WORD VARCHAR(12),COUNT INT);

INSERT INTO ORIGINAL VALUES('orange',1),('pineapple',1),('banana',1),('watermelon',1);

----End

Sample Code

SimpleJDBCTopology sample code: (Change the IP addresses and ports to the actual ones.)

```
public class SimpleJDBCTopology
 {
   private static final String WORD_SPOUT = "WORD_SPOUT";
   private static final String COUNT_BOLT = "COUNT_BOLT";
   private static final String JDBC_INSERT_BOLT = "JDBC_INSERT_BOLT";
   private static final String JDBC_LOOKUP_BOLT = "JDBC_LOOKUP_BOLT";
   @SuppressWarnings ("unchecked")
   public static void main(String[] args) throws Exception{
     //connectionProvider configuration
```



```
    Map hikariConfigMap = Maps.newHashMap();
    hikariConfigMap.put("dataSourceClassName", "org.apache.derby.jdbc.ClientDataSource");
    hikariConfigMap.put("dataSource.serverName", "192.168.0.1");//Set this parameter to the actual IP
address.
    hikariConfigMap.put("dataSource.portNumber", "1527");//Set this parameter to the actual port number.

    hikariConfigMap.put("dataSource.databaseName", "example");
    hikariConfigMap.put("connectionTestQuery", "select COUNT from  GOAL"); //The table name must be
consistent with that used during table creation.
    Config conf = new Config();

    ConnectionProvider connectionProvider = new HikariCPConnectionProvider(hikariConfigMap);
    //JdbcLookupBolt instantiation
    Fields outputFields = new Fields("WORD", "COUNT");
    List<Column> queryParamColumns = Lists.newArrayList(new Column("WORD", Types.VARCHAR));
    SimpleJdbcLookupMapper jdbcLookupMapper = new SimpleJdbcLookupMapper(outputFields,
queryParamColumns);
    String selectSql = "select COUNT from ORIGINAL where WORD = ?";
    JdbcLookupBolt wordLookupBolt = new JdbcLookupBolt(connectionProvider, selectSql,
jdbcLookupMapper);
    //JdbcInsertBolt instantiation
    String tableName = "GOAL";
    JdbcMapper simpleJdbcMapper = new SimpleJdbcMapper(tableName, connectionProvider);
    JdbcInsertBolt userPersistanceBolt = new JdbcInsertBolt(connectionProvider,
simpleJdbcMapper).withTableName("GOAL").withQueryTimeoutSecs(30);
    WordSpout wordSpout = new WordSpout();TopologyBuilder builder = new TopologyBuilder();
    builder.setSpout(WORD_SPOUT, wordSpout);
    builder.setBolt(JDBC_LOOKUP_BOLT, wordLookupBolt, 1).fieldsGrouping(WORD_SPOUT,new
Fields("WORD"));
    builder.setBolt(JDBC_INSERT_BOLT, userPersistanceBolt,1).fieldsGrouping(JDBC_LOOKUP_BOLT,new
Fields("WORD"));StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
  }
 }
```

Running the Application

Step 1 In the root directory of Storm sample code, run the **mvn package** command. After the command is executed successfully, the **storm-examples-1.0.jar** file is generated in the target directory.

Step 2 Run the related command to submit the topology. The submission command example is as follows (the topology name is **jdbc-test**):

storm jar /opt/jartarget/storm-examples-1.0.jar **com.huawei.storm.example.jdbc.SimpleJDBCTopology jdbc-test**

----End

Viewing Results

After the topology is submitted, go to the database and check whether data is inserted into the related tables.

Run the **select * from goal;** statement to query data in table **GOAL**. If data is inserted into table **GOAL**, the topology is executed successfully.

15.5.4 Storm-HDFS Development Guideline

Scenario

This topic applies only to the interaction between Storm and HDFS. Determine the versions of the JAR files described in this section based on the actual situation.

Login in security mode is classified into ticket login and keytab file login, and the procedures for these two login modes are the same. The ticket login mode is an open-source capability and requires manual ticket uploading, which may cause reliability and usability problems. Therefore, the keytab file login mode is recommended.

Procedure for Developing an Application

Step 1 Verify that the Storm and HDFS components have been installed and are running correctly.

Step 2 Import **storm-examples** to the Eclipse development environment. For details, see **Configuring and Importing a Project**.

Step 3 If the cluster is enabled with security services, perform the following operations based on the login mode.

- Keytab mode: You need to obtain a human-machine user from the administrator for authentication and obtain the keytab file of the user.
- Ticket mode: Obtain a human-machine user from the administrator for subsequent secure login, enable the renewable and forwardable functions of the Kerberos service, set the ticket update period, and restart Kerberos and related components.

NOTE

- The obtained user must belong to the storm group.
- The parameters for enabling the renewable and forwardable functions and setting the ticket update interval are on the System tab of the Kerberos service configuration page. The ticket update interval can be set to kdc_renew_lifetime or kdc_max_renewable_life based on the actual situation.

Step 4 Download and install the HDFS client. For details, see section "Preparing a Linux Client Operating Environment."

Step 5 Obtain the HDFS-related configuration files by performing the following operations:

Go to the **/opt/client/HDFS/hadoop/etc/hadoop** directory on the installed HDFS client, and obtain the configuration files **core-site.xml** and **hdfs-site.xml**.

In keytab mode, obtain the keytab file by following **Step 3**. In ticket mode, no extra configuration file is required.

Copy the obtained files to the **src/main/resources** directory of the sample project.

NOTE

The obtained keytab file is named as **user.keytab** by default. A user can directly change the file name as required. However, the user must upload the changed file name as a parameter when submitting a task.

----End

Eclipse Sample Code

Create a topology.

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



```
    TopologyBuilder builder = new TopologyBuilder();

    // Separator. Use | to replace the default comma (,) to separate fields in tuple.
    // Mandatory HdfsBolt parameter
    RecordFormat format = new DelimitedRecordFormat()
            .withFieldDelimiter("|");

    // Synchronization policy. Synchronize the file system for every 1000 tuples.
    // Mandatory HdfsBolt parameter
    SyncPolicy syncPolicy = new CountSyncPolicy(1000);

    // File size cyclic policy. If the size of a file reaches 5 MB, the file is written from the beginning.
    // Mandatory HdfsBolt parameter
    FileRotationPolicy rotationPolicy = new FileSizeRotationPolicy(5.0f, Units.MB);

    // Objective file written to HDFS
    // Mandatory HdfsBolt parameter
    FileNameFormat fileNameFormat = new DefaultFileNameFormat()
            .withPath("/user/foo/");


    //Create HdfsBolt.
    HdfsBolt bolt = new HdfsBolt()
            .withFileNameFormat(fileNameFormat)
            .withRecordFormat(format)
            .withRotationPolicy(rotationPolicy)
            .withSyncPolicy(syncPolicy);

    //Spout generates a random statement.
    builder.setSpout("spout", new RandomSentenceSpout(), 1);
    builder.setBolt("split", new SplitSentence(), 1).shuffleGrouping("spout");
    builder.setBolt("count", bolt, 1).fieldsGrouping("split", new Fields("word"));

    //Add the plugin required for Kerberos authentication to the list. This operation is mandatory in security
mode.
    setSecurityConf(conf,AuthenticationType.KEYTAB);

    Config conf = new Config();
    //Write the plugin list configured on the client to a specific config item. This operation is mandatory in
security mode.
    conf.put(Config.TOPOLOGY_AUTO_CREDENTIALS, auto_tgts);

    if(args.length >= 2)
    {
      // If the default keytab file name has been changed, configure the new keytab file name.
       conf.put(Config.STORM_CLIENT_KEYTAB_FILE, args[1]);
    }

    //Run the related command to submit the topology.
    StormSubmitter.submitTopology(args[0], conf, builder.createTopology());

  }
```

Running the Application and Viewing Results

Step 1 In the root directory of Storm sample code, run the **mvn package** command. After the command is executed successfully, the **storm-examples-1.0.jar** file is generated in the target directory.

Step 2 Run the related command to submit the topology.

In keytab mode, if the user changes the keytab file name, for example, **huawei.keytab**, the changed keytab file name must be added to the command as a parameter for description. The submission command example is as follows (the topology name is **hdfs-test**):

storm jar /opt/jartarget/storm-examples-1.0.jar **com.huawei.storm.example.hdfs.SimpleHDFSTopology hdfs-test huawei.keytab**

NOTE

In security mode, ensure that Kerberos security login has been performed before the **source.jar** file is submitted. In keytab mode, the login user and the user to whom the uploaded keytab file belongs must be the same user.

Step 3 After the topology is submitted successfully, log in to the HDFS cluster to check whether files are generated in the **/user/foo** directory.

Step 4 To perform login in ticket mode, perform the following operations to regularly upload a ticket. The interval for uploading the ticket depends on the deadline for updating the ticket.

1. Add the following content to a new line at the end of the **Storm/storm-0.10.0/conf/storm.yaml** file in the Storm client installation directory.

   topology.auto-credentials:

   - backtype.storm.security.auth.kerberos.AutoTGT

2. Run the **./storm upload-credentials hdfs-test** command.

----End

15.5.5 Storm-OBS Development Guideline

Scenario

This topic applies only to the interaction between Storm and OBS. Determine the versions of the JAR files described in this section based on the actual situation.

Procedure for Developing an Application

Step 1 Verify that the Storm component has been installed and is running correctly.

Step 2 Import **storm-examples** to the Eclipse development environment. For details, see **Configuring and Importing a Project**.

Step 3 Download and install the HDFS client. For details, see **Preparing an HDFS Application Running Environment**.

Step 4 Obtain the related configuration files by performing the following operations:

Go to the **/opt/client/HDFS/hadoop/etc/hadoop** directory on the installed HDFS client, and obtain the configuration files **core-site.xml** and **hdfs-site.xml**. Copy the obtained files to the **src/main/resources** directory of the sample project. Add the following configuration items to **core-site.xml**:

```
<property>
<name>fs.obs.connection.ssl.enabled</name>
<value>true</value>
</property>
<property>
<name>fs.obs.endpoint</name>
<value></value>
</property>
<property>
```



```
<name>fs.obs.access.key</name>
<value></value>
</property>
<property>
<name>fs.obs.secret.key</name>
<value></value>
</property>
```

For details about how to obtain the AK and SK, see the OBS documentation.

----End

Eclipse Sample Code

Create a topology.

```
private static final String DEFAULT_FS_URL = "obs://mybucket";

public static void main(String[] args) throws Exception
   {
     TopologyBuilder builder = new TopologyBuilder();

     // Separator. Use | to replace the default comma (,) to separate fields in tuple.
     // Mandatory HdfsBolt parameter
     RecordFormat format = new DelimitedRecordFormat()
             .withFieldDelimiter("|");

     // Synchronization policy. Synchronize the file system for every 1000 tuples.
     // Mandatory HdfsBolt parameter
     SyncPolicy syncPolicy = new CountSyncPolicy(1000);

     // File size cyclic policy. If the size of a file reaches 5 MB, the file is written from the beginning.
     // Mandatory HdfsBolt parameter
     FileRotationPolicy rotationPolicy = new FileSizeRotationPolicy(5.0f, Units.KB);

     // Objective file written to HDFS
     // Mandatory HdfsBolt parameter
     FileNameFormat fileNameFormat = new DefaultFileNameFormat()
             .withPath("/user/foo/");


     //Create HdfsBolt.
     HdfsBolt bolt = new HdfsBolt()
             .withFsUrl(DEFAULT_FS_URL)
             .withFileNameFormat(fileNameFormat)
             .withRecordFormat(format)
             .withRotationPolicy(rotationPolicy)
             .withSyncPolicy(syncPolicy);

     //Spout generates a random statement.
     builder.setSpout("spout", new RandomSentenceSpout(), 1);
     builder.setBolt("split", new SplitSentence(), 1).shuffleGrouping("spout");
     builder.setBolt("count", bolt, 1).fieldsGrouping("split", new Fields("word"));

     Config conf = new Config();


     //Run the related command to submit the topology.
     StormSubmitter.submitTopology(args[0], conf, builder.createTopology());

   }
```


Running the Application and Viewing Results

Step 1 In the root directory of Storm sample code, run the **mvn package** command. After the command is executed successfully, the **storm-examples-1.0.jar** file is generated in the target directory.

Step 2 Run the related command to submit the topology.

The submission command example is as follows (the topology name is **obs-test**):

storm jar /opt/jartarget/storm-examples-1.0.jar **com.huawei.storm.example.obs.SimpleOBSTopology obs://my-bucket obs-test**

Step 3 After the topology is submitted successfully, log in to OBS Browser to view the topology.

----End

15.5.6 Storm-HBase Development Guideline

Scenario

This topic applies only to the interaction between Storm and HBase. Determine the versions of the JAR files described in this section based on the actual situation.

Login in security mode is classified into ticket login and keytab file login, and the procedures for these two login modes are the same. The ticket login mode is an open-source capability and requires manual ticket uploading, which may cause reliability and usability problems. Therefore, the keytab file login mode is recommended.

Procedure for Developing an Application

Step 1 Verify that the Storm and HBase components have been installed and are running correctly.

Step 2 Import **storm-examples** to the Eclipse development environment. For details, see **Configuring and Importing a Project**.

Step 3 If security services are enabled in the cluster, perform the related configuration based on the login mode.

- Keytab mode: You need to obtain a human-machine user from the administrator for authentication and obtain the keytab file of the user.
- Ticket mode: Obtain a human-machine user from the administrator for subsequent secure login, enable the renewable and forwardable functions of the Kerberos service, set the ticket update period, and restart Kerberos and related components.

NOTE

- The obtained user must belong to the storm group.
- The parameters for enabling the renewable and forwardable functions and setting the ticket update interval are on the System tab of the Kerberos service configuration page. The ticket update interval can be set to kdc_renew_lifetime or kdc_max_renewable_life based on the actual situation.

Step 4 Download and install the HBase client program.

Step 5 Obtain the related configuration files by performing the following operations:

Go to the **/opt/client/HBase/hbase/conf** directory on the installed HBase client, and obtain configuration files **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml**. Copy the obtained files to the **src/main/resources** directory of the sample project.

In keytab mode, obtain the keytab file by following **Step 3**. In ticket mode, no extra configuration file is required.

NOTE

The obtained keytab file is named as **user.keytab** by default. A user can directly change the file name as required. However, the user must upload the changed file name as a parameter when submitting a task.

----End

Eclipse Sample Code

Create a topology.

```
public static void main(String[] args) throws Exception
    {
        Config conf = new Config();

    //Add the plugin required for Kerberos authentication to the list. This operation is mandatory in security mode.
        setSecurityConf(conf,AuthenticationType.KEYTAB);

        if(args.length >= 2)
        {
        //The default keytab file name is changed by the user. Specify the new keytab file name as a parameter.
        conf.put(Config.STORM_CLIENT_KEYTAB_FILE, args[1]);
        }
        //HBase client configuration. Only the hbase.rootdir configuration item is provided, which is optional.
        Map<String, Object> hbConf = new HashMap<String, Object>();
        if(args.length >= 3)
        {
            hbConf.put("hbase.rootdir", args[2]);
        }
        //Mandatory parameter. If it is not set, it is left blank.
        conf.put("hbase.conf", hbConf);

        //spout is a random word.
        WordSpout spout = new WordSpout();
        WordCounter bolt = new WordCounter();

        //HbaseMapper, which is used for parsing tuple content.
        SimpleHBaseMapper mapper = new SimpleHBaseMapper()
                .withRowKeyField("word")
                .withColumnFields(new Fields("word"))
                .withCounterFields(new Fields("count"))
                .withColumnFamily("cf");

        //HBaseBolt. The first parameter is a table name.
        //withConfigKey("hbase.conf") Transfer the HBase client configuration to HBaseBolt.
        HBaseBolt hbase = new HBaseBolt("WordCount", mapper).withConfigKey("hbase.conf");


        // wordSpout ==> countBolt ==> HBaseBolt
        TopologyBuilder builder = new TopologyBuilder();
```



```
        builder.setSpout(WORD_SPOUT, spout, 1);
        builder.setBolt(COUNT_BOLT, bolt, 1).shuffleGrouping(WORD_SPOUT);
        builder.setBolt(HBASE_BOLT, hbase, 1).fieldsGrouping(COUNT_BOLT, new Fields("word"));
        //Run the related command to submit the topology.
        StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
}
```

Running the Application and Viewing Results

Step 1 In the root directory of Storm sample code, run the **mvn package** command. After the command is executed successfully, the **storm-examples-1.0.jar** file is generated in the target directory.

Step 2 Run the related command to submit the topology.

In keytab mode, if the user changes the keytab file name, for example, **huawei.keytab**, the changed keytab file name must be added to the command as a parameter for description. The submission command example is as follows (the topology name is **hbase-test**):

storm jar /opt/jartarget/storm-examples-1.0.jar **com.huawei.storm.example.hbase.SimpleHBaseTopology hbase-test huawei.keytab**

NOTE

In security mode, ensure that Kerberos security login has been performed before the **source.jar** file is submitted. In keytab mode, the login user and the user to whom the uploaded keytab file belongs must be the same user.

HBaseBolt in the preceding example does not provide the function for creating tables. Therefore, you must verify that desired tables exist in HBase. If the tables do not exist, run the **create 'WordCount', 'cf'** statement to manually create HBase shell tables.

In HBase security mode, a user must have the permission to access related tables, column families, and columns. Therefore, the user must log in to the HBase cluster as an HBase administrator, run the **grant** command in HBase shell to apply for table access permission, such as WordCount, for the user, and submit the topology as the user.

Step 3 After the topology is submitted successfully, log in to the HBase cluster to view the topology.

Step 4 To perform login in ticket mode, perform the following operations to regularly upload a ticket. The interval for uploading the ticket depends on the deadline for updating the ticket.

1. Add the following content to a new line at the end of the **Storm/storm-0.10.0/conf/storm.yaml** file in the Storm client installation directory.

   topology.auto-credentials:

   - backtype.storm.security.auth.kerberos.AutoTGT

2. Run the **./storm upload-credentials hbase-test** command.

----End

15.5.7 Flux Development Guideline

Scenario

This topic applies only to the scenario of submitting and deploying a topology using the Flux framework in the Storm component of MRS. Determine the versions of the JAR files described in this section based on the actual situation.

The Flux framework is a framework provided by Storm 0.10.0. This framework is used to improve the topology deployment usability. Using the Flux framework, users can use a YAML file to define and deploy a topology and run the storm jar command to submit the topology. This mode facilitates topology deployment and submitting, and reduces the service development cycle.

Basic Syntax Description

Using Flux to define a topology can be classified into two scenarios: defining a new topology and defining an existing topology.

1. **Using Flux to define a new topology**

   Using Flux to define a topology indicates using a YAML file to describe a topology. A complete topology definition must contain the following parts:

   - Topology name
   - List of components used for defining the topology
   - Topology configuration
   - Topology definition, including the spout list, bolt list, and stream list

   Sample code for defining the topology name:

   ```
   name: "yaml-topology"
   ```

   Sample code for defining the component list:

   ```
   #Simple component definition
    components:
    - id: "stringScheme"
    className: "org.apache.storm.kafka.StringScheme"

    #Use a constructor to define a component.
    - id: "defaultTopicSelector"
    className: "org.apache.storm.kafka.bolt.selector.DefaultTopicSelector"
    constructorArgs:
    - "output"

    #Reference parameters as input arguments in a constructor, and use the `ref` tag to describe the
   reference.
    #When using a reference, ensure that the referenced object has been defined.
    - id: "stringMultiScheme"
    className: "org.apache.storm.spout.SchemeAsMultiScheme"
    constructorArgs:
    - ref: "stringScheme"

    #Reference the configuration items in the specified properties file as input arguments in a
   constructor, and use the `${}` tag to describe the reference.
    #If the properties file is referenced, use the --filter my-prop.properties mode to specify the path of
   the properties file when you run the storm jar command to submit the topology.
    - id: "zkHosts"
    className: "org.apache.storm.kafka.ZkHosts"
    constructorArgs:
    - "${kafka.zookeeper.root.list}"

    #Reference environment variables as input arguments in a constructor, and use the `${ENV-[NAME]}`
   ```



```
tag to describe the reference.
 #NAME must be a defined environment variable.
 - id: "zkHosts"
 className: "org.apache.storm.kafka.ZkHosts"
 constructorArgs:
 - "${ENV-ZK_HOSTS}"

 #Use the `properties` keyword to initialize the internal private variables.
 - id: spoutConfig
 className: "org.apache.storm.kafka.SpoutConfig"
 constructorArgs:
 - ref: "zkHosts"
 - "input"
 - "/kafka/input"
 - "myId"
 properties:
 - name: "scheme"
 ref: "stringMultiScheme"

 #Define the properties used by KafkaBolt.
- id: "kafkaProducerProps"
    className: "java.util.Properties"
    configMethods:
      - name: "put"
        args:
          - "bootstrap.servers"
          - "${metadata.broker.list}"
      - name: "put"
        args:
          - "acks"
          - "1"
      - name: "put"
        args:
          - "key.serializer"
          - "org.apache.kafka.common.serialization.StringSerializer"
      - name: "put"
        args:
          - "value.serializer"
          - "org.apache.kafka.common.serialization.StringSerializer"
```

Sample code for defining the topology configuration:

```
config:
 #Simple configuration item
 topology.workers: 1

 #If the configuration item value is a list, use `[]` to indicate it.
 topology.auto-credentials: ["class1","class2"]

 #The configuration item value is in the map structure.
 kafka.broker.properties:
 metadata.broker.list: "${metadata.broker.list}"
 producer.type: "async"
 request.required.acks: "0"
 serializer.class: "kafka.serializer.StringEncoder"
```

Sample code for defining the spout/bolt list:

```
#Define the spout list.
 spouts:
 - id: "spout1"
 className: "org.apache.storm.kafka.KafkaSpout"
 constructorArgs:
 - ref: "spoutConfig"
 parallelism: 1

 #Define the bolt list.
 bolts:
 - id: "bolt1"
 className: "com.huawei.storm.example.hbase.WordCounter"
 parallelism: 1
```



```
#Use a method to initialize an object, and the keyword is `configMethods`.
- id: "bolt2"
 className: "org.apache.storm.hbase.bolt.HBaseBolt"
 constructorArgs:
 - "WordCount"
 - ref: "mapper"
 configMethods:
 - name: "withConfigKey"
 args: ["hbase.conf"]
 parallelism: 1

- id: "kafkaBolt"
  className: "org.apache.storm.kafka.bolt.KafkaBolt"
  configMethods:
    - name: "withTopicSelector"
      args:
        - ref: "defaultTopicSelector"
    - name: "withProducerProperties"
      args: [ref: "kafkaProducerProps"]
    - name: "withTupleToKafkaMapper"
      args:
        - ref: "fieldNameBasedTupleToKafkaMapper"
```

Sample code for defining the stream list:

```
#To define the stream mode, you must specify the grouping mode. The keyword is `grouping`, and keywords for grouping methods provided currently are as follows:
 #`ALL`, `CUSTOM`, `DIRECT`, `SHUFFLE`, `LOCAL_OR_SHUFFLE`, `FIELDS`, `GLOBAL`, and `NONE`.
 #`CUSTOM` is used for a customized group.

 #For the definition of a simple stream, the grouping mode is SHUFFLE.
 streams:
 - name: "spout1 --> bolt1"
 from: "spout1"
 to: "bolt1"
 grouping:
 type: SHUFFLE

 #If the grouping mode is FIELDS, parameters must be entered.
 - name: "bolt1 --> bolt2"
 from: "bolt1"
 to: "bolt2"
 grouping:
 type: FIELDS
 args: ["word"]

 #If the grouping mode is CUSTOM, you must specify a customized grouping class.
 - name: "bolt-1 --> bolt2"
 from: "bolt-1"
 to: "bolt-2"
 grouping:
 type: CUSTOM
 customClass:
 className: "org.apache.storm.testing.NGrouping"
 constructorArgs:
 - 1
```

2. **Using Flux to define an existing topology**

   If a topology already exists (for example, a topology has already been defined by Java code), you can still use the Flux framework to submit and deploy the topology. In this situation, you must use the getTopology() method in the current topology definition (for example, MyTopology.java). The definition in Java is as follows:

```
public StormTopology getTopology(Config config)
Or
 public StormTopology getTopology(Map<String, Object> config)
```

   In this situation, you can use the following YAML file to define the topology:


```
name: "existing-topology" # You can specify the topology name to any value.
 topologySource:
 className: "custom-class"  #Specify the client class.
```

You can specify another method name to obtain StormTopology (non-getTopology() method). The YAML file example is as follows:

```
name: "existing-topology"
 topologySource:
 className: "custom-class "
 methodName: "getTopologyWithDifferentMethodName"
```

NOTE

The specified method must accept an input parameter of the Map<String, Object> type or the Config type and return an object of the backtype.storm.generated.StormTopology type. This method is the same as the getTopology() method.

Procedure for Developing an Application

Step 1 Verify that the Storm component has been installed and is running correctly. If the services need to connect to other components, install the required components and ensure that the components are running properly.

Step 2 Import **storm-examples** to the Eclipse development environment. For details, see **Preparing a Storm Application Development Environment**.

Step 3 Develop client services. For details, see the related YAML application examples in the **src/main/resources/flux-examples** directory of the storm-examples project.

Step 4 Obtain the related configuration files.

NOTE

This step applies only to the scenarios when other components, such as HDFS and HBase, need to be accessed for service requirements. For details about how to obtain the related configuration files, see **Storm-HDFS Development Guideline** or **Storm-HBase Development Guideline**. If the services do not require the related configuration files, skip this step.

----End

Example of the Flux Configuration File

It is a complete YAML file example for accessing the Kafka service.

```
name: "simple_kafka"

components:
- id: "zkHosts"  #Object name
className: "org.apache.storm.kafka.ZkHosts"  #Complete class name
constructorArgs:   #Constructor
- "${kafka.zookeeper.root.list}"  #Constructor parameter

- id: "stringScheme"
className: "org.apache.storm.kafka.StringScheme"

- id: "stringMultiScheme"
className: "org.apache.storm.spout.SchemeAsMultiScheme"
constructorArgs:
- ref: "stringScheme"  #A reference is used, and the value is stringScheme that has been defined.

- id: spoutConfig
className: "org.apache.storm.kafka.SpoutConfig"
```



```
constructorArgs:
- ref: "zkHosts"  #A reference is used.
- "input"
- "/kafka/input"
- "myId"
properties:  #Use properties to set the private variable whose name is "scheme" in this object.
- name: "scheme"
ref: "stringMultiScheme"

- id: "defaultTopicSelector"
className: "org.apache.storm.kafka.bolt.selector.DefaultTopicSelector"
constructorArgs:
- "output"

- id: "fieldNameBasedTupleToKafkaMapper"
className: "org.apache.storm.kafka.bolt.mapper.FieldNameBasedTupleToKafkaMapper"
constructorArgs:
- "words"  #The first input argument in the constructor
- "count"  #The second input argument in the constructor

config:
topology.workers: 1  #Set the number of workers of the topology to 1.
kafka.broker.properties:   #Set the parameters related to Kafka, and the values are in the map structure.
metadata.broker.list: "${metadata.broker.list}"
producer.type: "async"
request.required.acks: "0"
serializer.class: "kafka.serializer.StringEncoder"

spouts:
- id: "kafkaSpout"  #Spout name
className: "storm.kafka.KafkaSpout"#spout class name
constructorArgs:  #Use a constructor to perform initialization.
- ref: "spoutConfig"  #Reference parameters as input arguments in a constructor.
parallelism: 1  #Set the concurrency of the spout to 1.

bolts:
- id: "splitBolt"
className: "com.huawei.storm.example.common.SplitSentenceBolt"
parallelism: 1

- id: "countBolt"
className: "com.huawei.storm.example.kafka.CountBolt"
parallelism: 1

- id: "kafkaBolt"
className: "org.apache.storm.kafka.bolt.KafkaBolt"
configMethods:  #Invoke an internal method of the object to initialize the object.
- name: "withTopicSelector"  #Name of the invoked internal method
args:   #Parameter required by the internal method
- ref: "defaultTopicSelector"  #Only one input argument is set, and it is referenced.
- name: "withTupleToKafkaMapper"  #Invoke the second internal method.
args:
- ref: "fieldNameBasedTupleToKafkaMapper"

#Define the data stream.
streams:
- name: "kafkaSpout --> splitBolt"  #Name of the first data stream, which is only for display
from: "kafkaSpout"  #Data stream start, whose value is kafkaSpout defined in spouts
to: "splitBolt"  #Data stream end, whose value is splitBolt defined in bolts
grouping:#Define the grouping mode.
type: LOCAL_OR_SHUFFLE  #The grouping mode is local_or_shuffle.

- name: "splitBolt --> countBolt"  #Second data stream
from: "splitBolt"
to: "countBolt"
grouping:
type: FIELDS  #The grouping mode is fields.
args: ["word"]  #Parameters must be entered for the fields mode.
```



```
- name: "countBolt --> kafkaBolt"  #Third data stream
from: "countBolt"
to: "kafkaBolt"
grouping:
type: SHUFFLE  #The grouping mode is shuffle, and no parameter needs to be entered.
```

Running the Application and Viewing Results

Step 1 Run the **mvn package** command. After the command is executed successfully, the **storm-examples-1.0.jar** file is generated in the target directory.

Step 2 Copy the JAR file and developed YAML file and related **properties** files copied to any directory on the hose where the Storm client is located, for example, **/opt**.

Step 3 Run the related command to submit the topology.

storm jar /opt/jartarget/storm-examples-1.0.jar **org.apache.storm.flux.Flux --remote /opt/my-topology.yaml**

If the services are set to be started locally, run the following command to submit the topology:

storm jar /opt/jartarget/storm-examples-1.0.jar **org.apache.storm.flux.Flux --local /opt/my-topology.yaml**

NOTE

If a service is set to the local mode, ensure that the submitting environment is a normal one. Currently, services in local mode cannot be submitted by using commands in a security environment.

If the **properties** file is used, run the following command to submit the topology:

storm jar /opt/jartarget/storm-examples-1.0.jar **org.apache.storm.flux.Flux --remote /opt/my-topology.yaml --filter /opt/my-prop.properties**

Step 4 After the topology is submitted successfully, log in to the Storm UI to view the topology.

----End

