Data Lake Insight

SDK Reference

Issue 01

Date 2025-06-26

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

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

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

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to the *Vul. Response Process.* For details about this process, visit the following web page:
https://www.huawei.com/en/psirt/vul-response-process
For vulnerability information, enterprise customers can visit the following web page:
https://securitybulletin.huawei.com/enterprise/en/security-advisory

Contents

1 Overview

Data Lake Insight (DLI) software development kits (SDKs) encapsulate the RESTful APIs provided by DLI to simplify development.

DLI offers support for two types of SDKs: one is the general SDK V3, and the other is the SDK developed by DLI (including DLI SDK V1 and DLI SDK V2).

This document describes how to use the SDK developed by DLI.

Table 1-1 DLI SDK types

SDK	Version	Description	How to Use
General SDK	V3	The SDK V3 APIs are automatically generated from the YAML files defining the APIs, ensuring parameter consistency with the service's own APIs. You can directly call the API functions offered by SDK V3 to submit both DLI SQL and DLI Spark jobs, streamlining the development process. **The SDK you download on the API Explorer page is the universal version SDK V3.**	**General SDK V3**
SDK developed by DLI	DLI SDK V1	SDKs developed by DLI, corresponding to the **dli-sdk-*x*-1.*x*.*x*** version in the DLI SDK installation package.	This version will be deprecated and is not recommended.

SDK	Version	Description	How to Use
	DLI SDK V2	SDKs developed by DLI based on the general SDK V3, corresponding to the **dli-sdk-*x*-2.*x*.*x*** version in the DLI SDK installation package. You can directly call DLI SDK V2 to submit both DLI SQL and DLI Spark jobs, streamlining the development process. Download the DLI SDK V2 installation package on the DLI management console. **NOTE** Starting May 2024, new users can directly use DLI's SDK V2 without needing to have their accounts whitelisted. For users who started using DLI before May 2024, to use this function, they must submit a service ticket to have their accounts whitelisted.	**Submitting a SQL Job Using an SDK** **Submitting a Spark Job Using an SDK** **Submitting a Flink SQL Job Using an SDK** **Submitting a Flink Jar Job Using an SDK**

NOTE

The DLI SDK calls APIs using HTTPS, with certificates utilized by the server.

DLI SDK Introduction

DLI SDKs encapsulate the RESTful APIs provided by DLI to simplify development. You can directly call the API functions offered by DLI SDKs to submit both DLI SQL and DLI Spark jobs.

DLI SDKs consist of both the SDK V3 and other SDKs developed by DLI.

- (Recommended)DLI SDK V3: automatically generates APIs from the YAML files defining the APIs, ensuring parameter consistency with the service's own APIs.

  For details, see **SDK V3**.
- DLI SDK (self-developed): SDKs developed by the DLI team. This manual introduces the usage of DLI's self-developed SDKs.
  - Operation guidance of DLI SDK V2:
    - For details about how to use the Java SDK, see **Java SDK (DLI SDK V2)**.
    - For details about how to use the Python SDK, see **Python SDK (DLI SDK V2)**.

- Operation guidance of DLI SDK V1 (not recommended):
  - For details about how to use the Java SDK, see **Java SDK (DLI SDK V1)**.
  - For details about how to use the Python SDK, see **Python SDK (DLI SDK V1)**.

NOTE

The DLI SDK calls APIs using HTTPS, with certificates utilized by the server.

2 Configuring the Java SDK Environment

2.1 Preparing a Java Development Environment

Scenario

Before installing and using Java SDKs, make sure you have completed the basic configuration of your development environment.

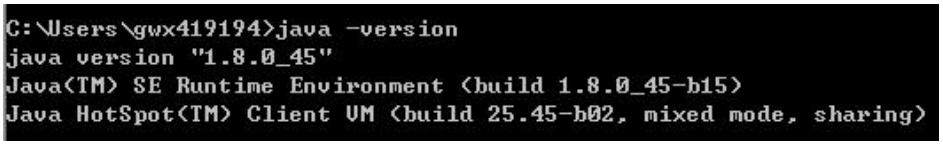
Java SDKs require JDK 1.8 or later. To ensure compatibility with future versions, you are advised to use version 1.8.

Once the Java runtime environment is configured, you can open the Windows command line and execute the command **Java -version** to check the version information.

Procedure

1. Install the JDK. Download the JDK 1.8 installation package from **Oracle official website** and install it.
2. Set environment variables. Specifically, choose **Start** > **Control Panel**. Click **System** > **Advanced system settings**. In the displayed **System Properties** dialog box, click **Advanced** and then click **Environment Variables** to switch to the **Environment Variables** dialog box.
3. Create the system variable **JAVA_HOME** with **Value** set to the JDK installation path, for example, **D:\Java\jdk1.8.0_45**.
4. Edit the **Path** variable and add **%JAVA_HOME%\bin** to **Variable value**.
5. Create the **CLASSPATH** variable with **Variable value** set to **.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar**.
6. Check whether the configurations succeed. Specifically, at the Start menu, enter **cmd** and press **Enter** to bring up the command prompt window. Enter ***java -version*** and press **Enter**. If the version information is displayed, as shown in **Figure 2-1**, Python is successfully installed and configured.

Figure 2-1 Checking the configuration



2.2 Obtaining and Installing the Java SDK

Obtaining DLI SDKs

On the DLI management console, download the SDK installation package **dli-sdk-java-*x.x.x*.zip** and extract it. The directory structure after extraction is shown in the table below.

Table 2-1 Directory structure

Parameter	Description
jars	SDK and its dependent JAR packages.
maven-install	Script and JAR package that are installed in the local Maven repository.
dli-sdk-java.version	Java SDK version description.

Installing the SDK

- **Method 1: Adding the SDK using the Maven central repository**

  The **Maven central repository** is part of the Apache Maven project that provides Java libraries and frameworks.

  When the SDK retrieval method is not specified, the default approach is to add the SDK driver using the Maven central repository.

  Use Maven to add the Maven configuration items on which **huaweicloud-dli-sdk** depends. (This is the default operation and does not need to be separately configured.)

  ```
  <dependency>
    <groupId>com.huawei.dli</groupId>
    <artifactId>huaweicloud-dli-sdk-java</artifactId>
    <version>x.x.x</version>
  </dependency>
  ```

- **Method 2: Obtaining the SDK using Maven to configure the Huawei image source**

  When using Maven to manage project dependencies, you can modify the **settings.xml** file to configure the Huawei image source to obtain the SDK.

  ```
  <mirror>
    <id>huaweicloud</id>
    <mirrorOf>*</mirrorOf>
    <url>https://mirrors.huaweicloud.com/repository/maven/</url>
  </mirror>
  ```

- **Method 3: Downloading the JDBC driver file from the DLI management console**

a. Log in to the DLI management console.

b. Click **SDK Download** in the **Common Links** area on the right of the **Overview** page.

c. On the **DLI SDK DOWNLOAD** page, select a driver and download it.

Click **huaweicloud-dli-sdk-java-*x.x.x*** to download a JDBC driver package.

NOTE

The JDBC driver package is named **huaweicloud-dli-sdk-java-*<version>*.zip**. It can be used in all versions of all platforms (such as Linux and Windows) and depends on JDK 1.7 or later.

d. The downloaded JDBC driver package contains **.bat** (Windows) or **.sh** (Linux/Mac) scripts, which are used to automatically install the SDK driver to the local Maven repository.

You can choose a script based on your OS to install the JDBC driver.

- Windows: Double-click the **.bat** file or run the file in the CLI.
- Linux/Mac: Run the **.sh** script.

2.3 Initializing the DLI Client

To use DLI SDK to access DLI, you need to initialize the DLI client. You can use either AK/SK or token-based authentication to initialize the client. Note that token-based authentication is only supported in DLI SDK V1, and it is advisable to use AK/SK-based authentication.

Prerequisites

- You have configured the Java SDK environment by referring to **Overview**.
- You have initialized the DLI client by referring to **Initializing the DLI Client**.

Example Code for AK/SK Authentication

- **Sample code**

```
String ak = System.getenv("xxx_SDK_AK");//Access key ID
String sk = System.getenv("xxx_SDK_SK");//Key used together with the access key ID
String regionName = "regionname";
String projectId = "project_id";
DLIInfo dliInfo = new DLIInfo(regionName, ak, sk, projectId);
DLIClient client = new DLIClient(AuthenticationMode.AKSK, dliInfo);
```

- **Parameter description and acquisition method**
  - Parameter description
    - **ak**: Account access key
    - **sk**: Account secret access key

NOTE

Hard coding AKs and SKs or storing them in code in plaintext poses significant security risks. You are advised to store them in encrypted form in configuration files or environment variables and decrypt them when needed to ensure security.

In this example, the AK and SK stored in the environment variables are used. Specify the environment variables *xxx*_**SDK_AK** and *xxx*_**SDK_SK** in the local environment first.

  - **regionName**: Region name
  - **projectId**: Project ID
- You can perform the following operations to obtain the **Access Keys**, **project ID**, and **Region**:
  i. Log in to the management console.
  ii. Hover over the username in the upper right corner and choose **My Credentials** from the drop-down list.
  iii. In the navigation pane on the left, choose **Access Keys**. On the displayed page, click **Create Access Key**. Confirm that you want to proceed with the operation and click **OK**.
  iv. On the displayed page, click **Download**. Open the file to obtain the AK/SK information.
  v. In the navigation pane on the left, choose **API Credentials**. In the **Projects** pane, locate **project_id** and obtain the region information.

Sample code for token-based authentication

- Sample code

```
String domainName = "domainname";
String userName = "username";
String password = "password";
String regionName = "regionname";
String projectId = "project_id";
DLIInfo dliInfo = new DLIInfo(regionName, domainName, userName, password, projectId);
DLIClient client = new DLIClient(AuthenticationMode.TOKEN, dliInfo);
```

- Parameter description
  - **domainname**: Account name
  - **username**: Username
  - **password**: User password
  - **regionname**: Region name
  - **project_id**: Project ID

NOTE

- Hard coding passwords or storing them in code in plaintext poses significant security risks. You are advised to store them in encrypted form in configuration files or environment variables and decrypt them when needed to ensure security.
- You can change the endpoint in **set** mode. Run the following statement: **dliInfo.setServerEndpoint(endpoint)**.

3 Configuring the Python SDK Environment

Scenario

If you are preparing for secondary development, you need to deploy the environment as required in **Table 3-1**.

Table 3-1 Development environment

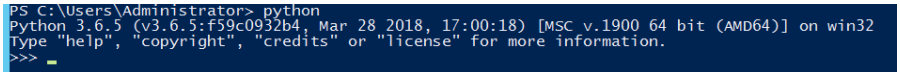
Item	Description
Operating system (OS)	Windows OS. Windows 7 or later is recommended.
Python	Python 2.7.10, 3.4.0, or later versions are required. The Visual C++ compilation environment is required, and Visual C++ build tools or Visual Studio must be installed.
Python dependencies	The DLI Python SDK dependencies include urllib3 1.15 or later, six 1.10 or later, certifi, and python-dateutil.

Procedure

Step 1 Download Python from **Python's official website** and install it.

1. Install Python according to the Python official guidelines.
2. Check whether the configurations succeed. Specifically, at the **Start** menu, enter **cmd** and press **Enter** to bring up the command prompt window. Enter **python** and press **Enter**. If the version information is displayed, as shown in **Figure 3-1**, Python is successfully installed and configured.

Figure 3-1 Checking the configuration



NOTE

If an error similar to "error: Microsoft Visual C++ xx.x is required. Get it with Build Tools for Visual Studio" is reported when you install the Python application package, the C++ compiler may not been installed. Install the Visual Studio compiler of the required version. For some OSs, restart the system after Visual Studio is installed.

Step 2 Install the DLI Python SDK.

1. Select the installation package obtained in **Obtaining and Installing Python SDKs** and decompress it.

   Decompress **dli-sdk-python-<version>.zip** to a local directory that can be adjusted.

2. Install the SDK.

   a. On a PC running the Windows OS, choose **Start** > **Run**, enter **cmd**, and press **Enter**.

   b. In the command-line interface (CLI) window, access the **windows** directory in the directory where **dli-sdk-python-<version>.zip** is decompressed. For example, **D:\tmp\dli-sdk-python-1.0.8**.

   c. Run the following command to install DLI Python SDK (During the installation, the third-party dependencies are automatically downloaded):

   python setup.py install

   Figure 3-2 shows the installation result.

Figure 3-2 Installing Python SDK

```
D:\tmp\dli-sdk-python-1.0.8>python setup.py install
running install
running bdist_egg
running egg_info
creating dli_sdk_python.egg-info
writing dli_sdk_python.egg-info\PKG-INFO
writing dependency_links to dli_sdk_python.egg-info\dependency_links.txt
writing requirements to dli_sdk_python.egg-info\requires.txt
writing top-level names to dli_sdk_python.egg-info\top_level.txt
writing manifest file 'dli_sdk_python.egg-info\SOURCES.txt'
reading manifest file 'dli_sdk_python.egg-info\SOURCES.txt'
writing manifest file 'dli_sdk_python.egg-info\SOURCES.txt'
installing library code to build\bdist.win-amd64\egg
```

----End

3.1 Preparing a Python Development Environment

Scenario

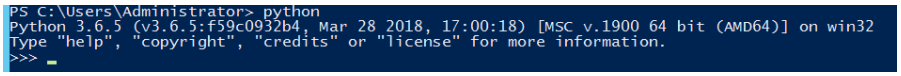
Before installing and using Python SDKs, make sure you have completed the basic configuration of your development environment.

Python 2.7.10, 3.4.0, or later versions are required. The Visual C++ compilation environment is required, and Visual C++ build tools or Visual Studio must be installed.

Procedure

1. Download Python from **Python's official website** and install it.
2. Install Python according to the Python official guidelines.
3. Check whether the configurations succeed. Specifically, at the **Start** menu, enter **cmd** and press **Enter** to bring up the command prompt window. Enter **python** and press **Enter**. If the version information is displayed, as shown in **Figure 3-3**, Python is successfully installed and configured.

   Figure 3-3 Checking the configuration

NOTE

If an error similar to "error: Microsoft Visual C++ xx.x is required. Get it with Build Tools for Visual Studio" is reported when you install the Python application package, the C++ compiler may not been installed. Install the Visual Studio compiler of the required version. For some OSs, restart the system after Visual Studio is installed.

3.2 Obtaining and Installing Python SDKs

Installing Python SDKs

This section provides instructions on how to install Python SDKs.

Obtaining DLI SDKs

On the displayed **DLI SDK DOWNLOAD** page, click the target link to obtain the desired SDK installation package.

Obtain the **dli-sdk-python-*x.x.x*.zip** package and decompress it. The following table shows the directory structure of the package.

Table 3-2 Directory structure

Parameter	Description
dli	DLI SDK basic module in the Python environment
examples	Python sample code
pyDLI	Implementation interface of PyHive
setup.py	Python SDK installation script

Installing DLI Python SDKs

1. Download and decompress the SDK installation package.

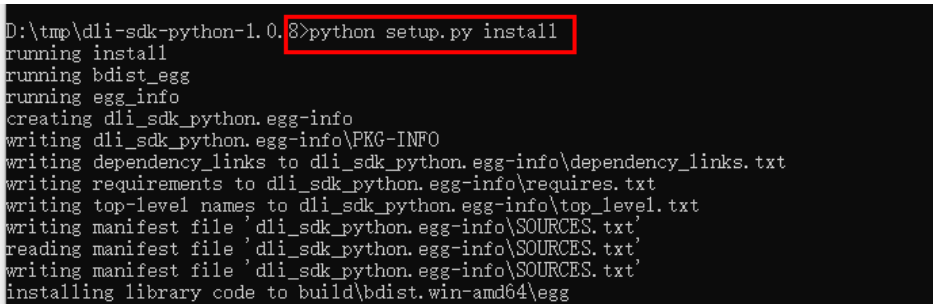
   Decompress **dli-sdk-python-<version>.zip** to a local directory that can be adjusted.

2. Install the SDK.
   a. On a PC running Windows, choose **Start** > **Run**, enter **cmd**, and press **Enter**.
   b. In the command-line interface (CLI) window, access the **windows** directory in the directory where **dli-sdk-python-<version>.zip** is decompressed. For example, **D:\tmp\dli-sdk-python-1.0.8**.
   c. Run the following command to install DLI Python SDK (During the installation, the third-party dependencies are automatically downloaded):

      python setup.py install

      Figure 3-4 shows the installation result.

      Figure 3-4 Installing Python SDK



3.3 Initializing the DLI Client

To use DLI Python SDK to access DLI, you need to initialize the DLI client. You can use either AK/SK or token-based authentication to initialize the client. Note that token-based authentication is only supported in DLI SDK V1, and it is advisable to use AK/SK-based authentication.

For details about the dependencies and complete sample code, see **Overview**.

Example Code for AK/SK Authentication

- Sample code

```
def init_aksk_dli_client():
    auth_mode = 'aksk'
    region = 'xxx'
    project_id = 'xxxx'
    ak = System.getenv("xxx_SDK_AK")//Access key ID
    sk = System.getenv("xxx_SDK_SK")//Key used together with the access key ID
    dli_client = DliClient(auth_mode=auth_mode, region=region, project_id=project_id,ak=ak, sk=sk)
    return dli_client
```

- Parameter description and acquisition method
  - Parameter description
    - **ak**: Account access key
    - **sk**: Account secret access key

NOTE

Hard coding AKs and SKs or storing them in code in plaintext poses significant security risks. You are advised to store them in encrypted form in configuration files or environment variables and decrypt them when needed to ensure security.

In this example, the AK and SK stored in the environment variables are used. Specify the environment variables *xxx*_**SDK_AK** and *xxx*_**SDK_SK** in the local environment first.

- **regionName**: Region name
- **projectId**: Project ID

– You can perform the following operations to obtain the **Access Keys**, **project ID**, and **Region**:

i. Log in to the management console.

ii. Hover over the username in the upper right corner and choose **My Credentials** from the drop-down list.

iii. In the navigation pane on the left, choose **Access Keys**. On the displayed page, click **Create Access Key**. Confirm that you want to proceed with the operation and click **OK**.

iv. On the displayed page, click **Download**. Open the file to obtain the AK/SK information.

v. In the navigation pane on the left, choose **API Credentials**. In the **Projects** pane, locate **project_id** and obtain the region information.

Example Code for Token-based Authentication

- Sample code

```
def init_token_dli_client():
   auth_mode = 'token'
    region = 'xxx'
    project_id = 'xxxx'
    account = 'xxx account'
    user = 'xxxx'
    password = 'xxxx'
    dli_client = DliClient(auth_mode=auth_mode, region=region, project_id=project_id,account=account,
user=user, password=password)
    return dli_client
```

- Parameter description
  - **domainname**: Account name
  - **username**: Username
  - **password**: User password
  - **regionName**: Region name
  - **project_id**: Project ID

NOTE

- Hard coding passwords or storing them in code in plaintext poses significant security risks. You are advised to store them in encrypted form in configuration files or environment variables and decrypt them when needed to ensure security.
- You can change the endpoint in **set** mode. Run the following statement: **dliInfo.setServerEndpoint(endpoint)**.

4 General SDK V3

DLI SDKs encapsulate the RESTful APIs provided by DLI to simplify development.

This section describes how to use the general SDK V3 and lists the addresses for obtaining the latest SDKs.

SDK V3 Developer Guide

Generating SDK Code Online

[Example]

Use API Explorer to dynamically generate SDK code.

You can view the SDK code of a range of programming languages in the **Sample Code** tab of an API.

5 DLI SDK V2

5.1 Java SDK (DLI SDK V2)

5.1.1 Submitting a SQL Job Using an SDK

This section describes how to submit a SQL job using DLI SDK V2.

NOTE

Starting May 2024, new users can directly use DLI's SDK V2 without needing to have their accounts whitelisted.

For users who started using DLI before May 2024, to use this function, they must submit a service ticket to have their accounts whitelisted.

Prerequisites

- You have configured the Java SDK environment by referring to **Overview**.
- You have initialized the DLI client by referring to **Initializing the DLI Client**.

Preparations

Obtain an AK/SK, project ID, and region information.

1. Log in to the management console.
2. In the upper right corner, hover over the username and choose **My Credentials** from the drop-down list.
3. In the navigation pane on the left, choose **Access Keys**. On the displayed page, click **Create Access Key**. Confirm that you want to proceed with the operation and click **OK**.
4. On the displayed page, click **Download**. Open the file to obtain the AK/SK information.
5. In the navigation pane on the left, choose **API Credentials**. In the **Projects** pane, locate **project_id** and obtain the region information.

Example Code

```
    private static final Logger logger = LoggerFactory.getLogger(SqlJobExample.class);
    private static final ThreadLocal<DateFormat> DATE_FORMAT = ThreadLocal.withInitial(
        () -> new SimpleDateFormat("yyyy-MM-dd"));

    private static final ThreadLocal<DateFormat> TIMESTAMP_FORMAT = ThreadLocal.withInitial(
        () -> new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSSZZ"));

    public static void main(String[] args) {
        String yourAccessKey = System.getenv("HUAWEICLOUD_SDK_AK");
        String yourSecretKey = System.getenv("HUAWEICLOUD_SDK_SK");
        DLIInfo dliInfo = new DLIInfo("RegionName", yourAccessKey, yourSecretKey, "YouProjectId");
        dliInfo.setQueueName("YourQueueName");

        try {
            // Step 1: Create a database and a table.
            prepare(dliInfo);

            /*
             * Step 2: Import data to the table.
             * The overall implementation process/ principle can be divided into the following three steps:
             * 1. Use the OBS API to upload data to YourOBSPathToWriteTmpData. You can configure a lifecycle policy in OBS to periodically delete these temporary data.
             * 2. Submit the LoadData statement to DLI to import data to DLI. For details, see Importing Data.
             * 3. Cyclically check the job status every second until the job is complete.
             */
            String yourOBSPathToWriteTmpData = String.format("obs://your_obs_bucket_name/your/path/%s", UUID.randomUUID());
            loadData(dliInfo, yourOBSPathToWriteTmpData);

            // Step 3: Submit the SQL statement, execute the query, and read the result.
            String selectSql = "SELECT * FROM demo_db.demo_tbl";
            String jobId = queryData(dliInfo, selectSql);

            // Step 4: If needed, you can also obtain the results by job ID.
            queryDataByJobId(dliInfo, jobId);

            // Query all jobs by page. You can use this API to query information of all SQL jobs within the current project.
            // Key SDK API: com.huaweicloud.sdk.dli.v1.DliClient#listSqlJobs(ListSqlJobsRequest).
            listSqlJobs(dliInfo).

            /*
             * Other scenarios:
             * 1. To cancel a submitted SQL job, use the following API.
             * Key SDK API: com.huaweicloud.sdk.dli.v1.DliClient#cancelSqlJob(CancelSqlJobRequest).
             * Note: If a job has been completed or failed, it cannot be canceled.
             * 2. To verify the syntax of an SQL statement, use the following API.
             * Key SDK API: com.huaweicloud.sdk.dli.v1.DliClient#checkSql(CheckSqlRequest).
             * Note: This API can only be used to verify the syntax, not the semantics. Use the Explain statement and submit it to DLI for execution to perform semantic verification.
             * 3. To obtain a submitted SQL job based on the job ID and view job details, use the following API.
             * Key SDK API: com.huaweicloud.sdk.dli.v1.DliClient#showSqlJobDetail(ShowSqlJobDetailRequest).
             * 4. Obtain the job execution progress. If the job is being executed, you can obtain the sub-job information. If the job has just started or has been completed, you cannot obtain the sub-job information.
             * Key SDK API: com.huaweicloud.sdk.dli.v1.DliClient#showSqlJobProgress(ShowSqlJobProgressRequest).
             */
        } catch (DLIException e) {
            // Handle the exception based on service requirements. The following is just an example.
        }
    }
```

Creating a Database and Table

Reference link:

- **Creating a Database**
- **Creating a Table**
- Key SDK API: **com.huawei.dli.sdk.Job#submit()**

Sample code:

```
private static String prepare(DLIInfo dliInfo) throws DLIException {
    // 1. Create a database.
    // default is the database built in DLI. You cannot create a database named default.
    String createDbSql = "CREATE DATABASE IF NOT EXISTS demo_db";
    new SQLJob(dliInfo, createDbSql).submit();
    // 2. Create a table. Note: Adjust the table structure, table data directory, and OBS storage path based on site requirements.
    String createTblSql = "CREATE TABLE IF NOT EXISTS `demo_tbl` (\n"
        + "  `bool_c` BOOLEAN,\n"
        + "  `byte_c` TINYINT,\n"
        + "  `short_c` SMALLINT,\n"
        + "  `int_c` INT,\n"
        + "  `long_c` BIGINT,\n"
        + "  `float_c` FLOAT,\n"
        + "  `double_c` DOUBLE,\n"
        + "  `dec_c` DECIMAL(10,2),\n"
        + "  `str_c` STRING,\n"
        + "  `date_c` DATE,\n"
        + "  `ts_c` TIMESTAMP,\n"
        + "  `binary_c` BINARY,\n"
        + "  `arr_c` ARRAY<INT>,\n"
        + "  `map_c` MAP<STRING, INT>,\n"
        + "  `struct_c` STRUCT<`s_str_c`: STRING, `s_bool_c`: BOOLEAN>)\n"
        + "USING parquet OPTIONS(path 'obs://demo_bucket/demo_db/demo_tbl')";
    new SQLJob(dliInfo, "demo_db", createTblSql).submit();
}
```

Importing Data

- **To import data using a DLI SDK, follow these 3 steps:**
  a. Use the OBS API to upload data to a temporary OBS directory, that is, **YourOBSPathToWriteTmpData**.
  b. Submit the LoadData statement to DLI to import data from the temporary OBS directory to DLI. **Importing Data**
  c. Cyclically check the job status every 1 second until the job is complete.
- **Data import description:**
  – Before submitting a data import job, you can configure the partition for importing data and specify whether to overwrite existing data (data is appended by default).
    - To insert data into a specific partition, use the following constructor:
      ```
      new SqlJobBase.TableInfo("demo_db", "demo_tbl", new LinkedHashMap<String,String>()
      {{put("YourPartitionColumnName","value");}});
      ```
    - By default, an import job appends data to the existing data. To overwrite the existing data with the written data, use the following constructor:
      ```
      new SqlJobBase.TableInfo("demo_db", "demo_tbl", true);
      ```
    - To overwrite the partition data, use the following constructor:
      ```
      new SqlJobBase.TableInfo("demo_db", "demo_tbl", new LinkedHashMap<String,String>()
      {{put("YourPartitionColumnName","value");}}, true);
      ```
  – If a folder and a file with the same name exist in an OBS bucket directory, data is preferentially loaded to the file rather than the folder at

that path. You are advised not to have files and folders with the same name at the same level when creating an OBS object.

- **Reference links**:
  - **Importing Data**
  - **Native Data Types**
  - **Complex Data Types**
- **Key SDK APIs**:
  - com.huawei.dli.sdk.write.Writer
  - com.huawei.dli.sdk.Job#asyncSubmit()
  - com.huaweicloud.sdk.dli.v1.DliClient#showSqlJobStatus
- **Sample code**:
  - (Recommended) Solution 1: Use the LoadData statement to import data.

```
private static void loadData(DLIInfo dliInfo, String uploadDataPath) throws DLIException {
    UploadJob uploadJob = new UploadJob(
        dliInfo, uploadDataPath, new SqlJobBase.TableInfo("demo_db", "demo_tbl"));

    // 1. Write data to the OBS temporary directory. Modify the following information based on site requirements. The following is just an example.
    // Note: This step involves directly calling the OBS data writing API. DLI only provides a default implementation for writing data in JSON format, meaning that files are stored on OBS in JSON format.
    // The writer here can be implemented based on service requirements. For example, you use a custom CSV writer, resulting in files being stored in CSV format on OBS.
    writeTmpData(uploadJob.createWriter(), genSchema(), 123, 50);

    // 2. Import data to DLI.
    // Submit the LoadData statement.
    // Note: The data_type here needs to be determined based on the writer implementation in step 1. By default, DLI provides JSON. If a custom writer is used, it should be modified to match the corresponding data_type.
    String loadSql =
        "LOAD DATA INPATH '" + uploadDataPath + "' INTO TABLE demo_db.demo_tbl OPTIONS(data_type 'json')";
    SQLJob sqlJob = new SQLJob(dliInfo, loadSql);
    sqlJob.asyncSubmit();

    // 3. Cyclically check the job status.
    checkRunning(V3ClientUtils.getDliClient(dliInfo), sqlJob.getJobId());
}
```

  - Solution 2: Use the DLI encapsulated SDK to submit data for import.

```
private static void loadData(DLIInfo dliInfo, String uploadDataPath) throws DLIException {
    UploadJob uploadJob = new UploadJob(
        dliInfo, uploadDataPath, new SqlJobBase.TableInfo("demo_db", "demo_tbl"));

    // 1. Write data to the OBS temporary directory. Modify the following information based on site requirements. The following is just an example.
    // Note: This step involves directly calling the OBS data writing API. DLI only provides a default implementation for writing data in JSON format, meaning that files are stored on OBS in JSON format.
    // The writer here can be implemented based on service requirements. For example, you use a custom CSV writer, resulting in files being stored in CSV format on OBS.
    writeTmpData(uploadJob.createWriter(), genSchema(), 123, 50);

    // 2. Import data to DLI.
    // Submit data using DLI encapsulation. Note: Since the import job may take a long time to run, use asyncSubmit to submit data and proactively check the status.
    uploadJob.asyncSubmit();

    // 3. Cyclically check the job status.
    checkRunning(V3ClientUtils.getDliClient(dliInfo), uploadJob.getJobId());
}
```


Querying Job Results

- **Reference link**:

  SELECT Query Statement

- **Key SDK API**:

  com.huawei.dli.sdk.read.ResultSet: API calls related to reading data from OBS. DLI provides a default OBS CSV reader implementation, which can be customized based on service requirements.

  com.huawei.dli.sdk.SQLJob#submitQuery(): The feature of writing results to the job bucket must be enabled; otherwise, only the first 1,000 data records are previewed by default.

  You can determine if the feature is enabled by checking the **result_path** parameter in the response body of the API for querying job status.

  After the job is completed, if **result_path** starts with **obs://**, the feature of writing job results to the job bucket is enabled; otherwise, it is not enabled.

```
private static String queryData(DLIInfo dliInfo, String selectSql)
    throws DLIException {
    SQLJob sqlJob = new SQLJob(dliInfo, selectSql);
    // If needed, you can set job parameters here, such as sqlJob.setConf().
    // 1. Submit a query job to DLI, implement submission using DLI encapsulation, and await the result.
    // Note 1: Set the timeout duration here based on the SQL execution duration, with a default of 5 minutes.
    // Note 2: The feature of writing job results to the job bucket needs to be enabled here. Otherwise, only the first 1,000 data records are previewed by default.
    sqlJob.setJobTimeout(30 * 60);
    ResultSet resultSet1 = null;
    try {
        resultSet1 = sqlJob.submitQuery();
        handleResult(resultSet1);
    } finally {
        if (resultSet1 != null) {
            resultSet1.close();
        }
    }
    return sqlJob.getJobId();
}
```

Querying the Result of a Specified Job

- **Instructions**

  com.huawei.dli.sdk.SQLJob#getResultSet(): API calls related to reading data from OBS. DLI provides an OBS CSV reader implementation, which can be customized based on service requirements.

  To use this method, you need to enable the feature of writing job results to the job bucket. You can determine if the feature is enabled by checking the **result_path** parameter in the response body of the API for querying job status. After the job is completed, if **result_path** starts with **obs://**, the feature of writing job results to the job bucket is enabled; otherwise, it is not enabled.

- **Reference links**:

  SQL Syntax Reference - Basic Statements

  OBS Documentation

- **Sample code**

```
private static void queryDataByJobId(DLIInfo dliInfo, String jobId)
    throws DLIException {
```



```
        // Check whether the job corresponding to the job ID has completed. If not, wait until the job completes.
        SQLJob sqlJob = new SQLJob(dliInfo, null);
        sqlJob.setJobId(jobId);
        checkRunning(V3ClientUtils.getDliClient(dliInfo), jobId);

        // Retrieve the schema of the result data and the storage path of the result data in the user's job bucket based on the job ID.
        ShowSqlJobStatusResponse resp = V3ClientUtils.getDliClient(dliInfo)
            .showSqlJobStatus(new ShowSqlJobStatusRequest().withJobId(jobId));
        sqlJob.setJobStatus(resp.getStatus().getValue());
        sqlJob.setResultSchema(SchemaUtils.getSchemaFromJson(resp.getDetail()));
        sqlJob.setResultPath(resp.getResultPath());
        sqlJob.setResultCount(resp.getResultCount() != null ? resp.getResultCount() : 0);

        ResultSet resultSet = null;
        try {
            // Obtain the query result corresponding to the job ID and return the result iterator.
            resultSet = sqlJob.getResultSet();
            handleResult(resultSet);
        } finally {
            if (resultSet != null) {
                resultSet.close();
            }
        }
    }
```

Querying the Job List

- **Instructions**

  com.huaweicloud.sdk.dli.v1.DliClient#listSqlJobs(ListSqlJobsRequest).

  If there are a large number of jobs, you must use the following pagination query method to query jobs in batches. Otherwise, only the jobs on the first page are returned.

  You can use **req.setStart()** and **req.setEnd()** to query jobs within a specified time period, with the unit being milliseconds.

- **Sample code**

```
    private static void listSqlJobs(DLIInfo dliInfo) {
        DliClient dliClient = V3ClientUtils.getDliClient(dliInfo);
        ListSqlJobsRequest req = new ListSqlJobsRequest();
        int currentPage = 1;
        req.setCurrentPage(currentPage);  // The default value is 1.
        req.setPageSize(100);  // The default value is 10.
        Integer jobCount = dliClient.listSqlJobs(req).getJobCount();
        Integer cur = 0;

        // Query jobs by page.
        while (cur < jobCount) {
            ListSqlJobsResponse listSqlJobsResponse = dliClient.listSqlJobs(req);
            List<SqlJob> jobs = listSqlJobsResponse.getJobs();
            for (SqlJob job : jobs) {

                // Add the service logic here to process each job.

                cur++;
                if (cur.equals(jobCount)) {
                    break;
                }
            }
            req.setCurrentPage(currentPage++);
        }
    }
```


Writing Data to OBS by Running writeTmpData

- **Instructions**

  You can implement the writer interface and customize the file writing logic based on your service requirements.

  This example writes data to OBS by calling the OBS SDK. Currently, UploadJob offers an implementation where data is written in JSON format by default.

- **Sample code**

```
    private static void writeTmpData(Writer writer, List<Column> schema, Integer totalRecords, Integer flushThreshold)
        throws DLIException {
        // Define a writer for writing data to OBS. You can define multiple writers based on the concurrency needs.
        // Define the column information to be written based on the target table. For details, see the genSchema() and genSchema2() methods in this document.
        // Define the loop size based on the actual service.
        for (int i = 0; i < totalRecords; i++) {
            // Retrieve data for each row based on the service.
            List<Object> record = genRecord();
            // Write data.
            Row row = new Row(schema);
            row.setRecord(record);
            writer.write(row);
            if (i % flushThreshold == 0) {
                // Refresh the data promptly after writing a certain amount of data.
                writer.flush();
            }
        }
        writer.close();
    }
```

Creating a Schema for a Table

- **Instructions**

  Construct the schema based on the actual service. The following is just an example.

- **Sample code**

```
    private static List<Column> genSchema() {
        return Arrays.asList(
            new Column("bool_c", new PrimitiveType(DataType.TypeName.BOOLEAN), "boolean col"),
            new Column("byte_c", new PrimitiveType(DataType.TypeName.TINYINT), "tinyint col"),
            new Column("short_c", new PrimitiveType(DataType.TypeName.SMALLINT), "smallint col"),
            new Column("int_c", new PrimitiveType(DataType.TypeName.INT), "int col"),
            new Column("long_c", new PrimitiveType(DataType.TypeName.BIGINT), "bigint col"),
            new Column("float_c", new PrimitiveType(DataType.TypeName.FLOAT), "float col"),
            new Column("double_c", new PrimitiveType(DataType.TypeName.DOUBLE), "double col"),
            new Column(
                "dec_c",
                new PrimitiveType(DataType.TypeName.DECIMAL, Arrays.asList("10", "2")),
                "decimal col"),
            new Column("str_c", new PrimitiveType(DataType.TypeName.STRING), "string col"),
            new Column("date_c", new PrimitiveType(DataType.TypeName.DATE), "date col"),
            new Column("ts_c", new PrimitiveType(DataType.TypeName.TIMESTAMP), "timestamp col"),
            new Column("binary_c", new PrimitiveType(DataType.TypeName.BINARY), "binary col"),
            new Column(
                "arr_c",
                new ArrayType(new PrimitiveType(DataType.TypeName.INT)),
                "array col"),
            new Column(
                "map_c",
                new MapType(
                    new PrimitiveType(DataType.TypeName.STRING),
                    new PrimitiveType(DataType.TypeName.INT)),
                "map col"),
```



```
            new Column(
                "struct_c",
                new StructType(Arrays.asList(
                    new Column("s_str_c", new PrimitiveType(DataType.TypeName.STRING), "struct string
col"),
                    new Column("s_bool_c", new PrimitiveType(DataType.TypeName.BOOLEAN), "struct
boolean col"))),
                "struct col"));
    }
```

Automatically Obtaining the Schema of the Target Table

- **Instructions**

  Automatically obtain the schema of the target table.

- **Sample code**

```
    private static List<Column> genSchema2(DLIInfo dliInfo, String yourDbName, String
yourTableName)
        throws DLIException {
        String tempSql = String.format("select * from %s.%s limit 1", yourDbName, yourTableName);
        SQLJob sqlJob = new SQLJob(dliInfo, tempSql);
        sqlJob.submit();

        ShowSqlJobStatusResponse resp = V3ClientUtils.getDliClient(dliInfo)
            .showSqlJobStatus(new ShowSqlJobStatusRequest().withJobId(sqlJob.getJobId()));
        if (!resp.getIsSuccess()) {
            throw new DLIException("Get sql job status failed, details: " + resp.getMessage());
        }
        return SchemaUtils.getSchemaFromJson(resp.getDetail());
    }
```

Generating Test Data List<Object> genRecord on Demand

- **Instructions**

  Construct each row of data based on service requirements. The following is just an example.

- **Sample code**

```
    private static List<Object> genRecord() {
        Map<String, Object> structData = new HashMap<>();
        structData.put("s_str_c", "Abc");
        structData.put("s_bool_c", true);
        return Arrays.asList(
            true,
            (byte) 1,
            (short) 123,
            65535,
            123456789012L,
            101.235f,
            256.012358,
            new BigDecimal("33.05"),
            "abc_123&",
            new Date(1683475200000L),
            new Timestamp(1683543480000L),
            "Hello".getBytes(StandardCharsets.UTF_8),
            Arrays.asList(1, 2, 3),
            Collections.singletonMap("k", 123),
            structData);
    }
```

Querying the Job Status

- **Sample code**

```
    private static void checkRunning(DliClient dliClient, String jobId) throws DLIException {
        while (true) {
            ShowSqlJobStatusResponse resp;
```



```
            try {
                resp = dliClient.showSqlJobStatus(new ShowSqlJobStatusRequest().withJobId(jobId));
            } catch (Exception e) {
                throw new DLIException("Failed to get job status by id: " + jobId, e);
            }
            String status = resp.getStatus().getValue();
            logger.info(String.format("SparkSQL Job id %s status: %s", jobId, status));

            if ("FINISHED".equals(status)) {
                return;
            }
            if ("FAILED".equals(status) || "CANCELLED".equals(status)) {
                throw new DLIException("Run job failed or cancelled, details: " + resp.getMessage());
            }

            try {
                Thread.sleep(1000L);
            } catch (InterruptedException e) {
                throw new DLIException("Check job running interrupted.");
            }
        }
    }
```

Processing Job Results

- **Instructions**

  Construct each row of data based on service requirements. The following is just an example.

- **Sample code**

```
    private static void handleResult(ResultSet resultSet) throws DLIException {
        while (resultSet.hasNext()) {
            Row row = resultSet.read();
            List<Column> schema = row.getSchema();
            List<Object> record = row.getRecord();
            // Process each row and each column.
            // Method 1: Call record.get(index) to get the Object, then perform type conversion according to the type of each column.
            // Method 2: Based on the type of each column, call row.getXXX(index) to obtain data of the corresponding type.
            for (int i = 0; i < schema.size(); i++) {
                DataType.TypeName typeName = DataType.TypeName.fromName(schema.get(i).getType().getName());
                switch (typeName) {
                    case STRING:
                        String strV = (String) record.get(i);
                        System.out.println(
                            "\t" + (strV == null ? null : strV.replaceAll("\r", "\\\\r").replaceAll("\n", "\\\\n")));
                        break;
                    case DATE:
                        Date dtV = (Date) record.get(i);
                        System.out.println("\t" + (dtV == null ? null : DATE_FORMAT.get().format(dtV)));
                        break;
                    case TIMESTAMP:
                        Timestamp tsV = (Timestamp) record.get(i);
                        System.out.println("\t" + (tsV == null ? null : TIMESTAMP_FORMAT.get().format(tsV)));
                        break;
                    case BINARY:
                        byte[] bytes = (byte[]) record.get(i);
                        System.out.println("\t" + (bytes == null ? null : Base64.getEncoder().encodeToString(bytes)));
                        break;
                    case ARRAY:
                    case MAP:
                    case STRUCT:
                        Object data = record.get(i);
                        System.out.println("\t" + (data == null ? null : JsonUtils.toJSON(data)));
```



```
                    break;
                default:
                    System.out.println("\t" + record.get(i));
            }
        }
        System.out.println();
    }
}
```

5.1.2 Submitting a Flink SQL Job Using an SDK

This section describes how to submit a Flink SQL job using DLI SDK V2.

NOTE

Starting May 2024, new users can directly use DLI's SDK V2 without needing to have their accounts whitelisted.

For users who started using DLI before May 2024, to use this function, they must submit a service ticket to have their accounts whitelisted.

Prerequisites

- You have configured the Java SDK environment by referring to **Overview**.
- You have initialized the DLI client by referring to **Initializing the DLI Client**.

Preparations

Obtain an AK/SK, project ID, and region information.

1. Log in to the management console.
2. In the upper right corner, hover over the username and choose **My Credentials** from the drop-down list.
3. In the navigation pane on the left, choose **Access Keys**. On the displayed page, click **Create Access Key**. Confirm that you want to proceed with the operation and click **OK**.
4. On the displayed page, click **Download**. Open the file to obtain the AK/SK information.
5. In the navigation pane on the left, choose **API Credentials**. In the **Projects** pane, locate **project_id** and obtain the region information.

Example Code

```
private static final Logger logger = LoggerFactory.getLogger(FlinkSqlJobExample.class);
public static void main(String[] args) {
    String yourAccessKey = System.getenv("HUAWEICLOUD_SDK_AK");
    String yourSecretKey = System.getenv("HUAWEICLOUD_SDK_SK");
    DliClient dliClient = DliClient.newBuilder()
        .withRegion(DliRegion.valueOf("RegionName"))
        .withCredential(new BasicCredentials()
            .withAk(yourAccessKey)
            .withSk(yourSecretKey)
            .withProjectId("YouProjectId"))
        .build();

    try {
        // Step 1: Create a Flink job. The job status changes to Draft.
        Long jobId = createFlinkSqlJob(dliClient, "YourQueueName");
        logger.info("jobId: " + jobId);
```



```
        // Step 2: Run the job. The job status changes from Draft to Submitting.
        List<FlinkSuccessResponse> resps = batchRunFlinkJobs(dliClient, Arrays.asList(jobId));
        logger.info("Response: " + ArrayUtils.toString(resps));

        // Step 3: Query the job status. If you wish to wait for the job to reach the Running state within the current thread, you can cyclically check the job status until it becomes Running.
        checkRunning(dliClient, jobId);

    } catch (DLIException e) {
        // Handle the exception based on service requirements. The following is just an example.
    }

}
```

Creating a Flink SQL Job

- **Function:**

  Create a Flink SQL job.

- **Reference link:**

  Key SDK API: **com.huaweicloud.sdk.dli.v1.DliClient#createFlinkSqlJob(com.huaweicloud.sdk.dli.v1.model.CreateFlinkSqlJobRequest)**

  Create a Flink SQL job. The job status changes to **Draft**.

- **Sample code**:

```
private static Long createFlinkSqlJob(DliClient client, String queueName) {
        // Set the parameters according to the actual situation. The following is just an example.
        CreateFlinkSqlJobResponse resp = client.createFlinkSqlJob(new CreateFlinkSqlJobRequest()
            .withBody(new CreateFlinkSqlJobRequestBody()
                .withName("demo_flink_sql") // Custom job name. The name can contain up to 57 characters.
                .withDesc("YourJobDescription") // Custom job description. The description can contain up to 512 characters.
                .withSqlBody("create table orders(\n"
                    + "  name string,\n"
                    + "  num INT\n"
                    + ") with (\n"
                    + "  'connector' = 'datagen',\n"
                    + "  'rows-per-second' = '1', \n"
                    + "  'fields.name.kind' = 'random', \n"
                    + "  'fields.name.length' = '5' \n"
                    + ");\n"
                    + "\n"
                    + "CREATE TABLE sink_table (\n"
                    + "   name string,\n"
                    + "   num INT\n"
                    + ") WITH (\n"
                    + "   'connector' = 'print'\n"
                    + ");\n"
                    + "\n"
                    + "INSERT into sink_table SELECT * from orders;")
                // Customize a stream SQL statement, which contains at least the following three parts: source, query, and sink. Length limit: 1024 x 1024 characters.
                // In this SQL statement, random source data is automatically generated and printed to the console.
                .withQueueName(queueName) // Queue name. The name can contain up to 128 characters.
                .withRunMode("exclusive_cluster") // Job running mode. Only the exclusive_cluster mode is supported.
                .withLogEnabled(true) // Enable the function of uploading job logs to OBS buckets.
                .withObsBucket("YourObsBucketName") // OBS bucket name, which is used to store logs and checkpoint data.
                .withJobType("flink_opensource_sql_job") // Job type. You are advised to select flink_opensource_sql_job.
                .withFlinkVersion("1.12") // Specify the Flink version.
            ));
```



```
        return resp.getJob().getJobId();
    }
```

Running Flink Jobs in Batches

- **Function:**

  Run Flink SQL jobs in batches.

- **Reference link:**

  Key SDK API:
  com.huaweicloud.sdk.dli.v1.DliClient#batchRunFlinkJobs(com.huaweicloud.sdk.dli.v1.model.BatchRunFlinkJobsRequest)

  Run Flink jobs in batches. The job status changes from **Draft** to **Submitting**.

- **Sample code**:

```
private static List<FlinkSuccessResponse> batchRunFlinkJobs(DliClient client, List<Long> jobIds) {
        BatchRunFlinkJobsResponse batchRunFlinkJobsResponse = client.batchRunFlinkJobs(
            new BatchRunFlinkJobsRequest()
                .withBody(new BatchRunFlinkJobsRequestBody().withJobIds(jobIds)));
        return batchRunFlinkJobsResponse.getBody();
    }
```

Querying the Job Status

- **Function:**

  Query the status of a Flink SQL job.

- **Reference link:**

  Key SDK API:
  com.huaweicloud.sdk.dli.v1.DliClient#showFlinkJob(com.huaweicloud.sdk.dli.v1.model.ShowFlinkJobRequest)}

  If you wish to wait for the job's transition into the **Running** state within the current thread, you can cyclically check the job status until it becomes **Running**.

- **Sample code**:

```
private static void checkRunning(DliClient client, Long jobId) throws DLIException {
        while (true) {
            ShowFlinkJobResponse resp;
            try {
                resp = client.showFlinkJob(new ShowFlinkJobRequest().withJobId(jobId));
            } catch (Exception e) {
                throw new DLIException("Failed to get Flink sql job status by id: " + jobId, e);
            }
            String status = resp.getJobDetail().getStatus();
            logger.info(String.format("FlinkSqlJob id %s status: %s", jobId, status));
            if ("job_running".equals(status)) {
                return;
            }
            if ("job_submit_fail".equals(status) || "job_running_exception".equals(status)) {
                throw new DLIException("Run Flink sql job failed: " + resp);
            }
            try {
                Thread.sleep(1000L);
            } catch (InterruptedException e) {
                throw new DLIException("Check job running interrupted.");
            }
        }
    }
```

5.1.3 Submitting a Flink Jar Job Using an SDK

This section describes how to submit a Flink Jar job using DLI SDK V2.

NOTE

Starting May 2024, new users can directly use DLI's SDK V2 without needing to have their accounts whitelisted.

For users who started using DLI before May 2024, to use this function, they must submit a service ticket to have their accounts whitelisted.

Prerequisites

- You have configured the Java SDK environment by referring to **Overview**.
- You have initialized the DLI client by referring to **Initializing the DLI Client**.

Preparations

Obtain an AK/SK, project ID, and region information.

1. Log in to the management console.
2. In the upper right corner, hover over the username and choose **My Credentials** from the drop-down list.
3. In the navigation pane on the left, choose **Access Keys**. On the displayed page, click **Create Access Key**. Confirm that you want to proceed with the operation and click **OK**.
4. On the displayed page, click **Download**. Open the file to obtain the AK/SK information.
5. In the navigation pane on the left, choose **API Credentials**. In the **Projects** pane, locate **project_id** and obtain the region information.

Example Code

```
private static final Logger logger = LoggerFactory.getLogger(FlinkJarJobExample.class);
public static void main(String[] args) {
        String yourAccessKey = System.getenv("HUAWEICLOUD_SDK_AK");
        String yourSecretKey = System.getenv("HUAWEICLOUD_SDK_SK");
        DliClient dliClient = DliClient.newBuilder()
            .withRegion(DliRegion.valueOf("RegionName"))  //
            .withCredential(new BasicCredentials()
                .withAk(yourAccessKey)
                .withSk(yourSecretKey)
                .withProjectId("YouProjectId"))
            .build();
        try {
            // Step 1: Create a Flink job. The job status changes to Draft.
            Long jobId = createFlinkJarJob(dliClient, "YourQueueName");
            logger.info("jobId: " + jobId);
            // Step 2: Run the job. The job status changes from Draft to Submitting.
            List<FlinkSuccessResponse> resps = batchRunFlinkJobs(dliClient, Arrays.asList(jobId));
            logger.info("Response: " + ArrayUtils.toString(resps));
            // Step 3: Query the job status. If you wish to wait for the job to reach the Running state within the current thread, you can cyclically check the job status until it becomes Running.
            checkRunning(dliClient, jobId);
        } catch (DLIException e) {
            // Handle the exception based on service requirements. The following is just an example.
        }
    }
```

Creating a Flink Jar Job

- **Function:**

  Create a Flink Jar job.

- **Reference link:**

  Key SDK API:
  com.huaweicloud.sdk.dli.v1.DliClient#createFlinkJarJob(com.huaweicloud.sdk.dli.v1.model.CreateFlinkJarJobRequest)

  Create a Flink Jar job. The job status changes to **Draft**.

- **Sample code**:

```
private static Long createFlinkJarJob(DliClient client, String queueName) {
        // Set the parameters according to the actual situation. The following is just an example.
        CreateFlinkJarJobResponse resp = client.createFlinkJarJob(new CreateFlinkJarJobRequest()
            .withBody(new CreateFlinkJarJobRequestBody()
                .withName("demo_flink_jar") // Custom job name. The name can contain up to 57 characters.
                .withDesc("YourJobDescription") // Custom job description. The description can contain up to 512 characters.
                .withQueueName(queueName) // Queue name. The name can contain up to 128 characters.
                .withFeature("basic") // Job feature. Type of the Flink image used by a job. basic: the base Flink image provided by DLI is used.
                .withFlinkVersion("1.12") // Flink version. This parameter is valid only when feature is set to basic.
                .withObsBucket("YourObsBucketName") // OBS bucket name, which is used to store logs and checkpoint data.
                .withLogEnabled(true) // Enable the function of uploading job logs to OBS buckets.
                .withEntrypoint("obs://YourObsBucketName/your/flink/job.jar") // JAR file uploaded to OBS that includes the user-defined job main class.
                .withMainClass("YourClassFullName") // Job entry class, for example, org.apache.flink.examples.JavaQueueStream.
                .withEntrypointArgs("YourAppArg1 YourAppAgr2") // Job entry parameter. Multiple parameters are separated by spaces. Delete this line if it is not required.
                .withDependencyJars(Arrays.asList("obs://YourObsBucketName/your/dependency1.jar",
                    "obs://YourObsBucketName/your/dependency2.jar")) // JAR file uploaded to OBS that includes the dependencies of the user-defined job. Delete this line if it is not required.
                .withDependencyJars(Arrays.asList("obs://YourObsBucketName/your/dependency1.csv",
                    "obs://YourObsBucketName/your/dependency2.json")) // File uploaded to OBS that stores the dependencies of the user-defined job. Delete this line if it is not required.
            ));
        return resp.getJob().getJobId();
    }
```

Running Flink Jobs in Batches

- **Function:**

  Run Flink SQL jobs in batches.

- **Reference link:**

  Key SDK API:
  com.huaweicloud.sdk.dli.v1.DliClient#batchRunFlinkJobs(com.huaweicloud.sdk.dli.v1.model.BatchRunFlinkJobsRequest)

  Run Flink jobs in batches. The job status changes from **Draft** to **Submitting**.

- **Sample code**:

```
private static List<FlinkSuccessResponse> batchRunFlinkJobs(DliClient client, List<Long> jobIds) {
        BatchRunFlinkJobsResponse batchRunFlinkJobsResponse = client.batchRunFlinkJobs(
            new BatchRunFlinkJobsRequest()
                .withBody(new BatchRunFlinkJobsRequestBody().withJobIds(jobIds)));
        return batchRunFlinkJobsResponse.getBody();
    }
```

Querying the Job Status

- **Function:**

  Query the status of a Flink SQL job.

- **Reference link:**

  Key SDK API:
  com.huaweicloud.sdk.dli.v1.DliClient#showFlinkJob(com.huaweicloud.sdk.dli.v1.model.ShowFlinkJobRequest)

  If you wish to wait for the job's transition into the **Running** state within the current thread, you can cyclically check the job status until it becomes **Running**.

- **Sample code**:

```
private static void checkRunning(DliClient client, Long jobId) throws DLIException {
        while (true) {
            ShowFlinkJobResponse resp;
            try {
                resp = client.showFlinkJob(new ShowFlinkJobRequest().withJobId(jobId));
            } catch (Exception e) {
                throw new DLIException("Failed to get Flink jar job status by id: " + jobId, e);
            }
            String status = resp.getJobDetail().getStatus();
            logger.info(String.format("FlinkJarJob id %s status: %s", jobId, status));
            if ("job_running".equals(status)) {
                return;
            }
            if ("job_submit_fail".equals(status) || "job_running_exception".equals(status)) {
                throw new DLIException("Run Flink jar job failed: " + resp);
            }
            try {
                Thread.sleep(1000L);
            } catch (InterruptedException e) {
                throw new DLIException("Check job running interrupted.");
            }
        }
    }
```

5.1.4 Submitting a Spark Job Using an SDK

This section describes how to submit a Spark job using DLI SDK V2.

NOTE

Starting May 2024, new users can directly use DLI's SDK V2 without needing to have their accounts whitelisted.

For users who started using DLI before May 2024, to use this function, they must submit a service ticket to have their accounts whitelisted.

Prerequisites

- You have configured the Java SDK environment by referring to **Overview**.
- You have initialized the DLI client by referring to **Initializing the DLI Client**.

Preparations

Obtain an AK/SK, project ID, and region information.

1. Log in to the management console.
2. In the upper right corner, hover over the username and choose **My Credentials** from the drop-down list.
3. In the navigation pane on the left, choose **Access Keys**. On the displayed page, click **Create Access Key**. Confirm that you want to proceed with the operation and click **OK**.

4. On the displayed page, click **Download**. Open the file to obtain the AK/SK information.
5. In the navigation pane on the left, choose **API Credentials**. In the **Projects** pane, locate **project_id** and obtain the region information.

Example Code

```
private static final Logger logger = LoggerFactory.getLogger(SparkJobExample.class);

public static void main(String[] args) {
    String yourAccessKey = System.getenv("HUAWEICLOUD_SDK_AK");
    String yourSecretKey = System.getenv("HUAWEICLOUD_SDK_SK");
    DliClient dliClient = DliClient.newBuilder()
        .withRegion(DliRegion.valueOf("RegionName"))
        .withCredential(new BasicCredentials()
            .withAk(yourAccessKey)
            .withSk(yourSecretKey)
            .withProjectId("YouProjectId"))
        .build();

    try {
        // Step 1: Submit a Spark job to DLI for execution.
        String jobId = runSparkJob(dliClient, "YourQueueName");
        // Step 2: If you wish to wait for the job execution to finish within the current thread, cycle through checking the status until the job completes.
        checkRunning(dliClient, jobId);
        // Step 3: To query one or more specific jobs based on conditions, use the following method:
        // This is just an example. In addition to jobId, you can also specify other filter criteria. For details, see Table 2 in https://console.huaweicloud.com/apiexplorer/#/openapi/DLI/doc?api=ListSparkJobs.
        listSparkJob(dliClient, jobId);

        /*
         * Other scenarios:
         * 1. During job execution, if you wish to cancel the job, you can call the API to cancel the batch processing job.
         * Key SDK API: com.huaweicloud.sdk.dli.v1.DliClient#cancelSparkJob(CancelSparkJobRequest),
         * Note: Batch processing jobs in the Successful or Failed state cannot be canceled.
         * 2. To query details about a specific job based on the job ID, perform the following operations:
         * Key SDK API: com.huaweicloud.sdk.dli.v1.DliClient#showSparkJob(ShowSparkJobRequest),
         */
    } catch (DLIException e) {
        // Handle the exception based on service requirements. The following is just an example.
    }
}
```

Creating a Spark Job

- **Function:**

  Execute Spark jobs.

- **Reference link:**

  Key SDK API: **com.huaweicloud.sdk.dli.v1.DliClient#createSparkJob(CreateSparkJobRequest)**

- **Sample code**:

```
private static String runSparkJob(DliClient client, String queueName) {
    // Set the parameters according to the actual situation. The following is just an example.
    Map<String, Object> confMap = new HashMap<>();
    confMap.put("SparkConfKey", "SparkConfValue");
    CreateSparkJobResponse resp = client.createSparkJob(new CreateSparkJobRequest()
        .withBody(new CreateSparkJobRequestBody()
            .withQueue(queueName)
            .withSparkVersion("2.4.5")
            .withName("demo_spark_app")
```



```
            .withFile("obs://your_bucket/your_spark_app.jar") // Mandatory
            .withClassName("YourClassFullName") // Mandatory
            .withArgs(Arrays.asList("YourAppArg1", "YourAppAgr2", "..."))
            .withConf(confMap)
            .withJars(Arrays.asList("YourDepJar1", "YourDepJar2", "..."))
            .withDriverCores(2)
            .withDriverMemory("8GB")
            .withNumExecutors(3)
            .withExecutorCores(4)
            .withExecutorMemory("16GB")));
    return resp.getId();
}
```

Querying the Status of a Batch Processing Job

- **Function:**

  Execute Spark jobs.

- **Reference link:**

  Key SDK API:
  com.huaweicloud.sdk.dli.v1.DliClient#showSparkJobStatus(ShowSparkJobStatusRequest)

- **Sample code**:

```
private static void checkRunning(DliClient client, String jobId) throws DLIException {
    while (true) {
        ShowSparkJobStatusResponse resp;
        try {
            resp = client.showSparkJobStatus(new ShowSparkJobStatusRequest().withBatchId(jobId));
        } catch (Exception e) {
            throw new DLIException("Failed to get job status by id: " + jobId, e);
        }
        String status = resp.getState();
        logger.info(String.format("SparkJob id %s status: %s", jobId, status));

        if ("success".equals(status)) {
            return;
        }
        if ("dead".equals(status)) {
            throw new DLIException("Run job failed");
        }

        try {
            Thread.sleep(1000L);
        } catch (InterruptedException e) {
            throw new DLIException("Check job running interrupted.");
        }
    }
}
```

Querying the Batch Processing Job List

- **Function:**

  Query the batch processing job list.

- **Reference link:**

  Key SDK API:
  com.huaweicloud.sdk.dli.v1.DliClient#listSparkJobs(ListSparkJobsRequest)

- **Sample code**:

```
private static void listSparkJob(DliClient client, String jobId) throws DLIException {
    ListSparkJobsResponse resp;
    try {
        resp = client.listSparkJobs(new ListSparkJobsRequest()
            // You can also use the .withXxx() method to specify other criteria to return Spark jobs
```



```
that meet the criteria. This is just an example.
            // See Table 2 in https://console.huaweicloud.com/apiexplorer/#/openapi/DLI/doc?api=ListSparkJobs.
            .withJobId(jobId)
            .withQueueName("YourQueueName")
            .withStart(1234567L) // You can specify the job start time.
            .withEnd(2345678L)); // You can specify the job end time.
        } catch (Exception e) {
            throw new DLIException("Failed to list Spark jobs: ", e);
        }
        logger.info(String.format("List SparkJobs : %s", resp.toString()));
        // For details about the response parameters in resp, see Table 3 and Table 4 in https://console.huaweicloud.com/apiexplorer/#/openapi/DLI/doc?api=ListSparkJobs.
    }
```

5.2 Python SDK (DLI SDK V2)

5.2.1 Submitting a SQL Job Using an SDK

This section describes how to submit a SQL job using DLI SDK V2.

NOTE

Starting May 2024, new users can directly use DLI's SDK V2 without needing to have their accounts whitelisted.

For users who started using DLI before May 2024, to use this function, they must submit a service ticket to have their accounts whitelisted.

Prerequisites

- You have configured the Python SDK environment by referring to **Preparing a Python Development Environment**.
- You have initialized the DLI client by referring to **Initializing the DLI Client**.

Preparations

Obtain an AK/SK, project ID, and region information.

1. Log in to the management console.
2. In the upper right corner, hover over the username and choose **My Credentials** from the drop-down list.
3. In the navigation pane on the left, choose **Access Keys**. On the displayed page, click **Create Access Key**. Confirm that you want to proceed with the operation and click **OK**.
4. On the displayed page, click **Download**. Open the file to obtain the AK/SK information.
5. In the navigation pane on the left, choose **API Credentials**. In the **Projects** pane, locate **project_id** and obtain the region information.

Example Code

```
def main():
    your_access_key = os.getenv("HUAWEICLOUD_SDK_AK")
   your_secret_key = os.getenv("HUAWEICLOUD_SDK_SK")
   kwargs = {
       'region': 'region_name',
```



```
        'project_id': 'your_project_id',
        'ak': your_access_key,
        'sk': your_secret_key
    }
    from dli.dli_client import DliClient
    dli_client = DliClient(**kwargs)

    try:
        # Step 1: Create a database and a table.
        queue_name = 'your_sql_queue_name'
        prepare(dli_client, queue_name)

        # Step 2: Import data to the table.
        # The overall implementation process/ principle can be divided into the following three steps:
        # 1. Use the OBS API to upload data to obs_path_to_write_tmp_data. You can configure a lifecycle policy in OBS to periodically delete these temporary data.
        # 2. Submit the Load Data statement to DLI to import OBS data to DLI.
        #    For details about the Load Data syntax, see Importing Data.
        # 3. Cyclically check the job status until the job is complete.
        obs_path_to_write_tmp_data = f"obs://your_obs_bucket_name/your/path/{uuid.uuid4()}"
        load_data(dli_client, obs_path_to_write_tmp_data, queue_name)

        # Step 3: Submit the SQL statement, execute the query, and read the result.
        select_sql = "SELECT * FROM demo_db.demo_tbl"
        job_id = query_data(dli_client, select_sql, queue_name)

        # Step 3: If needed, you can also obtain the results by job ID.
        query_data_by_jobid(dli_client, job_id)

        # Query all jobs by page. You can use this API to query information of all SQL jobs within the current project.
        # Key SDK API: huaweicloudsdkdli.v1.dli_client.DliClient.list_sql_jobs,
        list_sql_jobs(dli_client)

        # Other scenarios:
        # 1. To cancel a submitted SQL job, use the following API.
        # Key SDK API: huaweicloudsdkdli.v1.dli_client.DliClient.cancel_sql_job
        # Note: If a job has been completed or failed, it cannot be canceled.

        # 2. To verify the syntax of an SQL statement, use the following API.
        # Key SDK API: huaweicloudsdkdli.v1.dli_client.DliClient.check_sql
        # Note: This API can only be used to verify the syntax, not the semantics. Use the Explain statement and submit it to DLI for execution to perform semantic verification.

        # 3. To obtain a submitted SQL job based on the job ID and view job details, use the following API.
        # Key SDK API: huaweicloudsdkdli.v1.dli_client.DliClient.show_sql_job_detail
        # 4. Obtain the job execution progress. If the job is being executed, you can obtain the sub-job information. If the job has just started or has been completed, you cannot obtain the sub-job information.
        # Key SDK API: huaweicloudsdkdli.v1.dli_client.DliClient.show_sql_job_progress
    except DliException as e:
        # Handle the exception based on service requirements. The following is just an example.
        logger.error(e)
```

Creating a Database and Table

Reference links:

- Creating a Database
- Creating a Table
- Key SDK: **dli.dli_client.DliClient.execute_sql**
- Key API: **huaweicloudsdkdli.v1.dli_client.DliClient.create_sql_job**.
- Key API: **huaweicloudsdkdli.v1.dli_client.DliClient.show_sql_job_status**.

Sample code:

```
def prepare(dli_client, queue_name):
    """
```



```
    Create a database and a table.
    :param dli.dli_client.DliClient dli_client: DLI Client.
    :param str queue_name: Queue name for the SQL execute.
    :return: dli.job.SqlJob
    """
    # 1. Create a database.
    # default is the database built in DLI. You cannot create a database named default.
    createDbSql = "CREATE DATABASE IF NOT EXISTS demo_db"
    dli_client.execute_sql(sql=createDbSql, queue_name=queue_name) # Submit a SQL job and cyclically check the job status until the job is complete.

    # 2. Create a table. Note: Adjust the table structure, table data directory, and OBS storage path based on site requirements.
    createTblSql = "CREATE TABLE IF NOT EXISTS `demo_tbl` (" \
                   "  `bool_c` BOOLEAN," \
                   "  `byte_c` TINYINT," \
                   "  `short_c` SMALLINT," \
                   "  `int_c` INT," \
                   "  `long_c` BIGINT," \
                   "  `float_c` FLOAT," \
                   "  `double_c` DOUBLE," \
                   "  `decimal_c` DECIMAL(10,2)," \
                   "  `str_c` STRING," \
                   "  `date_c` DATE," \
                   "  `timestamp_c` TIMESTAMP," \
                   "  `binary_c` BINARY," \
                   "  `array_c` ARRAY<INT>," \
                   "  `map_c` MAP<STRING, INT>," \
                   "  `struct_c` STRUCT<`s_str_c`: STRING, `s_bool_c`: BOOLEAN>)" \
                   " LOCATION 'obs://demo_bucket/demo_db/demo_tbl'" \
                   " STORED as TEXTFILE"
    dli_client.execute_sql(sql=createTblSql, db_name='demo_db', queue_name=queue_name)
```

Importing Data

- **Reference links**:
- **Importing Data**
- **Native Data Types**
- **Complex Data Types**
- Key SDK: **dli.dli_client.DliClient.execute_sql**
- Key API: **huaweicloudsdkdli.v1.dli_client.DliClient.create_sql_job**.
- Key API: **huaweicloudsdkdli.v1.dli_client.DliClient.show_sql_job_status**.
- **Sample code**:

```
def load_data(dli_client, upload_data_path, queue_name):
    # 1. Write data to the OBS temporary directory. Modify the following information based on site requirements. The following is just an example.
    # Note: This step involves directly calling the OBS data writing API, entirely decoupled from DLI. This example only provides an implementation for writing data in JSON format, meaning that files are stored in JSON format on OBS.
    # You can customize the implementation based on service requirements. For example, you can save files as CSV files.
    write_tmp_data(get_schema(), upload_data_path, dli_client.dli_info, 1)

    # 2. Import the data written to OBS in step 1 to DLI.
    # Note: The data_type here needs to be determined based on the file type in step 1; in this example, it is JSON. If you use other formats, you should modify it to the corresponding data_type.
    loadSql = f"LOAD DATA INPATH '{upload_data_path}' INTO TABLE demo_db.demo_tbl OPTIONS(data_type 'json')"
    dli_client.execute_sql(sql=loadSql, queue_name=queue_name) # Submit a SQL job and cyclically check the job status until the job is complete.
```


Querying Job Results

- **Reference link**:

  SELECT Query Statement

- Key SDK: **dli.dli_client.DliClient.execute_sql**
- Key SDK: **dli.job.SqlJob.get_result**. The feature of writing results to the job bucket must be enabled; otherwise, an exception is thrown.

  You can determine if the feature is enabled by checking the **result_path** parameter in the response body of the API for querying job status.

  After the job is completed, if **result_path** starts with **obs://**, the feature of writing job results to the job bucket is enabled; otherwise, it is not enabled.
- Key API: **huaweicloudsdkdli.v1.dli_client.DliClient.create_sql_job**.
- Key API: **huaweicloudsdkdli.v1.dli_client.DliClient.show_sql_job_status**.

```
def query_data(dli_client, select_sql, queue_name):
    """
    :param dli.dli_client.DliClient dli_client: DLI Client.
    :param str select_sql: SQL statement
    :param str queue_name: Queue name for the SQL execute
    :return: str
    """
    sql_job = dli_client.execute_sql(sql=select_sql, queue_name=queue_name)
    print_result(sql_job.get_result())
    return sql_job.job_id
```

Querying the Result of a Specified Job

- **Instructions**
  - Key SDK: **dli.job.SqlJob.get_result**.

    This method can be used only when the function of writing job results to the job bucket is enabled. Otherwise, an exception is thrown during job running.

    You can determine if the feature is enabled by checking the **result_path** parameter in the response body of the API for querying job status. After the job is completed, if **result_path** starts with **obs://**, the feature of writing job results to the job bucket is enabled; otherwise, it is not enabled.
- **Sample code**

```
def query_data_by_jobid(dli_client, job_id):
    """
    :param dli.dli_client.DliClient dli_client: DLI Client.
    :param str job_id: job ID of a query job
    :return:
    """
    from dli.job import SqlJob
    sql_job = SqlJob(job_id=job_id, job_type="QUERY", client=dli_client)
    dli_client._cycle_check_sql_job(sql_job=sql_job)
    print_result(sql_job.get_result())
```

Querying the Job List

- **Instructions**

  Query information about SQL jobs in the current project.

  If there are a large number of jobs, you must use the following pagination query method in this example to query jobs in batches. Otherwise, only the jobs on the first page are returned.

Key SDK: **huaweicloudsdkdli.v1.dli_client.DliClient.list_sql_jobs**

- **Sample code**

```
def list_sql_jobs(client):
    """
Query information about SQL jobs in the current project. If there are a large number of jobs, you must use the pagination query method in this example to query jobs in batches. Otherwise, only the jobs on the first page are returned.
    Key SDK: huaweicloudsdkdli.v1.dli_client.DliClient.list_sql_jobs

    :param dli.dli_client.DliClient client: DLI Client.
    """
    req = ListSqlJobsRequest()
    req.current_page = 1 # The default value is 1.
    req.page_size = 100 # The default value is 10.

    # Obtain the total number of jobs.
    job_count = client.inner_client.list_sql_jobs(req).job_count
    cur = 0

    # Query jobs by page.
    while cur < job_count:
        list_sql_jobs_response = client.inner_client.list_sql_jobs(req)
        jobs = list_sql_jobs_response.jobs
        for job in jobs:

            # Add the service logic here to process each job.
            print(job)

            cur += 1
            if cur >= job_count:
                break
        req.current_page += 1
```

Printing Job Results

- **Instructions**

  Process each row of data based on service requirements. The following is just an example.

- **Sample code**

```
def print_result(obs_reader):
    """
    Process each row of data based on service requirements. The following is just an example.
    """
    count = 0
    for record in obs_reader:
        count += 1
        print(record)
    logger.info("total records: %d", count)
```

Writing Data to OBS by Running writeTmpData

- **Instructions**

  This method involves directly calling the OBS data writing API, entirely decoupled from DLI APIs. This example provides an implementation for writing data in JSON format, meaning that files are stored in JSON format on OBS.

  You can customize the implementation based on service requirements. For example, you can save files as CSV files.

- **Sample code**

```
def write_tmp_data(schema, upload_data_path, dli_info, total_records):
    """
    This method involves directly calling the OBS data writing API, entirely decoupled from DLI APIs.
```



```
This example provides an implementation for writing data in JSON format, meaning that files are stored in JSON format on OBS.
    You can customize the implementation based on service requirements. For example, you can save files as CSV files.

    :param list schema: Data structure schema
    :param str upload_data_path: This OBS temporary directory is used to store user service data.
    :param dli.dli_client.DliClient.dli_info dli_info: Authentication information passed during the initialization of the DLI client.
    :param int total_records: Number of simulated data rows. You can configure the number of data rows to be inserted using this parameter. This is for display purposes only and you can modify it based on service requirements.
    :return:
    """
    obs_client = ObsClient(access_key_id=dli_info.ak, secret_access_key=dli_info.sk, is_secure=True,
                           server=dli_info.obs)
    bucket_name = get_bucket_name(upload_data_path)
    object_prefix = get_object_prefix(upload_data_path)

    datas = ""
    try:
        for i in range(total_records):
            row = Row(schema=schema, columns=get_record())
            datas += to_json_string(row, schema)
        object_key = object_prefix + "/tmp_data.json"
        obs_client.putObject(bucket_name, object_key, datas)
        logger.info("Uploaded data to OBS bucket '%s' with object key '%s'", bucket_name, object_key)
    finally:
        obs_client.close()
```

Creating a Schema for a Table

- **Instructions**

  Construct the schema based on the actual service. The following is just an example.

- **Sample code**

```
def get_schema():
    """
    Construct the schema based on the actual service. The following is just an example.
    """
    from dli.column import Column
    return [Column(name="bool_c", data_type="boolean", desc="boolean col"),
            Column(name="byte_c", data_type="tinyint", desc="tinyint col"),
            Column(name="short_c", data_type="smallint", desc="smallint col"),
            Column(name="int_c", data_type="int", desc="int col"),
            Column(name="long_c", data_type="bigint", desc="bigint col"),
            Column(name="float_c", data_type="float", desc="float col"),
            Column(name="double_c", data_type="double", desc="double col"),
            Column(name="decimal_c", data_type="decimal(10,2)", desc="decimal col"),
            Column(name="str_c", data_type="string", desc="string col"),
            Column(name="date_c", data_type="date", desc="date col"),
            Column(name="timestamp_c", data_type="timestamp", desc="timestamp col"),
            Column(name="binary_c", data_type="binary", desc="binary col"),
            Column(name="array_c", data_type="array<int>", desc="array col"),
            Column(name="map_c", data_type="map<string, int>", desc="map col"),
            Column(name="struct_c", data_type="struct<s_str_c:string,s_bool_c:boolean>", desc="struct col")]
```

Generating Test Data List<Object> genRecord on Demand

- **Instructions**

  Construct each row of data based on service requirements. The following is just an example.

- **Sample code**


```
def get_record():
    record = [
        True,  # boolean
        1,  # byte
        123,  # short
        65535,  # int
        123456789012,  # long
        101.235,  # float
        256.012358,  # double
        33.05,  # decimal
        "abc_123&",  # string
        "2023-05-08",  # date
        "1716345295000",  # Timestamp, in milliseconds
        base64.b64encode("hello".encode('utf-8')),  # binary
        [1, 2, 3],  # array
        {"k": 123},  # map
        {"s_str_c": "Abc", "s_bool_c": True}  # struct
    ]
    return record
```

to_json_string

- **Instructions**

  Construct each row of data based on service requirements. The following is just an example.

- **Sample code**

```
def to_json_string(row, schema):
    json_obj = {}
    for i, column in enumerate(schema):
        if column.is_partition_column:
            continue
        if column.type == 'binary':
            json_obj[column.name] = base64.b64encode(row.columns[i]).decode('utf-8')
        elif column.type.startswith('decimal'):
            json_obj[column.name] = float(row.columns[i])
        else:
            json_obj[column.name] = row.columns[i]
    return json.dumps(json_obj) + "\n"
```

get_bucket_name

- **Instructions**

  Construct each row of data based on service requirements. The following is just an example.

- **Sample code**

```
def get_bucket_name(full_path):
    try:
        url = urlparse(full_path)
        return url.hostname
    except Exception as e:
        logger.error("Failed to get bucket name from full path", e)
        return None
```

get_object_prefix

- **Instructions**

  Construct each row of data based on service requirements. The following is just an example.

- **Sample code**

```
def get_object_prefix(full_path):
    try:
        url = urlparse(full_path)
```



```
        return url.path.lstrip('/')
    except Exception as e:
        logger.error("Failed to get object key from full path", e)
        return None
```

5.2.2 Submitting a Flink SQL Job Using an SDK

This section describes how to submit a Flink SQL job using DLI SDK V2.

NOTE

Starting May 2024, new users can directly use DLI's SDK V2 without needing to have their accounts whitelisted.

For users who started using DLI before May 2024, to use this function, they must submit a service ticket to have their accounts whitelisted.

Prerequisites

- You have configured the Python SDK environment by referring to **Preparing a Python Development Environment**.
- You have initialized the DLI client by referring to **Initializing the DLI Client**.

Preparations

Obtain an AK/SK, project ID, and region information.

1. Log in to the management console.
2. In the upper right corner, hover over the username and choose **My Credentials** from the drop-down list.
3. In the navigation pane on the left, choose **Access Keys**. On the displayed page, click **Create Access Key**. Confirm that you want to proceed with the operation and click **OK**.
4. On the displayed page, click **Download**. Open the file to obtain the AK/SK information.
5. In the navigation pane on the left, choose **API Credentials**. In the **Projects** pane, locate **project_id** and obtain the region information.

Example Code

```
# Configure logs.
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def main():
    your_access_key = os.getenv("HUAWEICLOUD_SDK_AK")
    your_secret_key = os.getenv("HUAWEICLOUD_SDK_SK")
    project_id = "your_project_id"
    region_name = "region_name"
    credentials = BasicCredentials(your_access_key, your_secret_key, project_id)
    dli_client = DliClient.new_builder() \
        .with_credentials(credentials) \
        .with_region(DliRegion.value_of(region_name)) \
        .build()

    try:
        # Step 1: Create a Flink job. The job status changes to Draft.
        job_id = create_flink_sql_job(dli_client, "your_queue_name")
        logger.info("job_id: %d", job_id)
```



```
    # Step 2: Run the job. The job status changes from Draft to Submitting.
    resps = batch_run_flink_sql_jobs(dli_client, [job_id])
    logger.info("Response: %s", resps)

    # Step 3: Query the job status. If you wish to wait for the job to reach the Running state within the current thread, you can cyclically check the job status until it becomes Running.
    check_running(dli_client, job_id)

  except exceptions.ClientRequestException as e:
    # Handle the exception based on service requirements. The following is just an example.
    logger.error("Failed to execute job:", e)
```

Creating a Flink SQL Job

- **Function:**

  Create a Flink SQL job.

- **Reference link:**

  Key SDK API: **huaweicloudsdkdli.v1.dli_client.DliClient.create_flink_sql_job**.

  Create a Flink SQL job. The job status changes to **Draft**.

- **Sample code**:

```
def create_flink_sql_job(client, queue_name):
    """
    Create a Flink job. The job status changes to Draft.
    Key SDK API: huaweicloudsdkdli.v1.dli_client.DliClient.create_flink_sql_job.

    :param huaweicloudsdkdli.v1.dli_client.DliClient client: DLI Client.
    :param str queue_name: Queue name for the job to execute
    :return: int
    """
    request_body = CreateFlinkSqlJobRequestBody(
        name="your_job_name",  # Job name, which must be unique, for example, flink_jar_job_demo. The name can contain up to 57 characters.
        desc="your flink job's description",  # User-defined description. The description can contain up to 512 characters.
        sql_body="""create table orders(
              name string,
              num INT
            ) with (
              'connector' = 'datagen',
              'rows-per-second' = '1',
              'fields.name.kind' = 'random',
              'fields.name.length' = '5'
            );
            CREATE TABLE sink_table (
               name string,
               num INT
            ) WITH (
               'connector' = 'print'
            );
            INSERT into sink_table SELECT * from orders;""",
        # Customize a stream SQL statement, which contains at least the following three parts: source, query, and sink. Length limit: 1024 x 1024 characters.
        # In this SQL statement, random source data is automatically generated and printed to the console.
        queue_name=***queue_name***,   # General queue name. The name can contain up to 128 characters.
        run_mode="exclusive_cluster",  # Job running mode. Only the exclusive_cluster mode is supported.
        log_enabled=True,  # Enable the function of uploading job logs to OBS buckets.
        obs_bucket="your_obs_bucket_name",  # OBS bucket name, which is used to store logs and checkpoints.
        job_type="flink_opensource_sql_job",  # Job type. You are advised to select flink_opensource_sql_job.
        flink_version="1.12" # Specify the Flink version.
    )
```



```
    request = CreateFlinkSqlJobRequest(body=request_body)
    response = client.create_flink_sql_job(request)
    return response.job.job_id
```

Running Flink Jobs in Batches

- **Function:**

  Run Flink SQL jobs in batches.

- **Reference link:**

  Key SDK API:

  huaweicloudsdkdli.v1.dli_client.DliClient.batch_run_flink_jobs.

  Run Flink jobs in batches. The job status changes from **Draft** to **Submitting**.

- **Sample code**:

```
def batch_run_flink_sql_jobs(client, job_ids):
    """
    Run jobs. The job status changes from **Draft** to **Submitting**.
    :param huaweicloudsdkdli.v1.dli_client.DliClient client: DLI Client.
    :param list[int] job_ids: The job ids for running.
    :return: The body of this BatchRunFlinkJobsResponse.
    :rtype: list[:class:`huaweicloudsdkdli.v1.FlinkSuccessResponse`]
    """
    request_body = BatchRunFlinkJobsRequestBody(job_ids=job_ids)
    request = BatchRunFlinkJobsRequest(body=request_body)
    response = client.batch_run_flink_jobs(request)
    return response.body
```

Querying the Job Status

- **Function:**

  Query the status of a Flink SQL job.

- **Reference link:**

  Key SDK API:

  huaweicloudsdkdli.v1.dli_client.DliClient.show_flink_job.

  If you wish to wait for the job's transition into the **Running** state within the current thread, you can cyclically check the job status until it becomes **Running**.

- **Sample code**:

```
def check_running(client, job_id):
    """
    If you wish to wait for the job's transition into the **Running** state within the current thread, you can execute this method to cyclically check the job status until it becomes **Running**.

    :param huaweicloudsdkdli.v1.dli_client.DliClient client: DLI Client.
    :param int job_id: The job id for getting status.
    :return:
    """
    while True:
        try:
            request = ShowFlinkJobRequest(job_id=job_id)
            response = client.show_flink_job(request)
        except exceptions.ClientRequestException as e:
            raise Exception(f"Failed to get Flink sql job status by id: {job_id}") from e

        status = response.job_detail.status
        logger.info("FlinkSqlJob id %d status: %s", job_id, status)

        if status == "job_running":
            return
        if status in ["job_submit_fail", "job_running_exception"]:
```



```
            raise Exception(f"Run Flink sql job failed: {response}")

        time.sleep(1)
```

5.2.3 Submitting a Flink Jar Job Using an SDK

This section describes how to submit a Flink Jar job using DLI SDK V2.

NOTE

Starting May 2024, new users can directly use DLI's SDK V2 without needing to have their accounts whitelisted.

For users who started using DLI before May 2024, to use this function, they must submit a service ticket to have their accounts whitelisted.

Prerequisites

- You have configured the Python SDK environment by referring to **Preparing a Python Development Environment**.
- You have initialized the DLI client by referring to **Initializing the DLI Client**.

Preparations

Obtain an AK/SK, project ID, and region information.

1. Log in to the management console.
2. In the upper right corner, hover over the username and choose **My Credentials** from the drop-down list.
3. In the navigation pane on the left, choose **Access Keys**. On the displayed page, click **Create Access Key**. Confirm that you want to proceed with the operation and click **OK**.
4. On the displayed page, click **Download**. Open the file to obtain the AK/SK information.
5. In the navigation pane on the left, choose **API Credentials**. In the **Projects** pane, locate **project_id** and obtain the region information.

Example Code

```
def main():
    your_access_key = os.getenv("HUAWEICLOUD_SDK_AK")
    your_secret_key = os.getenv("HUAWEICLOUD_SDK_SK")
    project_id = "your_project_id"
    region_name = "region_name"
    credentials = BasicCredentials(your_access_key, your_secret_key, project_id)
    dli_client = DliClient.new_builder() \
        .with_credentials(credentials) \
        .with_region(DliRegion.value_of(region_name)) \
        .build()

    try:
        # Step 1: Create a Flink job. The job status changes to Draft.
        job_id = create_flink_jar_job(dli_client, "your_queue_name")
        logger.info("job_id: %d", job_id)

        # Step 2: Run the job. The job status changes from Draft to Submitting.
        resps = batch_run_flink_jar_jobs(dli_client, [job_id])
        logger.info("Response: %s", resps)

        # Step 3: Query the job status. If you wish to wait for the job to reach the Running state within the
```



```
current thread, you can cyclically check the job status until it becomes Running.
        check_running(dli_client, job_id)

    except exceptions.ClientRequestException as e:
        # Handle the exception based on service requirements. The following is just an example.
        logger.error("Failed to execute job:", e)
```

Creating a Flink Jar Job

- **Function:**

  Create a Flink Jar job.

- **Reference link:**

  Key SDK API:

  huaweicloudsdkdli.v1.dli_client.DliClient.create_flink_jar_job

  Create a Flink Jar job. The job status changes to **Draft**.

- **Sample code**:

```
def create_flink_jar_job(client, queue_name):
    """
    Create a Flink Jar job. The job status changes to Draft.

    :param huaweicloudsdkdli.v1.dli_client.DliClient client: DLI Client
    :param str queue_name: Queue name for the job to execute
    :return: int
    """
    request_body = CreateFlinkJarJobRequestBody(
        name="your_job_name",  # Job name, which must be unique, for example, flink_jar_job_demo. The name can contain up to 57 characters.
        desc="your flink job's description",  # User-defined description. The description can contain up to 512 characters.
        queue_name=queue_name,  # General queue name. The name can contain up to 128 characters.
        feature="basic",  # Job feature. Type of the Flink image used by a job. basic: the base Flink image provided by DLI is used.
        flink_version="1.12",  # Flink version. This parameter takes effect when feature is set to basic. You can use this parameter together with feature to specify the version of the base Flink image used by the job.
        log_enabled=True,  # Whether to enable job logs.
        obs_bucket="your_obs_bucket_name",  # Name of the OBS bucket authorized by the user to store job logs when log_enabled is set to true.
        entrypoint="obs://your_obs_bucket_name/your/flink/job.jar",  # Program package uploaded to OBS that stores the user-defined job main class.
        main_class="your_class_fullname"  # Job entry class, for example, org.apache.flink.examples.WordCount.
    )
    request = CreateFlinkJarJobRequest(body=request_body)
    response = client.create_flink_jar_job(request)
    return response.job.job_id
```

Running Flink Jobs in Batches

- **Function:**

  Run Flink SQL jobs in batches.

- **Reference link:**

  Key SDK API:

  huaweicloudsdkdli.v1.dli_client.DliClient.batch_run_flink_jobs

  Run Flink jobs in batches. The job status changes from **Draft** to **Submitting**.

- **Sample code**:

```
def batch_run_flink_jar_jobs(client, job_ids):
    """
```



```
    Run jobs. The job status changes from Draft to Submitting.
    Key SDK API: huaweicloudsdkdli.v1.dli_client.DliClient.batch_run_flink_jobs.

    :param huaweicloudsdkdli.v1.dli_client.DliClient client: DLI Client.
    :param list[int] job_ids: The job ids for running.
    :return: The body of this BatchRunFlinkJobsResponse.
    :rtype: list[:class:`huaweicloudsdkdli.v1.FlinkSuccessResponse`]
    """
    request_body = BatchRunFlinkJobsRequestBody(job_ids=job_ids)
    request = BatchRunFlinkJobsRequest(body=request_body)
    response = client.batch_run_flink_jobs(request)
    return response.body
```

Querying the Job Status

- **Function:**

  Query the status of a Flink SQL job.

- **Reference link:**

  Key SDK API:

  huaweicloudsdkdli.v1.dli_client.DliClient.show_flink_job

  If you wish to wait for the job's transition into the **Running** state within the current thread, you can cyclically check the job status until it becomes **Running**.

- **Sample code**:

```
def check_running(client, job_id):
    """
    If you wish to wait for the job's transition into the Running state within the current thread, you can execute this method to cyclically check the job status until it becomes Running.
    Key SDK API: huaweicloudsdkdli.v1.dli_client.DliClient.show_flink_job.

    :param huaweicloudsdkdli.v1.dli_client.DliClient client: DLI Client.
    :param int job_id: The job id for getting status.
    :return:
    """
    while True:
        try:
            request = ShowFlinkJobRequest(job_id=job_id)
            response = client.show_flink_job(request)
        except exceptions.ClientRequestException as e:
            raise Exception(f"Failed to get Flink jar job status by id: {job_id}") from e

        status = response.job_detail.status
        logger.info("FlinkJarJob id %d status: %s", job_id, status)

        if status == "job_running":
            return
        if status in ["job_submit_fail", "job_running_exception"]:
            raise Exception(f"Run Flink jar job failed: {response}")

        time.sleep(1)
```

5.2.4 Submitting a Spark Job Using an SDK

This section describes how to submit a Spark job using DLI SDK V2.

NOTE

Starting May 2024, new users can directly use DLI's SDK V2 without needing to have their accounts whitelisted.

For users who started using DLI before May 2024, to use this function, they must submit a service ticket to have their accounts whitelisted.

Prerequisites

- You have configured the Python SDK environment by referring to **Preparing a Python Development Environment**.
- You have initialized the DLI client by referring to **Initializing the DLI Client**.

Preparations

Obtain an AK/SK, project ID, and region information.

1. Log in to the management console.
2. In the upper right corner, hover over the username and choose **My Credentials** from the drop-down list.
3. In the navigation pane on the left, choose **Access Keys**. On the displayed page, click **Create Access Key**. Confirm that you want to proceed with the operation and click **OK**.
4. On the displayed page, click **Download**. Open the file to obtain the AK/SK information.
5. In the navigation pane on the left, choose **API Credentials**. In the **Projects** pane, locate **project_id** and obtain the region information.

Example Code

```
# Configure logs.
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
def main():
    your_access_key = os.getenv("HUAWEICLOUD_SDK_AK")
    your_secret_key = os.getenv("HUAWEICLOUD_SDK_SK")
    project_id = "your_project_id"
    region_name = "region_name"
    credentials = BasicCredentials(your_access_key, your_secret_key, project_id)
    dli_client = DliClient.new_builder() \
        .with_credentials(credentials) \
        .with_region(DliRegion.value_of(region_name)) \
        .build()
    try:
        # Step 1: Submit a Spark job to DLI for execution.
        job_id = run_spark_job(dli_client, "your_queue_name")
        # Step 2: If you wish to wait for the job execution to finish within the current thread, cycle through
checking the status until the job completes.
        check_running(dli_client, job_id)
        # Step 3: To query one or more specific jobs based on conditions, use the following method:
        # This is just an example. In addition to jobId, you can also specify other filter criteria.
        list_spark_job(dli_client, job_id)
        # Other scenarios:
        # 1. If you want to cancel a running job, call the API below.
        # Key SDK API: huaweicloudsdkdli.v1.dli_client.DliClient.cancel_spark_job
        # Note: Batch processing jobs in the Successful or Failed state cannot be canceled.
        # 2. To query details about a specific job based on the job ID, perform the following operations:
        # Key SDK API: huaweicloudsdkdli.v1.dli_client.DliClient.show_spark_job
    except exceptions.ClientRequestException as e:
        # Handle the exception based on service requirements. The following is just an example.
        logger.error("Failed to execute job: ", e)
```

Creating a Spark Job

- **Function:**

  Execute Spark jobs.

- **Reference link:**

  Key SDK API: **huaweicloudsdkdli.v1.dli_client.DliClient.create_spark_job**

- **Sample code**:

```
def run_spark_job(client, queue_name):
    """
    Submit a Spark job to DLI for execution.
    Key SDK API: huaweicloudsdkdli.v1.dli_client.DliClient.create_spark_job.
    :param huaweicloudsdkdli.v1.dli_client.DliClient client: DLI Client.
    :param str queue_name: Queue name for the job to execute
    :return: str
    """
    conf_map = {
        "spark.dli.metaAccess.enable": "true"  # For example, "spark.dli.metaAccess.enable": "true"
    }
    request_body = CreateSparkJobRequestBody(
        # Set the parameters according to the actual situation. The following is just an example.
        queue=queue_name, # Name of a general queue
        spark_version="2.4.5",  # Version of the Spark component used by the job.
        name="demo_spark_app",  # User-defined job name, which is the batch processing job name specified by the user during job creation. The name can contain up to 128 characters.
        file="obs://your_bucket/your_spark_app.jar",  # (Mandatory) Location of the JAR file on OBS.
        class_name="your_class_fullname",  # (Mandatory) Class path of the main class ( --class), for example, org.example.DliCatalogTest.
        args=["YourAppArg1", "YourAppAgr2", "..."],  # Input parameters of the application. Delete this line if it is not required.
        conf=conf_map,  # Spark parameter (--conf)
        catalog_name="dli",  # When accessing metadata, set this parameter to dli and ensure conf_map["spark.dli.metaAccess.enable"] = "true".
        jars=["YourDepJar1", "YourDepJar2", "..."],  # Dependency JAR file (--jars). Delete this line if it is not required.
        feature="basic",  # Job feature, which indicates the Spark image type used by the user job. It is typically set to basic.
        driver_cores=1,  # Number of CPU cores for the driver in a Spark application
        driver_memory="1GB",  # Memory for the driver in a Spark application. It can be set to values like 2G or 2048M. The value must include a unit; otherwise, the startup will fail.
        num_executors=1,  # Number of executors in a Spark application
        executor_cores=1,  # Number of CPU cores for each executor in a Spark application
        executor_memory="1GB"  # Memory for executors in a Spark application. It can be set to values like 2G or 2048M. The value must include a unit; otherwise, the startup will fail.
    )
    request = CreateSparkJobRequest(body=request_body)
    response = client.create_spark_job(request)
    return response.id
```

Querying the Status of a Batch Processing Job

- **Function:**

  Execute Spark jobs.

- **Reference link:**

  Key SDK API:

  huaweicloudsdkdli.v1.dli_client.DliClient.show_spark_job_status

- **Sample code**:

```
def check_running(client, job_id):
    """
    If you wish to wait for the job execution to finish within the current thread, cycle through checking the status until the job completes.

    :param huaweicloudsdkdli.v1.dli_client.DliClient client: DLI Client.
    :param str job_id: The job id for getting status.
    :return:
    """
    while True:
        try:
            request = ShowSparkJobStatusRequest(batch_id=job_id)
```



```
        response = client.show_spark_job_status(request)
    except exceptions.ClientRequestException as e:
        raise Exception(f"Failed to get job status by id: {job_id}") from e
    status = response.state
    logger.info("SparkJob id %s status: %s", job_id, status)
    if status == "success":
        return
    if status == "dead":
        raise Exception("Run job failed")
    time.sleep(1)
```

Querying the Batch Processing Job List

- **Function:**

  Query the batch processing job list.

- **Reference link:**

  Key SDK API:

  huaweicloudsdkdli.v1.dli_client.DliClient.list_spark_jobs

- **Sample code**:

```
def list_spark_job(client, job_id):
    """
    :param huaweicloudsdkdli.v1.dli_client.DliClient client: DLI Client.
    :param str job_id: The job id for getting details.
    :return:
    """
    try:
        request = ListSparkJobsRequest(
            # This is just an example. In addition to jobId, you can also specify other filter criteria.
            job_id=job_id,
            queue_name="your_queue_name",  # Change the value to a DLI queue name. You can then query batch processing jobs running on the queue (recommended).
            start=1716195600000,  # Change the value to the start time of a user-defined job. Jobs whose start time is later than this specified time are queried. The time is in UNIX timestamp format, in milliseconds.
            end=1716199200000  # Change the value to the end time of a user-defined job. Jobs whose start time is earlier than this specified time are queried. The time is in UNIX timestamp format, in milliseconds.
        )
        response = client.list_spark_jobs(request)
    except exceptions.ClientRequestException as e:
        raise Exception("Failed to list Spark jobs") from e
    logger.info("List SparkJobs: %s", response)
```


6 DLI SDK V1 (Not Recommended)

6.1 DLI SDK V1 Function Matrix

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

Table 1 lists the function matrix of DLI's Java and Python SDKs.

Table 6-1 SDK function matrix

Language	Function	Content
Java	Submitting a SQL Job Using an SDK	This section provides instructions on how to authorize DLI's Java SDKs to access and operate on OBS buckets.
	Queue-Related SDKs	This section provides instructions on how to use DLI's Java SDKs to create queues, retrieve the default queue, query all queues, and delete queues.
	Resource-Related SDKs	This section provides instructions on how to use DLI's Java SDKs to upload resource packages, query all resource packages, query specific resource packages, and delete resource packages.
	SDKs Related to SQL Jobs	This section provides instructions on how to use DLI's Java SDKs for database-related operations, table-related operations, and job-related operations.
	SDKs Related to Flink Jobs	This section provides instructions on how to use DLI's Java SDKs to create new Flink jobs, retrieve job details, and list jobs.

Language	Function	Content
	SDKs Related to Spark Jobs	This section provides instructions on how to use DLI's Java SDKs to submit Spark jobs, query all Spark jobs, and delete Spark jobs.
	SDKs Related to Flink Job Templates	This section provides instructions on how to use DLI's Java SDKs to create new Flink job templates, update existing Flink job templates, and delete Flink job templates.
Python	Queue-Related SDKs	This section provides instructions on how to use DLI's Python SDKs to retrieve a list of all queues.
	Resource-Related SDKs	This section provides instructions on how to use DLI's Python SDKs to upload resource packages, query all resource packages, query specific resource packages, and delete resource packages.
	SDKs Related to SQL Jobs	This section provides instructions on how to use DLI's Python SDKs for database-related operations, table-related operations, and job-related operations.
	SDKs Related to Spark Jobs	This section provides instructions on how to use DLI's Python SDKs to submit Spark jobs, cancel Spark jobs, and delete Spark jobs.

6.2 Mapping Between DLI SDK V1 and APIs

OBS Authorization

Table 6-2 Mapping between OBS authorization APIs and SDKs

Class	Method	Java Method	Python Method	API
Authorize	OBS authorization	authorizeBucket	-	POST /v1.0/{project_id}/dli/obs-authorize

Queue-related SDKs

Table 6-3 Mapping between queue-related APIs and SDKs

Class	Method	Java Method	Python Method	API
Queue	Creating a Queue	createQueue	-	POST /v1.0/{project_id}/queues
	Deleting a Queue	deleteQueue	-	DELETE /v1.0/{project_id}/queues/{queue_name}
	Obtaining the Default Queue	getDefaultQueue	-	-
	Querying All Queues	listAllQueues	list_queues	GET/v1.0/{project_id}/queues

Resource-related SDKs

Table 6-4 Mapping between resource-related APIs and SDKs

Class	Method	Java Method	Python Method	API
package Resources	Uploading a Resource Package	uploadResources	upload_resource	POST /v2.0/{project_id}/resources
	Deleting a Resource Package	deleteResource	delete_resource	DELETE /v2.0/{project_id}/resources/{resource_name}
	Querying All Resource Packages	listAllResources	list_resources	GET /v2.0/{project_id}/resources
	Querying a Specified Resource Package	getResource	get_package_resource	GET /v2.0/{project_id}/resources/{resource_name}

SDKs Related to SQL Jobs

Table 6-5 Mapping between SQL job APIs and SDKs

Class	Method	Java Method	Python Method	API
Database	Creating a Database	**createDatabase**	**create_database**	POST /v1.0/{project_id}/databases
	Deleting a Database	**deleteDatabase**	**delete_database**	DELETE /v1.0/{project_id}/databases/{database_name}
	Querying All Databases	**listAllDatabases**	**list_databases**	GET /v1.0/{project_id}/databases
	Modifying a Database User	-	-	PUT /v1.0/{project_id}/databases/{database_name}/owner
Table	Creating a DLI Table	**createDLITable**	**create_dli_table**	POST /v1.0/{project_id}/databases/{database_name}/tables
	Creating an OBS Table	**createObsTable**	**create_obs_table**	POST /v1.0/{project_id}/databases/{database_name}/tables
	Deleting a Table	**deleteTable**	**delete_table**	DELETE /v1.0/{project_id}/databases/{database_name}/tables/{table_name}
	Querying All Tables	**listAllTables**	**list_tables**	GET /v1.0/{project_id}/databases/{database_name}/tables?keyword=tb&with-detail=true
	Describing Table Information	**getTableDetail**	**get_table_schema**	GET /v1.0/{project_id}/databases/{database_name}/tables/{table_name}
	Previewing a Table	-	-	GET /v1.0/{project_id}/databases/{database_name}/tables/{table_name}/preview
	Modifying a Table User	-	-	PUT /v1.0/{project_id}/databases/{database_name}/tables/{table_name}/owner

Class	Method	Java Method	Python Method	API
Job	Importing Data	**submit**	**import_table**	POST /v1.0/{project_id}/jobs/import-table
	Exporting Data	**submit**	**export_table**	POST /v1.0/{project_id}/jobs/export-table
	Submitting a Job	**submit**	**execute_sql**	POST /v1.0/{project_id}/jobs/submit-job
	Canceling a Job	**cancelJob**	-	DELETE /v1.0/{project_id}/jobs/{job_id}
	Querying All Jobs	**listAllJobs**	-	GET /v1.0/{project_id}/jobs?page-size=*{size}*&current-page=*{page_number}*&start=*{start_time}*&end=*{end_time}*&job-type={QUERY}&queue_name={test}&order={duration_desc}
	Querying Job Results	**queryJobResultInfo**	-	GET/v1.0/{project_id}/jobs/{job_id}?page-size=*{size}*&current-page=*{page_number}*
	Querying the Job Status	-	-	GET/v1.0/{project_id}/jobs/{job_id}/status
	Querying Job Details	-	-	GET/v1.0/{project_id}/jobs/{job_id}/detail
	Querying Jobs of the SQL Type	**listSQLJobs**	-	-
	Checking the SQL Syntax	-	-	POST /v1.0/{project_id}/jobs/check-sql
	Exporting Search Results	-	-	POST /v1.0/{project_id}/jobs/{job_id}/export-result

SDKs Related to Flink Jobs

Table 6-6 Mapping between Java and Python SDKs and APIs

Class	Method	Java Method	Python Method	API
Job	Creating a Flink SQL Job	**submitFlinkSqUob**	-	POST /v1.0/{project_id}/streaming/sql-jobs
	Creating a Custom Flink Job	**createFlinkJarJob**	-	POST /v1.0/{project_id}/streaming/flink-jobs
	Updating a Flink SQL Job	**updateFlinkSqUob**	-	PUT /v1.0/{project_id}/streaming/sql-jobs/{job_id}
	Updating a Custom Flink Job	**updateFlinkJarJob**	-	PUT /v1.0/{project_id}/streaming/flink-jobs/{job_id}
	Querying the Flink Job List	**getFlinkJobs**	-	GET /v1.0/{project_id}/streaming/jobs
	Querying Flink Job Details	**getFlinkJobDetail**	-	GET /v1.0/{project_id}/streaming/jobs/{job_id}
	Querying the Flink Job Execution Plan Diagram	**getFlinkJobExecuteGraph**	-	GET /v1.0/{project_id}/streaming/jobs/{job_id}/execute-graph
	Querying Flink Job Monitoring Information	**getFlinkJobsMetrics**	-	POST /v1.0/{project_id}/streaming/jobs/metrics
	Querying the APIG Address of a Flink Job	**getFlinkApigSinks**	-	GET /v1.0/{project_id}/streaming/jobs/{job_id}/apig-sinks
	Running a Flink Job	**runFlinkJob**	-	POST /v1.0/{project_id}/streaming/jobs/run
	Stopping a Flink Job	**stopFlinkJob**	-	POST /v1.0/{project_id}/streaming/jobs/stop

Class	Method	Java Method	Python Method	API
	Deleting Flink Jobs in Batches	deleteFlinkJobInBatch	-	POST /v1.0/{project_id}/streaming/jobs/delete

SDKs Related to Spark Jobs

Table 6-7 Mapping between Spark job APIs and SDKs

Class	Method	Java Method	Python Method	API
BatchJob	Submitting Batch Jobs	asyncSubmit	submit_spark_batch_job	POST /v2.0/{project_id}/batches
	Deleting Batch Jobs	deleteBatchJob	del_spark_batch_job	DELETE /v2.0/{project_id}/batches/{batch_id}
	Querying All Batch Jobs	listAllBatchJobs	-	GET /v2.0/{project_id}/batches
	Querying Batch Job Details	-	-	GET /v2.0/{project_id}/batches/{batch_id}
	Querying the Status of a Batch Processing Job	getStateBatchJob	-	GET /v2.0/{project_id}/batches/{batch_id}/state
	Querying Batch Job Logs	getBatchJobLog	-	GET /v2.0/{project_id}/batches/{batch_id}/log

SDKs Related to Flink Job Templates

Table 6-8 Mapping between Java and Python SDKs and APIs

Class	Java Method	Python Method	API
Template	createFlinkJobTemplate	-	POST /v1.0/{project_id}/streaming/job-templates
	updateFlinkJobTemplate	-	PUT /v1.0/{project_id}/streaming/job-templates/{template_id}

Class	Java Method	Python Method	API
	deleteFlinkJobTemplate	-	DELETE /v1.0/{project_id}/streaming/job-templates/{template_id}
	getFlinkJobTemplates	-	GET /v1.0/{project_id}/streaming/job-templates

6.3 Java SDK (DLI SDK V1)

6.3.1 Overview

Scenario

With DLI's Java SDKs, you can quickly and easily use DLI without worrying about the details of the requests. This section describes how to obtain and use Java SDKs.

Notes

- To use DLI's Java SDKs to access a specific service's API, you need to ensure that the current service has been enabled and authorized on the DLI management console.
- The Java SDKs can be used in Java JDK 1.8 or later. For details about how to configure the Java development environment, see **Configuring the Java SDK Environment**.
- For details about how to obtain and install Java SDKs, see **Obtaining and Installing the Java SDK**.
- To use SDKs to access DLI, you need to initialize the DLI client. During DLI client initialization, you can use the AK/SK or token for authentication. For details, see **Initializing the DLI Client**.

Java SDKs

Table 6-9 Java SDKs

SDK	Description
Submitting a SQL Job Using an SDK	This section provides instructions on how to authorize DLI's Java SDKs to access and operate on OBS buckets.
Queue-Related SDKs	This section provides instructions on how to use DLI's Java SDKs to create queues, retrieve the default queue, query all queues, and delete queues.

SDK	Description
Resource-Related SDKs	This section provides instructions on how to use DLI's Java SDKs to upload resource packages, query all resource packages, query specific resource packages, and delete resource packages.
SDKs Related to SQL Jobs	This section provides instructions on how to use DLI's Java SDKs for database-related operations, table-related operations, and job-related operations.
SDKs Related to Flink Jobs	This section provides instructions on how to use DLI's Java SDKs to create new Flink jobs, retrieve job details, and list jobs.
SDKs Related to Spark Jobs	This section provides instructions on how to use DLI's Java SDK to submit Spark jobs, query all Spark jobs, and delete Spark jobs.
SDKs Related to Flink Job Templates	This section provides instructions on how to use DLI's Java SDKs to create new Flink job templates, update existing Flink job templates, and delete Flink job templates.

6.3.2 Queue-Related SDKs

Prerequisites

- You have configured the Java SDK environment by following the instructions provided **Overview**.
- You have initialized the DLI Client by following the instructions provided in **Initializing the DLI Client**.

Creating a Queue

You can use the API provided by DLI to create a queue. The example code is as follows:

```
private static void createQueue(DLIClient client) throws DLIException {
  //Call the createQueue method of the DLIClient object to create a queue.
  String qName = "queueName";
  int cu = 16;
  String description = "test for sdk";
  Queue queue = client.createQueue(qName, cu, mode, description);
  System.out.println("---------- createQueue success ---------");
}
```

Deleting a Queue

You can use the API provided by DLI to delete the queue. The example code is as follows:

```
private static void deleteQueue(DLIClient client) throws DLIException {
  //Call the getQueue(queueName) method of the DLIClient object to obtain queue queueName.
  String qName = "queueName";
  Queue queue = client.getQueue(qName);
  //Call the deleteQueue() method to delete queue queueName.
```

```
    queue.deleteQueue();
}
```

Obtaining the Default Queue

DLI provides an API for querying the default queue. You can use the default queue to submit jobs. The example code is as follows:

```
private static void getDefaultQueue(DLIClient client) throws DLIException{
  //Call the getDefaultQueue method of the DLIClient object to query the default queue.
    Queue queue = client.getDefaultQueue();
    System.out.println("defaultQueue is:"+ queue.getQueueName());
}
```

NOTE

All users can use the default queue. However, DLI limits the maximum number of times a user can use the default queue.

Querying All Queues

You can use the API provided by DLI to query the queue list and select the corresponding queue to execute the job. The example code is as follows:

```
private static void listAllQueues(DLIClient client) throws DLIException {
  System.out.println("list all queues...");

  //Call the listAllQueues method of the DLIClient object to query the queue list.
  List<Queue> queues = client.listAllQueues();
  for (Queue queue : queues) {
    System.out.println("Queue name:" + queue.getQueueName() + "  " + "cu:" + queue.getCuCount());
  }
}
```

6.3.3 Resource-Related SDKs

Prerequisites

- You have configured the Java SDK environment by following the instructions provided **Overview**.
- You have initialized the DLI Client by following the instructions provided in **Initializing the DLI Client**.

Uploading a Resource Package

You can use the interface provided by the DLI to upload resource packages. The code example is as follows:

```
private static void uploadResources(DLIClient client) throws DLIException {
  String kind = "jar";
  String[] paths = new String[1];
  paths[0] = "https://bucketname.obs.com/jarname.jar";
  String description = "test for sdk";
  //Call the uploadResources method of the DLIClient object to upload resources.
  List<PackageResource> packageResources = client.uploadResources(kind, paths, description);
  System.out.println("---------- uploadResources success ---------");
}
```


NOTE

The following describes the request parameters. For details, see **Overview**.

- **kind**: resource package type. The options are as follows:
  - **jar**: JAR file
  - **Pyfile**: User Python file
  - **file**: User file
  - **modelfile**: User AI model file
- **paths**: OBS path of the resource package. The parameter format is {bucketName}.{obs domain name}/{jarPath}/{jarName}.
- **description**: description of the resource package

Querying All Resource Packages

You can use the API provided by DLI to query the list of uploaded resources. The example code is as follows:

```
private static void listAllResources(DLIClient client) throws DLIException {
  System.out.println("list all resources...");
  //Call the listAllResources method of the DLIClient object to query the queue resource list.
  Resources resources = client.listAllResources();
  for (PackageResource packageResource : resources.getPackageResources()) {
    System.out.println("Package resource name:" + packageResource.getResourceName());
  }
  for (ModuleResource moduleResource : resources.getModuleResources()) {
    System.out.println("Module resource name:" + moduleResource.getModuleName());
  }
}
```

Querying a Specified Resource Package

You can call an API to query information about the specified resource package. The sample code is as follows:

```
private static void getResource(DLIClient client) throws DLIException {
  String resourceName = "xxxxx";
  // group: If the resource package is not in the group, leave this parameter empty.
  String group= "xxxxxx";
  // Call getResource on the DLIClient object to query a specified resource package.
  PackageResource packageResource=client.getResource(resourceName,group);
  System.out.println(packageResource);
}
```

Deleting a Resource Package

You can call an API to delete an uploaded resource package. Sample code is as follows:

```
private static void deleteResource(DLIClient client) throws DLIException {
  String resourceName = "xxxxx";
  // group: If the resource package is not in the group, leave this parameter empty.
  String group= "xxxxxx";
  //Call deleteResource on the DLIClient object to upload resources.
  client.deleteResource(resourceName,group);
  System.out.println("---------- deleteResource success ---------");
}
```

6.3.4 SDKs Related to SQL Jobs

6.3.4.1 Database-Related SDKs

Prerequisites

- You have configured the Java SDK environment by following the instructions provided **Overview**.
- You have initialized the DLI Client by following the instructions provided in **Initializing the DLI Client** and created queues by following the instructions provided in **Queue-Related SDKs**.

Creating a Database

DLI provides an API for creating a database. You can use the API to create a database. The sample code is as follows:

```
private static Database createDatabase(DLIClient client) throws DLIException {
  //Call the createDatabase method of the DLIClient object to create a database.
  String dbName = "databasename";
  Database database = client.createDatabase(dbName);
  System.out.println("create database:" + database);
  return database;
}
```

NOTE

The **default** database is a built-in database. You are not allowed to create a database named **default**.

Deleting a Database

DLI provides an API for deleting a database. The example code is as follows:

```
  //Call the deleteDatabase interface of the Database object to delete a database.
  //Call the getDatabase(String databaseName) interface of the DLIClient object to obtain the Database object.
private static void deletedatabase(Database database) throws DLIException {
   String dbName = "databasename";
   database=client.getDatabase(dbName);
   database.deleteDatabase();
   System.out.println("delete db " + dbName);
}
```

NOTE

- A database that contains tables cannot be deleted. To delete a database that contains tables, delete the tables first.
- A deleted database cannot be restored. Therefore, exercise caution when deleting a database.

Querying All Databases

You can use the API provided by DLI to query the list of created databases. The example code is as follows:

```
private static void listDatabases(DLIClient client) throws DLIException {
  //Call the listAllDatabases method of the DLIClient object to query the database list.
  List<Database> databases = client.listAllDatabases();
  for (Database db : databases) {
    System.out.println("dbName:" + db.getDatabaseName() + " " + "tableCount:" + db.getTableCount());
  }
}
```

6.3.4.2 Table-Related SDKs

Creating a DLI Table

DLI provides an API for creating DLI tables. You can use it to create a table for storing DLI data. The example code is as follows:

```
private static Table createDLITable(Database database) throws DLIException {
  // Construct a table column set and instantiate the Column object to construct columns.
  List<Column> columns = new ArrayList<Column>();
  Column c1 = new Column("c1", DataType.STRING, "desc for c1");
  Column c2 = new Column("c2", DataType.INT, "desc for c2");
  Column c3 = new Column("c3", DataType.DOUBLE, "desc for c3");
  Column c4 = new Column("c4", DataType.BIGINT, "desc for c4");
  Column c5 = new Column("c5", DataType.SHORT, "desc for c5");
  Column c6 = new Column("c6", DataType.LONG, "desc for c6");
  Column c7 = new Column("c7", DataType.SMALLINT, "desc for c7");
  Column c8 = new Column("c8", DataType.BOOLEAN, "desc for c8");
  Column c9 = new Column("c9", DataType.DATE, "desc for c9");
  Column c10 = new Column("c10", DataType.TIMESTAMP, "desc for c10");
  Column c11 = new Column("c11", DataType.DECIMAL, "desc for c11");
  columns.add(c1);
  columns.add(c2);
  columns.add(c3);
  columns.add(c4);
  columns.add(c5);
  columns.add(c6);
  columns.add(c7);
  columns.add(c8);
  columns.add(c9);
  columns.add(c10);
  columns.add(c11);

  List<String> sortColumns = new ArrayList<String>();
  sortColumns.add("c1");
  String DLITblName = "tablename";
  String desc = "desc for table";
  // Call the createDLITable method of the Database object to create a DLI table.
  Table table = database.createDLITable(DLITblName, desc, columns, sortColumns);
  System.out.println(table);
  return table;
}
```

NOTE

The default precision of **DataType.DECIMAL** is **(10,0)**. To set the precision of data of the decimal type, perform the following operation:

```
Column c11 = new Column("c11", new DecimalTypeInfo(25,5), "test for c11");
```

Creating an OBS Table

DLI provides an API for creating OBS tables. You can use it to create a table for storing OBS data. The example code is as follows:

```
private static Table createObsTable(Database database) throws DLIException {
  // Construct a table column set and instantiate the Column object to construct columns.
  List < Column > columns = new ArrayList < Column > ();
  Column c1 = new Column("c1", DataType.STRING, "desc for c1");
  Column c2 = new Column("c2", DataType.INT, "desc for c2");
  Column c3 = new Column("c3", DataType.DOUBLE, "desc for c3");
  Column c4 = new Column("c4", DataType.BIGINT, "desc for c4");
  Column c5 = new Column("c5", DataType.SHORT, "desc for c5");
  Column c6 = new Column("c6", DataType.LONG, "desc for c6");
  Column c7 = new Column("c7", DataType.SMALLINT, "desc for c7");
  Column c8 = new Column("c8", DataType.BOOLEAN, "desc for c8");
  Column c9 = new Column("c9", DataType.DATE, "desc for c9");
```



```
    Column c10 = new Column("c10", DataType.TIMESTAMP, "desc for c10");
    Column c11 = new Column("c11", DataType.DECIMAL, "desc for c11");
    columns.add(c1);
    columns.add(c2);
    columns.add(c3);
    columns.add(c4);
    columns.add(c5);
    columns.add(c6);
    columns.add(c7);
    columns.add(c8);
    columns.add(c9);
    columns.add(c10);
    columns.add(c11);
    CsvFormatInfo formatInfo = new CsvFormatInfo();
    formatInfo.setWithColumnHeader(true);
    formatInfo.setDelimiter(",");
    formatInfo.setQuoteChar("\"");
    formatInfo.setEscapeChar("\\");
    formatInfo.setDateFormat("yyyy/MM/dd");
    formatInfo.setTimestampFormat("yyyy-MM-dd HH:mm:ss");
    String obsTblName = "tablename";
    String desc = "desc for table";
    String dataPath = "OBS path";
    // Call the createObsTable method of the Database object to create an OBS table.
    Table table = database.createObsTable(obsTblName, desc, columns,StorageType.CSV , dataPath,
formatInfo);
    System.out.println(table);
    return table;
}
```

NOTE

The default precision of **DataType.DECIMAL** is **(10,0)**. To set the precision of data of the decimal type, perform the following operation:

```
Column c11 = new Column("c11", new DecimalTypeInfo(25,5), "test for c11");
```

Deleting a Table

DLI provides an API for deleting tables. You can use it to delete all the tables in a database. The example code is as follows:

```
private static void deleteTables(Database database) throws DLIException {
    // Call the listAllTables interface of the Database object to query all tables.
    List<Table> tables = database.listAllTables();
    for (Table table : tables) {
    // Traverse tables and call the deleteTable interface of the Table object to delete tables.
      table.deleteTable();
      System.out.println("delete table " + table.getTableName());
    }
  }
```

NOTE

A deleted table cannot be restored. Therefore, exercise caution when deleting a table.

Querying All Tables

DLI provides an API for querying tables. You can use it to query all tables in a database. The example code is as follows:

```
 private static void listTables(Database database) throws DLIException {
    // Call the listAllTables method of the Database object to query all tables in a database.
    List<Table> tables = database.listAllTables(true);
    for (Table table : tables) {
      System.out.println(table);
    }
  }
```


Querying the Partition Information of a Table (Including the Creation and Modification Time of the Partition).

DLI provides an API for querying table partition information. You can use it to query the partition information (including the creation time and modification time) in the database table. The example code is as follows:

```
private static void showPartitionsInfo(DLIClient client) throws DLIException {
    String databaseName = "databasename";
    String tableName = "tablename";
    // Call the showPartitions method of the DLIClient object to query the partition information (including the partition creation and modification time) in the database table.
    PartitionResult partitionResult = client.showPartitions(databaseName, tableName);
    PartitionListInfo partitonInfos = partitionResult.getPartitions();
    // Obtain the creation and modification time of the partition.
    Long createTime = partitonInfos.getPartitionInfos().get(0).getCreateTime().longValue();
    Long lastAccessTime = partitonInfos.getPartitionInfos().get(0).getLastAccessTime().longValue();
    System.out.println("createTime:"+createTime+"\nlastAccessTime:"+lastAccessTime);
}
```

Describing Table Information

You can call an API to obtain the metadata description of a table. The example code is as follows:

```
private static void getTableDetail(Table table) throws DLIException {
  // Call getTableDetail on the Table object to obtain table information.
  //  TableSchema tableSchema=table.getTableDetail();
  // Output
  System.out.println(List<Column> columns = tableSchema.getColumns());
  System.out.println(List<String> sortColumns = tableSchema.getSortColumns());
  System.out.println(String createTableSql = tableSchema.getCreateTableSql());
  System.out.println(String tableComment = tableSchema.getTableComment());
}
```

6.3.4.3 Job-related SDKs

Importing Data

DLI provides an API for importing data. You can use it to import data stored in OBS to a created DLI or OBS table. The example code is as follows:

```
//Instantiate the importJob object. The input parameters of the constructor include the queue, database name, table name (obtained by instantiating the Table object), and data path.
private static void importData(Queue queue, Table DLITable) throws DLIException {
    String dataPath = "OBS Path";
    queue = client.getQueue("queueName");
    CsvFormatInfo formatInfo = new CsvFormatInfo();
    formatInfo.setWithColumnHeader(true);
    formatInfo.setDelimiter(",");
    formatInfo.setQuoteChar("\"");
    formatInfo.setEscapeChar("\\");
    formatInfo.setDateFormat("yyyy/MM/dd");
    formatInfo.setTimestampFormat("yyyy-MM-dd HH:mm:ss");
    String dbName = DLITable.getDb().getDatabaseName();
    String tableName = DLITable.getTableName();
    ImportJob importJob = new ImportJob(queue, dbName, tableName, dataPath);
    importJob.setStorageType(StorageType.CSV);
    importJob.setCsvFormatInfo(formatInfo);
    System.out.println("start submit import table: " + DLITable.getTableName());
    //Call the submit interface of the ImportJob object to submit the data importing job.
    importJob.submit(); //Call the getStatus interface of the ImportJob object to query the status of the data importing job.
    JobStatus status = importJob.getStatus();
```



```
    System.out.println("Job id: " + importJob.getJobId() + ", Status : " + status.getName());
}
```

NOTE

- Before submitting the data importing job, you can set the format of the data to be imported. In the sample code, the setStorageType interface of the ImportJob object is called to set the data storage type to csv. The data format is set by calling the setCsvFormatInfo interface of the ImportJob object.
- Before submitting the data import job, you can set the partition of the data to be imported and whether to overwrite the data. You can call the **setPartitionSpec** API of the **ImportJob** object to set the partition information, for example, **importJob.setPartitionSpec(new PartitionSpec("part1=value1,part2=value2"))**. You can also create the partition using parameters when creating the **ImportJob** object. By default, data is appended to an import job. To overwrite the existing data, call the **setOverWrite** API of the **ImportJob** object, for example, **importJob.setOverWrite(Boolean.TRUE)**.
- If a folder and a file under an OBS bucket directory have the same name, data is preferentially loaded to the file, instead of the folder. It is recommended that the files and folders of the same level have different names when you create an OBS object.

Importing the Partition Data

DLI provides an API for importing data. You can use it to import data stored in OBS to a specified partition of the created DLI or OBS table. The example code is as follows:

```
//Instantiate the importJob object. The input parameters of the constructor include the queue, database name, table name (obtained by instantiating the Table object), and data path.
private static void importData(Queue queue, Table DLITable) throws DLIException {
    String dataPath = "OBS Path";
    queue = client.getQueue("queueName");
    CsvFormatInfo formatInfo = new CsvFormatInfo();
    formatInfo.setWithColumnHeader(true);
    formatInfo.setDelimiter(",");
    formatInfo.setQuoteChar("\"");
    formatInfo.setEscapeChar("\\");
    formatInfo.setDateFormat("yyyy/MM/dd");
    formatInfo.setTimestampFormat("yyyy-MM-dd HH:mm:ss");
    String dbName = DLITable.getDb().getDatabaseName();
    String tableName = DLITable.getTableName();
    PartitionSpec partitionSpec = new PartitionSpec("part1=value1,part2=value2");
    Boolean isOverWrite = true;
    ImportJob importJob = new ImportJob(queue, dbName, tableName, dataPath, partitionSpec, isOverWrite);
    importJob.setStorageType(StorageType.CSV);
    importJob.setCsvFormatInfo(formatInfo);
    System.out.println("start submit import table: " + DLITable.getTableName());
    //Call the submit interface of the ImportJob object to submit the data importing job.
    importJob.submit(); //Call the getStatus interface of the ImportJob object to query the status of the data importing job.
    JobStatus status = importJob.getStatus();
    System.out.println("Job id: " + importJob.getJobId() + ", Status : " + status.getName());
}
```


NOTE

- When the **ImportJob** object is created, the partition information **PartitionSpec** can also be directly transferred as the partition character string.
- If some columns are specified as partition columns during **partitionSpec** import but the imported data contains only the specified partition information, the unspecified partition columns after data import contain abnormal values such as null.
- In the example, **isOverWrite** indicates whether to overwrite data. The value **true** indicates that data is overwritten, and the value **false** indicates that data is appended. Currently, **overwrite** is not supported to overwrite the entire table. Only the specified partition can be overwritten. To append data to a specified partition, set **isOverWrite** to **false** when creating the import job.

Exporting Data

DLI provides an API for exporting data. You can use it to export data from a DLI table to OBS. The example code is as follows:

```
//Instantiate the ExportJob object and transfer the queue, database name, table name (obtained by instantiating the Table object), and storage path of the exported data. The table type must be MANAGED.
private static void exportData(Queue queue, Table DLITable) throws DLIException {
    String dataPath = "OBS Path";
    queue = client.getQueue("queueName");
    String dbName = DLITable.getDb().getDatabaseName();
    String tableName = DLITable.getTableName();
    ExportJob exportJob = new ExportJob(queue, dbName, tableName, dataPath);
    exportJob.setStorageType(StorageType.CSV);
    exportJob.setCompressType(CompressType.GZIP);
    exportJob.setExportMode(ExportMode.ERRORIFEXISTS);
    System.out.println("start export DLI Table data...");
    // Call the submit interface of the ExportJob object to submit the data exporting job.
    exportJob.submit();
    // Call the getStatus interface of the ExportJob object to query the status of the data exporting job.
    JobStatus status = exportJob.getStatus();
    System.out.println("Job id: " + exportJob.getJobId() + ", Status : " + status.getName());
}
```

NOTE

- Before submitting the data exporting job, you can optionally set the data format, compression type, and export mode. In the preceding sample code, the setStorageType, setCompressType, and setExportMode interfaces of the ExportJob object are called to set the data format, compression type, and export mode, respectively. The setStorageType interface supports only the CSV format.
- If a folder and a file under an OBS bucket directory have the same name, data is preferentially loaded to the file, instead of the folder. It is recommended that the files and folders of the same level have different names when you create an OBS object.

Submitting a Job

DLI provides APIs for submitting and querying jobs. You can submit a job by calling the API. You can also call the API to query the job result. The example code is as follows:

```
//Instantiate the SQLJob object and construct input parameters for executing SQL, including the queue, database name, and SQL statements.
private static void runSqlJob(Queue queue, Table obsTable) throws DLIException {
    String sql = "select * from " + obsTable.getTableName();
    String queryResultPath = "OBS Path";
    SQLJob sqlJob = new SQLJob(queue, obsTable.getDb().getDatabaseName(), sql);
    System.out.println("start submit SQL job...");
    // Call the submit interface of the SQLJob object to submit the querying job.
```



```
    sqlJob.submit();
    // Call the getStatus interface of the SQLJob object to query the status of the querying job.
    JobStatus status = sqlJob.getStatus();
    System.out.println(status);
    System.out.println("start export Result...");
    //Call the exportResult interface of the SQLJob object to export the query result. queryResultPath refers
to the path of the data to be exported.
    sqlJob.exportResult(queryResultPath, StorageType.CSV,
            CompressType.GZIP, ExportMode.ERRORIFEXISTS, null);
    System.out.println("Job id: " + sqlJob.getJobId() + ", Status : " + status.getName());
}
```

Canceling a Job

DLI provides an API for canceling jobs. You can use it to cancel all jobs in the **Launching** or **Running** state. The following sample code is used for canceling jobs in the **Launching** state:

```
private static void cancelSqlJob(DLIClient client) throws DLIException {

  List<JobResultInfo> jobResultInfos = client.listAllJobs(JobType.QUERY);
  for (JobResultInfo jobResultInfo : jobResultInfos) {
      //Cancel jobs in the LAUNCHING state.
      if (JobStatus.LAUNCHING.equals(jobResultInfo.getJobStatus())) {
          //Cancel the job of a specific job ID.
          client.cancelJob(jobResultInfo.getJobId());
      }
  }
}
```

Querying All Jobs

DLI provides an API for querying jobs. You can use it to query all jobs of the current project. The example code is as follows:

```
private static void listAllSqlJobs(DLIClient client) throws DLIException {
   //Return the collection of JobResultInfo lists.
    List < JobResultInfo > jobResultInfos = client.listAllJobs();
   //Traverse the JobResultInfo lists to view job information.
    for (JobResultInfo jobResultInfo: jobResultInfos) {
        //job id
        System.out.println(jobResultInfo.getJobId());
        //Job description
        System.out.println(jobResultInfo.getDetail());
        //job status
        System.out.println(jobResultInfo.getJobStatus());
        //job type
        System.out.println(jobResultInfo.getJobType());
    }
   //Filter the query result by job type.
    List < JobResultInfo > jobResultInfos1 = client.listAllJobs(JobType.DDL);
   //Filter the query result by job type and start time that is in the Unix timestamp format.
    List < JobResultInfo > jobResultInfos2 = client.listAllJobs(1502349803729L, 1502349821460L,
JobType.DDL);
   //Filter the query result by page.
    List < JobResultInfo > jobResultInfos3 = client.listAllJobs(100, 1, JobType.DDL);
   //Filter the query result by page, start time, and job type.
    List < JobResultInfo > jobResultInfos4 = client.listAllJobs(100, 1, 1502349803729L, 1502349821460L,
JobType.DDL);

    // Use Tags to query jobs that meet the conditions.
    JobFilter jobFilter = new JobFilter();
    jobFilter.setTags("workspace=space002,jobName=name002");
    List < JobResultInfo > jobResultInfos1 = client.listAllJobs(jobFilter);
    // Use Tags to query target jobs of a specified page.
    JobFilter jobFilter = new JobFilter();
    jobFilter.setTags("workspace=space002,jobName=name002");
    jobFilter.setPageSize(100);
```



```
    jobFilter.setCurrentPage(0);
    List < JobResultInfo > jobResultInfos1 = client.listJobsByPage(jobFilter);
}
```

NOTE

- Parameters in the OVERRIDE method can be set to **null**, indicating that no filter conditions are specified. Ensure that all parameters are set to valid values. If the page parameter is set to **-1**, the query will fail.
- APIs in this SDK do not support SQL patterns. You cannot match SQL patterns for job query.

  To query DLI jobs, use the **Querying All Jobs** API.

Querying Job Results

DLI provides an API for querying job results. You can use it to query information about a job of the specific job ID. The example code is as follows:

```
private static void getJobResultInfo(DLIClient client) throws DLIException {
    String jobId = "4c4f7168-5bc4-45bd-8c8a-43dfc85055d0";
    JobResultInfo jobResultInfo = client.queryJobResultInfo(jobId);
   //View information about a job.
    System.out.println(jobResultInfo.getJobId());
    System.out.println(jobResultInfo.getDetail());
    System.out.println(jobResultInfo.getJobStatus());
    System.out.println(jobResultInfo.getJobType());
}
```

Querying Jobs of the SQL Type

DLI provides an API for querying SQL jobs. You can use it to query information about recently executed jobs submitted using SQL statements in the current project. The example code is as follows:

```
private static void getJobResultInfos(DLIClient client) throws DLIException {

  //Return the collection of JobResultInfo lists.
  List<JobResultInfo> jobResultInfos = client.listSQLJobs();
  //Traverse the list to view job information.
  for (JobResultInfo jobResultInfo : jobResultInfos) {
        //job id
        System.out.println(jobResultInfo.getJobId());
        //Job description
        System.out.println(jobResultInfo.getDetail());
        //job status
        System.out.println(jobResultInfo.getJobStatus());
        //job type
        System.out.println(jobResultInfo.getJobType());
   }

    // Use Tags to query SQL jobs that meet the conditions.
    JobFilter jobFilter = new JobFilter();
    jobFilter.setTags("workspace=space002,jobName=name002");
    List < JobResultInfo > jobResultInfos1 = client.listAllSQLJobs(jobFilter);
    // Use Tags to query target SQL jobs of a specified page.
    JobFilter jobFilter = new JobFilter();
    jobFilter.setTags("workspace=space002,jobName=name002");
    jobFilter.setPageSize(100);
    jobFilter.setCurrentPage(0);
    List < JobResultInfo > jobResultInfos1 = client.listSQLJobsByPage(jobFilter);
}
```


Exporting Query Results

DLI provides an API for exporting query results. You can use the API to export the query job result submitted in the editing box of the current project. The example code is as follows:

```
 //Instantiate the SQLJob object and construct input parameters for executing SQL, including the queue, database name, and SQL statements.
 private static void exportSqlResult(Queue queue, Table obsTable) throws DLIException {
   String sql = "select * from " + obsTable.getTableName();
   String queryResultPath = "OBS Path";
   SQLJob sqlJob = new SQLJob(queue, obsTable.getDb().getDatabaseName(), sql);
   System.out.println("start submit SQL job...");
   //Call the submit interface of the SQLJob object to submit the querying job.
    sqlJob.submit();
   //Call the getStatus interface of the SQLJob object to query the status of the querying job.
    JobStatus status = sqlJob.getStatus();
    System.out.println(status);
    System.out.println("start export Result...");
   //Call the exportResult interface of the SQLJob object to export the query result. exportPath indicates the path for exporting data. JSON indicates the export format. queueName indicates the queue for executing the export job. limitNum indicates the number of results of the export job. 0 indicates that all data is exported.
   sqlJob.exportResult(exportPath + "result", StorageType.JSON, CompressType.NONE,
       ExportMode.ERRORIFEXISTS, queueName, true, 5);
 }
```

Previewing Job Results

DLI provides an API for previewing job results. You can call this API to obtain the first 1000 records in the result set.

```
// Initialize a SQLJob object and pass the queue, database name, and SQL statement to execute the SQL.
private static void getPreviewJobResult(Queue queue, Table obsTable) throws DLIException {
   String sql = "select * from " + obsTable.getTableName();
   SQLJob sqlJob = new SQLJob(queue, obsTable.getDb().getDatabaseName(), sql);
   System.out.println("start submit SQL job...");
   // Call the submit method on the SQLJob object.
   sqlJob.submit();
   // Call the previewJobResult method on the SQLJob object to query the first 1000 records in the result set.
   List<Row> rows = sqlJob.previewJobResult();
   if (rows.size() > 0) {
       Integer value = rows.get(0).getInt(0);
       System.out.println("Obtain the data value in the first column at the first row." + value);
   }
   System.out.println("Job id: " + sqlJob.getJobId() + ", previewJobResultSize : " + rows.size());
}
```

Deprecated API

The **getJobResult** method has been discarded. You can call **DownloadJob** instead to obtain the job result.

For details about the **DownloadJob** method, obtain the **dli-sdk-java-x.x.x.zip** package by referring to **Obtaining and Installing the Java SDK** and decompress the package.

6.3.5 SDKs Related to Flink Jobs

Prerequisites

- You have configured the Java SDK environment by referring to **Overview**.
- You have initialized the DLI client by referring to **Initializing the DLI Client** and created queues by referring to **Queue-Related SDKs**.

Creating a SQL Job

DLI provides an API for creating a Flink streaming SQL job. You can use it to create a Flink streaming SQL job and submit it to DLI. Sample code is as follows:

```
private static void createSQLJob(DLIClient client) throws DLIException {
    SubmitFlinkSqlJobRequest body = new SubmitFlinkSqlJobRequest();
    body.name("job-name");
    body.runMode(SubmitFlinkSqlJobRequest.RunModeEnum.SHARED_CLUSTER);
    body.checkpointEnabled(false);
    body.checkpointMode(1);
    body.jobType(SubmitFlinkSqlJobRequest.JobTypeEnum.JOB);
    JobStatusResponse result = client.submitFlinkSqlJob(body);
    System.out.println(result);
}
```

Customizing a Job

DLI provides an API for creating a user-defined Flink job. Currently, the job supports the JAR format and runs in dedicated queues. The example code is as follows:

```
private static void createFlinkJob(DLIClient client) throws DLIException {
     CreateFlinkJarJobRequest body = new CreateFlinkJarJobRequest();
     body.name("jar-job");
     body.cuNumber(2);
     body.managerCuNumber(1);
     body.parallelNumber(1);
     body.entrypoint("dli/WindowJoin.jar");
     JobStatusResponse result = client.createFlinkJarJob(body);
     System.out.println(result);
 }
```

Updating a SQL Job

DLI provides an API for updating Flink streaming SQL jobs. You can use it to update a Flink streaming SQL job. Sample code is as follows:

```
private static void updateSQLJob(DLIClient client) throws DLIException {
     UpdateFlinkSqlJobRequest body = new UpdateFlinkSqlJobRequest();
     body.name("update-job");
     JobUpdateResponse result = client.updateFlinkSqlJob(body,203L);
     System.out.println(result);
 }
```

Updating a Custom Job

DLI provides an API for updating user-defined Flink jobs. You can use it to update custom jobs, which currently support the JAR format and run in dedicated queues. The example code is as follows:

```
  private static void updateFlinkJob(DLIClient client) throws DLIException {
      UpdateFlinkJarJobRequest body = new UpdateFlinkJarJobRequest();
      body.name("update-job");
      JobUpdateResponse result = client.updateFlinkJarJob(body,202L);
      System.out.println(result);
   }
```

Querying the List of Jobs

DLI provides an API for querying the Flink job list. The following parameters are involved in this API: **name**, **status**, **show_detail**, **cursor**, **next**, **limit**, and **order**. In

this example, the query results are displayed in descending order and information about the jobs whose IDs are less than the value of **cursor** is displayed. The example code is as follows:

```
private static void  QueryFlinkJobListResponse(DLIClient client) throws DLIException {
        QueryFlinkJobListResponse result = client.getFlinkJobs(null, "job_init", null, true, 0L, 10, null,
null,null,null,null);
        System.out.println(result);
    }
```

Querying Job Details

DLI provides an API for querying Flink job details. The example code is as follows:

```
private static void  getFlinkJobDetail(DLIClient client) throws DLIException {
Long jobId = 203L; //Job ID
    GetFlinkJobDetailResponse result = client.getFlinkJobDetail(jobId);
    System.out.println(result);
}
```

Querying the Job Execution Plan Diagram

DLI provides an API for querying the execution plan of a Flink job. The example code is as follows:

```
private static void  getFlinkJobExecuteGraph(DLIClient client) throws DLIException {
Long jobId = 203L; //Job ID
        FlinkJobExecutePlanResponse result = client.getFlinkJobExecuteGraph(jobId);
        System.out.println(result);
    }
```

Querying Job Monitoring Information

DLI provides an API for querying Flink job monitoring information. Monitoring information about multiple jobs can be queried at the same time. The example code is as follows:

```
public static void getMetrics(DLIClient client) throws DLIException{
    List < Long > job_ids = new ArrayList < > ();
    Long jobId = 6316L; //Job 1 ID
    Long jobId2 = 6945L; //Job 2 ID
    job_ids.add(jobId);
    job_ids.add(jobId2);
    GetFlinkJobsMetricsBody body = new GetFlinkJobsMetricsBody();
    body.jobIds(job_ids);
    QueryFlinkJobMetricsResponse result = client.getFlinkJobsMetrics(body);
    System.out.println(result);
}
```

Querying the APIG Address of a Job

DLI provides an API for querying the APIG access address of a Flink job. The example code is as follows:

```
 private static void  getFlinkApigSinks(DLIClient client) throws DLIException {
        Long jobId = 59L; //Job 1 ID
        FlinkJobApigSinksResponse result = client.getFlinkApigSinks(jobId);
        System.out.println(result);
    }
```

Running a Job

DLI provides APIs for running Flink jobs. The example code is as follows:


```
public static void runFlinkJob(DLIClient client) throws DLIException{
   RunFlinkJobRequest body = new RunFlinkJobRequest();
    List<Long> jobIds = new ArrayList<>();
    Long jobId = 59L; //Job 1 ID
    Long jobId2 = 192L; //Job 2 ID
    jobIds.add(jobId);
    jobIds.add(jobid2);
    body.resumeSavepoint(false);
    body.jobIds(jobIds);
    List<GlobalBatchResponse> result = client.runFlinkJob(body);
    System.out.println(result);
}
```

Stopping a Job

DLI provides an API for stopping Flink jobs. The example code is as follows:

```
public static void stopFlinkJob(DLIClient client) throws DLIException{
    StopFlinkJobRequest body = new StopFlinkJobRequest();
    List<Long> jobIds = new ArrayList<>();
    Long jobId = 59L; //Job 1 ID
    Long jobId2 = 192L; //Job 2 ID
    jobIds.add(jobId);
    jobIds.add(jobid2);
    body.triggerSavepoint(false);
    body.jobIds(jobIds);
    List<GlobalBatchResponse> result = client.stopFlinkJob(body);
    System.out.println(result);
}
```

Deleting Jobs in Batches

DLI provides an API for deleting Flink jobs in batches. You can use the API to batch delete Flink jobs in any status. The example code is as follows:

```
public static void deleteFlinkJob(DLIClient client) throws DLIException{
   DeleteJobInBatchRequest  body = new DeleteJobInBatchRequest ();
   List<Long> jobIds = new ArrayList<>();
   Long jobId = 202L; //Job 1 ID
   Long jobid2 = 203L; //Job 2 ID
   jobIds.add(jobId);
   jobIds.add(jobid2);
   body.jobIds(jobIds);
   List<GlobalBatchResponse>  result = client. deleteFlinkJobInBatch(body);
   System.out.println(result);
}
```

6.3.6 SDKs Related to Spark Jobs

Prerequisites

- You have configured the Java SDK environment by following the instructions provided **Overview**.
- You have initialized the DLI Client by following the instructions provided in **Initializing the DLI Client** and created queues by following the instructions provided in **Queue-Related SDKs**.

Submitting Batch Jobs

DLI provides an API to perform batch jobs. The example code is as follows:

```
private static void runBatchJob(Cluster cluster) throws DLIException {
   SparkJobInfo jobInfo = new SparkJobInfo();
```

```
    jobInfo.setClassName("your.class.name");
    jobInfo.setFile("xxx.jar");
    jobInfo.setCluster_name("queueName");
   // Call the asyncSubmit method on the BatchJob object to submit the batch job.
    BatchJob job = new BatchJob(cluster, jobInfo);
    job.asyncSubmit();
    while (true) {
        SparkJobStatus jobStatus = job.getStatus();
        if (SparkJobStatus.SUCCESS.equals(jobStatus)) {
            System.out.println("Job finished");
            return;
        }
        if (SparkJobStatus.DEAD.equals(jobStatus)) {
            throw new DLIException("The batch has already exited");
        }
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

NOTE

- A cluster is a queue created by a user.
- The input parameter cannot be in JSON format.
- DLI provides the following two APIs related to batch jobs:
  - asyncSubmit: This API is asynchronous. After the API is submitted, the job result is directly returned.
  - submit: This API is synchronous. After the API is submitted, the result is returned only after job execution is complete.

Deleting Batch Jobs

DLI provides an API for deleting batch processing jobs. The example code is as follows:

```
private static void deleteBatchJob(DLIClient client) throws DLIException {
    // Submit the ID of the Spark batch processing job.
    String batchId = "0aae0dc5-f009-4b9b-a8c3-28fbee399fa6";
   // Call the delBatch method on the BatchJob object to cancel the batch job.
    MessageInfo messageInfo = client.delBatchJob(batchId);
    System.out.println(messageInfo.getMsg());
}
```

Querying All Batch Jobs

DLI provides an API for querying batch processing jobs. You can use it to query all batch jobs of the current project. The example code is as follows:

```
private static void listAllBatchJobs(DLIClient client) throws DLIException {
  System.out.println("list all batch jobs...");
  // Call the listAllBatchJobs method on the DLIClient object to query the batch jobs.
  String queueName = "queueName";
  int from = 0;
  int size = 1000;
  // Set paging, start page, and size of each page.
  List<SparkJobResultInfo> jobResults = client.listAllBatchJobs(queueName, from, size);
  for (SparkJobResultInfo jobResult : jobResults) {
    // Job ID
    System.out.println(jobResult.getId());
    // Job app ID
    System.out.println(jobResult.getAppId());
```



```
    // Job status
    System.out.println(jobResult.getState());
  }
}
```

Querying a Batch Job Status

DLI provides an API for querying status of batch processing jobs. The following sample code calls the API to query the job status:

```
private static void getStateBatchJob(DLIClient client) throws DLIException  {
        BatchJob batchJob = null;
        SparkJobInfo jobInfo = new SparkJobInfo();
        jobInfo.setClusterName("queueName");
        jobInfo.setFile("xxx.jar");
        jobInfo.setClassName("your.class.name");
        batchJob = new BatchJob(client.getCluster("queueName"), jobInfo);
        batchJob.asyncSubmit();
        SparkJobStatus sparkJobStatus=batchJob.getStatus();
        System.out.println(sparkJobStatus);

}
```

Querying Batch Job Logs

DLI provides an API for querying logs of batch processing jobs. The following sample code calls the API to query the job logs:

```
private static void getBatchJobLog(DLIClient client) throws DLIException  {
        BatchJob batchJob = null;
        SparkJobInfo jobInfo = new SparkJobInfo();
        jobInfo.setClusterName("queueName");
        jobInfo.setFile("xxx.jar");
        jobInfo.setClassName("your.class.name");
        batchJob = new BatchJob(client.getCluster("queueName"), jobInfo);
        batchJob.submit();
        // Call getLog on the BatchJob object to query the batch processing job.
        int from = 0;
        int size = 1000;
        List<String> jobLogs = batchJob.getLog(from,size);
        System.out.println(jobLogs);

}
```

6.3.7 SDKs Related to Flink Job Templates

Prerequisites

- You have configured the Java SDK environment by following the instructions provided **Overview**.
- You have initialized the DLI Client by following the instructions provided in **Initializing the DLI Client**.

Creating a Job Template

DLI provides an API for creating a Flink job template. The example code is as follows:

```
public static void createFlinkJobTemplate(DLIClient client) throws DLIException{
    CreateFlinkJobTemplateRequest body = new CreateFlinkJobTemplateRequest();
    body.name("template");
    FlinkJobTemplateCreateResponse  result = client.createFlinkJobTemplate(body);
    System.out.println(result);
}
```

Updating a Job Template

DLI provides an API for updating a Flink job template. The example code is as follows:

```
public static void updateFlinkJobTemplate(DLIClient client) throws DLIException{
    Long templateId = 277L; //Template ID
    UpdateFlinkJobTemplateRequest body = new UpdateFlinkJobTemplateRequest();
    body.name("template-update");
    GlobalResponse result = client.updateFlinkJobTemplate(body,templateId);
    System.out.println(result);
}
```

Deleting a Job Template

DLI provides an API for deleting a Flink job template. A template used by jobs can also be deleted. The example code is as follows:

```
public static void deleteFlinkJobTemplate(DLIClient client) throws DLIException{
    Long templateId = 277L; //Template ID
    FlinkJobTemplateDeleteResponse result = client.deleteFlinkJobTemplate(templateId);
    System.out.println(result);
}
```

Querying the List of Job Templates

DLI provides an API for querying Flink job templates. In this example, the query results are displayed in descending order and information about the job templates whose IDs are less than the value of **cursor** is displayed. The example code is as follows:

```
public static void getFlinkJobTemplates(DLIClient client) throws DLIException{
    Long offset = 789L; // Long | Template offset.
    Integer limit = 56; // Integer | Maximum number of records that can be queried.
    String order = "asc"; // String | Query result display. The query results can be displayed in ascending or descending order.
    FlinkJobTemplateListResponse result = client.getFlinkJobTemplates(offset,limit,order);
    System.out.println(result);
}
```

6.4 Python SDK (DLI SDK V1)

6.4.1 Overview

Scenario

With DLI's SDKs, you can quickly and easily use DLI without worrying about the details of the requests. This section describes how to obtain and use SDKs in a Python environment.

Notes

- To use DLI's Python SDKs to access a specific service's API, you need to ensure that the current service has been enabled and authorized on the DLI management console.

- Python 2.7.10, 3.4.0, or later versions are required. The Visual C++ compilation environment is required, and Visual C++ build tools or Visual Studio must be installed.

  For details about how to configure the Python development environment, see **Configuring the Python SDK Environment**.
- The DLI Python SDK dependencies include urllib3 1.15 or later, six 1.10 or later, certifi, and python-dateutil.
- For details about how to obtain and install Python SDKs, see **Obtaining and Installing Python SDKs**.
- To use SDKs to access DLI, you need to initialize the DLI client. During DLI client initialization, you can use the AK/SK or token for authentication. For details, see **Initializing the DLI Client**.

Python SDKs

Table 6-10 Python SDKs

SDK	Description
Queue-Related SDKs	This section provides instructions on how to use DLI's Python SDKs to retrieve a list of all queues.
Resource-Related SDKs	This section provides instructions on how to use DLI's Python SDKs to upload resource packages, query all resource packages, query specific resource packages, and delete resource packages.
SDKs Related to SQL Jobs	This section provides instructions on how to use DLI's Python SDKs for database-related operations, table-related operations, and job-related operations.
SDKs Related to Spark Jobs	This section provides instructions on how to use DLI's Python SDKs to submit Spark jobs, cancel Spark jobs, and delete Spark jobs.

6.4.2 Queue-Related SDKs

Constraints

Jobs created using the SDK cannot run on the default queue.

Querying All Queues

You can use the API provided by DLI to query the queue list and select the corresponding queue to execute the job. Sample code is as follows:

```
def list_all_queues(dli_client):
    try:
        queues = dli_client.list_queues()
    except DliException as e:
        print(e)
```

```
    return

for queue in queues:
    print(queue.name)
```

For details about the dependencies and complete sample code, see **Overview**.

6.4.3 Resource-Related SDKs

Prerequisites

- You have configured the Java SDK environment by following the instructions provided **Overview**.
- You have initialized the DLI Client by following the instructions provided in **Initializing the DLI Client**.

Uploading a Resource Package

You can use the APIs provided by DLI to upload resource packages. The following is an example. For details about the dependencies and complete sample code, see **Overview**.

```
def upload_resource(dli_client, kind, obs_jar_paths, group_name):
    try:
        dli_client.upload_resource(kind, obs_jar_paths, group_name)
    except DliException as e:
        print(e)
        return
```

NOTE

The following describes the request parameters. For details, see **Overview**.

- **kind**: resource package type. The options are as follows:
  - **jar**: JAR file
  - **Pyfile**: User Python file
  - **file**: User file
  - **modelfile**: User AI model file
- **obs_jar_paths**: OBS path of the resource package. The parameter format is {bucketName}.{obs domain name}/{jarPath}/{jarName}.

  Example: "https://bucketname.obs.com/jarname.jar"
- **group_name**: Name of the group to which the resource package belongs

Querying All Resource Packages

You can use the API provided by DLI to query the list of uploaded resources. The example code is as follows:

```
def list_resources(dli_client):
    try:
        resources = dli_client.list_resources()
    except DliException as e:
        print(e)
        return

    for resources_info in resources.package_resources:
        print('Package resource name:' + resources_info.resource_name)

    for group_resource in resources.group_resources:
        print('Group resource name:' + group_resource.group_name)
```


For details about the dependencies and complete sample code, see **Overview**.

Querying a Specified Resource Package

You can call an API to query information about the specified resource package. The sample code is as follows:

```
def get_package_resource(dli_client, resource_name, group_name):
 try:
      pkg_resource = dli_client.get_package_resource(resource_name, group_name)
      print(pkg_resource)
   except DliException as e:
      print(e)
      return
```

Deleting a Resource Package

You can call an API to delete an uploaded resource package. The sample code is as follows:

```
def delete_resource(dli_client, resource_name, group_name):
   try:
      dli_client.delete_resource(resource_name, group_name)
   except DliException as e:
      print(e)
      return
```

6.4.4 SDKs Related to SQL Jobs

6.4.4.1 Database-Related SDKs

Creating a Database

DLI provides an API for creating a database. You can use the API to create a database. The sample code is as follows:

```
def create_db(dli_client):
   try:
      db = dli_client.create_database('db_for_test')
   except DliException as e:
      print(e)
      return

   print(db)
```

NOTE

- The **default** database is a built-in database. You are not allowed to create a database named **default**.
- For details about the dependencies and complete sample code, see **Overview**.

Deleting a Database

DLI provides an API for deleting a database. The example code is as follows:

```
def delete_db(dli_client, db_name):
   try:
      dli_client.delete_database(db_name)
   except DliException as e:
      print(e)
      return
```


NOTE

- A database that contains tables cannot be deleted. To delete a database that contains tables, delete the tables first.
- A deleted database cannot be restored. Therefore, exercise caution when deleting a database.
- For details about the dependencies and complete sample code, see **Overview**.

Querying All Databases

DLI provides an API for querying the database list. The example code is as follows:

```
def list_all_dbs(dli_client):
    try:
        dbs = dli_client.list_databases()
    except DliException as e:
        print(e)
        return

    for db in dbs:
        print(db)
```

For details about the dependencies and complete sample code, see **Overview**.

6.4.4.2 Table-Related SDKs

Creating a DLI Table

DLI provides an API for creating DLI tables. Sample code is as follows:

```
def create_dli_tbl(dli_client, db_name, tbl_name):
    cols = [
        Column('col_1', 'string'),
        Column('col_2', 'string'),
        Column('col_3', 'smallint'),
        Column('col_4', 'int'),
        Column('col_5', 'bigint'),
        Column('col_6', 'double'),
        Column('col_7', 'decimal(10,0)'),
        Column('col_8', 'boolean'),
        Column('col_9', 'date'),
        Column('col_10', 'timestamp')
    ]
    sort_cols = ['col_1']
    tbl_schema = TableSchema(tbl_name, cols, sort_cols)
    try:
        table = dli_client.create_dli_table(db_name, tbl_schema)
    except DliException as e:
        print(e)
        return

    print(table)
```

For details about the dependencies and complete sample code, see **Overview**.

Creating an OBS Table

DLI provides an API for creating OBS tables. The example code is as follows:

```
def create_obs_tbl(dli_client, db_name, tbl_name):
    cols = [
        Column('col_1', 'string'),
        Column('col_2', 'string'),
```

```
    Column('col_3', 'smallint'),
    Column('col_4', 'int'),
    Column('col_5', 'bigint'),
    Column('col_6', 'double'),
    Column('col_7', 'decimal(10,0)'),
    Column('col_8', 'boolean'),
    Column('col_9', 'date'),
    Column('col_10', 'timestamp')
]
tbl_schema = TableSchema(tbl_name, cols)
try:
    table = dli_client.create_obs_table(db_name, tbl_schema,
                                        'obs://bucket/obj',
                                        'csv')
except DliException as e:
    print(e)
    return

print(table)
```

NOTE

- You need to create an OBS path in advance and specify it when creating an OBS table.
- For details about the dependencies and complete sample code, see **Overview**.

Deleting a Table

DLI provides an API for deleting tables. The example code is as follows:

```
def delete_tbls(dli_client, db_name):
    try:
        tbls = dli_client.list_tables(db_name)
        for tbl in tbls:
            dli_client.delete_table(db_name, tbl.name)
    except DliException as e:
        print(e)
        return
```

NOTE

- A deleted table cannot be restored. Exercise caution when deleting a table.
- For details about the dependencies and complete sample code, see **Overview**.

Querying All Tables

DLI provides an API for querying tables. The example code is as follows:

```
def list_all_tbls(dli_client, db_name):
    try:
        tbls = dli_client.list_tables(db_name, with_detail=True)
    except DliException as e:
        print(e)
        return

    for tbl in tbls:
        print(tbl.name)
```

For details about the dependencies and complete sample code, see **Overview**.

Describing Table Information

You can call an API to obtain the metadata description of a table. The sample code is as follows:

```
def get_table_schema(dli_client, db_name, tbl_name):
 try:
```



```
    table_info = dli_client.get_table_schema(db_name, tbl_name)
    print(table_info)
except DliException as e:
    print(e)
    return
```

6.4.4.3 Job-related SDKs

For details about the dependencies and complete sample code, see **Overview**.

Importing Data

DLI provides an API for importing data. You can use this API to import OBS to a DLI table. Sample code is as follows:

```
def import_data(dli_client, db_name, tbl_name, queue_name):
    options = {
        "with_column_header": True,
        "delimiter": ",",
        "quote_char": "\"",
        "escape_char": "\\",
        "date_format": "yyyy/MM/dd",
        "timestamp_format": "yyyy/MM/dd hh:mm:ss"
        }

    try:
        job_id, status = \
            dli_client.import_table(tbl_name, db_name,
                                    'obs://bucket/obj/data.csv',
                                    'csv',
                                    queue_name=queue_name,
                                    options=options)
    except DliException as e:
        print(e)
        return

    print(job_id)
    print(status)
```

NOTE

- Before submitting the importing job, you can specify the **data_type** parameter to set the type of the data to be imported. For example, set **data_type** to **csv**. Use the **options** parameter to set details about the CSV data format, such as the delimiter and escape character.
- If a folder and a file under an OBS bucket directory have the same name, data is preferentially loaded to the file, instead of the folder. It is recommended that the files and folders of the same level have different names when you create an OBS object.

Exporting Data

DLI provides an API for exporting data. You can use this API to export DLI table data to an OBS bucket. The example code is as follows:

```
def export_data(dli_client, db_name, tbl_name, queue_name):
    try:
        job_id, status = dli_client.export_table(tbl_name, db_name,
                                                 'obs://bucket/obj',
                                                 queue_name=queue_name)
    except DliException as e:
        print(e)
        return

    print(job_id)
    print(status)
```

NOTE

- Before submitting the export job, you can set the data format, compression type, and export mode. The data can only be exported in the CSV format.
- If a folder and a file under an OBS bucket directory have the same name, data is preferentially loaded to the file, instead of the folder. It is recommended that the files and folders of the same level have different names when you create an OBS object.

Submitting a Job

DLI provides an API for querying jobs. The example code is as follows:

```
def run_sql(dli_client, db_name, queue_name):
    # execute SQL
    try:
        sql_job = dli_client.execute_sql('select * from tbl_dli_for_test', db_name, queue_name=queue_name)
        result_set = sql_job.get_result(queue_name=queue_name)
    except DliException as e:
        print(e)
        return

    if result_set.row_count == 0:
        return

    for row in result_set:
        print(row)

    # export the query result to obs
    try:
        status = sql_job.export_result('obs://bucket/obj',
                                       queue_name=queue_name)
    except DliException as e:
        print(e)
        return

    print(status)
```

Canceling a Job

DLI provides an API for canceling jobs. You can use it to cancel a submitted job. A job that has been completed or failed cannot be canceled. The example code is as follows:

```
def cancel_sql(dli_client, job_id):
    try:
        dli_client.cancel_sql(job_id)
    except DliException as e:
        print(e)
        return
```

Querying All Jobs

DLI provides an API for querying all jobs. You can use the API to query information about all jobs in the current project and obtain the query result. The example code is as follows:

```
def list_all_sql_jobs(dli_client):
    try:
        sql_jobs = dli_client.list_sql_jobs()
    except DliException as e:
        print(e)
        return
    for sql_job in sql_jobs:
        print(sql_job)
```


NOTE

APIs in this SDK do not support SQL patterns. You cannot match SQL patterns for job query. To query DLI jobs, use the **Querying All Jobs** API.

Querying SQL Jobs

You can call an API to query information about all SQL jobs in the current project and obtain the query result. The sample code is as follows:

```
def list_sql_jobs(dli_client):
    try:
        sql_jobs = dli_client.list_sql_jobs()
    except DliException as e:
        print(e)
        return
```

6.4.5 SDKs Related to Spark Jobs

For details about the dependencies and complete sample code, see **Overview**.

Submitting Batch Jobs

DLI provides an API to perform batch jobs. The example code is as follows:

```
def submit_spark_batch_job(dli_client, batch_queue_name, batch_job_info):
    try:
        batch_job = dli_client.submit_spark_batch_job(batch_queue_name, batch_job_info)
    except DliException as e:
        print(e)
        return

    print(batch_job.job_id)
    while True:
        time.sleep(3)
        job_status = batch_job.get_job_status()
        print('Job status: {0}'.format(job_status))
        if job_status == 'dead' or job_status == 'success':
            break

    logs = batch_job.get_driver_log(500)
    for log_line in logs:
        print(log_line)
```

Canceling a Batch Processing Job

DLI provides an API for canceling batch processing jobs. If the job execution is complete or fails, you cannot cancel this job. The example code is as follows:

```
def del_spark_batch(dli_client, batch_id):
    try:
        resp = dli_client.del_spark_batch_job(batch_id)
        print(resp.msg)
    except DliException as e:
        print(e)
        return
```

Deleting Batch Processing Jobs

DLI provides an API for deleting batch processing jobs. The following sample code calls the API to delete a batch processing job:

```
def del_spark_batch(dli_client, batch_id):
    try:
```



```
    resp = dli_client.del_spark_batch_job(batch_id)
    print(resp.msg)
except DliException as e:
    print(e)
    return
```


A Change History

Released On	Description
2023-12-05	This issue is the sixth official release. Optimized the structure of *Data Lake Insight SDK Reference* and added the description of using V3 SDKs.
2023-09-16	This issue is the fifth official release. Added the description of downloading the .sha256 file corresponding to the SDK installation package to **Obtaining and Installing the Java SDK**.
2023-07-18	This issue is the fourth official release. Modified the default precision of data of the decimal type in **Table-Related SDKs**.
2023-01-30	This issue is the third official release. Added the description about the Visual C++ compilation environment required for installing Python in **Configuring the Python SDK Environment**.
2020-04-28	This issue is the second official release. Adjusted the document structure.
2020-01-28	This issue is the first official release.

