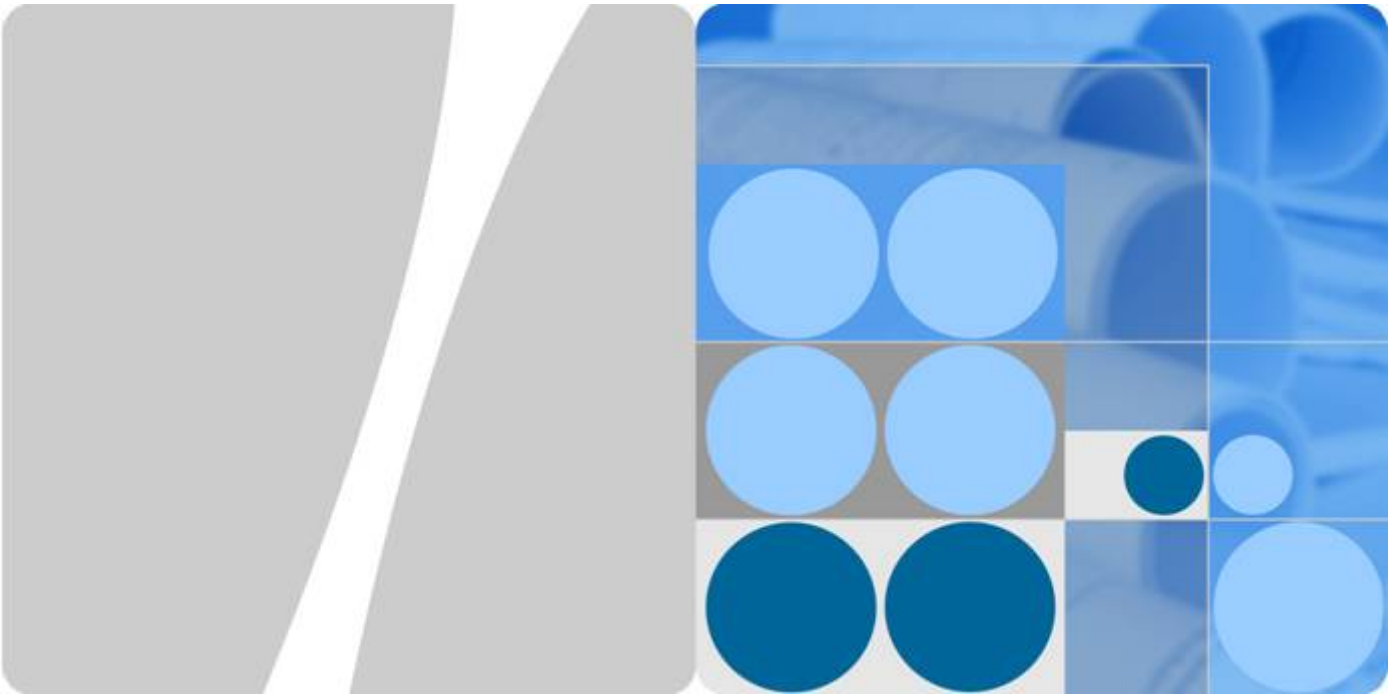


数据湖探索

SDK 参考

发布日期　2021-02-19

目 录

1 DLI SDK 简介

1.1 DLI SDK 能做什么

DLI 概述

数据湖探索（Data Lake Insight，简称DLI）是完全兼容Apache Spark、Apache Flink生态，提供一站式的流处理、批处理、交互式分析的Serverless融合处理分析服务。用户不需要管理任何服务器，即开即用。支持标准SQL/Spark SQL/Flink SQL，支持多种接入方式，并兼容主流数据格式。数据无需复杂的抽取、转换、加载，使用SQL或程序就可以对云上CloudTable、RDS、DWS、CSS、OBS、ECS自建数据库以及线下数据库的异构数据进行探索。

DLI SDK 概述

数据湖探索软件开发工具包（Data Lake Insight Software Development Kit，简称DLI SDK）对DLI提供的REST API进行封装，可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DLI业务能力的目的。

说明

DLI SDK调用接口使用https进行访问，有服务端使用证书。

1.2 内容导航

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。

章节		内容
简介		简要介绍DLI的概念和DLI SDK的概念。
准备环境	前提条件	介绍如何配置DLI SDK运行环境。
	配置Java环境	
	配置Python环境	

章节		内容
DLI SDK与API的对应关	DLI SDK与API的对应关系	当前已开发的DLI SDK，以及其对应的API一览表。
队列相关SDK	Java SDK	介绍队列相关Java SDK。
	Python SDK	介绍队列相关Python SDK。
资源相关SDK	Java SDK	介绍资源相关Java SDK。
	Python SDK	介绍资源相关Python SDK。
SQL作业相关SDK	Java SDK	介绍SQL作业相关Java SDK。
	Python SDK	介绍SQL作业相关Python SDK。
Flink作业相关SDK	Java SDK	介绍Flink作业相关Java SDK。
Spark作业相关SDK	Java SDK	介绍Spark作业相关Java SDK。
	Python SDK	介绍Spark作业相关Python SDK。
Flink作业模板相关SDK	Java SDK	介绍Flink作业模板相关Java SDK。

2 准备环境

2.1 前提条件

开通 DLI 服务

1. 登录云，在上方导航栏选择“产品”。
2. 在“EI企业智能”列表中，选择“数据湖探索”。
3. 单击“进入控制台”，进入数据湖探索控制台页面。

SDK 下载

1. 登录DLI管理控制台。
2. 单击总览页右侧“常用链接”中的“SDK下载”。
3. 在“DLI SDK DOWNLOAD”页面，单击选择所需的SDK链接，即可获取对应的SDK安装包。

   DLI目前提供Java和Python版本的SDK。

   - Java SDK

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

     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。

   - Python SDK

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

表 2-2 目录结构

名称	说明
dli	DLI Python SDK包。
examples	SDK使用样例。
pyDLI	基于pyHive实现的Python连接DLI SQL的连接器。
setup.py	SDK安装脚本。

2.2 配置 Java 环境

操作场景

在进行二次开发时，要准备的开发环境如表2-3所示。

表 2-3 开发环境

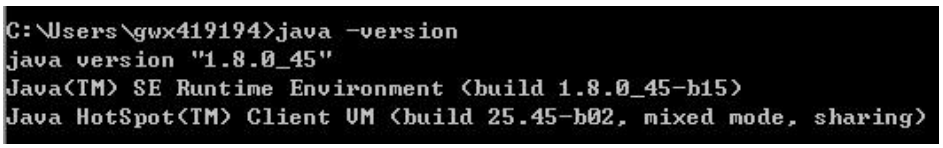
准备项	说明
操作系统	Windows系统，推荐Windows 7及以上版本。
安装JDK	开发环境的基本配置。版本要求：1.7或者1.8。考虑到后续版本的兼容性，强烈推荐使用1.8版本。
安装和配置Eclipse	用于开发DLI应用程序的工具。

操作步骤

步骤1 从Oracle官网下载并安装JDK1.8版本，配置好JAVA环境变量。

1. 安装JDK。
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。

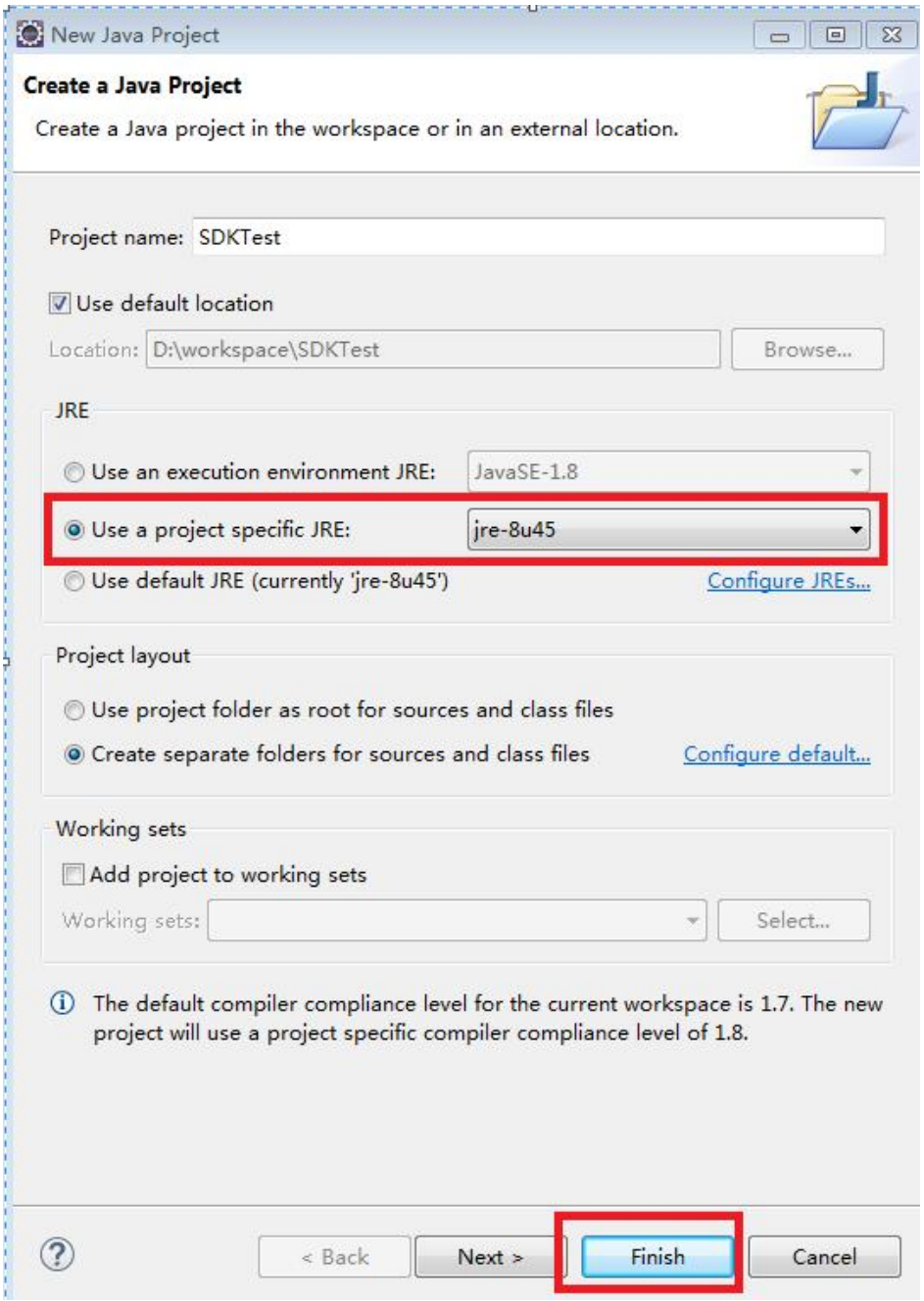
图 2-1 检验配置是否成功



步骤2 从**Eclipse官网**下载并安装Eclipse IDE for Java Developers最新版本。在Eclipse中配置好JDK。

1. 创建新工程，选择JRE版本，请参见**图2-2**

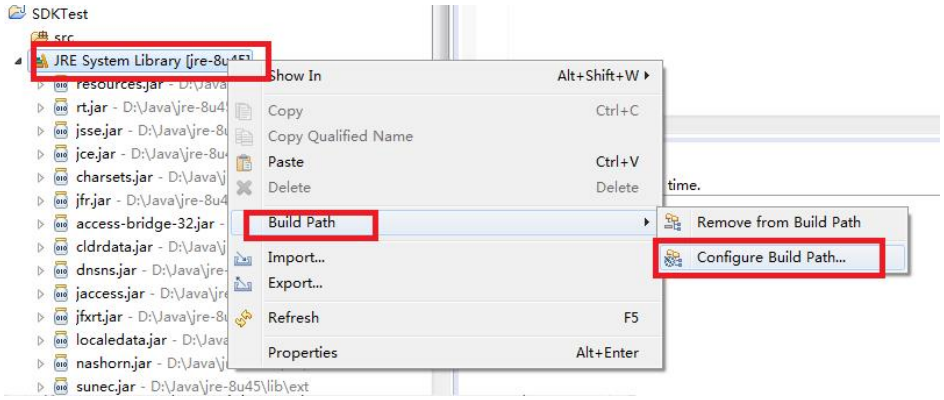
图 2-2 创建新工程



步骤3 配置并导入SDKjar包。

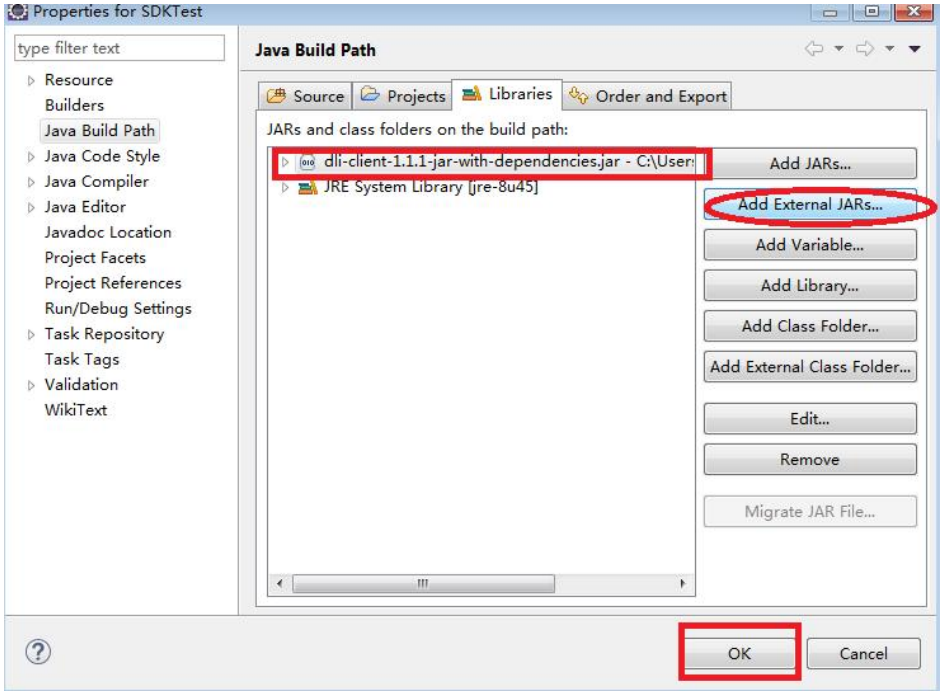
1. 在工程“JRE System Library”上单击右键，选择“Build Path”>“Configure Build Path”，请参见**图2-3**。

图 2-3 配置工程路径



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

图 2-4 选择 SDK jar 包



----结束

2.3 配置 Python 环境

操作场景

在进行二次开发时，要准备的开发环境如表2-4所示。

表 2-4 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。

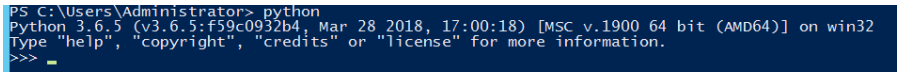
准备项	说明
安装Python	Python版本建议使用2.7.10和3.4.0以上版本。
安装Python依赖库	DLI Python SDK依赖第三方库包括：urllib3 1.15以上版本，six 1.10以上版本，certifi，python-dateutil。

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2-5**，显示版本信息，则说明安装和配置成功。

   图 2-5 检验配置是否成功

步骤2 安装DLI服务Python SDK。

1. 选择**SDK下载**的jar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-1.0.0”。

   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

----结束

3 DLI SDK 与 API 的对应关系

OBS 授权

表 3-1 OBS 授权相关 API&SDK 的对应关系表

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

队列相关

表 3-2 队列相关 API&SDK 的对应关系表

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

资源相关

表 3-3 资源相关 API&SDK 的对应关系表

Class	Method	Java Method	Python Method	API
package Resources	上传资源包	uploadResources	upload_resource	POST /v2.0/{project_id}/resources
	删除资源包	-	-	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	-	-	GET /v2.0/{project_id}/resources/{resource_name}

SQL 作业相关

表 3-4 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	-	-	GET /v1.0/{project_id}/databases/{database_name}/tables/{table_name}
	预览表内容	-	-	GET /v1.0/{project_id}/databases/{database_name}/tables/{table_name}/preview
	修改表用户	-	-	PUT /v1.0/{project_id}/databases/{database_name}/tables/{table_name}/owner
Job	导入数据	submit	import_table	POST /v1.0/{project_id}/jobs/import-table
	导出数据	submit	export_table	POST /v1.0/{project_id}/jobs/export-table
	提交作业	submit	execute_sql	POST /v1.0/{project_id}/jobs/submit-job
	取消作业	cancelJob	-	DELETE /v1.0/{project_id}/jobs/{job_id}
	查询所有作业	listAllJobs	-	GET /v1.0/{project_id}/jobs?page-size=*{size}*&current-page=*{page_number}*&start=*{start_time}*&end=*{end_time}*&job-type={QUERY}&queue_name={test}&order={duration_desc}
	查询作业结果	queryJobResultInfo	-	GET/v1.0/{project_id}/jobs/{job_id}?page-size=*{size}*&current-page=*{page_number}*
	查询作业状态	-	-	GET/v1.0/{project_id}/jobs/{job_id}/status
	查询作业详细信息	-	-	GET/v1.0/{project_id}/jobs/{job_id}/detail

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

Flink 作业相关

表 3-5 Flink 作业相关 API&SDK 的对应关系表

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

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

Spark 作业相关

表 3-6 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	submit	-	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	-	-	GET /v2.0/{project_id}/batches/{batch_id}/state
	查询批处理作业日志	-	-	GET /v2.0/{project_id}/batches/{batch_id}/log

Flink 作业模板相关

表 3-7 Flink 作业模板相关 API&SDK 的对应关系表

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

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

4 Java SDK

4.1 初始化 DLI 客户端

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

AK/SK 认证方式样例代码

```
String ak = "ak";
String sk = "sk";
String regionName = "regionname";
String projectId = "project_id";
DLIInfo dliInfo = new DLIInfo(regionName, ak, sk, projectId);
DLIClient client = new DLIClient(AuthenticationMode.AKSK, dliInfo);
```

说明

通过以下方式可获取AK/SK，项目ID及对应的region信息。

1. 登录管理控制台。
2. 单击页面右上角的用户名，在下拉列表中单击“我的凭证”。
3. 在“我的凭证”页面
   - 单击“管理访问密钥”页签，可以查看已创建的访问密钥。
   - 单击“项目列表”页签，可以查看项目ID。

Token 认证方式样例代码

```
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);
```

说明

可以通过set方式修改endpoint，即dliInfo.setServerEndpoint(endpoint)。

4.2 OBS 授权

样例代码

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

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

4.3 队列相关

创建队列

DLI提供创建队列的接口，您可以使用该接口创建队列。示例代码如下：

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

删除队列

DLI提供删除队列的接口，您可以使用该接口删除队列。示例代码如下：

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

获取默认队列

DLI提供查询默认队列的接口，您可以使用默认队列提交作业。示例代码如下：

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

说明

默认队列允许所有用户使用，DLI会限制用户使用默认队列的次数。

查询所有队列

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

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

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

4.4 资源相关

上传资源包

您可以使用DLI提供的接口上传资源包，示例代码如下：

```
private static void uploadResources(DLIClient client) throws DLIException {
  String kind = "jar";
  String[] paths = new String[1];
  paths[0] = "https://bucketname.obs.com/jarname.jar";
  String description = "test for sdk";
  // 调用DLIClient对象的uploadResources方法上传资源
  List<PackageResource> packageResources = client.uploadResources(kind, paths, description);
  System.out.println("---------- uploadResources success ---------");
}
```

说明

“paths”参数构成为：{bucketName}.{obs域名}/{jarPath}/{jarName}。

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());
  }
}
```

4.5 SQL 作业相关

4.5.1 数据库相关

创建数据库

DLI提供创建数据库的接口。您可以使用该接口创建数据库，示例代码如下：

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

```
    System.out.println("create database:" + database);
    return database;
}
```

说明

“default”为内置数据库，不能创建名为“default”的数据库。

删除数据库

DLI提供删除数据库的接口。您可以使用该接口删除数据库。示例代码如下：

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

说明

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

查询所有数据库

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

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

4.5.2 表相关

创建 DLI 表

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

```
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(38,5)，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(38,5)，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);
}
```

4.5.3 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
- 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();
    String 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如果导入时指定了部分分区信息，而导入的数据不包含非指定的分区信息，则数据导入后的非指定的数据分区列字段会存在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);
}
```

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。

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());
   }
}
```

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);
 }
```

4.6 Flink 作业相关

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。宁可以使用该接口触发运行作业。示例代码如下：

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

停止作业

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

```
  public static void stopFlinkJob(DLIClient client) throws DLIException{
        StopFlinkJobRequest body = new StopFlinkJobRequest();
```

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

批量删除作业

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

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

4.7 Spark 作业相关

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());
  }
}
```

4.8 Flink 作业模板相关

新建作业模板

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

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

更新作业模板

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

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

删除作业模板

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

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

查询作业模板列表

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

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

5 Python SDK

5.1 初始化 DLI 客户端

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

AK/SK 认证方式样例代码

```
def init_aksk_dli_client():
    auth_mode = 'aksk'
    region =
    project_id = 'xxxx'
    ak = 'xxxx'
    sk = 'xxxx'
    dli_client = DliClient(auth_mode=auth_mode, region=region, project_id=project_id,ak=ak, sk=sk)
    return dli_client
```

说明

通过以下方式可获取AK/SK，项目ID及对应的region信息。

1. 登录管理控制台。
2. 单击页面右上角的用户名，在下拉列表中单击“我的凭证”。
3. 在“我的凭证”页面
   - 单击“管理访问密钥”页签，可以查看已创建的访问密钥。
   - 单击“项目列表”页签，可以查看项目ID。

Token 认证方式样例代码

```
def init_token_dli_client():
    auth_mode = 'token'
    region = 'xxxx'
    project_id = 'xxxx'
    account = '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
```

说明

可以通过set方式修改endpoint，即dliInfo.setServerEndpoint(endpoint)。

5.2 队列相关

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)
```

5.3 资源相关

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：

```
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

其中，“obs_jar_paths”为集合。

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)
```

5.4 SQL 作业相关

5.4.1 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。

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。

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)
```

5.4.2 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)
```

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。

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。

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)
```

5.4.3 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)
        result_set = sql_job.get_result()
    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)
```

5.5 Spark 作业相关

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
```