数据湖探索

SDK 参考

文档版本	01
发布日期	2025-09-10

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

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进行封装，以简化开发工作。

DLI提供两种SDK支持：一种是通用的SDK V3，另一种是DLI服务自研的SDK（包括DLI SDK V1和DLI SDK V2）。

本手册介绍的是DLI服务自研的SDK的使用操作指导。

表 1-1 DLI SDK 类型简介

类型	版本	说明	使用说明
通用SDK	V3	SDK V3的接口是根据定义API的YAML文件统一自动生成的，接口参数与服务的API保持一致。您可以直接调用SDK V3提供的接口函数，实现使用提交DLI SQL和DLI Spark作业的目的，从而简化开发流程。 **您在API Explorer页面下载使用的SDK就是通用版本 SDK V3。** **图 1-1 API Explorer-SDK** 	**通用SDK V3**
DLI服务自行开发的SDK	DLI SDK V1	DLI服务自行开发的SDK，对应DLI SDK安装包中的 dli-sdk-x-1.x.x版本。	计划废弃，不推荐使用。

类型	版本	说明	使用说明
	DLI SDK V2	DLI服务基于通用通用的SDK V3自行开发的SDK，对应DLI SDK安装包中的 dli-sdk-x-2.x.x 版本。 可以直接调用DLI SDK V2提交DLI SQL和DLI Spark作业的目的，从而简化开发流程。 请在DLI管理控制台下载DLI SDK V2的安装包。 **说明** DLI SDK V2当前处于公测阶段，如需使用请提交工单申请开通。 • 2024年5月起，首次使用DLI的用户可以直接使用DLI SDK V2，无需申请。 • 对于2024年5月之前开通并使用DLI服务的用户，如需使用“DLI SDK V2”功能，必须提交工单申请开通试用权限。	**使用SDK提交SQL作业** **使用SDK提交Spark作业** **使用SDK提交Flink SQL作业** **使用SDK提交Flink Jar作业**

说明

DLI SDK调用接口使用https进行访问，有服务端使用证书。

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。相关开发包请从**华为云DLI 开发工具包（SDK）**获取。
  - DLI SDK V2版本操作指导：
    - Java SDK操作指导请参考**Java SDK（DLI SDK V2）**。
    - Python SDK操作指导请参考**Python SDK（DLI SDK V2）**
  - DLI SDK V1版本操作指导（不推荐使用）：
    - Java SDK操作指导请参考**Java SDK（DLI SDK V1）**
    - Python SDK操作指导请参考**Python SDK（DLI SDK V1）**

说明

- DLI SDK调用接口使用https进行访问，有服务端使用证书。
- 建议您同时下载SDK安装包中的.sha256文件，该文件可用于验证SDK包的完整性。

2 Java SDK 环境配置

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2-1**，显示版本信息，则说明安装和配置成功。

图 2-1 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)
```

2.2 Java SDK 的获取与安装

获取 DLI SDK

在“**DLI SDK DOWNLOAD**”页面，单击选择所需的SDK链接，即可获取对应的SDK安装包。

建议您同时下载SDK安装包中的.sha256文件，该文件可用于验证SDK包的完整性。验证方法请参考**验证SDK包的完整性**。

表 2-1 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。

验证 SDK 包的完整性

下载SDK安装包时，您需要同时下载.sha256文件，用于验证SDK包的完整性。本节介绍使用.sha256文件验证SDK包的完整性的操作步骤。

1. **下载文件**
   a. 在SDK安装包的下载页面找到对应的.sha256文件的下载链接。
      通常.sha256文件的名称会与SDK安装包的名称相关联，例如SDK安装包名为“dli-sdk-java-x.x.x”，对应的.sha256文件可能是“dli-sdk-java-x.x.x.sha256”。
   b. 分别下载SDK安装包和.sha256文件到本地主机，确保两个文件保存在同一个文件夹中。
2. **查询SDK安装包的SHA - 256哈希值**
   - **在Windows系统中**
     i. 在开始菜单中搜索“cmd”打开命令窗口。
     ii. 使用cd命令切换到保存SDK安装包的文件夹目录。
        例如：SDK安装包保存在D盘的“SDK”文件夹中，输入**cd D:\SDK**。
     iii. 执行以下命令
        certutil -hashfile *SDK安装包文件名* SHA256
        将“SDK安装包文件名”替换为实际的SDK安装包文件名，如**certutil -hashfile sdk - package.zip SHA256**。
     iv. 命令执行后，命令提示符会输出SDK安装包的SHA - 256哈希值。
   - **在Linux系统中**
     i. 使用cd命令切换到保存SDK安装包的文件夹目录。
        例如，如果SDK安装包保存在用户主目录下的“SDK”文件夹中，输入**cd ~/SDK**。

ii. 输入查询命令：

sha256sum *SDK安装包文件名*

将“SDK安装包文件名”替换为实际的SDK安装包文件名，如**sha256sum sdk - package.zip**。

iii. 命令执行后，终端会输出SDK安装包的SHA - 256哈希值。

3. **对比哈希值**

a. 使用文本编辑器打开下载的.sha256文件。

b. 在.sha256文件中找到SDK安装包的SHA - 256哈希值。

通常是一个长字符串，例如“a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6xxxxxxxxxxxxxxxxxxxxxxxx”。

c. 将步骤**2.查询SDK安装包的SHA - 256哈希值**得到的SDK安装包的SHA - 256哈希值与.sha256文件中的哈希值进行对比。

- 如果两个哈希值完全一致，说明SDK安装包在下载过程中没有被篡改，文件是完整的。
- 如果哈希值不一致，这可能意味着SDK安装包在下载过程中出现了错误，或者文件被恶意修改。

  此时，建议重新下载SDK安装包和对应的.sha256文件，然后再次进行验证。

安装 SDK

- **方式一：使用Maven中央库来添加SDK**

  Maven中央库是Apache Maven项目的一部分，提供了Java库和框架。

  在不指定SDK获取方式的情况下，默认使用Maven中央库的方式来添加SDK驱动。

  使用maven构加入huaweicloud-dli-sdk依赖的maven配置项为（此为默认操作无需单独配置。）

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

- **方式二：通过Maven配置华为镜像源来获取SDK**

  在使用Maven管理项目依赖时，可以通过修改settings.xml文件来配置华为镜像源以获取SDK。

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

- **方式三：在DLI管理控制台下载JDBC驱动文件**

  a. 登录DLI管理控制台。

  b. 单击总览页右侧“常用链接”中的“**SDK下载**”。

  c. 在“DLI SDK DOWNLOAD”页面，选择相应驱动下载。

  单击“huaweicloud-dli-sdk-java-x.x.x”即可下载对应版本的JDBC驱动包。

说明

JDBC驱动包命名为“huaweicloud-dli-sdk-java-<version>.zip”，支持在所有平台（Linux、Windows等）所有版本中使用，且依赖JDK 1.7及以上版本。

d. 下载的JDBC驱动包中包含了.bat（Windows）或.sh（Linux/Mac）脚本，这些脚本用于自动化安装SDK驱动到本地Maven仓库。

您可以根据操作系统运行相应的脚本安装JDBC驱动

- Windows：双击.bat文件或在命令行中运行。
- Linux/Mac：运行.sh脚本。

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。其中Token认证仅DLI SDK V1版本支持，推荐使用AK/SK认证方式。

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

  参数获取方式请参考**获取账号、IAM用户、项目、用户组、区域、委托的名称和ID**。

  - domainname：账号名。
  - 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)。

3 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3-1所示。

表 3-1 开发环境

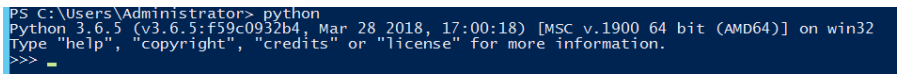
准备项	说明
操作系统	Windows系统，推荐Windows 7及以上版本。
安装Python	Python版本建议使用2.7.10和3.4.0以上版本，需要配置Visual C++编译环境Visual C++ build tools 或者 Visual Studio。
安装Python依赖库	DLI Python SDK依赖第三方库包括：urllib3 1.15以上版本，six 1.10以上版本，certifi，python-dateutil。

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3-1，显示版本信息，则说明安装和配置成功。

图 3-1 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**Python 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3-2**所示。

      图 3-2 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
      ```

----结束

3.1 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3-3**，显示版本信息，则说明安装和配置成功。

   图 3-3 检验配置是否成功

   ```
   PS C:\Users\Administrator> python
   Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
   Type "help", "copyright", "credits" or "license" for more information.
   >>>
   ```

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。

3.2 Python 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

在“**DLI SDK DOWNLOAD**”页面，单击选择所需的SDK链接，即可获取对应的SDK安装包。

建议您同时下载SDK安装包中的.sha256文件，该文件可用于验证SDK包的完整性。验证方法请参考**验证SDK包的完整性**。

“dli-sdk-python-*x.x.x*.zip”压缩包，解压后目录结构如下：

表 3-2 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。

验证 SDK 包的完整性

下载SDK安装包时，您需要同时下载.sha256文件，用于验证SDK包的完整性。本节介绍使用.sha256文件验证SDK包的完整性的操作步骤。

1. **下载文件**
   a. 在SDK安装包的下载页面找到对应的.sha256文件的下载链接。
      通常.sha256文件的名称会与SDK安装包的名称相关联，例如SDK安装包名为“dli-sdk-java-x.x.x”，对应的.sha256文件可能是“dli-sdk-java-x.x.x.sha256”。
   b. 分别下载SDK安装包和.sha256文件到本地主机，确保两个文件保存在同一个文件夹中。
2. **查询SDK安装包的SHA - 256哈希值**
   - **在Windows系统中**
     i. 在开始菜单中搜索“cmd”打开命令窗口。

ii. 使用cd命令切换到保存SDK安装包的文件夹目录。

例如：SDK安装包保存在D盘的“SDK”文件夹中，输入**cd D:\SDK**。

iii. 执行以下命令

certutil -hashfile *SDK安装包文件名* SHA256

将“SDK安装包文件名”替换为实际的SDK安装包文件名，如**certutil -hashfile sdk - package.zip SHA256**。

iv. 命令执行后，命令提示符会输出SDK安装包的SHA - 256哈希值。

– **在Linux系统中**

i. 使用cd命令切换到保存SDK安装包的文件夹目录。

例如，如果SDK安装包保存在用户主目录下的“SDK”文件夹中，输入**cd ~/SDK**。

ii. 输入查询命令：

sha256sum *SDK安装包文件名*

将“SDK安装包文件名”替换为实际的SDK安装包文件名，如**sha256sum sdk - package.zip**。

iii. 命令执行后，终端会输出SDK安装包的SHA - 256哈希值。

3. **对比哈希值**

a. 使用文本编辑器打开下载的.sha256文件。

b. 在.sha256文件中找到SDK安装包的SHA - 256哈希值。

通常是一个长字符串，例如“a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6xxxxxxxxxxxxxxxxxxxxxxxx”。

c. 将步骤**2.查询SDK安装包的SHA - 256哈希值**得到的SDK安装包的SHA - 256哈希值与.sha256文件中的哈希值进行对比。

- 如果两个哈希值完全一致，说明SDK安装包在下载过程中没有被篡改，文件是完整的。
- 如果哈希值不一致，这可能意味着SDK安装包在下载过程中出现了错误，或者文件被恶意修改。

  此时，建议重新下载SDK安装包和对应的.sha256文件，然后再次进行验证。

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。

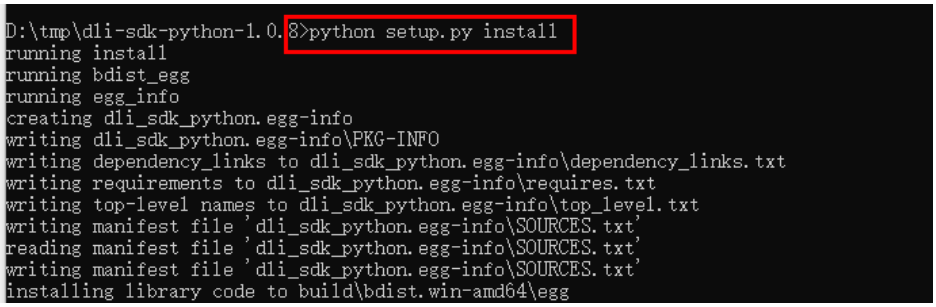
b. 在命令行窗口，进入“dli-sdk-python-<version>.zip”解压目录下的windows目录。例如：“D:\tmp\dli-sdk-python-1.0.8”。

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3-4**所示。

图 3-4 安装 Python SDK



3.3 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。其中Token认证仅DLI SDK V1版本支持，推荐使用AK/SK认证方式。

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

  参数获取方式请参考**获取账号、IAM用户、项目、用户组、区域、委托的名称和ID**。

  - domainname：账号名。
  - 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 通用 SDK V3

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进行的封装，以简化开发工作。

本文介绍了通用SDK V3版本的SDK的使用方法，列举了最新版本SDK的获取地址。

SDK V3版本开发指南。

SDK 列表

表4-1提供了通用SDK V3的DLI支持列表，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。

表 4-1 SDK 列表

编程语言	Github地址	参考文档	视频指导
Java	**huaweicloud-sdk-java-v3**	**Java SDK使用指导**	**Java SDK视频指导**
Python	**huaweicloud-sdk-python-v3**	**Python SDK使用指导**	**Python SDK视频指导**
PHP	**huaweicloud-sdk-php-v3**	**PHP SDK使用指导**	**PHP SDK视频指导**
Go	**huaweicloud-sdk-go-v3**	**Go SDK使用指导**	**Go SDK视频指导**
Node.js	**huaweicloud-sdk-nodejs-v3**	**Node.js SDK使用指导**	**Node.js SDK视频指导**
.NET	**huaweicloud-sdk-net-v3**	**.NET SDK使用指导**	**.NET SDK视频指导**

在线生成 SDK 代码

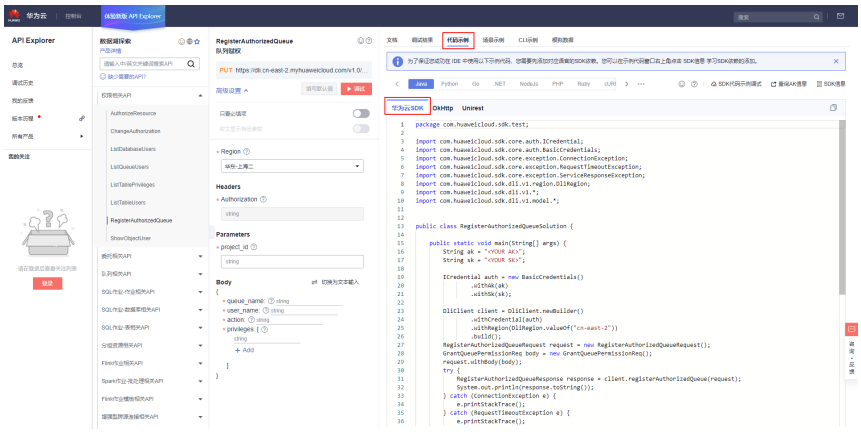
【样例】

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。

如图4-1所示。

图 4-1 获取 SDK 代码示例



5 DLI SDK V2

5.1 Java SDK（DLI SDK V2）

5.1.1 使用 SDK 提交 SQL 作业

本节操作介绍使用DLI SDK V2提交SQL作业的操作方法。

说明

DLI SDK V2当前处于公测阶段，如需使用请提交工单申请开通。

- 2024年5月起，首次使用DLI的用户可以直接使用DLI SDK V2，无需申请。
- 对于2024年5月之前开通并使用DLI服务的用户，如需使用“DLI SDK V2”功能，必须提交工单申请开通试用权限。

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。

操作前准备

获取AK/SK，项目ID及对应的Region信息。

1. 管理控制台。
2. 单击界面右上角的登录用户名，在下拉列表中单击“我的凭证”。
3. 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”。
4. 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。
5. 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。

样例代码

```
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 {
            // 步骤一：创建数据库、表。
            prepare(dliInfo);

            /*
             * 步骤二：导入数据到表中。
             * 整体实现过程/原理可分为以下3步：
             * 1. 用OBS的API把数据上传到 “YourOBSPathToWriteTmpData”。可在OBS中配置生命周期策略，定期删除这些临时数据。
             * 2. 向DLI提交执行LoadData语句，从而把数据导入到DLI。详见导入数据。
             * 3. 每隔1秒循环检查作业运行状态，直到作业结束。
             */
            String yourOBSPathToWriteTmpData = String.format("obs://your_obs_bucket_name/your/path/%s", UUID.randomUUID());
            loadData(dliInfo, yourOBSPathToWriteTmpData);

            // 步骤三：提交SQL语句，执行查询并读取结果。
            String selectSql = "SELECT * FROM demo_db.demo_tbl";
            String jobId = queryData(dliInfo, selectSql);

            // 步骤四': 如有需要，用户也可以通过作业ID来获取结果。
            queryDataByJobId(dliInfo, jobId);

            // 分页查询所有作业，用户可以使用该接口查询当前工程下的所有SQL作业信息
            // 关键SDK API：com.huaweicloud.sdk.dli.v1.DliClient#listSqlJobs(ListSqlJobsRequest)。
            //详见ListSqlJobs
            listSqlJobs(dliInfo)。

            /*
             * 其它场景：
             * 1. 如果用户想取消已经提交的SQL作业，可使用以下接口。
             * 关键SDK API：com.huaweicloud.sdk.dli.v1.DliClient#cancelSqlJob(CancelSqlJobRequest)，详见取消作业。
             * 注：若作业已经执行结束或失败则无法取消。
             * 2. 如果用户想对SQL语句做语法校验，可使用以下接口。
             * 关键SDK API：com.huaweicloud.sdk.dli.v1.DliClient#checkSql(CheckSqlRequest)，详见检查SQL语法。
             * 注：本接口只能做语法校验，无法做语义校验。请使用Explain语句，提交到DLI执行，进行语义校验。
             * 3. 如果用户想根据jobId获取某个已提交的SQL作业，并查看作业详情，可使用以下接口。
             * 关键SDK API：com.huaweicloud.sdk.dli.v1.DliClient#showSqlJobDetail(ShowSqlJobDetailRequest)，详见查询作业详细信息。
             * 4. 获取作业执行进度信息，如果作业正在执行，可以获取到子作业的信息，如果作业刚开始或者已经结束，则无法获取到子作业信息
             * 关键SDK API：com.huaweicloud.sdk.dli.v1.DliClient#showSqlJobProgress(ShowSqlJobProgressRequest)，详见查询作业执行进度信息。
             */
        } catch (DLIException e) {
            // 请根据业务实际情况处理异常信息，此处仅作样例。
        }
    }
```

创建数据库和表

相关链接：

- 创建数据库语法参考

- 创建表语法参考
- 关键SDK API：com.huawei.dli.sdk.Job#submit()

示例代码：

```
private static String prepare(DLIInfo dliInfo) throws DLIException {
    // 1. 创建数据库。
    // “default”为内置数据库，不能创建名为“default”的数据库。
    String createDbSql = "CREATE DATABASE IF NOT EXISTS demo_db";
    new SQLJob(dliInfo, createDbSql).submit();
    // 2. 创建表。注：根据实际情况调整表结构和表数据目录,以及OBS存储路径！！！
    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();
}
```

导入数据

- **DLI SDK导入数据实现过程可分为以下3步：**
  a. 使用OBS的API把数据上传到临时的OBS目录下，即“YourOBSPathToWriteTmpData”。
  b. 向DLI提交执行LoadData语句，把数据从临时的OBS目录导入到DLI。导入数据
  c. 循环检查作业运行状态，间隔1秒，直至作业结束。
- **导入数据说明：**
  - 在提交导入数据的作业前，可选择配置导入数据的分区、配置是否是覆盖写入数据（默认追加写入数据）。
    - 如需将数据插入某些具体的分区, 则可以使用以下构造器:
      ```
      new SqlJobBase.TableInfo("demo_db", "demo_tbl", new LinkedHashMap<String,String>()
      {{put("YourPartitionColumnName","value");}});
      ```
    - 导入作业默认是追加写入，如需覆盖写入数据，则可以使用以下构造器:
      ```
      new SqlJobBase.TableInfo("demo_db", "demo_tbl", true);
      ```
    - 如需导入覆盖写分区数据，则可以使用以下构造器:
      ```
      new SqlJobBase.TableInfo("demo_db", "demo_tbl", new LinkedHashMap<String,String>()
      {{put("YourPartitionColumnName","value");}}, 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。
- **相关链接：**
  - 导入数据

  - 原生数据类型
  - 复杂数据类型
  - 查询作业状态
- **关键SDK API：**
  - com.huawei.dli.sdk.write.Writer、
  - com.huawei.dli.sdk.Job#asyncSubmit()
  - com.huaweicloud.sdk.dli.v1.DliClient#showSqlJobStatus
- **示例代码：**
  - （推荐）方案1：使用LoadData语句执行数据导入

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

    // 1. 写入数据到OBS临时目录。请根据业务实际情况做调整，此处仅作样例。
    // 注：此步骤为直接调用OBS写数据相关API，DLI仅提供了一个默认写Json的实现，即文件在OBS上的保存格式为Json。
    // 此处Writer可依据业务需求自定义实现，比如，用户可使用自定义的 csv writer，则文件在OBS上的保存格式为csv。
    writeTmpData(uploadJob.createWriter(), genSchema(), 123, 50);

    // 2. 将数据导入到DLI。
    // 提交LoadData语句执行。
    // 注：此处的data_type需要根据第一步中的Writer实现来确定，DLI默认提供的为JSON。如用户使用自定义的Writer，则需要修改成响应的 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. 循环检查作业运行状态。
    checkRunning(V3ClientUtils.getDliClient(dliInfo), sqlJob.getJobId());
}
```

  - 方案2：使用DLI封装SDK提交数据导入

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

    // 1. 写入数据到OBS临时目录。请根据业务实际情况做调整，此处仅作样例。
    // 注：此步骤为直接调用OBS写数据相关API，DLI仅提供了一个默认写Json的实现，即文件在OBS上的保存格式为Json。
    // 此处Writer可依据业务需求自定义实现，比如，用户可使用自定义的 csv writer，则文件在OBS上的保存格式为csv。
    writeTmpData(uploadJob.createWriter(), genSchema(), 123, 50);

    // 2. 将数据导入到DLI。
    // 使用DLI封装实现提交。注：因导入作业可能运行时间较长，因此使用asyncSubmit提交数据，并主动检查状态。
    uploadJob.asyncSubmit();

    // 3. 循环检查作业运行状态。
    checkRunning(V3ClientUtils.getDliClient(dliInfo), uploadJob.getJobId());
}
```

查询作业结果

- **相关链接：**

  SELECT查询语句

  作业参数设置

- **关键SDK API：**

  com.huawei.dli.sdk.read.ResultSet，纯OBS读数据相关API调用，DLI提供了一个默认 OBS csv reader实现，可依据业务需求自定义实现。具体操作参考**OBS API参考**。

  com.huawei.dli.sdk.SQLJob#submitQuery()，需要开启结果写作业桶特性，否则默认只预览前1000条数据。

  您可以通过**查询作业状态API**响应体中的 result_path 来判断是否已开启作业结果写作业桶特性。

  待作业运行结束后，如果result_path 以 obs:// 开头，则已开启作业结果写作业桶特性，否则未开启。

```
private static String queryData(DLIInfo dliInfo, String selectSql)
    throws DLIException {
    SQLJob sqlJob = new SQLJob(dliInfo, selectSql);
    // 如有需要，您可在此处设置作业参数，比如：sqlJob.setConf()
    // 1. 提交查询作业到DLI，使用DLI封装实现提交并等待结果。
    // 注1：此处需要根据SQL执行时长预取设置超时时间，默认5min。
    // 注2：此处需要开启作业结果写作业桶特性，否则默认只预览前1000条数据。
    sqlJob.setJobTimeout(30 * 60);
    ResultSet resultSet1 = null;
    try {
        resultSet1 = sqlJob.submitQuery();
        handleResult(resultSet1);
    } finally {
        if (resultSet1 != null) {
            resultSet1.close();
        }
    }
    return sqlJob.getJobId();
}
```

查询指定作业的结果

- **使用说明**

  com.huawei.dli.sdk.SQLJob#getResultSet()，OBS读数据相关API调用，DLI提供了一个OBS csv reader实现，可依据业务需求自定义实现。

  使用本方法的前提是需要开启结果写作业桶特性。可通过查询作业状态API响应体中的 result_path 来判断是否已开启作业结果写作业桶特性。待作业运行结束后，如果result_path 以 obs:// 开头，则已开启作业结果写作业桶特性，否则未开启。

- **相关链接**

  SQL语法参考基本语句

  OBS文档

  查询作业状态API

- **示例代码**

```
private static void queryDataByJobId(DLIInfo dliInfo, String jobId)
    throws DLIException {
    // 检查该jobId对应的作业是否已运行结束，如未运行结束，则等待运行结束
    SQLJob sqlJob = new SQLJob(dliInfo, null);
    sqlJob.setJobId(jobId);
    checkRunning(V3ClientUtils.getDliClient(dliInfo), jobId);

    // 根据jobId，获取结果数据的schema、结果数据在用户作业桶中的存储路径
    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 {
        // 获取该jobId对应的查询结果, 并返回结果迭代器。
        resultSet = sqlJob.getResultSet();
        handleResult(resultSet);
    } finally {
        if (resultSet != null) {
            resultSet.close();
        }
    }
}
```

查询作业

- **使用说明**

  com.huaweicloud.sdk.dli.v1.DliClient#listSqlJobs(ListSqlJobsRequest) **查询所有作业**

  如果作业较多，必须使用下述分页查询的方式分批查询，否则只能返回第一页的内容，无法返回全部作业。

  可通过设置 req.setStart() 和 req.setEnd() 查询指定时间段内的作业，单位毫秒。

- **示例代码**

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

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

            // 在此处添加业务逻辑，对每个作业进行处理

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

写入数据到 OBS writeTmpData

- **使用说明**

  您可以根据您自己的业务需求实现Writer接口并自定义相关写文件逻辑。

  本例写入数据到OBS，调用OBS SDK能力，与DLI无关，当前UploadJob提供了一个默认写JSON的实现。

- **示例代码**

```
private static void writeTmpData(Writer writer, List<Column> schema, Integer totalRecords, Integer flushThreshold)
    throws DLIException {
    // 定义写数据到OBS的Writer，可以根据并发需要定义多个
    // 根据目标表定义写入数据的列信息，详见本文的genSchema()方法和genSchema2()方法
    // 根据实际业务定义循环大小
```

```
        for (int i = 0; i < totalRecords; i++) {
            // 根据业务获取每一行的数据
            List<Object> record = genRecord();
            // 写入数据
            Row row = new Row(schema);
            row.setRecord(record);
            writer.write(row);
            if (i % flushThreshold == 0) {
                // 写入一定数据量后及时刷新
                writer.flush();
            }
        }
        writer.close();
    }
```

构建表的 Schema

- **使用说明**

  请根据业务实际情况构造相应的Schema，此处仅作样例。

- **示例代码**

```
    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"));
    }
```

自动获取目标表的 Schema

- **使用说明**

  自动获取目标表的schema。

- **示例代码**

```
    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());
}
```

按需生成测试数据 List<Object> genRecord

- **使用说明**

  请根据业务实际情况构造相应的每一行数据，此处仅作样例。

- **示例代码**

```
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);
}
```

查询作业状态

- **相关链接**

  查询作业状态

- **示例代码**

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

处理作业结果

- **使用说明**

  请根据业务实际情况构造相应的每一行数据，此处仅作样例。

- **示例代码**

```
    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();
            // 对每一行每一列进行处理
            //   方法一：调用record.get(index)获取Object，然后根据各列的类型做类型转换
            //   方法二：根据各列的类型调用row.getXXX(index)获取对应类型的数据
            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 使用 SDK 提交 Flink SQL 作业

本节操作介绍使用DLI SDK V2提交Flink SQL作业的操作方法。

说明

DLI SDK V2当前处于公测阶段，如需使用请提交工单申请开通。

- 2024年5月起，首次使用DLI的用户可以直接使用DLI SDK V2，无需申请。
- 对于2024年5月之前开通并使用DLI服务的用户，如需使用“DLI SDK V2”功能，必须提交工单申请开通试用权限。

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。

操作前准备

获取AK/SK，项目ID及对应的Region信息。

1. 管理控制台。
2. 单击界面右上角的登录用户名，在下拉列表中单击“我的凭证”。
3. 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”。
4. 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。
5. 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。

样例代码

```
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 {
        // 步骤一：创建Flink作业。作业状态将变成 “草稿”
        Long jobId = createFlinkSqlJob(dliClient, "YourQueueName");
        logger.info("jobId: " + jobId);

        // 步骤二：运行作业。作业状态将从 “草稿” 变成 “提交中”
        List<FlinkSuccessResponse> resps = batchRunFlinkJobs(dliClient, Arrays.asList(jobId));
        logger.info("Response: " + ArrayUtils.toString(resps));

        // 步骤三：查询作业状态。如果您想在当前线程等待作业进入“运行中”状态，可循环检查状态，直到作业进入“运行中”状态。
        checkRunning(dliClient, jobId);

    } catch (DLIException e) {
        // 请根据业务实际情况处理异常信息，此处仅作样例。
    }

}
```

创建 Flink SQL 作业

- **功能介绍：**

  用于创建Flink SQL作业。

- **相关链接：**

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

新建Flink SQL作业。

创建 Flink SQL作业，此时作业状态将变成 “草稿”。

- **示例代码**：

```
private static Long createFlinkSqlJob(DliClient client, String queueName) {
        // 请根据业务实际情况设置相应的参数，此处仅作样例。
        CreateFlinkSqlJobResponse resp = client.createFlinkSqlJob(new CreateFlinkSqlJobRequest()
            .withBody(new CreateFlinkSqlJobRequestBody()
                .withName("demo_flink_sql") // 自定义作业名称。长度限制：1-57个字符
                .withDesc("YourJobDescription") // 自定义作业描述。长度限制：0-512个字符
                .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;")
                // 自定义 Stream SQL语句，至少包含source, query, sink三个部分。长度限制：1024*1024个字
符。
                // 本SQL示例：自动生成随机source数据，并打印到控制台。
                .withQueueName(queueName) // 队列名称。长度限制：0-128个字符
                .withRunMode("exclusive_cluster") // 作业运行模式。只支持 exclusive_cluster 独享模式。
                .withLogEnabled(true) // 开启作业的日志上传到用户的OBS功能
                .withObsBucket("YourObsBucketName") // OBS桶名。用于保存 日志 和 checkpoint数据
                .withJobType("flink_opensource_sql_job") // 作业类型。建议选择: "flink_opensource_sql_job"
                .withFlinkVersion("1.12") // 指定Flink版本
            ));
        return resp.getJob().getJobId();
    }
```

批量运行 Flink 作业

- **功能介绍**：

  批量运行Flink SQL作业。

- **相关链接**：

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

  批量运行Flink作业

  批量运行Flink作业，作业状态将从 “草稿” 变成 “提交中”。

- **示例代码**：

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

查询作业状态

- **功能介绍**：

查询Flink SQL作业状态。

- **相关链接：**

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

  查询Flink作业详情

  如果想在当前线程等待作业进入“运行中”状态，可循环检查状态，直到作业进入“运行中”状态。

- **示例代码**：

```
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 使用 SDK 提交 Flink Jar 作业

本节操作介绍使用DLI SDK V2提交Flink Jar作业的操作方法。

说明

DLI SDK V2当前处于公测阶段，如需使用请提交工单申请开通。

- 2024年5月起，首次使用DLI的用户可以直接使用DLI SDK V2，无需申请。
- 对于2024年5月之前开通并使用DLI服务的用户，如需使用“DLI SDK V2”功能，必须提交工单申请开通试用权限。

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。

操作前准备

获取AK/SK，项目ID及对应的Region信息。

1. 管理控制台。
2. 单击界面右上角的登录用户名，在下拉列表中单击“我的凭证”。
3. 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”。

4. 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。
5. 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。

样例代码

```
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 {
            // 步骤一：创建Flink作业。作业状态将变成 “草稿”
            Long jobId = createFlinkJarJob(dliClient, "YourQueueName");
            logger.info("jobId: " + jobId);
            // 步骤二：运行作业。作业状态将从 “草稿” 变成 “提交中”
            List<FlinkSuccessResponse> resps = batchRunFlinkJobs(dliClient, Arrays.asList(jobId));
            logger.info("Response: " + ArrayUtils.toString(resps));
            // 步骤三：查询作业状态。如果您想在当前线程等待作业进入“运行中”状态，可循环检查状态，直到作业进入“运行中”状态。
            checkRunning(dliClient, jobId);
        } catch (DLIException e) {
            // 请根据业务实际情况处理异常信息，此处仅作样例。
        }
    }
```

创建 Flink Jar 作业

- **功能介绍：**

  用于创建Flink Jar作业。

- **相关链接：**

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

  新建Flink Jar作业

  创建 Flink Jar作业。作业状态将变成 “草稿”

- **示例代码：**

```
private static Long createFlinkJarJob(DliClient client, String queueName) {
        // 请根据业务实际情况设置相应的参数，此处仅作样例。
        CreateFlinkJarJobResponse resp = client.createFlinkJarJob(new CreateFlinkJarJobRequest()
            .withBody(new CreateFlinkJarJobRequestBody()
                .withName("demo_flink_jar") // 自定义作业名称。长度限制：1-57个字符。
                .withDesc("YourJobDescription") // 自定义作业描述。长度限制：0-512个字符
                .withQueueName(queueName) // 队列名称。长度限制：0-128个字符
                .withFeature("basic") // 作业特性。表示用户作业使用的Flink镜像类型。basic：表示使用DLI提供的基础Flink镜像。
                .withFlinkVersion("1.12") // Flink版本。当用户设置“feature”为“basic”时，该参数生效
                .withObsBucket("YourObsBucketName") // OBS桶名。用于保存 日志 和 checkpoint数据
                .withLogEnabled(true) // 开启作业的日志上传到用户的OBS功能
                .withEntrypoint("obs://YourObsBucketName/your/flink/job.jar") // 用户已上传到OBS的jar包，用户自定义作业主类所在的jar包。
                .withMainClass("YourClassFullName") // 作业入口类,比如:org.apache.flink.examples.JavaQueueStream
                .withEntrypointArgs("YourAppArg1 YourAppAgr2") // 作业入口类参数，多个参数之间空格分隔。如不需要，删除此行
```



```
                .withDependencyJars(Arrays.asList("obs://YourObsBucketName/your/dependency1.jar",
                    "obs://YourObsBucketName/your/dependency2.jar")) // 用户已上传到OBS的jar包，用户自定义作业的其他依赖包。如不需要，删除此行
                .withDependencyJars(Arrays.asList("obs://YourObsBucketName/your/dependency1.csv",
                    "obs://YourObsBucketName/your/dependency2.json")) // 用户已上传到OBS的文件，用户自定义作业的依赖文件。如不需要，删除此行
            ));
        return resp.getJob().getJobId();
    }
```

批量运行 Flink 作业

- **功能介绍：**

  批量运行Flink SQL作业。

- **相关链接：**

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

  批量运行Flink作业

  批量运行Flink作业，作业状态将从 “草稿” 变成 “提交中”。

- **示例代码：**

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

查询作业状态

- **功能介绍：**

  查询Flink SQL作业状态。

- **相关链接：**

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

  查询Flink作业详情。

  如果想在当前线程等待作业进入“运行中”状态，可循环检查状态，直到作业进入“运行中”状态。

- **示例代码：**

```
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 使用 SDK 提交 Spark 作业

本节操作介绍使用DLI SDK V2提交Spark作业的操作方法。

说明

DLI SDK V2当前处于公测阶段，如需使用请提交工单申请开通。

- 2024年5月起，首次使用DLI的用户可以直接使用DLI SDK V2，无需申请。
- 对于2024年5月之前开通并使用DLI服务的用户，如需使用“DLI SDK V2”功能，必须提交工单申请开通试用权限。

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。

操作前准备

获取AK/SK，项目ID及对应的Region信息。

1. 管理控制台。
2. 单击界面右上角的登录用户名，在下拉列表中单击“我的凭证”。
3. 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”。
4. 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。
5. 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。

样例代码

```
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 {
        // 步骤一：提交Spark作业到DLI执行。
        String jobId = runSparkJob(dliClient, "YourQueueName");
        // 步骤二：如果您想在当前线程等待作业执行结束，可循环检查状态，直到作业结束。
        checkRunning(dliClient, jobId);
        // 步骤三：如果您想根据条件查询一个或多个特定作业，可执行以下方法。
        // 此处仅作样例，除了jobId，您还可指定其它筛选条件。详见 https://console.huaweicloud.com/apiexplorer/#/openapi/DLI/doc?api=ListSparkJobs 表2
```

```
            listSparkJob(dliClient, jobId);

            /*
             * 其它场景：
             * 1. 作业运行期间，如果您想取消作业，可调用接口取消批处理作业。具体操作请参考取消批处理作业。
             * 关键SDK API：com.huaweicloud.sdk.dli.v1.DliClient#cancelSparkJob(CancelSparkJobRequest)，
             * 注：作业状态为“已成功”或者“已失败”的批处理作业无法取消。
             * 2. 如果您想根据jobId查询某个特定作业的详情，可执行以下方法。
             * 关键SDK API：com.huaweicloud.sdk.dli.v1.DliClient#showSparkJob(ShowSparkJobRequest)，
             * 详见ShowSparkJob
             */
        } catch (DLIException e) {
            // 请根据业务实际情况处理异常信息，此处仅作样例。
        }
    }
```

创建 Spark 作业

- **功能介绍：**

  用于执行Spark作业。

- **相关链接：**

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

  创建批处理作业

- **示例代码：**

```
    private static String runSparkJob(DliClient client, String queueName) {
        // 请根据业务实际情况设置相应的参数，此处仅作样例。
        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") // 必选
                .withClassName("YourClassFullName") // 必选
                .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();
    }
```

查询批处理作业状态

- **功能介绍：**

  用于执行Spark作业。

- **相关链接：**

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

  查询批处理作业状态

- **示例代码：**

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

查询批处理作业列表

- **功能介绍：**

  查询批处理作业列表。

- **相关链接：**

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

  查询批处理作业列表

- **示例代码：**

```
  private static void listSparkJob(DliClient client, String jobId) throws DLIException {
      ListSparkJobsResponse resp;
      try {
          resp = client.listSparkJobs(new ListSparkJobsRequest()
              // 您还可用.withXxx()方法指定其它条件来返回满足条件的SparkJobs，此处仅做样例。
              // 详见 https://console.huaweicloud.com/apiexplorer/#/openapi/DLI/doc?api=ListSparkJobs
表2
              .withJobId(jobId)
              .withQueueName("YourQueueName")
              .withStart(1234567L) // 可以指定作业开始时间
              .withEnd(2345678L)); // 可以指定作业结束时间
      } catch (Exception e) {
          throw new DLIException("Failed to list Spark jobs: ", e);
      }
      logger.info(String.format("List SparkJobs : %s", resp.toString()));
      // resp中的响应参数详见：https://console.huaweicloud.com/apiexplorer/#/openapi/DLI/doc?
api=ListSparkJobs 表3和表4
  }
```

5.2 Python SDK（DLI SDK V2）

5.2.1 使用 SDK 提交 SQL 作业

本节操作介绍使用DLI SDK V2提交SQL作业的操作方法。

说明

DLI SDK V2当前处于公测阶段，如需使用请提交工单申请开通。

- 2024年5月起，首次使用DLI的用户可以直接使用DLI SDK V2，无需申请。
- 对于2024年5月之前开通并使用DLI服务的用户，如需使用“DLI SDK V2”功能，必须提交工单申请开通试用权限。

前提条件

- 已参考**Python开发环境配置**配置Python SDK环境。
- 已参考**初始化DLI客户端**完成客户端DLIClient的初始化。

操作前准备

获取AK/SK，项目ID及对应的Region信息。

1. 管理控制台。
2. 单击界面右上角的登录用户名，在下拉列表中单击“我的凭证”。
3. 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”。
4. 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。
5. 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。

样例代码

```
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:
      # 步骤一：创建数据库、表。
      queue_name = 'your_sql_queue_name'
      prepare(dli_client, queue_name)

      # 步骤二：导入数据到表中。
      # 整体实现过程/原理可分为以下3步：
      # 1. 用OBS的API把数据上传到 “obs_path_to_write_tmp_data”。可在OBS中配置生命周期策略，定期删除这些临时数据。
      # 2. 向DLI提交执行Load Data语句，从而把OBS的数据导入到DLI。
      #    Load Data语法详见导入数据。
      # 3. 循环检查作业运行状态，直到作业结束。
      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)

      # 步骤三：提交SQL语句，执行查询并读取结果。
      select_sql = "SELECT * FROM demo_db.demo_tbl"
      job_id = query_data(dli_client, select_sql, queue_name)

      # 步骤三': 如有需要，用户也可以通过作业ID来获取结果。
      query_data_by_jobid(dli_client, job_id)
```

```
        # 分页查询所有作业，用户可以使用该接口查询当前工程下的所有SQL作业信息
        # 关键SDK API：huaweicloudsdkdli.v1.dli_client.DliClient.list_sql_jobs，
        # 详见 ListSqlJobs
        list_sql_jobs(dli_client)

        # 其它场景：
        # 1. 如果用户想取消已经提交的SQL作业，可使用以下接口。
        # 关键SDK API：huaweicloudsdkdli.v1.dli_client.DliClient.cancel_sql_job
        # 详见 CancelSqlJob
        # 注：若作业已经执行结束或失败则无法取消。

        # 2. 如果用户想对SQL语句做语法校验，可使用以下接口。
        # 关键SDK API：huaweicloudsdkdli.v1.dli_client.DliClient.check_sql
        # 详见 CheckSql
        # 注：本接口只能做语法校验，无法做语义校验。请使用Explain语句，提交到DLI执行，进行语义校验。

        # 3. 如果用户想根据jobId获取某个已提交的SQL作业，并查看作业详情，可使用以下接口。
        # 关键SDK API：huaweicloudsdkdli.v1.dli_client.DliClient.show_sql_job_detail
        # 详见 ShowSqlJobDetail
        # 4. 获取作业执行进度信息，如果作业正在执行，可以获取到子作业的信息，如果作业刚开始或者已经结束，则无法获取到子作业信息
        # 关键SDK API：huaweicloudsdkdli.v1.dli_client.DliClient.show_sql_job_progress
        # 详见 ShowSqlJobProgress
    except DliException as e:
        # 请根据业务实际情况处理异常信息，此处仅作样例。
        logger.error(e)
```

创建数据库和表

相关链接：

- 创建数据库语法参考
- 创建表语法参考
- 关键SDK: dli.dli_client.DliClient.execute_sql
- 关键API: huaweicloudsdkdli.v1.dli_client.DliClient.create_sql_job。详见CreateSqlJob。
- 关键API: huaweicloudsdkdli.v1.dli_client.DliClient.show_sql_job_status。详见ShowSqlJobStatus。

示例代码：

```
def prepare(dli_client, queue_name):
    """
    创建数据库、表。
    :param dli.dli_client.DliClient dli_client: DLI Client.
    :param str queue_name: Queue name for the SQL execute.
    :return: dli.job.SqlJob
    """
    # 1. 创建数据库。
    # “default”为内置数据库，不能创建名为“default”的数据库。
    createDbSql = "CREATE DATABASE IF NOT EXISTS demo_db"
    dli_client.execute_sql(sql=createDbSql, queue_name=queue_name)  # 提交SQL作业，并循环检查作业状态直到作业结束

    # 2. 创建表。注：根据实际情况调整表结构和表数据目录，以及OBS存储路径！！！
    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)
```

导入数据

- **相关链接**:
- **导入数据**
- **原生数据类型**
- **复杂数据类型**
- 关键SDK: dli.dli_client.DliClient.execute_sql
- 关键API: huaweicloudsdkdli.v1.dli_client.DliClient.create_sql_job。详见 **CreateSqlJob**。
- 关键API: huaweicloudsdkdli.v1.dli_client.DliClient.show_sql_job_status。详见 **ShowSqlJobStatus**
- **示例代码:**

```
def load_data(dli_client, upload_data_path, queue_name):
  # 1. 写入数据到OBS临时目录。请根据业务实际情况做调整，此处仅作样例。
  # 注：此步骤纯粹为直接调用OBS写数据相关API，与DLI完全解耦，本示例仅提供了一个写Json的实现，即文件在OBS上的保存格式为Json。
  # 用户可依据业务需求自定义实现，比如，用户可将文件保存为csv。
  write_tmp_data(get_schema(), upload_data_path, dli_client.dli_info, 1)

  # 2. 将第一步中写到OBS上的数据导入到DLI。
  # 注：此处的data_type需要根据第一步中的文件类型来确定，本示例中是JSON。如用户使用其它格式，则需要修改成相应的 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)  # 提交SQL作业，并循环检查作业状态直到作业结束
```

查询作业结果

- **相关链接**:

  SELECT查询语句
- 关键SDK：dli.dli_client.DliClient.execute_sql
- 关键SDK：dli.job.SqlJob.get_result. 需要开启结果写作业桶特性，否则会抛出异常。

  可通过**查询作业状态API**响应体中的 result_path 来判断是否已开启作业结果写作业桶特性。

  待作业运行结束后，如果result_path 以 obs:// 开头，则已开启作业结果写作业桶特性，否则未开启。
- 关键API: huaweicloudsdkdli.v1.dli_client.DliClient.create_sql_job。详见 **CreateSqlJob**。
- 关键API: huaweicloudsdkdli.v1.dli_client.DliClient.show_sql_job_status。详见 **ShowSqlJobStatus**

```
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
```

查询指定作业的结果

- **使用说明**
  - 关键API: **查询作业状态API**
  - 关键SDK：dli.job.SqlJob.get_result.

    使用本方法的前提是需要开启结果写作业桶特性，否则作业运行时会抛出异常。

    可通过**查询作业状态API**响应体中的 result_path 来判断是否已开启作业结果写作业桶特性。待作业运行结束后，如果 result_path 以 obs:// 开头，则已开启作业结果写作业桶特性，否则未开启。
- **示例代码**

```
def query_data_by_jobid(dli_client, job_id):
    """
    :param dli.dli_client.DliClient dli_client: DLI Client.
    :param str job_id: Query类型作业的job id
    :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())
```

查询作业

- **使用说明**

  查询当前工程下的SQL作业信息。

  如果作业较多，必须使用本示例中的分页查询的方式分批查询，否则只能返回第一页的内容，无法返回全部作业。

  关键SDK: huaweicloudsdkdli.v1.dli_client.DliClient.list_sql_jobs
- **示例代码**

```
def list_sql_jobs(client):
    """
    查询当前工程下的SQL作业信息。如果作业较多，必须使用本示例中的分页查询的方式分批查询，否则只能返回第一页的内容，无法返回全部作业。
    关键SDK: huaweicloudsdkdli.v1.dli_client.DliClient.list_sql_jobs

    :param dli.dli_client.DliClient client: DLI Client.
    """
    req = ListSqlJobsRequest()
    req.current_page = 1  # 默认值 1
    req.page_size = 100  # 默认值 10

    # 获取作业总数
    job_count = client.inner_client.list_sql_jobs(req).job_count
    cur = 0

    # 分页查询
    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:
```

```
        # 在此处添加业务逻辑，对每个作业进行处理
        print(job)

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

打印作业结果

- **使用说明**

  请根据业务实际情况处理相应的每一行数据，此处仅作样例。

- **示例代码**

```
def print_result(obs_reader):
    """
    请根据业务实际情况处理相应的每一行数据，此处仅作样例。
    """
    count = 0
    for record in obs_reader:
        count += 1
        print(record)
    logger.info("total records: %d", count)
```

写入数据到 OBS writeTmpData

- **使用说明**

  此方法纯粹为直接调用OBS写数据相关API，与DLI API完全解耦，本示例提供了一个写Json的实现，即文件在OBS上的保存格式为Json。

  用户可依据业务需求自定义实现，比如，用户可将文件保存为csv。

- **示例代码**

```
def write_tmp_data(schema, upload_data_path, dli_info, total_records):
    """
    此方法纯粹为直接调用OBS写数据相关API，与DLI API完全解耦，本示例提供了一个写Json的实现，即文件在OBS上的保存格式为Json。
    用户可依据业务需求自定义实现，比如，用户可将文件保存为csv。

    :param list schema: 数据的结构 schema
    :param str upload_data_path: 该OBS临时目录用于保存用户的业务数据
    :param dli.dli_client.DliClient.dli_info dli_info: 初始化DliClient时传入的认证信息。
    :param int total_records: 模拟数据行数。通过该参数配置插入的数据行数。此处仅做展示，用户可根据实际业务需求修改。
    :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()
```

构建表的 Schema

- **使用说明**

  请根据业务实际情况构造相应的Schema，此处仅作样例。

- **示例代码**

```
def get_schema():
    """
    请根据业务实际情况构造相应的Schema，此处仅作样例。
    """
    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")]
```

按需生成测试数据 List<Object> genRecord

- **使用说明**

  请根据业务实际情况构造相应的每一行数据，此处仅作样例。

- **示例代码**

```
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,毫秒
        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

- **使用说明**

  请根据业务实际情况构造相应的每一行数据，此处仅作样例。

- **示例代码**

```
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

- **使用说明**

  请根据业务实际情况构造相应的每一行数据，此处仅作样例。

- **示例代码**

```
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

- **使用说明**

  请根据业务实际情况构造相应的每一行数据，此处仅作样例。

- **示例代码**

```
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 使用 SDK 提交 Flink SQL 作业

本节操作介绍使用DLI SDK V2提交Flink SQL作业的操作方法。

说明

DLI SDK V2当前处于公测阶段，如需使用请提交工单申请开通。

- 2024年5月起，首次使用DLI的用户可以直接使用DLI SDK V2，无需申请。
- 对于2024年5月之前开通并使用DLI服务的用户，如需使用“DLI SDK V2”功能，必须提交工单申请开通试用权限。

前提条件

- 已参考**Python开发环境配置**配置Python SDK环境。
- 已参考**初始化DLI客户端**完成客户端DLIClient的初始化。

操作前准备

获取AK/SK，项目ID及对应的Region信息。

1. 管理控制台。
2. 单击界面右上角的登录用户名，在下拉列表中单击“我的凭证”。
3. 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”。

4. 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。
5. 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。

样例代码

```
# 配置日志
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:
        # 步骤一：创建Flink作业。作业状态将变成 “草稿”
        job_id = create_flink_sql_job(dli_client, "your_queue_name")
        logger.info("job_id: %d", job_id)

        # 步骤二：运行作业。作业状态将从 “草稿” 变成 “提交中”
        resps = batch_run_flink_sql_jobs(dli_client, [job_id])
        logger.info("Response: %s", resps)

        # 步骤三：查询作业状态。如果您想在当前线程等待作业进入“运行中”状态，可循环检查状态，直到作业进入“运行中”状态。
        check_running(dli_client, job_id)

    except exceptions.ClientRequestException as e:
        # 请根据业务实际情况处理异常信息，此处仅作样例。
        logger.error("Failed to execute job:", e)
```

创建 Flink SQL 作业

- **功能介绍：**

  用于创建Flink SQL作业。

- **相关链接：**

  关键SDK API：huaweicloudsdkdli.v1.dli_client.DliClient.create_flink_sql_job。

  新建Flink SQL作业。

  创建 Flink SQL作业。作业状态将变成 “草稿”

- **示例代码：**

```
def create_flink_sql_job(client, queue_name):
    """
    创建Flink作业。作业状态将变成 “草稿”
    关键SDK API：huaweicloudsdkdli.v1.dli_client.DliClient.create_flink_sql_job。详见 CreateFlinkSqUob

    :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", # 作业名称，名字必须唯一,比如 flink_jar_job_demo。长度限制：1-57个字符。
        desc="your flink job's description", # 用户自定义描述。长度限制：0-512个字符。
        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;""",
        # 自定义 Stream SQL语句，至少包含source, query, sink三个部分。长度限制：1024*1024个字符。
        # 本SQL示例：自动生成随机source数据，并打印到控制台。
        queue_name=queue_name,  # 通用队列名称。队列名称。长度限制：0-128个字符。
        run_mode="exclusive_cluster",  # 作业运行模式。只支持 exclusive_cluster 独享模式。
        log_enabled=True,  # 开启作业的日志上传到用户的OBS功能
        obs_bucket="your_obs_bucket_name",  # OBS桶名。用于保存 日志 和 checkpoint数据
        job_type="flink_opensource_sql_job",  # 作业类型。建议选择: "flink_opensource_sql_job"
        flink_version="1.12"  # 指定Flink版本
    )
    request = CreateFlinkSqlJobRequest(body=request_body)
    response = client.create_flink_sql_job(request)
    return response.job.job_id
```

批量运行 Flink 作业

- **功能介绍：**

  批量运行Flink SQL作业。

- **相关链接：**

  关键SDK API：

  huaweicloudsdkdli.v1.dli_client.DliClient.batch_run_flink_jobs。**批量运行Flink作业**。

  批量运行Flink作业，作业状态将从 “草稿” 变成 “提交中”。

- **示例代码：**

```
def batch_run_flink_sql_jobs(client, job_ids):
    """
    运行作业。作业状态将从 “草稿” 变成 “提交中”
    :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
```

查询作业状态

- **功能介绍：**

  查询Flink SQL作业状态。

- **相关链接：**

  关键SDK API：

  huaweicloudsdkdli.v1.dli_client.DliClient.show_flink_job。**查询Flink作业详情**。

  如果想在当前线程等待作业进入“运行中”状态，可循环检查状态，直到作业进入“运行中”状态。

- **示例代码**：

```
def check_running(client, job_id):
    """
    如果您想在当前线程等待作业进入“运行中”状态，可执行此方法，循环检查状态，直到作业进入“运行中”状态。

    :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 使用 SDK 提交 Flink Jar 作业

本节操作介绍使用DLI SDK V2提交Flink Jar作业的操作方法。

说明

DLI SDK V2当前处于公测阶段，如需使用请提交工单申请开通。

- 2024年5月起，首次使用DLI的用户可以直接使用DLI SDK V2，无需申请。
- 对于2024年5月之前开通并使用DLI服务的用户，如需使用“DLI SDK V2”功能，必须提交工单申请开通试用权限。

前提条件

- 已参考**Python开发环境配置**配置Python SDK环境。
- 已参考**初始化DLI客户端**完成客户端DLIClient的初始化。

操作前准备

获取AK/SK，项目ID及对应的Region信息。

1. 管理控制台。
2. 单击界面右上角的登录用户名，在下拉列表中单击“我的凭证”。
3. 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”。
4. 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。
5. 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。

样例代码

```
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:
    # 步骤一：创建Flink作业。作业状态将变成 “草稿”
    job_id = create_flink_jar_job(dli_client, "your_queue_name")
    logger.info("job_id: %d", job_id)

    # 步骤二：运行作业。作业状态将从 “草稿” 变成 “提交中”
    resps = batch_run_flink_jar_jobs(dli_client, [job_id])
    logger.info("Response: %s", resps)

    # 步骤三：查询作业状态。如果您想在当前线程等待作业进入“运行中”状态，可循环检查状态，直到作业进入“运行中”状态。
    check_running(dli_client, job_id)

except exceptions.ClientRequestException as e:
    # 请根据业务实际情况处理异常信息，此处仅作样例。
    logger.error("Failed to execute job:", e)
```

创建 Flink Jar 作业

- **功能介绍：**

  用于创建Flink Jar作业。

- **相关链接：**

  关键SDK API：

  huaweicloudsdkdli.v1.dli_client.DliClient.create_flink_jar_job

  新建Flink Jar作业。

  创建 Flink Jar作业。作业状态将变成 “草稿”

- **示例代码**：

```
def create_flink_jar_job(client, queue_name):
    """
    创建 Flink jar 作业。作业状态将变成 “草稿”

    :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",  # 作业名称，名字必须唯一,比如 flink_jar_job_demo。长度限制：1-57个字符。
        desc="your flink job's description",  # 用户自定义描述。长度限制：0-512个字符。
        queue_name=***queue_name***,  # 通用队列名称。队列名称。长度限制：0-128个字符。
        feature="basic",  # 作业特性。表示用户作业使用的Flink镜像类型。basic：表示使用DLI提供的基础Flink镜像。
        flink_version="1.12",  # Flink版本。当用户设置“feature”为“basic”时，该参数生效。用户可通过与“feature”参数配合使用，指定作业运行使用的DLI基础Flink镜像的版本
        log_enabled=True,  # 是否开启作业日志。
        obs_bucket="***your_obs_bucket_name***",  # 当“log_enabled”为“true”时, 用户授权保存作业日志的OBS桶名。
        entrypoint="obs://***your_obs_bucket_name***/your/flink/job.jar",  # 用户已上传到OBS的程序包，用户自定义作业主类所在的jar包。
        main_class="***your_class_fullname***"  # 作业入口类,比如:org.apache.flink.examples.WordCount
    )
    request = CreateFlinkJarJobRequest(body=request_body)
    response = client.create_flink_jar_job(request)
    return response.job.job_id
```

批量运行 Flink 作业

- **功能介绍：**

  批量运行Flink SQL作业。

- **相关链接：**

  关键SDK API：

  huaweicloudsdkdli.v1.dli_client.DliClient.batch_run_flink_jobs

  批量运行Flink作业

  批量运行Flink作业，作业状态将从 “草稿” 变成 “提交中”。

- **示例代码：**

```
def batch_run_flink_jar_jobs(client, job_ids):
    """
    运行作业。作业状态将从 “草稿” 变成 “提交中”
    关键SDK API：huaweicloudsdkdli.v1.dli_client.DliClient.batch_run_flink_jobs。详见
BatchRunFlinkJobs

    :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
```

查询作业状态

- **功能介绍：**

  查询Flink SQL作业状态。

- **相关链接：**

  关键SDK API：

  huaweicloudsdkdli.v1.dli_client.DliClient.show_flink_job

  查询Flink作业详情

  如果想在当前线程等待作业进入“运行中”状态，可循环检查状态，直到作业进入“运行中”状态。

- **示例代码：**

```
def check_running(client, job_id):
    """
    如果您想在当前线程等待作业进入“运行中”状态，可执行此方法，循环检查状态，直到作业进入“运
行中”状态。
    关键SDK API：huaweicloudsdkdli.v1.dli_client.DliClient.show_flink_job。详见ShowFlinkJob

    :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 使用 SDK 提交 Spark 作业

本节操作介绍使用DLI SDK V2提交Spark作业的操作方法。

说明

DLI SDK V2当前处于公测阶段，如需使用请提交工单申请开通。

- 2024年5月起，首次使用DLI的用户可以直接使用DLI SDK V2，无需申请。
- 对于2024年5月之前开通并使用DLI服务的用户，如需使用“DLI SDK V2”功能，必须提交工单申请开通试用权限。

前提条件

- 已参考**Python开发环境配置**配置Python SDK环境。
- 已参考**初始化DLI客户端**完成客户端DLIClient的初始化。

操作前准备

获取AK/SK，项目ID及对应的Region信息。

1. 管理控制台。
2. 单击界面右上角的登录用户名，在下拉列表中单击“我的凭证”。
3. 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”。
4. 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。
5. 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。

样例代码

```
# 配置日志
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:
        # 步骤一：提交Spark作业到DLI执行。
        job_id = run_spark_job(dli_client, "your_queue_name")
        # 步骤二：如果您想在当前线程等待作业执行结束，可循环检查状态，直到作业结束。
        check_running(dli_client, job_id)
        # 步骤三：如果您想根据条件查询一个或多个特定作业，可执行以下方法。
        # 此处仅做样例，除了jobId，您还可指定其它筛选条件。详见ListSparkJobs
        list_spark_job(dli_client, job_id)
        # 其它场景：
```

```
    # 1. 作业运行期间，如果您想取消作业。可调用接口CancelSparkJob
    # 关键SDK API: huaweicloudsdkdli.v1.dli_client.DliClient.cancel_spark_job
    # 注：作业状态为“已成功”或者“已失败”的批处理作业无法取消。
    # 2. 如果您想根据jobId查询某个特定作业的详情，可执行以下方法。
    # 关键SDK API：huaweicloudsdkdli.v1.dli_client.DliClient.show_spark_job
    # 详见ShowSparkJob
except exceptions.ClientRequestException as e:
    # 请根据业务实际情况处理异常信息，此处仅作样例。
    logger.error("Failed to execute job: ", e)
```

创建 Spark 作业

- **功能介绍：**

  用于执行Spark作业。

- **相关链接：**

  关键SDK API：huaweicloudsdkdli.v1.dli_client.DliClient.create_spark_job

  详见**创建批处理作业**。

- **示例代码：**

```
def run_spark_job(client, queue_name):
    """
    提交Spark作业到DLI执行。
    关键SDK API：huaweicloudsdkdli.v1.dli_client.DliClient.create_spark_job。详见CreateSparkJob
    :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"  # 比如 "spark.dli.metaAccess.enable": "true"
    }
    request_body = CreateSparkJobRequestBody(
        # 请根据业务实际情况设置相应的参数，此处仅作样例。
        queue=queue_name,  # 填写通用队列的名称
        spark_version="2.4.5",  # 作业使用Spark组件的版本号。
        name="demo_spark_app",  # 用户自定义这个任务的名字，创建时用户指定的批处理名称，不能超过128个字符。
        file="obs://your_bucket/your_spark_app.jar",  # jar包在OBS上的位置。必选
        class_name="your_class_fullname",  # 必选。主类（--class）的类路径，比如"org.example.DliCatalogTest"
        args=["YourAppArg1", "YourAppAgr2", "..."],  # 应用程序参数的传入参数，如不需要请删除此行
        conf=conf_map,  # Spark参数（--conf）
        catalog_name="dli",  # 访问元数据时，需要将该参数配置为dli。且需conf_map["spark.dli.metaAccess.enable"] = "true"
        jars=["YourDepJar1", "YourDepJar2", "..."],  # 依赖的jar包（--jars），如不需要请删除此行
        feature="basic",  # 作业特性,表示用户作业使用的Spark镜像类型。 一般选择basic即可。
        driver_cores=1,  # Spark应用Driver的CPU核数
        driver_memory="1GB",  # Spark应用的Driver内存，参数配置例如2G, 2048M。使用时必须带单位，否则会启动失败。
        num_executors=1,  # Spark应用Executor的个数
        executor_cores=1,  # Spark应用每个Executor的CPU核数
        executor_memory="1GB"  # Spark应用的Executor内存，参数配置例如2G, 2048M。使用时必须带单位，否则会启动失败
    )
    request = CreateSparkJobRequest(body=request_body)
    response = client.create_spark_job(request)
    return response.id
```

查询批处理作业状态

- **功能介绍：**

  用于执行Spark作业。

- **相关链接：**

  关键SDK API：

huaweicloudsdkdli.v1.dli_client.DliClient.show_spark_job_status

查询批处理作业状态。

- **示例代码：**

```
def check_running(client, job_id):
    """
    如果您想在当前线程等待作业执行结束，可循环检查状态，直到作业结束。

    :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)
```

查询批处理作业列表

- **功能介绍：**

  查询批处理作业列表。

- **相关链接：**

  关键SDK API：

  huaweicloudsdkdli.v1.dli_client.DliClient.list_spark_jobs

  查询批处理作业列表

- **示例代码：**

```
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(
            # 此处仅做样例，除了jobId，您还可指定其它筛选条件。
            # 详见ListSparkJobs
            job_id=job_id,
            queue_name="your_queue_name", # 此处需修改成DLI队列名。根据队列查询批作业（推荐使用）。
            start=1716195600000, # 此处需修改成用户自定义作业开始时间。用于查询开始时间在该时间点之后的作业。时间格式为unix时间戳,单位:毫秒。
            end=1716199200000 # 此处需修改成用户自定义作业结束时间。用于查询开始时间在该时间点之前的作业。时间格式为unix时间戳,单位:毫秒。
        )
        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)
    # resp中的响应参数详见ListSparkJobs
```


6 DLI SDK V1（不推荐使用）

6.1 DLI SDK V1 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 6-1 SDK 功能矩阵

语言	功能	内容
Java	使用SDK提交SQL作业	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、查询指定资源包、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。

6.2 DLI SDK V1 与 API 的对应关系

OBS 授权

表 6-2 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 6-3 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 6-4 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 6-5 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 6-6 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 6-7 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 6-8 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

6.3 Java SDK（DLI SDK V1）

6.3.1 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**Java 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 6-9 Java SDK 列表

类型	说明
使用SDK提交SQL作业	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。

类型	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。

6.3.2 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;
  ChargingMode mode = ChargingMode.CHARGING_MODE_CU;
  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());
  }
 }
```

6.3.3 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.cn-north-1.myhuaweicloud.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 ---------");
}
```

6.3.4 SQL 作业相关

6.3.4.1 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());
  }
}
```

6.3.4.2 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());
}
```

6.3.4.3 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**Java SDK的获取与安装**中的操作步骤获取。

6.3.5 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);
}
```

6.3.6 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);

}
```

6.3.7 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);
}
```

6.4 Python SDK（DLI SDK V1）

6.4.1 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**Python 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 6-10 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、查询指定资源包、删除资源包的Python SDK使用说明。
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。

类型	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。

6.4.2 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**。

6.4.3 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.cn-north-1.myhuaweicloud.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
```

6.4.4 SQL 作业相关

6.4.4.1 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**。

6.4.4.2 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
```

6.4.4.3 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
```

6.4.5 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
```

A 修订记录

发布日期	修订说明
2023-12-05	第十五次正式发布。 优化DLI SDK参考手册结构，补充V3版本SDK使用说明。
2023-09-16	第十四次正式发布。 **Java SDK的获取与安装**补充关于SDK安装包中的.sha256文件的下载说明。
2023-07-18	第十三次正式发布。 **表相关**修改DECIMAL默认精度。
2023-03-09	第十二次正式发布。 **Java SDK的获取与安装**新增下载sdk的方式。
2023-01-30	第十一次正式发布。 **Python SDK环境配置**，补充关于安装Python需要配置Visual C++编译环境的相关说明。
2020-4-28	第十次正式发布。 调整章节目录。
2019-8-30	第九次正式发布。 调整章节目录，设置代码高亮。
2019-8-16	第八次正式发布。 将“集群”修改为“队列”。
2019-1-20	第七次正式发布。 Python SDK支持复杂数据类型。
2018-11-15	第六次正式发布。 Java SDK支持复杂数据类型。

发布日期	修订说明
2018-9-26	第五次正式发布。 ● 创建/删除队列 ● 查询队列列表 ● 上传资源包 ● 查询资源列表 ● 执行批处理作业 ● 查询所有批处理任务
2018-7-26	第四次正式发布。 ● 增加Python SDK内容。
2018-04-23	第三次正式发布。 ● 服务更名。
2018-02-24	第二次正式发布。 ● 修改**Java SDK的获取与安装**章节中的下载地址。
2018-02-08	第一次正式发布。