Data Ingestion Service

SDK Reference

Issue	01
Date	2025-02-27

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Contents

1 Overview

1.1 What Is DIS SDK

Introduction to DIS

Data Ingestion Service (DIS) addresses the challenge of transmitting data from outside the cloud to inside the cloud. DIS builds data intake streams for custom applications capable of processing or analyzing streaming data. DIS continuously captures, transmits, and stores terabytes of data from hundreds of thousands of sources every hour, such as logs, social media feeds, website clickstreams, and location-tracking events.

Cloud services feature flexible expansion and allow infrastructure to be deployed in multiple regions, while delivering high reliability. Users can deploy DIS in specific regions based on site requirements to obtain rapid access speed at an affordable price.

DIS manages the infrastructure, storage, networking, and configuration needed to stream your data. You do not have to worry about provisioning, deployment, and constant maintenance of hardware. In addition, DIS synchronously replicates data across availability zones, providing high availability and data durability.

Introduction to SDK

Data Ingestion Service Software Development Kit (DIS SDK) is the encapsulation of RESTful APIs provided by DIS to simplify user development. Users can directly use API functions provided by DIS SDK to obtain the DIS service capabilities.

1.2 Content Navigation

This guide describes how to install and configure an environment and how to call functions provided by DIS SDK for secondary development.

Chapter	Describes
What Is DIS SDK **Content Navigation**	Concepts of DIS and DIS SDK.
SDK Download **Compatibility** **How Do I Check Software Package Integrity?**	Resources required by the DIS SDK for secondary development.
Enabling DIS	How to enable DIS.
Obtaining Authentication Information	Initialization operations performed before using the DIS SDK to complete secondary development.
Java: **Preparing the Environment** to **Changing Partition Quantity**	How to use the DIS SDK to perform common operations (matching Java).
DIS Server-Side Error Codes	Errors that may occur during the use of DIS SDK.

2 Related Resources

2.1 SDK Download

Download the DIS Java SDK from **https://github.com/huaweicloud/huaweicloud-sdk-java-dis**.

After obtaining the DIS SDK software package and verification file, verify the integrity of the software package. For details, see **How Do I Check Software Package Integrity?**.

2.2 Compatibility

Supported JDK versions: 1.8.0 and later versions

Supported Python versions: 2.7 and later versions

2.3 How Do I Check Software Package Integrity?

This section describes how to verify integrity of the DIS SDK software package on a Linux system by using a verification file.

Prerequisites

- The PuTTY tool is available.
- The WinSCP tool is available.

Procedure

Step 1 Upload the DIS SDK software package **huaweicloud-sdk-dis-x.x.x1.2.3.zip** to any directory on the Linux system by using WinSCP.

NOTE

In **huaweicloud-sdk-dis-x.x.x.zip**, **x.x.x** indicates the version number of the DIS SDK software package.

Step 2 Log in to the Linux system by using PuTTY. In the directory in which **huaweicloud-sdk-dis-x.x.x1.2.3.zip** is stored, run the following command to obtain the verification code of the DIS SDK software package

sha256sum huaweicloud-sdk-dis-x.x.x.zip

Example verification code:

```
# sha256sum dis-sdk-x.x.x.zip
8be2c937e8d78b1a9b99777cee4e7131f8bf231de3f839cf214e7c5b5ba3c088 huaweicloud-sdk-dis-x.x.x.zip
```

Step 3 Open the DIS SDK verification file **huaweicloud-sdk-dis-x.x.x1.2.3.zip.sha256sum** and compare it with the verification code obtained in **Step 2**.

- If they are consistent, the DIS-SDK compression package has not been tampered with.
- If they are inconsistent, the DIS SDK software package is tampered with and you need to obtain it again.

----End

3 Enabling DIS

3.1 Enabling DIS

Step 1 Register an account.

Step 2 Enable the DIS service.

Top up your account before using DIS.

1. Log in to the DIS console.
2. Click **Fees** at the upper right corner.
3. Click **Top Up**. The Top-Up page is displayed.
4. Top up your account as instructed.
5. After your account is topped up, close the Top-Up page, and return to the management console homepage.
6. Click **Data Ingestion Service** to enable the service.

Step 3 Create access keys.

DIS uses AKs and SKs for signature verification to ensure that only authorized accounts can access specified DIS resources.

1. Log in to the DIS console.
2. Click your user name in the upper right corner of the page, and choose **My Credentials** from the drop-down list.
3. On the **My Credentials** page, click the **Access Keys** tab. Then click **Add Access Key**.
4. Enter the required information, and click **OK**.

NOTE

- Each user can create two sets of access keys at most.
- Keep the access key file confidential in order to prevent information leakage. If you click **Cancel**, the access key will not be downloaded and cannot be downloaded later. You must delete the access key and create one later.

----End

4 Creating a DIS Stream

For details about how to create a DIS stream, see "Creating a DIS Stream" in the *Data Ingestion Service User Guide*.

For details, see **Creating a DIS Stream**.

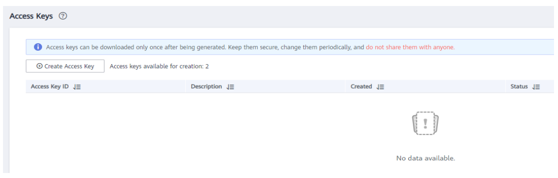
5 Obtaining Authentication Information

Obtaining an Access Key

To obtain an access key, perform the following steps:

1. Log in to the management console, move the cursor to the username in the upper right corner, and select **My Credentials** from the drop-down list.
2. On the **My Credentials** page, choose **Access Keys**, and click **Create Access Key**. See **Figure 5-1**.

Figure 5-1 Clicking Create Access Key



Access Keys

Access keys can be downloaded only once after being generated. Keep them secure, change them periodically, and do not share them with anyone.

Create Access Key   Access keys available for creation: 2

Access Key ID	Description	Created	Status

No data available.

3. Click **OK** and save the access key file as prompted. The access key file will be saved to your browser's configured download location. Open the **credentials.csv** file to view **Access Key Id** and **Secret Access Key**.

NOTE

- Only two access keys can be added for each user.
- To ensure access key security, the access key is automatically downloaded only when it is generated for the first time and cannot be obtained from the management console later. Keep them properly.

Obtaining a Project ID and Account ID

A project is a group of tenant resources, and an account ID corresponds to the current account. The IAM ID corresponds to the current user. You can view the project IDs, account IDs, and user IDs in different regions on the corresponding pages.

1. Register with and log in to the management console.
2. Hover the cursor on the username in the upper right corner and select **My Credentials** from the drop-down list.

3. On the **API Credentials** page, obtain the account name, account ID, IAM username, and IAM user ID, and obtain the project and its ID from the project list.

Obtaining Regions and Endpoints

An endpoint is the **request address** for calling an API. Endpoints vary depending on services and regions. You can obtain endpoints of the service from **Endpoints**.

6 Getting Started with SDK

6.1 Using the Java SDK

6.1.1 Preparing the Environment

- Download JDK1.8 or a later version from the **Oracle official website** and install it, and configure Java environment variables.
- Download Eclipse IDE for Java Developers of the latest version from the **Eclipse's official website**, and install it.
- Configure the JDK in Eclipse.

6.1.2 Configuring a Sample Project

The **huaweicloud-sdk-dis-java-*X.X.X*.zip** package downloaded from **SDK Download** provides a sample project. You can use a development tool (such as Eclipse) to compile and run the sample project on a local device. You can also develop applications based on the sample project. The sample project code is available in the **\dis-sdk-demo\src\main\java\com\bigdata\dis\sdk\demo** directory.

Sample Code	Describes
ConsumerDemo.java	How to download data.
ProducerDemo.java	How to upload data.

Procedure

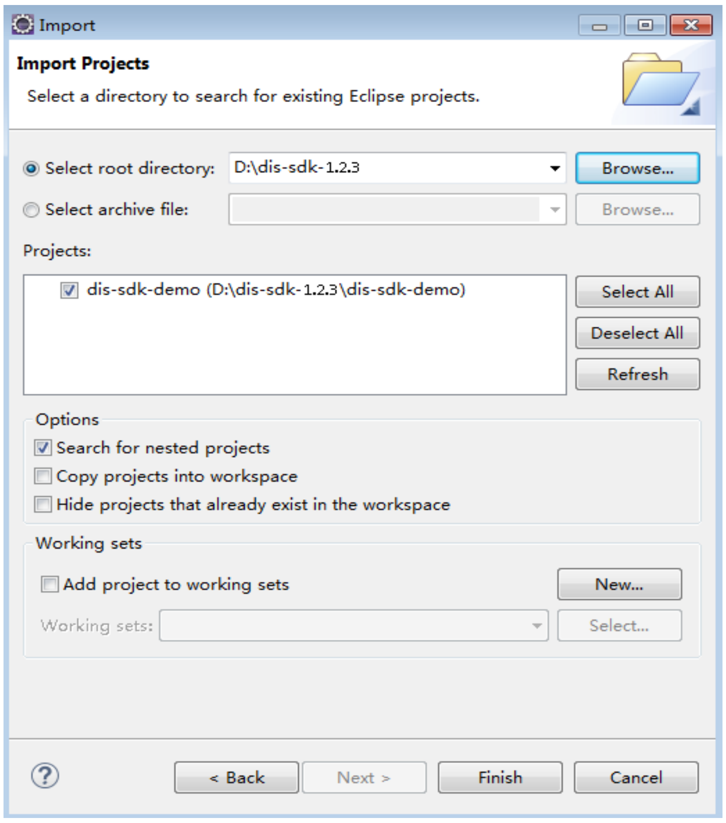
Step 1 Decompress the **huaweicloud-sdk-dis-java-*X.X.X*.zip** package downloaded from **SDK Download** to obtain the **dis-sdk-demo** package and sample project.

Step 2 Import the Eclipse project.

1. Start Eclipse. Choose **File** > **Import**. The **Import** dialog box is displayed.

2. Choose **Maven** > **Existing Maven Projects**, and click **Next**. The **Import** dialog box is displayed.
3. Click **Browse** and select a root directory for the **dis-sdk-demo** sample project. In the **Projects** area, select a sample project.

Figure 6-1 Import Maven Projects



Import

Import Projects

Select a directory to search for existing Eclipse projects.

(●) Select root directory: D:\dis-sdk-1.2.3

( ) Select archive file:

Projects:

☑ dis-sdk-demo (D:\dis-sdk-1.2.3\dis-sdk-demo)

Options

☑ Search for nested projects

☐ Copy projects into workspace

☐ Hide projects that already exist in the workspace

Working sets

☐ Add project to working sets

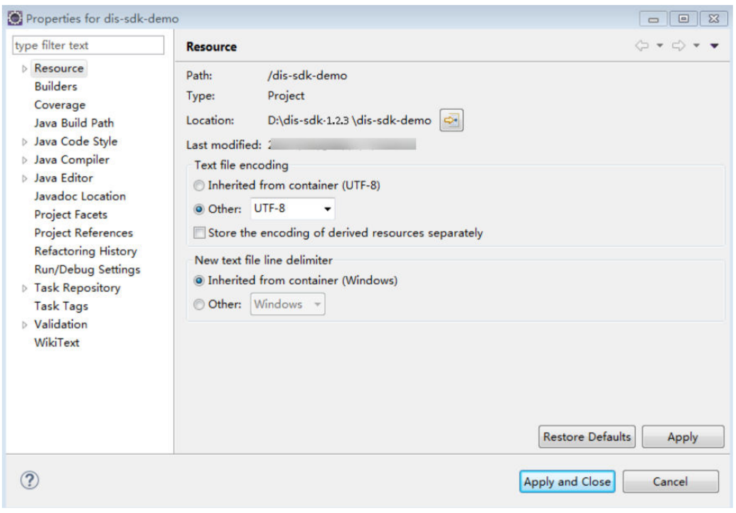
Working sets:

4. Click **Finish**.

Step 3 Configure the demo project.

1. Set the project code to UTF-8.
   a. In the navigation tree on the left, right-click the required project under **Project Explorer** and choose **Properties** from the shortcut menu. The **Properties for dis-sdk-demo** page is displayed.
   b. In the left list, select **Resource**. The **Resource** pane is displayed.
   c. Set **Text file encoding** to **Other**. In the **Others** drop-down list, select **UTF-8**.
   d. Click **Apply and Close** to complete the encoding configuration.

Figure 6-2 Selecting Resource


Properties for dis-sdk-demo

type filter text

- Resource
- Builders
- Coverage
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- Project Facets
- Project References
- Refactoring History
- Run/Debug Settings
- Task Repository
- Task Tags
- Validation
- WikiText

Resource

Path: /dis-sdk-demo
Type: Project
Location: D:\dis-sdk-1.2.3 \dis-sdk-demo
Last modified: [redacted]

Text file encoding
- Inherited from container (UTF-8)
- Other: UTF-8 (selected)
- Store the encoding of derived resources separately

New text file line delimiter
- Inherited from container (Windows) (selected)
- Other: Windows

2. Add a JDK.
   a. In the navigation tree on the left, right-click the required project under **Project Explorer** and choose **Properties** from the shortcut menu. The **Properties for dis-sdk-demo** page is displayed.
   b. In the left list, select **Java Build Path**. The **Java Build Path** pane is displayed.
   c. Click the **Libraries** tab, and then click **Add Library**. The **Add Library** dialog box is displayed.
   d. Select **JRE System Library** and click **Next**. Ensure that the version of **Workspace default JRE** is jdk1.8 or later.
   e. Click **Finish** to exit the **Add Library** dialog box.
   f. Click **Apply and Close** to add the JDK.

----End

6.1.3 Initializing a DIS SDK Client Instance

You can initialize the DIS SDK client instance using either of the following methods. The first method is recommended. For details about **endpoint**, **ak**, **sk**, **region**, and **projectId**, see **Obtaining Authentication Information**.

- Use codes to initialize the DIS SDK client instance.
```
//Create a DIS client instance.
DIS dic = DISClientBuilder.standard()
    .withEndpoint("YOUR_ENDPOINT")
    .withAk("YOUR_AK")
    .withSk("YOUR_SK")
    .withProjectId("YOUR_PROJECT_ID")
    .withRegion("YOUR_REGION")
    // Configure the number of retries upon a failure.
    .withProperty(DISConfig.PROPERTY_PRODUCER_RECORDS_RETRIES, "-1")
    .withProperty(DISConfig.PROPERTY_PRODUCER_EXCEPTION_RETRIES, "-1")
    .build();
```
- Perform the following steps to initialize the DIS client instance if a proxy needs to be used:
```
//Create a DIS client instance.
DIS dic = DISClientBuilder.standard()
    .withEndpoint("YOUR_ENDPOINT")
    .withAk("YOUR_AK")
    .withSk("YOUR_SK")
    .withProjectId("YOUR_PROJECT_ID")
    .withRegion("YOUR_REGION")
.withProxyHost("YOUR_PROXY_HOST") //Proxy IP address.
.withProxyPort("YOUR_PROXY_PORT") //Proxy port.
.withProxyProtocol(Protocol.HTTP) //Proxy protocol. The default value is HTTP.
.withProxyUsername("YOUR_PROXY_USER_NAME") //Proxy username (optional).
.withProxyPassword("YOUR_PROXY_PASSWORD") //Proxy password (optional)
    // Configure the number of retries upon a failure.
    .withProperty(DISConfig.PROPERTY_PRODUCER_RECORDS_RETRIES, "-1")
    .withProperty(DISConfig.PROPERTY_PRODUCER_EXCEPTION_RETRIES, "-1")
    .build();
```

- To enable transmission compression, initialize the DIS client as follows:

```
//Create a DIS client instance.
DIS dic = DISClientBuilder.standard()
    .withEndpoint("YOUR_ENDPOINT")
    .withAk("YOUR_AK")
    .withSk("YOUR_SK")
    .withProjectId("YOUR_PROJECT_ID")
    .withRegion("YOUR_REGION")
    .withBodyCompressEnabled(true)
.withBodyCompressType(CompressionType.ZSTD) //Configure the compression algorithm. Currently, 
lz4 and zstd are supported. The default value is lz4.
    // Configure the number of retries upon a failure.
    .withProperty(DISConfig.PROPERTY_PRODUCER_RECORDS_RETRIES, "-1")
    .withProperty(DISConfig.PROPERTY_PRODUCER_EXCEPTION_RETRIES, "-1")
    .build();
```

- If data needs to be encrypted before being uploaded to DIS on the client, use the encryption method provided by the DIS SDK. That is, add the **DataEncryptEnabled** and **data.password** parameters when building the disclient.

```
// Create a DIS client instance.
DIS dic = DISClientBuilder.standard()
    .withEndpoint("YOUR_ENDPOINT")
    .withAk("YOUR_AK")
    .withSk("YOUR_SK")
    .withProjectId("YOUR_PROJECT_ID")
    .withRegion("YOUR_REGION")
    .withDataEncryptEnabled(true)
.withProperty("data.password", "xxx")//xxx indicates the data encryption key configured by the user.
    // Configure the number of retries upon a failure.
    .withProperty(DISConfig.PROPERTY_PRODUCER_RECORDS_RETRIES, "-1")
    .withProperty(DISConfig.PROPERTY_PRODUCER_EXCEPTION_RETRIES, "-1")
    .build();
```

NOTE

If JAVA SDK is used to encrypt the uploaded data, you also need to use JAVA SDK to configure the same key for data reading.

- Use the configuration file to initialize a DIS SDK client instance.

  Add the following configuration items to the **dis.properties** file in the **dis-sdk-demo\src\main\resources** directory:

  – ak/sk: AK/SK created on the IAM
  – region: region of the stream
  – endpoint: access address of the DIS
  – projectId: project ID of the stream

```
//Create a DIS SDK client instance.
        DIS dic = DISClientBuilder.standard().build();
```

6.1.4 Creating a Stream

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

When you use the DIS SDK to create a DIS stream, specify the stream name, number of partitions in the stream, and stream type.

STREAM_TYPE_COMMON indicates a common stream, and **STREAM_TYPE_ADVANCED** indicates an advanced stream.

```
CreateStreamRequest createStreamRequest = new CreateStreamRequest();
//Configure a stream name.
String streamName = "myStream";
createStreamRequest.setStreamName(streamName);
//Configure a stream type. The value can be Common or Advanced.
createStreamRequest.setStreamType(StreamType.COMMON.name());
//Configure the number of partitions in the stream.
createStreamRequest.setPartitionCount(3);
//Configure the retention period of the stream in units of hours. The value is N x 24, where N ranges from 1
to 7.
createStreamRequest.setDataDuration(24);
//Configure the source data type of the stream. Default value: BLOB
createStreamRequest.setDataType(DataTypeEnum.BLOB.name());
```

After configuring **CreateStreamRequest**, you can create a stream by calling createStream.

```
dic.createStream(createStreamRequest);
```

6.1.5 Creating a Dump Task

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

When using the DIS SDK to create a dump task, you need to specify the stream name, dump task name, dump interval, and dump destination.

Creating an OBS Dump Task

```
CreateTransferTaskRequest request = new CreateTransferTaskRequest();

//Configure the stream name. You can create streams can be created on the DIS console.
request.setStreamName(streamName);

//Configure the dump task name.
OBSDestinationDescriptorRequest descriptor = new OBSDestinationDescriptorRequest();
descriptor.setTransferTaskName(taskName);

//Configure the OBS bucket name and folder name. You can create OBS buckets and files on the OBS
console or client.
descriptor.setObsBucketPath("obs-dis");
descriptor.setFilePrefix("transfertask");

//Configure the dump interval that is expressed in units of seconds.
descriptor.setDeliverTimeInterval(900);

//(Optional) Create an IAM agency named dis_admin_agency on the DIS management page and use it to
access specific cloud services. For the first time to create an IAM agency, you have to authorize it.
descriptor.setAgencyName("dis_admin_agency");

//(Optional) Configure the dump file format. By default, the value is Text. Other available options are
Parquet and CarbonData.
descriptor.setDestinationFileType(DestinationFileTypeEnum.TEXT.getType());
```

```
//Configure the initial offset when data is pulled from the DIS stream. The value can be LATEST or
TRIM_HORIZON. LATEST is the default value, indicating that data is read from the latest uploaded records
in the stream. TRIM_HORIZON indicates that data is read from the earliest unexpired records in the stream.
descriptor.setConsumerStrategy(PartitionCursorTypeEnum.LATEST.name());

request.setObsDestinationDescriptor(descriptor);
```

After configuring **CreateTransferTaskRequest**, you can call the **createTransferTask** method to create the dump task.

```
dic.createTransferTask(request);
```

Create an MRS Dump Task

```
CreateTransferTaskRequest request = new CreateTransferTaskRequest();

//Configure the stream name. You can create streams on the DIS console.
request.setStreamName(streamName);

//Configure the dump task name.
MRSDestinationDescriptorRequest descriptor = new MRSDestinationDescriptorRequest();
descriptor.setTransferTaskName(taskName);

//Configure the MRS cluster information about the cluster name and ID. You can create and query cluster
information on the MRS console. The cluster must be in non-security mode.
descriptor.setMrsClusterName("mrs_dis");
descriptor.setMrsClusterId("fe69a732-c7d3-4b0f-8cda-ec9eca0cf141");

//Configure the OBS bucket and folder that is used to temporarily store the data to be dumped to MRS and
the dump failure data. You can create OBS buckets and folders on the OBS console.
descriptor.setObsBucketPath("obs-dis");
descriptor.setFilePrefix("transfertask");

//Configure the dump interval that is expressed in units of seconds.
descriptor.setDeliverTimeInterval(900);

//(Optional) Create an IAM agency named dis_admin_agency on the DIS management page and use it to
access specific cloud services. For the first time to create an IAM agency, you have to authorize it.
descriptor.setAgencyName("dis_admin_agency");

//(Optional) Configure the dump file format. By default, the value is Text. Other available options are
Parquet and CarbonData.
descriptor.setDestinationFileType(DestinationFileTypeEnum.TEXT.getType());

//Configure the initial offset when data is pulled from the DIS stream. The value can be LATEST or
TRIM_HORIZON. LATEST is the default value, indicating that data is read from the latest uploaded records
in the stream. TRIM_HORIZON indicates that data is read from the earliest unexpired records in the stream.
descriptor.setConsumerStrategy(PartitionCursorTypeEnum.LATEST.name());

request.setMrsDestinationDescriptor(descriptor);
```

After configuring **CreateTransferTaskRequest**, you can call the **createTransferTask** method to create the dump task.

```
dic.createTransferTask(request);
```

Creating a DLI Dump Task

```
CreateTransferTaskRequest request = new CreateTransferTaskRequest();

//Configure the stream name. You can create streams on the DIS console.
request.setStreamName(streamName);

//Configure the dump task name.
UqueryDestinationDescriptorRequest descriptor = new UqueryDestinationDescriptorRequest();
descriptor.setTransferTaskName(taskName);

//Configure DLI information about the database and internal table names. You can create and query DLI on
```

```
the DLI console. The DLI table must be the internal table.
descriptor.setDliDatabaseName("dis_dli");
descriptor.setDliTableName("dis_test");

//Configure the OBS bucket and folder that is used to temporarily store the data to be dumped to DLI and
the dump failure data. You can create OBS buckets and folders on the OBS console or client.
descriptor.setObsBucketPath("obs-dis");
descriptor.setFilePrefix("transfertask");

//Configure the dump interval that is expressed in units of seconds.
descriptor.setDeliverTimeInterval(900);

//(Optional) Create an IAM agency named dis_admin_agency on the DIS management page and use it to
access specific cloud services. For the first time to create an IAM agency, you have to authorize it.
descriptor.setAgencyName("dis_admin_agency");

//Configure the initial offset when data is pulled from the DIS stream. The value can be LATEST or
TRIM_HORIZON. LATEST is the default value, indicating that data is read from the latest uploaded records
in the stream. TRIM_HORIZON indicates that data is read from the earliest unexpired records in the stream.
descriptor.setConsumerStrategy(PartitionCursorTypeEnum.LATEST.name());

request.setDliDestinationDescriptor(descriptor);
```

After configuring **CreateTransferTaskRequest**, you can call the **createTransferTask** method to create the dump task.

```
dic.createTransferTask(request);
```

Creating a DWS Dump Task

```
CreateTransferTaskRequest request = new CreateTransferTaskRequest();

//Configure the stream name. You can create streams on the DIS console.
request.setStreamName(streamName);

//Configure the dump task name.
DwsDestinationDescriptorRequest descriptor = new DwsDestinationDescriptorRequest();
descriptor.setTransferTaskName(taskName);

//Configure the DWS cluster information about the cluster name, ID, and database. You can create and
query clusters on the DWS console, and create tables using its client or other methods.
descriptor.setDwsClusterName("dis_test");
descriptor.setDwsClusterId("92f90f6a-de4d-4689-82f6-320c328b0062");
descriptor.setDwsDatabaseName("postgres");
descriptor.setDwsSchema("dbadmin");
descriptor.setDwsTableName("distable01");
descriptor.setDwsDelimiter("|");
descriptor.setUserName("dbadmin");
descriptor.setUserPassword("xxxx");

//Call KMS to encrypt the DWS password to keep user data secure. You can create and query KMS on the
KMS console.
descriptor.setKmsUserKeyName("qiyinshan");
descriptor.setKmsUserKeyId("9521c600-64a8-4971-ad36-7bbfa6d00c41");

//Configure the OBS bucket and folder that is used to temporarily store the data to be dumped to DWS and
the dump failure data. You can create OBS buckets and folders on the OBS console or client.
descriptor.setObsBucketPath("obs-dis");
descriptor.setFilePrefix("transfertask");

//Configure the dump interval that is expressed in units of seconds.
descriptor.setDeliverTimeInterval(900);

//(Optional) Create an IAM agency named dis_admin_agency on the DIS management page and use it to
access specific cloud services. For the first time to create an IAM agency, you have to authorize it.
descriptor.setAgencyName("dis_admin_agency");

//Configure the initial offset when data is pulled from the DIS stream. The value can be LATEST or
TRIM_HORIZON. LATEST is the default value, indicating that data is read from the latest uploaded records
```

```
in the stream. TRIM_HORIZON indicates that data is read from the earliest unexpired records in the stream.
descriptor.setConsumerStrategy(PartitionCursorTypeEnum.LATEST.name());

request.setDwsDestinationDescriptor(descriptor);
```

After configuring **CreateTransferTaskRequest**, you can call the **createTransferTask** method to create the dump task.

```
dic.createTransferTask(request);
```

Creating a CloudTable Dump Task

```
CreateTransferTaskRequest request = new CreateTransferTaskRequest();

//Configure the stream name. You can create streams on the DIS console.
request.setStreamName(streamName);

//Configure the dump task name.
CloudtableDestinationDescriptorRequest descriptor = new CloudtableDestinationDescriptorRequest();
descriptor.setTransferTaskName(taskName);

//Configure the CloudTable cluster information about the cluster name, ID, and database. You can create
and query clusters on the CloudTable console, and create tables using its client or other methods.
descriptor.setCloudtableClusterName("dis_test");
descriptor.setCloudtableClusterId("92f90f6a-de4d-4689-82f6-320c328b0062");
descriptor.setCloudtableTableName("dis");

//To configure CloudtableSchema, see CloudTable User Guide.
CloudtableSchema cloudtableSchema = new CloudtableSchema();
List<SchemaField> rowKeySchema = new ArrayList<>();
SchemaField field1 = new SchemaField();
field1.setValue("id");
field1.setType("String");
rowKeySchema.add(field1);
SchemaField rField1 = new SchemaField();
rField1.setValue("group.users.id");
rField1.setType("String");
rowKeySchema.add(rField1);
List<SchemaField> columnsSchema = new ArrayList<>();
SchemaField field2 = new SchemaField();
field2.setColumnFamilyName("user");
field2.setQualifierName("id");
field2.setValue("group.users.id");
field2.setType("String");
SchemaField field3 = new SchemaField();
field3.setColumnFamilyName("user");
field3.setQualifierName("age");
field3.setValue("group.users.age");
field3.setType("Int");
columnsSchema.add(field2);
columnsSchema.add(field3);
cloudtableSchema.setRowKeySchema(rowKeySchema);
cloudtableSchema.setColumnsSchema(columnsSchema);
descriptor.setCloudtableSchema(cloudtableSchema);

//Configure the OBS bucket and folder that is used to store dump failure data. You can create OBS buckets
and folders on the OBS console or client.
descriptor.setObsBackupBucketPath("obs-dis");
descriptor.setBackupfilePrefix("transfertask");

//(Optional) Create an IAM agency named dis_admin_agency on the DIS management page and use it to
access specific cloud services. For the first time to create an IAM agency, you have to authorize it.
descriptor.setAgencyName("dis_admin_agency");

//Configure the initial offset when data is pulled from the DIS stream. The value can be LATEST or
TRIM_HORIZON. LATEST is the default value, indicating that data is read from the latest uploaded records
in the stream. TRIM_HORIZON indicates that data is read from the earliest unexpired records in the stream.
descriptor.setConsumerStrategy(PartitionCursorTypeEnum.LATEST.name());
```

```
request.setCloudtableDestinationDescriptor(descriptor);
```

After configuring **CreateTransferTaskRequest**, you can call the **createTransferTask** method to create the dump task.

```
dic.createTransferTask(request);
```

Creating a CloudTable OpenTSDB Dump Task

```
CreateTransferTaskRequest request = new CreateTransferTaskRequest();

//Configure the stream name. You can create streams on the DIS console.
request.setStreamName(streamName);

//Configure the dump task name.
CloudtableDestinationDescriptorRequest descriptor = new CloudtableDestinationDescriptorRequest();
descriptor.setTransferTaskName(taskName);

//Configure the CloudTable OpenTSDB cluster information about the cluster name, ID, and database. You
can create and query clusters on the CloudTable console, and create tables using its client or other methods.
descriptor.setCloudtableClusterName("dlf_test");
descriptor.setCloudtableClusterId("92f90f6a-de4d-4689-82f6-320c328b0062");

//To configure OpenTSDBSchema, see CloudTable User Guide.
List<SchemaField> metricSchema = new ArrayList<>();
SchemaField field1 = new SchemaField();
field1.setValue("group.users.id");
field1.setType("String");
metricSchema.add(field1);
SchemaField timestampSchema = new SchemaField();
timestampSchema.setColumnFamilyName("user");
timestampSchema.setFormat("yyyy/MM/dd HH:mm:ss");
timestampSchema.setValue("group.users.birthday");
timestampSchema.setType("String");
SchemaField valueSchema = new SchemaField();
valueSchema.setValue("group.users.age");
valueSchema.setType("Int");
List<SchemaField> tagsSchema = new ArrayList<>();
SchemaField field2 = new SchemaField();
field2.setName("group.users.id");
field2.setValue("group.users.id");
field2.setType("String");
SchemaField field3 = new SchemaField();
field3.setName("age");
field3.setValue("group.users.age");
field3.setType("Int");
tagsSchema.add(field2);
tagsSchema.add(field3);
OpenTSDBSchema openTSDBSchema = new OpenTSDBSchema();
openTSDBSchema.setMetricSchema(metricSchema);
openTSDBSchema.setTimestampSchema(timestampSchema);
openTSDBSchema.setValueSchema(valueSchema);
openTSDBSchema.setTagsSchema(tagsSchema);
List<OpenTSDBSchema> openTSDBSchemaList = new ArrayList<>();
openTSDBSchemaList.add(openTSDBSchema);
descriptor.setOpentsdbSchema(openTSDBSchemaList);

//Configure the OBS bucket and folder that is used to store dump failure data. You can create OBS buckets
and folders on the OBS console or client.
descriptor.setObsBackupBucketPath("obs-dis");
descriptor.setBackupfilePrefix("transfertask");

//(Optional) Create an IAM agency named dis_admin_agency on the DIS management page and use it to
access specific cloud services. For the first time to create an IAM agency, you have to authorize it.
descriptor.setAgencyName("dis_admin_agency");

//Configure the initial offset when data is pulled from the DIS stream. The value can be LATEST or
TRIM_HORIZON. LATEST is the default value, indicating that data is read from the latest uploaded records
```

```
in the stream. TRIM_HORIZON indicates that data is read from the earliest unexpired records in the stream.
descriptor.setConsumerStrategy(PartitionCursorTypeEnum.LATEST.name());

request.setCloudtableDestinationDescriptor(descriptor);
```

After configuring **CreateTransferTaskRequest**, you can call the **createTransferTask** method to create the dump task.

```
dic.createTransferTask(request);
```

6.1.6 Updating a Dump Task

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

When using the DIS SDK to update a dump task, you need to specify the stream name, dump task name, dump interval, and dump destination.

```
//Configure global parameters of the dump task. A single-parameter update is not supported.
UpdateTransferTaskRequest request = new UpdateTransferTaskRequest();

//Configure the name of the stream to which the dump task to be updated belongs.
request.setStreamName(streamName);

//Configure the name of the dump task to be updated.
OBSDestinationDescriptorRequest descriptor = new OBSDestinationDescriptorRequest();
descriptor.setTransferTaskName(taskName);

//Configure the OBS bucket name and folder name. You can create OBS buckets and files on the OBS
console or client.
descriptor.setObsBucketPath("obs-dis1");
descriptor.setFilePrefix("transfertask");

//Configure the dump interval that is expressed in units of seconds.
descriptor.setDeliverTimeInterval(300);

//(Optional) Configure the dump file format. By default, the value is Text. Other available options are
Parquet and CarbonData.
descriptor.setDestinationFileType(DestinationFileTypeEnum.TEXT.getType());

request.setObsDestinationDescriptor(descriptor);
```

After configuring **UpdateTransferTaskRequest**, you can call the **updateTransferTask** method to update the dump task.

```
dic.updateTransferTask(request);
```

6.1.7 Deleting a Dump Task

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

You can use the DIS SDK to delete a specified dump task.

```
DeleteTransferTaskRequest request = new DeleteTransferTaskRequest();

//Configure the name of the stream to which the dump task belongs.
request.setStreamName(streamName);

//Configure the name of the dump task to be deleted.
request.setTransferTaskName(taskName);
```

After configuring **DeleteTransferTaskRequest**, you can call the **deleteTransferTask** method to create the dump task.

```
dic.deleteTransferTask(request);
```

6.1.8 Querying a Dump Task List

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

You can use the DIS SDK to query the dump task list of a specified stream.

```
ListTransferTasksRquest request = new ListTransferTasksRquest();

//Configure the name of the stream to be queried.
request.setStreamName(streamName);
```

After configuring **ListTransferTaskRequest**, you can call the **listTransferTask** method to query the dump task list of a specified stream.

```
ListTransferTasksResult result = dic.listTransferTasks(request);
```

Information similar to the following is displayed when you query the dump task list:

```
{
    "tasks":[
        {
            "destination_type":"DLI",
            "task_name":"task_Ztab",
            "create_time":1552457808502,
            "state":"RUNNING",
            "last_transfer_timestamp":1552458085454
        },
        {
            "destination_type":"OBS",
            "task_name":"task_qTd9",
            "create_time":1552355757885,
            "state":"RUNNING",
            "last_transfer_timestamp":1552458158527
        }
    ],
    "total_number":2
}
```

6.1.9 Querying Dump Details

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

You can use the DIS SDK to query the details of a specified dump task.

```
DescribeTransferTaskRequest request = new DescribeTransferTaskRequest();

//Configure the name of the stream to be queried.
request.setStreamName(streamName);

//Configure the name of the dump task to be queried.
request.setTransferTaskName(taskName);
```

After configuring **DescribeTransferTaskRequest**, you can call the **describeTransferTask** method to query the details of a specified dump task.

```
DescribeTransferTaskResult result = dic.describeTransferTask(request);
```

Information similar to the following is displayed when you query the dump details:

```
{
    "partitions":[
        {
            "partitionId":"shardId-0000000000",
            "discard":0,
            "state":"RUNNING",
            "last_transfer_timestamp":1552458085454,
            "last_transfer_offset":56
        }
    ],
    "stream_name":"dis_test1",
    "task_name":"task_Ztab",
    "task_id":"gGGu2WN88XbmRTm64nJ",
    "destination_type":"DLI",
    "state":"RUNNING",
    "create_time":1552457808502,
    "last_transfer_timestamp":1552458085454,
    "dli_destination_description":{
        "agency_name":"dis_admin_agency",
        "file_prefix":"dli",
        "obs_bucket_path":"dis.test.not.delete",
        "deliver_time_interval":300,
        "consumer_strategy":"LATEST"
    }
}
```

6.1.10 Deleting a Stream

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

Use the DIS SDK to delete a specified DIS stream.

```
//Specify the name of the stream to be deleted.
String streamName = "myStream";
DeleteStreamRequest deleteStreamRequest = new DeleteStreamRequest();
deleteStreamRequest.setStreamName(streamName);
```

After configuring **DeleteStreamRequest**, you can delete a stream by invoking deleteStream.

```
dic.deleteStream(deleteStreamRequest);
```

6.1.11 Querying a Stream List

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

You can use the DIS SDK to list active streams.

Use the **setLimit** method to set the number of streams returned each time. If **setLimit** is not specified, a maximum of 10 streams are returned by default.

```
ListStreamsRequest listStreamsRequest = new ListStreamsRequest();
listStreamsRequest.setLimit(5);
System.out.println("listStream: " + JsonUtils.objToJson(dic.listStreams(listStreamsRequest)));
```

Table 6-1 Parameters

Parameter	Type	Description
limit	long	The maximum number of DIS streams to list in a single API call. Value range: 1 to 100 Default value: 10

Parameter	Type	Description
exclusiveStartStream-Name	string	Name of the DIS stream to start the stream list with. The returned stream list does not contain this DIS stream name. If pagination query is required, this parameter is not transferred when you query data on the first page. If the value of **has_more_streams** is **true**, the query is performed on the next page. The value of **exclusiveStartStream-Name** is the name of the last stream in the query result of the first page.

NOTE

In this demo, **start_Stream_Name** is defined as a stream name before **stream0**, and **limit** is set to **5**. The following information is returned:

```
listStream: {"total_number":20,"stream_names":
["Stream0","Stream1","Stream2","Stream3","Stream4"],"has_more_streams":true}
```

Table 6-2 Response parameter description

Parameter	Type	Description
total_number	Int	Total number of all the DIS streams created by the current tenant.
stream_names	List<String>	List of the streams meeting the current requests.
has_more_streams	Boolean	Specify whether there are more matching DIS streams to list. Possible values: ● **true**: There are more streams. ● **false**: There are no more partitions.

6.1.12 Querying Stream Details

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

Use the DIS SDK to query the details about a specified stream.

```
String streamName = "myStream";
DescribeStreamRequest describeStreamRequest = new DescribeStreamRequest();
describeStreamRequest.setStreamName(streamName);
System.out.println("descStream: " + JsonUtils.objToJson(dic.describeStream(describeStreamRequest)));
```

The returned stream details are as follows:

```
descStream: DescribeStreamResult [streamId=JnYbsMfWNn81e8n2mOC, streamName=myStream,
createTime=1540977519187, lastModifiedTime=1540977519983, retentionPeriod=24, status=RUNNING,
streamType=ADVANCED, dataType=BLOB, writablePartitionCount=2, readablePartitionCount=2,
partitions=[PartitionResult{partitionId='shardId-0000000000', hashRange='[0 : 4611686018427387902]',
status='ACTIVE', parentPartitionIds='null', sequenceNumberRange='[0 : 13286444]'},
PartitionResult{partitionId='shardId-0000000001', hashRange='[4611686018427387903 :
9223372036854775807]', status='ACTIVE', parentPartitionIds='null', sequenceNumberRange='[0 :
13288589]'}], hasMorePartitions=false, updatePartitionCounts=null]
```

6.1.13 Downloading Streaming Data

Context

To download streaming data, you need to determine the position where the data is obtained from the partition, that is, to obtain the cursor. After the start position is determined, the data is obtained cyclically.

There are five cursor types available:

- AT_SEQUENCE_NUMBER
- AFTER_SEQUENCE_NUMBER
- TRIM_HORIZON
- LATEST
- AT_TIMESTAMP

To better understand the cursor type, you need to understand the following basic concepts:

- A sequence number (SN) is the unique identifier of each record. DIS automatically allocates an SN when a data producer calls the PutRecords operation to add data to the DIS stream. SN of the same partition key usually changes with time. A longer interval between PutRecords requests results in a larger sequence number.
- SN of each partition increases from 0. Each data record corresponds to a unique SN. As a lifecycle ends, the SN expires. For example, after a data record is uploaded to a new partition and its SN starts from **0**. After 100 data records are uploaded, the SN of the last data record is **99**. When the lifecycle ends, SNs 0 to 99 become unavailable.
- The SN range of a partition can be obtained by calling the **describeStream** API for querying stream details. **sequenceNumberRange** indicates the data validity range. The first value is the SN of the earliest data, the last value is the SN of the next uploaded data, and the SN of the latest data is one less than the last value.

  For example, [100, 200] indicates that a total of 200 data records have been uploaded to the partition, data records 0 to 99 have expired, the earliest valid data record is 100, the latest data record is 199, and the SN of the next data record to be uploaded is 200.

Scenario Description

The following table describes the application scenarios of the five cursor types:

Table 6-3 Scenario description

Cursor Type	Description	Application Scenario	Remarks
AT_SEQUENCE_NUMBER	Data is read from the position denoted by a specific SN. The SN is defined by **starting-sequence-number** in the demo. This is the default cursor type.	This type of cursor is applicable to the scenario where the start SN has been specified.	It is closely related to sequenceNumber and sequenceNumber Range[A and B] of the partition data. The specified SN must meet the following condition: A<=sequenceNumber<=B
AFTER_SEQUENCE_NUMBER	Data is read from the position right after the position denoted by a specific SN. The SN is defined by **starting-sequence-number**.	This type of cursor is applicable to the scenario where the last consumption position is saved. For example, each consumption record is saved to a file or checkpoint. If the program is restarted, data will be restored from the position right after the save position. Comparatively, **AT_SEQUENCE_NUMBER** will restore from the save position, leading to a data duplicate.	It is closely related to sequenceNumber and sequenceNumber Range[A and B] of the partition data. The specified SN must meet the following condition: (A-1)<=sequence Number<=(B-1)

Cursor Type	Description	Application Scenario	Remarks
TRIM_HORIZON	Data is read from the earliest data record in the partition. For example, if sequenceNumber Range [100, 200] is applied, data consumption starts from record 100.	This type of cursor is applicable to scenarios where the consumption position is unknown and all valid data in the partition will be consumed.	None.
LATEST	Data is read from the position just after the most recent record in the partition. That is, the system does not read the existing data in the partition but starts from the data to be uploaded. For example, if sequenceNumber Range [100, 200] is applied, data consumption starts from record 200. If no data is uploaded, the obtained data is empty. If the data is uploaded, record 200, 201, 202... will be obtained.	This type of cursor is applicable to the scenario where the consumption position is unknown, so the existing data in the partition is discarded and consumption position starts from the newly uploaded data.	None.

Cursor Type	Description	Application Scenario	Remarks
AT_TIMESTAMP	Data is read from the position denoted by a specific timestamp. A 13-bit timestamp is required when the cursor is obtained. For example, if 1541742263206 is specified, data uploaded from 2018-11-09 13:44:23 is read.	This type of cursor is applicable to the scenario where the consumption position is unknown, but the user wants to consume data from a specific time or from the end time of the last consumption.	● If the upload time of the earliest data record is C, timestamp ≥ C. ● If the timestamp is greater than the timestamp of the latest data or the future time, the system reads from the data right after the latest data.

Sample Code

Use the initialized client instance to obtain data through the DIS stream.

- When the cursor type is set to **AT_SEQUENCE_NUMBER** or **AFTER_SEQUENCE_NUMBER**, the sample code is as follows:

```
//Initialize the DIS SDK client instance.
DIS dic = DISClientBuilder.standard()
            .withEndpoint("xxxx")
            .withAk("xxxx")
            .withSk("xxxx")
            .withProjectId("xxxx")
            .withRegion("xxxx")
            .build();
//Configure the stream name.
String streamName = "streamName";
// Configure the ID of the partition for data download.
String partitionId = "shardId-0000000000";
//Configure the SN for data download.
String startingSequenceNumber = "0";
//Configure the data download mode.
//AT_SEQUENCE_NUMBER: Data is read from the position denoted by a specific SN. The SN is defined by
starting-sequence-number in the demo.
// AFTER_SEQUENCE_NUMBER: Data is read from the position right after the position denoted by a specific
SN. The SN is defined by starting-sequence-number.
String cursorType = PartitionCursorTypeEnum.AT_SEQUENCE_NUMBER.name();

try
{
//Obtain the data cursor.
      GetPartitionCursorRequest request = new GetPartitionCursorRequest();
      request.setStreamName(streamName);
      request.setPartitionId(partitionId);
      request.setCursorType(cursorType);
      request.setStartingSequenceNumber(startingSequenceNumber);
      GetPartitionCursorResult response = dic.getPartitionCursor(request);
      String cursor = response.getPartitionCursor();
      LOGGER.info("Get stream {}[partitionId={}] cursor success : {}", streamName, partitionId, cursor);
      GetRecordsRequest recordsRequest = new GetRecordsRequest();
      GetRecordsResult recordResponse = null;
      while (true)
            {
                recordsRequest.setPartitionCursor(cursor);
                recordResponse = dic.getRecords(recordsRequest);
//Obtain the next-batch of data cursor.
                cursor = recordResponse.getNextPartitionCursor();

                for (Record record : recordResponse.getRecords())
                {
                    LOGGER.info("Get record [{}], partitionKey [{}], sequenceNumber [{}].",
                        new String(record.getData().array()),
                        record.getPartitionKey(),
                        record.getSequenceNumber());
                }


            }
        }
    catch (DISClientException e)
        {
            LOGGER.error("Failed to get a normal response, please check params and retry. Error message [{}]",
                e.getMessage(),
                e);
        }
        catch (Exception e)
        {
            LOGGER.error(e.getMessage(), e);
        }
    }
}
```

- When the cursor type is set to **TRIM_HORIZON** or **LATEST**, the sample code example is as follows:

```
//Initialize the DIS SDK client instance.
DIS dic = DISClientBuilder.standard()
            .withEndpoint("xxxx")
            .withAk("xxxx")
            .withSk("xxxx")
            .withProjectId("xxxx")
            .withRegion("xxxx")
            .build();
//Configure the stream name.
String streamName = "streamName";
//Configure the partition ID.
String partitionId = "shardId-0000000000";
//Configure the SN.
//String startingSequenceNumber = "0";
//Configure the cursor type.
//TRIM_HORIZON: Data is read from the earliest data record in the partition.
//LATEST: Data is read just after the most recent record in the partition. This setting ensures that you
always read the most recent data in the partition.
String cursorType = PartitionCursorTypeEnum.TRIM_HORIZON.name();

try
{
//Obtain the data cursor.
      GetPartitionCursorRequest request = new GetPartitionCursorRequest();
      request.setStreamName(streamName);
      request.setPartitionId(partitionId);
      request.setCursorType(cursorType);
      //request.setStartingSequenceNumber(startingSequenceNumber);
      GetPartitionCursorResult response = dic.getPartitionCursor(request);
      String cursor = response.getPartitionCursor();
      LOGGER.info("Get stream {}[partitionId={}] cursor success : {}", streamName, partitionId, cursor);
      GetRecordsRequest recordsRequest = new GetRecordsRequest();
      GetRecordsResult recordResponse = null;
      while (true)
       {
```

```
                recordsRequest.setPartitionCursor(cursor);
                recordsRequest.setLimit(limit);
                recordResponse = dic.getRecords(recordsRequest);
//Obtain the next-batch of data cursor.
                cursor = recordResponse.getNextPartitionCursor();

                for (Record record : recordResponse.getRecords())
                {
                    LOGGER.info("Get record [{}], partitionKey [{}], sequenceNumber [{}].",
                        new String(record.getData().array()),
                        record.getPartitionKey(),
                        record.getSequenceNumber());
                }

                if (recordResponse.getRecords().size() == 0)
                {
                    Thread.sleep(1000);
                }
            }
        }
        catch (DISClientException e)
        {
            LOGGER.error("Failed to get a normal response, please check params and retry. Error message [{}]",
                e.getMessage(),
                e);
        }
        catch (Exception e)
        {
            LOGGER.error(e.getMessage(), e);
        }
    }
}
```

- When the cursor type is set to **AT_TIMESTAMP**, the sample code example is as follows:

```
//Initialize the DIS SDK client instance.
DIS dic = DISClientBuilder.standard()
            .withEndpoint("xxxx")
            .withAk("xxxx")
            .withSk("xxxx")
            .withProjectId("xxxx")
            .withRegion("xxxx")
            .build();
//Configure the stream name.
String streamName = "streamName";
//Configure the partition ID.
String partitionId = "shardId-0000000000";
//Configure the SN.
//String startingSequenceNumber = "0";
//Configure the timestamp.
long timestamp = 1542960693804L;
//Configure the cursor type.
//AT_TIMESTAMP: Data is read from the position denoted by a specific timestamp.
String cursorType = PartitionCursorTypeEnum.AT_TIMESTAMP.name();

try
{
//Obtain the data cursor.
      GetPartitionCursorRequest request = new GetPartitionCursorRequest();
      request.setStreamName(streamName);
      request.setPartitionId(partitionId);
      request.setCursorType(cursorType);
      //request.setStartingSequenceNumber(startingSequenceNumber);
      request.setTimestamp(timestamp);
      GetPartitionCursorResult response = dic.getPartitionCursor(request);
      String cursor = response.getPartitionCursor();
      LOGGER.info("Get stream {}[partitionId={}] cursor success : {}", streamName, partitionId, cursor);

            GetRecordsRequest recordsRequest = new GetRecordsRequest();
            GetRecordsResult recordResponse = null;
            while (true)
            {
                recordsRequest.setPartitionCursor(cursor);
                recordsRequest.setLimit(limit);
                recordResponse = dic.getRecords(recordsRequest);
//Obtain the next-batch of data cursor.
                cursor = recordResponse.getNextPartitionCursor();

                for (Record record : recordResponse.getRecords())
                {
                    LOGGER.info("Get record [{}], partitionKey [{}], sequenceNumber [{}].",
                        new String(record.getData().array()),
                        record.getPartitionKey(),
                        record.getSequenceNumber());
                }

                if (recordResponse.getRecords().size() == 0)
                {
                    Thread.sleep(1000);
                }
            }
        }
        catch (DISClientException e)
        {
            LOGGER.error("Failed to get a normal response, please check params and retry. Error message [{}]",
                e.getMessage(),
                e);
        }
        catch (Exception e)
        {
            LOGGER.error(e.getMessage(), e);
        }
    }
}
```

Parameters

Table 6-4 Parameter description

Parameter	Type	Description
partitionId	String	Partition ID. **NOTE** Define the return information fields on the console, such as **partitionId [shardId-0000000000]**, based on the execution results obtained from Uploading Streaming Data.
startingSequenceNumber	String	Sequence number of an individual data record. Each data record has a sequence number that is unique within its partition. The sequence number is assigned by DIS when a data producer calls PutRecords to add data to a DIS stream. Sequence numbers for the same partition key generally increase over time; the longer the time period between write requests (PutRecords requests), the larger the sequence numbers become. **NOTE** Define the return information fields on the console, such as **sequenceNumber [1]**, based on the execution results obtained from Uploading Streaming Data.

Parameter	Type	Description
cursorType	String	Cursor type. ● **AT_SEQUENCE_NUMBER**: Data is read from the position denoted by a specific SN. The SN is defined by **starting-sequence-number** in the demo. This is the default cursor type. ● **AFTER_SEQUENCE_NUMBER**: Data is read from the position right after the position denoted by a specific sequence number. The SN is defined by **starting-sequence-number** in the demo. ● **TRIM_HORIZON**: Data is read from the earliest data record in the partition. For example, a tenant uses a DIS stream to upload three pieces of data A1, A2, and A3. N days later, A1 has expired and A2 and A3 are still in the validity period. In this case, if the tenant sets the cursor type to **TRIM_HORIZON**, the system downloads data from A2. ● **LATEST**: Data is read just after the most recent record in the partition. This setting ensures that you always read the most recent data in the partition. ● **AT_TIMESTAMP**: Data is read from the position denoted by a specific timestamp.

Running the Program

Right-click the program and choose **Run As > 1 Java Application** from the shortcut menu. If the program is run successfully, you can view the information similar to the following on the console:

```
14:55:42.954 [main] INFOcom.bigdata.dis.sdk.DISConfig - get from classLoader
14:55:44.103 [main] INFOcom.bigdata.dis.sdk.util.config.ConfigurationUtils - get from classLoader
14:55:44.105 [main] INFOcom.bigdata.dis.sdk.util.config.ConfigurationUtils - propertyMapFromFile size : 2
14:55:45.235 [main] INFOcom.bigdata.dis.sdk.demo.ConsumerDemo - Get stream
streamName[partitionId=0] cursor success :
eyJnZXRJdGVyYXRvclBhcmFtIjp7InN0cmVhbS1uYW1lIjoiZGlzLTEzbW9uZXkiLCJwYXJ0aXRpb24taWQiOiIwIiwiY
3Vyc29yLXR5cGUiOiJBVF9TRVFVRU5DRV9OVU1CRVIiLCJzdGFydGluZy1zZXF1ZW5jZS1udW1iZXIiOiIxMDY4O
TcyIn0sImdlbmVyYXRlVGltZXN0YW1wIjoxNTEzNjY2NjMxMTYxfQ
14:55:45.305 [main] INFOcom.bigdata.dis.sdk.demo.ConsumerDemo - Get Record [hello world.],
partitionKey [964885], sequenceNumber [0].
14:55:45.305 [main] INFOcom.bigdata.dis.sdk.demo.ConsumerDemo - Get Record [hello world.],
partitionKey [910960], sequenceNumber [1].
14:55:46.359 [main] INFOcom.bigdata.dis.sdk.demo.ConsumerDemo - Get Record [hello world.],
partitionKey [528377], sequenceNumber [2].
```

Table 6-5 Parameter description

Parameter	Type	Description
partition_key	String	Partition key set when data is being uploaded. **NOTE** If **partition_key** is specified when data is uploaded, **partition_key** is returned when data is downloaded. If **partition_id** instead of **partition_key** is specified when data is uploaded, no **partition_key** is returned.
startingSequenceNumber	String	Sequence number of an individual data record. Each data record has a sequence number that is unique within its partition. The sequence number is assigned by DIS when a data producer calls PutRecords to add data to a DIS stream. Sequence numbers for the same partition key generally increase over time; the longer the time period between write requests (PutRecords requests), the larger the sequence numbers become.

6.1.14 Uploading Streaming Data

Sample Code

Use the initialized client instance to upload your streaming data to DIS through a DIS stream.

The code for uploading streaming data is as follows:

```
//Initialize the DIS SDK client instance. For details about endpoints, AKs, SKs, regions, and project IDs, see
the following:


DIS dic = DISClientBuilder.standard()
            .withEndpoint("xxxx")
            .withAk("xxxx")
            .withSk("xxxx")
            .withProjectId("xxxx")
            .withRegion("xxxx")
            .build();

//Configure the stream name.
String streamName = "xxxx";
//Configure the data to be uploaded.
String message = "hello world.";
 PutRecordsRequest putRecordsRequest = new PutRecordsRequest();
        putRecordsRequest.setStreamName(streamName);
        List<PutRecordsRequestEntry> putRecordsRequestEntryList = new
ArrayList<PutRecordsRequestEntry>();
        ByteBuffer buffer = ByteBuffer.wrap(message.getBytes());
        for (int i = 0; i < 3; i++)
        {
            PutRecordsRequestEntry putRecordsRequestEntry = new PutRecordsRequestEntry();
            putRecordsRequestEntry.setData(buffer);

putRecordsRequestEntry.setPartitionKey(String.valueOf(ThreadLocalRandom.current().nextInt(1000000)));
            putRecordsRequestEntryList.add(putRecordsRequestEntry);
        }
        putRecordsRequest.setRecords(putRecordsRequestEntryList);
```

```
    LOGGER.info("========== BEGIN PUT ============");

    PutRecordsResult putRecordsResult = null;
    try
    {
        putRecordsResult = dic.putRecords(putRecordsRequest);
    }
    catch (DISClientException e)
    {
        LOGGER.error("Failed to get a normal response, please check params and retry. Error message [{}]",
            e.getMessage(),
            e);
    }
    catch (Exception e)
    {
        LOGGER.error(e.getMessage(), e);
    }
```

Running the Program

Right-click the program and choose **Run As > 1 Java Application** from the shortcut menu. If the program is run successfully, you can view the information similar to the following on the console:

```
15:19:29.298 [main] INFO  com.bigdata.dis.sdk.demo.ProducerDemo - ========== BEGIN PUT
============
15:19:30.992 [main] INFO  com.bigdata.dis.sdk.demo.ProducerDemo - Put 3 records[3 successful / 0 failed].
15:19:30.992 [main] INFO  com.bigdata.dis.sdk.demo.ProducerDemo - [hello world.] put success, partitionId
[shardId-0000000000], partitionKey [261045], sequenceNumber [1]
15:19:30.992 [main] INFO  com.bigdata.dis.sdk.demo.ProducerDemo - [hello world.] put success, partitionId
[shardId-0000000000], partitionKey [958815], sequenceNumber [2]
15:19:30.992 [main] INFO  com.bigdata.dis.sdk.demo.ProducerDemo - [hello world.] put success, partitionId
[shardId-0000000000], partitionKey [416421], sequenceNumber [3]
15:19:30.992 [main] INFO  com.bigdata.dis.sdk.demo.ProducerDemo - ========== END PUT ============
```

6.1.15 Obtaining the Data Cursor

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

Use the DIS SDK to obtain the information about the data cursor.

```
//Configure the stream name.
String streamName = "myStream";
// Configure the ID of the partition for data download.
String partitionId = "0";
//Configure the sequence number for data download.
String startingSequenceNumber = "0";
//Configure the data download mode.
String cursorType = PartitionCursorTypeEnum.AT_SEQUENCE_NUMBER.name();

GetPartitionCursorRequest request = new GetPartitionCursorRequest();
request.setStreamName(streamName);
request.setPartitionId(partitionId);
request.setStartingSequenceNumber(startingSequenceNumber);
request.setCursorType(cursorType);
GetPartitionCursorResult response = dic.getPartitionCursor(request);
String cursor = response.getPartitionCursor();
```

6.1.16 Creating an Application

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

When using DIS SDK to create an application, you need to specify an application name.

```
//Specify the application name.
String appName = "myApp";
```

After the application name is specified, create the application by calling createApp.

```
dic.createApp(myApp);
```

6.1.17 Deleting an Application

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

When using DIS SDK to delete an application, you need to specify an application name.

```
//Specify the application to be deleted.
String appName = "myApp";
```

After the application name is specified, delete the application by calling deleteApp.

```
dic.deleteApp(myApp);
```

6.1.18 Adding a Checkpoint

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

When using the DIS SDK to create a checkpoint, you need to specify the stream name, application name, partition ID, sequence number, and checkpoint type.

```
//Specify the stream name.
String streamName = "myStream";
//Specify the application name.
String appName = "myApp";
CommitCheckpointRequest commitCheckpointRequest = new CommitCheckpointRequest();
commitCheckpointRequest.setStreamName(streamName);
commitCheckpointRequest.setAppName(appName);
//Specify the sequence number to be submitted.
commitCheckpointRequest.setSequenceNumber("100");
//Specify the partition No.
commitCheckpointRequest.setPartitionId("0");
//Specify the checkpoint type.
commitCheckpointRequest.setCheckpointType(CheckpointTypeEnum.LAST_READ.name());
```

After configuring **CommitCheckpointRequest**, add a checkpoint by calling commitCheckpoint.

```
dic.commitCheckpoint(commitCheckpointRequest);
```

6.1.19 Querying a Checkpoint

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

```
GetCheckpointRequest getCheckpointRequest = new GetCheckpointRequest();
//Specify the application name. (Note: setAppId is used in 1.3.0 and earlier versions, and setAppName is
used in 1.3.1 and later versions.)
getCheckpointRequest.setAppName(appName);
getCheckpointRequest.setStreamName(streamName);
//Specify the partition No.
getCheckpointRequest.setPartitionId("0");
```

```
//Specify the checkpoint type.
getCheckpointRequest.setCheckpointType(CheckpointTypeEnum.LAST_READ.name());
System.out.println("getCheckpoint: " + JsonUtils.objToJson(dic.getCheckpoint(getCheckpointRequest)));
```

The returned checkpoint details are as follows:

```
getCheckpoint:
{"sequence_number": "10", "metadata": "metadata"}
```

6.1.20 Changing Partition Quantity

Initialize a DIS SDK client instance named **dic**. For details, see Initializing a DIS SDK Client Instance.

```
//Specify the number of target partitions.
int targetPartitionCount = 2;

UpdatePartitionCountRequest update = new UpdatePartitionCountRequest();
update.setStreamName(streamName);
update.setTargetPartitionCount(targetPartitionCount);
try
{
    UpdatePartitionCountResult updatePartitionCountResult = dic.updatePartitionCount(update);
    LOGGER.info("Success to update partition count, {}", updatePartitionCountResult);
}
catch (Exception e)
{
    LOGGER.error("Failed to update partition count", e);
}
```

If the number of partitions is changed successfully, information similar to the following is returned:

```
Success to update partition count, UpdatePartitionCountResult [currentPartitionCount=2,
streamName=mystream, targetPartitionCount=2]
```

6.2 Using Kafka Adapter to Upload and Download Data

6.2.1 Kafka Adapter Overview

dis-kafka-adapter is a software development kit (SDK) provided by DIS.

Users who originally use Kafka Client can use dis-kafka-adapter instead to upload data to DIS in the similar way.

6.2.2 Preparations

Configuring the pom.xml File

If a maven project exists, use the following dependency in **pom.xml**:

```
<dependency>
    <groupId>com.huaweicloud.dis</groupId>
    <artifactId>huaweicloud-dis-kafka-adapter</artifactId>
    <version>1.2.18</version>
</dependency>
```

Using the DIS Sample Project

Download the package from **https://dis-publish.obs-website.cn-north-1.myhuaweicloud.com/**.

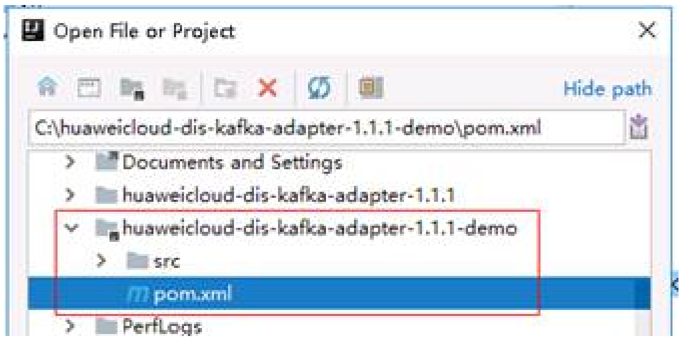
The .zip package contains two directories.

- The **huaweicloud-dis-kafka-adapter-***X.X.X* directory contains all JAR packages. If a non-maven project is used, import all JAR packages in the lib directory to the environment.
- **huaweicloud-dis-kafka-adapter-**-*X.X.X***-demo** is a sample project and is compiled using maven.

You can use IntelliJ IDEA to import the sample project as follows:

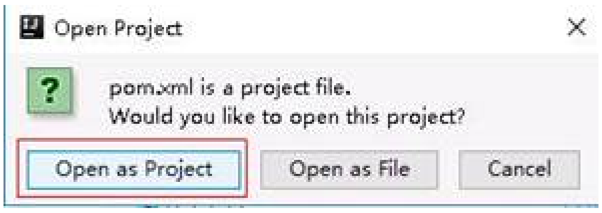
Step 1 Run IntelliJ IDEA and choose **File** > **Open**.

In the displayed dialog box, expand **huaweicloud-dis-kafka-adapter-***X.X.X***-demo** and double-click **pom.xml**.



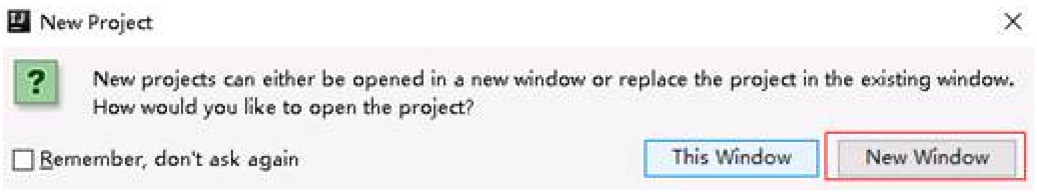
C:\huaweicloud-dis-kafka-adapter-1.1.1-demo\pom.xml
Documents and Settings
huaweicloud-dis-kafka-adapter-1.1.1
huaweicloud-dis-kafka-adapter-1.1.1-demo
src
pom.xml
PerfLogs

Step 2 When the following dialog box is displayed, select **Open as Project**.



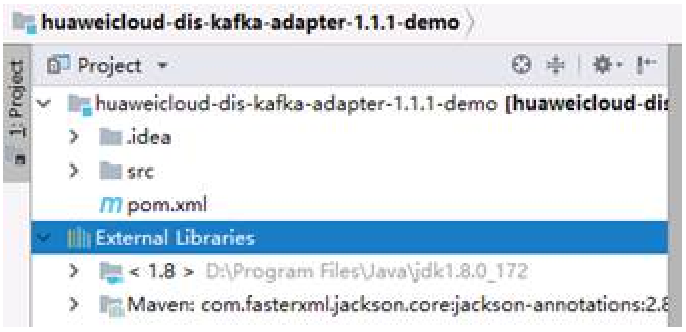
pom.xml is a project file.
Would you like to open this project?

Step 3 Click **New Window** to open the project in a new window.



New projects can either be opened in a new window or replace the project in the existing window.
How would you like to open the project?

Step 4 Wait when IntelliJ IDEA is building the project. After the project is built, the directory and files are displayed.



```
huaweicloud-dis-kafka-adapter-1.1.1-demo
Project
v huaweicloud-dis-kafka-adapter-1.1.1-demo [huaweicloud-dis
  > .idea
  > src
    pom.xml
v External Libraries
  > < 1.8 > D:\Program Files\Java\jdk1.8.0_172
  > Maven: com.fasterxml.jackson.core:jackson-annotations:2.8
```

----End

Checking Authentication Information

- AK/SK file

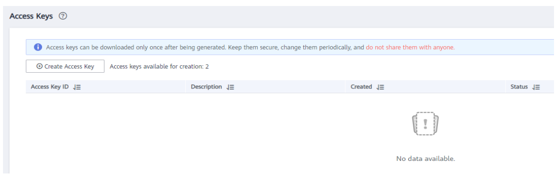
  Access Key ID/Secret Access Key (AK/SK) files are created by the Identity and Access Management (IAM) service to authenticate calls to application programming interfaces (APIs) on the public cloud.

  To obtain an access key, perform the following steps:

  a. Log in to the management console, move the cursor to the username in the upper right corner, and select **My Credentials** from the drop-down list.

  b. On the **My Credentials** page, choose **Access Keys**, and click **Create Access Key**. See **Figure 6-3**.

  Figure 6-3 Clicking Create Access Key

  Access Keys

  Access keys can be downloaded only once after being generated. Keep them secure, change them periodically, and do not share them with anyone.

  Create Access Key   Access keys available for creation: 2

Access Key ID	Description	Created	Status

  No data available.

  c. Click **OK** and save the access key file as prompted. The access key file will be saved to your browser's configured download location. Open the **credentials.csv** file to view **Access Key Id** and **Secret Access Key**.

  NOTE

  - Only two access keys can be added for each user.
  - To ensure access key security, the access key is automatically downloaded only when it is generated for the first time and cannot be obtained from the management console later. Keep them properly.

- Project ID

  A project is a group of tenant resources, and an account ID corresponds to the current account. The IAM ID corresponds to the current user. You can view the project IDs, account IDs, and user IDs in different regions on the corresponding pages.

  a. Register with and log in to the management console.

  b. Hover the cursor on the username in the upper right corner and select **My Credentials** from the drop-down list.

c. On the **API Credentials** page, obtain the account name, account ID, IAM username, and IAM user ID, and obtain the project and its ID from the project list.

6.2.3 Uploading Data

Sample Code

For details about how to obtain the AK, SK, and project ID, see **Checking Authentication Information**.

```
package com.huaweicloud.dis.demo.adapter;
import com.huaweicloud.dis.DISConfig;
import com.huaweicloud.dis.adapter.kafka.clients.producer.*;
import com.huaweicloud.dis.adapter.kafka.common.serialization.StringSerializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.Future;
import java.util.concurrent.ThreadLocalRandom;

public class DISKafkaProducerDemo
{
    private static final Logger LOGGER = LoggerFactory.getLogger(DISKafkaProducerDemo.class);


    public static void main(String[] args)
    {

        // There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt
the AK/SK in the configuration file or environment variables for storage;
       // In this example, the AK and SK stored in the environment variables are used for identity
authentication. Before running this example, configure environment variables HUAWEICLOUD_SDK_AK and
HUAWEICLOUD_SDK_SK in the local environment.
       String ak = System.getenv("HUAWEICLOUD_SDK_AK");
       String sk = System.getenv("HUAWEICLOUD_SDK_SK");
       // YOU ProjectId
       String projectId = "YOU_PROJECT_ID";
      // YOU DIS Stream
       String streamName = "YOU_STREAM_NAME";
      // Consumption group ID, which is used to record the offset.
      String groupId = "YOU_GROUP_ID";
     // DIS region
      String region = "your region";
```

```
        Properties props = new Properties();
        props.setProperty(DISConfig.PROPERTY_AK, ak);
        props.setProperty(DISConfig.PROPERTY_SK, sk);
        props.setProperty(DISConfig.PROPERTY_PROJECT_ID, projectId);
        props.setProperty(DISConfig.PROPERTY_REGION_ID, region);
        props.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        props.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, 
StringSerializer.class.getName());

//By default, the domain name is automatically used for access instead of configuring an endpoint. If an 
endpoint is required, remove the following comments and set the endpoint:
        // props.setProperty(DISConfig.PROPERTY_ENDPOINT, "https://dis-${region}.myhuaweicloud.com");

        // Create dis producer
        Producer<String, String> producer = new DISKafkaProducer<>(props);

                    //Send data synchronously.
        synchronousSendDemo(producer, streamName);

                //Send data asynchronously.
        asynchronousSendDemo(producer, streamName);

// Disable the producer to prevent resource leakage.
        producer.close();
    }

    public static void synchronousSendDemo(Producer<String, String> producer, String streamName)
    {
        LOGGER.info("===== synchronous send =====");
        for (int i = 0; i < 5; i++)
        {
 //If the key is set to Random or Null, data is evenly distributed to all partitions.
            String key = String.valueOf(ThreadLocalRandom.current().nextInt(1000000));
            String value = "Hello world[sync]. " + i;

            Future<RecordMetadata> future = producer.send(new ProducerRecord<>(streamName, key, value));

            try
            {
 //Calling future.get will block waiting until sending is complete.
                RecordMetadata recordMetadata = future.get();
//Data is successfully sent.
                LOGGER.info("Success to send [{}], Partition [{}], Offset [{}].",
                        value, recordMetadata.partition(), recordMetadata.offset());
            }
            catch (Exception e)
            {
//Data failed to be sent.
                LOGGER.error("Failed to send [{}], Error [{}]", value, e.getMessage(), e);
            }
        }
    }

    public static void asynchronousSendDemo(Producer<String, String> producer, String streamName)
    {
        LOGGER.info("===== asynchronous send =====");
        int totalSendCount = 5;
        CountDownLatch countDownLatch = new CountDownLatch(totalSendCount);
        for (int i = 0; i < totalSendCount; i++)
        {
 //If the key is set to Random or Null, data is evenly distributed to all partitions.
            String key = String.valueOf(ThreadLocalRandom.current().nextInt(1000000));
            String value = "Hello world[async]. " + i;

            try
            {
//Data is sent in callback mode and is not blocked.
                producer.send(new ProducerRecord<>(streamName, key, value), new Callback()
                {
                    @Override
                    public void onCompletion(RecordMetadata recordMetadata, Exception e)
                    {
                        countDownLatch.countDown();
                        if (e == null)
                        {
//Data is successfully sent.
                            LOGGER.info("Success to send [{}], Partition [{}], Offset [{}].",
                                    value, recordMetadata.partition(), recordMetadata.offset());
                        }
                        else
                        {
//Data failed to be sent.
                            LOGGER.error("Failed to send [{}], Error [{}]", value, e.getMessage(), e);
                        }
                    }
                });
            }
            catch (Exception e)
            {
                countDownLatch.countDown();
                LOGGER.error(e.getMessage(), e);
            }
        }

        try
        {
// Wait until all data is sent.
            countDownLatch.await();
        }
        catch (InterruptedException e)
        {
            LOGGER.error(e.getMessage(), e);
        }
    }
}
```

After running the preceding program, if the data is successfully sent, the following log is generated:

```
09:32:52.001 INFO  c.h.d.d.a.DISKafkaProducerDemo - ===== synchronous send =====
09:32:53.523 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[sync]. 0], Partition [0], Offset [114].
09:32:53.706 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[sync]. 1], Partition [0], Offset [115].
09:32:53.956 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[sync]. 2], Partition [0], Offset [116].
09:32:54.160 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[sync]. 3], Partition [0], Offset [117].
09:32:54.450 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[sync]. 4], Partition [0], Offset [118].
09:32:54.450 INFO  c.h.d.d.a.DISKafkaProducerDemo - ===== asynchronous send =====
09:32:54.673 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[async]. 0], Partition [0], Offset [119].
09:32:54.674 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[async]. 1], Partition [0], Offset [120].
09:32:54.674 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[async]. 2], Partition [0], Offset [121].
09:32:54.674 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[async]. 3], Partition [0], Offset [122].
09:32:54.674 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[async]. 4], Partition [0], Offset [123].
```

Adaptation to the Native KafkaProducer API

DISKafkaProducer is implemented in a different way from KafkaProducer. The DISKafkaProducer client and server are implemented through the REST API, whereas KafkaProducer is implemented based on TCP. Their API compatibility differences are as follows:

Table 6-6 Adaptation description

Native KafkaProducer	Type	DISKafkaProducer	Description
Future<RecordMetadata> send(ProducerRecord<K, V> record)	API	Supported	Send a single data record.
Future<RecordMetadata> send(ProducerRecord<K, V> record, Callback callback)	API	Supported	Send a single data record and set the callback processing function.
void close()	API	Supported	Disable Producer.
void close(long timeout, TimeUnit timeUnit)	API	Supported	Disable Producer and set the timeout period.
List<PartitionInfo> partitionsFor(String topic)	API	Supported	Obtain the partition information of the stream.
void flush(long timeout, TimeUnit unit)	API	Not supported	Forcibly send the current cached data.
Map<MetricName, ? extends Metric> metrics()	API	Not supported	Obtain statistics.
key.serializer	Parameter	Supported	The meaning of this parameter is the same as that in Kafka. The default value is **StringSerializer**. In Kafka, this parameter has no default value, and you must configure a value for it.

Native KafkaProducer	Type	DISKafkaProducer	Description
value.serializer	Parameter	Supported	The meaning of this parameter is the same as that in Kafka. The default value is **StringSerializer**. In Kafka, this parameter has no default value, and you must configure a value for it.
linger.ms	Parameter	Supported	The meaning of this parameter is the same as that in Kafka. The default value is **50**. In Kafka, the default value is **0**. This parameter is configured for improving the upload efficiency of the REST API.
batch.size	Parameter	Supported	The meaning of this parameter is the same as that in Kafka. The default value is 1 MB. In Kafka, the default value is 16 KB. This parameter is configured for matching the flow control.
buffer.memory	Parameter	Supported	Same as the default setting in Kafka, which is 32 MB.

Native KafkaProducer	Type	DISKafkaProducer	Description
max.in.flight.requests.per.connection	Parameter	Supported	Limit the maximum number of not-responded requests that can be sent by a client on a single connection. The default value is **100**. In Kafka, the default value is 5. This parameter is configured for improving the sending performance. However, the data sequence may be inconsistent. You are advised to set this parameter to 1 to ensure data sequence.
block.on.buffer.full	Parameter	Supported	Same as the default setting in Kafka, which is **false**. ● true: When the sending buffer is full, sending is blocked and does not time out. ● false: After the sending buffer is full, data is blocked based on max.block.ms. If the time is exceeded, an exception is issued.

Native KafkaProducer	Type	DISKafkaProducer	Description
max.block.ms	Parameter	Supported	Same as the default setting in Kafka, which is **60000**. When the sending buffer is full and **block.on.buffer.full** is **false**, control the block time (ms) of send ().
retries	Parameter	Supported, but the parameter name is changed to **exception.retries**.	The default value in Kafka is **0**, and the default value in DIS is **8**. Number of retry attempts that are made when the network or server is abnormal.
Others	Parameter	Not supported	-

6.2.4 Consumption Modes

Similar to Kafka, dis kafka adapter supports three consumption modes.

assign Mode

Users specify the partitions to be consumed by the consumer instance are manually. In this case, the group management mechanism is not used. That is, when the number of consumers in the group changes or the stream scaling is performed, the partitions will not be reallocated. A code example is provided as follows:

```
package com.huaweicloud.dis.demo.adapter;

import com.huaweicloud.dis.DISConfig;
import com.huaweicloud.dis.adapter.kafka.clients.consumer.*;
import com.huaweicloud.dis.adapter.kafka.common.PartitionInfo;
import com.huaweicloud.dis.adapter.kafka.common.TopicPartition;
import com.huaweicloud.dis.adapter.kafka.common.serialization.StringDeserializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Properties;

public class DISKafkaConsumerAssignDemo
```

```
{
    private static final Logger LOGGER = LoggerFactory.getLogger(DISKafkaConsumerAssignDemo.class);

    public static void main(String[] args)
  {

        // There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt
the AK/SK in the configuration file or environment variables for storage;
       // In this example, the AK and SK stored in the environment variables are used for identity
authentication. Before running this example, configure environment variables HUAWEICLOUD_SDK_AK and
HUAWEICLOUD_SDK_SK in the local environment.
       String ak = System.getenv("HUAWEICLOUD_SDK_AK");
       String sk = System.getenv("HUAWEICLOUD_SDK_SK");
       // YOU ProjectId
       String projectId = "YOU_PROJECT_ID";
      // YOU DIS Stream
       String streamName = "YOU_STREAM_NAME";
//Consumption group ID, which is used to record the offset.
      String groupId = "YOU_GROUP_ID";
     // DIS region
      String region = "your region";

        Properties props = new Properties();
        props.setProperty(DISConfig.PROPERTY_AK, ak);
        props.setProperty(DISConfig.PROPERTY_SK, sk);
        props.setProperty(DISConfig.PROPERTY_PROJECT_ID, projectId);
        props.setProperty(DISConfig.PROPERTY_REGION_ID, region);
        props.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
        props.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
        props.setProperty(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        props.setProperty(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");
        props.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
OffsetResetStrategy.LATEST.name());


//By default, the domain name is automatically used for access instead of configuring an endpoint. If an
endpoint is required, remove the following comments and set the endpoint:
        // props.setProperty(DISConfig.PROPERTY_ENDPOINT, "https://dis-${region}.myhuaweicloud.com");

        Consumer<String, String> consumer = new DISKafkaConsumer<>(props);
        List<TopicPartition> topicPartitions = new ArrayList<>();
        for (PartitionInfo partitionInfo : consumer.partitionsFor(streamName))
        {
            topicPartitions.add(new TopicPartition(partitionInfo.topic(), partitionInfo.partition()));
        }

//Use the assign mode to specify the partition to be consumed.
        consumer.assign(topicPartitions);

        while (true)
        {
            try
            {
                ConsumerRecords<String, String> records = consumer.poll(Long.MAX_VALUE);

                if (!records.isEmpty())
                {
                    for (TopicPartition partition : records.partitions())
                    {
                        List<ConsumerRecord<String, String>> partitionRecords = records.records(partition);
                        for (ConsumerRecord<String, String> record : partitionRecords)
                        {
                            LOGGER.info("Value [{}], Partition [{}], Offset [{}], Key [{}]",
                                    record.value(), record.partition(), record.offset(), record.key());
                        }
                    }
//Submit the current offset asynchronously after data processing is complete or submit commitSync
synchronously.
                    consumer.commitAsync(new OffsetCommitCallback()
                    {
                        @Override
                        public void onComplete(Map<TopicPartition, OffsetAndMetadata> map, Exception e)
                        {
                            if (e == null)
                            {
                                LOGGER.debug("Success to commit offset [{}]", map);
                            }
                            else
                            {
                                LOGGER.error("Failed to commit offset [{}]", e.getMessage(), e);
                            }
                        }
                    });
                }
            }
            catch (Exception e)
            {
                LOGGER.info(e.getMessage(), e);
            }
        }
    }
}
```

After running the preceding program, if the data is sent to the stream, the following log is generated:

```
09:36:45.071 INFO  c.h.d.a.k.c.DISKafkaConsumer - create DISKafkaConsumer successfully
09:36:49.842 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[sync]. 0], Partition [0],
Offset [134], Key [769066]
09:36:49.963 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[sync]. 1], Partition [0],
Offset [135], Key [700005]
09:36:50.145 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[sync]. 2], Partition [0],
Offset [136], Key [338044]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[sync]. 3], Partition [0],
Offset [137], Key [537495]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[sync]. 4], Partition [0],
Offset [138], Key [980016]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[async]. 0], Partition [0],
Offset [139], Key [182772]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[async]. 1], Partition [0],
Offset [140], Key [830271]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[async]. 2], Partition [0],
Offset [141], Key [927054]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[async]. 3], Partition [0],
Offset [142], Key [268122]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[async]. 4], Partition [0],
Offset [143], Key [992787]
```

subscribe Mode

Users only need to specify stream names without specifying specific partitions. The server will trigger the group management mechanism based on the number of consumers or stream scaling changes to automatically allocate partitions to each consumer. This ensures that a partition is consumed by only one consumer.

A code example is provided as follows:

```
package com.huaweicloud.dis.demo.adapter;

import com.huaweicloud.dis.DISConfig;
import com.huaweicloud.dis.adapter.kafka.clients.consumer.*;
import com.huaweicloud.dis.adapter.kafka.common.TopicPartition;
import com.huaweicloud.dis.adapter.kafka.common.serialization.StringDeserializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.Properties;

public class DISKafkaConsumerSubscribeDemo
{
    private static final Logger LOGGER = LoggerFactory.getLogger(DISKafkaConsumerSubscribeDemo.class);

    public static void main(String[] args)
    {

        // There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt
the AK/SK in the configuration file or environment variables for storage;
       // In this example, the AK and SK stored in the environment variables are used for identity
authentication. Before running this example, configure environment variables HUAWEICLOUD_SDK_AK and
HUAWEICLOUD_SDK_SK in the local environment.
       String ak = System.getenv("HUAWEICLOUD_SDK_AK");
       String sk = System.getenv("HUAWEICLOUD_SDK_SK");
       // YOU ProjectId
       String projectId = "YOU_PROJECT_ID";
      // YOU DIS Stream
       String streamName = "YOU_STREAM_NAME";
//Consumption group ID, which is used to record the offset.
      String groupId = "YOU_GROUP_ID";
     // DIS region
      String region = "your region";

        Properties props = new Properties();
        props.setProperty(DISConfig.PROPERTY_AK, ak);
        props.setProperty(DISConfig.PROPERTY_SK, sk);
        props.setProperty(DISConfig.PROPERTY_PROJECT_ID, projectId);
        props.setProperty(DISConfig.PROPERTY_REGION_ID, region);
        props.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
        props.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
        props.setProperty(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        props.setProperty(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");
        props.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
OffsetResetStrategy.LATEST.name());

//By default, the domain name is automatically used for access instead of configuring an endpoint. If an
endpoint is required, remove the following comments and set the endpoint:
        // props.setProperty(DISConfig.PROPERTY_ENDPOINT, "https://dis-${region}.myhuaweicloud.com");

        Consumer<String, String> consumer = new DISKafkaConsumer<>(props);
      //When using the subscribe mode, specify the name of the stream to be consumed.
        consumer.subscribe(Collections.singleton(streamName));

        while (true)
        {
            try
            {
                ConsumerRecords<String, String> records = consumer.poll(Long.MAX_VALUE);

                if (!records.isEmpty())
                {
                    for (TopicPartition partition : records.partitions())
                    {
                        List<ConsumerRecord<String, String>> partitionRecords = records.records(partition);
                        for (ConsumerRecord<String, String> record : partitionRecords)
                        {
                            LOGGER.info("Value [{}], Partition [{}], Offset [{}], Key [{}]",
                                    record.value(), record.partition(), record.offset(), record.key());
                        }
                    }
//Submit the current offset asynchronously after data processing is complete or submit commitSync
synchronously.
                    consumer.commitAsync(new OffsetCommitCallback()
                    {
                        @Override
                        public void onComplete(Map<TopicPartition, OffsetAndMetadata> map, Exception e)
                        {
                            if (e == null)
                            {
                                LOGGER.debug("Success to commit offset [{}]", map);
                            }
                            else
                            {
                                LOGGER.error("Failed to commit offset [{}]", e.getMessage(), e);
                            }
                        }
                    });
                }
            }
            catch (Exception e)
            {
                LOGGER.info(e.getMessage(), e);
            }
        }
    }
}
```

When the program runs, it sends a heartbeat request (Heartbeat) every 10 seconds and then sends a (JoinGroup) request to join the consumer group. The server starts to allocate partitions to consumers in the consumer group. This process takes about 20s. After the operation is complete, the consumers send synchronization requests (SyncGroup) to obtain allocation results. If the information about heartbeat {"state": "STABLE"} is recorded in the log, it indicates that allocation to the entire consumer group has been completed, data is ready to be consumed.

The key logs in this process are described as follows:

- Heartbeat {"state":"JOINING"}

Heartbeat: indicates a heartbeat request. The heartbeat request is initiated every 10 seconds and is used to keep a connection with the server. If the server does not receive the heartbeat message within 1 minute, it considers that the consumer is offline and the partitions will be reallocated to the consumers in the consumer group. If the heartbeat result is JOINING, the consumer needs to join the consumer group again. If the heartbeat result is STABLE, the consumer group is stable.

- JoinGroup

If the heartbeat result is not STABLE, the consumer sends a joinGroup request to notify the server that it needs to join the consumer group. After receiving the join request from the client, the server will reallocate partitions for the consumer group. In this case, syncDelayedTimeMs is returned, indicating the allocation duration. After allocation is completed, the client sends a synchronization request (SyncGroup) to obtain the allocation result.

- SyncGroup

A SyncGroup request is used to obtain the allocation result. The returned assignment contains the stream name and partition to be consumed.

Run the sample program. After allocation is completed, send data to the stream. The complete log is as follows:

```
09:42:37.296 INFO  c.h.d.a.k.c.DISKafkaConsumer - create DISKafkaConsumer successfully
09:42:37.354 INFO  c.h.d.a.k.c.Coordinator - Heartbeat {"state":"JOINING"}
09:42:37.363 INFO  c.h.d.a.k.c.Coordinator - joinGroupRequest {"groupId":"ding","clientId":"consumer-
c2d43144-0823-4eea-aaa8-7af95c536144","interestedStream":["liuhao12"]}
09:42:37.406 INFO  c.h.d.a.k.c.Coordinator - joinGroupResponse {"state":"OK","syncDelayedTimeMs":21000}
09:42:58.408 INFO  c.h.d.a.k.c.Coordinator - syncGroup {"groupId":"ding","clientId":"consumer-
c2d43144-0823-4eea-aaa8-7af95c536144","generation":-1}
09:42:58.451 INFO  c.h.d.a.k.c.Coordinator - syncGroup {"state":"OK","generation":33,"assignment":{"dis-
test":[0]}}
09:42:58.488 INFO  c.h.d.a.k.c.Coordinator - Heartbeat {"state":"STABLE"}
09:43:08.960 INFO  c.h.d.a.k.c.Coordinator - Heartbeat {"state":"STABLE"}
09:46:56.227 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[sync]. 0], Partition [0],
Offset [144], Key [799704]
09:46:56.327 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[sync]. 1], Partition [0],
Offset [145], Key [683757]
09:46:56.449 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[sync]. 2], Partition [0],
Offset [146], Key [439062]
09:46:56.535 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[sync]. 3], Partition [0],
Offset [147], Key [374939]
09:46:56.654 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[sync]. 4], Partition [0],
Offset [148], Key [321528]
09:46:56.749 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[async]. 0], Partition
[0], Offset [149], Key [964841]
09:46:56.749 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[async]. 1], Partition
[0], Offset [150], Key [520262]
09:46:56.749 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[async]. 2], Partition
[0], Offset [151], Key [619119]
09:46:56.749 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[async]. 3], Partition
[0], Offset [152], Key [257094]
09:46:56.749 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[async]. 4], Partition
[0], Offset [153], Key [310331]
```

subscribePattern Mode

Users specify a wildcard instead of specifying a stream name. For example, stream.* indicates that stream1, stream2, stream_123, and so on are consumed. An existing, added, or deleted stream can be consumed by a consumer group as long as it matches the wildcard.

A code example is provided as follows:

```
package com.huaweicloud.dis.demo.adapter;

import com.huaweicloud.dis.DISConfig;
import com.huaweicloud.dis.adapter.kafka.clients.consumer.*;
import com.huaweicloud.dis.adapter.kafka.common.TopicPartition;
import com.huaweicloud.dis.adapter.kafka.common.serialization.StringDeserializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.regex.Pattern;

public class DISKafkaConsumerSubscribePatternDemo
{
    private static final Logger LOGGER =
LoggerFactory.getLogger(DISKafkaConsumerSubscribePatternDemo.class);

    public static void main(String[] args)
    {
        // There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt
```

```
the AK/SK in the configuration file or environment variables for storage;
       // In this example, the AK and SK stored in the environment variables are used for identity
authentication. Before running this example, configure environment variables HUAWEICLOUD_SDK_AK and
HUAWEICLOUD_SDK_SK in the local environment.
       String ak = System.getenv("HUAWEICLOUD_SDK_AK");
       String sk = System.getenv("HUAWEICLOUD_SDK_SK");
       // YOU ProjectId
       String projectId = "YOU_PROJECT_ID";
      // YOU DIS Stream
       String streamName = "YOU_STREAM_NAME";
//Consumption group ID, which is used to record the offset.
      String groupId = "YOU_GROUP_ID";
      // DIS region
      String region = "your region";

        Properties props = new Properties();
        props.setProperty(DISConfig.PROPERTY_AK, ak);
        props.setProperty(DISConfig.PROPERTY_SK, sk);
        props.setProperty(DISConfig.PROPERTY_PROJECT_ID, projectId);
        props.setProperty(DISConfig.PROPERTY_REGION_ID, region);
        props.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
        props.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
        props.setProperty(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        props.setProperty(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");
        props.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
OffsetResetStrategy.LATEST.name());

//By default, the domain name is automatically used for access instead of configuring an endpoint. If an
endpoint is required, remove the following comments and set the endpoint:
        // props.setProperty(DISConfig.PROPERTY_ENDPOINT, "https://dis-${region}.myhuaweicloud.com");

        Consumer<String, String> consumer = new DISKafkaConsumer<>(props);
//With the subscribePattern mode, you only need to specify a wildcard.
        consumer.subscribe(Pattern.compile(streamNamePattern), new ConsumerRebalanceListener()
        {
            @Override
            public void onPartitionsRevoked(Collection<TopicPartition> collection)
            {
                LOGGER.info("onPartitionsRevoked [{}]", collection);
            }

            @Override
            public void onPartitionsAssigned(Collection<TopicPartition> collection)
            {
                LOGGER.info("onPartitionsAssigned [{}]", collection);
            }
        });

        while (true)
        {
            try
            {
                ConsumerRecords<String, String> records = consumer.poll(Long.MAX_VALUE);

                if (!records.isEmpty())
                {
                    for (TopicPartition partition : records.partitions())
                    {
                        List<ConsumerRecord<String, String>> partitionRecords = records.records(partition);
                        for (ConsumerRecord<String, String> record : partitionRecords)
                        {
                            LOGGER.info("Value [{}], Partition [{}], Offset [{}], Key [{}]",
                                    record.value(), record.partition(), record.offset(), record.key());
                        }
                    }
//Submit the current offset asynchronously after data processing is complete or submit commitSync
synchronously.
```

```
                    consumer.commitAsync(new OffsetCommitCallback()
                    {
                        @Override
                        public void onComplete(Map<TopicPartition, OffsetAndMetadata> map, Exception e)
                        {
                            if (e == null)
                            {
                                LOGGER.debug("Success to commit offset [{}]", map);
                            }
                            else
                            {
                                LOGGER.error("Failed to commit offset [{}]", e.getMessage(), e);
                            }
                        }
                    });
                }
            }
            catch (Exception e)
            {
                LOGGER.info(e.getMessage(), e);
            }
        }
    }
}
```

After the program runs, wait about 20s until the allocation is completed. The data can be consumed. The following is an example code execution log:

```
10:10:36.420 INFO  c.h.d.a.k.c.DISKafkaConsumer - create DISKafkaConsumer successfully
10:10:36.481 INFO  c.h.d.a.k.c.Coordinator - Heartbeat {"state":"JOINING"}
10:10:36.486 INFO  c.h.d.a.k.c.Coordinator - joinGroupRequest {"groupId":"ding","clientId":"consumer-
cad967ba-70ab-4e02-b184-f60b95fe3256","streamPattern":"stream.*"}
10:10:36.697 INFO  c.h.d.a.k.c.Coordinator - joinGroupResponse {"state":"OK","subscription":
["stream_hello","stream_world"],"syncDelayedTimeMs":21000}
10:10:57.699 INFO  c.h.d.a.k.c.Coordinator - syncGroup {"groupId":"ding","clientId":"consumer-
cad967ba-70ab-4e02-b184-f60b95fe3256","generation":-1}
10:10:57.746 INFO  c.h.d.a.k.c.Coordinator - syncGroup {"state":"OK","generation":34,"assignment":
{"stream_hello":[0],"stream_world":[0]}}
10:10:57.770 INFO  c.h.d.d.a.DISKafkaConsumerSubscribePatternDemo - onPartitionsAssigned
[[stream_hello-0, stream_world-0]]
10:10:57.770 INFO  c.h.d.a.k.c.Coordinator - Heartbeat {"state":"STABLE"}
10:11:08.466 INFO  c.h.d.a.k.c.Coordinator - Heartbeat {"state":"STABLE"}
10:11:09.992 INFO  c.h.d.d.a.DISKafkaConsumerSubscribePatternDemo - Value [Hello world[sync]. 0],
Partition [0], Offset [154], Key [181881]
10:11:09.993 INFO  c.h.d.d.a.DISKafkaConsumerSubscribePatternDemo - Value [Hello world[sync]. 1],
Partition [0], Offset [155], Key [483023]
10:11:09.993 INFO  c.h.d.d.a.DISKafkaConsumerSubscribePatternDemo - Value [Hello world[sync]. 2],
Partition [0], Offset [156], Key [32453]
10:11:10.093 INFO  c.h.d.d.a.DISKafkaConsumerSubscribePatternDemo - Value [Hello world[sync]. 3],
Partition [0], Offset [157], Key [111948]
10:11:10.180 INFO  c.h.d.d.a.DISKafkaConsumerSubscribePatternDemo - Value [Hello world[sync]. 4],
Partition [0], Offset [158], Key [822860]
```

6.2.5 Consumption Offset

Two offset committing policies are available: automatic and manual. When creating a DISKafkaConsumer object, set the **enable.auto.commit** parameter to specify a desired offset committing policy. If the value is set to **true**, automatic offset committing is used. If the value is set to **false**, manual offset committing is used.

With automatic offset committing, the consumer enables the coordinator to commit offsets every *auto.commit.interval.ms*. With manual offset committing, instead of relying on the consumer to periodically commit consumed offsets, users can control when records should be considered as consumed and hence commit their offsets.

- Automatic

  When a consumer is created, automatic offset committing is set by default. The default committing interval is 5,000 ms. Parameters for automatic offset committing are as follows:

```
Props.setProperty("enable.auto.commit", "true");//Automatic offset committing is used.
Props.setProperty("auto.commit.interval.ms", "5000");//Offsets are committed at an interval of 5,000 ms.
```

- Automatic

In some scenarios, offsets need to be more strictly managed to ensure that messages are not repeatedly consumed or are not lost. For example, the pulled messages need to be written to the database for processing, or are used for complex service processing such as processing other network access requests. In such scenarios, the messages are regarded as successfully consumed only after all services are processed. In this case, you must manually control offset committing. Parameters for manual offset committing are as follows:

```
props.put("enable.auto.commit", "false");//Manual offset committing is used.
```

After the services are successfully processed, call the commitAsync() or commitSync() method to commit offsets. commitAsync() is used to commit offsets asynchronously. With this method, the consumer threads will not be blocked, and the next pull operation may be started before the offset committing result is returned. To obtain the committing result, add the OffsetCommitCallback method. After the offsets are committed, the onComplete() method is automatically called for processing of different logics based on the callback results.

CommitSync() is used to commit offsets synchronously. With this method, the consumer thread will be blocked until the offset committing result is returned.

In addition, the specific offset data of the specific partition may be further controlled. The confirmed offset is the maximum offset of the accepted data plus 1. For example, when a batch of data is consumed and the offset of the last record is 100, commit offset 101. In this case, consumption starts from the record whose offset is 101 and data will not be consumed repeatedly. The code sample is as follows:

```
ConsumerRecords<String, String> records = consumer.poll(Long.MAX_VALUE);

if (!records.isEmpty())
{
    for (TopicPartition partition : records.partitions())
    {
        List<ConsumerRecord<String, String>> partitionRecords = records.records(partition);
        for (ConsumerRecord<String, String> record : partitionRecords)
        {
            LOGGER.info("Value [{}], Partition [{}], Offset [{}], Key [{}]",
                    record.value(), record.partition(), record.offset(), record.key());
        }
        if (!partitionRecords.isEmpty())
        {
// Confirm the specific offset of a partition synchronously.
            long lastOffset = partitionRecords.get(partitionRecords.size() - 1).offset();
            consumer.commitSync(Collections.singletonMap(partition, new OffsetAndMetadata(lastOffset +
1)));
        }
    }
}
```

6.2.6 Adaptation to the Native KafkaConsumer API

Table 6-7 API adaptation description

Native KafkaConsumer	Type	DISKafkaConsumer	Description
Set<TopicPartition> assignment()	API	Supported	Obtain information about the consumed stream and partition.
Set<String> subscription()	API	Supported	Obtain the name of the stream that has been subscribed to by the consumer.
void assign(Collection<TopicPartition> var1)	API	Supported	Allocate a specified partition.
void subscribe(Collection<String> var1)	API	Supported	Subscribe to a specified stream.
void subscribe(Collection<String> var1, ConsumerRebalanceListener var2)	API	Supported	Subscribe to a specified stream and supports callback of ConsumerRebalanceListener.
void subscribe(Pattern var1, ConsumerRebalanceListener var2)	API	Supported	Subscribe to all streams that match wildcards and supports callback of ConsumerRebalanceListener.
void unsubscribe()	API	Supported	Cancels all subscription.

Native KafkaConsumer	Type	DISKafkaConsumer	Description
ConsumerRecords <K, V> poll(long var1)	API	Supported	The message is obtained. Checksum (CRC32 check value of the message), serializedKeySize (byte length after key serialization), and serializedValue-Size (byte length after key serialization).
void commitSync()	API	Supported	Commit the consumed offsets synchronously.
void commitSync(final Map<TopicPartition, OffsetAndMetadata> offsets)	API	Supported	Commit the specified offset synchronously.
void commitAsync()	API	Supported	Commit the consumed offsets asynchronously.
public void commitAsync(OffsetCommitCallback callback)	API	Supported	Commit the current consumed offset asynchronously and supports callback of OffsetCommitCall-back.
void commitAsync(Map<TopicPartition, OffsetAndMetadata> offsets, OffsetCommitCallback callback)	API	Supported	Commit the specified offset asynchronously and supports callback of OffsetCommitCall-back.
void seek(TopicPartition partition, long offset)	API	Supported	Set the specified offset for a partition.

Native KafkaConsumer	Type	DISKafkaConsumer	Description
void seekToBeginning(Collection<TopicPartition> partitions)	API	Supported	Set the earliest value for the partition offset.
void seekToEnd(Collection<TopicPartition> partitions)	API	Supported	Set the latest value for the partition offset.
long position(TopicPartition partition)	API	Supported	Obtain the offset of the current consumed data in the partition.
OffsetAndMetadata committed(TopicPartition partition)	API	Supported	Obtain the committed offset of the partition.
List<PartitionInfo> partitionsFor(String topic)	API	Supported	Obtain the partition information of the stream, but leader, replicas, and inSyncReplicas in PartitionInfo are not implemented.
Map<String, List<PartitionInfo>> listTopics()	API	Supported	Obtain the stream information, but leader, replicas, and inSyncReplicas in PartitionInfo are not implemented.
void pause(Collection<TopicPartition> partitions)	API	Supported	Suspends the partition consumption.
void resume(Collection<TopicPartition> partitions)	API	Supported	Resumes the partition consumption.
Set<TopicPartition> paused()	API	Supported	Obtain all partitions that stop being consumed.

Native KafkaConsumer	Type	DISKafkaConsumer	Description
close()	API	Supported	Disable the consumer.
Map<MetricName, ? extends Metric> metrics()	API	Not supported	Obtain statistics.
wakeup()	API	Not supported	The internal implementation principles are different.
group.id	Parameter	Supported	Consumer group ID.
client.id	Parameter	Supported	**client.id** of each consumer must be unique. If **client.id** is not specified, the dis kafka consumer will generate a UUID as a **client.id**.
key.deserializer	Parameter	Supported	The meaning of this parameter is the same as that in Kafka. The default value is **StringDeserializer**. In Kafka, **StringDeserializer** has no default value, and you must configure a value for it.
value.deserializer	Parameter	Supported	The meaning of this parameter is the same as that in Kafka. The default value is **StringDeserializer**. In Kafka, this parameter has no default value, and you must configure a value for it.

Native KafkaConsumer	Type	DISKafkaConsumer	Description
enable.auto.commit	Parameter	Supported	The default value is the same as that in Kafka, which is **true**. ● **true**: The automatic offset committing is enabled and offsets are automatically committed at an interval of *auto.commit.interval.ms*. ● **false**: Offsets are not automatically committed.
auto.commit.interval.ms	Parameter	Supported	Interval for automatically committing offsets, in milliseconds. The default value is **5000**.

Native KafkaConsumer	Type	DISKafkaConsumer	Description
auto.offset.reset	Parameter	Supported	The default value is the same as that in Kafka, which is **latest**. This parameter is used to automatically set the offset position when there is no initial offset or the offset is incorrect. ● **earliest**: The offset is automatically reset to the earliest value. ● **latest**: The offset is automatically reset to the latest value. ● **none**: If the previous offset is not found in the consumer group, an exception is issued to the consumer.
Others	Parameter	Not supported	-

6.3 Using the Python SDK

6.3.1 Preparing the Installation Environment

The following environment has been prepared:

- Python 2.7 or a later version has been installed and its environment variables have been configured.
- PyCharm has been installed.

6.3.2 Configuring a Sample Project

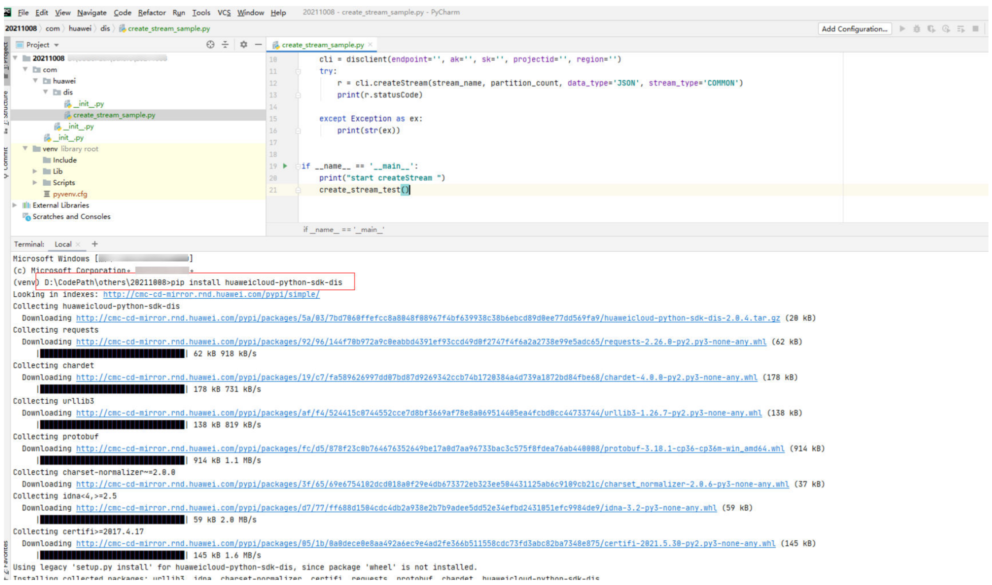
For details about the sample code, see https://github.com/huaweicloud/huaweicloud-sdk-python-dis/tree/master/dis_sdk_python_demo.

Procedure

Step 1 After **huaweicloud-python-sdk-dis** has been published to PyPI, open Command Prompt and run the **pip install huaweicloud-python-sdk-dis** command to install **huaweicloud-python-sdk-dis**.

Step 2 Import the PyCharm project.

1. Choose **File** > **Open**. The **Open File or Project** window is displayed.
2. Select **dis_sdk_python** from the **\Lib\site-packages** Python installation directory. If you cannot find **dis_sdk_python**, upgrade pip or install **huaweicloud-python-sdk-dis**.



```
    cli = disclient(endpoint='', ak='', sk='', projectid='', region='')
    try:
        r = cli.createStream(stream_name, partition_count, data_type='JSON', stream_type='COMMON')
        print(r.statusCode)

    except Exception as ex:
        print(str(ex))


if __name__ == '__main__':
    print("start createStream ")
    create_stream_test()
```

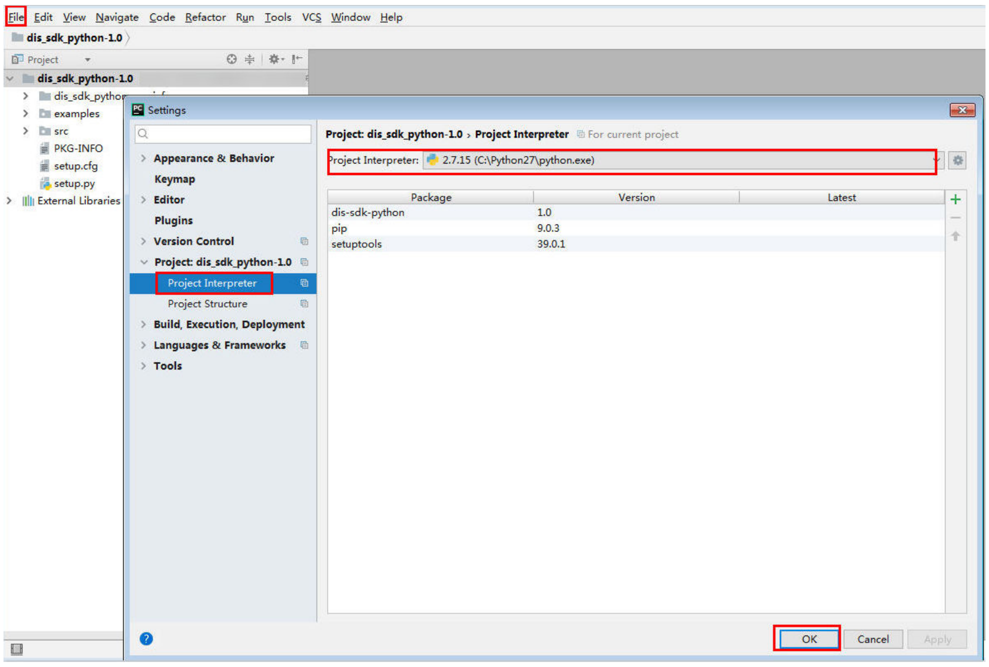
```
Microsoft Windows [[redacted]]
(c) Microsoft Corporation。[redacted]。
(venv) D:\CodePath\others\20211008>pip install huaweicloud-python-sdk-dis
Looking in indexes: http://cmc-cd-mirror.rnd.huawei.com/pypi/simple/
Collecting huaweicloud-python-sdk-dis
  Downloading http://cmc-cd-mirror.rnd.huawei.com/pypi/packages/5a/03/7bd7060ffefcc8a8048f08967f4bf639938c38b6ebcd89d0ee77dd569fa9/huaweicloud-python-sdk-dis-2.0.4.tar.gz (20 kB)
Collecting requests
  Downloading http://cmc-cd-mirror.rnd.huawei.com/pypi/packages/92/96/144f70b972a9c0eabbd4391ef93ccd49d0f2747f4f6a2a2738e99e5adc65/requests-2.26.0-py2.py3-none-any.whl (62 kB)
     |████████████████████████████████| 62 kB 918 kB/s
Collecting chardet
  Downloading http://cmc-cd-mirror.rnd.huawei.com/pypi/packages/19/c7/fa589626997dd07bd87d9269342ccb74b1720384a4d739a1872bd84fbe68/chardet-4.0.0-py2.py3-none-any.whl (178 kB)
     |████████████████████████████████| 178 kB 731 kB/s
Collecting urllib3
  Downloading http://cmc-cd-mirror.rnd.huawei.com/pypi/packages/af/f4/524415c0744552cce7d8bf3669af78e8a069514405ea4fcbd0cc44733744/urllib3-1.26.7-py2.py3-none-any.whl (138 kB)
     |████████████████████████████████| 138 kB 819 kB/s
Collecting protobuf
  Downloading http://cmc-cd-mirror.rnd.huawei.com/pypi/packages/fc/d5/878f23c0b764676352649be17a0d7aa96733bac3c575f8fdea76ab440008/protobuf-3.18.1-cp36-cp36m-win_amd64.whl (914 kB)
     |████████████████████████████████| 914 kB 1.1 MB/s
Collecting charset-normalizer~=2.0.0
  Downloading http://cmc-cd-mirror.rnd.huawei.com/pypi/packages/3f/65/69e6754102dcd018a0f29e4db673372eb323ee504431125ab6c9109cb21c/charset_normalizer-2.0.6-py3-none-any.whl (37 kB)
Collecting idna<4,>=2.5
  Downloading http://cmc-cd-mirror.rnd.huawei.com/pypi/packages/d7/77/ff688d1504cdc4db2a938e2b7b9adee5dd52e34efbd2431051efc9984de9/idna-3.2-py3-none-any.whl (59 kB)
     |████████████████████████████████| 59 kB 2.0 MB/s
Collecting certifi>=2017.4.17
  Downloading http://cmc-cd-mirror.rnd.huawei.com/pypi/packages/05/1b/0a0dece0e8aa492a6ec9e4ad2fe366b511558cdc73fd3abc82ba7348e875/certifi-2021.5.30-py2.py3-none-any.whl (145 kB)
     |████████████████████████████████| 145 kB 1.6 MB/s
Using legacy 'setup.py install' for huaweicloud-python-sdk-dis, since package 'wheel' is not installed.
Installing collected packages: urllib3, idna, charset-normalizer, certifi, requests, protobuf, chardet, huaweicloud-python-sdk-dis
```

3. Click **OK**.

Step 3 Configure the **sdk_python** project.

1. In the navigation tree on the left, choose **File** > **Settings** > **Editor** > **Font** and set the font background color.
2. In the navigation tree on the left, choose **File** > **Settings** > **Project Interpreter** and add Python.
3. Select a proper project interpreter and click **OK**.

Figure 6-4 Adding Python



Project: dis_sdk_python-1.0 › Project Interpreter For current project

Project Interpreter: 2.7.15 (C:\Python27\python.exe)

Package	Version	Latest
dis-sdk-python	1.0	
pip	9.0.3	
setuptools	39.0.1	

4. In the navigation tree on the left, choose **File** > **Settings** > **Editor** > **File Encodings** and set the PyCharm code.

   Set **Global Encoding**, **Project Encoding**, and **Default encoding for properties files** to **UTF-8**.

----End

6.3.3 Initializing a DIS SDK Client Instance

You can use either of the following methods to initialize the DIS SDK client instance: For details about **endpoint**, **ak**, **sk**, **region**, and **projectId**, see **Obtaining Authentication Information**.

-
  ```
  cli = disclient(endpoint='**your-endpoint**',
              ak='**your-ak**',
              sk='**your-sk**',
              projectid='**your-projectid**',
              region='**your-region**')
  ```

6.3.4 Creating a Stream

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

For details about how to configure parameters, see configuration methods in .

- stream_type= " " #Not specified.

If **stream_type** is not specified, configure the parameters of **Dump_switch** in the **createstream_sample.py** file.

- stream_type= "FILE" #Set to file.

If **stream_type** is set to **File**, configure the parameters of **Dump_switch_FILE** in the **createstream_sample.py** file.

After configuring the parameters, run the **createstream_sample.py** file to call **createStream_test** by default. If response code 201 is returned, the stream is successfully created.

6.3.5 Creating a Dump Task

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

For details about how to configure parameters, see configuration methods in .

```
Configure the following parameters:
streamname ="dis-test1"  #Name of an existing stream
task_name='113'
```

Adding an OBS dump task is used as an example. The **value** parameter corresponds to the **key** parameter.

```
basic_Schema=DumpTask.setSchema(key=['consumer_strategy','deliver_time_interval','agency_name','retry_d
uration'],
      value=['LATEST', 30, 'dis_admin_agency',1800])
obs_dump_task =['destination_file_type','obs_bucket_path','file_prefix', 'partition_format','record_delimiter']
obs_Schema = DumpTask.setSchema(basic_Schema=basic_Schema,
                key=obs_dump_task,value=['text','obs-1253', '','yyyy', '|'])
```

#Add an OBS dump task and configure **obs_Schema**.

```
 cli.add_dump_task(streamname, task_name,'OBS',obs_Schema)
```

- After configuring the parameters, run the **add_dump_task_sample.py** file to call **add_dump_task_test** by default. If response code 201 is returned, the dump task is successfully created.

6.3.6 Deleting a Stream

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameter:

```
streamname = ""  #Name of an existing stream.
```

After configuring the parameter, run the **deleteStream_sample.py** file to call **deleteStream_test** by default. If response code 204 is returned, the stream is successfully deleted.

6.3.7 Deleting a Dump Task

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameters:

```
streamname = ""  #Name of an existing stream.
task_name="xx"
```

NOTE

After configuring **task_name**, the dump task will be deleted from the stream.

After configuring the parameters, run the **delete_dump_task_sample.py** file to call **delete_dump_task_test** by default. If response code 204 is returned, the dump task is successfully deleted.

6.3.8 Querying a Stream List

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameter:

```
start_stream_name = ""  #Leave unspecified or set to an existing stream name.
```

Run the **listStream_sample.py** file to call **listStream_test**. If response code 200 is returned, the stream list is displayed.

The following is an example response:

```
200
{'stream_names': ['dis-jLGp', 'dis-w_p', 'dis_test1', 'dis_test2'], 'has_more_streams': False, 'total_number': 4}
```

6.3.9 Querying a Dump Task List

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameter:

```
streamname = "XXX"  #Name of an existing stream.
```

Run the **list_dump_task_sample.py** file to call **list_dump_task_test**. If response code 200 is returned, the dump list is displayed.

The following is an example response:

```
200
{'total_number': 1, 'tasks': [{'last_transfer_timestamp': 1538018769241, 'state': 'RUNNING', 'create_time':
1537949648144, 'destination_type': 'OBS', 'task_name': 'task_test1'}]}
```

6.3.10 Querying Stream Details

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameter:

```
streamname = "dis-test1"  #Name of an existing stream.
```

After configuring the parameter, run the **describeStream_sample.py** file to call **describeStream_test** by default.

The following is an example response:

```
200
{"status": "RUNNING", "stream_name": "dis-test1", "data_type": "BLOB", "has_more_partitions": false,
"stream_type": "COMMON", "stream_id": "L84hxfES223eVrFyxiE", "retention_period": 168, "create_time":
1532423353637, "last_modified_time": 1532423354625, "partitions": [{"status": "ACTIVE", "hash_range":
"[0 : 9223372036854775807]", "sequence_number_range": "[0 : 10]", "partition_id":
"shardId-0000000000"}]}
```

6.3.11 Querying Dump Details

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameters:

```
streamname ="dis-test1"  #Name of an existing stream.
task_name="test_1"  #Query the xx dump task of the stream.
```

After configuring the parameters, run the **describe_dump_task_sample.py** file to call **describe_dump_task_test** by default.

The following is an example response:

```
200
{'state': 'RUNNING', 'stream_name': 'dis-test1', 'create_time': 1537949648144, 'last_transfer_timestamp': 1538018072564, 'destination_type': 'OBS', 'obs_destination_description': {'obs_bucket_path': '002', 'deliver_time_interval': 30, 'retry_duration': 0, 'agency_name': 'all', 'partition_format': 'yyyy/MM/dd/HH/mm', 'destination_file_type': 'text', 'record_delimiter': '|', 'consumer_strategy': 'LATEST', 'file_prefix': ''}, 'task_name': 'test_1', 'partitions': [{'state': 'RUNNING', 'discard': 0, 'last_transfer_offset': 500, 'partitionId': 'shardId-0000000000', 'last_transfer_timestamp': 1538018072564}, {'state': 'RUNNING', 'discard': 0, 'last_transfer_offset': 500, 'partitionId': 'shardId-0000000001', 'last_transfer_timestamp': 1538018072564}]}
```

6.3.12 Uploading Streaming Data in JSON Format

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameter:

```
streamname ="dis-test1"  #Name of an existing stream.
```

records of the **putRecords_test** method in the **putRecords_sample.py** file is the data to be uploaded. The data is uploaded in the following format:

```
records=[{"data": "abcdefd", "partition_key": "1"}]
#data: data to be uploaded. The value is user-definable; partition_key: partition to which data is written. The value is user-definable.
 record1 = {"data": "xxx","partition_key": partition_key}
#You can write multiple pieces of data. The data format is shown in record1. Each time a piece of data is written, the append method is used to transfer the data to the records.
```

After configuring the parameters, run the **putRecords_sample.py** file to call **putRecords_test**. The following is an example response:

```
200
{'failed_record_count': 0, 'records': [{'partition_id': 'shardId-0000000001', 'sequence_number': '15'}]}
```

6.3.13 Uploading Streaming Data in Protobuf Format

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Initialize the DIS client and add the **bodySerializeType** parameter as follows:

```
cli = disclient(endpoint='', ak='', sk='', projectid='', region='',bodySerializeType='protobuf')
```

Configure the following parameter:

```
streamname ="dis-test1"  #Name of an existing stream.
```

Parameter **bodySerializeType** in the **test** method must be set to **protobuf** so that streaming data is uploaded in protobuf format.

records of the **protobuf_putRecords_test** method in the **protobuf_putRecords_sample.py** file is the data to be uploaded. The data is uploaded in the following format:

```
records=[{"data": "abcdefd", "partition_key": "1"}]
#data: data to be uploaded. The value is user-definable; partition_key: partition to which data is written. The value is user-definable.
 record1 = {"data": "xxx","partition_key": partition_key}
```

```
#You can write multiple pieces of data. The data format is shown in record1. Each time a piece of data is written, the append method is used to transfer the data to the records.
```

After configuring the parameters, run the **protobuf_putRecords_sample.py** file to call **protobuf_putRecords_test**. The following is an example response:

```
200
{'failed_record_count': 0, 'records': [{'partition_id': 'shardId-0000000001', 'sequence_number': '15'}]}
```

6.3.14 Downloading Streaming Data

Downloading Stream Data in JSON Format

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameters:

```
streamname=" "   #Name of the stream.
startSeq=' 0' # Sequence number.
partitionId="shardId-0000000000"
```

Run **getCursor_test** to change the value of **cursorType** to **AT_SEQUENCE_NUMBER**.

After configuring the parameters, run the **getRecords_sample.py** file to call **getRecords_test**. The following is an example response:

```
200
{'next_partition_cursor':
'eyJnZXRJdGVyYXRvclBhcmFtIjp7InN0cmVhbS1uYW1lIjoiZGlzX3Rlc3QxIiwicGFydGl0aW9uLWlkIjoic2hhcmRJZC0wMDAwMDAwMDAwIiwiY3Vyc29yLXR5cGUiOiJBVF9TRVFVRU5DRV9OVU1CRVIiLCJzdGFydGluZy1zZXF1ZW5jZS1udW1iZXIiOiI2In0sImdlbmVyYXRlVGltZXN0YW1wIjoxNTU0NzA2NTc5MzA5fQ', 'records':
[{'sequence_number': '4', 'data': b'xxxxx', 'partitionKey': '0', 'timestamp': 1554705842558, 'timestamp_type':
'CreateTime'}, {'sequence_number': '5', 'data': b'xxxxx', 'partitionKey': '0', 'timestamp': 1554705842558,
'timestamp_type': 'CreateTime'}]}
```

Downloading Streaming Data in Protobuf Format

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

```
Initialize the DIS client and add the bodySerializeType parameter as follows:
cli = disclient(endpoint='', ak='', sk='', projectid='', region='',bodySerializeType='protobuf')
```

Configure the following parameters:

```
streamname=" "   #Name of the stream.
partitionId="shardId-0000000000"
```

Run **getCursor_test** to change the value of **cursorType** to **AT_SEQUENCE_NUMBER**.

NOTE

The **test_0** method is used to obtain a cursor and the **test** method is used to download data. Compared to **test_0**, **test** has the **bodySerializeType="protobuf"** parameter, which is not provided in **test_0**.

After configuring the parameters, run the **protobuf_getrecords_sample.py** file to call **getRecords_test**. The following is an example response:

```
200
{'next_partition_cursor':
```

```
'eyJnZXRJdGVyYXRvclBhcmFtIjp7InN0cmVhbS1uYW1lIjoiZGlzX3Rlc3QxIiwicGFydGl0aW9uLWlkIjoic2hhcmRJZ
C0wMDAwMDAwMDAwIiwiY3Vyc29yLXR5cGUiOiJBVF9TRVFVRU5DRV9OVU1CRVIiLCJzdGFydGluZy1zZXF1Z
W5jZS1udW1iZXIiOiI2In0sImdlbmVyYXRlVGltZXN0YW1wIjoxNTU0NzA2NTc5MzA5fQ', 'records':
[{'sequence_number': '4', 'data': b'xxxxx', 'partitionKey': '0', 'timestamp': 1554705842558, 'timestamp_type':
'CreateTime'}, {'sequence_number': '5', 'data': b'xxxxx', 'partitionKey': '0', 'timestamp': 1554705842558,
'timestamp_type': 'CreateTime'}]}
```

6.3.15 Creating an Application

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameter:

```
appName = ""  #Application name.
```

After configuring the parameter, run the **createApp_sample.py** file to call **createApp_test**. If response code 201 is returned, the application is successfully created.

6.3.16 Deleting an Application

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameter:

```

appName = "" #Name of the application to be deleted.
```

After configuring the parameter, run the **deleteApp_sample.py** file to call **deleteApp_test**. If response code 204 is returned, the application is successfully deleted.

6.3.17 Viewing Application Details

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameter:

```
appname= "app1"  #Name of the application to be queried.
```

After configuring the parameter, run the **describeApp_sample.py** file to call **describeApp_test**.

The following is an example response:

```
200
{'app_name': 'app1', 'app_id': 'OPKQuggQVtfqhyvK0cs', 'create_time': 1532425956631}
```

6.3.18 Querying an Application List

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

In the listApp_test method, limit specifies the maximum number of apps that can be returned in a single request. The value range is 1 to 100.

Configure the following parameter:

```
startAppName="app1"  #Application name. The application list is returned from the current stream and does not contain the application.
```

After configuring the parameter, run the **listApp_sample.py** file to call **Applist_test**.

The following is an example response:

```
200
{'has_more_app': False, 'apps': [{'app_id': 'kpvGNrFYfKjpqTSdPIX', 'create_time': 1543301301992, 'app_name': 'sadfghjkl'}, {'app_id': 'MtPG1lD1E7IesDuOcNt', 'create_time': 1542765418080, 'app_name': 'testAppName2'}]}
```

6.3.19 Adding a Checkpoint

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameters:

```
streamname=" "   #Name of the stream.
appName="xx"  #Name of an existing application.
partitionId="shardId-0000000000"   #Unique identifier of the partition.
seqNumber="0"   #Sequence number.
metadata=""  #Metadata information of the consumer application. The maximum length of the metadata information is 1,000 characters.
```

NOTE

Before obtaining the partition ID, import the stream name and then refer to the instructions in **Querying Stream Details**.

After configuring the parameters, run the **commitCheckpoint_sample.py** file to call **commitCheckpoint_test**. If response code 201 is returned, the checkpoint is successfully added.

6.3.20 Querying a Checkpoint

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameters\:

```
streamname=" "   #Name of the stream.
appName="xx"  #Name of an existing application.
partitionId="shardId-0000000000"   #Unique identifier of the partition.
```

NOTE

Before obtaining the partition ID, import the stream name and then refer to the instructions in **Querying Stream Details**.

After configuring the parameters, run the **getCheckpoint_sample.py** file to call **getCheckpoint_test**. The following is an example response:

```
{
  "sequence_number": "10",
  "metadata": "metadata"
}
```

6.3.21 Changing Partition Quantity

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameters:

```
streamname = "" #Name of the running stream.
target_partition_count = "3"  #Number of the target partitions.
```

After configuring the parameters, run the **changepartitionQuantity_sample.py** file to call **changepartitionQuantity_test**. The following is an example response:

```
{
 "stream_name":"stream_name_test",
 "current_partition_count":2
 "target_partition_count":5
 }
```

6.3.22 Obtaining a Data Cursor

Initialize a DIS SDK client instance as instructed in **Initializing a DIS SDK Client Instance**.

Configure the following parameters:

```
partitionId="shardId-0000000000"
streamname ="dis-test1"  #Name of an existing stream.
Configure the five cursors as follows:
# startSeq and AT_SEQUENCE_NUMBER/AFTER_SEQUENCE_NUMBER are used together.
r = cli.getCursor(streamname, partitionId, cursorType='AT_SEQUENCE_NUMBER', startSeq="0")
# r = cli.getCursor(streamname, partitionId, cursorType='AFTER_SEQUENCE_NUMBER', startSeq="0")
# timestamp and AT_TIMESTAMP are used together.
# r = cli.getCursor(streamname, partitionId, cursorType='AT_TIMESTAMP',timestamp=1554694135190)
# r = cli.getCursor(streamname, partitionId, cursorType='TRIM_HORIZON')
# r = cli.getCursor(streamname, partitionId, cursorType='LATEST')
```

After configuring the parameters, run the **getCursor_sample.py** file to call **getCursor_test**. The following is an example response:

```
200
{"partition_cursor":
"eyJnZXRJdGVyYXRvclBhcmFtIjp7InN0cmVhbS1uYW1lIjoiSCIsInBhcnRpdGlvbi1pZCI6InNoYXJkSWQtMDAwMD
AwMDAwMCIsImN1cnNvci10eXBlIjoiQVRfU0VRVUVOQ0VfTlVNQkVSIiwic3RhcnRpbmctc2VxdWVuY2UtbnVt
YmVyIjoiMCJ9LCJnZW5lcmF0ZVRpbWVzdGFtcCI6MTUzMjQyNDg4NzE1NH0"}
```

7 Error Codes

7.1 DIS Server-Side Error Codes

If an error occurs when you use the SDK, an error code is displayed on the DIS console.

HTTP Status Code	Error Code	Error Message	Description	Solution
441	DIS.4100	Authorization error.	The signature information generated using AK and SK is incorrect.	Ensure that the signature information in the request header is correct.
441	DIS.4101	Authorization header cannot be empty.	The signature information generated using AK and SK is blank.	Ensure that the signature information is generated.
441	DIS.4102	Incorrectly parsed authorization header.	The signature cannot be parsed.	Ensure that the signature information in the request header is correct.
441	DIS.4103	Empty X-Sdk-Date header.	The **X-Sdk-Date** field in the request header is blank.	Ensure that the **X-Sdk-Date** field in the request header is not blank.

HTTP Status Code	Error Code	Error Message	Description	Solution
441	DIS.4104	Error parsing X-Sdk-Date header.	The **X-Sdk-Date** field in the request header cannot be parsed.	Ensure that the **X-Sdk-Date** field in the request header is correct.
441	DIS.4105	Invalid X-Sdk-Date header.	The **X-Sdk-Date** field in the request header is invalid.	Ensure that the **X-Sdk-Date** field in the request header is correct.
441	DIS.4106	Empty AcessKey header.	The **Authorization** field of the request header does not contain AK.	Ensure that AK is contained in the **Authorization** field.
441	DIS.4107	Invalid AcessKey header.	AK in the **Authorization** field of the request header is invalid.	Ensure that AK is valid and correct.
441	DIS.4108	Empty ServiceName header.	The **Authorization** field of the request header does not contain the service name.	Ensure that the **Authorization** field of the request header contain service name **dis**.
441	DIS.4109	The Authorization header must contain the following field: {Credential,SignedHeaders,Signature;}	The **Authorization** field of the request header is incorrect.	Ensure that the **Authorization** field of the request header contains **Credential**, **SignedHeaders**, and **Signature**.

HTTP Status Code	Error Code	Error Message	Description	Solution
441	DIS.4110	Empty Signature header.	The **Authorization** field of the request header does not contain **SignedHeaders**.	Ensure that the signature generation mode is correct.
441	DIS.4111	Invalid Region header.	The region in the **Authorization** field of the request header is invalid.	Ensure that the region is valid.
441	DIS.4112	Invalid authorization request.	The signature information generated using AK and SK is incorrect.	Ensure that the signature generation mode and the information about AK, SK, and region are correct.
441	DIS.4113	Empty Token header.	When token authentication is used, the **X-Auth-Token** field of the request header is blank.	Ensure that the **X-Auth-Token** field of the request header is not blank.
441	DIS.4114	Invalid Token header.	When token authentication is used, the **X-Auth-Token** field of the request header is invalid.	Ensure that the **X-Auth-Token** field of the request header is valid.
403	DIS.4116	Invalid RBAC.	User operations are restricted.	Ensure that all bills have been paid and DIS operating permissions have been obtained.

HTTP Status Code	Error Code	Error Message	Description	Solution
400	DIS.4117	Invalid Project Id.	Invalid project ID.	Ensure that the project ID is valid and correct.
400	DIS.4200	Invalid request.	The user request is invalid.	Ensure that the request is invalid by referring to *API Reference*.
400	DIS.4201	Invalid partition_id.	Invalid partition ID.	Ensure that the partition ID is valid.
400	DIS.4202	Empty request.	The user request is empty.	Enter a valid request.
400	DIS.4203	Invalid monitoring period.	The start time for querying monitoring information is invalid.	Enter a valid timestamp.
400	DIS.4204	The monitoring period cannot be longer than 7 days.	Only the monitoring information generated in the recent seven days can be queried.	Query the monitoring information generated in the recent seven days.
400	DIS.4208	Invalid MRS cluster.	The MRS cluster entered during MRS dump task creation is invalid.	Ensure the MRS cluster name and ID are correct and the cluster is running in security mode.
400	DIS.4209	Invalid metrics label.	The monitoring metric entered during monitoring information query is invalid.	Check and correct the monitoring metric by referring to *API Reference*.

HTTP Status Code	Error Code	Error Message	Description	Solution
400	DIS.4215	Invalid cursor type.	The cursor type entered during the data cursor acquisition is invalid.	Check and correct the cursor type by referring to *API Reference*.
400	DIS.4216	Invalid sequence_number.	The starting sequence number entered during data cursor acquisition is invalid.	Enter a valid starting sequence number.
400	DIS.4217	Invalid partition cursor.	The partition cursor entered during data download from DIS is invalid.	Obtain the partition cursor again and download the data.
400	DIS.4219	The file is constantly resent.	The file has been received.	Do not upload the file again.
400	DIS.4220	The block whose sequence number is %s needs to be resent.	The file block needs to be uploaded again.	Upload the corresponding file block as instructed.
400	DIS.4221	Block seq %s is expected	Duplicate file blocks are uploaded.	Upload the file block from the one expected by the system.
400	DIS.4222	Block seq %s is expected.	The uploaded file blocks are not consecutive.	Upload the file block from the one expected by the system.
400	DIS.4223	The file size exceeds the limit.	The file capacity exceeds the upper limit.	Split the file and upload it again.

HTTP Status Code	Error Code	Error Message	Description	Solution
400	DIS.4224	The sequence number is out of range.	The starting sequence number entered during data cursor acquisition is invalid.	Enter a valid starting sequence number.
400	DIS.4225	Expired partition cursor.	The partition cursor entered during data download from DIS expires.	Obtain the partition cursor again and download the data.
400	DIS.4226	A partition iterator error occurred or a record to which the SN corresponds has expired. Try to obtain the partition iterator again.	The starting sequence number of the partition cursor entered during data acquisition expires.	Obtain the data cursor and use the new cursor to obtain data.
400	DIS.4300	Request error.	The request body is incorrect.	Correct the request body by referring to *API Reference*.
400	DIS.4301	The stream does not exist.	The stream does not exist.	Ensure that the stream exists.
400	DIS.4302	The partition does not exist.	The stream partition does not exist.	Ensure that the partition ID exists.
400	DIS.4303	Exceeded traffic control limit.	The flow control limit is exceeded.	Add partitions or reduce the upload rate.
400	DIS.4305	Too many stream requests.	An excessive number of user requests are generated at the same time.	Decrease the requesting frequency and try again.

HTTP Status Code	Error Code	Error Message	Description	Solution
400	DIS.4306	The bucket does not exist.	The OBS bucket does not exist.	Ensure that the OBS bucket exists.
400	DIS.4307	The stream already exists.	The stream already exists.	Enter a new stream name.
400	DIS.4308	Insufficient quota.	Insufficient stream or partition quotas.	Release the resources that will not be used to ensure that the quota limit is not exceeded or submit a service ticket to increase the quota limit.
400	DIS.4309	Too many request failures. Please try again later.	The IP address is added to the blacklist.	Ensure that the authentication information and request are valid and try again later.
400	DIS.4310	OBS access error.	OBS failed to be accessed.	Ensure that the user has the permission to access OBS.
400	DIS.4329	app quota exceeded.	The application quota exceeds the limit.	Release the applications that are not used.
400	DIS.4330	app already exist.	An application with the same name already exists.	Enter a new application name.
400	DIS.4331	app is using.	The application fails to be deleted.	Ensure that the application that you want to delete is not being used.
400	DIS.4332	app not found.	The specified application does not exist.	Ensure the application name is correct.

HTTP Status Code	Error Code	Error Message	Description	Solution
400	DIS.4335	Invalid IAM agency.	The IAM agency used during dump task creation is invalid.	Ensure that **dis_admin_agency** created by DIS or the user-defined IAM agency exists and permission is complete.
400	DIS.4336	Invalid HDFS path.	The MRS HDFS path entered during MRS dump task creation is invalid.	Ensure that the MRS HDFS path exists.
400	DIS.4337	The DLI database does not exist.	The DLI database entered during DLI dump task creation does not exist.	Ensure that the DLI database exists.
400	DIS.4338	The DLI table does not exist.	The DLI table entered during DLI dump task creation does not exist.	Ensure that the DLI table exists and is an internal table.
400	DIS.4350	Invalid DWS cluster.	The DWS cluster entered during DWS dump task creation does not exist.	Ensure that the DWS cluster exists and is running properly.
400	DIS.4351	Invalid KMS userKey.	The KMS key entered during DWS dump task creation is invalid.	Ensure that the KMS key exists.
400	DIS.4354	The transfer task does not exist.	The dump task to be deleted or updated does not exist.	Ensure that the dump task exists.

HTTP Status Code	Error Code	Error Message	Description	Solution
400	DIS.4355	The transfer task already exists.	When you create a dump task in a stream, another dump task with the same name already exists in this stream.	Enter a new dump task name.
400	DIS.4357	Exceeded transfer task quota.	A maximum of five dump tasks can be created for one stream at the same time. Creating six dump tasks will exceed the quota limit.	Delete the discarded dump tasks and then add dump tasks again.
400	DIS.4358	The stream supports specific transfer tasks. Check the data type of the stream.	Common dump tasks cannot be created in the stream for small file dump.	Create a new stream and then dump tasks in the stream.
400	DIS.4360	Invalid data schema.	The data schema entered during stream creation or update is invalid.	Ensure that the data schema format is correct and try again.
400	DIS.4601	The number of resource tags has reached the maximum.	A resource has a maximum of 10 tags. Adding 11 tags will exceed the quota limit.	Delete the discarded tags and then add tags again.

HTTP Status Code	Error Code	Error Message	Description	Solution
400	DIS.4602	Invalid resource type.	The resource type is invalid.	Ensure that the resource type is valid.
400	DIS.4603	The resource does not exist.	The resource does not exist.	Ensure that the resource is not deleted.
400	DIS.4604	The key does not exist.	The tag key does not exist.	Ensure that the tag key exists.
400	DIS.4605	The action is not supported.	The current tag operation is not supported.	Ensure that the current tag operation is valid. Currently, only the create and delete operations are supported.
500	DIS.5000-DIS.5999	System error. **NOTE** Contact technical support to handle system errors.	-	-

A Change History

Release Date	Description
2019-05-08	This is the first official release.