数据湖探索

SDK 参考

文档版本	01
发布日期	2025-02-11

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 DLI SDK 简介.....1
2 （推荐）DLI SDK V3.....2
3 DLI SDK.....3
3.1 DLI SDK 功能矩阵.....3
3.2 DLI SDK 与 API 的对应关系.....4
4 Java SDK.....10
4.1 Java SDK 概述.....10
4.2 Java SDK 环境配置.....11
4.2.1 Java 开发环境配置.....11
4.2.2 SDK 的获取与安装.....11
4.2.3 初始化 DLI 客户端.....17
4.3 OBS 授权.....19
4.4 队列相关.....19
4.5 资源相关.....20
4.6 SQL 作业相关.....21
4.6.1 数据库相关.....22
4.6.2 表相关.....23
4.6.3 作业相关.....25
4.7 Flink 作业相关.....30
4.8 Spark 作业相关.....33
4.9 Flink 作业模板相关.....35
5 Python SDK.....36
5.1 Python SDK 概述.....36
5.2 Python SDK 环境配置.....37
5.2.1 Python 开发环境配置.....38
5.2.2 SDK 获取与安装.....39
5.2.3 初始化 DLI 客户端.....40
5.3 队列相关.....41
5.4 资源相关.....41
5.5 SQL 作业相关.....42
5.5.1 数据库相关.....43
5.5.2 表相关.....43

5.5.3 作业相关 45
5.6 Spark 作业相关 47

1 DLI SDK 简介

DLI SDK 简介

数据湖探索服务软件开发工具包（DLI SDK，Data Lake Insight Software Development Kit）是对DLI服务提供的REST API进行的作业提交的封装，以简化用户的开发工作。用户直接调用DLI SDK提供的接口函数即可实现使用提交DLI SQL和DLI Spark作业。

DLI支持的SDK分为SDK V3和DLI服务自行开发的SDK。

- （推荐）DLI SDK V3：是根据定义API的YAML文件统一自动生成，其接口参数与服务的API一致。

  具体操作请参考**SDK V3版本开发指南**。
- DLI SDK（服务自研）：是DLI服务自行开发的SDK，本手册介绍DLI 自研SDK的使用方法。
  - Java SDK操作指导请参考**Java SDK**
  - Python SDK操作指导请参考**Python SDK**

说明

DLI SDK调用接口使用https进行访问，有服务端使用证书。

2 （推荐）DLI SDK V3

写作说明

本文介绍了DLI服务提供的V3版本的SDK，列举了最新版本SDK的获取地址。

SDK 列表

SDK列表提供了DLI云服务支持的SDK列表，您可以在GitHub仓库查看SDK更新历史、获取安装包以及查看指导文档。

SDK V3版本简介。

SDK列表提供了DLI云服务支持的SDK列表，您可以在GitHub仓库查看SDK更新历史、获取安装包以及查看指导文档。

在线生成 SDK 代码

【样例】

API Explorer能根据需要动态生成SDK代码功能，降低您使用SDK的难度，推荐使用。

您可以在API Explorer中具体API页面的“代码示例”页签查看对应编程语言类型的SDK代码。

3 DLI SDK

3.1 DLI SDK 功能矩阵

SDK开发指南指导您如何安装和配置开发环境、如何通过调用DLI SDK提供的接口函数进行二次开发。

Java、Python SDK功能矩阵请参见**表1**

表 3-1 SDK 功能矩阵

语言	功能	内容
Java	**OBS授权**	介绍将OBS桶的操作权限授权给DLI的Java SDK使用说明。
	队列相关	介绍创建队列、获取默认队列、查询所有队列、删除队列的Java SDK使用说明。
	资源相关	介绍上传资源包、查询所有资源包、查询指定资源包、删除资源包的Java SDK使用说明。
	SQL作业相关	介绍数据库相关、表相关、作业相关Java SDK使用说明。
	Flink作业相关	介绍新建Flink作业、查询作业详情、查询作业列表等Java SDK使用说明。
	Spark作业相关	介绍提交Spark作业、查询所有Spark作业、删除Spark作业等Java SDK使用说明。
	Flink作业模板相关	介绍新建Flink作业模板、更新Flink作业模板、删除Flink作业模板的JavaSDK使用说明。
Python	**队列相关**	介绍查询所有队列的Python SDK使用说明。
	资源相关	介绍上传资源包、查询所有资源包、查询制定资源包、删除资源包的Python SDK使用说明。

语言	功能	内容
	SQL作业相关	介绍数据库相关、表相关、作业相关的Python SDK使用说明。
	Spark作业相关	介绍提交Spark作业、取消Spark作业、删除Spark作业等Python SDK使用说明。

3.2 DLI SDK 与 API 的对应关系

OBS 授权

表 3-2 OBS 授权相关 API&SDK 的对应关系表

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

队列相关

表 3-3 队列相关 API&SDK 的对应关系表

Class	Method	Java Method	Python Method	API
Queue	创建队列	createQueue	-	POST /v1.0/{project_id}/queues
	删除队列	deleteQueue	-	DELETE /v1.0/{project_id}/queues/{queue_name}
	获取默认队列	getDefaultQueue	-	-
	查询所有队列	listAllQueues	list_queues	GET/v1.0/{project_id}/queues

资源相关

表 3-4 资源相关 API&SDK 的对应关系表

Class	Method	Java Method	Python Method	API
packageResources	上传资源包	uploadResources	upload_resource	POST /v2.0/{project_id}/resources
	删除资源包	deleteResource	delete_resource	DELETE /v2.0/{project_id}/resources/{resource_name}
	查询所有资源包	listAllResources	list_resources	GET /v2.0/{project_id}/resources
	查询指定资源包	getResource	get_package_resource	GET /v2.0/{project_id}/resources/{resource_name}

SQL 作业相关

表 3-5 SQL 作业相关 API&SDK 的对应关系表

Class	Method	Java Method	Python Method	API
Database	创建数据库	createDatabase	create_database	POST /v1.0/{project_id}/databases
	删除数据库	deleteDatabase	delete_database	DELETE /v1.0/{project_id}/databases/{database_name}
	查询所有数据库	listAllDatabases	list_databases	GET /v1.0/{project_id}/databases
	修改数据库用户	-	-	PUT /v1.0/{project_id}/databases/{database_name}/owner
Table	创建DLI表	createDLITable	create_dli_table	POST /v1.0/{project_id}/databases/{database_name}/tables
	创建OBS表	createObsTable	create_obs_table	POST /v1.0/{project_id}/databases/{database_name}/tables
	删除表	deleteTable	delete_table	DELETE /v1.0/{project_id}/databases/{database_name}/tables/{table_name}

Class	Method	Java Method	Python Method	API
	查询所有表	listAllTables	list_tables	GET /v1.0/{project_id}/databases/{database_name}/tables?keyword=tb&with-detail=true
	描述表信息	getTableDetail	get_table_schema	GET /v1.0/{project_id}/databases/{database_name}/tables/{table_name}
	预览表内容	-	-	GET /v1.0/{project_id}/databases/{database_name}/tables/{table_name}/preview
	修改表用户	-	-	PUT /v1.0/{project_id}/databases/{database_name}/tables/{table_name}/owner
Job	导入数据	submit	import_table	POST /v1.0/{project_id}/jobs/import-table
	导出数据	submit	export_table	POST /v1.0/{project_id}/jobs/export-table
	提交作业	submit	execute_sql	POST /v1.0/{project_id}/jobs/submit-job
	取消作业	cancelJob	-	DELETE /v1.0/{project_id}/jobs/{job_id}
	查询所有作业	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}
	查询作业结果	queryJobResultInfo	-	GET/v1.0/{project_id}/jobs/{job_id}?page-size=*{size}*&current-page=*{page_number}*
	查询作业状态	-	-	GET/v1.0/{project_id}/jobs/{job_id}/status
	查询作业详细信息	-	-	GET/v1.0/{project_id}/jobs/{job_id}/detail

Class	Method	Java Method	Python Method	API
	查询SQL类型作业	listSQLJobs	-	-
	检查SQL语法	-	-	POST /v1.0/{project_id}/jobs/check-sql
	导出查询结果	-	-	POST /v1.0/{project_id}/jobs/{job_id}/export-result

Flink 作业相关

表 3-6 Flink 作业相关 API&SDK 的对应关系表

Class	Method	Java Method	Python Method	API
Job	创建Flink SQL作业	submitFlinkSqUob	-	POST /v1.0/{project_id}/streaming/sql-jobs
	创建Flink自定义作业	createFlinkJarJob	-	POST /v1.0/{project_id}/streaming/flink-jobs
	更新Flink SQL作业	updateFlinkSqUob	-	PUT /v1.0/{project_id}/streaming/sql-jobs/{job_id}
	更新Flink自定义作业	updateFlinkJarJob	-	PUT /v1.0/{project_id}/streaming/flink-jobs/{job_id}
	查询Flink作业列表	getFlinkJobs	-	GET /v1.0/{project_id}/streaming/jobs
	查询Flink作业详情	getFlinkJobDetail	-	GET /v1.0/{project_id}/streaming/jobs/{job_id}
	查询Flink作业执行计划图	getFlinkJobExecuteGraph	-	GET /v1.0/{project_id}/streaming/jobs/{job_id}/execute-graph
	查询Flink作业监控信息	getFlinkJobsMetrics	-	POST /v1.0/{project_id}/streaming/jobs/metrics
	查询Flink作业APIG网关服务访问地址	getFlinkApigSinks	-	GET /v1.0/{project_id}/streaming/jobs/{job_id}/apig-sinks
	运行Flink作业	runFlinkJob	-	POST /v1.0/{project_id}/streaming/jobs/run

Class	Method	Java Method	Python Method	API
	停止Flink作业	stopFlinkJob	-	POST /v1.0/{project_id}/streaming/jobs/stop
	批量删除Flink作业	deleteFlinkJobInBatch	-	POST /v1.0/{project_id}/streaming/jobs/delete

Spark 作业相关

表 3-7 Spark 作业相关 API&SDK 的对应关系表

Class	Method	Java Method	Python Method	API
BatchJob	提交批处理作业	asyncSubmit	submit_spark_batch_job	POST /v2.0/{project_id}/batches
	删除批处理作业	deleteBatchJob	del_spark_batch_job	DELETE /v2.0/{project_id}/batches/{batch_id}
	查询所有批处理作业	listAllBatchJobs	-	GET /v2.0/{project_id}/batches
	查询批处理作业详情	-	-	GET /v2.0/{project_id}/batches/{batch_id}
	查询批处理作业状态	getStateBatchJob	-	GET /v2.0/{project_id}/batches/{batch_id}/state
	查询批处理作业日志	getBatchJobLog	-	GET /v2.0/{project_id}/batches/{batch_id}/log

Flink 作业模板相关

表 3-8 Flink 作业模板相关 API&SDK 的对应关系表

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}
	deleteFlinkJobTemplate	-	DELETE /v1.0/{project_id}/streaming/job-templates/{template_id}

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

4 Java SDK

4.1 Java SDK 概述

操作场景

DLI Java SDK 让您无需关心请求细节即可快速使用数据湖探索服务。本节操作介绍如何获取并使用Java SDK 。

使用须知

- 要使用DLI Java SDK 访问指定服务的 API ，您需要确认已在DLI控制台开通当前服务并完成服务授权。
- Java SDK 支持 Java JDK 1.8 及其以上版本。关于Java开发环境的配置请参考**Java SDK环境配置**。
- 关于Java SDK的获取与安装请参考**SDK的获取与安装**。
- 使用SDK工具访问DLI，需要用户初始化DLI客户端。用户可以使用AK/SK(Access Key ID/Secret Access Key)或Token两种认证方式初始化客户端，具体操作请参考**初始化DLI客户端**

Java SDK 列表

表 4-1 Java SDK 列表

类型	说明
OBS授权	介绍将OBS桶的操作权限授权给DLI的Java SDK使用说明。
队列相关	介绍创建队列、获取默认队列、查询所有队列、删除队列的Java SDK使用说明。
资源相关	介绍上传资源包、查询所有资源包、查询指定资源包、删除资源包的Java SDK使用说明。
SQL作业相关	介绍数据库相关、表相关、作业相关Java SDK使用说明。

类型	说明
Flink作业相关	介绍新建Flink作业、查询作业详情、查询作业列表等Java SDK使用说明。
Spark作业相关	介绍提交Spark作业、查询所有Spark作业、删除Spark作业等Java SDK使用说明。
Flink作业模板相关	介绍新建Flink作业模板、更新Flink作业模板、删除Flink作业模板的JavaSDK使用说明。

4.2 Java SDK 环境配置

4.2.1 Java 开发环境配置

操作场景

在安装和使用Java SDK前，确保您已经完成开发环境的基本配置。

Java SDK要求使用JDK1.8或更高版本。考虑到后续版本的兼容性，推荐使用1.8版本。

在Java运行环境配置好的情况下，打开windows的命令行，执行命令**Java -version**，可以检查版本信息。

操作步骤

1. 安装JDK。从**Oracle官网**下载并安装JDK1.8版本安装包。
2. 配置环境变量，在“控制面板”选择“系统”属性，单击“环境变量”。
3. 选择“系统变量”，新建 “JAVA_HOME 变量”，路径配置为JDK安装路径，例如：“D:\Java\jdk1.8.0_45”。
4. 编辑 “Path 变量”，在“变量值”中增加“%JAVA_HOME%\bin;”。
5. 新建 “CLASSPATH 变量”，在“变量值”中填写 “.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar”。
6. 检验是否配置成功，运行cmd ，输入 ***java -version***。运行结果，请参见**图4-1**，显示版本信息，则说明安装和配置成功。

图 4-1 检验配置是否成功

```
C:\Users\gwx419194>java -version
java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b15)
Java HotSpot(TM) Client VM (build 25.45-b02, mixed mode, sharing)
```

4.2.2 SDK 的获取与安装

Java SDK 安装方式

您可以通过以下两种方式安装Java SDK。

- 导入Maven依赖，适用于使用Maven管理的Java项目。具体操作请参考**方法一：通过Maven安装DLI SDK依赖**。
- 在集成开发环境中导入JAR文件，适用于使用Eclipse作为集成开发环境的项目。具体操作请参考**方法二：通过在Eclipse中导入JAR文件安装SDK**。

获取 DLI SDK

1. 登录DLI管理控制台。
2. 单击总览页右侧“常用链接”中的“**SDK下载**”。
3. 在“DLI SDK DOWNLOAD”页面，选择相应驱动下载。
   “dli-sdk-java-*x.x.x*.zip”压缩包，解压后目录结构如下：

表 4-2 目录结构

名称	说明
jars	SDK及其依赖的jar包。
maven-install	安装至本地Maven仓库的脚本及对应jar包。
dli-sdk-java.version	Java SDK版本说明。

方法一：通过 Maven 安装 DLI SDK 依赖

推荐您通过Maven安装依赖的方式使用华为云 Java SDK：

- **安装服务级SDK依赖**
  a. 首先您需要在您的操作系统中 **下载** 并 **安装**Maven 。
  b. 安装配置完成后，输入命令“mvn -v”,显示如下图 Maven版本信息 表示成功。

```
D:\>mvn -v
D:\
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-11T00:41:47+08:00)
Maven home: D:\maven\apache-maven-3.3.9\bin\..
Java version: 1.8.0_262, vendor: Huawei Technologies Co., Ltd
Java home: D:\develop\jdk_1.8\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "dos"
```

  c. Maven安装完成后，可根据开发需要，直接引入依赖到已有的Maven工程或先用开发工具创建Maven工程。
  创建项目以idea开发工具为例(已有Maven 工程可跳过此步骤)：
     i. 打开IntelliJ IDEA 开发工具。
     ii. 点击File - New - project...
     iii. 在New Project弹窗点击-Maven-点击Next。
     iv. 输入GroupId和ArtifactId，点击Next。
     v. 输入Project name 和 Project location,点击Finish。
  d. 在Maven 项目的 pom.xml文件加入相应的依赖项即可。
  以引入最新版本SDK为例，请在获取最新的sdk包版本，替换代码中版本。

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

```
  <version>x.x.x</version>
</dependency>
```

- **安装其他服务SDK依赖。**

  DLI依赖SDK（例如，OBS SDK），可以通过配置华为云的maven镜像源仓库下载。

  - **（推荐）以华为镜像源作为主仓库：**

    配置华为maven镜像源的具体方法可参见：华为开源镜像站>选择“华为SDK”>单击“HuaweiCloud SDK”。

    使用maven构建时，settings.xml文件需要修改，增加以下内容：

    i. 在profiles节点中添加如下内容：

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

    ii. 在mirrors节点中增加：

    ```
    <mirror>
       <id>huaweicloud</id>
       <mirrorOf>*,!HuaweiCloudSDK</mirrorOf>
       <url>https://repo.huaweicloud.com/repository/maven/</url>
    </mirror>
    ```

    iii. 增加activeProfiles标签激活配置：

    ```
    <activeProfiles>
     <activeProfile>MyProfile</activeProfile>
    </activeProfiles>
    ```

  - **以非华为镜像源作为主仓库（例如用户自定义镜像源）使用“HuaweiCloud SDK”：**

    使用maven构建时，settings.xml文件需要修改为如下内容：

    ```
    <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://
    maven.apache.org/xsd/settings-1.0.0.xsd">
       <servers>
        <server>
         <id>demo-releases</id>
         <username>deployment</username>
         <password><![CDATA[xxx]]></password>
       </server>
     </servers>
    ```

```
<mirrors>
  <mirror>
    <id>demo-releases</id>
    <mirrorOf>*,!HuaweiCloudSDK</mirrorOf>
    <url>http://maven.demo.com:8082/demo/content/groups/public</url>
  </mirror>
</mirrors>
<profiles>
  <profile>
    <id>demo</id>
    <activation>
      <activeByDefault>true</activeByDefault>
      <jdk>1.8</jdk>
    </activation>
    <properties>
      <maven.compiler.source>1.8</maven.compiler.source>
      <maven.compiler.target>1.8</maven.compiler.target>
      <maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>
    </properties>
    <repositories>
       <repository>
        <id>demo-releases</id>
        <url>http://demo-releases</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>true</enabled>
        </snapshots>
      </repository>
    </repositories>
    <pluginRepositories>
      <pluginRepository>
        <id>demo-releases</id>
        <url>http://demo-releases</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>true</enabled>
        </snapshots>
      </pluginRepository>
    </pluginRepositories>
  </profile>
  <profile>
    <id>huaweicloudrepo</id>
    <repositories>
      <repository>
        <id>HuaweiCloudSDK</id>
        <url>https://repo.huaweicloud.com/repository/maven/huaweicloudsdk/</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </repository>
    </repositories>
    <pluginRepositories>
      <pluginRepository>
        <id>HuaweiCloudSDK</id>
        <url>https://repo.huaweicloud.com/repository/maven/huaweicloudsdk/</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
```

```
    </pluginRepositories>
  </profile>
 </profiles>
 <activeProfiles>
  <activeProfile>demo</activeProfile>
  <activeProfile>huaweicloudrepo</activeProfile>
 </activeProfiles>
</settings>
```

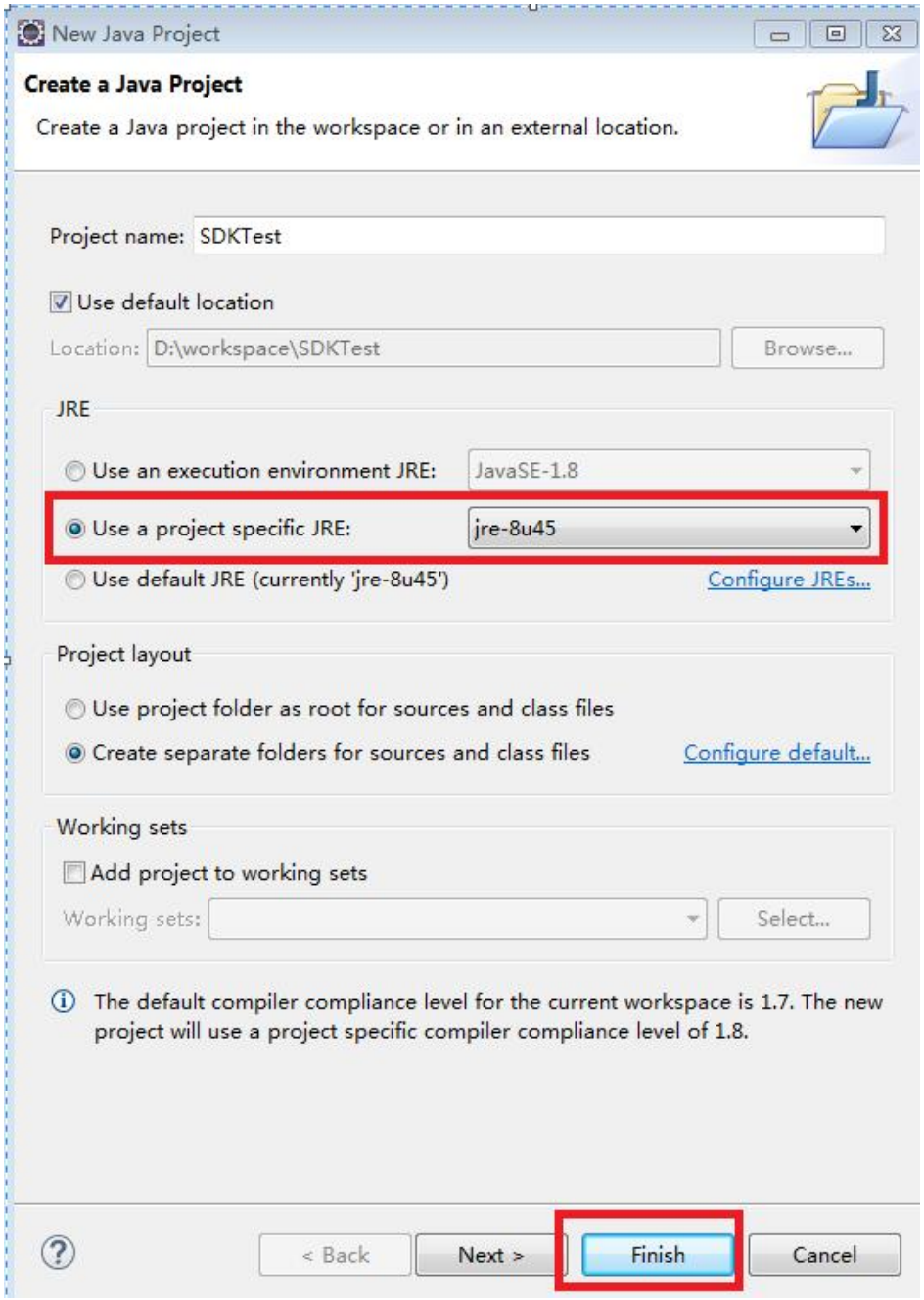
方法二：通过在 Eclipse 中导入 JAR 文件安装 SDK

使用Eclipse集成开发环境的项目，在集成开发环境中导入JAR文件。

步骤1 从**Eclipse官网**下载并安装Eclipse IDE for Java Developers最新版本。在Eclipse中配置好JDK。

1. 创建新工程，选择JRE版本，请参见**图4-2**

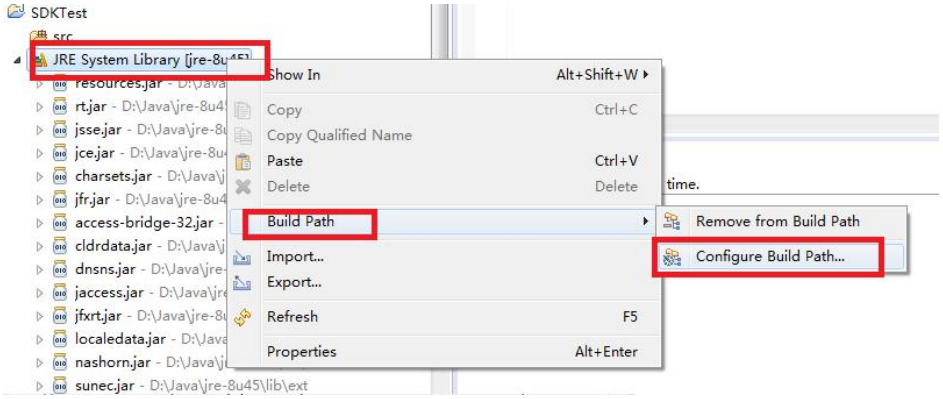
图 4-2 创建新工程



步骤2 配置并导入SDKjar包。

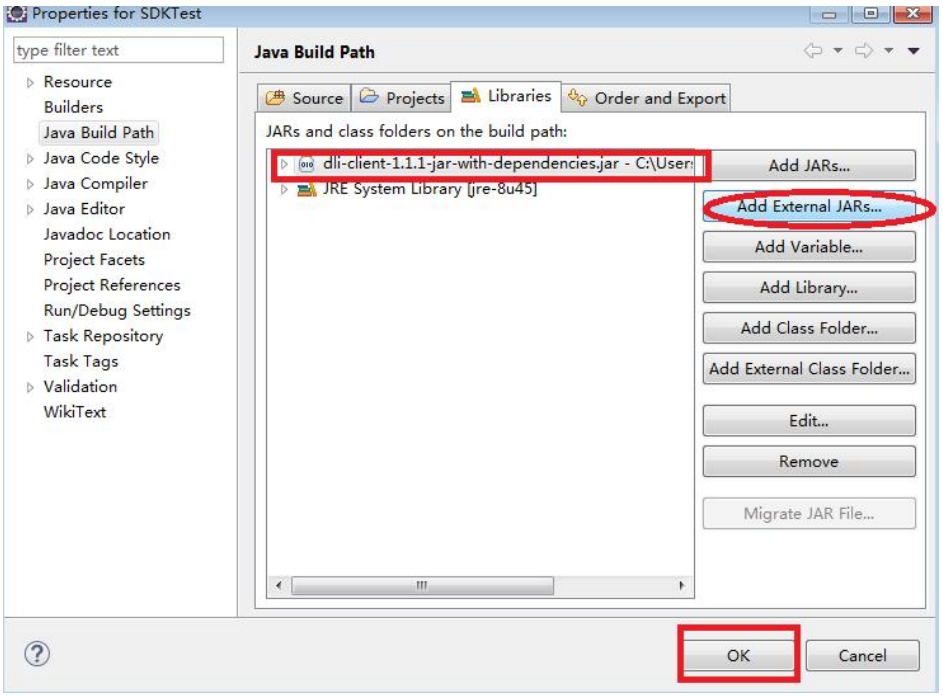
1. 在工程“JRE System Library”上单击右键，选择“Build Path”>“Configure Build Path”，请参见图4-3。

图 4-3 配置工程路径



2. 单击“Add External JARs”，选择**SDK下载**的jar包，单击OK。

图 4-4 选择 SDK jar 包



----结束

4.2.3 初始化 DLI 客户端

使用DLI SDK工具访问DLI，需要用户初始化DLI客户端。用户可以使用AK/SK(Access Key ID/Secret Access Key)或Token两种认证方式初始化客户端，示例代码如下：

前提条件

- 已参考**Java SDK概述**配置Java SDK环境。
- 已参考**初始化DLI客户端**完成客户端DLIClient的初始化。

AK/SK 认证方式样例代码

- **代码样例**

```
String ak = System.getenv("xxx_SDK_AK");//访问密钥ID。
String sk = System.getenv("xxx_SDK_SK");//与访问密钥ID结合使用的密钥。
String regionName = "regionname";
String projectId = "project_id";
DLIInfo dliInfo = new DLIInfo(regionName, ak, sk, projectId);
DLIClient client = new DLIClient(AuthenticationMode.AKSK, dliInfo);
```

- **参数说明及获取方式**
  - 参数说明
    - ak：账号 Access Key
    - sk：账号 Secret Access Key

      说明

      认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放, 使用时解密, 确保安全。

      本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量xxx_SDK_AK和xxx_SDK_SK。

    - regionName ： 所属区域名称
    - projectId ： 项目ID
  - 通过以下方式可获取AK/SK，项目ID及对应的region信息。
    i. 登录管理控制台。
    ii. 鼠标指向界面右上角的登录用户名，在下拉列表中单击“我的凭证”。
    iii. 在左侧导航栏中选择“访问密钥”，单击“新增访问密钥”。根据提示输入对应信息，单击“确定”。
    iv. 在弹出的提示页面单击“立即下载”。下载成功后，打开凭证文件，获取AK/SK信息。
    v. 左侧导航栏单击“API凭证”，在“项目列表”中获取“项目ID”即为project_id值，对应的“项目”即为region的值。

Token 认证方式样例代码

- 代码样例

```
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);
```

- 参数说明
  - domainname：帐号名。
  - username：用户名
  - password：用户名密码
  - regionname：所属区域名称
  - project_id：项目ID

说明

- 认证用的password硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
- 可以通过set方式修改endpoint，即dliInfo.setServerEndpoint(endpoint)。

4.3 OBS 授权

前提条件

- 已参考**Java SDK概述**配置Java SDK环境。
- 已参考**初始化DLI客户端**完成客户端DLIClient的初始化。

样例代码

用户可以使用OBS授权操作的接口，将OBS桶的操作权限授权给DLI, 用于保存用户作业的数据和作业的运行日志等。

示例代码如下：

```
private static void authorizeBucket(DLIClient client) throws DLIException {
        String bucketName ="obs_name";
        ObsBuckets obsBuckets = new ObsBuckets();
        obsBuckets.addObsBucketsItem(bucketName);
        GlobalResponse res = client.authorizeBucket(obsBuckets);
        System.out.println(res);
}
```

4.4 队列相关

前提条件

- 已参考**Java SDK概述**配置Java SDK环境。
- 已参考**初始化DLI客户端**完成客户端DLIClient的初始化。

创建队列

DLI提供创建队列的接口，您可以使用该接口创建队列。示例代码如下：

```
 private static void createQueue(DLIClient client) throws DLIException {
   //通过调用DLIClient对象的createQueue方法创建队列
   String qName = "queueName";
   int cu = 16;
   String description = "test for sdk";
   Queue queue = client.createQueue(qName, cu, mode, description);
   System.out.println("---------- createQueue success ---------");
 }
```

删除队列

DLI提供删除队列的接口，您可以使用该接口删除队列。示例代码如下：

```
 private static void deleteQueue(DLIClient client) throws DLIException {
   //调用DLIClient对象的getQueue("queueName")方法获取queueName这个队列
   String qName = "queueName";
   Queue queue = client.getQueue(qName);
   //使用deleteQueue()方法删除queueName队列
```

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

获取默认队列

DLI提供查询默认队列的接口，您可以使用默认队列提交作业。示例代码如下：

```
private static void getDefaultQueue(DLIClient client) throws DLIException{
    //调用DLIClient对象的getDefaultQueue方法查询默认队列
     Queue queue = client.getDefaultQueue();
     System.out.println("defaultQueue is:"+ queue.getQueueName());
}
```

说明

默认队列允许所有用户使用，DLI会限制用户使用默认队列的次数。

查询所有队列

DLI提供查询队列列表接口，您可以使用该接口并选择相应的队列来执行作业。示例代码如下：

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

  //通过调用DLIClient对象的listAllQueues方法查询队列列表
  List<Queue> queues = client.listAllQueues();
  for (Queue queue : queues) {
    System.out.println("Queue name:" + queue.getQueueName() + "  " + "cu:" + queue.getCuCount());
  }
}
```

4.5 资源相关

前提条件

- 已参考**Java SDK概述**配置Java SDK环境。
- 已参考**初始化DLI客户端**完成客户端DLIClient的初始化。

上传资源包

您可以使用DLI提供的接口上传资源包，示例代码如下：

```
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";
  // 调用DLIClient对象的uploadResources方法上传资源
  List<PackageResource> packageResources = client.uploadResources(kind, paths, description);
  System.out.println("---------- uploadResources success ---------");
}
```

说明

请求参数说明如下，详细参数使用可以参考**Python SDK概述**下载样例代码。

- kind：资源包类型，当前支持包类型分别为：
  - **jar**：用户jar文件
  - **pyfile**：用户Python文件
  - **file**：用户文件
  - **modelfile**：用户AI模型文件
- paths：对应资源包的OBS路径，参数构成为：{bucketName}.{obs域名}/{jarPath}/{jarName}。
- description：资源包描述信息。

查询所有资源包

DLI提供查询资源列表接口，您可以使用该接口并选择相应的资源来执行作业。示例代码如下：

```
private static void listAllResources(DLIClient client) throws DLIException {
  System.out.println("list all resources...");
  // 通过调用DLIClient对象的listAllResources方法查询队列资源列表
  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());
  }
}
```

查询指定资源包

您可以使用该接口查询指定的资源包信息，示例代码如下：

```
private static void getResource(DLIClient client) throws DLIException {
  String resourceName = "xxxxx";
  //group:资源包不在分组内，可不传入该参数
  String group= "xxxxxx";
  // 调用DLIClient对象的getResource方法查询指定资源包
  PackageResource packageResource=client.getResource(resourceName,group);
  System.out.println(packageResource);
}
```

删除资源包

您可以使用该接口删除已上传的资源包，示例代码如下：

```
private static void deleteResource(DLIClient client) throws DLIException {
  String resourceName = "xxxxx";
  //group:资源包不在分组内，可不传入该参数
  String group= "xxxxxx";
  // 调用DLIClient对象的deleteResource方法删除资源
  client.deleteResource(resourceName,group);
  System.out.println("---------- deleteResource success ---------");
}
```

4.6 SQL 作业相关

4.6.1 数据库相关

前提条件

- 已参考**Java SDK概述**配置Java SDK环境。
- 已参考**初始化DLI客户端**完成客户端DLIClient的初始化，参考**队列相关**完成队列创建等操作。

创建数据库

DLI提供创建数据库的接口。您可以使用该接口创建数据库，示例代码如下：

```
private static Database createDatabase(DLIClient client) throws DLIException {
  //通过调用DLIClient对象的createDatabase方法创建数据库
  String dbName = "databasename";
  Database database = client.createDatabase(dbName);
  System.out.println("create database:" + database);
  return database;
}
```

说明

“default”为内置数据库，不能创建名为“default”的数据库。

删除数据库

DLI提供删除数据库的接口。您可以使用该接口删除数据库。示例代码如下：

```
//调用Database对象的deleteDatabase接口删除数据库,
//其中Database对象通过调用对象DLIClient的getDatabase(String databaseName)接口获得.
private static void deletedatabase(Database database) throws DLIException {
   String dbName = "databasename";
   database=client.getDatabase(dbName);
   database.deleteDatabase();
   System.out.println("delete db " + dbName);
}
```

说明

- 含表的数据库不能直接删除，请先删除数据库的表再删除数据库。
- 数据库删除后，将不可恢复，请谨慎操作。

查询所有数据库

DLI提供查询数据库列表接口，您可以使用该接口查询当前已创建的数据库列表。示例代码如下：

```
private static void listDatabases(DLIClient client) throws DLIException {
  //通过调用DLIClient的listAllDatabases方法查询数据库列表
  List<Database> databases = client.listAllDatabases();
  for (Database db : databases) {
    System.out.println("dbName:" + db.getDatabaseName() + " " + "tableCount:" + db.getTableCount());
  }
}
```

4.6.2 表相关

创建 DLI 表

DLI提供创建DLI表的接口。您可以使用该接口创建数据存储在DLI内部的表。示例代码如下：

```
private static Table createDLITable(Database database) throws DLIException {
  //构造表列集合，通过实例化Column对象构建列
  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";
  //通过调用Database对象的createDLITable方法创建DLI表
  Table table = database.createDLITable(DLITblName, desc, columns, sortColumns);
  System.out.println(table);
  return table;
}
```

说明

DataType.DECIMAL的默认精度为(10,0)，设置Decimal类型精度的方法如下：

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

创建 OBS 表

DLI提供创建OBS表的接口。您可以使用该接口创建数据存储在OBS的表。示例代码如下：

```
private static Table createObsTable(Database database) throws DLIException {
  //构造表列集合，通过实例化Column对象构建列
  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";
    //通过调用Database对象的createObsTable方法创建OBS表
    Table table = database.createObsTable(obsTblName, desc, columns,StorageType.CSV , dataPath,
formatInfo);
    System.out.println(table);
    return table;
}
```

说明

DataType.DECIMAL的默认精度为(10,0)，设置Decimal类型精度的方法如下：

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

删除表

DLI提供删除表的接口。您可以使用该接口删除数据库下的所有表。示例代码如下：

```
private static void deleteTables(Database database) throws DLIException {
    //调用Database对象的listAllTables接口查询所有表
    List<Table> tables = database.listAllTables();
    for (Table table : tables) {
      //遍历表，调用Table对象的deleteTable接口删除表
      table.deleteTable();
      System.out.println("delete table " + table.getTableName());
    }
  }
```

说明

表删除后，将不可恢复，请谨慎操作。

查询所有表

DLI提供创建查询表的接口。您可以使用该接口查询数据库下的所有表。示例代码如下：

```
  private static void listTables(Database database) throws DLIException {
    //调用Database对象的listAllTables方法查询数据库下的所有表
    List<Table> tables = database.listAllTables(true);
    for (Table table : tables) {
      System.out.println(table);
    }
  }
```

查询表的分区信息（包含分区的创建和修改时间）

DLI提供查询表分区信息的接口。您可以使用该接口查询数据库下表的分区信息（包括分区的创建和修改时间）。示例代码如下：

```
private static void showPartitionsInfo(DLIClient client) throws DLIException {
    String databaseName = "databasename";
    String tableName = "tablename";
    //调用DLIClient对象的showPartitions方法查询数据库下表的分区信息（包括分区的创建和修改时间）
    PartitionResult partitionResult = client.showPartitions(databaseName, tableName);
    PartitionListInfo partitonInfos = partitionResult.getPartitions();
    //获取分区的创建和修改时间
    Long createTime = partitonInfos.getPartitionInfos().get(0).getCreateTime().longValue();
    Long lastAccessTime = partitonInfos.getPartitionInfos().get(0).getLastAccessTime().longValue();
    System.out.println("createTime:"+createTime+"\nlastAccessTime:"+lastAccessTime);
}
```

描述表信息

您可以使用该接口获取表的元数据描述信息。示例代码如下：

```
private static void getTableDetail(Table table) throws DLIException {
  // 调用Table对象的getTableDetail方法获取描述表信息
  //  TableSchema tableSchema=table.getTableDetail();
  //输出信息
  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());
}
```

4.6.3 作业相关

导入数据

DLI提供导入数据的接口。您可以使用该接口将存储在OBS中的数据导入到已创建的DLI表或者OBS表中。示例代码如下：

```
//实例化importJob对象，构造函数的入参包括队列、数据库名、表名（通过实例化Table对象获取）和数据路径
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());
    //调用ImportJob对象的submit接口提交导入作业
    importJob.submit(); //调用ImportJob对象的getStatus接口查询导入作业状态
    JobStatus status = importJob.getStatus();
    System.out.println("Job id: " + importJob.getJobId() + ", Status : " + status.getName());
}
```

说明

- 在提交导入作业前，可选择设置导入数据的格式，如样例所示，调用ImportJob对象的setStorageType接口设置数据存储类型为csv，数据的具体格式通过调用ImportJob对象的setCsvFormatInfo接口进行设置。
- 在提交导入作业前，可选择设置导入数据的分区并配置是否是overwrite写入，分区信息可以调用ImportJob对象的setPartitionSpec接口设置，如：importJob.setPartitionSpec(new PartitionSpec("part1=value1,part2=value2"))，也可以在创建ImportJob对象的时候直接通过参数的形式创建 。导入作业默认是追加写，如果需要覆盖写，则可以调用ImportJob对象的setOverWrite接口设置，如：importJob.setOverWrite(Boolean.TRUE)。
- 当OBS桶目录下有文件夹和文件同名时，加载数据会优先指向该路径下的文件而非文件夹。建议创建OBS对象时，在同一级中不要出现同名的文件和文件夹。

导入分区数据

DLI提供导入数据的接口。您可以使用该接口将存储在OBS中的数据导入到已创建的DLI表或者OBS表指定分区中。示例代码如下：

```
//实例化importJob对象，构造函数的入参包括队列、数据库名、表名（通过实例化Table对象获取）和数据路径
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());
    //调用ImportJob对象的submit接口提交导入作业
    importJob.submit(); //调用ImportJob对象的getStatus接口查询导入作业状态
    JobStatus status = importJob.getStatus();
    System.out.println("Job id: " + importJob.getJobId() + ", Status : " + status.getName());
}
```

说明

- 在创建ImportJob对象的时候分区信息PartitionSpec也可以直接传入分区字符串。
- partitionSpec如果导入时指定部分列为分区列，而导入的数据只包含了指定的分区信息，则数据导入后的未指定的分区列字段会存在null值等异常值。
- 示例中isOverWrite表示是否是覆盖写，为true表示覆盖写，为false表示追加写。目前不支持overwrite覆盖写整表，只支持overwrite写指定分区。如果需要追加写指定分区，则在创建ImportJob的时候指定isOverWrite为false。

导出数据

DLI提供导出数据的接口。您可以使用该接口将DLI表中的数据导出到OBS中。示例代码如下：

```
//实例化ExportJob对象，传入导出数据所需的队列、数据库名、表名（通过实例化Table对象获取）和导出数据的存储路径，仅支持Table类型为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...");
    //调用ExportJob对象的submit接口提交导出作业
    exportJob.submit();
    //调用ExportJob对象的getStatus接口查询导出作业状态
    JobStatus status = exportJob.getStatus();
    System.out.println("Job id: " + exportJob.getJobId() + ", Status : " + status.getName());
}
```

说明

- 在提交导出作业前，可选设置数据格式，压缩类型，导出模式等，如样例所示，分别调用ExportJob对象的setStorageType、setCompressType、setExportMode接口设置，其中setStorageType仅支持csv格式。
- 当OBS桶目录下有文件夹和文件同名时，加载数据会优先指向该路径下的文件而非文件夹。建议创建OBS对象时，在同一级中不要出现同名的文件和文件夹。

提交作业

DLI提供提交作业和查询作业的接口。您可以通过提交接口提交作业，如果需要查询结果可以调用查询接口查询该作业的结果。示例代码如下：

```
//实例化SQLJob对象，传入执行SQL所需的queue,数据库名，SQL语句
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...");
    //调用SQLJob对象的submit接口提交查询作业
    sqlJob.submit();
    //调用SQLJob对象的getStatus接口查询作业状态
    JobStatus status = sqlJob.getStatus();
    System.out.println(status);
    System.out.println("start export Result...");
    //调用SQLJob对象的exportResult接口导出查询结果，其中queryResultPath为导出数据的路径
    sqlJob.exportResult(queryResultPath, StorageType.CSV,
            CompressType.GZIP, ExportMode.ERRORIFEXISTS, null);
    System.out.println("Job id: " + sqlJob.getJobId() + ", Status : " + status.getName());
}
```

取消作业

DLI提供取消作业的接口。您可以使用该接口取消所有Launching或Running状态的Job，以取消Launching状态的Job为例，示例代码如下：

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

  List<JobResultInfo> jobResultInfos = client.listAllJobs(JobType.QUERY);
  for (JobResultInfo jobResultInfo : jobResultInfos) {
      //如果Job为“LAUNCHING”状态，则取消
      if (JobStatus.LAUNCHING.equals(jobResultInfo.getJobStatus())) {
          //通过JobId参数取消Job
          client.cancelJob(jobResultInfo.getJobId());
      }
  }
}
```

查询所有作业

DLI提供查询作业的接口。您可以使用该接口查询当前工程下的所有作业信息。示例代码如下：

```
private static void listAllSqlJobs(DLIClient client) throws DLIException {
    //返回JobResultInfo List集合
    List < JobResultInfo > jobResultInfos = client.listAllJobs();
    //遍历List集合查看Job信息
    for (JobResultInfo jobResultInfo: jobResultInfos) {
        //job id
        System.out.println(jobResultInfo.getJobId());
        //job 描述信息
        System.out.println(jobResultInfo.getDetail());
        //job 状态
        System.out.println(jobResultInfo.getJobStatus());
        //job 类型
        System.out.println(jobResultInfo.getJobType());
    }
    //通过JobType过滤
    List < JobResultInfo > jobResultInfos1 = client.listAllJobs(JobType.DDL);
    //通过起始时间和JobType过滤,起始时间的格式为unix时间戳
    List < JobResultInfo > jobResultInfos2 = client.listAllJobs(1502349803729L, 1502349821460L,
JobType.DDL);
    //通过分页过滤
    List < JobResultInfo > jobResultInfos3 = client.listAllJobs(100, 1, JobType.DDL);
    //分页，起始时间，Job类型
    List < JobResultInfo > jobResultInfos4 = client.listAllJobs(100, 1, 1502349803729L, 1502349821460L,
JobType.DDL);

    //通过Tags过滤查询满足条件的所有作业列表
    JobFilter jobFilter = new JobFilter();
    jobFilter.setTags("workspace=space002,jobName=name002");
    List < JobResultInfo > jobResultInfos1 = client.listAllJobs(jobFilter);
    //通过Tags过滤查询满足条件的指定page的作业列表
    JobFilter jobFilter = new JobFilter();
    jobFilter.setTags("workspace=space002,jobName=name002");
    jobFilter.setPageSize(100);
    jobFilter.setCurrentPage(0);
    List < JobResultInfo > jobResultInfos1 = client.listJobsByPage(jobFilter);
}
```

说明

- 重载方法的参数，可以设置为“null”，表示不设置过滤条件。同时也要注意参数的合法性，例如分页参数设置为“-1”，会导致查询失败。
- 该SDK接口不支持sql_pattern，即通过指定sql片段作为作业过滤条件进行查询。
  如果需要则可以通过**查询所有作业**API接口指定该参数进行查询。

查询作业结果

DLI提供查询作业结果的接口。您可以使用该接口通过JobId查询该作业信息。示例代码如下：

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

查询 SQL 类型作业

DLI提供查询SQL类型作业的接口。您可以使用该接口查询当前工程下，在编辑框中提交的最近执行的作业的信息(即可用SQL语句提交的Job)。示例代码如下：

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

 //返回JobResultInfo List集合
  List<JobResultInfo> jobResultInfos = client.listSQLJobs();
  //遍历集合查询job信息
  for (JobResultInfo jobResultInfo : jobResultInfos) {
        //job id
        System.out.println(jobResultInfo.getJobId());
        //job 描述信息
        System.out.println(jobResultInfo.getDetail());
        //job 状态
        System.out.println(jobResultInfo.getJobStatus());
        //job 类型
        System.out.println(jobResultInfo.getJobType());
   }

    //通过Tags过滤查询满足条件的所有SQL作业列表
    JobFilter jobFilter = new JobFilter();
    jobFilter.setTags("workspace=space002,jobName=name002");
    List < JobResultInfo > jobResultInfos1 = client.listAllSQLJobs(jobFilter);
    //通过Tags过滤查询满足条件的指定page的SQL作业列表
    JobFilter jobFilter = new JobFilter();
    jobFilter.setTags("workspace=space002,jobName=name002");
    jobFilter.setPageSize(100);
    jobFilter.setCurrentPage(0);
    List < JobResultInfo > jobResultInfos1 = client.listSQLJobsByPage(jobFilter);
}
```

导出查询结果

DLI提供导出查询结果的接口。您可以使用该接口导出当前工程下，在编辑框中提交的查询作业的结果。示例代码如下：

```
//实例化SQLJob对象，传入执行SQL所需的queue,数据库名，SQL语句
 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...");
    //调用SQLJob对象的submit接口提交查询作业
     sqlJob.submit();
     //调用SQLJob对象的getStatus接口查询作业状态
     JobStatus status = sqlJob.getStatus();
     System.out.println(status);
     System.out.println("start export Result...");
     //调用SQLJob对象的exportResult接口导出查询结果，其中exportPath为导出数据的路径,JSON为导出格式，
queueName为执行导出作业的队列，limitNum为导出作业结果条数，0表示全部导出
    sqlJob.exportResult(exportPath + "result", StorageType.JSON, CompressType.NONE,
        ExportMode.ERRORIFEXISTS, queueName, true, 5);
 }
```

预览作业结果

DLI提供预览作业结果的接口。您可以使用该接口获取结果集的前1000条记录。

```
//实例化SQLJob对象，传入执行SQL所需的queue,数据库名和SQL语句
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...");
    //调用SQLJob对象的submit接口提交查询作业
    sqlJob.submit();
    //调用SQLJob对象的previewJobResult接口查询结果集的前1000条记录
    List<Row> rows = sqlJob.previewJobResult();
```

```
    if (rows.size() > 0) {
        Integer value = rows.get(0).getInt(0);
        System.out.println("获取第一行结果中的第一列数据值" + value);
    }
    System.out.println("Job id: " + sqlJob.getJobId() + ", previewJobResultSize : " + rows.size());
}
```

废弃的接口

getJobResult接口当前已废弃，如果需要getJobResult类似功能可以通过调用DownloadJob接口获取。

DownloadJob接口详情可以在“dli-sdk-java-x.x.x.zip”压缩包中获取。“dli-sdk-java-x.x.x.zip”压缩包可以参考**SDK的获取与安装**中的操作步骤获取。

4.7 Flink 作业相关

前提条件

- 已参考**Java SDK概述**配置Java SDK环境。
- 已参考**初始化DLI客户端**完成客户端DLIClient的初始化，参考**队列相关**完成队列创建等操作。

新建 SQL 作业

DLI提供新建Flink SQL作业的接口。您可以使用该接口新建Flink SQL作业并提交到DLI，示例代码如下：

```
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);
}
```

新建自定义作业

DLI提供新建Flink自定义作业的接口。您可以使用该接口创建一个用户自定义作业，目前支持jar格式，运行在独享队列中。示例代码如下：

```
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);
}
```

更新 SQL 作业

DLI提供更新Flink SQL作业接口。您可以使用该接口更新Flink SQL作业，示例代码如下：

```
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);
}
```

更新自定义作业

DLI提供更新Flink自定义作业的接口。您可以使用该接口更新已经创建的自定义作业，目前仅支持Jar格式和运行在独享队列中。示例代码如下：

```
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);
}
```

查询作业列表

DLI提供查询Flink作业列表的接口。您可以使用该接口查询作业列表。作业列表查询支持以下参数: name，status，show_detail，cursor，next，limit，order。本示例排序方式选择降序desc，将会列出作业id小于cursor的作业列表信息。示例代码如下：

```
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);
}
```

查询作业详情

DLI提供查询Flink作业详情的接口。您可以使用该接口查询作业的详情。示例代码如下：

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

查询作业执行计划图

DLI提供查询Flink作业执行计划图的接口。您可以使用该接口查询作业的执行计划图。示例代码如下：

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

查询作业监控信息

DLI提供查询Flink作业监控信息的接口。您可以使用该接口查询作业监控信息，支持同时查询多个作业监控信息。示例代码如下：

```
public static void getMetrics(DLIClient client) throws DLIException{
    List < Long > job_ids = new ArrayList < > ();
    Long jobId = 6316L; //作业1ID
    Long jobId2 = 6945L; //作业2ID
```

```
    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);
}
```

查询作业 APIG 网关服务访问地址

DLI提供查询Flink作业APIG访问地址的接口。您可以使用该接口查询作业APIG网关服务访问地址。示例代码如下：

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

运行作业

DLI提供运行Flink作业的接口。您可以使用该接口触发运行作业。示例代码如下：

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

停止作业

DLI提供停止Flink作业的接口。您可以使用该接口停止一个正在运行的Flink作业。示例代码如下：

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

批量删除作业

DLI提供批量删除Flink作业的接口。您可以使用该接口批量删除任何状态的Flink作业。示例代码如下：

```
public static void deleteFlinkJob(DLIClient client) throws DLIException{
   DeleteJobInBatchRequest  body = new DeleteJobInBatchRequest ();
   List<Long> jobIds = new ArrayList<>();
   Long jobId = 202L;//作业1ID
   Long jobid2 = 203L;//作业2ID
```

```
    jobIds.add(jobId);
    jobIds.add(jobid2);
    body.jobIds(jobIds);
    List<GlobalBatchResponse>  result = client. deleteFlinkJobInBatch(body);
    System.out.println(result);
}
```

4.8 Spark 作业相关

前提条件

- 已参考**Java SDK概述**配置Java SDK环境。
- 已参考**初始化DLI客户端**完成客户端DLIClient的初始化，参考**队列相关**完成队列创建等操作。

提交批处理作业

DLI提供执行批处理作业的接口。您可以使用该接口执行批处理作业。示例代码如下：

```
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");
    // 调用BatchJob对象的asyncSubmit接口提交批处理作业
    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();
        }
    }
}
```

说明

- Cluster为用户自建的队列。
- 传参不能为JSON格式。
- 对应批处理作业提交提供两个接口：
  - 异步 asyncSubmit，提交后直接返回，不等待
  - 同步 submit，提交后会一直等待作业执行结束

删除批处理作业

DLI提供删除批处理作业的接口。您可以使用该接口删除批处理作业。示例代码如下：

```
private static void deleteBatchJob(DLIClient client) throws DLIException {
    //提交Spark批处理运行作业的Id
    String batchId = "0aae0dc5-f009-4b9b-a8c3-28fbee399fa6";
    // 调用BatchJob对象的delBatch接口取消批处理作业
```

```
    MessageInfo messageInfo = client.delBatchJob(batchId);
    System.out.println(messageInfo.getMsg());
}
```

查询所有批处理作业

DLI提供查询批处理作业的接口。您可以使用该接口查询当前工程下的所有批处理作业信息。示例代码如下：

```
private static void listAllBatchJobs(DLIClient client) throws DLIException {
  System.out.println("list all batch jobs...");
  // 通过调用DLIClient对象的listAllBatchJobs方法查询批处理作业
  String queueName = "queueName";
  int from = 0;
  int size = 1000;
  // 分页，起始页，每页大小
  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状态
    System.out.println(jobResult.getState());
  }
}
```

查询批处理作业状态

DLI提供查询批处理作业状态的接口。您可以使用该接口查询批处理作业当前的状态信息。示例代码如下：

```
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);

}
```

查询批处理作业日志

DLI提供查询批处理作业日志的接口。您可以使用该接口查询批处理作业的日志信息。示例代码如下：

```
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();
        // 调用BatchJob对象的getLog接口查询批处理作业日志
        int from = 0;
        int size = 1000;
        List<String> jobLogs = batchJob.getLog(from,size);
        System.out.println(jobLogs);

}
```

4.9 Flink 作业模板相关

前提条件

- 已参考**Java SDK概述**配置Java SDK环境。
- 已参考**初始化DLI客户端**完成客户端DLIClient的初始化。

新建作业模板

DLI提供新建Flink作业模板的接口。您可以使用该接口新建一个Flink作业模板。示例代码如下：

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

更新作业模板

DLI提供更新Flink作业模板的接口。您可以使用该接口修改一个Flink作业模板。示例代码如下：

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

删除作业模板

DLI提供删除Flink作业模板的接口。您可以使用该接口删除已经创建的作业模板，如果当前模板被引用也允许删除模板。示例代码如下：

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

查询作业模板列表

DLI提供查询Flink作业模板的接口。您可以使用该接口查询作业模板列表。本示例排序方式选择降序desc，将会列出作业模板ID小于cursor的作业模板列表信息。示例代码如下：

```
  public static void getFlinkJobTemplates(DLIClient client) throws DLIException{
      Long offset = 789L; // Long | 模板偏移量。
      Integer limit = 56; // Integer | 查询条数限制
      String order = "asc"; // String | 查询结果排序, 升序和降序两种可选
      FlinkJobTemplateListResponse result = client.getFlinkJobTemplates(offset,limit,order);
      System.out.println(result);
  }
```

5 Python SDK

5.1 Python SDK 概述

操作场景

DLI SDK让您无需关心请求细节即可快速使用数据湖探索服务。本节操作介绍如何在Python环境获取并使用SDK。

使用须知

- 要使用DLI Python SDK访问指定服务的 API ，您需要确认已在DLI管理控制台开通当前服务并完成服务授权。
- Python版本建议使用2.7.10和3.4.0以上版本，需要配置Visual C++编译环境Visual C++ build tools 或者 Visual Studio。

  关于Python开发环境的配置请参考**Python SDK环境配置**。
- DLI Python SDK依赖第三方库包括：urllib3 1.15以上版本，six 1.10以上版本，certifi，python-dateutil。
- 关于Python SDK的获取与安装请参考**SDK获取与安装**。
- 使用SDK工具访问DLI，需要用户初始化DLI客户端。用户可以使用AK/SK(Access Key ID/Secret Access Key)或Token两种认证方式初始化客户端，具体操作请参考**初始化DLI客户端**

Python SDK 列表

表 5-1 Python SDK 列表

类型	说明
队列相关	介绍查询所有队列的Python SDK使用说明。
资源相关	介绍上传资源包、查询所有资源包、查询制定资源包、删除资源包的Python SDK使用说明。

类型	说明
SQL作业相关	介绍数据库相关、表相关、作业相关的Python SDK使用说明。
Spark作业相关	介绍提交Spark作业、取消Spark作业、删除Spark作业等Python SDK使用说明。

5.2 Python SDK 环境配置

操作场景

在进行二次开发时，要准备的开发环境如表5-2所示。

表 5-2 开发环境

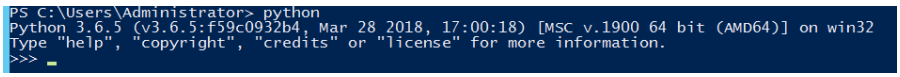
准备项	说明
操作系统	Windows系统，推荐Windows 7及以上版本。
安装Python	Python版本建议使用2.7.10和3.4.0以上版本，需要配置Visual C++编译环境Visual C++ build tools 或者 Visual Studio。
安装Python依赖库	DLI Python SDK依赖第三方库包括：urllib3 1.15以上版本，six 1.10以上版本，certifi，python-dateutil。

操作步骤

步骤1 从Python官网下载并安装Python版本。

1. 根据Python官方指导安装Python版本。
2. 检验是否配置成功，运行cmd ，输入 python。运行结果，请参见图5-1，显示版本信息，则说明安装和配置成功。

图 5-1 检验配置是否成功



说明

python安装应用包时出现错误类似错误“error: Microsoft Visual C++ xx.x is required. Get it with Build Tools for Visual Studio ”，可能是由于缺少C++编译器导致的报错，建议您根据提示信息安装相应版本的Visual Studio编译器解决。部分操作系统Visual Studio安装后需重启才可以生效。

步骤2 安装DLI服务Python SDK。

1. 选择SDK获取与安装获取的安装包，解压安装包。

   将"dli-sdk-python-<version>.zip"解压到本地目录，目录可自行调整。

2. 安装SDK。

   a. 打开Windows操作系统“开始”菜单，输入cmd命令。

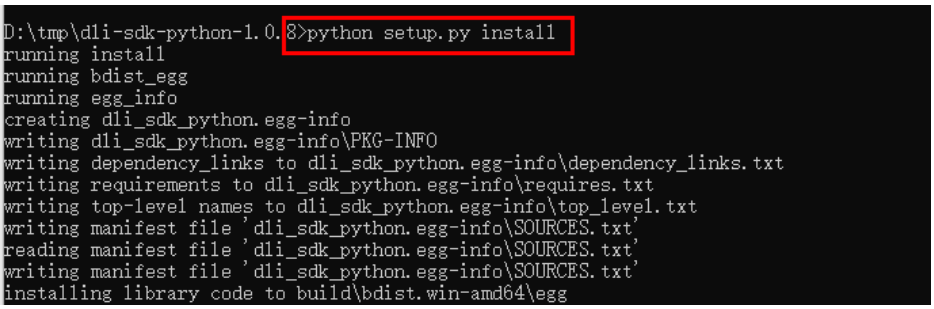
   b. 在命令行窗口，进入“dli-sdk-python-<version>.zip”解压目录下的windows目录。例如：“D:\tmp\dli-sdk-python-1.0.8”。

   c. 执行如下命令安装DLI服务Python SDK，安装过程中会自动下载第三方依赖库。

      python setup.py install

      运行结果参见图5-2所示。

      图 5-2 安装 Python SDK

----结束

5.2.1 Python 开发环境配置

操作场景

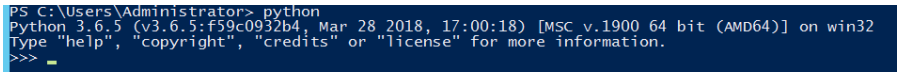
在安装和使用Python SDK前，确保您已经完成开发环境的基本配置。

Python版本建议使用2.7.10和3.4.0以上版本，需要配置Visual C++编译环境Visual C++ build tools 或者 Visual Studio。

操作步骤

1. 从**Python官网**下载并安装Python版本。
2. 根据Python官方指导安装Python版本。
3. 检验是否配置成功，运行cmd ，输入 python。运行结果，请参见图5-3，显示版本信息，则说明安装和配置成功。

   图 5-3 检验配置是否成功

说明

python安装应用包时出现错误类似错误“error: Microsoft Visual C++ xx.x is required. Get it with Build Tools for Visual Studio ”，可能是由于缺少C++编译器导致的报错，建议您根据提示信息安装相应版本的Visual Studio编译器解决。部分操作系统Visual Studio安装后需重启才可以生效。

5.2.2 SDK 获取与安装

Python SDK 安装方式

本节操作介绍安装Python SDK的操作指导。

获取 DLI SDK

1. 登录DLI管理控制台。
2. 单击总览页右侧“常用链接”中的“**SDK下载**”。
3. 在“DLI SDK DOWNLOAD”页面，选择相应驱动下载。
   “dli-sdk-python-*x.x.x*.zip”压缩包，解压后目录结构如下：

表 5-3 目录结构

名称	说明
dli	python环境的DLI SDK基础模块。
examples	python样例代码。
pyDLI	pyHive的实现接口。
setup.py	Python SDK安装脚本。

安装 DLI Python SDK

1. 下载并解压SDK安装包。
   将"dli-sdk-python-<version>.zip"解压到本地目录，目录可自行调整。
2. 安装SDK。
   a. 打开Windows操作系统“开始”菜单，输入cmd命令。
   b. 在命令行窗口，进入“dli-sdk-python-<version>.zip”解压目录下的windows目录。例如：“D:\tmp\dli-sdk-python-1.0.8”。
   c. 执行如下命令安装DLI服务Python SDK，安装过程中会自动下载第三方依赖库。
      python setup.py install
      运行结果参见**图5-4**所示。

图 5-4 安装 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
```

5.2.3 初始化 DLI 客户端

使用DLI Python SDK工具访问DLI，需要用户初始化DLI客户端。用户可以使用AK/SK(Access Key ID/Secret Access Key)或Token两种认证方式初始化客户端，示例代码如下。完整样例代码和依赖包说明请参考：**Python SDK概述**。

AK/SK 认证方式样例代码

- 代码样例

```
def init_aksk_dli_client():
    auth_mode = 'aksk'
    region = 'xxx'
    project_id = 'xxxx'
    ak =  System.getenv("xxx_SDK_AK")//访问密钥ID。
    sk = System.getenv("xxx_SDK_SK")//与访问密钥ID结合使用的密钥。
    dli_client = DliClient(auth_mode=auth_mode, region=region, project_id=project_id,ak=ak, sk=sk)
    return dli_client
```

- 参数说明与获取方式
  - 参数说明
    - ak：账号 Access Key
    - sk：账号 Secret Access Key

      说明

      认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。

      本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量xxx_SDK_AK和xxx_SDK_SK。
    - regionName ： 所属区域名称
    - projectId ： 项目ID
  - 通过以下方式可获取AK/SK，项目ID及对应的region信息。
    i. 登录管理控制台。
    ii. 鼠标指向界面右上角的登录用户名，在下拉列表中单击“我的凭证”。
    iii. 在左侧导航栏中选择“访问密钥”，单击“新增访问密钥”。根据提示输入对应信息，单击“确定”。
    iv. 在弹出的提示页面单击“立即下载”。下载成功后，打开凭证文件，获取AK/SK信息。
    v. 左侧导航栏单击“API凭证”，在“项目列表”中获取“项目ID”即为project_id值，对应的“项目”即为region的值。

Token 认证方式样例代码

- 代码样例

```
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
```

- 参数说明
  - domainname：帐号名。
  - username：用户名
  - password：用户名密码
  - regionname：所属区域名称
  - project_id：项目ID

  说明

  - 认证用的password硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
  - 可以通过set方式修改endpoint，即dliInfo.setServerEndpoint(endpoint)。

5.3 队列相关

约束限制

当前使用SDK创建的作业不支持在default队列上运行。

查询所有队列

DLI提供查询队列列表接口，您可以使用该接口并选择相应的队列来执行作业。示例代码如下：

```
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)
```

完整样例代码和依赖包说明请参考：**Python SDK概述**。

5.4 资源相关

前提条件

- 已参考**Python SDK概述**配置Java SDK环境。
- 已参考**初始化DLI客户端**完成客户端DLIClient的初始化。

上传资源包

您可以使用DLI提供的接口上传资源包，示例代码如下。完整样例代码和依赖包说明请参考：**Python SDK概述**。

```
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
```

说明

请求参数说明如下，详细参数使用可以参考**Python SDK概述**下载样例代码。

- kind：资源包类型，当前支持的包类型分别为：
  - **jar**：用户jar文件
  - **pyfile**：用户Python文件
  - **file**：用户文件
  - **modelfile**：用户AI模型文件
- obs_jar_paths：对应资源包的OBS路径，参数构成为：{bucketName}.{obs域名}/{jarPath}/{jarName}。

  例如："https://bucketname.obs.com/jarname.jar"
- group_name：资源包所属分组名称。

查询所有资源包

DLI提供查询资源列表接口，您可以使用该接口并选择相应的资源来执行作业。示例代码如下：

```
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)
```

完整样例代码和依赖包说明请参考：**Python SDK概述**。

查询指定资源包

您可以使用该接口查询指定的资源包信息，示例代码如下：

```
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
```

删除资源包

您可以使用该接口删除已上传的资源包，示例代码如下：

```
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
```

5.5 SQL 作业相关

5.5.1 数据库相关

创建数据库

DLI提供创建数据库的接口。您可以使用该接口创建数据库，示例代码如下：

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

    print(db)
```

说明

- “default”为内置数据库，不能创建名为“default”的数据库。
- 完整样例代码和依赖包说明请参考：**Python SDK概述**。

删除数据库

DLI提供删除数据库的接口。您可以使用该接口删除数据库。示例代码如下：

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

说明

- 含表的数据库不能直接删除，请先删除数据库的表再删除数据库。
- 数据库删除后，将不可恢复，请谨慎操作。
- 完整样例代码和依赖包说明请参考：**Python SDK概述**。

查询所有数据库

DLI提供查询数据库列表接口。您可以使用该接口查询当前已创建的数据库列表。示例代码如下：

```
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)
```

完整样例代码和依赖包说明请参考：**Python SDK概述**。

5.5.2 表相关

创建 DLI 表

DLI提供创建DLI表的接口。您可以使用该接口创建数据存储在DLI内部的表。示例代码如下：

```
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)
```

完整样例代码和依赖包说明请参考：**Python SDK概述**。

创建 OBS 表

DLI提供创建OBS表的接口。您可以使用该接口创建数据存储在OBS的表。示例代码如下：

```
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)
```

说明

- 创建OBS表需要指定OBS路径，且该路径需要提前创建。
- 完整样例代码和依赖包说明请参考：**Python SDK概述**。

删除表

DLI提供删除表的接口。您可以使用该接口删除数据库下的所有表。示例代码如下：

```
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
```

说明

- 表删除后，将不可恢复，请谨慎操作。
- 完整样例代码和依赖包说明请参考：**Python SDK概述**。

查询所有表

DLI提供查询表的接口。您可以使用该接口查询数据库下的所有表。示例代码如下：

```
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)
```

完整样例代码和依赖包说明请参考：**Python SDK概述**。

描述表信息

您可以使用该接口获取表的元数据描述信息。示例代码如下：

```
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
```

5.5.3 作业相关

完整样例代码和依赖包说明请参考：**Python SDK概述**。

导入数据

DLI提供导入数据的接口。您可以使用该接口将存储在OBS中的数据导入到已创建的DLI表中。示例代码如下：

```
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)
```

说明

- 在提交导入作业前，可选择通过data_type参数设置导入数据的类型，例如将data_type设置为csv。csv数据的具体格式通可过options参数设置，例如：csv的分隔符，转义符等。
- 当OBS桶目录下有文件夹和文件同名时，加载数据会优先指向该路径下的文件而非文件夹。建议创建OBS对象时，在同一级中不要出现同名的文件和文件夹。

导出数据

DLI提供导出数据的接口。您可以使用该接口将DLI表中的数据导出到OBS中。示例代码如下：

```
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)
```

说明

- 在提交导出作业前，可选设置数据格式、压缩类型、导出模式等，导出格式只支持csv格式。
- 当OBS桶目录下有文件夹和文件同名时，加载数据会优先指向该路径下的文件而非文件夹。建议创建OBS对象时，在同一级中不要出现同名的文件和文件夹。

提交作业

DLI提供查询作业的接口。您可以使用该接口执行查询并获取查询结果。示例代码如下：

```
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)
```

取消作业

DLI提供取消作业的接口。您可以使用该接口取消已经提交的作业，若作业已经执行结束或失败则无法取消。示例代码如下：

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

查询所有作业

DLI提供查询所有作业的接口。您可以使用该接口执行查询当前工程下的所有作业的信息并获取查询结果。示例代码如下：

```
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)
```

说明

该SDK接口不支持sql_pattern，即通过指定sql片段作为作业过滤条件进行查询。

如果需要则可以通过**查询所有作业**API接口指定该参数进行查询。

查询 SQL 类型作业

您可以使用该接口查询当前工程下的所有SQL类型作业的信息并获取查询结果。示例代码如下：

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

5.6 Spark 作业相关

完整样例代码和依赖包说明请参考：**Python SDK概述**。

提交批处理作业

DLI提供执行批处理作业的接口。您可以使用该接口执行批处理作业。示例代码如下：

```
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)
```

取消批处理作业

DLI提供取消批处理作业的接口。您可以使用该接口取消批处理作业。若作业已经执行结束或失败则无法取消。示例代码如下：

```
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
```

删除批处理作业

DLI提供删除批处理作业的接口。您可以使用该接口删除批处理作业。示例代码如下：

```
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
```