阿里云
大数据计算服务

SDK参考

文档版本：20200709

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止：** 重置操作将丢失用户配置数据。
	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告：** 重启操作将导致业务中断，恢复业务时间约十分钟。
	用于警示信息、补充说明等，是用户必须了解的内容。	 **注意：** 权重设置为0，该服务器不会再接受新请求。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明：** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	单击**设置** > **网络** > **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令。	执行cd /d C:/window命令，进入Windows系统文件夹。
斜体	表示参数、变量。	bae log list --instanceid Instance_ID
[]或者[a|b]	表示可选项，至多选择一个。	ipconfig [-all|-t]
{}或者{a|b}	表示必选项，至多选择一个。	switch {active|stand}

目录

法律声明.....I
通用约定.....I
1 Java SDK.....1
1.1 Java SDK介绍.....1
1.2 Java SDK示例.....6
1.2.1 运行安全命令.....7
1.2.2 Instance Logview.....9
1.2.3 输出错误日志.....10
1.2.4 设置SQL的Flag.....10
1.2.5 SQLTask配合Tunnel实现大量数据导出.....11
2 Python SDK.....14
2.1 Python SDK概述.....14
2.2 Python SDK方法说明.....16

1 Java SDK

1.1 Java SDK介绍

本文从实例、资源、表、函数等几个方面为您介绍Java SDK。

> **说明:**
> 使用SDK调用MaxCompute产生的计算、存储等费用与直接使用MaxCompute产生的费用一致，详情请参见计量计费

背景信息

较为常用的MaxCompute核心接口详情请参见SDK Java Doc。

您可以通过Maven管理配置新SDK的版本，Maven的配置示例如下。

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-core</artifactId>
  <version>xxxx-public</version>
</dependency>
```

> **说明:**
> - 0.27.2-public版本及以上才支持MaxCompute 2.0新数据类型 。
> - 您可以在search.maven.org搜索odps-sdk-core获取最新版本的SDK。

MaxCompute提供的SDK包整体信息，如下表所示。

包名	描述
odps-sdk-core	MaxCompute的基础功能，例如对表、项目的操作，以及Tunnel均在此包中。
odps-sdk-commons	一些Util封装。
odps-sdk-udf	UDF功能的主体接口。
odps-sdk-mapred	MapReduce功能。
odps-sdk-graph	Graph Java SDK，搜索关键词odps-sdk-graph 。

AliyunAccount

阿里云认证账号。输入参数为AccessKey ID及AccessKey Secret，是阿里云用户的身份标识和认证密钥。此类用来初始化MaxCompute。

MaxCompute

MaxCompute SDK的入口，您可通过此类来获取项目空间下的所有对象集合，包括Projects、Tables、Resources、Functions、Instances。

说明：

MaxCompute原名ODPS，因此在现有的SDK版本中，入口类仍命名为ODPS。

您可以通过传入AliyunAccount实例来构造MaxCompute对象。程序示例如下。

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Table t : odps.tables()) {
    ....
}
```

批量数据通道

MaxCompute Tunnel数据通道是基于Tunnel SDK编写的。您可以通过Tunnel向MaxCompute中上传或者下载数据，详细内容请参见批量数据通道。目前Tunnel仅支持表（不包括视图View）和数据的上传和下载。

MapReduce

MapReduce支持的MapReduce SDK请参见#unique_13。

Projects

Projects是MaxCompute中所有项目空间的集合。集合中的元素为项目（Project）。程序示例如下。

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Project p = odps.projects().get("my_exists");
p.reload();
Map<String, String> properties = prj.getProperties();
...
```

Project

Project是对项目信息的描述。您可以通过Projects获取相应的项目。

SQLTask

SQLTask是用于运行、处理SQL任务的接口。您可以通过运行接口直接运行SQL（注意：每次只能提交运行一个SQL语句）。

运行接口返回Instance实例，通过Instance获取SQL的运行状态及运行结果。程序示例如下。

```
import java.util.List;
import com.aliyun.odps.Instance;
import com.aliyun.odps.Odps;
import com.aliyun.odps.OdpsException;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.task.SQLTask;
public class testSql {
      private static final String accessId = "";
   private static final String accessKey = "";
      private static final String endPoint = "http://service.odps.aliyun.com/api";
      private static final String project = "";
      private static final String sql = "select category from iris;";
   public static void
      main(String[] args) {
      Account account = new AliyunAccount(accessId, accessKey);
      Odps odps = new Odps(account);
      odps.setEndpoint(endPoint);
      odps.setDefaultProject(project);
      Instance i;
      try {
         i = SQLTask.run(odps, sql);
         i.waitForSuccess();
         List<Record> records = SQLTask.getResult(i);
         for(Record r:records){
            System.out.println(r.get(0).toString());
         }
      } catch (OdpsException e) {
         e.printStackTrace();
      }
   }
}
```

说明：

如果您想创建表，则需要通过SQLTask接口，而不是Table接口。您需要将#unique_14的语句传入SQLTask。

Instances

Instances是MaxCompute中所有实例（Instance）的集合。集合中的元素为Instance。程序示例如下。

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Instance i : odps.instances()) {
```

```
    ....
}
```

Instance

Instance是对实例信息的描述。您可以通过Instances获取相应的实例。程序示例如下。

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Instance instance= odps.instances().get("instance id");
Date startTime = instance.getStartTime();
Date endTime = instance.getEndTime();
...
    Status instanceStatus = instance.getStatus();
String instanceStatusStr = null;
if (instanceStatus == Status.TERMINATED) {
    instanceStatusStr = TaskStatus.Status.SUCCESS.toString();
    Map<String, TaskStatus> taskStatus = instance.getTaskStatus();
    for (Entry<String, TaskStatus> status : taskStatus.entrySet()) {
        if (status.getValue().getStatus() != TaskStatus.Status.SUCCESS) {
            instanceStatusStr = status.getValue().getStatus().toString();
            break;
        }
    }
} else {
    instanceStatusStr = instanceStatus.toString();
}
...
    TaskSummary summary = instance.getTaskSummary("task name");
String s = summary.getSummaryText();
```

Tables

Tables是MaxCompute中所有表的集合。集合中的元素为Table。程序示例如下。

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Table t : odps.tables()) {
    ....
}
```

Table

Table是对表信息的描述。您可以通过Tables获取相应的表。程序示例如下。

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Table t = odps.tables().get("table name");
t.reload();
Partition part = t.getPartition(new PartitionSpec(tableSpec[1]));
part.reload();
```

```
...
```

您可以通过如下程序获取表分区数据。

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");

Table table = odps.tables().get("tablename");
for(Column c : table.getSchema().getColumns())
{
String name = c.getName();
TypeInfo type = c.getTypeInfo();
 }
```

Resources

Resources是MaxCompute中所有资源的集合。集合中的元素为Resource。程序示例如下。

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Resource r : odps.resources()) {
    ....
}
```

Resource

Resource是对资源信息的描述。您可以通过Resources获取相应的资源。程序示例如下。

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Resource r = odps.resources().get("resource name");
r.reload();
if (r.getType() == Resource.Type.TABLE) {
    TableResource tr = new TableResource(r);
    String tableSource = tr.getSourceTable().getProject() + "."
        + tr.getSourceTable().getName();
    if (tr.getSourceTablePartition() != null) {
        tableSource += " partition(" + tr.getSourceTablePartition().toString()
            + ")";
    }
    ....
}
```

创建文件资源的示例，如下所示。

```
String projectName = "my_porject";
String source = "my_local_file.txt";
File file = new File(source);
InputStream is = new FileInputStream(file);
FileResource resource = new FileResource();
String name = file.getName();
```

```
resource.setName(name);
odps.resources().create(projectName, resource, is);
```

创建表资源的示例，如下所示。

```
TableResource resource = new TableResource(tableName, tablePrj, partitionSpec);
//resource.setName(INVALID_USER_TABLE);
resource.setName("table_resource_name");
odps.resources().update(projectName, resource);
```

Functions

Functions是MaxCompute中所有函数的集合。集合中的元素为Function。程序示例如下。

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Function f : odps.functions()) {
    ....
}
```

Function

Function是对函数信息的描述。您可以通过Functions获取相应的函数。程序示例如下。

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Function f = odps.functions().get("function name");
List<Resource> resources = f.getResources();
```

创建函数的示例，如下所示。

```
String resources = "xxx:xxx";
String classType = "com.aliyun.odps.mapred.open.example.WordCount";
ArrayList<String> resourceList = new ArrayList<String>();
for (String r : resources.split(":")) {
    resourceList.add(r);
}
Function func = new Function();
func.setName(name);
func.setClassType(classType);
func.setResources(resourceList);
odps.functions().create(projectName, func);
```

1.2 Java SDK示例

1.2.1 运行安全命令

本文为您介绍如何在MaxCompkeute客户端上使用Java SDK接口运行安全相关的命令。

前提条件

您需要完成以下操作：

- 准备IntelliJ IDEA开发工具，请参见安装Studio。
- 配置MaxCompute Studio连接MaxCompute项目空间，请参见创建MaxCompute项目连接。
- 在MaxCompute Studio上添加项目依赖。

  SecurityManager类在odps-sdk-core包中，因此在使用时需要执行如下配置。

```
<dependency>
 <groupId>com.aliyun.odps</groupId>
 <artifactId>odps-sdk-core</artifactId>
 <version>0.29.11-oversea-public</version>
</dependency>
```

背景信息

您可以使用如下两种方式运行安全相关命令：

- 通过MaxCompute客户端运行，使用说明请参见安全指南和安全相关语句汇总。

  以下关键字开头的命令为MaxCompute安全相关的操作命令。

```
GRANT/REVOKE ...
SHOW  GRANTS/ACL/PACKAGE/LABEL/ROLE/PRINCIPALS
SHOW  PRIV/PRIVILEGES
LIST/ADD/REOVE  USERS/ROLES/TRUSTEDPROJECTS
DROP/CREATE   ROLE
CLEAR EXPIRED  GRANTS
DESC/DESCRIBE   ROLE/PACKAGE
CREATE/DELETE/DROP  PACKAGE
ADD ... TO  PACKAGE
REMOVE ... FROM  PACKAGE
ALLOW/DISALLOW  PROJECT
INSTALL/UNINSTALL  PACKAGE
LIST/ADD/REMOVE   ACCOUNTPROVIDERS
SET  LABLE  ...
```

- 使用Java SDK接口SecurityManager.runQuery()方式运行，方法说明请参见MaxCompute SDK Java Doc。

  说明：

  MaxCompute安全相关的命令不是SQL命令，不能创建Instance通过SQL Task方式来运行。

操作步骤

1. 创建测试表test_label。

```
CREATE TABLE IF NOT EXISTS test_label(
```

```
 key  STRING,
 value BIGINT
);
```

2. 在MaxCompute Studio上运行如下Java代码执行**set label**命令。MaxCompute Studio使用方法请参见#unique_21。

```
import com.aliyun.odps.Column;
import com.aliyun.odps.Odps;
import com.aliyun.odps.OdpsException;
import com.aliyun.odps.OdpsType;
import com.aliyun.odps.TableSchema;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.security.SecurityManager;
public class test {
  public static void main(String [] args) throws OdpsException {
    try {
      // init odps
      Account account = new AliyunAccount("<your_accessid>", "<your_accesskey>");
      Odps odps = new Odps(account);
      odps.setEndpoint("http://service-corp.odps.aliyun-inc.com/api");
      odps.setDefaultProject("<your_project>");
      // set label 2 to table columns
      SecurityManager securityManager = odps.projects().get().getSecurityManager();
      String res = securityManager.runQuery("SET LABEL 2 TO TABLE test_label(key, value
);", false);
      System.out.println(res);
    } catch (OdpsException e) {
      e.printStackTrace();
    }
  }
}
```

3. 在控制台查看运行结果。

4. 结果验证。

   程序运行完成后，在MaxCompute客户端中运行desc test_label命令，可以看到**set label**命令已经生效。

```
-----+
| Native Columns:
     |
+-----------------------------------------------------------------------------------
-----+
| Field           | Type        | Label | Comment
     |
+-----------------------------------------------------------------------------------
-----+
| key             | string      | 2     |
     |
| value           | bigint      | 2     |
     |
+-----------------------------------------------------------------------------------
-----+
```

   说明：

   更多安全命令请参考#unique_22、#unique_23、#unique_24以及#unique_25。

1.2.2 Instance Logview

本文为您介绍如何使用MaxCompute Java SDK生成Instance Logview链接。Instance Logview可以帮助您快速定位问题。

背景信息

您可以通过Logview查看和Debug提交的MaxCompute作业，详情请参见#unique_27。

MaxCompute Java SDK提供Logview接口，详情请参见SDK Java Doc。

接口说明

```
Instance i = odps.instances().get("<instance_id>");
String logviewUrl = odps.logview().generateLogView(i, 7 * 24);
```

参数说明：

- **Instance**：要查看的Instance名称。
- **Logview token**：超时时间，单位为小时。例如**7*24**。

说明：

您也可以在MaxCompute客户端中使用**wait< instance_id>**命令获得Logview链接。

1.2.3 输出错误日志

本文为您介绍如何使用MaxCompute Java SDK输出错误日志。

接口说明

MaxCompute Java SDK提供了抽象类RetryLogger，详情请参见SDK Java Doc。

```
public static abstract class RetryLogger {
/**
 * 当RestClent发生重试前的回调函数
 * @param e
 *     错误异常
 * @param retryCount
 *     重试计数
 * @param retrySleepTime
 *     下次需要的重试时间
 */
public abstract void onRetryLog(Throwable e, long retryCount, long retrySleepTime);
}
```

您只需实现一个自己的RetryLogger子类，再在初始化ODPS对象时使用`odps.getRestClient().setRetryLogger(new UserRetryLogger());`进行日志输出。

示例

```
// init odps
odps.getRestClient().setRetryLogger(new UserRetryLogger());
// your retry logger
public class UserRetryLogger extends RetryLogger {
 @Override
 public void onRetryLog(Throwable e, long retryCount, long sleepTime) {
   if (e != null && e instanceof OdpsException) {
     String requestId = ((OdpsException) e).getRequestId();
           if (requestId != null) {
               System.err.println(String.format(
           "Warning: ODPS request failed, requestID:%s, retryCount:%d, will retry in %d
seconds.", requestId, retryCount, sleepTime));
       return;
     }
   }
   System.err.println(String.format(
       "Warning: ODPS request failed:%s, retryCount:%d, will retry in %d seconds.", e.
getMessage(), retryCount, sleepTime));
 }
}
```

1.2.4 设置SQL的Flag

本文为您介绍如何使用MaxCompute Java SDK设置SQL的Flag。

使用DataWorks或MaxCompute客户端提交SQL时，通常需要设置SQL的Flag。例如，Session级别使用MaxCompute新数据类型时，需要在涉及新数据类型的SQL前加设置Flag的语句`set odps.sql.type.system.odps2=true;`。

使用Java SDK提交SQL时，不能简单地把Set Flag语句直接放到SQL查询中执行。设置Flag的正确方式如下。

```
String sql ="SELECT...";
HashMap<String, String> hints = new LinkedHashMap<String, String>();
hints.put("SQL flag name, e.g. odps.sql.type.system.odps2", "SQL flag value");
hints.put("SQL flag name, e.g. odps.sql.type.system.odps2", "SQL flag value");
hints.put("SQL flag name, e.g. odps.sql.type.system.odps2", "SQL flag value");
Instance i = SQLTask.run(odps, odps.getDefaultProject(), sql, hints, null);
i.waitForSuccess();
```

1.2.5 SQLTask配合Tunnel实现大量数据导出

本文为您介绍如何通过使用SQLTask配合Tunnel实现大量数据导出。

SQLTask

SQLTask是MaxCompute SQL的SDK接口。借助SQLTask，您可以方便地运行SQL并获得其返回结果。关于SQLTask的详细描述，请参见Java SDK介绍。

SQLTask.getResult(i)返回的是一个list，您可以循环迭代这个list以获得完整的SQL计算返回结果。但是，使用此方法时select语句返回至客户端的数据条数最大值为1万（参见#unique_31中READ_TABLE_MAX_ROW的取值范围）。即如果在客户端上（包括使用SQLTask SDK）直接select，相当于查询结果上最后加上了limit 10000（如果使用**CREATE TABLE XX AS SELECT**或用**INSERT INTO/OVERWRITE TABLE**把结果固化到具体的表里，则无这项限制）。

Tunnel

如果需要导出的查询结果是某张表或具体某个分区的全部内容，可以使用Tunnel命令行共计完成。MaxCompute提供了Tunnel命令行工具和Tunnel SDK，详情请参见#unique_32和#unique_12。

使用Tunnel命令行工具导出数据示例

```
>tunnel d wc_out c:\wc_out.dat;
2017-12-16 19:32:08 - new session: 201712161932082dxxxxx    total lines: 3
2017-12-16 19:32:08 - file [0]: [0, 3), c:\wc_out.dat
downloading 3 records into 1 file
2017-12-16 19:32:08 - file [0] start
2017-12-16 19:32:08 - file [0] OK. total: 21 bytes
download OK
```

SQLTask配合Tunnel实现超出1万行的运行结果导出

SQLTask不能处理超过1万条的记录，但是Tunnel可以，两者互补。因此可以基于两者实现数据的导出，代码示例如下。

```
private static final String accessId = "userAccessId";
private static final String accessKey = "userAccessKey";
private static final String endPoint = "http://service.cn-shanghai.maxcompute.aliyun.
com/api";
```

```
private static final String project = "userProject";
private static final String sql = "userSQL";
private static final String table = "Tmp_" + UUID.randomUUID().toString().replace("-", "_
");//此处使用随机字符串作为临时导出存放数据的表的名字。
private static final Odps odps = getOdps();
public static void main(String[] args) {
    System.out.println(table);
    runSql();
    tunnel();
}
/*
     * 下载SQLTask的结果。
     * */
private static void tunnel() {
    TableTunnel tunnel = new TableTunnel(odps);
    try {
        DownloadSession downloadSession = tunnel.createDownloadSession(project, table
);
        System.out.println("Session Status is : "+ downloadSession.getStatus().toString());
        long count = downloadSession.getRecordCount();
        System.out.println("RecordCount is: " + count);
        RecordReader recordReader = downloadSession.openRecordReader(0, count);
        Record record;
        while ((record = recordReader.read()) != null) {
            consumeRecord(record, downloadSession.getSchema());
        }
        recordReader.close();
    } catch (TunnelException e) {
        e.printStackTrace();
    } catch (IOException e1) {
        e1.printStackTrace();
    }
}
/*
     * 保存这条数据。
     * 如果数据量少，可以直接打印结果后拷贝。实际使用场景下也可以使用JAVA.IO写到本地文
件，或在远端服务器上保存数据结果。
     * */
private static void consumeRecord(Record record, TableSchema schema) {
    System.out.println(record.getString("username")+","+record.getBigint("cnt"));
}
/*
     * 运行SQL ,把查询结果保存成临时表，方便后续用Tunnel下载。
     * 保存数据的lifecycle此处设置为1天，如果删除步骤失败，也不会浪费过多存储空间。
     * */
private static void runSql() {
    Instance i;
    StringBuilder sb = new StringBuilder("Create Table ").append(table)
        .append(" lifecycle 1 as ").append(sql);
    try {
        System.out.println(sb.toString());
        i = SQLTask.run(getOdps(), sb.toString());
        i.waitForSuccess();
    } catch (OdpsException e) {
        e.printStackTrace();
    }
}
/*
     * 初始化MaxCompute的连接信息。
     * */
private static Odps getOdps() {
    Account account = new AliyunAccount(accessId, accessKey);
    Odps odps = new Odps(account);
    odps.setEndpoint(endPoint);
```

```
    odps.setDefaultProject(project);
    return odps;
}
```

2 Python SDK

2.1 Python SDK概述

本文为您介绍Python SDK及其常见方法。

背景信息

PyODPS是MaxCompute的Python SDK，提供DataFrame框架和MaxCompute对象的基本操作方法。您可以通过MaxCompute轻松地分析数据。

获取详细PyODPS信息的方式如下：

- 了解PyODPS：PyODPS文档。
- 下载odps-python-sdk：Github。
- 安装PyODPS：PyODPS安装指南。
- 开发PyODPS：PyODPS开发指南。

您也可以通过如下方式参与PyODPS的生态开发：

- 编写PyODPS文档：PyODPS。
- 开发PyODPS代码：代码。
- 技术交流：加入钉钉技术交流群11701793。

初始化入口

在使用PyODPS前，您需要用阿里云账号初始化一个MaxCompute的入口，执行命令如下。

```
from odps import ODPS
odps = ODPS('**your-access-id**', '**your-secret-access-key**', '**your-default-project**',
endpoint='**your-end-point**')
```

参数说明：

- **your-access-id**：账号的AccessKey ID。
- **your-secret-access-key**：账号的AccessKey Secret。
- **your-default-project**：使用的项目空间名称。
- **your-end-point**：MaxCompute服务所在区域的Endpoint。详情请参见#unique_37。

完成上述操作即可对对象（例如，表、资源和函数）进行操作。

方法说明

PyODPS提供MaxCompute对象的基本操作方法，详情如下。

操作类型	方法名称	方法说明
项目空间	**get_project(project_name)**	获取项目空间。
	exist_project(project_name)	判断某个项目空间是否存在。
表	**list_tables()**	列出项目空间下的所有表。
	exist_table(table_name)	判断表是否存在。
	get_table(table_name, project=project_name)	获取指定表。允许跨项目获取表。
	create_table()	创建表。
	read_table()	读取表数据。
	write_table()	写入表数据。
	delete_table()	删除已经存在的表。
表分区	**exist_partition()**	判断分区是否存在。
	get_partition()	获取分区。
	create_partition()	创建分区。
	delete_partition()	删除分区。
SQL	**execute_sql()/run_sql()**	执行SQL语句。
	open_reader()	读取执行结果。
任务实例	**list_instances()**	获取项目空间下的所有Instance。
	exist_instance()	判断Instance是否存在。
	get_instance()	获取Instance。
	stop_instance()	停止Instance。
资源	**create_resource()**	创建资源。
	open_resource()	打开资源。
	get_resource()	获取资源。
	list_resources()	列出所有资源。
	exist_resource()	判断资源是否存在。
	delete_resource()	删除资源。
函数	**create_function()**	创建函数。
	delete_function()	删除函数。
数据上传下载通道	**create_upload_session()**	创建上传数据会话。

操作类型	方法名称	方法说明
	create_download_session()	创建下载数据会话。

2.2 Python SDK方法说明

本文为您介绍Python SDK实现创建、查看和删除表等操作的常用方法。

获取PyODPS项目空间

项目空间是MaxCompute的基本组织单元，详情请参见#unique_39。

项目空间的基本操作如下：

- 获取项目空间：使用MaxCompute入口对象的get_project()方法获取项目空间。DataWorks的PyODPS节点中，将会包含一个全局变量odps或者o，即为MaxCompute入口。您不需要手动定义MaxCompute入口。

  ```
  project = o.get_project('project_name') # 指定项目空间时，获取特定项目。
  project = o.get_project()           # 不指定项目空间时，获取当前项目。
  ```

  该方法需要提供**project_name**，即项目空间名称。
- 验证项目空间是否存在：使用exist_project()方法检验某个项目空间是否存在。

基本表操作

常见基本表操作如下：

- 通过调用入口对象的list_tables()方法可以列出项目空间下的所有表。

  ```
  # 处理每张表。
  for table in odps.list_tables():
  ```
- 通过调用入口对象的exist_table()方法判断表是否存在；通过调用get_table()方法获取表。

  ```
  t = odps.get_table('dual')
  t.schema
  odps.Schema {
    c_int_a             bigint
    c_int_b             bigint
    c_double_a          double
    c_double_b          double
    c_string_a          string
    c_string_b          string
    c_bool_a            boolean
    c_bool_b            boolean
    c_datetime_a        datetime
    c_datetime_b        datetime
  }
  t.lifecycle
  -1
  print(t.creation_time)
  2014-05-15 14:58:43
  ```

```
t.is_virtual_view
False
t.size
1408
t.schema.columns
[<column c_int_a, type bigint>,
 <column c_int_b, type bigint>,
 <column c_double_a, type double>,
 <column c_double_b, type double>,
 <column c_string_a, type string>,
 <column c_string_b, type string>,
 <column c_bool_a, type boolean>,
 <column c_bool_b, type boolean>,
 <column c_datetime_a, type datetime>,
 <column c_datetime_b, type datetime>]
```

创建表的Schema

初始化方法有如下两种：

- 通过表的列以及可选的分区进行初始化。

```
from odps.models import Schema, Column, Partition
columns = [Column(name='num', type='bigint', comment='the column'),
           Column(name='num2', type='double', comment='the column2')]
partitions = [Partition(name='pt', type='string', comment='the partition')]
schema = Schema(columns=columns, partitions=partitions)
schema.columns
[<column num, type bigint>,
 <column num2, type double>,
 <partition pt, type string>]
schema.partitions
[<partition pt, type string>]
schema.names  # 获取非分区字段的字段名。
['num', 'num2']
schema.types  # 获取非分区字段的字段类型。
[bigint, double]
```

- 使用Schema.from_lists()方法。该方法更容易调用，但无法直接设置列和分区的注释。

```
schema = Schema.from_lists(['num', 'num2'], ['bigint', 'double'], ['pt'], ['string'])
schema.columns
[<column num, type bigint>,
 <column num2, type double>,
 <partition pt, type string>]
```

创建表

使用create_table()方法创建表的方式有如下两种：

- 使用表Schema来创建表，方法如下。

```
table = o.create_table('my_new_table', schema)
table = o.create_table('my_new_table', schema, if_not_exists=True)  #只有不存在表时才创建。
```

```
table = o.create_table('my_new_table', schema, lifecycle=7) #设置生命周期。
```

- 采用字段名及字段类型字符串创建表，方法如下。

```
#创建非分区表。
table = o.create_table('my_new_table', 'num bigint, num2 double', if_not_exists=True)
#创建分区表可传入（表字段列表, 分区字段列表）。
table = o.create_table('my_new_table', ('num bigint, num2 double', 'pt string'),
if_not_exists=True)
```

在未经设置的情况下，创建表时，只允许使

用BIGINT、DOUBLE、DECIMAL、STRING、DATETIME、BOOLEAN、MAP和ARRAY类型。如果

您需要支持TINYINT和STRUCT等新类型，可以打开options.sql.use_odps2_extension = True开

关，示例如下。

```
from odps import options
options.sql.use_odps2_extension = True
table = o.create_table('my_new_table', 'cat smallint, content struct<title:varchar(100),
body string>')
```

删除表

使用delete_table()方法删除已经存在的表。

```
o.delete_table('my_table_name', if_exists=True) # 只有表存在时删除。
 t.drop() # Table对象存在的时候可以直接执行drop函数。
```

表分区

- 判断是否为分区表。

```
if table.schema.partitions:
 print('Table %s is partitioned.' % table.name)
```

- 遍历表全部分区。

```
for partition in table.partitions:
    print(partition.name)
for partition in table.iterate_partitions(spec='pt=test'):
# 遍历二级分区。
```

- 判断分区是否存在。

```
table.exist_partition('pt=test,sub=2015')
```

- 获取分区。

```
partition = table.get_partition('pt=test')
 print(partition.creation_time)
2015-11-18 22:22:27
 partition.size
```

```
0
```

- 创建分区。

```
t.create_partition('pt=test', if_not_exists=True) # 指定if_not_exists参数，分区不存在时候才创建。
```

- 删除分区。

```
t.delete_partition('pt=test', if_exists=True) # 指定if_exists参数，分区存在时才删除分区。
partition.drop() # 分区对象存在的时，直接对分区对象调用drop方法删除。
```

PyODPS SQL

PyODPS支持MaxCompute SQL查询，并可以读取执行的结果。

- 执行SQL。

  入口对象的execute_sql('statement')和run_sql('statement')方法可以执行SQL语句，返回值请参见任务实例。

```
odps.execute_sql('select * from dual') # 同步的方式执行，会阻塞直到SQL执行完成。
instance = odps.run_sql('select * from dual') # 异步的方式执行。
instance.wait_for_success() # 阻塞直到完成。
```

- 读取SQL执行结果。

  运行SQL的Instance能够直接执行open_reader操作读取SQL执行结果。读取时会出现以下两种情况：

  - SQL返回了结构化的数据。

```
with o.execute_sql('select * from dual').open_reader() as reader:
    for record in reader:
    # 处理每一个record。
```

  - SQL可能执行了desc命令，这时可以通过reader.raw取到原始的SQL执行结果。

```
with o.execute_sql('desc dual').open_reader() as reader:
print(reader.raw)
```

PyODPS资源

MaxCompute中的资源常用在UDF和MapReduce中。基本操作如下：

- list_resources()：列出该项目空间下的所有资源。
- exist_resource()：判断资源是否存在。
- delete_resource()：删除资源。您也可以通过Resource对象调用drop方法实现。
- create_resource()：创建资源。
- open_resource()：读取资源。

在PyODPS中，主要支持文件和表两种资源类型。

- 文件资源

  文件资源包括基础的FILE、PY、JAR和ARCHIVE。

  > **说明：**
  > 在DataWorks中，PY格式的文件资源请以FILE形式上传，详情请参见Python UDF文档。

  文件资源常见操作如下：

  - 创建文件资源

    您可以通过给定资源名、文件类型和一个file-like的对象（或字符串对象）调用create_resource()来创建。

    ```
    resource = o.create_resource('test_file_resource', 'file', file_obj=open('/to/path/file')) # 使用file-like的对象创建文件资源。
    resource = o.create_resource('test_py_resource', 'py', file_obj='import this') # 使用字符串创建文件资源。
    ```

  - 读取和修改文件资源

    打开一个资源有如下两种方法：

    ■ 对文件资源调用open方法。

    ■ 在MaxCompute入口调用open_resource()方法。

    打开后的对象是file-like的对象。类似于Python内置的open()方法，文件资源也支持打开的模式，示例如下。

    ```
    with resource.open('r') as fp:  # 以读模式打开资源。
        content = fp.read()  # 读取全部的内容。
        fp.seek(0)  # 回到资源开头。
        lines = fp.readlines()  # 读成多行。
        fp.write('Hello World')  # 报错，读模式下无法写资源。

     with o.open_resource('test_file_resource', mode='r+') as fp:  # 读写模式打开资源。
        fp.read()
        fp.tell()  # 定位当前位置。
        fp.seek(10)
        fp.truncate()  # 截断后面的内容。
        fp.writelines(['Hello\n', 'World\n'])  # 写入多行。
        fp.write('Hello World')
    ```

```
    fp.flush() # 手动调用会将更新提交到ODPS。
```

所有支持的打开类型包括：

- r：读模式，只能打开不能写。
- w：写模式，只能写入而不能读文件，注意用写模式打开，文件内容会先被清空。
- a：追加模式，只能写入内容到文件末尾。
- r+：读写模式，可以任意读写内容。
- w+：类似于r+，但会先清空文件内容。
- a+：类似于r+，但写入时只能写入文件末尾。

同时，PyODPS中文件资源支持以二进制模式打开，例如一些压缩文件需要以这种模式打开。

- rb：指以二进制读模式打开文件。
- r+b：指以二进制读写模式打开。

- 表资源
  - 创建表资源

```
o.create_resource('test_table_resource', 'table', table_name='my_table', partition='
pt=test')
```

  - 更新表资源

```
table_resource = o.get_resource('test_table_resource')
table_resource.update(partition='pt=test2', project_name='my_project2')
```

  - 获取表及分区

```
table_resource = o.get_resource('test_table_resource')
table = table_resource.table
print(table.name)
partition = table_resource.partition
print(partition.spec)
```

  - 读写内容

```
table_resource = o.get_resource('test_table_resource')
with table_resource.open_writer() as writer:
    writer.write([0, 'aaaa'])
    writer.write([1, 'bbbbb'])
    with table_resource.open_reader() as reader:
        for rec in reader:
            print(rec)
```

DataFrame

PyODPS提供了DataFrame API，它提供了类似Pandas的接口，但是能充分利用MaxCompute的计算能力。完整的DataFrame文档请参见DataFrame。

假设已经存在三张表，分别是pyodps_ml_100k_movies（电影相关的数据）、pyodps_ml_100k_users（用户相关的数据）和pyodps_ml_100k_ratings（评分有关的数据）。

1. 首先创建MaxCompute的入口对象。

```
#创建MaxCompute入口对象。
o = ODPS('**your-access-id**', '**your-secret-access-key**',project='**your-project**',
endpoint='**your-end-point**'))
```

2. 传入Table对象，创建DataFrame对象users。

```
from odps.df import DataFrame
users = DataFrame(o.get_table('pyodps_ml_100k_users'))
```

3. 对DataFrame对象可以执行如下操作：

   - 通过dtypes属性可以查看DataFrame的字段和类型，如下所示。

     ```
     users.dtypes
     ```

   - 通过head方法，可以获取前N条数据，方便快速预览数据。

     ```
     users.head(10)
     ```

     返回结果如下。

-	user_id	age	sex	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213
5	6	42	M	executive	98101
6	7	57	M	administrator	91344
7	8	36	M	administrator	05201
8	9	29	M	student	01002

-	user_id	age	sex	occupation	zip_code
9	10	53	M	lawyer	90703

- 对字段进行筛选。
  - 筛选部分字段。

    ```
    users[['user_id', 'age']].head(5)
    ```

    返回结果如下。

-	user_id	age
0	1	24
1	2	53
2	3	23
3	4	24
4	5	33

  - 排除个别字段，如下所示。

    ```
    >>> users.exclude('zip_code', 'age').head(5)
    ```

    返回结果如下。

-	user_id	sex	occupation
0	1	M	technician
1	2	F	other
2	3	M	writer
3	4	M	technician
4	5	F	other

  - 排除掉一些字段的同时，通过计算得到一些新的列。例如，将**sex**为**M**设置为**True**，否则设置为**False**，并将此列取名为**sex_bool**。如下所示。

    ```
    >>> users.select(users.exclude('zip_code', 'sex'), sex_bool=users.sex == 'M').head(5)
    ```

    返回结果如下。

-	user_id	age	occupation	sex_bool
0	1	24	technician	True
1	2	53	other	False

-	user_id	age	occupation	sex_bool
2	3	23	writer	True
3	4	24	technician	True
4	5	33	other	False

- 查询年龄在20~25岁之间的人数，如下所示。

```
>>> users.age.between(20, 25).count().rename('count')
943
```

- 查询男女用户的数量。

```
>>> users.groupby(users.sex).count()
```

返回结果如下。

-	sex	count
0	F	273
1	M	670

- 将用户按职业划分，从高到底，获取人数最多的前10个职业。

```
>>> df = users.groupby('occupation').agg(count=users['occupation'].count())
>>> df.sort(df['count'], ascending=False)[:10]
```

返回结果如下。

-	occupation	count
0	student	196
1	other	105
2	educator	95
3	administrator	79
4	engineer	67
5	programmer	66
6	librarian	51
7	writer	45
8	executive	32

-	occupation	count
9	scientist	31

DataFrame API提供了value_counts方法来快速达到同样的目的。

```
>>> users.occupation.value_counts()[:10]
```

返回结果如下。

-	occupation	count
0	student	196
1	other	105
2	educator	95
3	administrator	79
4	engineer	67
5	programmer	66
6	librarian	51
7	writer	45
8	executive	32

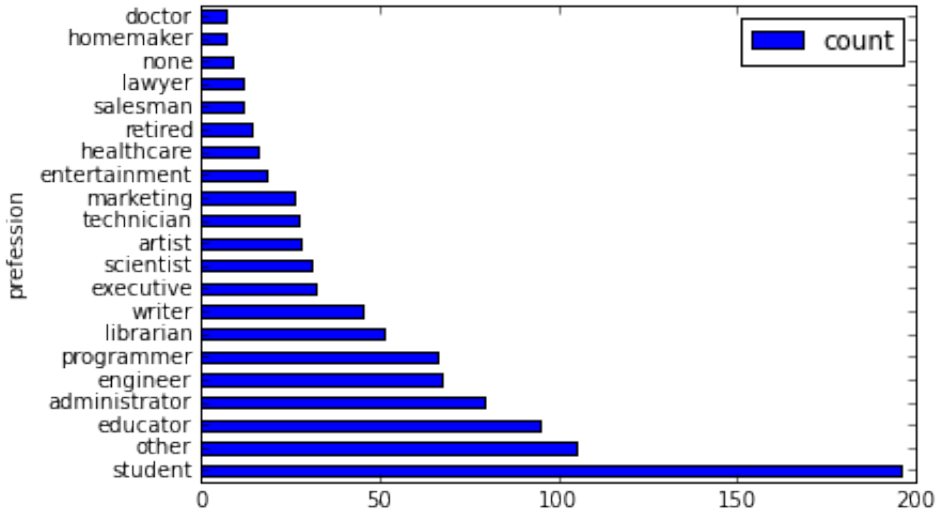
-	occupation	count
9	scientist	31

- 使用更直观的图来查看这份数据。

```
%matplotlib inline
```

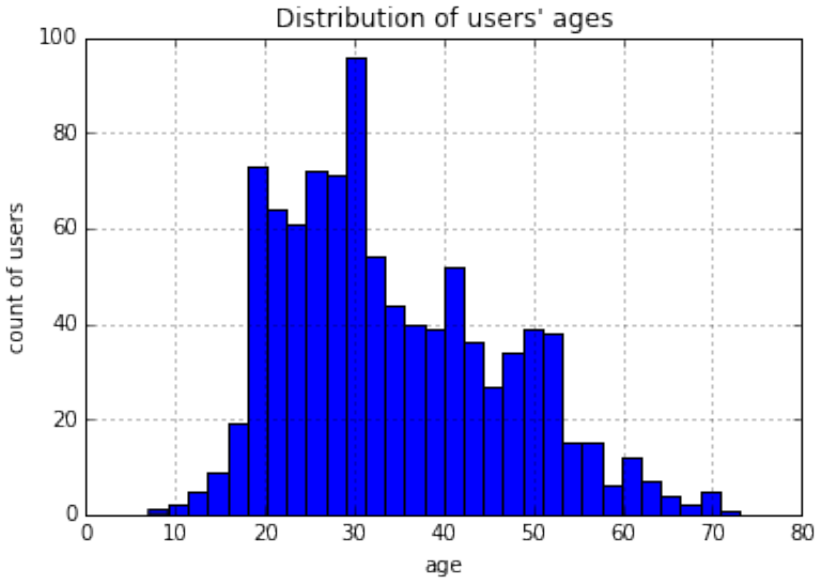
- 使用横向的柱状图来可视化。

```
users['occupation'].value_counts().plot(kind='barh', x='occupation', ylabel='
prefession')
```



- 使用直方图来可视化。将年龄分成30组，查看各年龄分布的直方图，如下所示。

```
>>> users.age.hist(bins=30, title="Distribution of users' ages", xlabel='age', ylabel='
count of users')
```



- 使用JOIN将三张表进行联合后，保存成一张新的表。

```
movies = DataFrame(o.get_table('pyodps_ml_100k_movies'))
ratings = DataFrame(o.get_table('pyodps_ml_100k_ratings'))
o.delete_table('pyodps_ml_100k_lens', if_exists=True)
lens = movies.join(ratings).join(users).persist('pyodps_ml_100k_lens')
lens.dtypes
```

结果如下。

```
odps.Schema {
  movie_id                int64
  title                   string
  release_date            string
  video_release_date      string
  imdb_url                string
  user_id                 int64
  rating                  int64
  unix_timestamp          int64
  age                     int64
  sex                     string
  occupation              string
  zip_code                string
}
```

- 把0~80岁的年龄，分成8个年龄段。

```
labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79']
cut_lens = lens[lens, lens.age.cut(range(0, 81, 10), right=False, labels=labels).
rename('年龄分组')]
```

- 取分组和年龄唯一的前10条数据来进行查看。

```
>>> cut_lens['年龄分组', 'age'].distinct()[:10]
```

结果如下。

-	年龄分组	age
0	0-9	7
1	10-19	10
2	10-19	11
3	10-19	13
4	10-19	14
5	10-19	15
6	10-19	16
7	10-19	17
8	10-19	18

-	年龄分组	age
9	10-19	19

- 对各个年龄分组下，用户的评分总数和评分均值进行查看，如下所示。

```
cut_lens.groupby('年龄分组').agg(cut_lens.rating.count().rename('评分总数'),
cut_lens.rating.mean().rename('评分均值'))
```

结果如下。

-	年龄分组	评分均值	评分总数
0	0-9	3.767442	43
1	10-19	3.486126	8181
2	20-29	3.467333	39535
3	30-39	3.554444	25696
4	40-49	3.591772	15021
5	50-59	3.635800	8704
6	60-69	3.648875	2623
7	70-79	3.649746	197

Configuration

PyODPS提供了一系列的配置选项，可通过odps.options命令获得。可配置的MaxCompute选项，如下所示。

- 通用配置

选项	说明	默认值
end_point	MaxCompute Endpoint	None
default_project	默认项目空间	None
log_view_host	Logview主机名	None
log_view_hours	Logview保持时间（小时）	24
local_timezone	使用的时区。True表示本地时间，False表示UTC，也可用pytz时区	1
lifecycle	所有表生命周期	None
temp_lifecycle	临时表生命周期	1
biz_id	用户ID	None

选项	说明	默认值
verbose	是否打印日志	False
verbose_log	日志接收器	None
chunk_size	写入缓冲区大小	1496
retry_times	请求重试次数	4
pool_connections	缓存在连接池的连接数	10
pool_maxsize	连接池最大容量	10
connect_timeout	连接超时	5
read_timeout	读取超时	120
completion_size	对象补全列举条数限制	10
notebook_repr_widget	使用交互式图表	True
sql.settings	MaxCompute SQL运行全局hints	None
sql.use_odps2_extension	启用MaxCompute 2.0语言扩展	False

- 数据上传或下载配置

选项	说明	默认值
tunnel.endpoint	Tunnel Endpoint	None
tunnel.use_instance_tunnel	使用Instance Tunnel获取执行结果	True
tunnel.limited_instance_tunnel	限制Instance Tunnel获取结果的条数	True
tunnel.string_as_binary	在STRING类型中使用Bytes而非Unicode	False

- DataFrame配置

选项	说明	默认值
interactive	是否在交互式环境	根据检测值
df.analyze	是否启用非MaxCompute内置函数	True
df.optimize	是否开启DataFrame全部优化	True
df.optimizes.pp	是否开启DataFrame谓词下推优化	True

选项	说明	默认值
df.optimizes.cp	是否开启DataFrame列剪裁优化	True
df.optimizes.tunnel	是否开启DataFrame使用Tunnel优化执行	True
df.quote	MaxCompute SQL后端是否用``来标记字段和表名	True
df.libraries	DataFrame运行使用的第三方库（资源名）	None

- PyODPS ML配置

选项	说明	默认值
ml.xflow_project	默认Xflow工程名	algo_public
ml.use_model_transfer	是否使用ModelTransfer获取模型PMML	True
ml.model_volume	在使用ModelTransfer时使用的Volume名称	pyodps_volume