

ALIBABA CLOUD

阿里云

专有云企业版

大数据计算服务
开发指南

产品版本：v3.16.2

文档版本：20220915

 阿里云

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. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，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
 危险	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。	 危险 重置操作将丢失用户配置数据。
 警告	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。
 注意	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。
 说明	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	命令或代码。	执行 <code>cd /d C:/window</code> 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	<code>bae log list --instanceid</code> <code>Instance_ID</code>
[] 或者 [a b]	表示可选项，至多选择一个。	<code>ipconfig [-all -t]</code>
{ } 或者 {a b}	表示必选项，至多选择一个。	<code>switch {active stand}</code>

目录

1.Java SDK	06
1.1. Java SDK介绍	06
1.2. 环境准备	06
1.2.1. 概述	06
1.2.2. 获取Accesskey	07
1.2.2.1. 登录Apsara Uni-manager运营控制台	07
1.2.2.2. 获取组织Accesskey	07
1.2.2.3. 获取阿里云AccessKey	08
1.2.3. 获取Endpoint	08
1.3. 核心接口	11
1.3.1. AliyunAccount	11
1.3.2. Odps	11
1.3.3. Projects	11
1.3.4. Project	12
1.3.5. SQLTask	12
1.3.6. Instances	12
1.3.7. Instance	13
1.3.8. Tables	13
1.3.9. Table	14
1.3.10. Resources	14
1.3.11. Resource	14
1.3.12. Functions	15
1.3.13. Function	16
1.3.14. Spark Shell	16
1.3.15. Spark R	16
1.3.16. Spark SQL	17

1.3.17. Spark JDBC	17
1.3.18. Document APIs	18
1.3.19. Search APIs	18
1.3.20. CheckPermissionTest	18
2. Python SDK	19

1.Java SDK

1.1. Java SDK介绍

本章节为您介绍MaxCompute提供的SDK包整体信息，并提供完整接口信息的链接。

[MaxCompute完整接口信息](#)。


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>0.36.2</version>
</dependency>
```

MaxCompute提供的SDK包整体信息，如下表所示。

包名	描述
odps-sdk-core	MaxCompute基础功能。 封装了基础的MaxCompute概念及其编程接口，包括Odps、Project和Table等，Tunnel相关的功能也在此包。
odps-sdk-core-internal	MaxCompute扩展功能。 封装了一些不常用的MaxCompute概念和操作，例如Event和XFlow等。
odps-sdk-commons	MaxCompute基础设施。 包含TableSchema、Column、Record和OdpsType等基础设施以及一些Util的封装。
odps-sdk-udf	MaxCompute UDF编程接口。
odps-sdk-mapred	MaxCompute MapReduce作业编程接口。
odps-sdk-graph	MaxCompute Graph编程接口。

SDK包下载地址为[SDK包下载](#)。

 **说明** 对于MaxCompute提供的SDK包，如果您使用的MaxCompute版本在V3.8.0以下，请下载0.27.2版本的SDK包使用；如果您使用的MaxCompute版本是V3.8.0到V3.11.0，请下载0.30.8或0.30.9版本的SDK包使用；如果您使用的MaxCompute版本是V3.12.0及以上，请下载0.36.2版本的SDK包使用。

1.2. 环境准备

1.2.1. 概述

本章节为您介绍使用SDK前需要进行的环境准备。

- 需要一个授权账号以及一对AccessKey ID和AccessKey Secret。
- 需要获取产品的Endpoint。

1.2.2. 获取Accesskey

1.2.2.1. 登录Apsara Uni-manager运营控制台

本文主要向您介绍如何登录Apsara Uni-manager运营控制台。


前提条件

- 登录Apsara Uni-manager运营控制台前，确认您已从部署人员处获取Apsara Uni-manager运营控制台的服务域名地址。
- 推荐使用Chrome浏览器。

操作步骤

1. 在浏览器地址栏中，输入Apsara Uni-manager运营控制台的服务域名地址，按回车键。
2. 输入正确的用户名及密码。


请向运营管理员获取登录控制台的用户名和密码。

 **说明** 首次登录Apsara Uni-manager运营控制台时，需要修改登录用户名的密码，请按照提示完成密码修改。为提高安全性，密码长度必须为8~20位，且至少包含以下两种类型：

- 英文大写或小写字母（A~Z、a~z）
- 阿拉伯数字（0~9）
- 特殊符号（感叹号（!）、at（@）、井号（#）、美元符号（\$）、百分号（%）等）

3. 单击登录。
4. 如果账号已激活MFA多因素认证，请根据以下两种情况进行操作：
 - 管理员强制开启MFA后的首次登录：
 - a. 在绑定虚拟MFA设备页面中，按页面提示步骤绑定MFA设备。
 - b. 按照步骤2重新输入账号和密码，单击登录。
 - c. 输入6位MFA码后单击**认证**。
 - 您已开启并绑定MFA：

输入6位MFA码后单击**认证**。

 **说明** 绑定并开启MFA的操作请参见Apsara Uni-manager运营控制台用户指南中的章节**绑定并开启虚拟MFA设备**。


1.2.2.2. 获取组织Accesskey


本章节为您介绍如何获取组织AccessKey。

前提条件

只有运营管理员和一级组织管理员可以获取组织AccessKey。

操作步骤

1. 管理员登录Apsara Uni-manager运营控制台。
2. 在页面顶部菜单栏上，单击**企业**。
3. 在**企业**页面的左侧导航栏中，单击**组织管理**。
4. 在组织结构中，单击要添加的上级组织后面的.
5. 在弹出的下拉菜单中，选择**获取 accesskey**。
6. 在弹出的对话框中，查看组织Accesskey信息。


 **说明** 一级组织的AccessKey为系统自动分配，下级组织使用一级组织的AccessKey。


1.2.2.3. 获取阿里云AccessKey

为了保障云资源的安全性，系统需要验证访问者的身份，以确保访问者具有相关权限。如果您需要访问云资源，您需要获取个人账号的AccessKey ID和AccessKey Secret，用于登录授权。本章节为您介绍如何获取阿里云AccessKey。

操作步骤

1. 管理员登录Apsara Uni-manager运营控制台。
2. 在系统界面右上角，单击当前登录用户的头像，选择**个人信息**。
3. 在**阿里云AccessKey**模块，您可以查看个人账户的AccessKey信息。

阿里云AccessKey 访问阿里云资源时，请使用此AccessKey		
区域	AccessKey ID 	AccessKey Secret 
cn-qingdao-sg-d01	xxxxxx	

 **说明** Accesskey ID和AccessKey Secret是您访问云资源时的密钥，具有该账号完整的权限，请您妥善保管。

1.2.3. 获取Endpoint

本章节为您介绍如何通过Apsara Uni-manager运维控制台获取产品的Endpoint。

背景信息

- 已从部署人员或管理员处获取Apsara Uni-manager运维控制台的访问地址、用户名和密码。
Apsara Uni-manager运维控制台访问地址格式为 *ops.asconsole.intranet-domain-id.com*。
- 推荐使用Chrome浏览器。

操作步骤

1. 打开浏览器，地址栏中输入Apsara Uni-manager运维控制台的访问地址 *ops.asconsole.intranet-domain-id.com*，按回车键。



 **说明** 您可以单击页面右上角的下拉按钮来进行语言切换。

2. 输入正确的用户名及密码。

 **说明** 登录Apsara Uni-manager运维控制台的用户名和密码请从部署人员或管理员处获取。

首次登录Apsara Uni-manager运维控制台时，需要修改登录用户名的密码，请按照提示完成密码修改。
为提高安全性，密码必须满足以下要求：

- 英文大小写字母
- 阿拉伯数字（0~9）
- 特殊符号，包括感叹号（！）、at（@）、井号（#）、美元符号（\$）、百分号（%）等
- 密码长度10~20个字符

- 单击登录，进入Apsara Uni-manager运维控制台。
- 在Apsara Uni-manager运维控制台的顶部菜单栏单击**运维**。在左侧导航栏单击**产品运维管理 > 产品列表**，在云平台运维区域单击**飞天基础运维平台**，直接打开并进入飞天基础运维平台首页。
- 在飞天基础运维平台页面左侧导航栏中，单击**运维 > 机器运维**，进入**机器运维**页面。
- 在机器列表中，通过筛选，选择任意一台机器，单击**Terminal**，登录到机器。

运维 / 机器运维

机器列表

Project 全部 集群 请输入集群名称 机器 请输入主机名或IP模糊查询, 支持批量粘贴 批量Terminal

<input type="checkbox"/>	主机名	集群	Project	Region	状态	详情	机器监控	操作
<input type="checkbox"/>		amtest27	network	cn-qd-hyq-d01	正常	详情	查看	运维 Terminal 机器操作
<input type="checkbox"/>		amtest27	network	cn-qd-hyq-d01	正常	详情	查看	运维 Terminal 机器操作
<input type="checkbox"/>		amtest27	network	cn-qd-hyq-d01	正常	详情	查看	运维 Terminal 机器操作
<input type="checkbox"/>		amtest27	network	cn-qd-hyq-d01	正常	详情	查看	运维 Terminal 机器操作
<input type="checkbox"/>		amtest27	network	cn-qd-hyq-d01	正常	详情	查看	运维 Terminal 机器操作
<input type="checkbox"/>		amtest27	network	cn-qd-hyq-d01	正常	详情	查看	运维 Terminal 机器操作
<input type="checkbox"/>		OssHybridCluster-A-20190827-5b04	oss	cn-qd-hyq-d01	正常	详情	查看	运维 Terminal 机器操作
<input type="checkbox"/>		OssHybridCluster-A-20190827-5b04	oss	cn-qd-hyq-d01	正常	详情	查看	运维 Terminal 机器操作
<input type="checkbox"/>		OssHybridCluster-A-20190827-5b04	oss	cn-qd-hyq-d01	正常	详情	查看	运维 Terminal 机器操作
<input type="checkbox"/>		OssHybridCluster-A-						

7. 在机器中执行如下命令，获取JSON数据。

```
curl -l "http://127.0.0.1:7070/api/v3/column/c.sr.service_registration,c.sr.id?c.sr.service_registration"
```

```
[admin@ ~]$ curl -l http://127.0.0.1:7070/api/v3/column/c.sr.service_registration,c.sr.id?c.sr.service_registration | grep frontend
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload  Total   Spent    Left     Speed
100 3973    0 3973    0    0  122k      0  --:--:-- --:--:-- --:--:-- 122k    "c.sr.id": "datahub-frontend",
  "c.sr.service_registration": "{ \"datahub_frontend.endpoint\": \"datahub.cn-qingdao-env12-d01.dh.env12.shuguang.com\", \"datahub_frontend.port\": \"80,443\" }"
  "c.sr.service_registration": "{ \"biggraph_dbhost\": \"biggraph.mysql.minirds.env12.ops.shuguang.com\", \"biggraph_dbname\": \"biggraph\", \"biggraph_dbpasswd\": \"rod2Cfsc8Rueboqd\", \"biggraph_dbport\": \"3122\", \"biggraph_dbuser\": \"biggraph\", \"biggraph_frontend_server.endpoint\": \"biggraph.cn-qingdao-env12-d01.odps.env12.ops.shuguang.com\", \"biggraph_frontend_server.httpsport\": \"443\", \"biggraph_frontend_server.port\": \"80\", \"biggraph_frontend_server_public.endpoint\": \"biggraph.cn-qingdao-env12-d01.odps.env12.shuguang.com\", \"biggraph_frontend_server_public.httpsport\": \"443\", \"biggraph_frontend_server_public.port\": \"80\", \"biggraph_project\": \"biggraph_internal_project\", \"cluster.name\": \"HybridOdpsCluster-A-20181110-d3cc\", \"quota.id\": \"9249\", \"quota.name\": \"biggraph_quota\" }"
  "c.sr.id": "odps-service-frontend",
  "c.sr.service_registration": "{ \"odps_frontend_server.endpoint\": \"service.cn-qingdao-env12-d01.odps.env12.ops.shuguang.com\", \"odps_frontend_server.httpsport\": \"443\", \"odps_frontend_server.port\": \"80\", \"odps_frontend_server.vip\": \"\", \"odps_frontend_server_internet.endpoint\": \"service.cn-qingdao-env12-d01.odps.public.env12.shuguang.com\", \"odps_frontend_server_internet.httpsport\": \"443\", \"odps_frontend_server_internet.ip\": \"\", \"odps_frontend_server_internet.port\": \"80\", \"odps_frontend_server_public.endpoint\": \"service.cn-qingdao-env12-d01.odps.env12.shuguang.com\", \"odps_frontend_server_public.httpsport\": \"443\", \"odps_frontend_server_public.ip\": \"\", \"odps_frontend_server_public.port\": \"80\", \"tunnel_frontend_server.endpoint\": \"\" }
```

8. 根据MaxCompute产品的ID (odps-service-frontend) 和Key值 (odps_frontend_server.endpoint) , 找到MaxCompute的Endpoint。

```
[admin@
$curl -I http:// /api/v3/column/c.sr.service_registration,c.sr.id?c.sr.service_registration | grep f
rontend
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left   Speed
100 3973    0 3973    0 0    122k      0 --:--:-- --:--:-- --:--:-- 122k      "c.sr.id": "datahub-fronten
d",
  "c.sr.service_registration": "{\"datahub_frontend.endpoint\":\"datahub.cn-qingdao-env12-d01.dh.env12.shug
uang.com\", \"datahub_frontend.port\":\"80,443\"}"
  "c.sr.service_registration": "{\"biggraph_dbhost\":\"biggraph.mysql.minirds.env12.ops.shuguang.com\", \"bi
ggraph_dbname\":\"biggraph\", \"biggraph_dbpasswd\":\"rod2Cfsc8Rueboqd\", \"biggraph_dbport\":\"3122\", \"biggraph_d
buser\":\"biggraph\", \"biggraph_frontend_server.endpoint\":\"biggraph.cn-qingdao-env12-d01.odps.env12.ops.shuguan
g.com\", \"biggraph_frontend_server.httpsport\":\"443\", \"biggraph_frontend_server.port\":\"80\", \"biggraph_fron
end_server_public.endpoint\":\"biggraph.cn-qingdao-env12-d01.odps.env12.shuguang.com\", \"biggraph_frontend_ser
ver_public.httpsport\":\"443\", \"biggraph_frontend_server_public.port\":\"80\", \"biggraph_project\":\"biggraph_intern
al_project\", \"cluster.name\":\"HybridOdpsCluster-A-20181110-d3cc\", \"quota.id\":\"9249\", \"quota.name\":\"biggra
ph_quota\"}"
      ID
  "c.sr.id": "odps-service-frontend",
      key
  "c.sr.service_registration": "{\"odps_frontend_server.endpoint\":\"service.cn-qingdao-env12-d01.odps.env1
2.ops.shuguang.com\", \"odps_frontend_server.httpsport\":\"443\", \"odps_frontend_server.port\":\"80\", \"odps_fron
end_server.vip\":\"\", \"odps_frontend_server_internet.endpoint\":\"service.cn-qingdao-env12-d01.odps.p
ublic.env12.shuguang.com\", \"odps_frontend_server_internet.httpsport\":\"443\", \"odps_frontend_server_internet.ip
\":\"\", \"odps_frontend_server_internet.port\":\"80\", \"odps_frontend_server_public.endpoint\":\"serv
ice.cn-qingdao-env12-d01.odps.env12.shuguang.com\", \"odps_frontend_server_public.httpsport\":\"443\", \"odps_fron
end_server_public.ip\":\"\", \"odps_frontend_server_public.port\":\"80\", \"tunnel_frontend_server.endpo
```

1.3. 核心接口

1.3.1. AliyunAccount

本章节为您介绍Java SDK核心接口中的AliyunAccount接口。

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

 说明 accessId对应AccessKey ID，accessKey对应AccessKey Secret。

1.3.2. Odps

本章节为您介绍Java SDK核心接口中的Odps接口。

MaxCompute SDK的入口，您可以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和Instances。

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()) {
    ....
}
```

1.3.3. Projects

本章节为您介绍Java SDK核心接口中的Projects接口。

MaxCompute中所有项目空间的集合，集合中的元素为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();
...
```

1.3.4. Project

本章节为您介绍Java SDK核心接口中的Project接口。

对项目空间信息的描述，可以通过Projects获取相应的项目空间。

1.3.5. SQLTask

本章节为您介绍Java SDK核心接口中的SQLTask接口。

用于运行和处理SQL任务的接口。可以通过Run接口直接运行SQL。Run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。

```
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 = SQLTask.run(odps, "my_project", "select ...");
String id = instance.getId();
instance.waitForSuccess();
Set<String> taskNames = instance.getTaskNames();
for (String name
taskNames) {
TaskSummary summary = instance.getTaskSummary(name);
String s = summary.getSummaryText();
}
Map<String, String> results = instance.getTaskResults();
Map<String, TaskStatus> taskStatus = instance.getTaskStatus();
for (Entry<String, TaskStatus> status : taskStatus.entrySet()) {
String result = results.get(status.getKey());
}
}
```

② 说明 如果您想创建表，需要通过SQLTask接口，而不是Table接口。您需要将创建表（CREATE TABLE）的语句传入SQLTask。

1.3.6. Instances

本章节为您介绍Java SDK核心接口中的Instances接口。

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()) {
    ....
}
```

1.3.7. Instance

本章节为您介绍Java SDK核心接口中的Instance接口。

对实例信息的描述，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 ins = 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("instance name");
String s = summary.getSummaryText();
```

1.3.8. Tables

本章节为您介绍Java SDK核心接口中的Tables接口。

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()) {
    ....
}
```

1.3.9. Table

本章节为您介绍Java SDK核心接口中的Table接口。

对表信息的描述，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();
...
```

1.3.10. Resources

本章节为您介绍Java SDK核心接口中的Resources接口。

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()) {
    ....
}
```

1.3.11. Resource

本章节为您介绍Java SDK核心接口中的Resource接口。

对资源信息的描述，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。

```
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。

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

1.3.12. Functions

本章节为您介绍Java SDK核心接口中的Functions接口。

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()) {
....
}
```


1.3.13. Function

本章节为您介绍Java SDK核心接口中的Function接口。

对函数信息的描述，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。

```
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);
```

1.3.14. Spark Shell

本章节为您介绍Java SDK核心接口中的Spark Shell接口。

Spark向集群提交任务时的常用接口之一。

执行如下命令，启动接口应用。

```
$cd $SPARK_HOME
-- 进入Spark目录。
$bin/spark-shell --master yarn
-- 选择运行模式并启动应用。
```

示例如下。

```
sc.parallelize(0 to 100, 2).collect
sql("show tables").show
sql("select * from spark_user_data").show(200,100)
```

1.3.15. Spark R

本章节为您介绍Java SDK核心接口中的Spark R接口。

Spark向集群提交任务时的常用接口之一。

执行如下命令，启动接口应用。

```
$mkdir -p /home/admin/R && unzip ./R/R.zip -d /home/admin/R/  
-- 创建一个目录R并解压该目录下的R.zip。  
$export PATH=/home/admin/R/bin/:$PATH  
-- 设置环境变量。  
$bin/sparkR --master yarn --archives ./R/R.zip  
-- 选择运行模式并启动应用。
```

示例如下。

```
df <- as.DataFrame(faithful)  
df  
head(select(df, df$eruptions))  
head(select(df, "eruptions"))  
head(filter(df, df$waiting < 50))  
results <- sql("FROM spark_user_data SELECT *")  
head(results)
```

1.3.16. Spark SQL

本章节为您介绍Java SDK核心接口中的Spark SQL接口。

Spark向集群提交任务时的常用接口之一。

执行如下命令，启动接口应用。

```
$cd $SPARK_HOME  
-- 进入Spark目录。  
$bin/spark-sql --master yarn  
-- 选择运行模式并启动应用。
```

示例如下。

```
show tables;  
select * from spark_user_data limit 3;  
quit;
```

1.3.17. Spark JDBC

本章节为您介绍Java SDK核心接口中的Spark JDBC接口。

Spark向集群提交任务时的常用接口之一。

执行如下命令，启动接口应用。

```
$sbin/stop-thriftserver.sh
-- 停止线程。
$sbin/start-thriftserver.sh
-- 重启线程。
$bin/beeline
-- 启动应用。
```

示例如下。

```
!connect jdbc:hive2://localhost:10000/odps_smoke_test
show tables;
select * from mr_input limit 3;
!quit
```

1.3.18. Document APIs

本章节为您介绍Java SDK核心接口中的Document APIs接口。

ElasticSearch on MaxCompute兼容的开源API接口集合之一，具体接口信息请参见[Document APIs](#)。

1.3.19. Search APIs

本章节为您介绍Java SDK核心接口中的Search APIs接口。

ElasticSearch on MaxCompute兼容的开源API接口集合之一，具体接口信息请参见[Search APIs](#)。

1.3.20. CheckPermissionTest

本章节为您介绍Java SDK核心接口中的CheckPermissionTest接口。

通过CheckPermissionTest接口，提供上层应用指定用户是否拥有某资源指定权限的查询接口。

程序示例如下。

```
Account account = new AliyunAccount("accessId", "accessKey");
Odps odps = new Odps(account);
SecurityManager sm = odps.projects().get().getSecurityManager();
SecurityManager.PermissionDesc desc = new SecurityManager.PermissionDesc(
    odps.getDefaultProject(),
    CheckPermissionConstants.ObjectType.Table,
    "table_name",
    CheckPermissionConstants.ActionType.Describe);
desc.setColumns(Arrays.asList("col1", "col2"));
desc.setPrincipal("ALIYUN$test@aliyun.com");
SecurityManager.CheckPermissionResultInfo res = sm.checkPermission(desc);
System.out.println("Result: " + res.getResult());
System.out.println("Message: " + res.getMessage());
```

2. Python SDK

本章节为您介绍如何通过Python SDK实现创建、查看及删除表等操作。

PyODPS是MaxCompute的Python版本的SDK，提供简单方便的Python编程接口。PyODPS支持类似Pandas的快速、灵活和富有表现力的数据结构。您可以通过PyODPS提供的DataFrame API使用Pandas的数据结果处理功能。更多详细信息请参见[Github项目](#)和[PyODPS文档](#)。

如果您需要了解更多关于PyODPS的内容，请参见《[大数据计算服务\(MaxCompute\)用户指南](#)》中的PyODPS章节。

下面将简单的介绍如何通过Python SDK实现创建、查看及删除表等操作。

安装PyODPS


PyODPS支持Python2.6以上（包括Python3），系统安装pip后，只需运行 `pip install pyodps`，PyODPS的相关依赖便会自动安装。具体操作请参见《[大数据计算服务\(MaxCompute\)用户指南](#)》中的安装指南章节。

初始化

您需要用阿里云账号初始化一个MaxCompute的入口，示例如下。

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

根据上述操作初始化后，便可以对表、资源及函数等进行操作。

 **说明** 后文中的o对象如无特殊说明，均指的是MaxCompute入口对象。

获取PyODPS项目空间

项目空间是MaxCompute的基本组织单元，类似于Database的概念。

您可通过 `get_project` 命令获取到某个项目空间，示例如下。

```
project = odps.get_project('my_project') # 取到某个项目。
project = odps.get_project()             # 取到默认项目。
```

 **说明**

- 如果不提供参数，则获取到默认项目空间。
- 通过 `exist_project`，可以查看某个项目空间是否存在。
- 表是MaxCompute的数据存储单元。

PyODPS表操作

通过调用 `list_tables` 可以列出项目空间下的所有表，示例如下。

```
for table in odps.list_tables(): # 处理每张表。
```

通过调用 `exist_table` 可以判断表是否存在，通过调用 `get_table` 可以获取表。

```
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PyODPS表的Schema

初始化的方法有两种，如下所示：

- 通过表的列和可选的分区来初始化。

```
from odps.models import Schema, Column, Partition
columns = [Column(name='num', type='bigint', comment='the column')]
partitions = [Partition(name='pt', type='string', comment='the partition')]
schema = Schema(columns=columns, partitions=partitions)
schema.columns
[<column num, type bigint>, <partition pt, type string>]
```

- 通过调用 `Schema.from_lists`。此方法虽然调用更加方便，但是无法直接设置列和分区的注释。

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

创建PyODPS表

您可以使用表的Schema来创建表，示例如下。

```
table = odps.create_table('my_new_table', schema)
table = odps.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。
```

也可以使用英文逗号(,)连接的字段名+字段类型的字符串组合来创建表，示例如下。

```
# 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。

如果需要在支持TINYINT、STRUCT等新类型，可以设置 `options.sql.use_odps2_extension = True;`，以打开这些类型的支持，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>')
```

获取PyODPS表数据

您可通过以下三种方法获取表数据：

- 通过调用 `head` 获取表数据，但仅限于查看每张表开始的小于1万条的数据，示例如下。

```
t = odps.get_table('dual')
for record in t.head(3):
    print(record[0]) # 取第0个位置的值。
    print(record['c_double_a']) # 通过字段取值。
    print(record[0:3]) # 切片操作。
    print(record[0, 2, 3]) # 取多个位置的值。
    print(record['c_int_a', 'c_double_a']) # 通过多个字段取值。
```

- 通过在Table上执行 `open_reader` 操作，打开一个Reader来读取数据。您可以使用With表达式，也可以不使用。

```
# 使用With表达式。
with t.open_reader(partition='pt=test') as reader:
    count = reader.count
    for record in reader[5:10] # 可以执行多次,直到将Count数量的Record读完,这里可以改造成并行操作。
        # 处理一条记录。
# 不使用With表达式。
reader = t.open_reader(partition='pt=test')
count = reader.count
for record in reader[5:10]
    # 处理一条记录。
```

- 通过使用Tunnel API读取表数据, `open_reader` 操作其实也是对Tunnel API的封装。

PyODPS写入数据

类似于 `open_reader`, Table对象同样可以执行 `open_writer` 来打开Writer, 并写数据。示例如下。

```
# 使用With表达式。
with t.open_writer(partition='pt=test') as writer:
    writer.write(records) # 这里records可以是任意可迭代的records,默认写到block 0。
with t.open_writer(partition='pt=test', blocks=[0, 1]) as writer: # 这里同是打开两个block。
    writer.write(0, gen_records(block=0))
    writer.write(1, gen_records(block=1)) # 这里两个写操作可以多线程并行,各个block间是独立的。
# 不使用With表达式。
writer = t.open_writer(partition='pt=test', blocks=[0, 1])
writer.write(0, gen_records(block=0))
writer.write(1, gen_records(block=1))
writer.close() # 不要忘记关闭 writer, 否则数据可能写入不完全。
```

同样, 向表中写入数据也是对Tunnel API的封装。

删除PyODPS表

删除表的操作, 示例如下。

```
odps.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。
```

PyODPS表分区

- 基本操作。

遍历表的全部分区, 示例如下。

```
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, 示例如下。

```
table.exist_partition('pt=test,sub=2019')
```

获取分区，示例如下。

```
partition = table.get_partition('pt=test')
print(partition.creation_time)
2019-09-18 22:22:27
partition.size
0
```

- 创建分区。

```
t.create_partition('pt=test', if_not_exists=True) # 不存在的时候才创建。
```

- 删除分区。

```
t.delete_partition('pt=test', if_exists=True) # 存在的时候才删除。
partition.drop() # Partition对象存在的时候直接drop。
```

PyODPS SQL

PyODPS支持MaxCompute SQL的查询，并可以读取执行的结果。

- 执行SQL。

```
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` 的操作，一种情况是SQL返回了结构化的数据。

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

另一种情况是SQL可能执行的是例如 `desc` 操作，这时通过 `reader.raw` 属性取到原始的SQL执行结果。

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

PyODPS Resource

资源在MaxCompute上常用在UDF和MapReduce中。

列出所有资源可以使用 `list_resources`，判断资源是否存在可以使用 `exist_resource`，删除资源时，可以调用 `delete_resource`，或直接对于Resource对象调用 `drop` 方法。

在PyODPS中，主要支持两种资源类型，一种是文件，另一种是表。

- 文件资源。

文件资源包括基础的FILE类型、以及PY、JAR和ARCHIVE类型。

- 创建文件资源。

创建文件资源可以通过给定资源名、文件类型、以及一个file-like的对象（或者是字符串对象）来创建，示例如下。

```
resource = odps.create_resource('test_file_resource', 'file', file_obj=open('/to/path/file')) # 使用file-like的对象。
resource = odps.create_resource('test_py_resource', 'py', file_obj='import this') # 使用字符串。
```

- 读取和修改文件资源。

对文件资源调用 `open` 方法，或在MaxCompute入口调用 `open_resource` 都能打开一个资源，打开后的对象会是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。
    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 odps.open_resource('test_file_resource', mode='r+') as fp: # 读写模式打开。
    fp.read()
    fp.tell() # 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MaxCompute。
```

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，文件内容会被先清空。
- `a`：追加模式，只能写入内容到文件末尾。
- `r+`：读写模式，能任意读写内容。
- `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，例如一些压缩文件等就需要以这种模式打开，因此`rb`就是指以二进制读模式打开文件，`r+b`是指以二进制读写模式打开。

- 表资源。

创建表资源。

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


更新表资源。

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


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](#)。

DataFrame的相关示例如下。

 **说明** 在所有步骤开始前，需要创建MaxCompute对象。

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

假设已经有三张表，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）以及pyodps_ml_100k_ratings（评分有关的数据）。

只需传入Table对象，便可创建一个DataFrame对象。示例如下。

```
from odps.df import DataFrame
```

```
users = DataFrame(o.get_table('pyodps_ml_100k_users'))
```

通过dtypes属性来查看这个DataFrame有哪些字段，分别是什么类型，示例如下。

```
users.dtypes
```

通过 `head` 方法，可以获取前N条数据，方便快速预览数据。示例如下。

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

number	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
9	10	53	M	lawyer	90703

有时候，并不需要都看到所有字段，便可以从中筛选出一部分。示例如下。

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

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

有时候只是排除个别字段。示例如下。

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

number	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，想要通过计算得到一些新的列，例如将sex为M的置为True，否则为False，并取名叫做sex_bool。示例如下。

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

number	user_id	age	occupation	sex_bool
0	1	24	technician	True
1	2	53	other	False
2	3	23	writer	True
3	4	24	technician	True
4	5	33	other	False

如果想知道年龄在20到25岁之间的人有多少个，示例如下。

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

如果想知道男女用户分别有多少，示例如下。

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

number	sex	count
0	F	273
1	M	670

如果想将用户按职业划分，从高到底，获取人数最多的前10个职业，示例如下。

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

number	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
9	scientist	31

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

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

number	occupation	count
0	student	196
1	other	105

number	occupation	count
2	educator	95
3	administrator	79
4	engineer	67
5	programmer	66
6	librarian	51
7	writer	45
8	executive	32
9	scientist	31

使用更直观的图来查看这份数据，示例如下。

```
%matplotlib inline
```

使用横向的柱状图来可视化，示例如下。

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

将年龄分成30组，查看各年龄分布的直方图，示例如下。

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

使用Join把这三张表进行联合后，把它保存成一张新的表。示例如下。

```
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
```

```
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到80岁的年龄，分成8个年龄段，示例如下。

```
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，示例如下。

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

number	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
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，示例如下。

```
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'))
```

number	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` 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Project	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的时区	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
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

选项	说明	默认值
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MaxCompute2.0语言扩展	False

- 数据上传和下载配置。

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
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