阿里云
MaxCompute

最佳实践

文档版本：20181214

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
	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，恢复业务所需时间约10分钟。
	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。	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。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 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。	`swich` *`{stand | slave}`*

目录

1 与标准SQL的主要区别及解决方法

本文将从习惯使用关系型数据库 SQL 用户的实践角度出发，列举用户在使用 MaxCompute SQL 时比较容易遇见的问题。

MaxCompute SQL 基本区别

应用场景

- 不支持事物（没有 commit 和 rollback，建议代码具有等幂性支持重跑，不推荐使用 Insert Into，推荐 Insert Overwrite 写入数据）。
- 不支持索引和主外键约束。
- 不支持自增字段和默认值。如果有默认值，请在数据写入时自行赋值。

表分区

- 单表支持 6 万个分区。
- 一次查询输入的分区数不能大于 *1* 万，否则执行会报错。另外如果是2级分区且查询时只根据2级分区进行过滤，总的分区数大于1万也可能导致报错。
- 一次查询输出的分区数不能大于*2048*。

精度

- Double 类型因为存在精度问题，不建议在关联时候进行直接等号关联两个 Double 字段。一个比较推荐的做法是把两个数做下减法，如果差距小于一个预设的值就认为是相同，比如 `abs(a1 - a2) < 0.000000001`。
- 目前产品上已经支持高精度的类型 Decimal。但如果有更高精度要求的，可以先把数据存为 String 类型，然后使用 UDF 来实现对应的计算。

数据类型转换

- 为防止出现各种预期外的错误，如果有 2 个不同的字段类型需要做 Join，建议您先把类型转好后再 Join，同时更容易维护代码。
- 关于日期型和字符串的隐式转换。在需要传入日期型的函数里如果传入一个字符串，字符串和日期类型的转换根据 yyyy-mm-dd hh:mi:ss 格式进行转换。如果是其他格式请参见 日期函数 > *TO_DATE*。

DDL 的区别及解法

表结构

- 不能修改分区列列名，只能修改分区列对应的值。具体分区列和分区的区别请参见 分区和分区列的区别。
- 支持增加列，但是不支持删除列以及修改列的数据类型，请参见 *SQL*常见问题。

DML 的区别及解法

INSERT

- 语法上最直观的区别是：Insert into/overwrite 后面有个关键字**Table**。
- 数据插入表的字段映射不是根据 Select 的别名做的，而是根据 Select 的字段的顺序和表里的字段的顺序。

UPDATE/DELETE

- 目前不支持 Update/Delete 语句，如果有需要请参见 更新和删除数据。

SELECT

- 输入表的数量不能超过*16*张。
- Group by查询中的Select 字段，要么是Group By的分组字段，要么需要使用聚合函数。从逻辑角度理解，如发现一个非分组列同一个Group By Key中的数据有多条，不使用聚合函数的话就没办法展示。
- MaxCompute不支持Group by cube（group by with rollup），可以通过union来模拟，比如 `select k1, null, sum(v) from t group by k1 union all select k1, k2, sum(v) from t group by k1, k2;`。

子查询

子查询必须要有别名。建议查询都带别名。

IN/NOT IN

- 关于 In/Not In,Exist/Not Exist，后面的子查询数据量不能超过 1000 条，解决办法请参见 如何使用*Not In*。如果业务上已经保证了子查询返回结果的唯一性，可以考虑去掉 Distinct，从而提升查询性能。

SQL 返回 10000 条

- MaxCompute 限制了单独执行Select语句时返回的数据条数，具体配置请参见 其他操作，设置上限为1万。如果需要查询的结果数据条数很多，请参见 如何获取所有数据，配合Tunnel命令获取全部数据。

MAPJOIN

- Join不支持笛卡尔积，也就是Join必须要用on设置关联条件。如果有一些小表需要做广播表，需要用Mapjoin Hint。详情请参见 如何解决*Join*报错。

ORDER BY

- **Order By**后面需要配合**Limit n**使用。如果希望做很大的数据量的排序，甚至需要做全表排序，可以把这个N设置的很大。不过请谨慎使用，因为无法使用到分布式系统的优势，可能会有性能问题。详情请参见 *MaxCompute* 查询数据的排序。

UNION ALL

- 参与UNION ALL运算的所有列的数据类型、列个数、列名称必须完全一致，否则抛异常。
- UNION ALL查询外面需要再嵌套一层子查询。

2 解决Dataworks 10M文件限制问题

本文将向您介绍如何解决用户在DataWorks上执行MapReduce作业的时候，文件大于10M的JAR和资源文件不能上传到Dataworks的问题,使用调度去定期执行MapReduce作业。

解决方案：

1. 将大于10M的resources通过MaxCompute CLI客户端上传。

   - 客户端下载地址：客户端。
   - 客户端配置AK、EndPoint请参考：安装并配置客户端。

     向客户端添加资源命令：

     ```
     //添加资源
     add jar C:\test_mr\test_mr.jar -f;
     ```

2. 目前通过MaxCompute CLI客户端上传的资源，在Dataworks左侧资源列表是找不到的，只能通过list resources查看确认资源：

   ```
   //查看资源
   list resources;
   ```

3. 减小Jar文件(瘦身Jar)，因为Dataworks执行MR作业的时候，一定要在本地执行，所以保留一个main函数就可以。

   ```
   jar

   -resources test_mr.jar,test_ab.jar --resources在客户端注册后直接引用

   -classpath test_mr.jar --瘦身策略：在gateway上提交要有main和相关的mapper和reducer，额外的三方依赖可以不需要，其他都可以放到resources

   com.aliyun.odps.examples.mr.test_mr wc_in wc_out;
   ```

通过上述方法，我们可以在Dataworks上运行大于10M的MR作业。

3 MaxCompute(ODPS) SQL中的JOIN ON条件

MaxCompute(ODPS) SQL中，很常用的一个操作就是关联(Join)。

概述

目前MaxCompute提供了以下几种Join类型：

类型	含义
Inner Join	输出符合关联条件的数据
Left Join	输出左表的所有记录，对于右表符合关联的数据，输出右表，没有符合的，右表补null。
Right Join	输出右表的所有记录，对于左表符合关联的数据，输出左表，没有符合的，左表补null。
Full Join	输出左表和右表的所有记录，对于没有关联上的数据，未关联的另一侧补null。
Left Semi Join	对于左表中的一条数据，如果右表存在符合关联条件的行，则输出左表。
Left Anti Join	对于左表中的一条数据，如果对于右表所有的行，不存在符合关联条件的数据，则输出左表。

 说明：

User Defined Join 指定两个输入流，您可以自己实现Join的逻辑，这里不展开讨论。

根据不同的场景，用户可以使用不同的Join类型来实现对应的关联操作。但是在实际使用过程中，存在这样的错误示例：

```
A (LEFT/RIGHT/FULL/LEFT SEMI/LEFT ANTI) JOIN B
ON a.key = b.key and A.ds='20180101' and B.ds='20180101';
```

这里用户的本意是希望在A和B中获取某一个分区的数据进行JOIN操作，也就是：

```
(SELECT * FROM A WHERE ds='20180101') A
(LEFT/RIGHT/FULL/LEFT SEMI/LEFT ANTI)  JOIN
(SELECT * FROM B WHERE ds='20180101') B
ON a.key = b.key
```

然而针对不同的Join类型，两者可能并不等价，不仅无法将分区条件下推，导致全表扫描，而且会导致正确性问题。这里简要辨析一下过滤条件分别在：

1. 子查询的WHERE条件。
2. JOIN ON条件。
3. JOIN ON后的WHERE条件。

原理

这里先说明一个JOIN和WHERE条件的计算顺序，对于：

```
(SELECT * FROM A WHERE {subquery_where_condition} A) A
JOIN
(SELECT * FROM B WHERE {subquery_where_condition} B) B
ON {on_condition}
WHERE {where_condition}
```

来说，计算顺序为：

1. 子查询中的`{subquery_where_condition}`
2. JOIN的`{on_condition}`的条件
3. JOIN结果集合`{where_condition}`的计算

对于不同的JOIN类型，过滤语句放在`{subquery_where_condition}`、`{on_condition}`和`{where_condition}`中，有时结果是一致的，有时候结果又是不一致的。下面分情况进行讨论。

实验

1. 准备

   首先构造表A：

   ```
   CREATE TABLE A AS SELECT * FROM VALUES (1, 20180101),(2, 20180101),(2, 20180102) t (key, ds);
   ```

key	ds
1	20180101
2	20180101
2	20180102

   表B：

   ```
   CREATE TABLE B AS SELECT * FROM VALUES (1, 20180101),(3, 20180101),(2, 20180102) t (key, ds);
   ```

key	ds
1	20180101

key	ds
3	20180101
2	20180102

则他们的笛卡尔乘积为：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101
1	20180101	3	20180101
1	20180101	2	20180102
2	20180101	1	20180101
2	20180101	3	20180101
2	20180101	2	20180102
2	20180102	1	20180101
2	20180102	3	20180101
2	20180102	2	20180102

2. Inner Join

结论：过滤条件在`{subquery_where_condition}`、`{on_condition}`和`{where_condition}`中都是等价的。

Inner Join的处理逻辑是将左右表进行笛卡尔乘积，然后选择满足ON表达式的行进行输出。

a. 第一种情况，子查询中过滤：

```
SELECT A.*, B.*
FROM
(SELECT * FROM A WHERE ds='20180101') A
JOIN
(SELECT * FROM B WHERE ds='20180101') B
ON a.key = b.key;
```

非常简单，结果只有一条：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101

b. 第二种情况，JOIN 条件中过滤：

```
SELECT A.*, B.*
FROM A JOIN B
```

```
ON a.key = b.key and A.ds='20180101' and B.ds='20180101';
```

笛卡尔积的结果有9条，满足ON条件的结果同样只有1条：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101

c. 第三种情况，JOIN后的WHERE条件过滤：

```
SELECT A.*, B.*
FROM A JOIN B
ON a.key = b.key
WHERE A.ds='20180101' and B.ds='20180101';
```

来说，笛卡尔积的结果有9条，满足ON条件`a.key = b.key`的结果有3条，分别是：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101
2	20180102	2	20180102
2	20180101	2	20180102

此时对于这个结果再进行过滤`A.ds='20180101' and B.ds='20180101'`，结果只有1条：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101

可以看到，将过滤条件放在三个不同的地方，得到了三种不同的结果。

3. Left Join

结论：过滤条件在`{subquery_where_condition}`、`{on_condition}`和`{where_condition}`不一定等价。

对于左表的过滤条件，放在`{subquery_where_condition}`和`{where_condition}`是等价的。

对于右表的过滤条件，放在`{subquery_where_condition}`和`{on_condition}`中是等价的。

Left Join的处理逻辑是将左右表进行笛卡尔乘积，然后对于满足ON表达式的行进行输出，对于左表中不满足ON表达式的行，输出左表，右表补NULL。

a. 第一种情况，子查询中过滤：

```
SELECT A.*, B.*
FROM
(SELECT * FROM A WHERE ds='20180101') A
LEFT JOIN
(SELECT * FROM B WHERE ds='20180101') B
ON a.key = b.key;
```

过滤后，左右侧有两条，右侧有一条，结果有两条：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101
2	20180101	NULL	NULL

b. 第二种情况，JOIN 条件中过滤：

```
SELECT A.*, B.*
FROM A JOIN B
ON a.key = b.key and A.ds='20180101' and B.ds='20180101';
```

笛卡尔积的结果有9条，满足ON条件的结果同样只有1条，则对于左表剩余的两条输出左表，右表补NULL：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101
2	20180101	NULL	NULL
2	20180102	NULL	NULL

c. 第三种情况，JOIN后的WHERE条件过滤：

```
SELECT A.*, B.*
FROM A JOIN B
ON a.key = b.key
WHERE A.ds='20180101' and B.ds='20180101';
```

来说，笛卡尔积的结果有9条，满足ON条件`a.key = b.key`的结果有3条，分别是：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101
2	20180101	2	20180102
2	20180102	2	20180102

此时对于这个结果再进行过滤`A.ds='20180101' and B.ds='20180101'`，结果只有1条：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101

可以看到，将过滤条件放在三个不同的地方，得到了三种不同的结果。

4. Right Join

Right Join和Left Join是类似的，只是左右表的区别。结论：过滤条件在`{subquery_where_condition}`、`{on_condition}`和`{where_condition}`不一定等价。对于右表的过滤条件，放在`{subquery_where_condition}`和`{where_condition}`是等价的。对于左表的过滤条件，放在`{subquery_where_condition}`和`{on_condition}`中是等价的。

5. Full Join

结论：过滤条件写在`{subquery_where_condition}`、`{on_condition}`和`{where_condition}`均不等价。

FULL Join的处理逻辑是将左右表进行笛卡尔乘积，然后对于满足ON表达式的行进行输出，对于两侧表中不满足ON表达式的行，输出有数据的表，另一侧补NULL。

a. 第一种情况，子查询中过滤：

```
SELECT A.*, B.*
FROM
(SELECT * FROM A WHERE ds='20180101') A
FULL JOIN
(SELECT * FROM B WHERE ds='20180101') B
ON a.key = b.key;
```

过滤后，左右侧有两条，右侧有两条，结果有三条：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101
2	20180101	NULL	NULL
NULL	NULL	3	20180101

b. 第二种情况，JOIN 条件中过滤：

```
SELECT A.*, B.*
FROM A FULL JOIN B
```

```
ON a.key = b.key and A.ds='20180101' and B.ds='20180101';
```

笛卡尔积的结果有9条，满足ON条件的结果同样只有1条，则对于左表剩余的两条输出左表，右表补NULL。右表剩余的两条输出右表，左表补NULL：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101
2	20180101	NULL	NULL
2	20180102	NULL	NULL
NULL	NULL	3	20180101
NULL	NULL	2	20180102

c. 第三种情况，JOIN后的WHERE条件过滤：

```
SELECT A.*, B.*
FROM A FULL JOIN B
ON a.key = b.key
WHERE A.ds='20180101' and B.ds='20180101';
```

笛卡尔积的结果有9条，满足ON条件`a.key = b.key`的结果有3条，分别是：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101
2	20180101	2	20180102
2	20180102	2	20180102

再对没有JOIN上的数据进行输出，另一侧补NULL，得到结果：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101
2	20180101	2	20180102
2	20180102	2	20180102
NULL	NULL	3	20180101

此时对于这个结果再进行过滤`A.ds='20180101' and B.ds='20180101'`，结果只有1条：

a.key	a.ds	b.key	b.ds
1	20180101	1	20180101

可以看到，和LEFT JOIN类似，得到了三种不同的结果。

6. Left Semi Join

结论：过滤条件写在{subquery_where_condition}、{on_condition}和{where_condition}是等价的。

LEFT SEMI Join的处理逻辑是对于左表的每一条记录，都去和右表进行匹配，如果匹配成功，则输出左表。这里需要注意的是由于只输出左表，所以JOIN后的Where条件中不能写右侧的过滤条件。LEFT SEMI JOIN常用来实现exists的语义：

a. 第一种情况，子查询中过滤：

```
SELECT A.*
FROM
(SELECT * FROM A WHERE ds='20180101') A
LEFT SEMI JOIN
(SELECT * FROM B WHERE ds='20180101') B
ON a.key = b.key;
```

过滤后，左右侧有两条，最终符合a.key = b.key的只有一条：

a.key	a.ds
1	20180101

b. 第二种情况，JOIN 条件中过滤：

```
SELECT A.*
FROM A LEFT SEMI JOIN B
ON a.key = b.key and A.ds='20180101' and B.ds='20180101';
```

对于左侧的三条记录，满足ON条件的结果同样只有1条：

a.key	a.ds
1	20180101

c. 第三种情况，JOIN后的WHERE条件过滤：

```
SELECT A.*
FROM A LEFT SEMI JOIN
(SELECT * FROM B WHERE ds='20180101') B
ON a.key = b.key
WHERE A.ds='20180101';
```

左侧能符合ON条件的有一条：

a.key	a.ds
1	20180101

此时对于这个结果再进行过滤A.ds='20180101'，结果仍然保持1条：

a.key	a.ds
1	20180101

可以看到，LEFT SEMI JOIN和INNER JOIN类似，无论过滤条件放在哪里，结果都是一致的。

7. Left Anti Join

结论：过滤条件写在{subquery_where_condition}、{on_condition}和{where_condition}不一定等价。

对于左表的过滤条件，放在{subquery_where_condition}和{where_condition}是等价的。

对于右表的过滤条件，放在{subquery_where_condition}和{on_condition}中是等价的，右表表达式不能放在{where_condition}中。

LEFT ANTI Join的处理逻辑是对于左表的每一条记录，都去和右表进行匹配，如果右表所有的记录都没有匹配成功，则输出左表。同样由于只输出左表，所以JOIN后的Where条件中不能写右侧的过滤条件。LEFT SEMI JOIN常常用来实现not exists的语义。

a. 第一种情况，子查询中过滤：

```
SELECT A.*
FROM
(SELECT * FROM A WHERE ds='20180101') A
LEFT ANTI JOIN
(SELECT * FROM B WHERE ds='20180101') B
ON a.key = b.key;
```

过滤后，左侧有两条，右侧有两条，结果有1条：

a.key	a.ds
2	20180101

b. 第二种情况，JOIN 条件中过滤：

```
SELECT A.*
FROM A LEFT ANTI JOIN B
```

```
ON a.key = b.key and A.ds='20180101' and B.ds='20180101';
```

对于左侧的三条记录，只有第一条有满足ON条件的结果，所以输出剩余的两条记录：

a.key	a.ds
2	20180101
2	20180102

c. 第三种情况，JOIN后的WHERE条件过滤：

```
SELECT A.*
FROM A LEFT ANTI JOIN
(SELECT * FROM B WHERE ds='20180101') B
ON a.key = b.key
WHERE A.ds='20180101';
```

左侧能通过ON条件的有两条：

a.key	a.ds
2	20180101
2	20180102

此时对于这个结果再进行过滤`A.ds='20180101'`，结果为1条：

a.key	a.ds
2	20180101

可以看到，LEFT ANTI JOIN中，过滤条件WHERE语句分别放在JOIN ON条件中、条件前和条件后，得到的结果是不相同的。

以上内容只是针一个常用场景测试的几种不同的写法，没有具体的推导过程，对于涉及到不等值表达式的场景会更加复杂，如果您有兴趣可以尝试推导一下。

总结

过滤条件放在不同的位置语义可能大不相同，对于用户而言，如果只是进行过滤数据后再JOIN的操作，可以简要记住以下几点。

1. INNER JOIN/LEFT SEMI JOIN 对于两侧的表达式可以随便写。
2. LEFT JOIN/LEFT ANTI JOIN 左表的过滤条件要放到`{subquery_where_condition}`或者`{where_condition}`，右表的过滤条件要放到`{subquery_where_condition}`或者`{on_condition}`中。

3. RIGHT JOIN和LEFT JOIN相反，右表的过滤条件要放到{subquery_where_condition}或者{where_condition}，左表的过滤条件要放到{subquery_where_condition}或者{on_condition}。
4. FULL OUTER JOIN 只能放到{subquery_where_condition}中。

当然如果还是觉得规则比较复杂的话，最好的方法就是每次都把过滤条件写到子查询中。

4 导出SQL的运行结果

本文将通过示例，为您介绍几种下载MaxCompute SQL计算结果的方法。

说明：

本文中所有SDK部分仅举例介绍Java的例子。

您可以通过以下几种方法导出SQL的运行结果：

- 如果数据比较少，可以直接用*SQL Task*得到全部的查询结果。
- 如果只是想导出某个表或者分区，可以用 *Tunnel*直接导出数据。
- 如果SQL比较复杂，需要Tunnel和SQL相互配合才行。
- *DataWorks* 可以方便地帮您运行SQL，同步数据，并有定时调度，配置任务依赖的功能。
- 开源工具 **DataX** 可帮助您方便地把 MaxCompute 中的数据导出到目标数据源，详情请参见 *DataX* 概述。

SQLTask方式导出

SQL Task 是 SDK直接调用MaxCompute SQL的接口，能很方便地运行SQL并获得其返回结果。

从文档可以看到，`SQLTask.getResult(i);` 返回的是一个 List，可以循环迭代这个List，获得完整的SQL计算返回结果。不过此方法有个缺陷，详情请参见 其他操作中提到的`SetProject READ_TABLE_MAX_ROW`功能。

目前Select语句返回给客户端的数据条数最大可以调整到 **1**万。也就是说如果在客户端上（包括SQLTask）直接Select，那相当于查询结果上最后加了个Limit N（如果是CREATE TABLE XX AS SELECT或者用INSERT INTO/OVERWRITE TABLE把结果固化到具体的表里就没关系）。

Tunnel 方式导出

如果您需要导出的查询结果是某张表的全部内容（或者是具体的某个分区的全部内容），可以通过Tunnel来实现，详情请参见 命令行工具 和基于SDK编写的 *Tunnel SDK*。

在此提供一个Tunnel命令行导出数据的简单示例，Tunnel SDK的编写是在有一些命令行没办法支持的情况下才需要考虑，详情请参见 批量数据通道概述。

```
tunnel d wc_out c:\wc_out.dat;
2016-12-16 19:32:08 - new session: 201612161932082d3c9b0a012f68e7
total lines: 3
2016-12-16 19:32:08 - file [0]: [0, 3), c:\wc_out.dat
downloading 3 records into 1 file
2016-12-16 19:32:08 - file [0] start
```

```
2016-12-16 19:32:08 - file [0] OK. total: 21 bytes
download OK
```

SQLTask+Tunnel方式导出

从前面SQL Task方式导出的介绍可以看到，SQL Task不能处理超过1万条记录，而 Tunnel可以，两者可以互补。所以可以基于两者实现数据的导出。

代码实现的示例如下：

```
private static final String accessId = "userAccessId";
private static final String accessKey = "userAccessKey";
private static final String endPoint = "http://service.odps.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("-", "_");//其实也就是随便找了个随机字符串作为临时表的名字
private static final Odps odps = getOdps();
public static void main(String[] args) {
System.out.println(table);
runSql();
tunnel();
}
/*
* 把SQLTask的结果下载过来
* */
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
* 数据量少的话直接打印后拷贝走也是一种取巧的方法。实际场景可以用Java.io写到本地文件，或者写到远端数据等各种目标保存起来。
* */
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，方便后面用Tunnel下载
* 这里保存数据的lifecycle为1天，所以哪怕删除步骤出了问题，也不会太浪费存储空间
* */
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(原ODPS)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;
}
```

大数据开发套件的数据同步方式导出

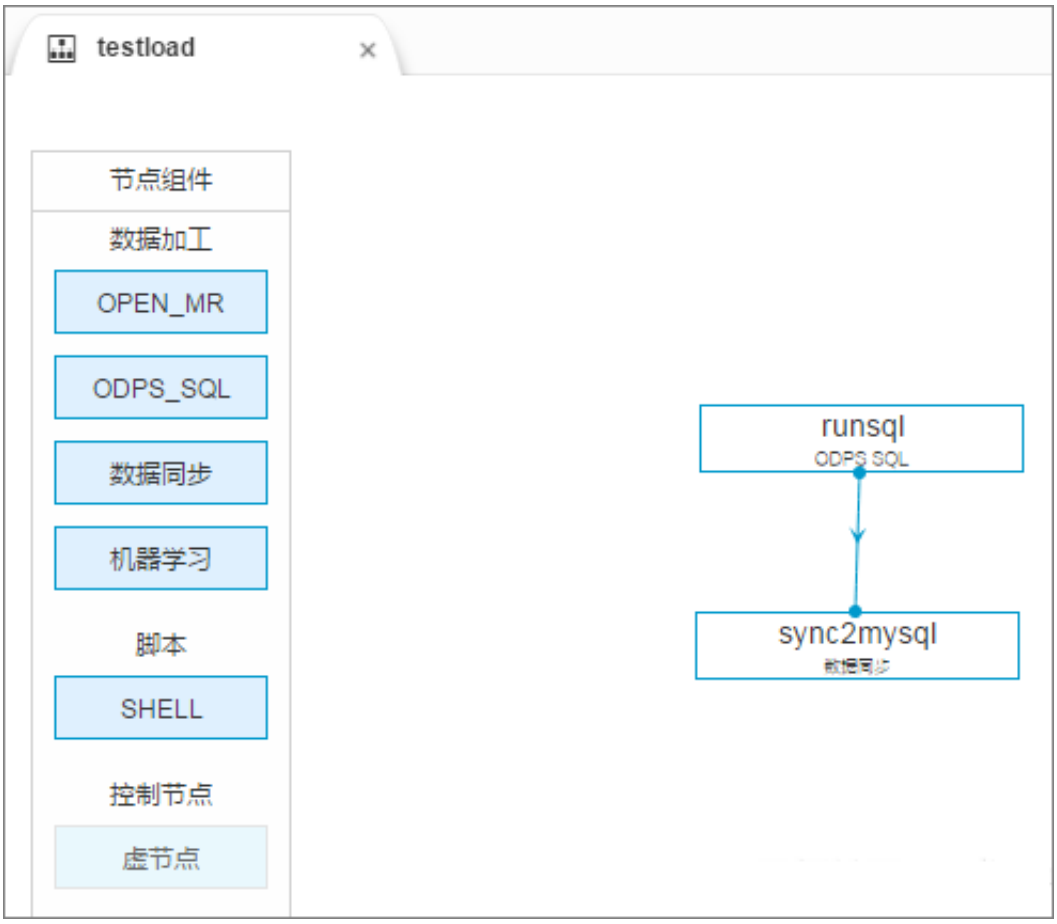
前面介绍的方式解决了数据下载后保存的问题，但是没解决数据的生成以及两个步骤之间的调度依赖的问题。

数加*DataWorks* 可以运行SQL、配置数据同步任务，还可以设置自动 周期性运行 和 多任务之间依赖，彻底解决了前面的烦恼。

接下来将用一个简单示例，为您介绍如何通过大数据开发套件运行SQL并配置数据同步任务，以完成数据生成和导出需求。

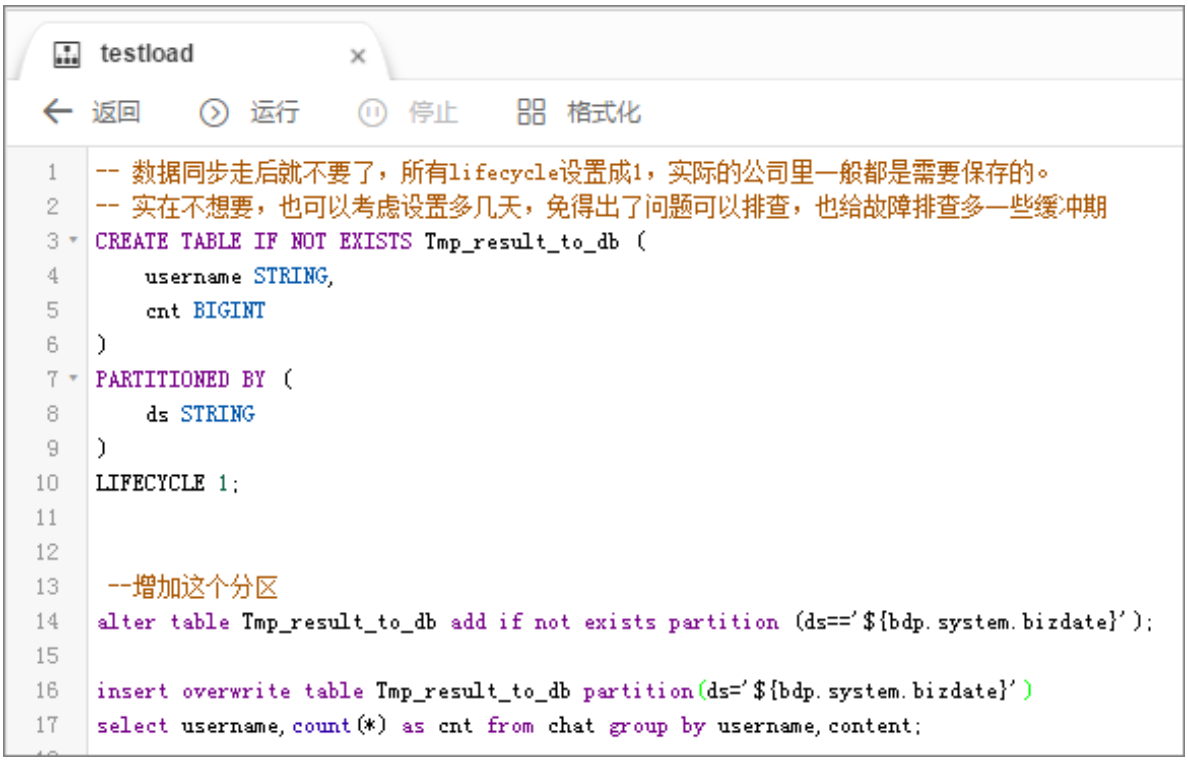
操作步骤

1. 创建一个工作流，工作流里创建一个SQL节点和一个数据同步节点，并将两个节点连线配置成依赖关系，SQL节点作为数据产出的节点，数据同步节点作为数据导出节点。



2. 配置SQL节点。

> **说明：**
> SQL这里的创建表要先执行一次再去配置同步（否则表都没有，同步任务没办法配置）。



3. 配置数据同步任务。

   a. 选择来源。

   b. 选择目标。

c. 字段映射。

d. 通道控制。

e. 预览保存。

4. 工作流调度配置完成后（可以直接使用默认配置），保存并提交工作流，然后单击 **测试运行**。

   查看数据同步的运行日志，如下所示：

```
2016-12-17 23:43:46.394 [job-15598025] INFO JobContainer -
任务启动时刻 : 2016-12-17 23:43:34
任务结束时刻 : 2016-12-17 23:43:46
任务总计耗时 : 11s
任务平均流量 : 31.36KB/s
记录写入速度 : 1668rec/s
读出记录总数 : 16689
读写失败总数 : 0
```

5. 输入SQL语句查看数据同步的结果，如下图所示：

5 修改不兼容SQL实战

MaxCompute 开发团队近期已经完成了 MaxCompute2.0 灰度升级。新升级的版本完全拥抱开源生态，支持更多的语言功能，带来更快的运行速度，同时新版本会执行更严格的语法检测，以致于一些在老编译器下正常执行的不严谨的语法 case 在 MaxCompute2.0 下会报错。

为了使 MaxCompute2.0 灰度升级更加平滑，MaxCompute 框架支持回退机制，如果 MaxCompute2.0 任务失败，会回退到 MaxCompute1.0 执行。回退本身会增加任务 E2E 时延。鼓励大家提交作业之前，手动关闭回退`set odps.sql.planner.mode=lot;`以避免 MaxCompute 框架回退策略修改对大家造成影响。

MaxCompute 团队会根据线上回退情况，邮件或者钉钉等通知有问题任务的 Owner，请大家尽快完成 SQL 任务修改，否则会导致任务失败。烦请大家仔细 check 以下报错情况，进行自检，以免通知遗漏造成任务失败。

下面列举常见的一些会报错的语法：

group.by.with.star

SELECT * …GROUP BY… 的问题。

旧版 MaxCompute 中，即使 * 中覆盖的列不在 group by key 内，也支持 `select * from group by key` 的语法，但 MaxCompute2.0 和 Hive 兼容，并不允许这种写法，除非 group by 列表是所有源表中的列。示例如下：

场景一：**group by key** 不包含所有列

错误写法：

```
SELECT * FROM t GROUP BY key;
```

报错信息：

```
FAILED: ODPS-0130071:[1,8] Semantic analysis exception - column
reference t.value should appear in GROUP BY key
```

正确改法：

```
SELECT DISTINCT key FROM t;
```

场景二：**group by key** 包含所有列

不推荐写法：

```
SELECT * FROM t GROUP BY key, value; -- t has columns key and value
```

虽然 MaxCompute2.0 不会报错，但推荐改为：

```
SELECT DISTINCT key, value FROM t;
```

bad.escape

错误的 escape 序列问题。

按照 MaxCompute 文档的规定，在 string literal 中应该用反斜线加三位8进制数字表示从 0 到 127 的 ASCII 字符，例如：使用 \001， \002 表示 0，1 等。但目前\01，\0001 也被当作 \001 处理了。

这种行为会给新用户带来困扰，比如需要用“\0001”表示“\000”+“1”，便没有办法实现。同时对于从其他系统迁移而来的用户而言，会导致正确性错误。

 **说明：**

\000后面在加数字，如\0001 - \0009或\00001的写法可能会返回错误。

MaxCompute2.0 会解决此问题，需要 script 作者将错误的序列进行修改，示例如下：

错误写法：

```
SELECT split(key, "\01"), value like "\0001" FROM t;
```

报错信息：

```
FAILED: ODPS-0130161:[1,19] Parse exception - unexpected escape
sequence: 01
ODPS-0130161:[1,38] Parse exception - unexpected escape sequence: 0001
```

正确改法：

```
SELECT split(key, "\001"), value like "\001" FROM t;
```

column.repeated.in.creation

create table 时列名重复的问题。

如果 create table 时列名重复，MaxCompute2.0 将会报错，示例如下：

错误写法：

```
CREATE TABLE t (a BIGINT, b BIGINT, a BIGINT);
```

报错信息：

```
FAILED: ODPS-0130071:[1,37] Semantic analysis exception - column
repeated in creation: a
```

正确改法：

```
CREATE TABLE t (a BIGINT, b BIGINT);
```

string.join.double

写 **JOIN** 条件时，等号的左右两边分别是 **String** 和 **Double** 类型。

出现上述情况，旧版 MaxCompute 会把两边都转成 Bigint，但会导致严重的精度损失问题，例如：1.1 = “1” 在连接条件中会被认为是相等的。但 MaxCompute2.0 会与 Hive 兼容转为 Double。

不推荐写法：

```
SELECT * FROM t1 JOIN t2 ON t1.double_value = t2.string_value;
```

warning 信息：

```
WARNING:[1,48]  implicit conversion from STRING to DOUBLE, potential
data loss, use CAST function to suppress
```

推荐改法：

```
select * from t1 join t2 on t.double_value = cast(t2.string_value as
double);
```

除以上改法外，也可使用用户期望的其他转换方式。

window.ref.prev.window.alias

Window Function 引用同级 **Select List** 中的其他 **Window Function Alias** 的问题。

示例如下：

如果 rn 在 t1 中不存在，错误写法如下：

```

SELECT row_number() OVER (PARTITION BY c1 ORDER BY c1) rn,
row_number() OVER (PARTITION by c1 ORDER BY rn) rn2
```

```
FROM t1;
```

报错信息：

```
FAILED: ODPS-0130071:[2,45] Semantic analysis exception - column rn
cannot be resolved
```

正确改法：

```
SELECT row_number() OVER (PARTITION BY c1 ORDER BY rn) rn2
FROM
(
SELECT c1, row_number() OVER (PARTITION BY c1 ORDER BY c1) rn
FROM t1
) tmp;
```

select.invalid.token.after.star

select * 后面接 **alias** 的问题。

Select 列表里面允许用户使用 * 代表选择某张表的全部列，但 * 后面不允许加 alias（即使 * 展开之后只有一列也不允许），新一代编译器将会对类似语法进行报错，示例如下：

错误写法：

```
select * as alias from dual;
```

报错信息：

```
FAILED: ODPS-0130161:[1,10] Parse exception - invalid token 'as'
```

正确改法：

```
select * from dual;
```

agg.having.ref.prev.agg.alias

有 **Having** 的情况下，**Select List** 可以出现前面 **Aggregate Function Alias** 的问题。示例如下：

错误写法：

```
SELECT count(c1) cnt,
sum(c1) / cnt avg
FROM t1
GROUP BY c2
```

```
HAVING cnt > 1;
```

报错信息：

```
FAILED: ODPS-0130071:[2,11] Semantic analysis exception - column cnt
cannot be resolved
ODPS-0130071:[2,11] Semantic analysis exception - column reference cnt
 should appear in GROUP BY key
```

其中 s、cnt 在源表 t1 中都不存在，但因为有 HAVING，旧版 MaxCompute 并未报错，

MaxCompute2.0 则会提示 column cannot be resolve，并报错。

正确改法：

```
SELECT cnt, s, s/cnt avg
FROM
(
SELECT count(c1) cnt,
sum(c1) s
FROM t1
GROUP BY c2
HAVING count(c1) > 1
) tmp;
```

order.by.no.limit

ORDER BY 后没有 LIMIT 语句的问题。

MaxCompute 默认 order by 后需要增加 limit 限制数量，因为 order by 是全量排序，没有 limit 时执

行性能较低。示例如下：

错误写法：

```
select * from (select *
from (select cast(login_user_cnt as int) as uv, '3' as shuzi
from test_login_cnt where type = 'device' and type_name = 'mobile') v
order by v.uv desc) v
order by v.shuzi limit 20;
```

报错信息：

```
FAILED: ODPS-0130071:[4,1] Semantic analysis exception - ORDER BY must
 be used with a LIMIT clause
```

正确改法：

在子查询 order by v.uv desc 中增加 limit。

另外，MaxCompute1.0 对于 view 的检查不够严格。比如在一个不需要检查 LIMIT 的 Projec (odps.sql.validate.orderby.limit=false) 中，创建了一个 View：

```
CREATE VIEW dual_view AS SELECT id FROM dual ORDER BY id;
```

若访问此 View：

```
SELECT * FROM dual_view;
```

MaxCompute1.0 不会报错，而 MaxCompute2.0 会报如下错误信息：

```
FAILED: ODPS-0130071:[1,15] Semantic analysis exception - while
resolving view xdj.xdj_view_limit - ORDER BY must be used with a LIMIT
 clause
```

generated.column.name.multi.window

使用自动生成的 **alias** 的问题。

旧版 MaxCompute 会为 Select 语句中的每个表达式自动生成一个 alias，这个 alias 会最后显示在 console 上。但是，它并不承诺这个 alias 的生成规则，也不承诺这个 alias 的生成规则会保持不变，所以不建议用户使用自动生成的 alias。

MaxCompute2.0 会对使用自动生成 alias 的情况给予警告，由于牵涉面较广，暂时无法直接给予禁止。

对于某些情况，MaxCompute 的不同版本间生成的 alias 规则存在已知的变动，但因为已有一些线上作业依赖于此类 alias，这些查询在 MaxCompute 版本升级或者回滚时可能会失败，存在此问题的用户，请修改您的查询，对于感兴趣的列，显式地指定列的别名。示例如下：

不推荐写法：

```
SELECT _c0 FROM (SELECT count(*) FROM dual) t;
```

建议改法：

```
SELECT c FROM (SELECT count(*) c FROM dual) t;
```

non.boolean.filter

使用了非 **boolean** 过滤条件的问题。

MaxCompute 不允许布尔类型与其他类型之间的隐式转换，但旧版 MaxCompute 会允许用户在某些情况下使用 Bigint 作为过滤条件。MaxCompute2.0 将不再允许，如果您的脚本中存在这样的过滤条件，请及时修改。示例如下：

错误写法：

```
select id, count(*) from dual group by id having id;
```

报错信息：

```
FAILED: ODPS-0130071:[1,50] Semantic analysis exception - expect a
BOOLEAN expression
```

正确改法：

```
select id, count(*) from dual group by id having id <> 0;
```

post.select.ambiguous

在 **order by、 cluster by、 distribute by、sort by** 等语句中，引用了名字冲突的列的问题。

旧版 MaxCompute 中，系统会默认选取 Select 列表中的后一列作为操作对象，MaxCompute2.0 将会进行报错，请及时修改。示例如下：

错误写法：

```
select a, b as a from t order by a limit 10;
```

报错信息：

```
FAILED: ODPS-0130071:[1,34] Semantic analysis exception - a is
ambiguous, can be both t.a or null.a
```

正确改法：

```
select a as c, b as a from t order by a limit 10;
```

本次推送修改会包括名字虽然冲突但语义一样的情况，虽然不会出现歧义，但是考虑到这种情况容易导致错误，作为一个警告，希望用户进行修改。

duplicated.partition.column

在 **query** 中指定了同名的 **partition** 的问题。

旧版 MaxCompute 在用户指定同名 partition key 时并未报错， 而是后一个的值直接覆盖了前一个，容易产生混乱。MaxCompute2.0 将会对此情况进行报错，示例如下：

错误写法一：

```
insert overwrite table partition (ds = '1', ds = '2')select ... ;
```

实际上，在运行时 ds = ‘1’ 被忽略。

正确改法：

```
insert overwrite table partition (ds = '2')select ... ;
```

错误写法二：

```
create table t (a bigint, ds string) partitioned by (ds string);
```

正确改法：

```
create table t (a bigint) partitioned by (ds string);
```

order.by.col.ambiguous

Select list 中 **alias** 重复，之后的 **Order by** 子句引用到重复的 **alias** 的问题。

错误写法：

```
SELECT id, id
FROM dual
ORDER BY id;
```

正确改法：

```
SELECT id, id id2
FROM dual
ORDER BY id;
```

需要去掉重复的 alias，Order by 子句再进行引用。

in.subquery.without.result

colx in subquery 没有返回任何结果，则 **colx** 在源表中不存在的问题。

错误写法：

```
SELECT * FROM dual
WHERE not_exist_col IN (SELECT id FROM dual LIMIT 0);
```

报错信息：

```
FAILED: ODPS-0130071:[2,7] Semantic analysis exception - column
not_exist_col cannot be resolved
```

ctas.if.not.exists

目标表语法错误问题。

如果目标表已经存在，旧版 MaxCompute 不会做任何语法检查，MaxCompute2.0 则会做正常的语法检查，这种情况会出现很多错误信息，示例如下：

错误写法：

```
CREATE TABLE IF NOT EXISTS dual
AS
SELECT * FROM not_exist_table;
```

报错信息：

```
FAILED: ODPS-0130131:[1,50] Table not found - table meta_dev.
not_exist_table cannot be resolved
```

worker.restart.instance.timeout

旧版 MaxCompute UDF 每输出一条记录，便会触发一次对分布式文件系统的写操作，同时会向 Fuxi 发送心跳，如果 UDF 10 分钟没有输出任何结果，会得到如下错误提示：

```
FAILED: ODPS-0123144: Fuxi job failed - WorkerRestart errCode:252,
errMsg:kInstanceMonitorTimeout, usually caused by bad udf performance.
```

MaxCompute2.0 的 Runtime 框架支持向量化，一次会处理某一列的多行来提升执行效率。但向量化可能导致原来不会报错的语句（2 条记录的输出时间间隔不超过 10 分钟），因为一次处理多行，没有及时向 Fuxi 发送心跳而导致 timeout。

遇到这个错误，建议首先检查 UDF 是否有性能问题，每条记录需要数秒的处理时间。如果无法优化 UDF 性能，可以尝试手动设置 batch row 大小来绕开（默认为1024）：

```
set odps.sql.executionengine.batch.rowcount=16;
```

divide.nan.or.overflow

旧版 **MaxCompute** 不会做除法常量折叠的问题。

比如如下语句，旧版 MaxCompute 对应的物理执行计划如下：

```
EXPLAIN
SELECT IF(FALSE, 0/0, 1.0)
FROM dual;
In Task M1_Stg1:
    Data source: meta_dev.dual
    TS: alias: dual
      SEL: If(False, Divide(UDFToDouble(0), UDFToDouble(0)), 1.0)
```

```
        FS: output: None
```

由此可以看出，IF 和 Divide 函数仍然被保留，运行时因为 IF 第一个参数为 false，第二个参数 Divide 的表达式不需要求值，所以不会出现除零异常。

而 MaxCompute2.0 则支持除法常量折叠，所以会报错。如下所示：

错误写法：

```
SELECT IF(FALSE, 0/0, 1.0)
FROM dual;
```

报错信息：

```
FAILED: ODPS-0130071:[1,19] Semantic analysis exception - encounter
 runtime exception while evaluating function /, detailed message:
DIVIDE func result NaN, two params are 0.000000 and 0.000000
```

除了上述的 nan，还可能遇到 overflow 错误，比如：

错误写法：

```
SELECT IF(FALSE, 1/0, 1.0)
FROM dual;
```

报错信息：

```
FAILED: ODPS-0130071:[1,19] Semantic analysis exception - encounter
 runtime exception while evaluating function /, detailed message:
DIVIDE func result overflow, two params are 1.000000 and 0.000000
```

正确改法：

建议去掉 /0 的用法，换成合法常量。

CASE WHEN 常量折叠也有类似问题，比如：CASE WHEN TRUE THEN 0 ELSE 0/0，MaxCompute2.0 常量折叠时所有子表达式都会求值，导致除0错误。

CASE WHEN 可能涉及更复杂的优化场景，比如：

```
SELECT CASE WHEN key = 0 THEN 0 ELSE 1/key END
FROM (
SELECT 0 AS key FROM src
UNION ALL
```

```
SELECT key FROM src) r;
```

优化器会将除法下推到子查询中，转换类似于：

```
M (
SELECT CASE WHEN 0 = 0 THEN 0 ELSE 1/0 END c1 FROM src
UNION ALL
SELECT CASE WHEN key = 0 THEN 0 ELSE 1/key END c1 FROM src) r;
```

报错信息：

```
FAILED: ODPS-0130071:[0,0] Semantic analysis exception - physical plan
 generation failed: java.lang.ArithmeticException: DIVIDE func result
overflow, two params are 1.000000 and 0.000000
```

其中 UNION ALL 第一个子句常量折叠报错，建议将 SQL 中的 CASE WHEN 挪到子查询中，并去掉无用的 CASE WHEN 和去掉/0用法：

```
SELECT c1 END
FROM (
SELECT 0 c1 END FROM src
UNION ALL
SELECT CASE WHEN key = 0 THEN 0 ELSE 1/key END) r;
```

small.table.exceeds.mem.limit

旧版 MaxCompute 支持 Multi-way Join 优化，多个 Join 如果有相同 Join Key，会合并到一个 Fuxi Task 中执行，比如下面例子中的 J4_1_2_3_Stg1：

```
EXPLAIN
SELECT t1.*
FROM t1 JOIN t2 ON t1.c1 = t2.c1
JOIN t3 ON t1.c1 = t3.c1;
```

旧版 MaxCompute 物理执行计划：

```
In Job job0:
root Tasks: M1_Stg1, M2_Stg1, M3_Stg1
J4_1_2_3_Stg1 depends on: M1_Stg1, M2_Stg1, M3_Stg1

In Task M1_Stg1:
    Data source: meta_dev.t1

In Task M2_Stg1:
    Data source: meta_dev.t2

In Task M3_Stg1:
    Data source: meta_dev.t3

In Task J4_1_2_3_Stg1:
    JOIN: t1 INNER JOIN unknown INNER JOIN unknown
```

```
        SEL: t1._col0, t1._col1, t1._col2
            FS: output: None
```

如果增加 MapJoin hint，旧版 MaxCompute 物理执行计划不会改变。也就是说对于旧版 MaxCompute 优先应用 Multi-way Join 优化，并且可以忽略用户指定 MapJoin hint。

```
EXPLAIN
SELECT /*+mapjoin(t1)*/ t1.*
FROM t1 JOIN t2 ON t1.c1 = t2.c1
JOIN t3 ON t1.c1 = t3.c1;
```

旧版 MaxCompute 物理执行计划同上。

MaxCompute2.0 Optimizer 会优先使用用户指定的 MapJoin hint，对于上述例子，如果 t1 比较大的话，会遇到类似错误：

```
FAILED: ODPS-0010000:System internal error - SQL Runtime Internal
 Error: Hash Join Cursor HashJoin_REL··· small table exceeds, memory
 limit(MB) 640, fixed memory used ···, variable memory used ···
```

对于这种情况，如果 MapJoin 不是期望行为，建议去掉 MapJoin hint。

sigkill.oom

同 small.table.exceeds.mem.limit，如果用户指定了 MapJoin hint，并且用户本身所指定的小表比较大。在旧版 MaxCompute 下有可能被优化成 Multi-way Join 从而成功。但在 MaxCompute 2.0 下，用户可能通过设定 odps.sql.mapjoin.memory.max 来避免小表超限的错误，但每个 MaxCompute worker 有固定的内存限制，如果小表本身过大，则 MaxCompute worker 会由于内存超限而被杀掉，错误类似于：

```
Fuxi job failed - WorkerRestart errCode:9,errMsg:SigKill(OOM), usually
 caused by OOM(outof memory).
```

这里建议您去掉 MapJoin hint，使用 Multi-way Join。

wm_concat.first.argument.const

聚合函数 中关于 WM_CONCAT 的说明，一直要求 WM_CONCAT 第一个参数为常量，旧版 MaxCompute 检查不严格，比如源表没有数据，就算 WM_CONCAT 第一个参数为 ColumnReference，也不会报错。

```
函数声明：
string wm_concat(string separator, string str)
参数说明：
```

`separator`：String类型常量，分隔符。其他类型或非常量将引发异常。

MaxCompute2.0，会在 plan 阶段便检查参数的合法性，假如 WM_CONCAT 的第一个参数不是常量，会立即报错。示例如下：

错误写法：

```
SELECT wm_concat(value, ',') FROM src GROUP BY value;
```

报错信息：

```
FAILED: ODPS-0130071:[0,0] Semantic analysis
 exception - physical plan generation failed:
 com.aliyun.odps.lot.cbo.validator.AggregateCallValidator
$AggregateCallValidationException: Invalid argument type - The first
 argument of WM_CONCAT must be constant string.
```

pt.implicit.convertion.failed

srcpt 是一个分区表，并有两个分区：

```
CREATE TABLE srcpt(key STRING, value STRING) PARTITIONED BY (pt STRING
);
ALTER TABLE srcpt ADD PARTITION (pt='pt1');
ALTER TABLE srcpt ADD PARTITION (pt='pt2');
```

对于以上 SQL，String 类型 pt 列 IN INT 类型常量，都会转为 Double 进行比较。即使 Project 设置了 odps.sql.udf.strict.mode=true，旧版 MaxCompute 不会报错，所有 pt 都会过滤掉，而 MaxCompute2.0 会直接报错。示例如下：

错误写法：

```
SELECT key FROM srcpt WHERE pt IN (1, 2);
```

报错信息：

```
FAILED: ODPS-0130071:[0,0] Semantic analysis exception - physical
plan generation failed: java.lang.NumberFormatException: ODPS-0123091
:Illegal type cast - In function cast, value 'pt1' cannot be casted
from String to Double.
```

建议避免 String 分区列和 INT 类型常量比较，将 INT 类型常量改成 String 类型。

having.use.select.alias

SQL 规范定义 Group by + Having 子句是 Select 子句之前阶段，所以 Having 中不应该使用 Select 子句生成的 Column alias，示例如下：

错误写法：

```
SELECT id id2 FROM DUAL GROUP BY id HAVING id2 > 0;
```

报错信息：

```
FAILED: ODPS-0130071:[1,44] Semantic analysis exception - column id2
 cannot be resolvedODPS-0130071:[1,44] Semantic analysis exception -
column reference id2 should appear in GROUP BY key
```

其中 id2 为 Select 子句中新生成的 Column alias，不应该在 Having 子句中使用。

dynamic.pt.to.static

MaxCompute2.0 动态分区某些情况会被优化器转换成静态分区处理，示例如下：

```
INSERT OVERWRITE TABLE srcpt PARTITION(pt) SELECT id, 'pt1' FROM dual;
```

会被转化成

```
INSERT OVERWRITE TABLE srcpt PARTITION(pt='pt1') SELECT id FROM dual;
```

如果用户指定的分区值不合法，比如错误的使用了'${bizdate}'，MaxCompute2.0 语法检查阶段便会报错。详情请参见分区 。

错误写法：

```
INSERT OVERWRITE TABLE srcpt PARTITION(pt) SELECT id, '${bizdate}'
FROM dual LIMIT 0;
```

报错信息：

```
FAILED: ODPS-0130071:[1,24] Semantic analysis exception - wrong
columns count 2 in data source, requires 3 columns (includes dynamic
partitions if any)
```

旧版 MaxCompute 因为 LIMIT 0，SQL 最终没有输出任何数据，动态分区不会创建，所以最终不报错。

lot.not.in.subquery

In subquery 中 null 值的处理问题。

在标准 SQL 的 IN 运算中，如果后面的值列表中出现 null，则返回值不会出现 false，只可能是 null 或者 true。如 1 in (null, 1, 2, 3) 为 true，而 1 in (null, 2, 3) 为 null，null in (null, 1, 2, 3) 为 null。同理 not in 操作在列表中有 null 的情况下，只会返回 false 或者 null，不会出现 true。

MaxCompute2.0 会用标准的行为进行处理，收到此提醒的用户请注意检查您的查询，IN 操作中的子查询中是否会出现空值，出现空值时行为是否与您预期相符，如果不符合预期请做相应的修改。示例如下：

```
select * from t where c not in (select accepted from c_list);
```

若 accepted 中不会出现 null 值，则此问题可忽略。若出现空值，则 c not in (select accepted from c_list) 原先返回 true，则新版本返回 null。

正确改法：

```
select * from t where c not in (select accepted from c_list where
accepted is not null)
```

6 长周期指标的计算优化方案

实验背景

电子商务公司（如淘宝）对用户数据分析的角度和思路可谓是应有尽有、层出不穷，所以在电商数据仓库和商业分析场景中，经常需要计算最近 N 天的访客数、购买用户数、老客数等类似的指标。

这些指标有一个共同点：都需要根据用户在电商平台上（或网上店铺）一段时间积累的数据进行计算（这里讨论的前提是数据都存储在 MaxCompute 上）。

一般情况下，这些指标的计算方式就是从日志明细表中计算就行了，如下代码计算商品最近 30 天的访客数：

```
select item_id --商品id
  ,count(distinct visitor_id) as ipv_uv_1d_001
from 用户访问商品日志明细表
where ds <= ${bdp.system.bizdate}
and ds >=to_char(dateadd(to_date(${bdp.system.bizdate},'yyyymmdd'),-29
,'dd'),'yyyymmdd')
 group by item_id;
```

说明：

代码中的变量都是DataWorks的调度变量，仅适用于DataWorks的调度任务。为了方便后文不再提醒。

当每天的日志量很大时，上面代码存在一个严重的问题，需要的 **Map Instance** 个数太多，甚至会超过 **99999** 个 Instance 个数的限制，Map Task 就没有办法顺利执行，更别说后续的操作了。

为什么 Instance 个数需要那么多呢？是因为每天的日志数据很大，30 天的数据量更是惊人。此时 Select 操作需要大量的 Map Instance，结果超过了 Instance 的上限，导致代码无法运行。

实验目的

如何计算长周期的指标，又不影响性能呢？通常有以下两种思路：

- 多天汇总的问题根源是数据量的问题，如果把数据量给降低了，便可解决此问题。
- 减少数据量最直接的办法是把每天的数据量都给减少，因此需要构建临时表，对 1d 的数据进行轻度汇总，这样便可去掉很多重复数据，减少数据量。

实验方案

操作步骤

1. 构建中间表，每天汇总一次。

   比如对于上面的例子，可以构建一个 item_id+visitor_id 粒度的中间表。即构建item_id+visitior_id 粒度的日汇总表，记作 A。如下所示：

```
insert overwrite table mds_itm_vsr_xx(ds='${bdp.system.bizdate} ')
select item_id,visitor_id,count(1) as pv
  from
  (
  select item_id,visitor_id
  from 用户访问商品日志明细表
  where ds =${bdp.system.bizdate}
  group by item_id,visitor_id
  )a;
```

2. 计算多天的数据，依赖中间表进行汇总。

   对 A 进行 30 天的汇总，如下所示：

```
select item_id
       ,count(distinct visitor_id) as uv
       ,sum(pv) as pv
  from mds_itm_vsr_xx
  where ds <= '${bdp.system.bizdate} '
  and ds >= to_char(dateadd(to_date('${bdp.system.bizdate} ','
yyyymmdd'),-29,'dd'),'yyyymmdd')
  group by item_id;
```

影响及思考

上面讲述的方法，对每天的访问日志明细数据进行单天去重，从而减少了数据量，提高了性能。缺点是每次计算多天的数据的时候，都需要读取 N 个分区的数据。

那么是否有一种方式，不需要读取 N 个分区的数据，而是把 N 个分区的数据压缩合并成一个分区的数据，让一个分区的数据包含历史数据的信息呢？

业务上是有类似场景的，可以通过 **增量累计方式计算长周期指标**。

场景示例

求最近 1 天店铺商品的老买家数。老买家数的算法定义为：过去一段时间有购买的买家（比如过去 30 天）。

一般情况下，老买家数计算方式如下所示：

```
select item_id --商品id
       ,buyer_id as old_buyer_id
from 用户购买商品明细表
```

```
where ds < ${bdp.system.bizdate}
and ds >=to_char(dateadd(to_date(${bdp.system.bizdate},'yyyymmdd'),-29
,'dd'),'yyyymmdd')
group by item_id
        ,buyer_id;
```

改进思路：

- 维护一张店铺商品和买家购买关系的维表记作表 A，记录买家和店铺的购买关系，以及第一次购买时间，最近一次购买时间，累计购买件数，累计购买金额等信息。
- 每天使用最近 1 天的支付明细日志更新表 A 的相关数据。
- 计算老买家时，最需要判断最近一次购买时间是否是 30天 之内就行了，从而做到最大程度上的数据关系对去重，减少了计算输入数据量。

7 计算长尾调优

长尾问题是分布式计算中最常见的问题之一，也是典型的疑难杂症。究其原因，是因为数据分布不均，导致各个节点的工作量不同，整个任务需要等最慢的节点完成才能结束。

处理这类问题的思路就是把工作分给多个 Worker 去执行，而不是一个 Worker 单独运行最重的那份工作。本文将为您介绍平时工作中遇到的一些典型的长尾问题的场景及其解决方案。

Join 长尾

问题原因：

Join 出现长尾，是因为 Join 时出现某个 Key 里的数据特别多的情况。

解决办法：

您可以从以下四方面进行考虑：

- 排除两张表都是小表的情况，若两张表里有一张大表和一张小表，可以考虑使用 mapjoin，对小表进行缓存，具体的语法和说明请参见*Select*语法介绍。如果是 MapReduce 作业，可以使用资源表的功能，对小表进行缓存。
- 但是如果两张表都比较大，就需要先尽量去重。
- 若还是不能解决，就需要从业务上考虑，为什么会有这样的两个大数据量的 Key 要做笛卡尔积，直接考虑从业务上进行优化。
- 小表leftjoin大表，odps直接leftjoin较慢。此时可以先小表和大表mapjoin，这样能拿到小表和大表的交集中间表，且这个中间表一定是不大于大表的（只要不是有很大的key倾斜就不会膨胀的很大）。然后小表再和这个中间表进行leftjoin，这样效果等于小表leftjoin大表。

Group By 长尾

问题原因：

Group By Key 出现长尾，是因为某个 Key 内的计算量特别大。

解决办法：

您可以通过以下两种方法解决：

- 可对 SQL 进行改写，添加随机数，把长 Key 进行拆分。如下所示：

```
SELECT Key,COUNT(*) AS Cnt FROM TableName GROUP BY Key;
```

不考虑 Combiner，M 节点会 Shuffle 到 R 上，然后 R 再做 Count 操作。对应的执行计划是 M > R。但是如果对长尾的 Key 再做一次工作再分配，就变成如下语句：

```
-- 假设长尾的Key已经找到是KEY001
SELECT a.Key
  , SUM(a.Cnt) AS Cnt
FROM (
  SELECT Key
    , COUNT(*) AS Cnt
FROM TableName
GROUP BY Key,
  CASE
    WHEN Key = 'KEY001' THEN Hash(Random()) % 50
    ELSE 0
   END
) a
GROUP BY a.Key;
```

由上可见，这次的执行计划变成了 M > R > R。虽然执行的步骤变长了，但是长尾的 Key 经过 2 个步骤的处理，整体的时间消耗可能反而有所减少。

 说明：

若数据的长尾并不严重，用这种方法人为地增加一次 R 的过程，最终的时间消耗可能反而更大。

- 使用通用的优化策略 — 系统参数，设置如下：

```
set odps.sql.groupby.skewindata=true。
```

但是通用性的优化策略无法针对具体的业务进行分析，得出的结果不总是最优的。您可以根据实际的数据情况，用更加高效的方法来改写 SQL。

Distinct 长尾

对于 Distinct，上述 Group By 长尾时，把长 Key 进行拆分的策略已经不生效了。对这种场景，您可以考虑通过其他方式解决。

解决办法：

```
--原始SQL,不考虑Uid为空
SELECT COUNT(uid) AS Pv
    , COUNT(DISTINCT uid) AS Uv
```

```
FROM UserLog;
```

可以改写成如下语句：

```
SELECT SUM(PV) AS Pv
    , COUNT(*) AS UV
FROM (
    SELECT COUNT(*) AS Pv
      , uid
    FROM UserLog
    GROUP BY uid
) a;
```

该解法是把 Distinct 改成了普通的 Count，这样的计算压力不会落到同一个 Reducer 上。而且这样改写后，既能支持前面提到的 Group By 优化，系统又能做 Combiner，性能会有较大的提升。

动态分区长尾

问题原因：

- 动态分区功能为了整理小文件，会在最后启一个 Reduce，对数据进行整理，所以如果使用动态分区写入数据时若有倾斜，就会发生长尾。
- 一般情况下，滥用动态分区的功能也是产生这类长尾的一个常见原因。

解决办法：

若写入的数据已经确定需要把数据写入某个具体分区，那可以在 Insert 的时候指定需要写入的分区，而不是使用动态分区。

通过 Combiner 解决长尾

对于 MapRedcuce 作业，使用 Combiner 是一种常见的长尾优化策略。在 WordCount 的示例中，已提到这种做法。通过 Combiner，减少 Mapper Shuffle 往 Reducer 的数据，可以大大减少网络传输的开销。对于 MaxCompute SQL，这种优化会由系统自动完成。

 说明：

Combiner 只是 Map 端的优化，需要保证是否执行 Combiner 的结果是一样的。以 WordCount 为例，传 2 个 (KEY,1) 和传 1 个 (KEY,2) 的结果是一样的。但是比如在做平均值时，便不能直接在 Combiner 里把 (KEY,1) 和 (KEY,2) 合并成 (KEY,1.5)。

通过系统优化解决长尾

针对长尾这种场景，除了前面提到的 Local Combiner，MaxCompute 系统本身还做了一些优化。比如在跑任务的时候，日志里突然打出如下的内容（+N backups 部分）：

```
M1_Stg1_job0:0/521/521[100%] M2_Stg1_job0:0/1/1[100%] J9_1_2_Stg5_job0
:0/523/523[100%] J3_1_2_Stg1_job0:0/523/523[100%] R6_3_9_Stg2_job0:1/
1046/1047[100%]
M1_Stg1_job0:0/521/521[100%] M2_Stg1_job0:0/1/1[100%] J9_1_2_Stg5_job0
:0/523/523[100%] J3_1_2_Stg1_job0:0/523/523[100%] R6_3_9_Stg2_job0:1/
1046/1047[100%]
M1_Stg1_job0:0/521/521[100%] M2_Stg1_job0:0/1/1[100%] J9_1_2_Stg5_job0
:0/523/523[100%] J3_1_2_Stg1_job0:0/523/523[100%] R6_3_9_Stg2_job0:1/
1046/1047(+1 backups)[100%]
M1_Stg1_job0:0/521/521[100%] M2_Stg1_job0:0/1/1[100%] J9_1_2_Stg5_job0
:0/523/523[100%] J3_1_2_Stg1_job0:0/523/523[100%] R6_3_9_Stg2_job0:1/
1046/1047(+1 backups)[100%]
```

可以看到 1047 个 Reducer，有 1046 个已经完成了，但是最后一个一直没完成。系统识别出这种情况后，自动启动了一个新的 Reducer，跑一样的数据，然后看两个哪个快，取快的数据归并到最后的结果集里。

通过业务优化解决长尾

虽然前面的优化策略有很多，但仍然不能解决所有问题。有时碰到的长尾问题，还需要从业务角度上去考虑是否有更好的解决方法，示例如下：

- 实际数据可能包含非常多的噪音。比如：需要根据访问者的 ID 进行计算，看每个用户的访问记录的行为。需要先去掉爬虫的数据（现在的爬虫已越来越难识别），否则爬虫数据很容易长尾计算的长尾。类似的情况还有根据 xxid 进行关联的时候，需要考虑这个关联字段是否存在为空的情况。
- 一些业务特殊情况。比如：ISV 的操作记录，在数据量、行为方式上都会和普通的个人会有很大的区别。那么可以考虑针对大客户，使用特殊的分析方式进行单独处理。
- 数据分布不均匀的情况下，不要使用常量字段做 Distribute by 字段来实现全排序。

8 快速掌握SQL写法

本文通过课程实践的方式，为您介绍MaxCompute SQL，让您快速掌握SQL的写法，并清楚MaxCompute SQL和标准SQL的区别，请结合 *MaxCompute SQL* 基础文档 进行阅读。

数据集准备

这里选择大家比较熟悉的Emp/Dept表做为数据集。为方便大家操作，特提供相关的MaxCompute建表语句和数据文件（*emp*表数据文件，*dept*表数据文件），您可自行在MaxCompute项目上创建表并上传数据。

创建emp表的DDL语句，如下所示：

```
CREATE TABLE IF NOT EXISTS emp (
  EMPNO string ,
  ENAME string ,
  JOB string ,
  MGR bigint ,
  HIREDATE datetime ,
  SAL double ,
  COMM double ,
  DEPTNO bigint );
```

创建 dept 表的 DDL 语句，如下所示：

```
CREATE TABLE IF NOT EXISTS dept (
  DEPTNO bigint ,
  DNAME string ,
  LOC string);
```

SQL操作

初学SQL常遇到的问题点

- 使用Group by，那么Select的部分要么是分组项，要么就得是聚合函数。
- Order by后面必须加Limit n。
- Select表达式中不能用子查询，可以改写为Join。
- Join不支持笛卡尔积，以及MapJoin的用法和使用场景。
- Union all需要改成子查询的格式。
- In/Not in语句对应的子查询只能有一列，而且返回的行数不能超过1000，否则也需要改成Join。

编写SQL进行解题

题目一：列出至少有一个员工的所有部门。

为了避免数据量太大的情况下导致 常遇问题点 中的第6点，您需要使用Join 进行改写。如下所示：

```
SELECT d.*
FROM dept d
JOIN (
    SELECT DISTINCT deptno AS no
    FROM emp
) e
ON d.deptno = e.no;
```

题目二：列出薪金比**SMITH**多的所有员工。

MapJoin的典型场景，如下所示：

```
SELECT /*+ MapJoin(a) */ e.empno
    , e.ename
    , e.sal
FROM emp e
JOIN (
    SELECT MAX(sal) AS sal
    FROM `emp`
    WHERE `ENAME` = 'SMITH'
) a
ON e.sal > a.sal;
```

题目三：列出所有员工的姓名及其直接上级的姓名。

非等值连接，如下所示：

```
SELECT a.ename
    , b.ename
FROM emp a
LEFT OUTER JOIN emp b
ON b.empno = a.mgr;
```

题目四：列出最低薪金大于**1500**的各种工作。

Having 的用法，如下所示：

```
SELECT emp.`JOB`
    , MIN(emp.sal) AS sal
FROM `emp`
GROUP BY emp.`JOB`
HAVING MIN(emp.sal) > 1500;
```

题目五：列出在每个部门工作的员工数量、平均工资和平均服务期限。

时间处理上有很多好用的内建函数，如下所示：

```
SELECT COUNT(empno) AS cnt_emp
```

```
    , ROUND(AVG(sal), 2) AS avg_sal
    , ROUND(AVG(datediff(getdate(), hiredate, 'dd')), 2) AS avg_hire
FROM `emp`
GROUP BY `DEPTNO`;
```

题目六：列出每个部门的薪水前**3**名的人员的姓名以及他们的名次（**Top n**的需求非常常见）。

SQL 语句如下所示：

```
SELECT *
FROM (
  SELECT deptno
    , ename
    , sal
    , ROW_NUMBER() OVER (PARTITION BY deptno ORDER BY sal DESC) AS
nums
  FROM emp
) emp1
WHERE emp1.nums < 4;
```

题目七：用一个**SQL**写出每个部门的人数、**CLERK**（办事员）的人数占该部门总人数占比。

SQL语句如下所示：

```
SELECT deptno
    , COUNT(empno) AS cnt
    , ROUND(SUM(CASE
      WHEN job = 'CLERK' THEN 1
      ELSE 0
    END) / COUNT(empno), 2) AS rate
FROM `EMP`
GROUP BY deptno;
```

9 Hadoop数据迁移MaxCompute最佳实践

本文向您详细介绍如何通过使用DataWorks数据同步功能，将HDFS上的数据迁移到阿里云MaxCompute大数据计算服务上或从MaxCompute将数据迁移到HDFS。无论您是使用Hadoop还是Spark，均可以参考本文进行与MaxCompute之间的数据双向同步。

环境准备

1. Hadoop集群搭建

   进行数据迁移前，您需要保证自己的Hadoop集群环境正常。本文使用阿里云EMR服务自动化搭建Hadoop集群，详细过程请参见：创建集群。

   本文使用的EMR Hadoop版本信息如下：

   EMR版本: EMR-3.11.0

   集群类型: HADOOP

   软件信息: HDFS2.7.2 / YARN2.7.2 / Hive2.3.3 / Ganglia3.7.2 / Spark2.2.1 / HUE4.1.0 / Zeppelin0.7.3 / Tez0.9.1 / Sqoop1.4.6 / Pig0.14.0 / ApacheDS2.0.0 / Knox0.13.0

   Hadoop集群使用经典网络，区域为华东1（杭州），主实例组ECS计算资源配置公网及内网IP，高可用选择为否（非HA模式），具体配置如下所示。

2. MaxCompute

   请参考：开通*MaxCompute*。

开通MaxCompute服务并创建好项目，本文中在华东1（杭州）区域创建项目bigdata_DOC，同时启动DataWorks相关服务，如下所示。

数据准备

1. Hadoop集群创建测试数据

   进入EMR Hadoop集群控制台界面，使用交互式工作台，新建交互式任务doc。本例中HIVE建表语句：

```
CREATE TABLE IF NOT
EXISTS hive_doc_good_sale(
   create_time timestamp,
   category STRING,
   brand STRING,
   buyer_id STRING,
   trans_num BIGINT,
   trans_amount DOUBLE,
   click_cnt BIGINT
   )
   PARTITIONED BY (pt string) ROW FORMAT
```

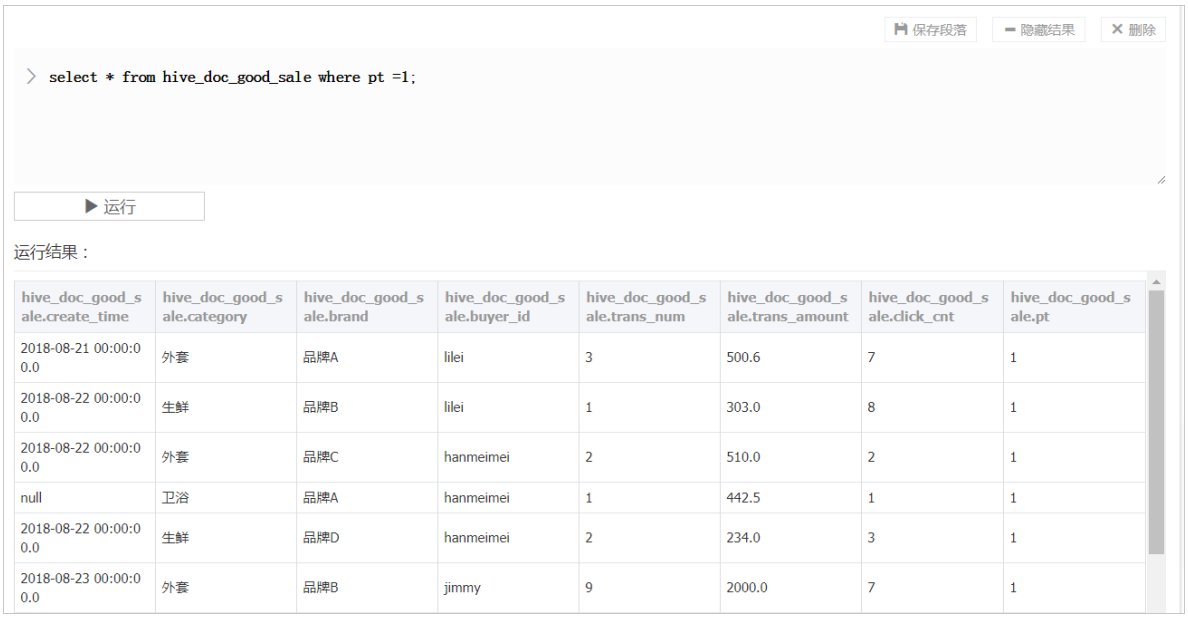
```
DELIMITED FIELDS TERMINATED BY ',' lines terminated by '\n'
```

选择运行，观察到Query executed successfully提示则说明成功在EMR Hadoop集群上创建了测试用表格hive_doc_good_sale，如下图所示。

插入测试数据，您可以选择从OSS或其他数据源导入测试数据，也可以手动插入少量的测试数据。本文中手动插入数据如下：

```
insert into
hive_doc_good_sale PARTITION(pt =1 ) values('2018-08-21','外套','品牌A','lilei',3,500.6,7),('2018-08-22','生鲜','品牌B','lilei',1,303,8),('2018-08-22','外套','品牌C','hanmeimei',2,510,2),(2018-08-22,'卫浴','品牌A','hanmeimei',1,442.5,1),('2018-08-22','生鲜','品牌D','hanmeimei',2,234,3),('2018-08-23','外套','品牌B','jimmy',9,2000,7),('2018-08-23','生鲜','品牌A','jimmy',5,45.1,5),('2018-08-23','外套','品牌E','jimmy',5,100.2,4),('2018-08-24','生鲜','品牌G','peiqi',10,5560,7),('2018-08-24','卫浴','品牌F','peiqi',1,445.6,2),('2018-08-24','外套','品牌A','ray',3,777,3),('2018-08-24','卫浴','品牌G','ray',3,122,3),('2018-08-24','外套','品牌C','ray',1,62,7) ;
```

完成插入数据后，您可以使用select * from hive_doc_good_sale where pt =1;语句检查Hadoop集群表中是否已存在数据可用于迁移：



hive_doc_good_sale.create_time	hive_doc_good_sale.category	hive_doc_good_sale.brand	hive_doc_good_sale.buyer_id	hive_doc_good_sale.trans_num	hive_doc_good_sale.trans_amount	hive_doc_good_sale.click_cnt	hive_doc_good_sale.pt
2018-08-21 00:00:00.0	外套	品牌A	lilei	3	500.6	7	1
2018-08-22 00:00:00.0	生鲜	品牌B	lilei	1	303.0	8	1
2018-08-22 00:00:00.0	外套	品牌C	hanmeimei	2	510.0	2	1
null	卫浴	品牌A	hanmeimei	1	442.5	1	1
2018-08-22 00:00:00.0	生鲜	品牌D	hanmeimei	2	234.0	3	1
2018-08-23 00:00:00.0	外套	品牌B	jimmy	9	2000.0	7	1

2. 利用DataWorks新建目标表

在管理控制台，选择对应的MaxCompute项目，点击进入数据开发页面，点击新建表，如下所示。

在弹框中输入SQL建表语句，本例中使用的建表语句如下：

```
CREATE TABLE IF NOT EXISTS hive_doc_good_sale(
   create_time string,
   category STRING,
   brand STRING,
   buyer_id STRING,
   trans_num BIGINT,
   trans_amount DOUBLE,
   click_cnt BIGINT
   )
```

```
    PARTITIONED BY (pt string) ;
```

在建表过程中，需要考虑到HIVE数据类型与MaxCompute数据类型的映射，当前数据映射关系可参见：与*Hive*数据类型映射表。

由于本文使用DataWorks进行数据迁移，而DataWorks数据同步功能当前暂不支持timestamp类型数据，因此在DataWorks建表语句中，将create_time设置为string类型。 上述步骤同样可通过odpscmd命令行工具完成，命令行工具安装和配置请参考：安装并配置客户端。执行过程如下所示：

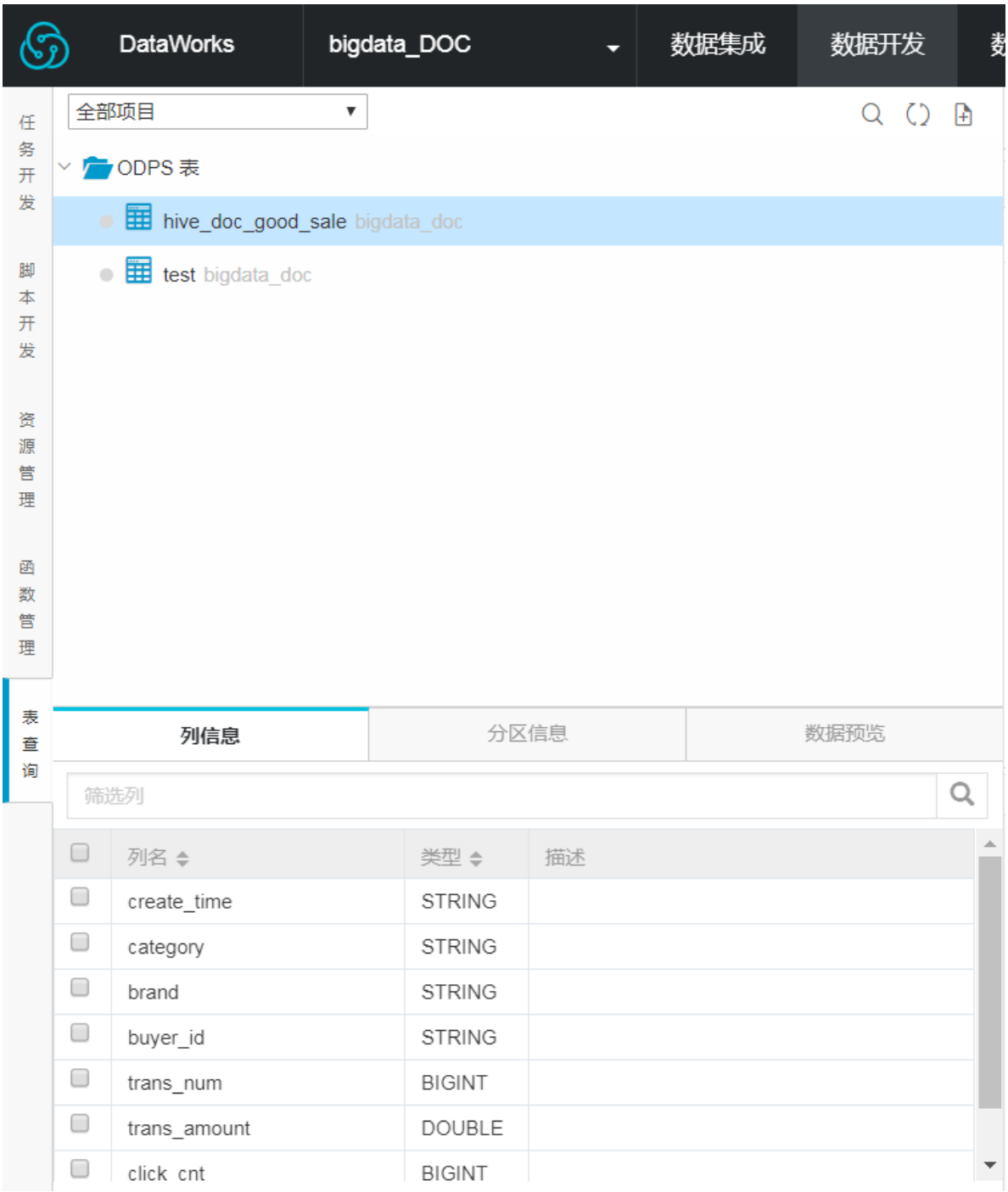
```
odps@ bigdata_DOC>CREATE TABLE IF NOT EXISTS hive_doc_good_sale(
   create_time timestamp,
   category STRING,
   brand STRING,
   buyer_id STRING,
   trans_num BIGINT,
   trans_amount DOUBLE,
   click_cnt BIGINT
   )
   PARTITIONED BY (pt string) ;
                >                >               >                >
        >               >               >               >
 >
ID = 20180906110540873gev1bpim
OK
odps@ bigdata_DOC>drop table hive_doc_good_sale;
Confirm to "drop table hive_doc_good_sale;" (yes/no)? yes

ID = 20180906110825180gxh66292
```

说明：

考虑到部分HIVE与MaxCompute数据类型的兼容问题，建议在odpscmd客户端上执行以下命令。

```
set odps.sql.type.system.odps2=true;set
odps.sql.hive.compatible=true;
```

完成建表后，可在DataWorks数据开发>表查询一栏查看到当前创建的MaxCompute上的表，如下所示。



数据同步

1. 新建自定义资源组

   由于MaxCompute项目所处的网络环境与Hadoop集群中的数据节点（data node）网络通常不可达，我们可通过自定义资源组的方式，将DataWorks的同步任务运行在Hadoop集群的Master节点上（Hadoop集群内Master节点和数据节点通常可达）。

   a. 查看Hadoop集群datanode

在EMR控制台上首页/集群管理/集群/主机列表页查看，如下图所示，通常非HA模式的EMR上Hadoop集群的master节点主机名为 emr-header-1，datanode主机名为emr-worker-X。

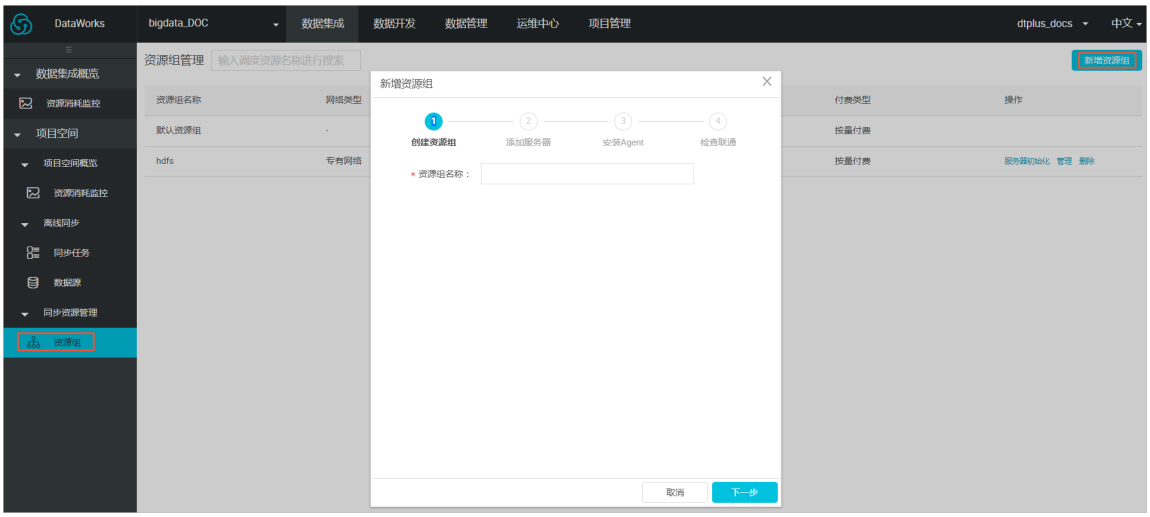
您也可以通过点击上图中Master节点的ECS ID，进入ECS实例详情页，通过点击远程连接进入ECS，通过 hadoop dfsadmin －report命令查看datenode，如下图所示。

```
DFS Used%: 0.05%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
Missing blocks (with replication factor 1): 0

-------------------------------------------------
Live datanodes (2):

Name: 10.31.122.189:50010 (emr-worker-1.cluster-74503)
Hostname: emr-worker-1.cluster-74503
Decommission Status : Normal
Configured Capacity: 333373341696 (310.48 GB)
DFS Used: 155725824 (148.51 MB)
Non DFS Used: 325541888 (310.46 MB)
DFS Remaining: 332892073984 (310.03 GB)
DFS Used%: 0.05%
DFS Remaining%: 99.86%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Thu Sep 06 19:41:01 CST 2018


Name: 10.81.78.209:50010 (emr-worker-2.cluster-74503)
Hostname: emr-worker-2.cluster-74503
Decommission Status : Normal
Configured Capacity: 333373341696 (310.48 GB)
DFS Used: 155725824 (148.51 MB)
Non DFS Used: 325451776 (310.38 MB)
DFS Remaining: 332892164096 (310.03 GB)
DFS Used%: 0.05%
DFS Remaining%: 99.86%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Thu Sep 06 19:41:02 CST 2018
```

由上图可以看到，在本例中，datanode只具有内网地址，很难与DataWorks默认资源组互通，所以我们需要设置自定义资源组，将master node设置为执行DataWorks数据同步任务的节点。

b. 新建自定义资源组

进入DataWorks数据集成页面，选择资源组，点击新增资源组，如下图所示。关于自定义资源组的详细信息请参考新增调度资源。



在添加服务器步骤中，需要输入ECS UUID和机器IP等信息（对于经典网络类型，需输入服务器名称，对于专有网络类型，需输入服务器UUID。目前仅DataWorks V2.0 华东2区支持经典网络类型的调度资源添加，对于其他区域，无论您使用的是经典网络还是专有网络类型，在添加调度资源组时都请选择专有网络类型），机器IP需填写master node公网IP（内网IP有可能不可达）。ECS的UUID需要进入master node管理终端，通过命令dmidecode | grep UUID获取（如果您的hadoop集群并非搭建在EMR环境上，也可以通过该命令获取），如下所示：

完成添加服务器后，需保证master node与DataWorks网络可达，如果您使用的是ECS服务器，需设置服务器安全组。如果您使用的内网IP互通，可参考添加安全设置。

如果您使用的是公网IP，可直接设置安全组公网出入方向规则，本文中设置公网入方向放通所有端口（实际应用场景中，为了您的数据安全，强烈建议设置详细的放通规则），如下图所示。

完成上述步骤后，按照提示安装自定义资源组agent，观察到当前状态为可用，说明新增自定义资源组成功：

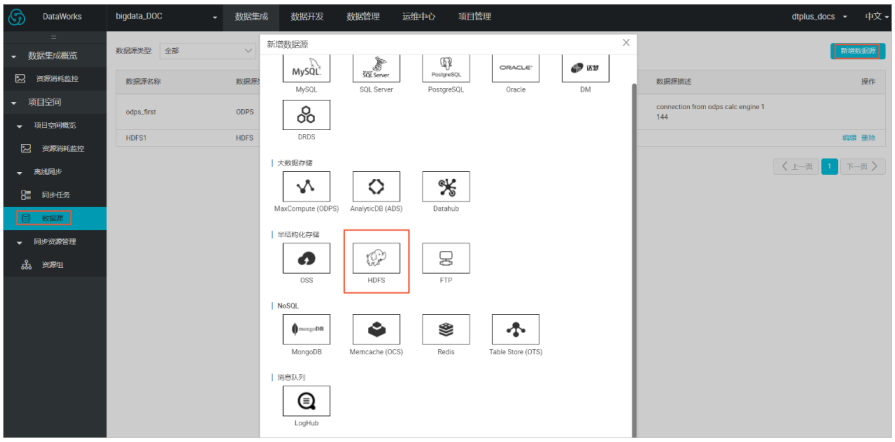
如果状态为不可用，您可以登录master node，使用`tail -f/home/admin/alisatasknode/logs/heartbeat.log`命令查看DataWorks与master node之间心跳报文是否超时，如下图所示。

```
[root@emr-header-1 logs]# hdfs dfs -ls /user/hive/warehouse/hive_doc_good_sale/
Found 1 items
drwxr-x--x   - hive hadoop          0 2018-09-03 17:46 /user/hive/warehouse/hive_doc_good_sale/pt=1
[root@emr-header-1 logs]# tail -f /home/admin/alisatasknode/logs/heartbeat.log
2018-09-06 21:47:34,440 INFO  [pool-6-thread-1] [HeartbeatReporter.java:104] [] - heartbeat start, current status:2
2018-09-06 21:47:34,465 INFO  [pool-6-thread-1] [HeartbeatReporter.java:133] [] - heartbeat end■ cost time:0.025s
2018-09-06 21:47:39,465 INFO  [pool-6-thread-1] [HeartbeatReporter.java:104] [] - heartbeat start, current status:2
2018-09-06 21:47:39,491 INFO  [pool-6-thread-1] [HeartbeatReporter.java:133] [] - heartbeat end■ cost time:0.026s
2018-09-06 21:47:44,491 INFO  [pool-6-thread-1] [HeartbeatReporter.java:104] [] - heartbeat start, current status:2
2018-09-06 21:47:44,515 INFO  [pool-6-thread-1] [HeartbeatReporter.java:133] [] - heartbeat end■ cost time:0.024s
2018-09-06 21:47:49,516 INFO  [pool-6-thread-1] [HeartbeatReporter.java:104] [] - heartbeat start, current status:2
2018-09-06 21:47:49,538 INFO  [pool-6-thread-1] [HeartbeatReporter.java:133] [] - heartbeat end■ cost time:0.022s
2018-09-06 21:47:54,539 INFO  [pool-6-thread-1] [HeartbeatReporter.java:104] [] - heartbeat start, current status:2
2018-09-06 21:47:54,555 INFO  [pool-6-thread-1] [HeartbeatReporter.java:133] [] - heartbeat end■ cost time:0.016s
^C
```

2. 新建数据源

关于DataWorks新建数据源详细步骤，请参见：数据源配置。

DataWorks新建项目后，默认设置自己为数据源odps_first。因此我们只需添加Hadoop集群数据源:在DataWorks数据集成页面，点击数据源>新增数据源，在弹框中选择HDFS类型的数据源：



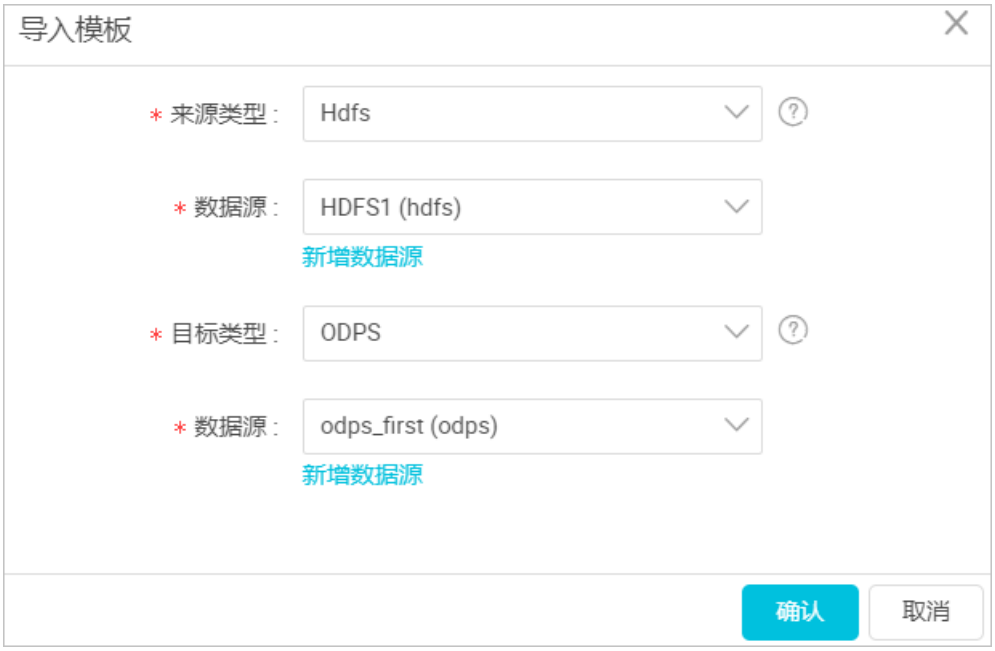
在弹出窗口中填写数据源名称及defaultFS。对于EMR Hadoop集群而言，如果Hadoop集群为HA集群，则此处地址为hdfs://emr-header-1的IP:8020，如果Hadoop集群为非HA集群，则此处地址为hdfs://emr-header-1的IP:9000。在本文中，emr-header-1与DataWorks通过公网连接，因此此处填写公网IP并放通安全组。

完成配置后，点击测试连通性，如果提示“测试连通性成功”，则说明数据源添加正常。

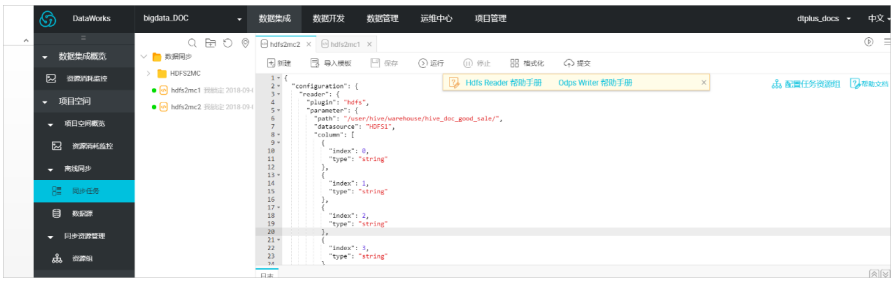
> 说明：
> 如果EMR Hadoop集群设置网络类型为专有网络，则不支持连通性测试。

3. 配置数据同步任务

在DataWorks数据集成页面点击同步任务，选择新建>脚本模式，在导入模板弹窗选择数据源类型如下：



完成导入模板后，同步任务会转入脚本模式，本文中配置脚本如下，相关解释请参见：脚本模式配置。



在配置数据同步任务脚本时，需注意DataWorks同步任务和HIVE表中数据类型的转换如下：

在**Hive**表中的数据类型	**DataX/DataWorks** 内部类型
TINYINT,SMALLINT,INT,BIGINT	Long
FLOAT,DOUBLE,DECIMAL	Double
String,CHAR,VARCHAR	String
BOOLEAN	Boolean
Date,TIMESTAMP	Date
Binary	Binary

详细代码如下：

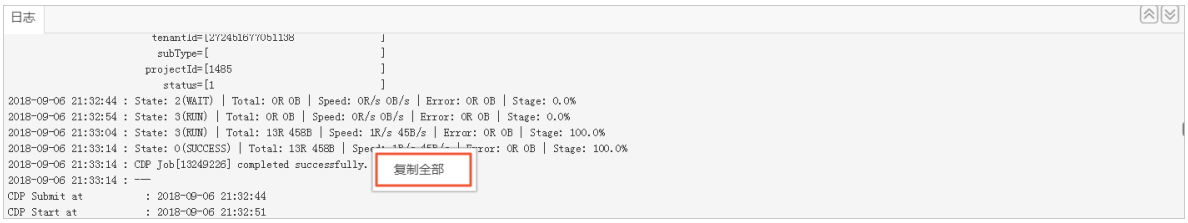
```
{
  "configuration": {
    "reader": {
      "plugin": "hdfs",
      "parameter": {
        "path": "/user/hive/warehouse/hive_doc_good_sale/",
        "datasource": "HDFS1",
        "column": [
          {
            "index": 0,
            "type": "string"
          },
          {
            "index": 1,
            "type": "string"
          },
          {
            "index": 2,
            "type": "string"
          },
          {
            "index": 3,
            "type": "string"
          },
          {
            "index": 4,
            "type": "long"
          },
          {
            "index": 5,
            "type": "double"
```

```
            },
            {
              "index": 6,
              "type": "long"
            }
          ],
          "defaultFS": "hdfs://121.199.11.138:9000",
          "fieldDelimiter": ",",
          "encoding": "UTF-8",
          "fileType": "text"
        }
      },
      "writer": {
        "plugin": "odps",
        "parameter": {
          "partition": "pt=1",
          "truncate": false,
          "datasource": "odps_first",
          "column": [
            "create_time",
            "category",
            "brand",
            "buyer_id",
            "trans_num",
            "trans_amount",
            "click_cnt"
          ],
          "table": "hive_doc_good_sale"
        }
      },
      "setting": {
        "errorLimit": {
          "record": "1000"
        },
        "speed": {
          "throttle": false,
          "concurrent": 1,
          "mbps": "1",
          "dmu": 1
        }
      }
    },
    "type": "job",
    "version": "1.0"
}
```

其中，path参数为数据在Hadoop集群中存放的位置，您可以在登录master node后，使用hdfs dfs –ls /user/hive/warehouse/hive_doc_good_sale命令确认。对于分区表，您可以不指定分区，DataWorks数据同步会自动递归到分区路径，如下图所示。

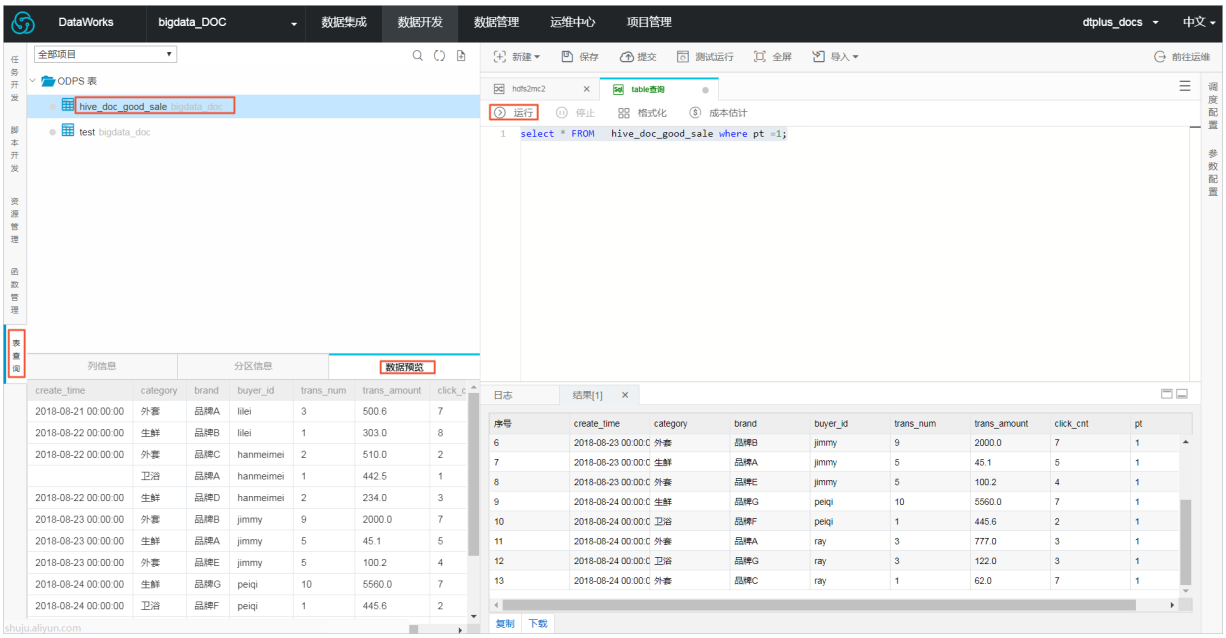
```
[root@emr-header-1 logs]# hdfs dfs -ls /user/hive/warehouse/hive_doc_good_sale/
Found 1 items
drwxr-x--x   - hive hadoop          0 2018-09-03 17:46 /user/hive/warehouse/hive_doc_good_sale/pt=1
```

完成配置后，点击运行。如果提示任务运行成功，则说明同步任务已完成。如果运行失败，可通过复制日志进行进一步排查。



验证结果

在DataWorks数据开发/表查询页面，选择表hive_doc_good_sale后，点击数据预览可查看HIVE数据是否已同步到MaxCompute。您也可以通过新建一个table查询任务，在任务中输入脚本`select * FROM hive_doc_good_sale where pt =1;`后，点击运行来查看表结果，如下图所示。



当然，您也可以通过在odpscmd命令行工具中输入`select * FROM hive_doc_good_sale where pt =1;`查询表结果。

MaxCompute数据迁移到Hadoop

如果您想实现MaxCompute数据迁移到Hadoop。步骤与上述步骤类似，不同的是同步脚本内的reader和writer对象需要对调，具体实现脚本举例如下。

```
{
  "configuration": {
    "reader": {
      "plugin": "odps",
      "parameter": {
      "partition": "pt=1",
      "isCompress": false,
      "datasource": "odps_first",
      "column": [
        "create_time",
```

```
        "category",
        "brand",
      "buyer_id",
      "trans_num",
      "trans_amount",
      "click_cnt"
    ],
    "table": "hive_doc_good_sale"
    }
  },
  "writer": {
    "plugin": "hdfs",
    "parameter": {
    "path": "/user/hive/warehouse/hive_doc_good_sale",
    "fileName": "pt=1",
    "datasource": "HDFS_data_source",
    "column": [
      {
        "name": "create_time",
        "type": "string"
      },
      {
        "name": "category",
        "type": "string"
      },
      {
        "name": "brand",
        "type": "string"
      },
      {
        "name": "buyer_id",
        "type": "string"
      },
      {
        "name": "trans_num",
        "type": "BIGINT"
      },
      {
        "name": "trans_amount",
        "type": "DOUBLE"
      },
      {
        "name": "click_cnt",
        "type": "BIGINT"
      }
    ],
    "defaultFS": "hdfs://47.99.162.100:9000",
    "writeMode": "append",
    "fieldDelimiter": ",",
    "encoding": "UTF-8",
    "fileType": "text"
    }
  },
  "setting": {
    "errorLimit": {
      "record": "1000"
  },
  "speed": {
    "throttle": false,
    "concurrent": 1,
    "mbps": "1",
    "dmu": 1
```

```
  }
  }
},
"type": "job",
"version": "1.0"
}
```

您需要参考配置*HDFS Writer*在运行上述同步任务前对Hadoop集群进行设置，在运行同步任务后手动拷贝同步过去的文件。

10 RDS迁移到MaxCompute实现动态分区

本文向您详细介绍如何使用DataWorks数据集成同步功能，自动创建分区，动态的将RDS中的数据，迁移到MaxCompute大数据计算服务上。

准备工作

1. MaxCompute

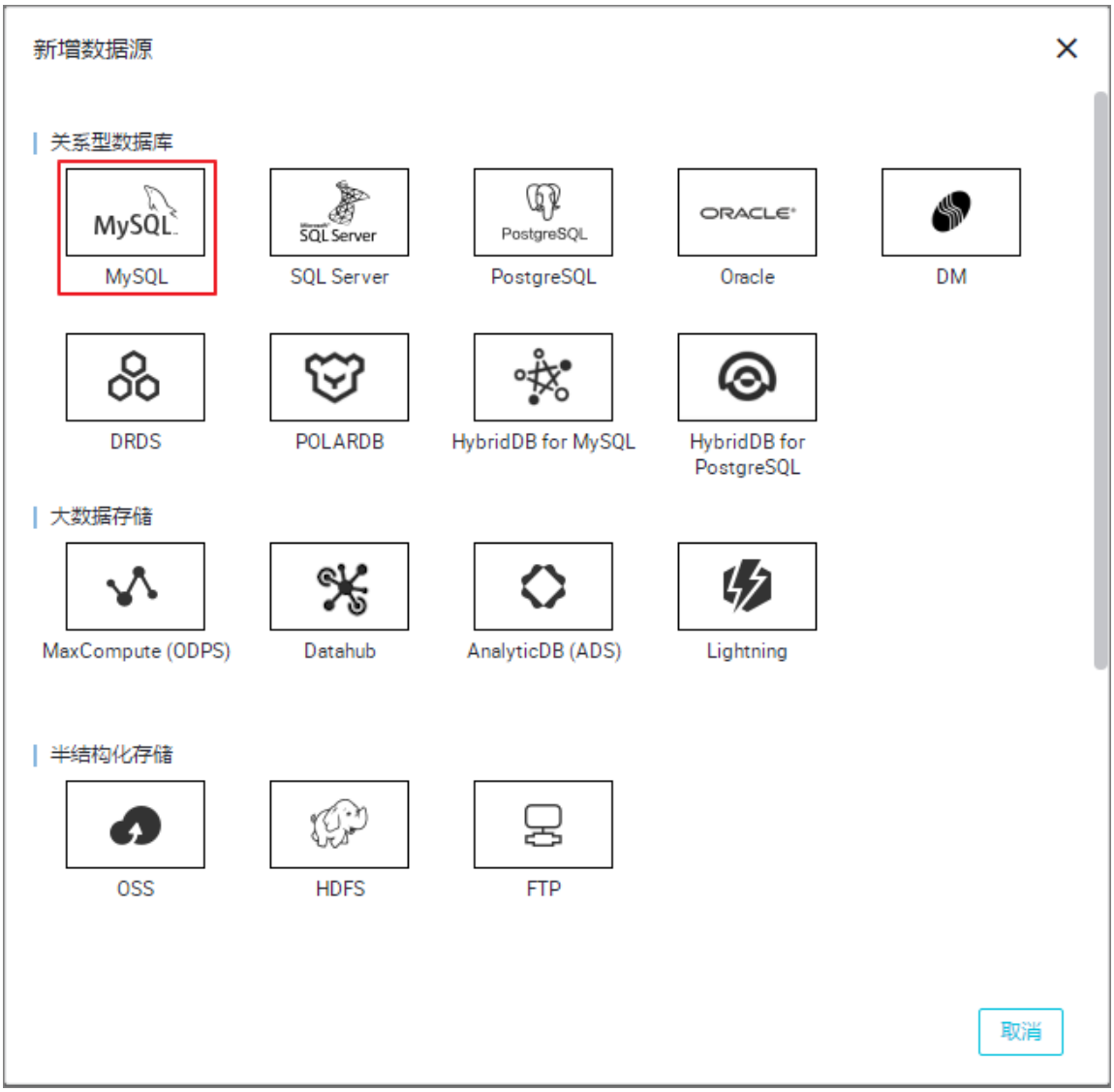
   开通MaxCompute服务并创建工作空间，本文选择的区域是华北2（北京）。同时启动DataWorks相关服务。如下图所示。

> **说明：**
> 如果您是第一次使用DataWorks，请确认已经根据准备工作，准备好账号和项目角色、项目空间等内容，开通MaxCompute请参见开通*MaxCompute*。进入DataWorks管理控制台，单击对应项目后的进入数据开发，开始数据开发操作。

2. 新增数据源

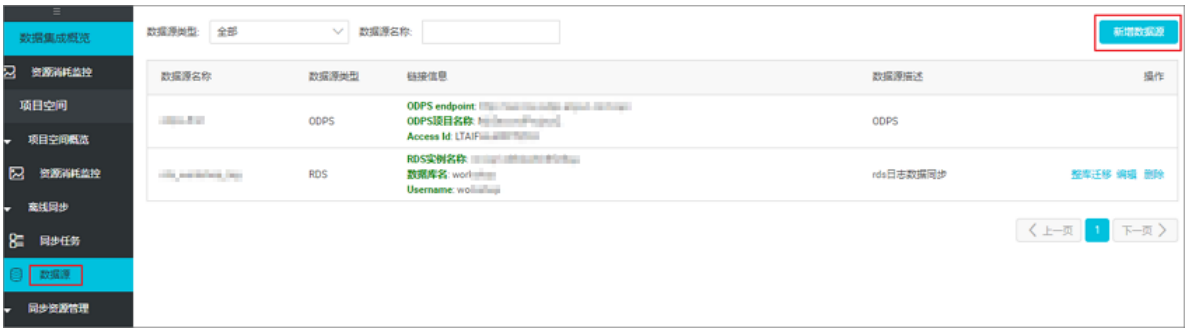
   a. 新增RDS数据源

      进入DataWorks数据集成控制台，新增*MySQL*数据源。 如下图。



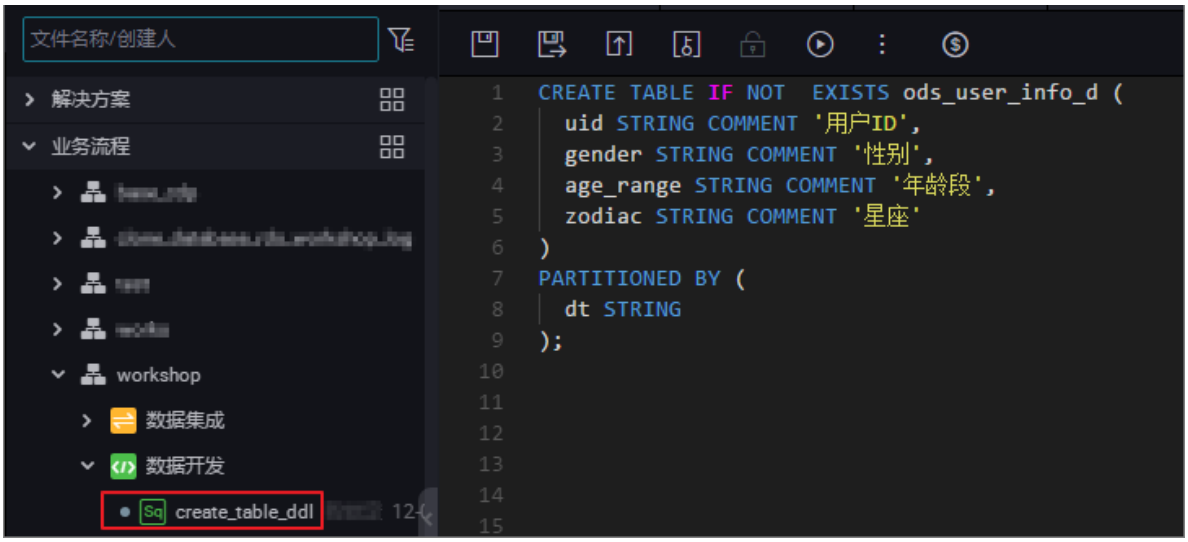
b. 新增ODPS数据源。详细描述请参考文档配置*MaxCompute*数据源。

配置完成后，如下图所示。



3. 确认作为目标的ODPS数据库中有表

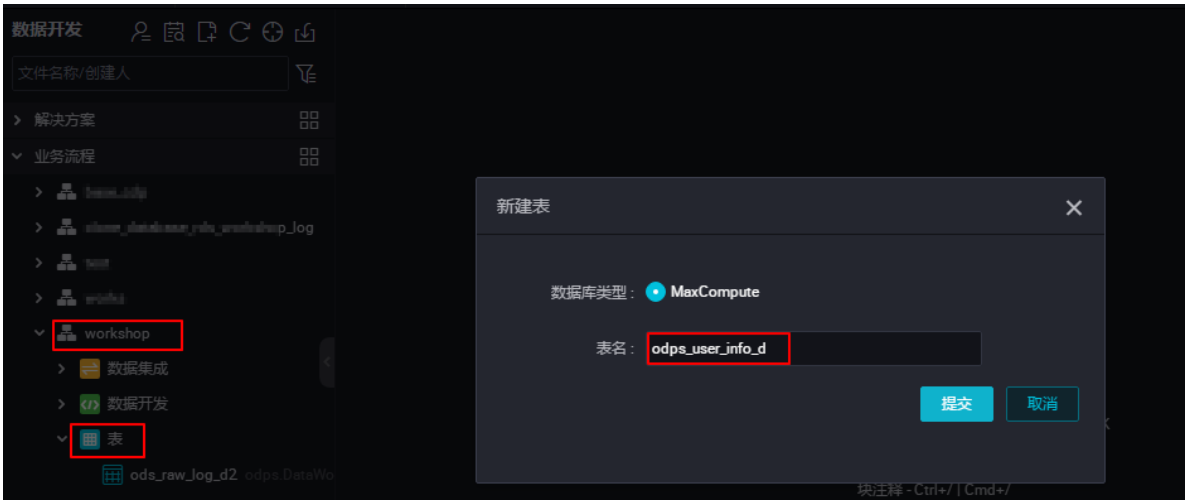
在ODPS数据库中创建RDS对应目标表ods_user_info_d。在数据开发选项下右键单击新建**ODPS SQL**节点create_table_ddl，输入建表语句。如下图。



SQL如下：

```
CREATE TABLE IF NOT EXISTS ods_user_info_d (
uid STRING COMMENT '用户ID',
gender STRING COMMENT '性别',
age_range STRING COMMENT '年龄段',
zodiac STRING COMMENT '星座'
)
PARTITIONED BY (
dt STRING
);
```

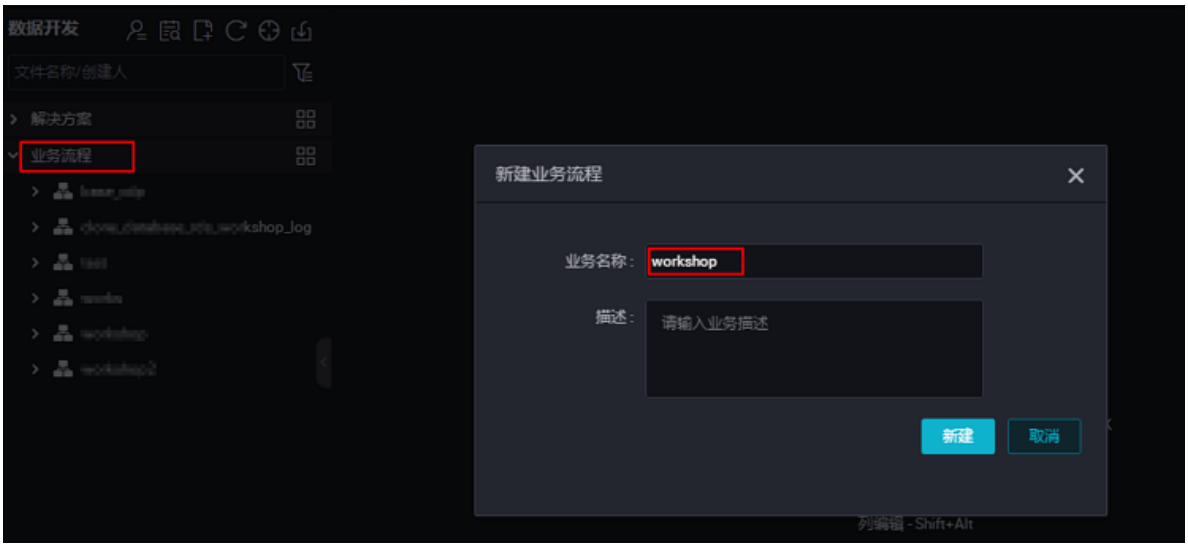
您也可以选择在业务流程 > 表下新建表。如下图。



详细的信息请参见建表并上传数据。

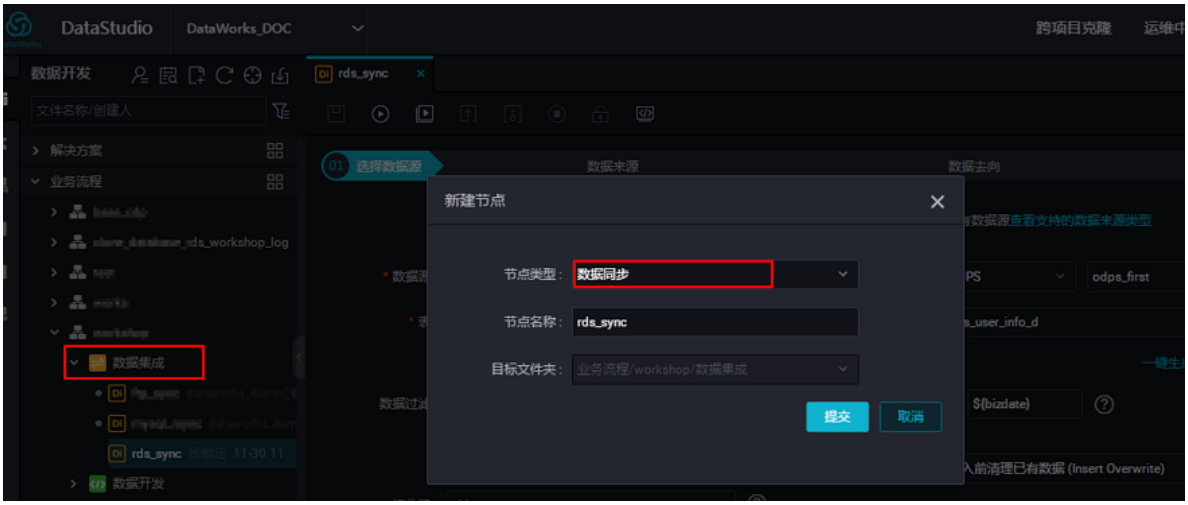
4. 新建业务流程

选择数据开发 > 业务流程下的新建业务流程workshop，如下图。



5. 新建并配置同步任务节点

   在新创建的业务流程workshop下，新建同步节点rds_sync。如下图。



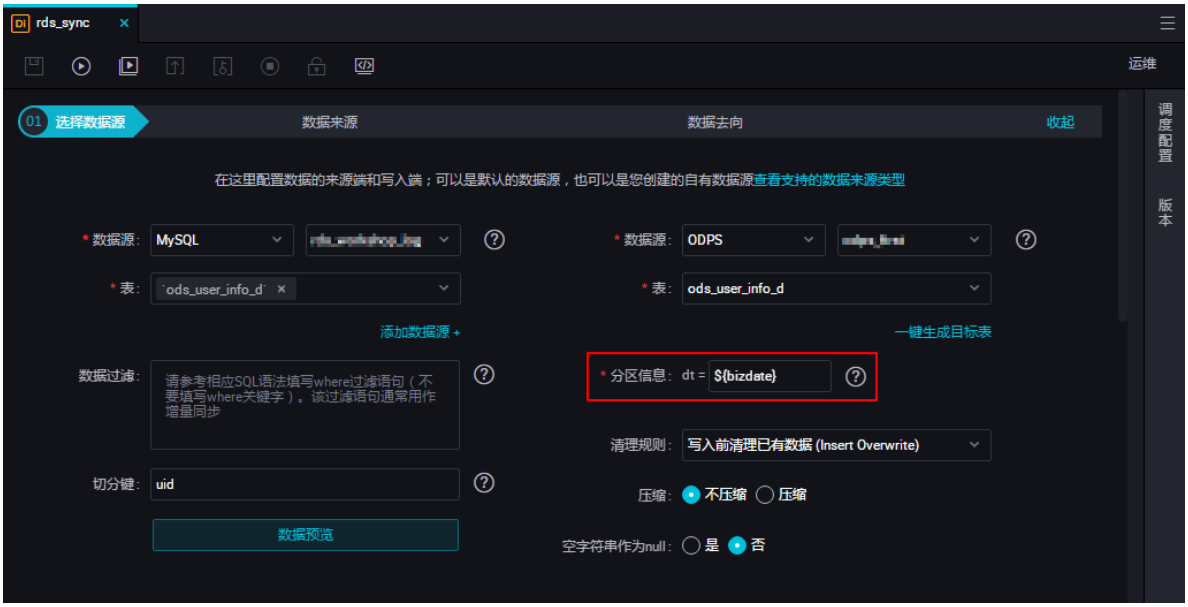
详细的数据同步任务的操作和配置请参见*DataWorks数据开发和运维*。

自动创建分区

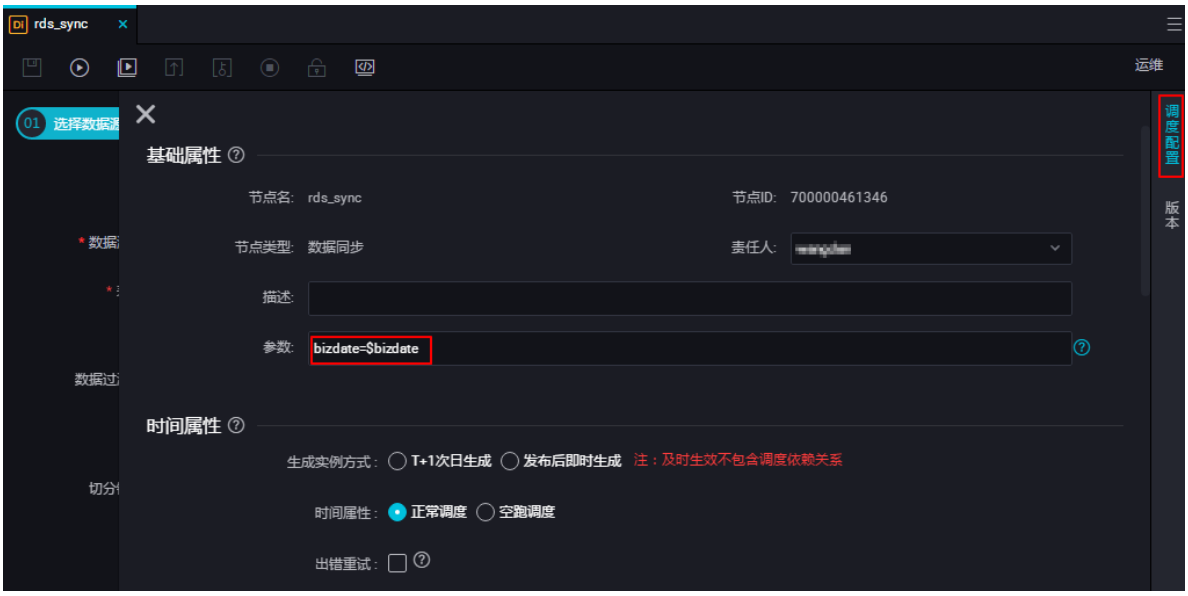
准备工作完成后，需要将RDS中的数据定时每天同步到MaxCompute中，自动创建按天日期的分区。

1. 选择数据来源和数据去向，如下图。

2. 参数配置，如下图。



一般配置到这个地方的时候，默认是系统自带的时间参数：`${bizdate}`，格式为yyyymmdd，对应也数据去向下的分区信息。就是说在调度执行这个任务的时候，这个分区会被自动替换为任务执行日期的前一天，一般用户会在当前跑前一天的业务数据，这个日期也叫业务日期。如果用户要使用当天任务运行的日期作为分区值，需要自定义这个参数。选择调度配置。如下图。



自定义参数设置，格式非常灵活，日期是当天日期，用户可以自由选择哪一天以及格式。可供参考的变量参数配置方式如下：

后N年：`$[add_months(yyyymmdd,12*N)]`

前N年：`$[add_months(yyyymmdd,-12*N)]`

后N月：`$[add_months(yyyymmdd,N)]`

前N月：`$[add_months(yyyymmdd,-N)]`

后N周：`$[yyyymmdd+7*N]`

前N周：`$[yyyymmdd-7*N]`

后N天：`$[yyyymmdd+N]`

前N天：`$[yyyymmdd-N]`

后N小时：`$[hh24miss+N/24]`

前N小时：`$[hh24miss-N/24]`

后N分钟：`$[hh24miss+N/24/60]`

前N分钟：`$[hh24miss-N/24/60]`

说明：

- 请以中括号 [] 编辑自定义变量参数的取值计算公式，例如 `key1=$[yyyy-mm-dd]`。

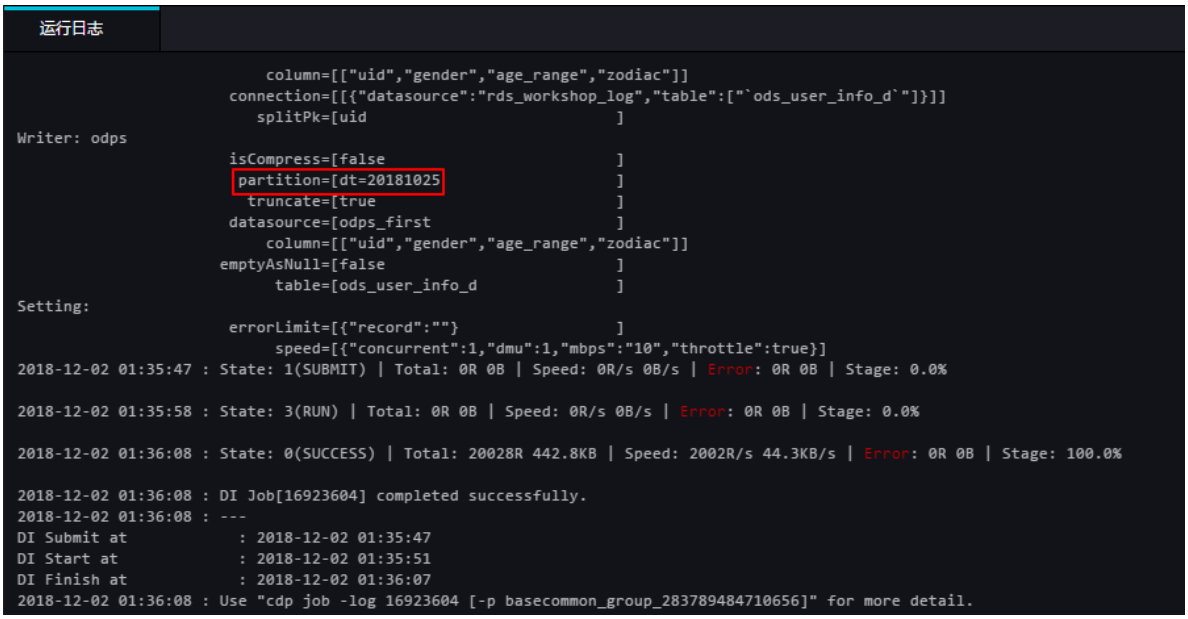
- 默认情况下，自定义变量参数的计算单位为天。例如 `$[hh24miss-N/24/60]` 表示 `(yyyymmddhh24miss-(N/24/60 * 1天))` 的计算结果，然后按 hh24miss 的格式取时分秒。
- 使用 add_months 的计算单位为月。例如 `$[add_months(yyyymmdd,12 N)-M/24/60]` 表示 `(yyyymmddhh24miss-(12 N 1月))-(M/24/60 1天)` 的结果，然后按 yyyymmdd 的格式取年月日。

详细的参数设置请参见参数配置。

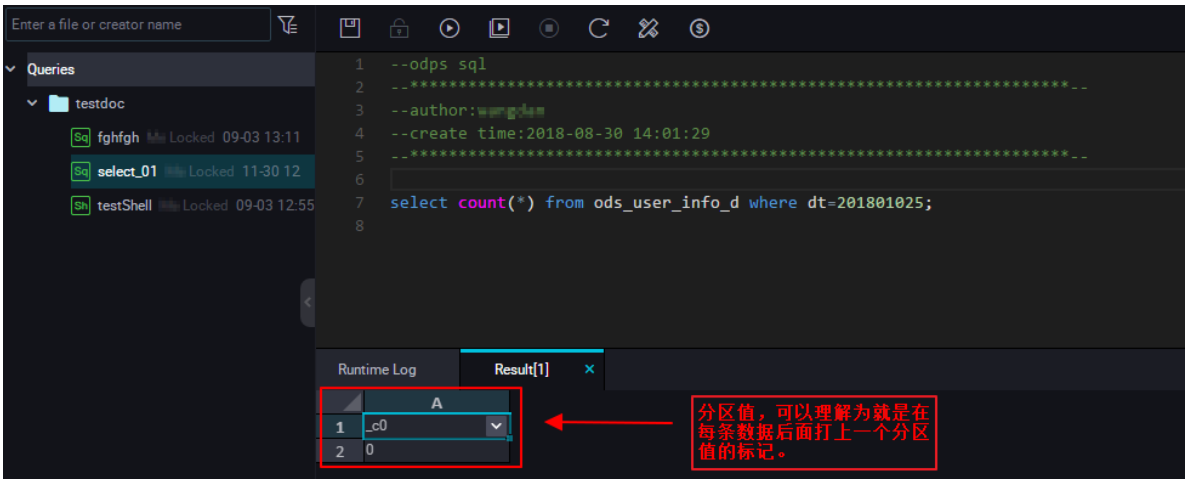
3. 测试运行。

保存所有配置，单击运行，如下图。

查看运行日志，如下图。



可以看到日志中，MaxCompute（日志中打印原名ODPS）的信息中partition分区，dt=20181025，自动替换成功。检查下实际的数据有没有转移到ODPS表中，如下图。



说明：

在maxcompute2.0中分区表查询需要添加分区筛选，不支持全量查询。SQL语句如下。

```
--查看是否成功写入MaxCompute
select count(*) from ods_user_info_d where dt=业务日期;
```

SELECT命令详情请参见*Select*操作。

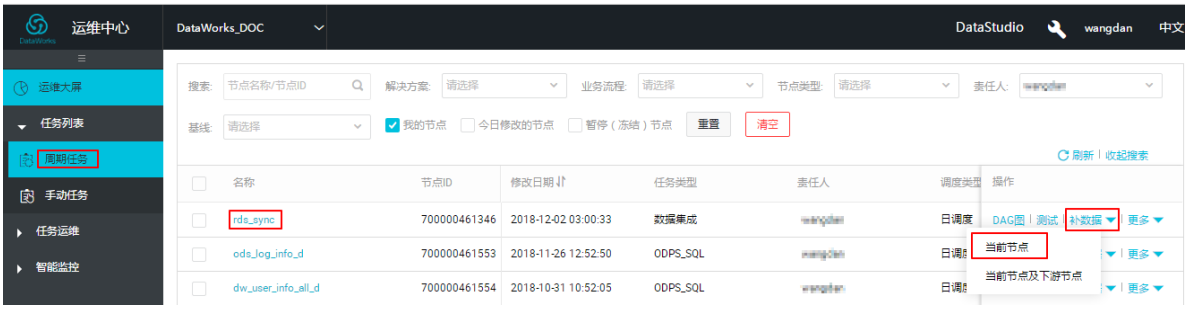
此时看到数据已经迁移到ODPS表中，并且成功创建了一个分区值。那么这个任务在执行定时调度的时候，会每天将RDS中的数据同步到MaxCompute中按照日期自动创建的分区中。

补数据实验

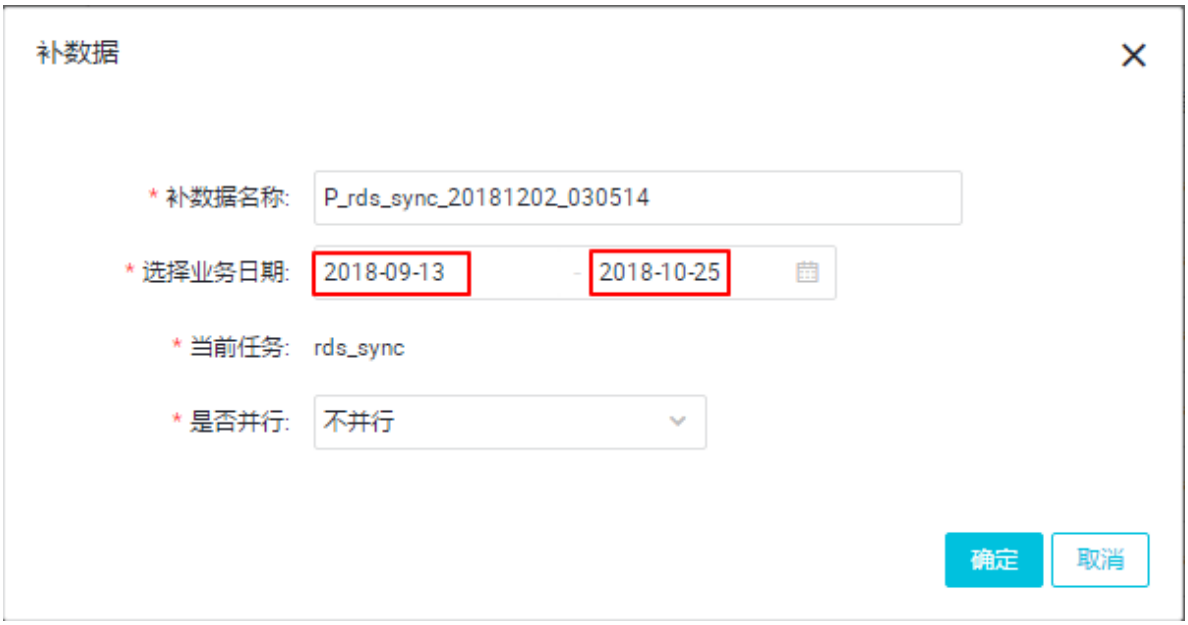
如果用户的数据有很多运行日期之前的历史数据，想要实现自动同步和自动分区，此时您可以进入DataWorks的运维中心，选择当前的同步数据节点，选择**补数据**功能来实现。

1. 首先，我们需要在RDS端把历史数据按照日期筛选出来，比如历史数据2018-09-13这天的数据，我们要自动同步到MaxCompute的20180825的分区中。在RDS阶段可以设置where过滤条件，如图。

2. 补数据操作。然后**保存** > **提交**。提交后到**运维中心** > **任务列表** > **周期任务**中的rds_sync节点，单击**补数据** > **当前节点**。如下图。

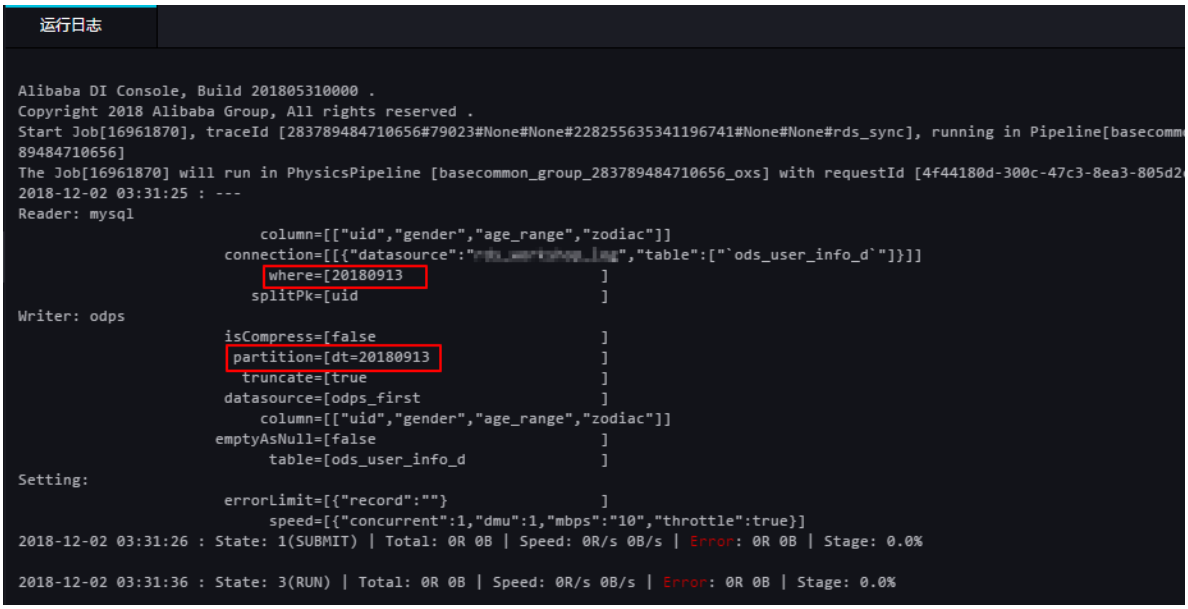


3. 跳转至补数据节点页面。选择日期区间，如下图。

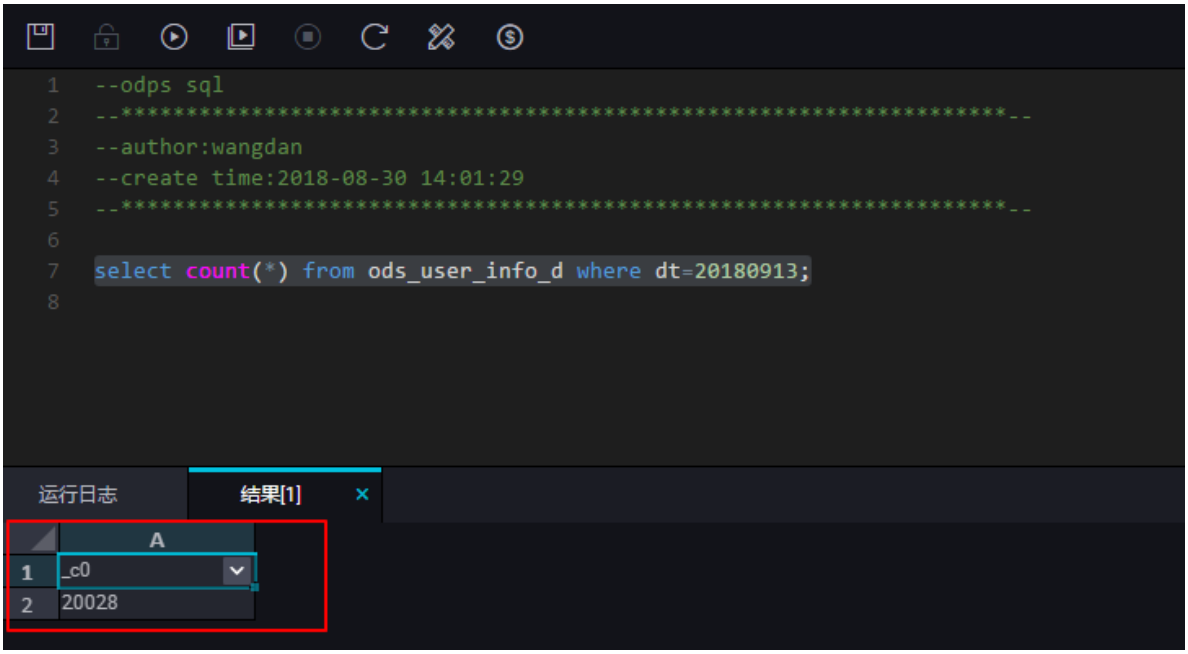


4. 单击确定。此时会同时生成多个同步的任务实例按顺序执行。如下图。

5. 查看运行的日志，可以看到运行过程中对RDS数据的抽取。此时MaxCompute已自动创建分区，如图。



查看运行结果。数据写入的情况，以及是否自动创建了分区，数据是否已同步到分区表中，如下图。



查询对应分区信息，如下图。

说明：

在maxcompute2.0中分区表查询需要添加分区筛选，SQL语句如下。其中分区列需要更新为业务日期，如任务运行的日期为20180717，那么业务日期为20180716。

```
--查看是否成功写入MaxCompute
select count(*) from ods_user_info_d where dt=业务日期;
```

Hash实现非日期字段分区

如果用户数据量比较巨大，或者第一次全量的数据并不是按照日期字段进行分区，而是按照省份等非日期字段分区，那么此时数据集成操作就不能做到自动分区了。也就是说，可以按照RDS中某个字段进行hash，将相同的字段值自动存放到这个字段对应值的MaxCompute分区中。

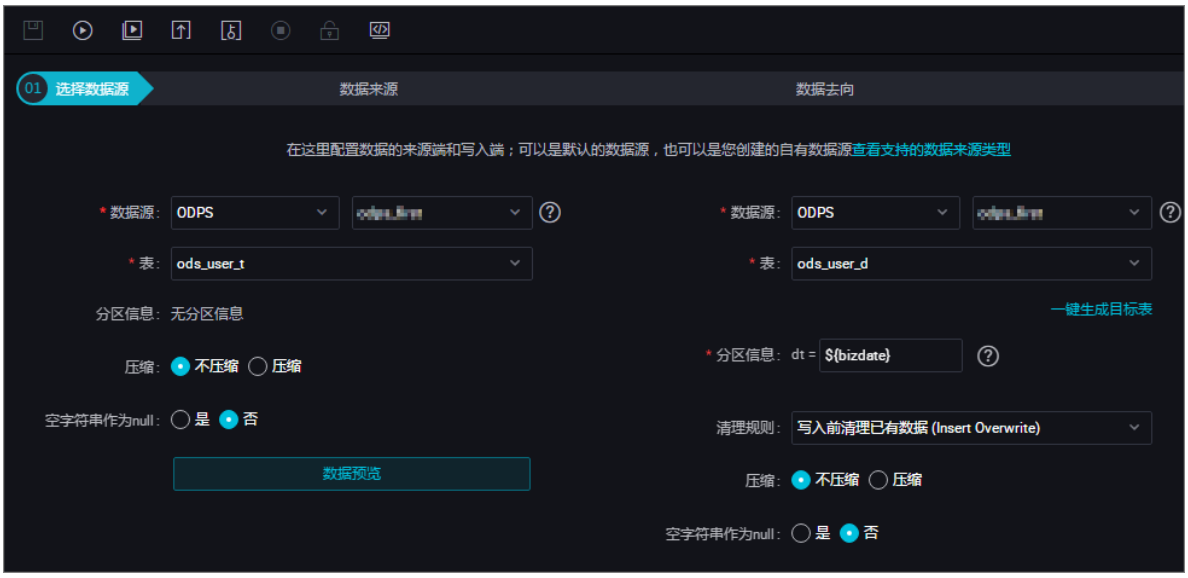
流程如下：

1. 首先我们需要把数据全量同步到MaxCompute的一个临时表。创建一个SQL脚本节点。单击运行 > 保存 > 提交。

   sql命令如下。

   ```
   drop table if exists ods_user_t;
   CREATE TABLE ods_user_t (
           dt STRING,
    uid STRING,
           gender STRING,
           age_range STRING,
           zodiac STRING);
   insert overwrite table ods_user_t select dt,uid,gender,age_range,
   zodiac from ods_user_info_d;--将ODPS表中的数据存入临时表。
   ```

2. 创建同步任务的节点mysql_to_odps，就是简单的同步任务，将RDS数据全量同步到MaxCompute，不需要设置分区。如下图。

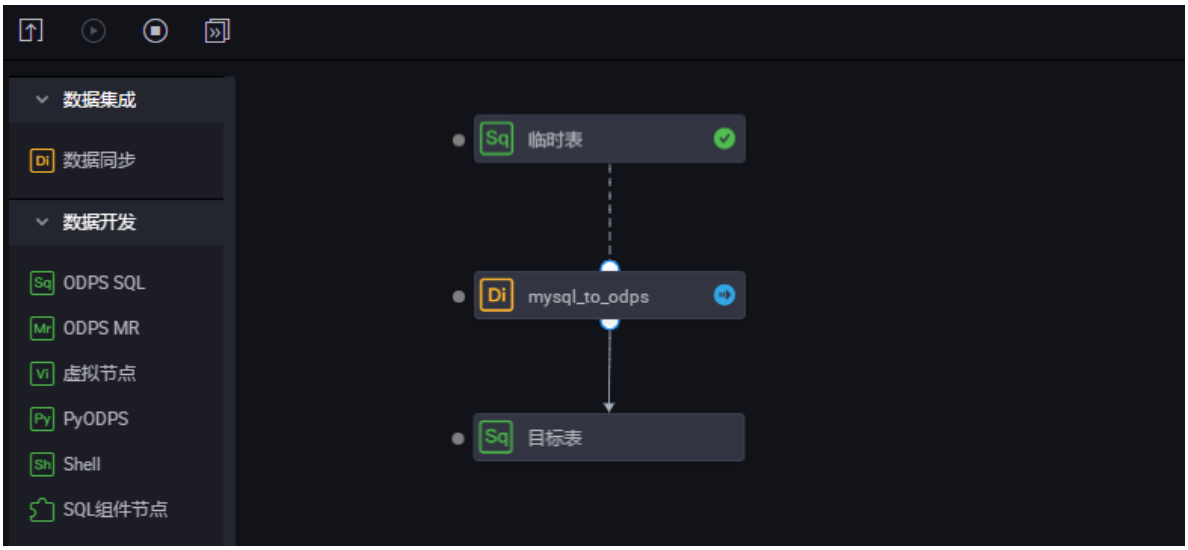


3. 使用sql语句进行动态分区到目标表。命令如下。

```
drop table if exists ods_user_d;
--创建一个ODPS分区表（最终目的表）
 CREATE TABLE ods_user_d (
 uid STRING,
        gender STRING,
        age_range STRING,
        zodiac STRING
)
PARTITIONED BY (
 dt STRING
);
--执行动态分区sql，按照临时表的字段dt自动分区，dt字段中相同的数据值，会按照这个数据值自动创建一个分区值
--例如dt中有些数据是20180913，会自动在ODPS分区表中创建一个分区，dt=20181025
--动态分区sql如下
--可以注意到sql中select的字段多写了一个dt，就是指定按照这个字段自动创建分区
insert overwrite table ods_user_d partition(dt)select dt,uid,gender,
age_range,zodiac from ods_user_t;
--导入完成后，可以把临时表删除，节约存储成本
drop table if exists ods_user_t;
```

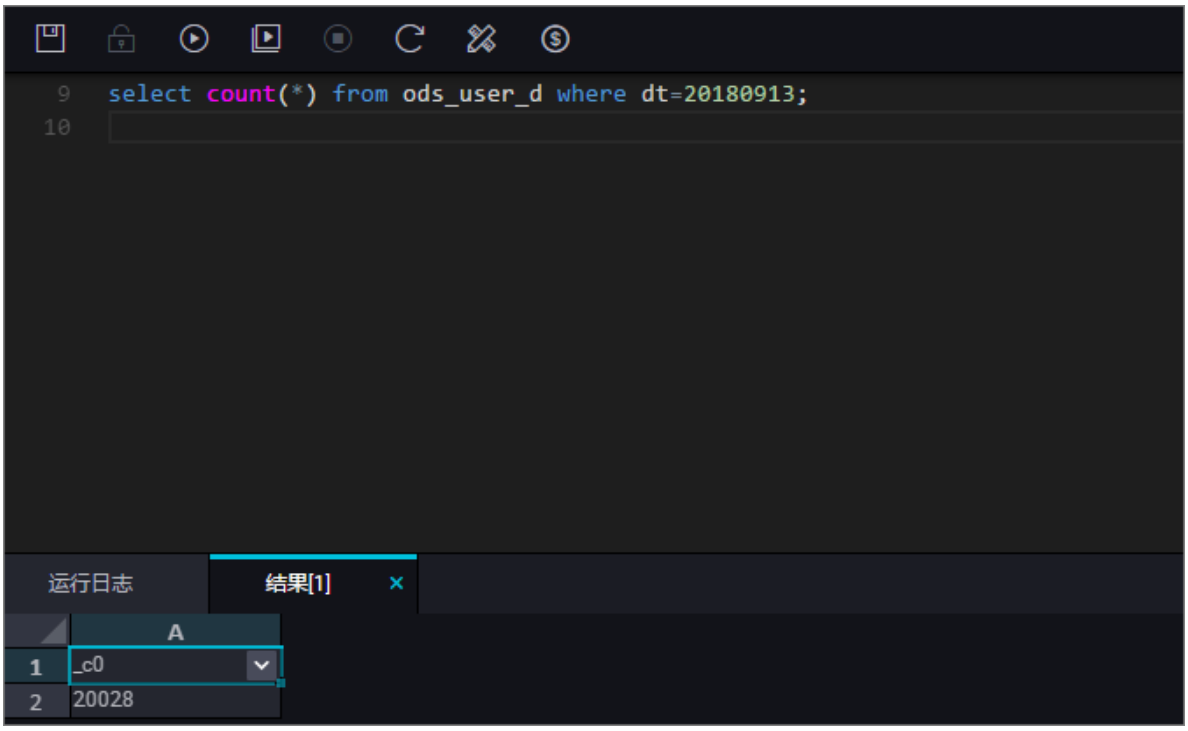
在MaxCompute中我们通过SQL语句来完成同步。详细的SQL语句介绍请参见阿里云大数据利器*MaxCompute*学习之--分区表的使用。

4. 最后将三个节点配置成一个工作流，按顺序执行。如下图。

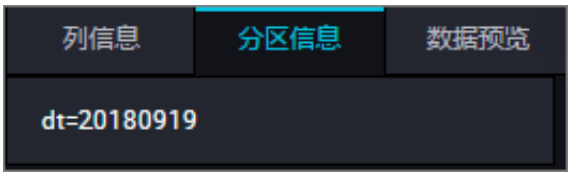


5. 查看执行过程。我们可以重点观察最后一个节点的动态分区过程，如下图。

查看数据。动态的自动化分区完成。相同的日期数据迁移到了同一个分区中。如下图。



对应查询分区信息，如下图。



如果是以省份字段命名分区，执行步骤参请考上述内容。

DataWorks数据同步功能可以完成大部分自动化作业，尤其是数据的同步迁移，调度等，了解更多的调度配置请参见调度配置时间属性。

11 JSON数据从OSS迁移到MaxCompute最佳实践

本文为您介绍如何利用DataWorks数据集成将JSON数据从OSS迁移到MaxCompute，并使用MaxCompute内置字符串函数GET_JSON_OBJECT提取JSON信息。

数据上传OSS

将您的JSON文件重命名后缀为TXT文件，并上传到OSS。本文中使用的JSON文件示例如下。

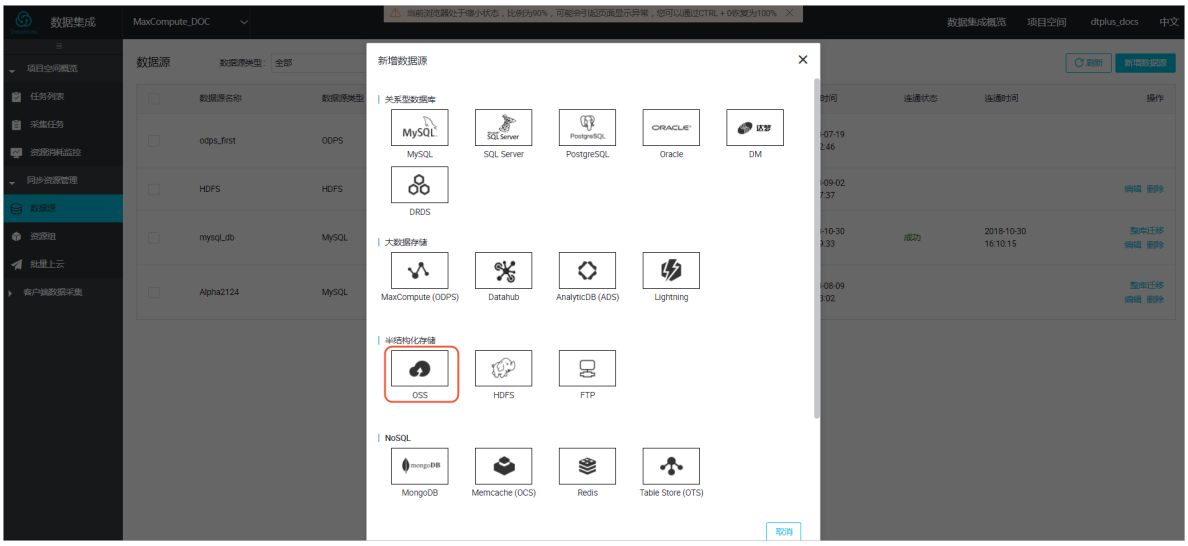
```
{
    "store": {
        "book": [
            {
               "category": "reference",
               "author": "Nigel Rees",
               "title": "Sayings of the Century",
               "price": 8.95
            },
            {
               "category": "fiction",
               "author": "Evelyn Waugh",
               "title": "Sword of Honour",
               "price": 12.99
            },
            {
                "category": "fiction",
                "author": "J. R. R. Tolkien",
                "title": "The Lord of the Rings",
                "isbn": "0-395-19395-8",
                "price": 22.99
            }
        ],
        "bicycle": {
            "color": "red",
            "price": 19.95
        }
    },
    "expensive": 10
}
```

将applog.txt文件上传到OSS，本文中OSS Bucket位于华东2区。

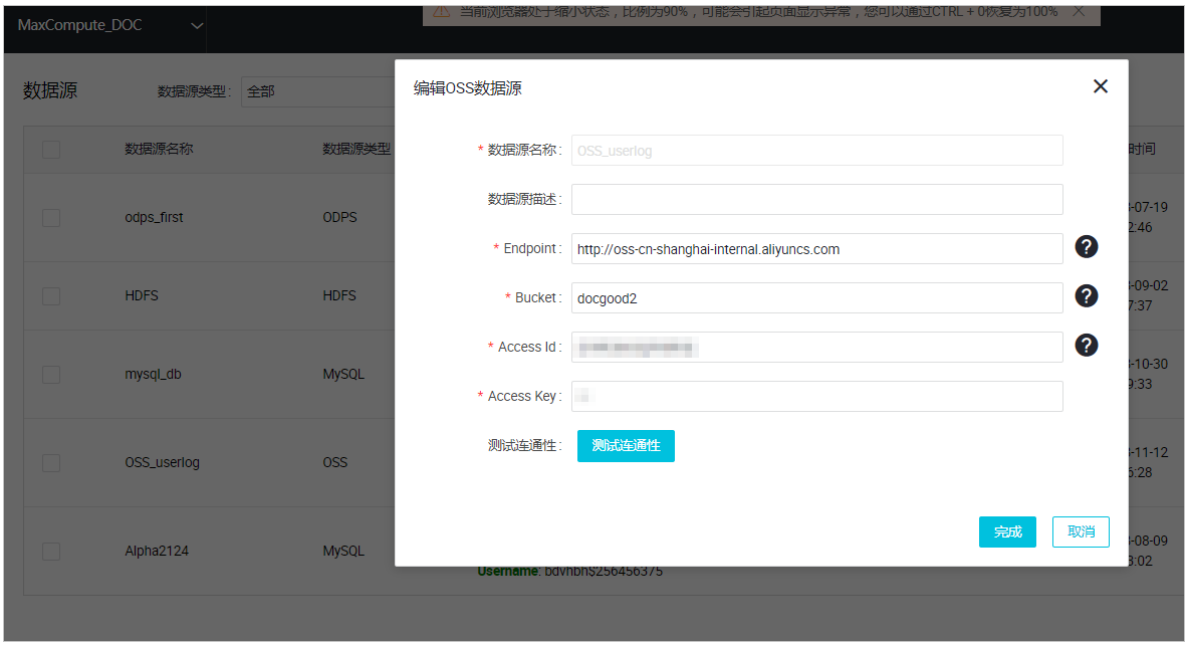
使用DataWorks导入数据到MaxCompute

1. 新增OSS数据源

   进入DataWorks数据集成控制台，新增OSS类型数据源。

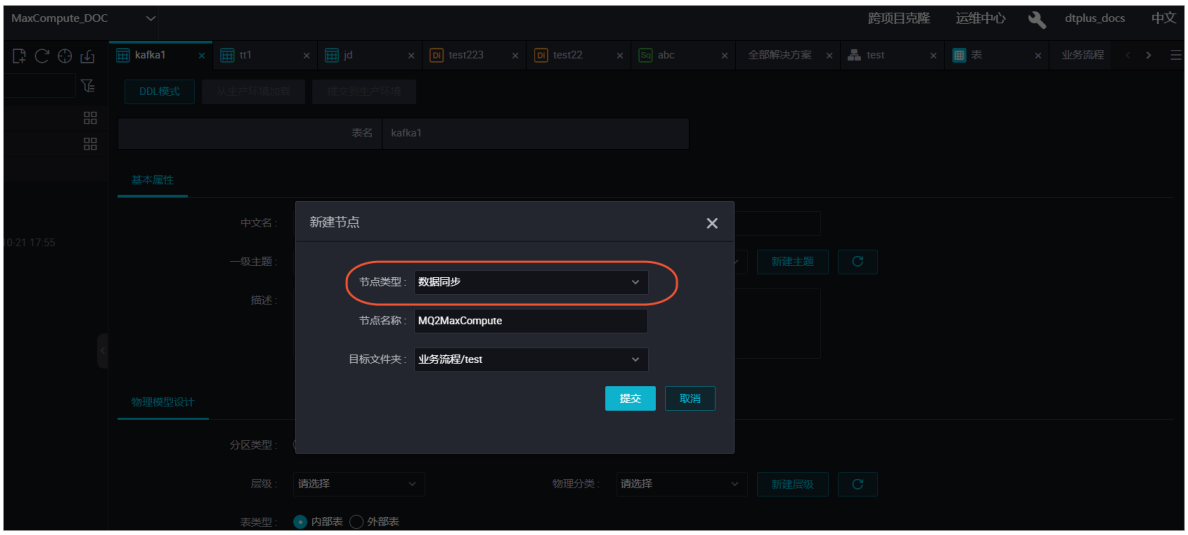


   具体参数如下所示，测试数据源连通性通过即可点击完成。Endpoint地址请参见OSS各区域的外网、内网地址，本例中为`http://oss-cn-shanghai.aliyuncs.com`或`http://oss-cn-shanghai-internal.aliyuncs.com`（由于本文中OSS和DataWorks项目处于同一个region中，本文选用后者，通过内网连接）。



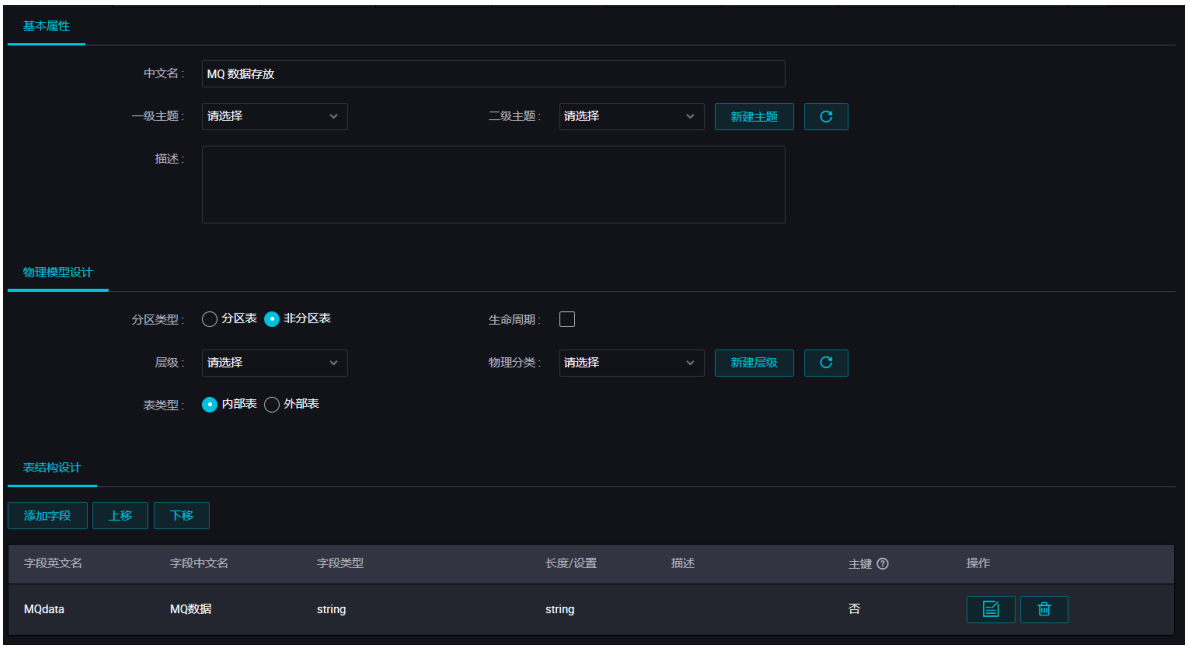
2. 新建数据同步任务

   在DataWorks上新建数据同步类型节点。

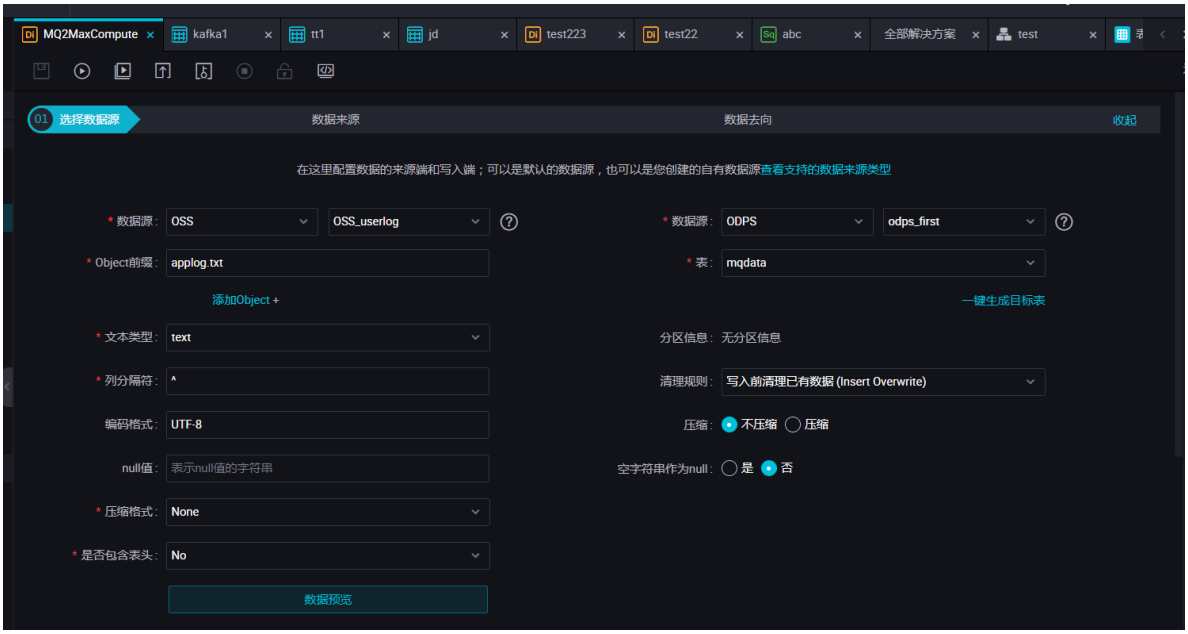


新建的同时，在DataWorks新建一个建表任务，用于存放JSON数据，本例中新建表名为mqdata。

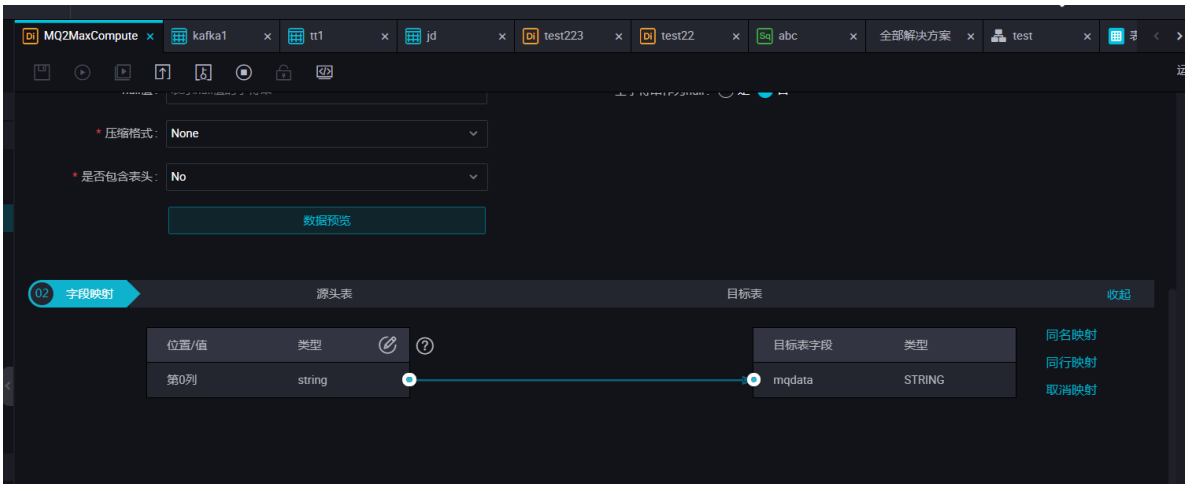
表参数可以通过图形化界面完成。本例中mqdata表仅有一列，类型为string，列名为MQ data。



完成上述新建后，您可以在图形化界面配置数据同步任务参数，如下图所示。选择目标数据源名称为odps_first，选择目标表为刚建立的mqdata。数据来源类型为OSS，Object前缀可填写文件路径及名称。列分隔符使用TXT文件中不存在的字符即可，本文中使用^（对于OSS中的TXT格式数据源，Dataworks支持多字符分隔符，所以您可以使用例如%&%#^$$^%这样很难出现的字符作为列分隔符，保证分割为一列）。



映射方式选择默认的同行映射即可。



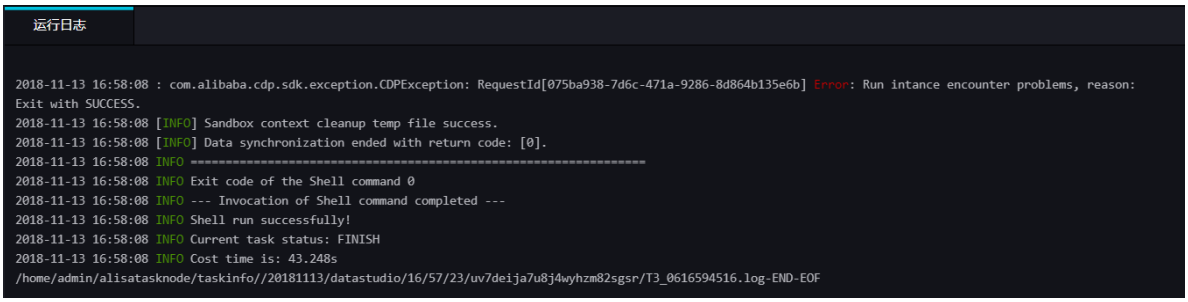
点击左上方的切换脚本按钮，切换为脚本模式。修改fileFormat参数为：`"fileFormat":"binary"`。该步骤可以保证OSS中的JSON文件同步到MaxCompute之后存在同一行数据中，即为一个字段。其他参数保持不变，脚本模式代码示例如下。

```
{
    "type": "job",
    "steps": [
        {
```

```
            "stepType": "oss",
            "parameter": {
                "fieldDelimiterOrigin": "^",
                "nullFormat": "",
                "compress": "",
                "datasource": "OSS_userlog",
                "column": [
                    {
                        "name": 0,
                        "type": "string",
                        "index": 0
                    }
                ],
                "skipHeader": "false",
                "encoding": "UTF-8",
                "fieldDelimiter": "^",
                "fileFormat": "binary",
                "object": [
                    "applog.txt"
                ]
            },
            "name": "Reader",
            "category": "reader"
        },
        {
            "stepType": "odps",
            "parameter": {
                "partition": "",
                "isCompress": false,
                "truncate": true,
                "datasource": "odps_first",
                "column": [
                    "mqdata"
                ],
                "emptyAsNull": false,
                "table": "mqdata"
            },
            "name": "Writer",
            "category": "writer"
        }
    ],
    "version": "2.0",
    "order": {
        "hops": [
            {
                "from": "Reader",
                "to": "Writer"
            }
        ]
    },
    "setting": {
        "errorLimit": {
            "record": ""
        },
        "speed": {
            "concurrent": 2,
            "throttle": false,
            "dmu": 1
        }
    }
```

```
}
```

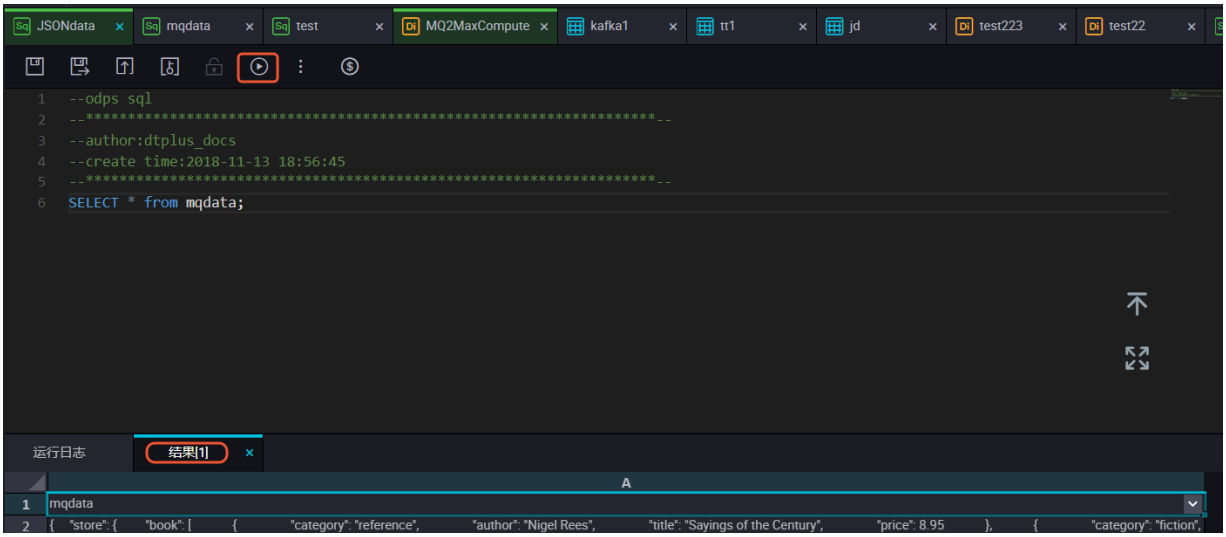
完成上述配置后，点击运行接即可。运行成功日志示例如下所示。



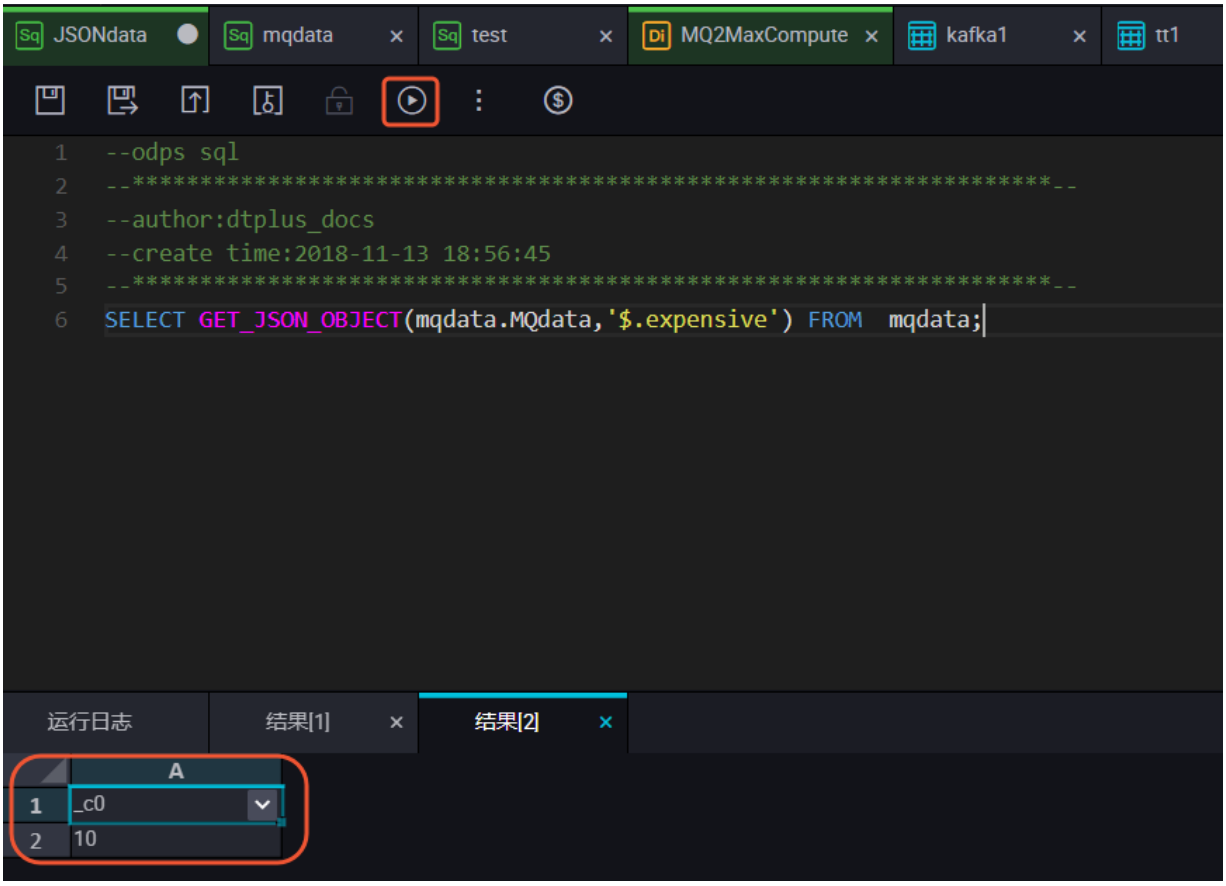
获取JSON字段信息

在您的业务流程中新建一个ODPS SQL节点。

您可以首先输入`SELECT*from mqdata;`语句，查看当前mqdata表中数据。当然这一步及后续步骤，您也可以直接在*MaxCompute*客户端中输入命令运行。



确认导入表中的数据结果无误后，您可以使用MaxCompute内建字符串函数*GET_JSON_OBJECT*获取您想要的JSON数据。本例中使用`SELECT GET_JSON_OBJECT(mqdata.MQdata,'$.expensive') FROM mqdata;`获取JSON文件中的**expensive**值。如下图所示，可以看到已成功获取数据。



12 JSON数据从MongoDB迁移到MaxCompute最佳实践

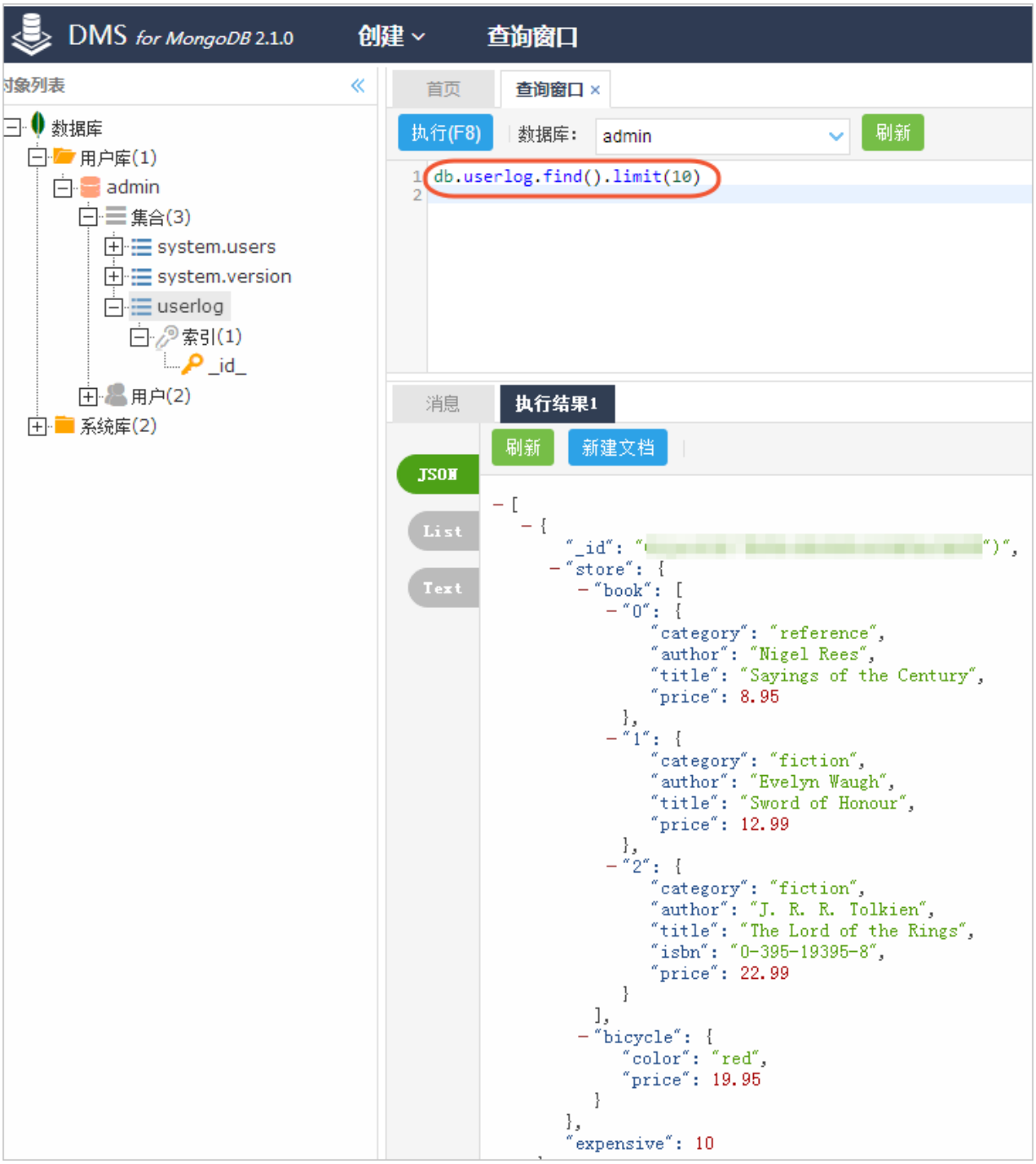
本文为您介绍如何利用DataWorks数据集成直接从MongoDB提取JSON字段到MaxCompute。

数据及账号准备

首先您需要将数据上传至您的MongoDB数据库。本例中使用阿里云的云数据库 *MongoDB* 版，网络类型为VPC（需申请公网地址，否则无法与DataWorks默认资源组互通），测试数据如下。

```
{
    "store": {
        "book": [
            {
               "category": "reference",
               "author": "Nigel Rees",
               "title": "Sayings of the Century",
               "price": 8.95
            },
            {
               "category": "fiction",
               "author": "Evelyn Waugh",
               "title": "Sword of Honour",
               "price": 12.99
            },
            {
                "category": "fiction",
                "author": "J. R. R. Tolkien",
                "title": "The Lord of the Rings",
                "isbn": "0-395-19395-8",
                "price": 22.99
            }
        ],
        "bicycle": {
            "color": "red",
            "price": 19.95
        }
    },
    "expensive": 10
}
```

登录MongoDB的DMS控制台，本例中使用的数据库为**admin**，集合为**userlog**，您可以在查询窗口使用**db.userlog.find().limit(10)**命令查看已上传好的数据，如下图所示。

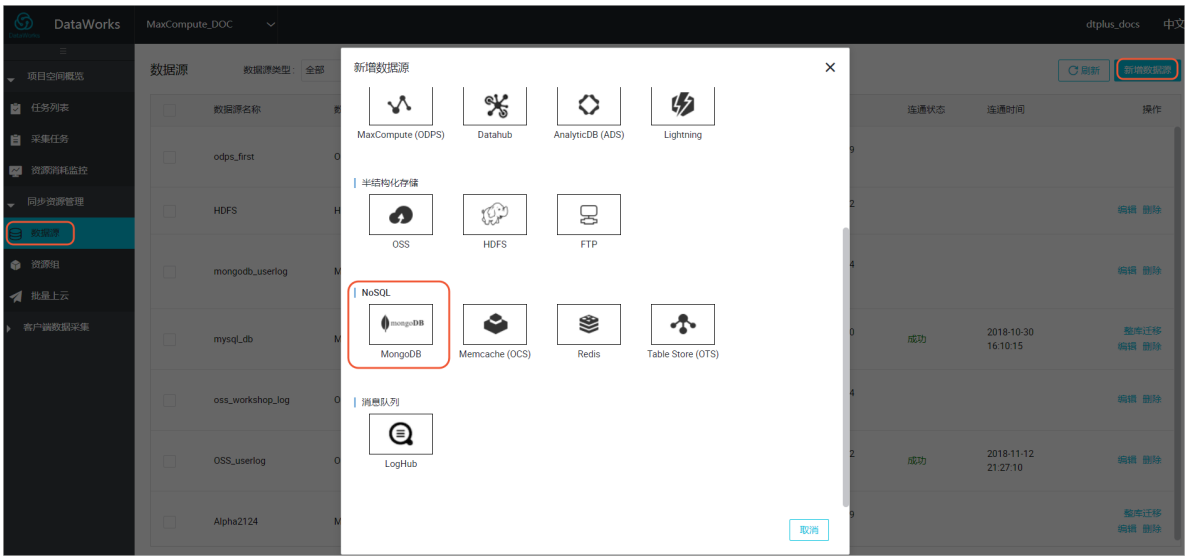


此外，需提前在数据库内新建用户，用于DataWorks添加数据源。本例中使用命令`db.createUser({user:"bookuser",pwd:"123456",roles:["root"]})`，新建用户名为**bookuser**，密码为**123456**，权限为**root**。

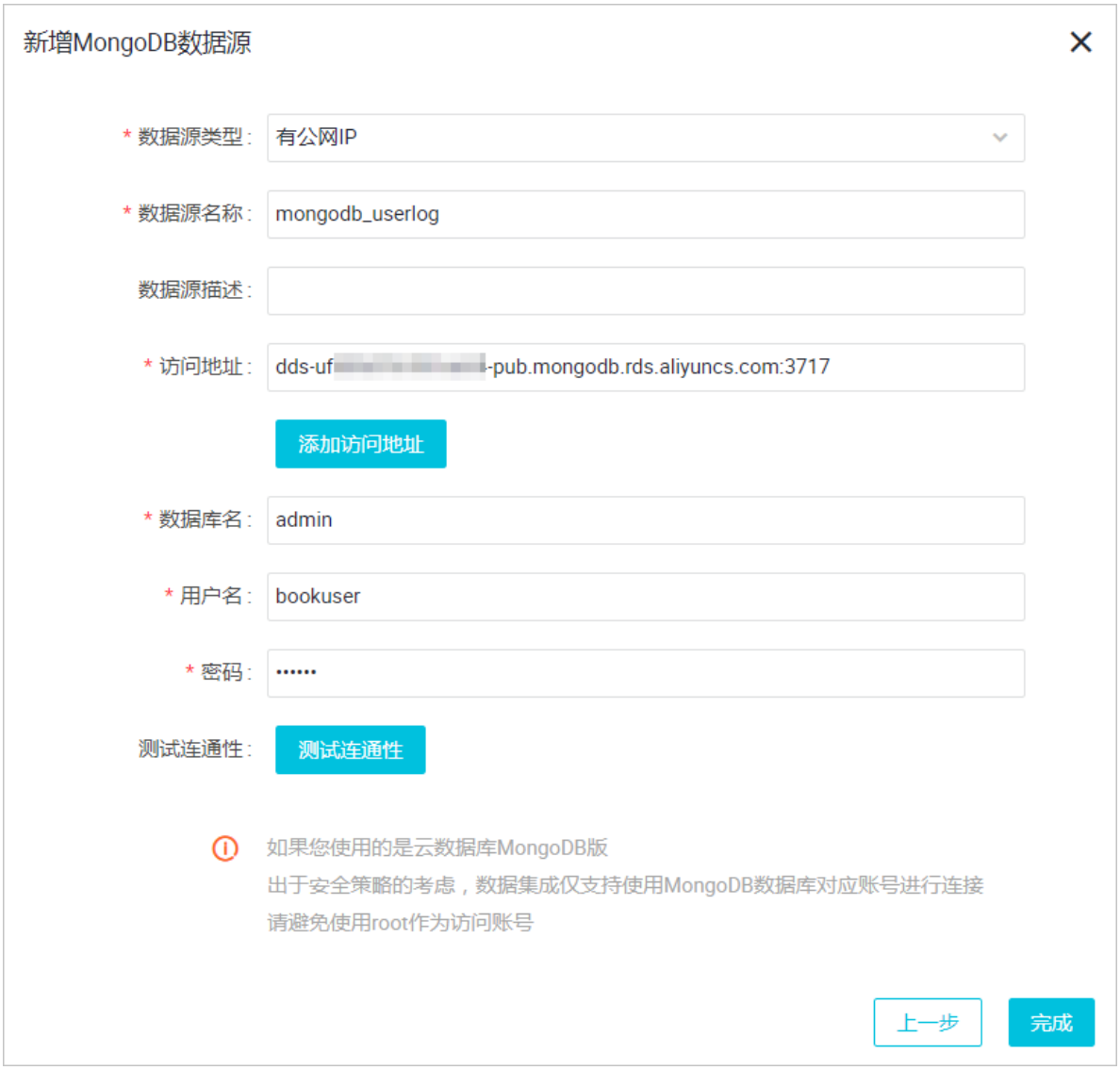
使用DataWorks提取数据到MaxCompute

1. 新增MongoDB数据源

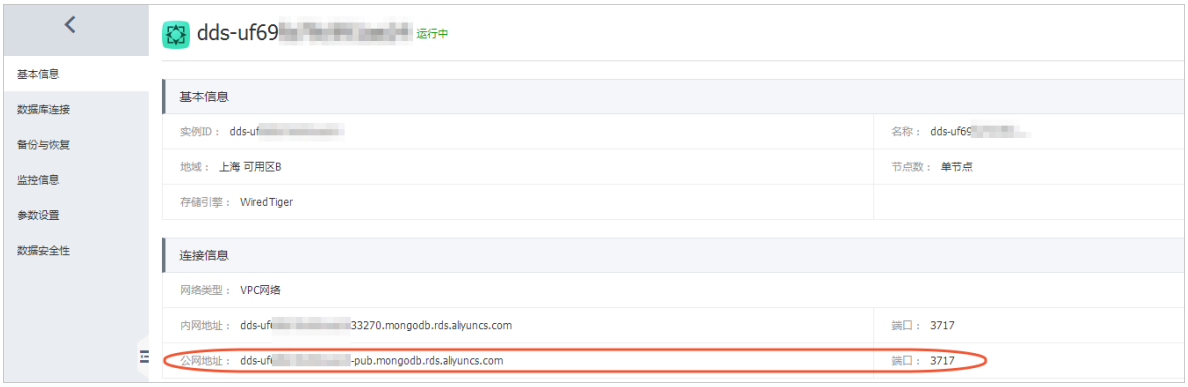
   进入DataWorks数据集成控制台，新增*MongoDB*类型数据源。



具体参数如下所示，测试数据源连通性通过即可点击**完成**。由于本文中MongoDB处于VPC环境下，因此**数据源类型**需选择**有公网IP**。



访问地址及端口号可通过在*MongoDB*管理控制台点击实例名称获取，如下图所示。

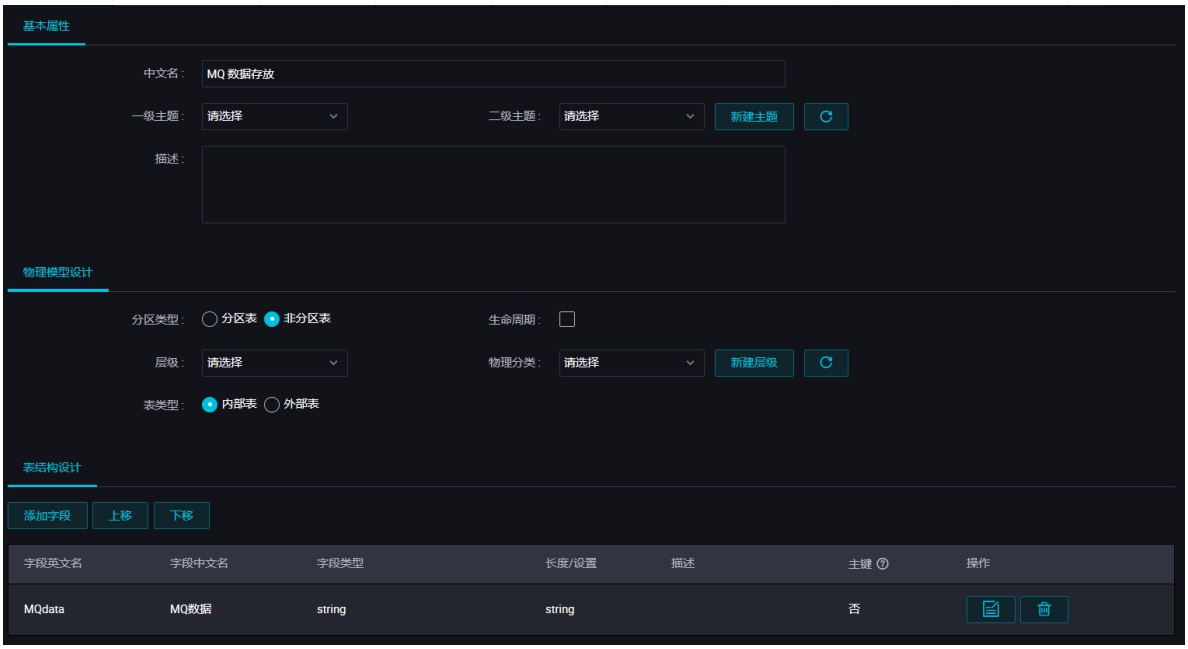


2. 新建数据同步任务

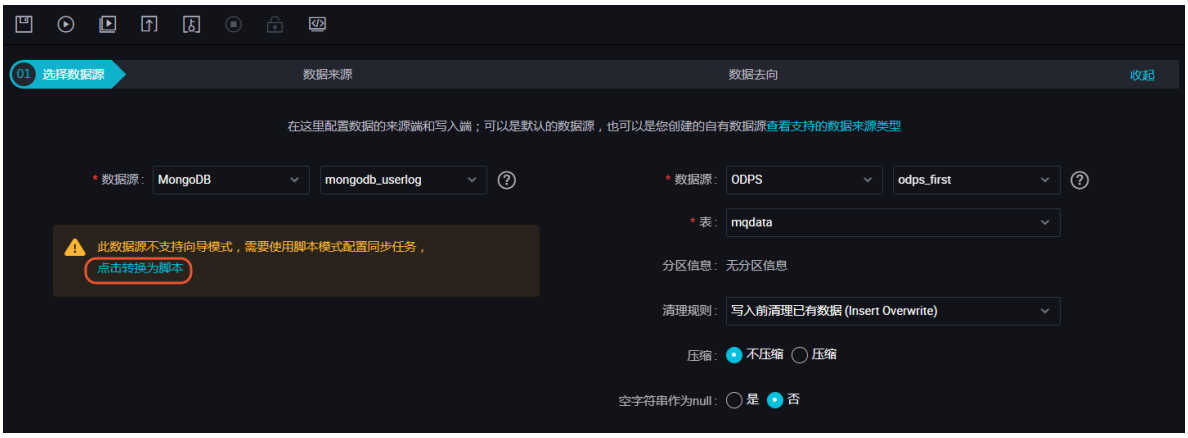
在DataWorks上新建数据同步类型节点。

新建的同时，在DataWorks新建一个建表任务，用于存放JSON数据，本例中新建表名为mqdata。

表参数可以通过图形化界面完成。本例中mqdata表仅有一列，类型为string，列名为MQ data。



完成上述新建后，您可以在图形化界面进行数据同步任务参数的初步配置，如下图所示。选择目标数据源名称为odps_first，选择目标表为刚建立的mqdata。数据来源类型为MongoDB，选择我们刚创建的数据源mongodb_userlog。完成上述配置后，点击**转换为脚本**，跳转到脚本模式。

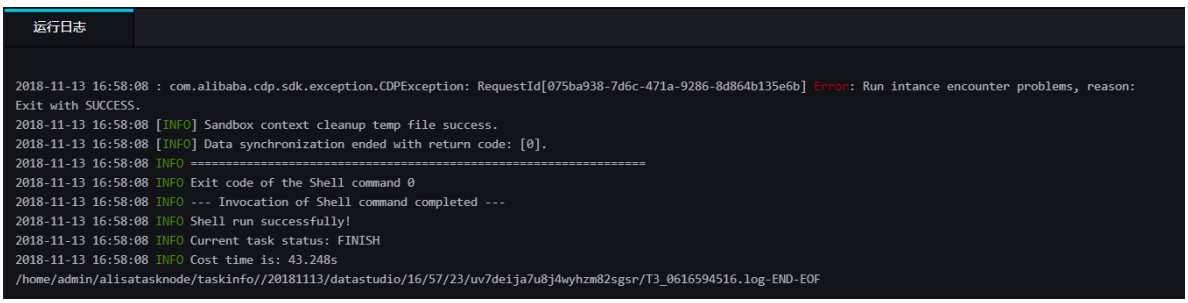


脚本模式代码示例如下。

```
{
    "type": "job",
    "steps": [
        {
            "stepType": "mongodb",
            "parameter": {
                "datasource": "mongodb_userlog",
 //数据源名称
                "column": [
                    {
                        "name": "store.bicycle.color", //JSON字段路
径，本例中提取color值
```

```
                            "type": "document.document.string" //本栏目的字段数需和name一致。假如您选取的JSON字段为一级字段，如本例中的expensive，则直接填写string即可。
                        }
                    ],
                    "collectionName //集合名称": "userlog"
                },
                "name": "Reader",
                "category": "reader"
            },
            {
                "stepType": "odps",
                "parameter": {
                    "partition": "",
                    "isCompress": false,
                    "truncate": true,
                    "datasource": "odps_first",
                    "column": [
                                "mqdata"  //MaxCompute表列名
                    ],
                    "emptyAsNull": false,
                    "table": "mqdata"
                },
                "name": "Writer",
                "category": "writer"
            }
        ],
        "version": "2.0",
        "order": {
            "hops": [
                {
                    "from": "Reader",
                    "to": "Writer"
                }
            ]
        },
        "setting": {
            "errorLimit": {
                "record": ""
            },
            "speed": {
                "concurrent": 2,
                "throttle": false,
                "dmu": 1
            }
        }
}
```

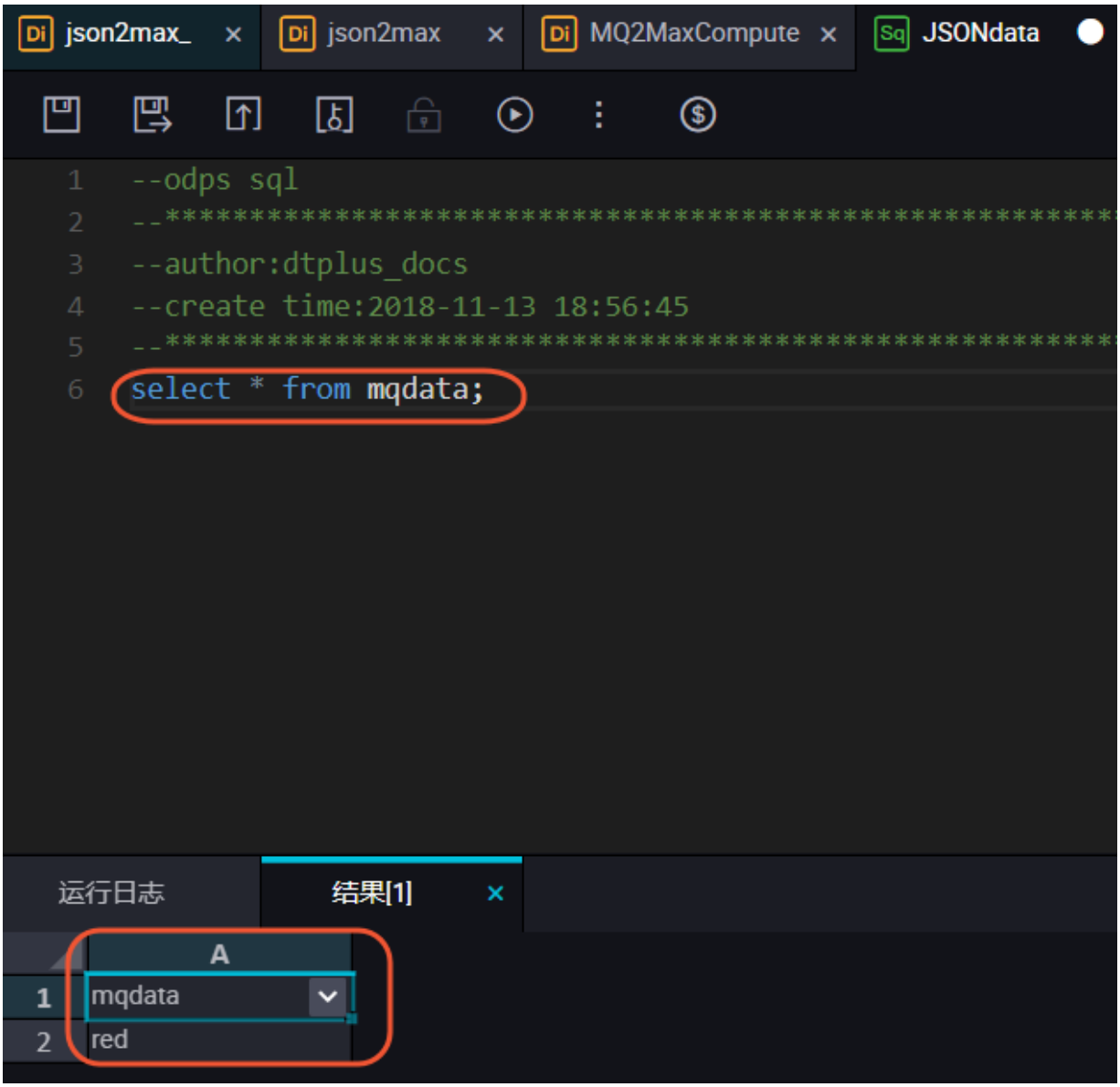
完成上述配置后，点击运行接即可。运行成功日志示例如下所示。



结果验证

在您的业务流程中新建一个ODPS SQL节点。

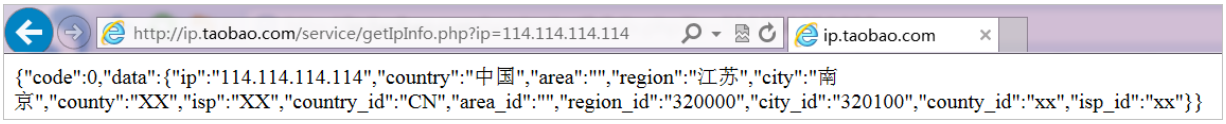
您可以输入`SELECT * from mqdata;`语句，查看当前mqdata表中数据。当然这一步您也可以直接在*MaxCompute*客户端中输入命令运行。


json2max_
json2max
MQ2MaxCompute
JSONdata
--odps sql
--author:dtplus_docs
--create time:2018-11-13 18:56:45
select * from mqdata;
运行日志
结果[1]
A
1
mqdata
2
red

13 使用MaxCompute分析IP来源最佳实践

本文介绍如何在MaxCompute上分析IP来源的方法，包括下载、上传淘宝IP地址库数据及编写UDF函数、编写SQL四个步骤。

淘宝IP地址库的查询接口为IP地址字串。使用示例如下。



由于在MaxCompute中禁止使用HTTP请求，如何实现在MaxCompute中进行IP的查询？目前有三种方式：

1. 用SQL将数据查询到本地，再发起HTTP请求查询。

   > 说明：
   > 效率低下，且淘宝IP库查询频率需小于10QPS，否则拒绝请求。

2. 下载IP地址库到本地，进行查询。

   > 说明：
   > 同样效率低，且不利于数仓等分析使用。

3. 将IP地址库定期维护上传至MaxCompute，进行连接查询。

   > 说明：
   > 比较高效，但是IP地址库需自己定期维护。

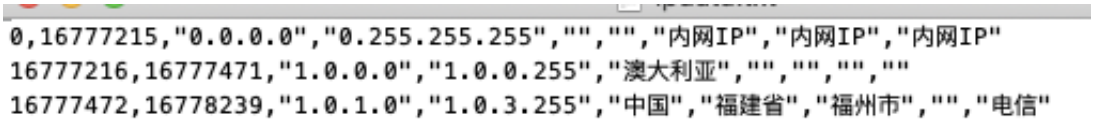
本文重点为您介绍第三种方式。

下载IP地址库

1. 在IP数据下载链接获取您需要的格式数据。本文以基本数据格式为例。

数据格式描述	文件名前缀	下载链接（!!!友情提示：由于接口数据量较大，通过浏览器打开可能会速度过慢，建议通过wget下载至本地后用其他编辑器查看）
1，基本格式数据 格式描述	ipdata.txt	gbk数据： http://ip.taobao.org:9999/ipdata/ipdata.txt.gbk gbk哈希： http://ip.taobao.org:9999/ipdata/ipdata.txt.gbk.md5 gbk打包下载： http://ip.taobao.org:9999/ipdata/ipdata.txt.gbk.tar.gz utf8数据： http://ip.taobao.org:9999/ipdata/ipdata.txt.utf8 uf8哈希： http://ip.taobao.org:9999/ipdata/ipdata.txt.utf8.md5 utf8打包下载： http://ip.taobao.org:9999/ipdata/ipdata.txt.utf8.tar.gz

2. 下载**utf8**数据到本地，检查数据格式。



前四个数据是IP地址的起始地址与结束地址：前两个是十进制整数形式，后两个是点分形式。这里我们使用整数形式，以便计算IP是否属于这个网段。

上传IP地址库数据

1. 创建表DDL，您可以使用*MaxCompute*客户端进行操作，也可以使用DataWorks进行图形化建表。

```
DROP TABLE IF EXISTS ipresource ;

CREATE TABLE IF NOT EXISTS ipresource
(
    start_ip BIGINT
    ,end_ip BIGINT
    ,start_ip_arg string
    ,end_ip_arg string
    ,country STRING
    ,area STRING
    ,city STRING
    ,county STRING
    ,isp STRING
);
```

2. 使用*Tunnel*命令操作上传您的文件，本例中ipdata.txt.utf8文件存放在D盘。

```
odps@ workshop_demo>tunnel upload D:/ipdata.txt.utf8 ipresource;
```

可以通过SQL语句`select count(*) from ipresource;`查看到表中上传的数据条数（不断更新增长中）。

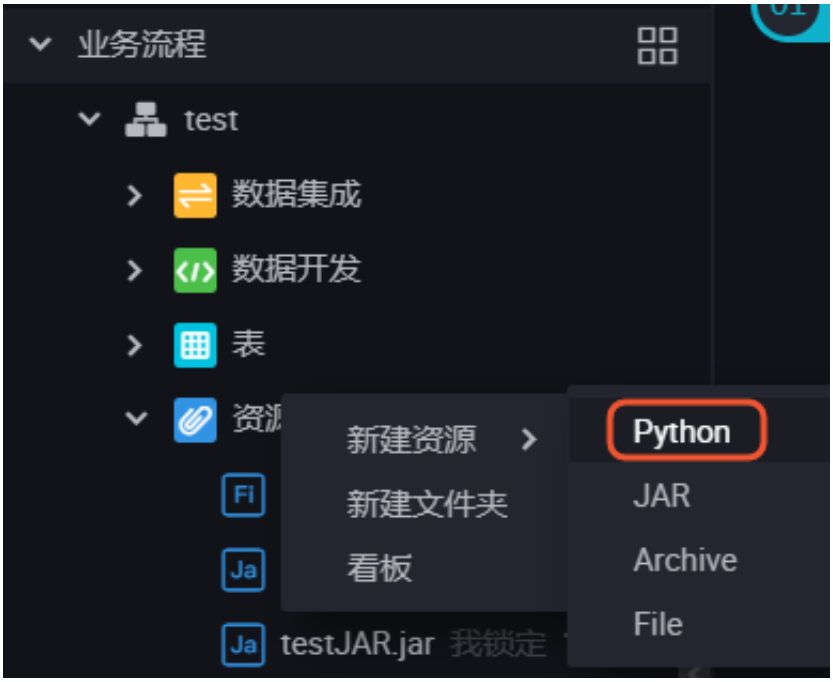
3. 使用SQL语句`select count(*) from ipresource limit 0,10;`查看ipresource表前10条的样本数据。

```
Job Queueing...
+------------+------------+--------------+------------+---------+------+------+--------+-----+
| start_ip   | end_ip     | start_ip_arg | end_ip_arg | country | area | city | county | isp |
+------------+------------+--------------+------------+---------+------+------+--------+-----+
| 3395369026 | 3395369026 | "202.97.56.66" | "202.97.56.66" | "中国" | "湖南省" | "长沙市" | "" | "电信" |
| 3395369027 | 3395369028 | "202.97.56.67" | "202.97.56.68" | "中国" | "黑龙江省" | "" | "" | "电信" |
| 3395369029 | 3395369029 | "202.97.56.69" | "202.97.56.69" | "中国" | "安徽省" | "合肥市" | "" | "电信" |
| 3395369030 | 3395369030 | "202.97.56.70" | "202.97.56.70" | "中国" | "湖南省" | "长沙市" | "" | "电信" |
| 3395369031 | 3395369033 | "202.97.56.71" | "202.97.56.73" | "中国" | "黑龙江省" | "" | "" | "电信" |
| 3395369034 | 3395369034 | "202.97.56.74" | "202.97.56.74" | "中国" | "湖南省" | "长沙市" | "" | "电信" |
| 3395369035 | 3395369036 | "202.97.56.75" | "202.97.56.76" | "中国" | "黑龙江省" | "" | "" | "电信" |
| 3395369037 | 3395369037 | "202.97.56.77" | "202.97.56.77" | "中国" | "江苏省" | "南京市" | "" | "电信" |
| 3395369038 | 3395369038 | "202.97.56.78" | "202.97.56.78" | "中国" | "湖南省" | "长沙市" | "" | "电信" |
| 3395369039 | 3395369040 | "202.97.56.79" | "202.97.56.80" | "中国" | "黑龙江省" | "" | "" | "电信" |
+------------+------------+--------------+------------+---------+------+------+--------+-----+
```

编写UDF函数

通过编写Python UDF将点号分割的IP地址转化为int类型的IP，本例中利用DataWorks的*PyODPS节点*完成，详细说明如下。

1. 首先您需要在**数据开发 > 业务流程 > 资源**中右键新建Python类型资源。在弹框中输入新建的Python资源名称，勾选**上传为ODPS资源**，完成创建。。

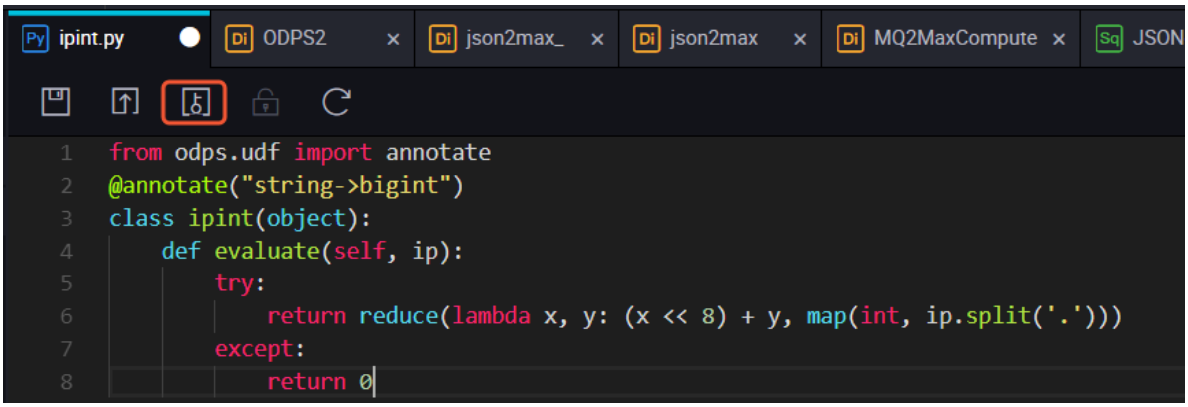


2. 在您新建的Python资源内编写Python资源代码，示例如下。

```
from odps.udf import annotate
@annotate("string->bigint")
class ipint(object):
 def evaluate(self, ip):
  try:
   return reduce(lambda x, y: (x << 8) + y, map(int, ip.split('.')))
  except:
```

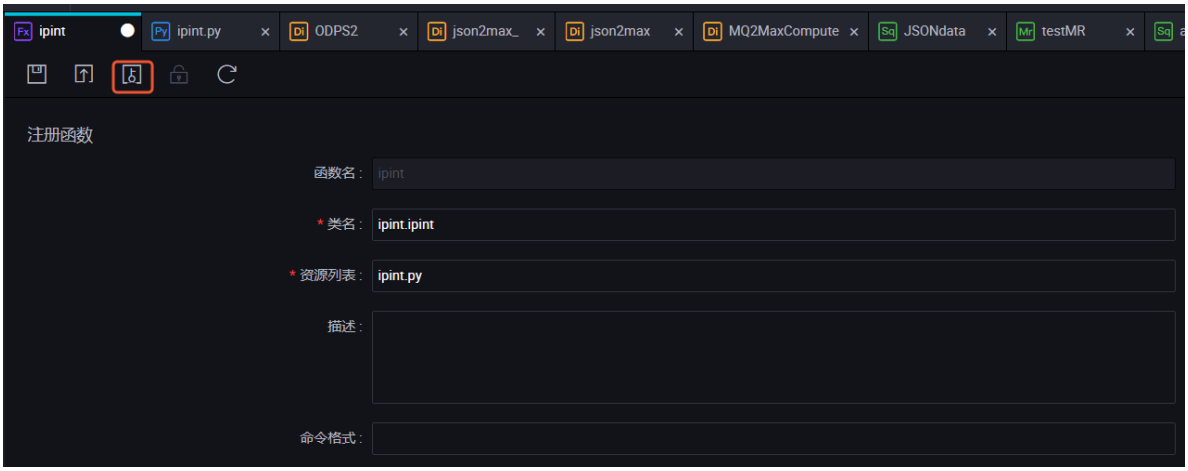
```
    return 0
```

点击提交并解锁。

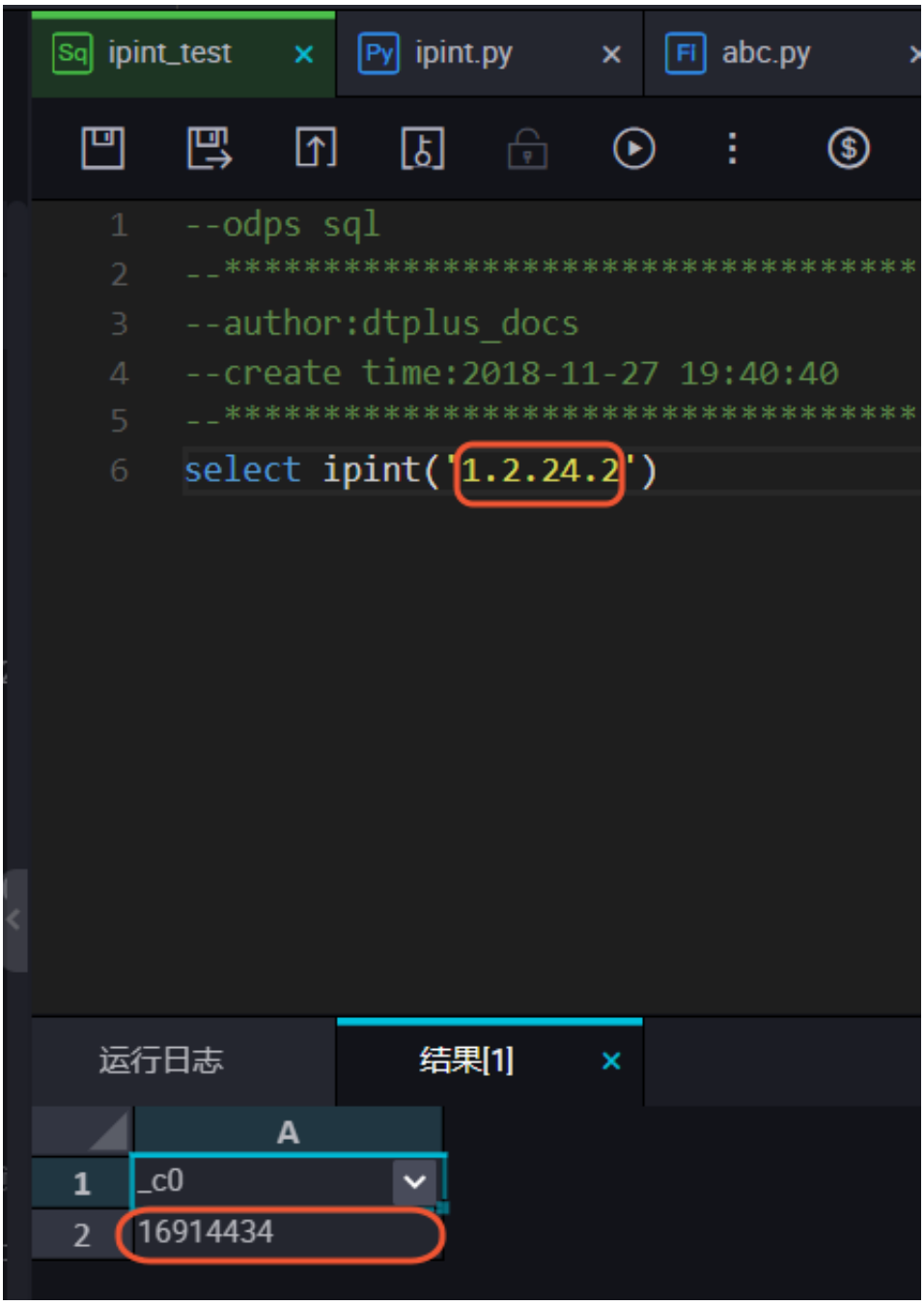


3. 在数据开发 > 业务流程 > 函数中右键新建自定义函数。

   填写函数的类名，本例中为`ipint.ipint`,资源列表填写刚刚我们提交的资源名称，提交并解锁。



4. 验证ipint函数是否生效并满足预期值，您可以在DataWorks上新建一个ODPS SQL类型节点运行SQL语句查询，示例如下。



您也可以在本地创建ipint.py文件，使用*MaxCompute*客户端上传资源。

```
odps@ MaxCompute_DOC>add py D:/ipint.py;
OK: Resource 'ipint.py' have been created.
```

完成上传后，使用客户端直接注册函数。

```
odps@ MaxCompute_DOC>create function ipint as ipint.ipint using ipint.
py;
```

```
Success: Function 'ipint' have been created.
```

完成注册后，即可正常使用该函数，您可以在客户端运行`select ipint('1.2.24.2');`进行测试。

说明：

如果同一主账号下其他项目需要使用这个UDF，您可以进行跨项目授权。

1. 创建名为ipint的package。

```
odps@ MaxCompute_DOC>create package ipint;
OK
```

2. 将已经创建好的UDF函数加入package。

```
odps@ MaxCompute_DOC>add function ipint to package ipint;
OK
```

3. 允许另外一个项目bigdata_DOC安装这个package。

```
odps@ MaxCompute_DOC> allow project bigdata_DOC to install package
ipint;
OK
```

4. 切换到另一个需要使用UDF的项目bigdata_DOC，安装package。

```
odps@ MaxCompute_DOC>use bigdata_DOC;
odps@ bigdata_DOC>install package MaxCompute_DOC.ipint;
OK
```

5. 现在您就可以使用这个UDF函数了， 如果项目空间bigdata_DOC的用户Bob需要访问这些资源，那么管理员可以通过ACL给Bob自主授权。

```
odps@ bigdata_DOC>grant Read on package MaxCompute_DOC.ipint to
user aliyun$bob@aliyun.com; --通过ACL授权Bob使用package
```

在SQL中使用

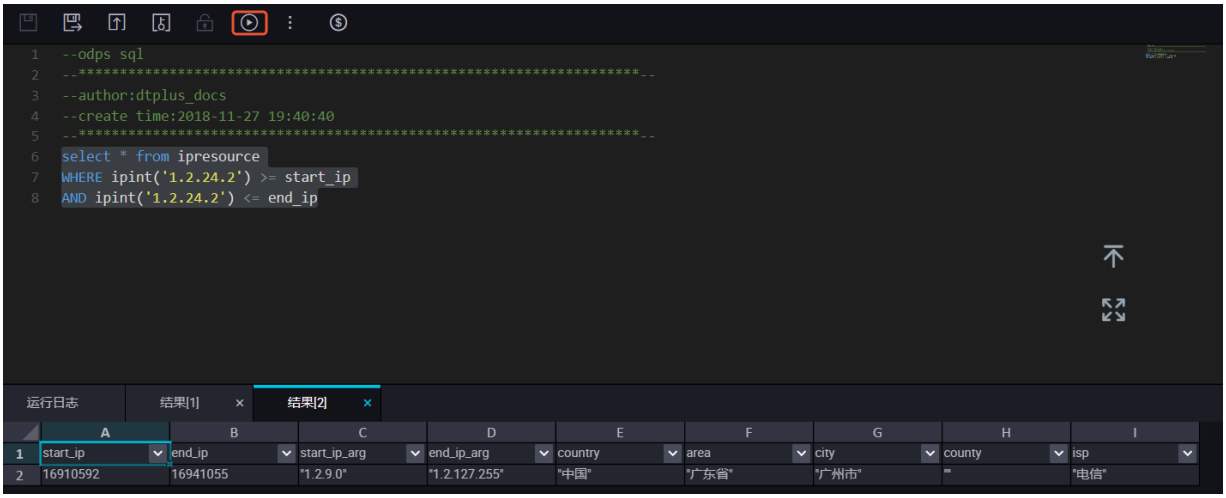
说明：

本例中以一个随机的具体IP 1.2.24.2 地址为例，您在正常使用时可以用具体表的字段来读入。

测试使用的SQL代码如下，点击运行即可查看查询结果。。

```
select * from ipresource
WHERE ipint('1.2.24.2') >= start_ip
```

```
AND ipint('1.2.24.2') <= end_ip
```



通过为保证数据准确性，您可以定期从淘宝IP库获取数据来维护ipresource表。

14 JAVA UDF开发最佳实践

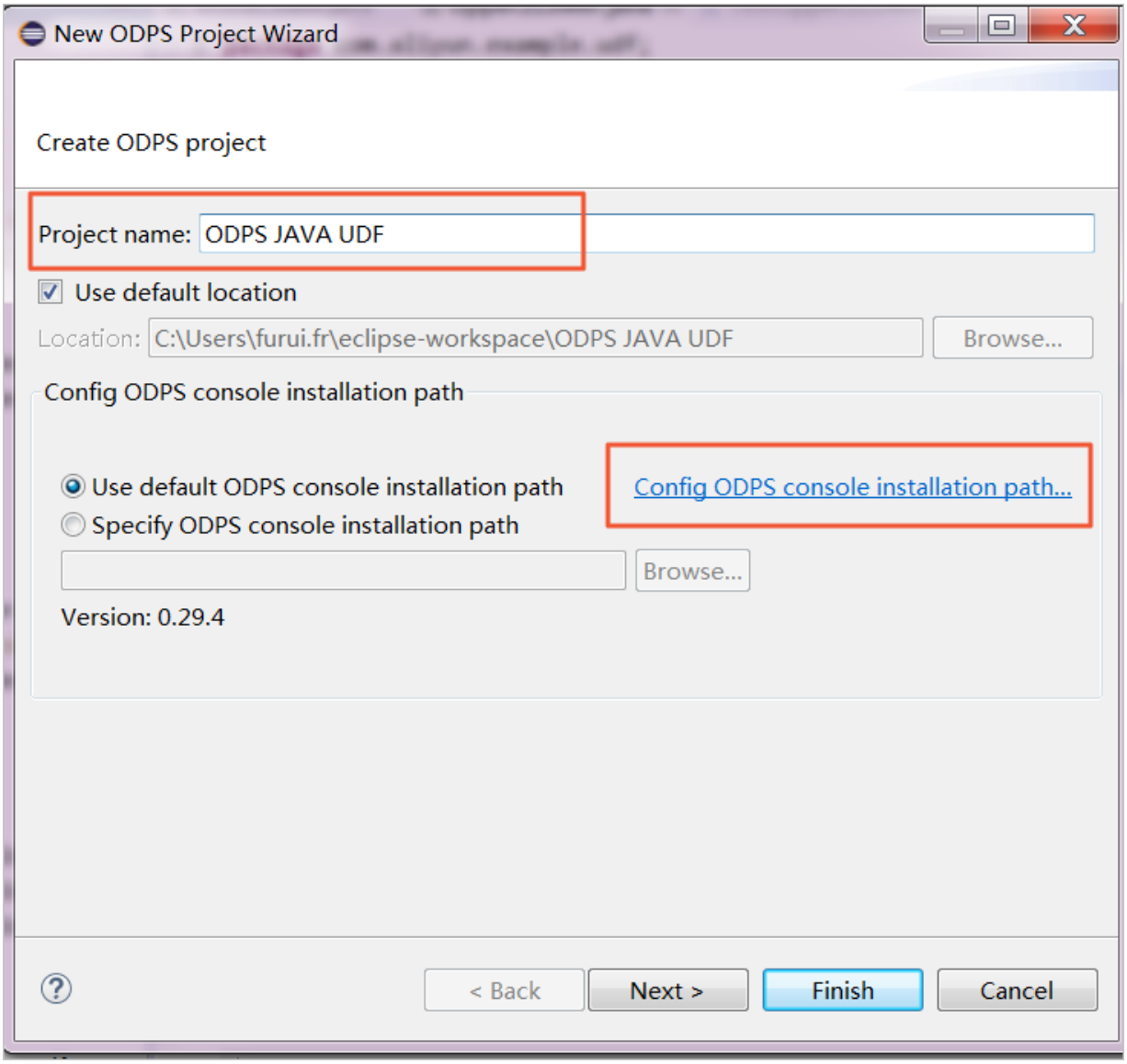
本文为您详细介绍使用Eclipse配合ODPS插件进行JAVA UDF开发的全流程操作。

> 说明：
> 更多资料请参考*JAVA UDF*。

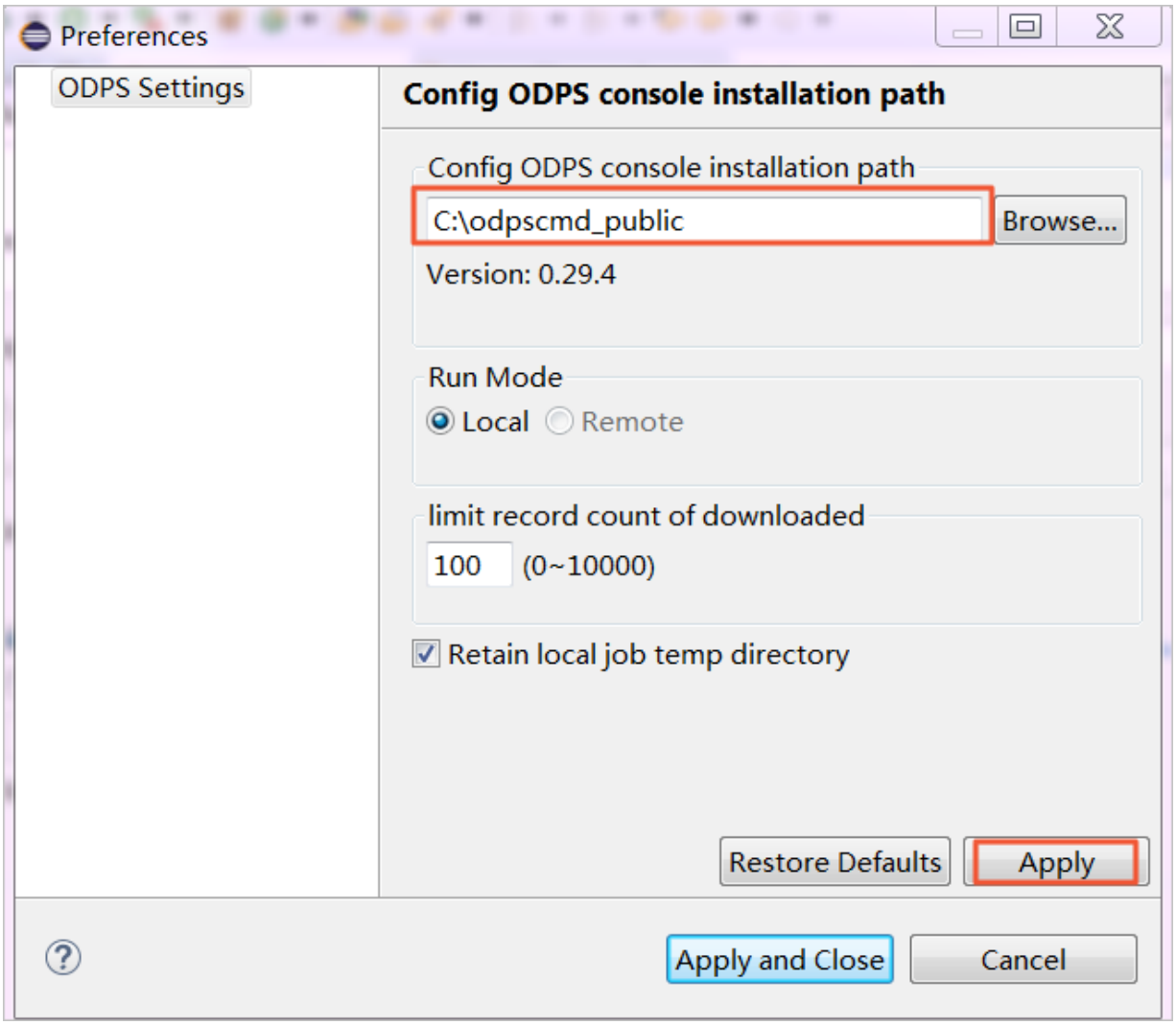
使用Eclipse创建ODPS Project

在进行ODPS Project创建之前，请先确保您已成功安装*ODPS插件*。

在Eclipse中单击**File > New > ODPS Project**，输入您的项目名称后，点击**Config ODPS console installation path**，配置您的*odpscmd客户端*安装路径。



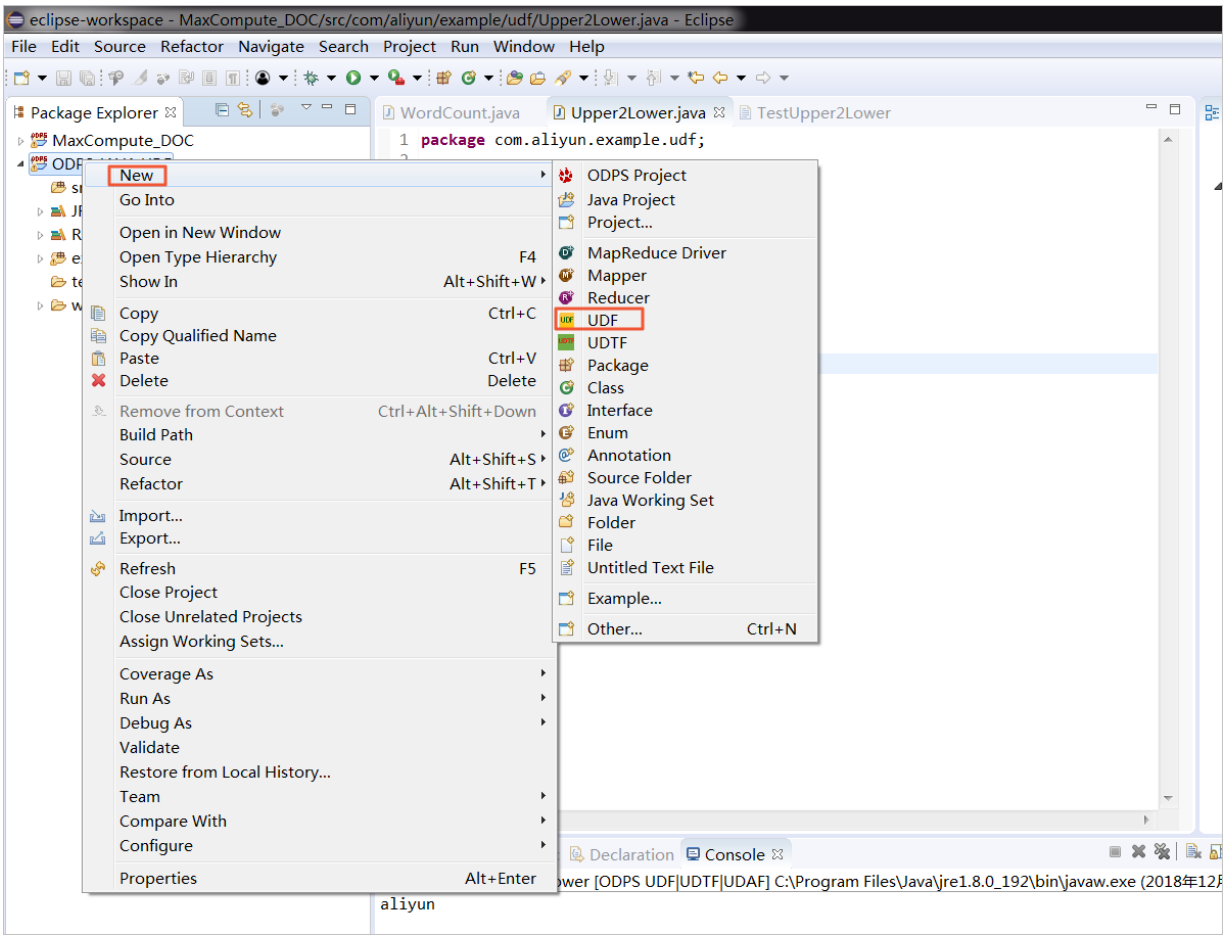
输入您的客户端整体安装包的路径后，点击**Apply**。ODPS插件会为您自动解析出客户端Version。



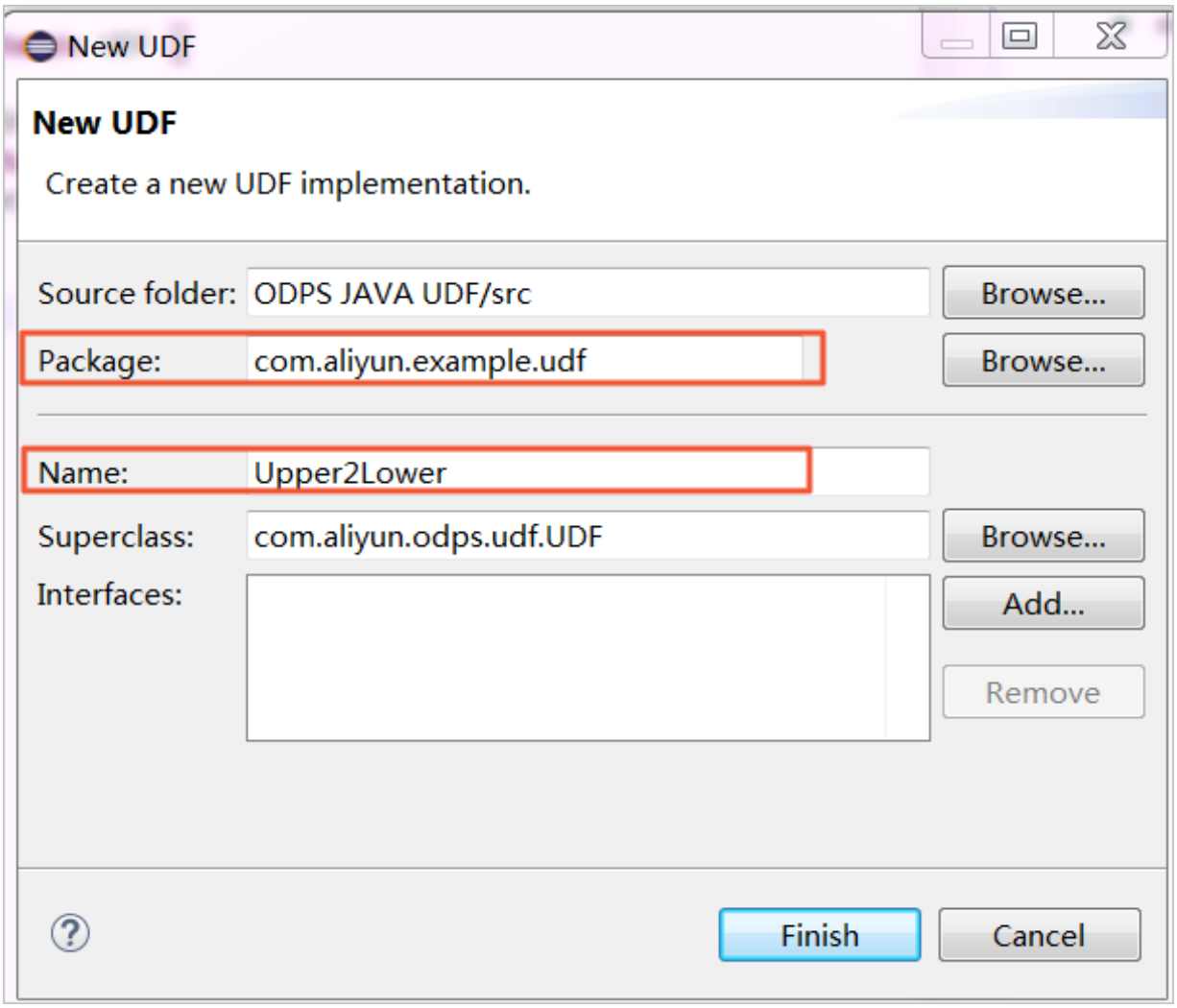
点击**Finish**完成项目创建。

在**ODPS Project**中创建**JAVA UDF**

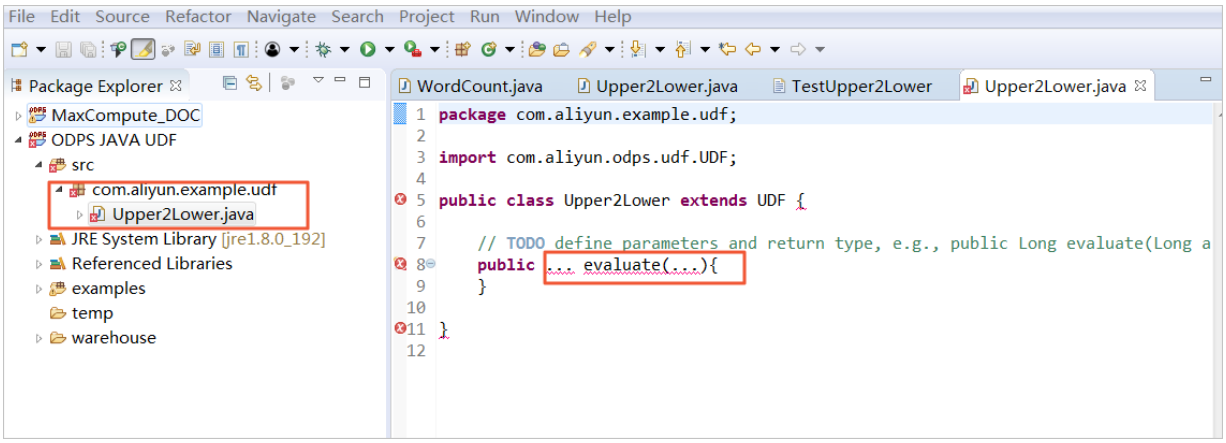
在左侧**Package Exploer**中找到我们刚刚创建的ODPS JAVA UDF项目，右键选**New**，并在列表中选择**UDF**。



输入UDF的Package名称（本例中为com.aliyun.example.udf）和Name（本例中为Upper2Lower），单击**Finish**完成UDF创建。

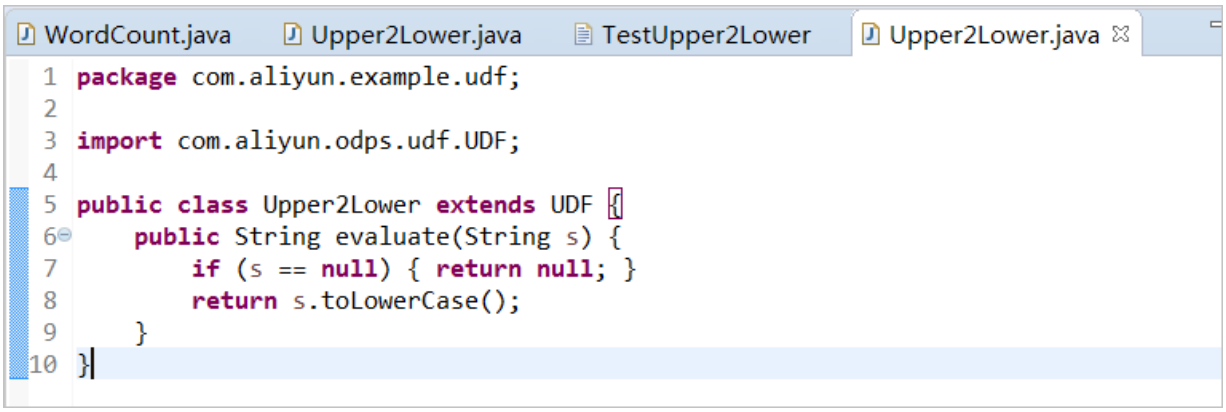


完成UDF创建后，您可以看到生成了默认JAVA代码，请注意不要改变实现**evaluate()**方法名称。



实现UDF类文件中的evaluate方法

将您想要实现的功能代码写到evaluate方法中，且不要改变evaluate()方法的名称。这里为您简单示例一个大写字母转化为小写字母的功能。




```
package com.aliyun.example.udf;

import com.aliyun.odps.udf.UDF;

public class Upper2Lower extends UDF {
    public String evaluate(String s) {
        if (s == null) { return null; }
        return s.toLowerCase();
    }
}
```

完成代码后，请及时保存。

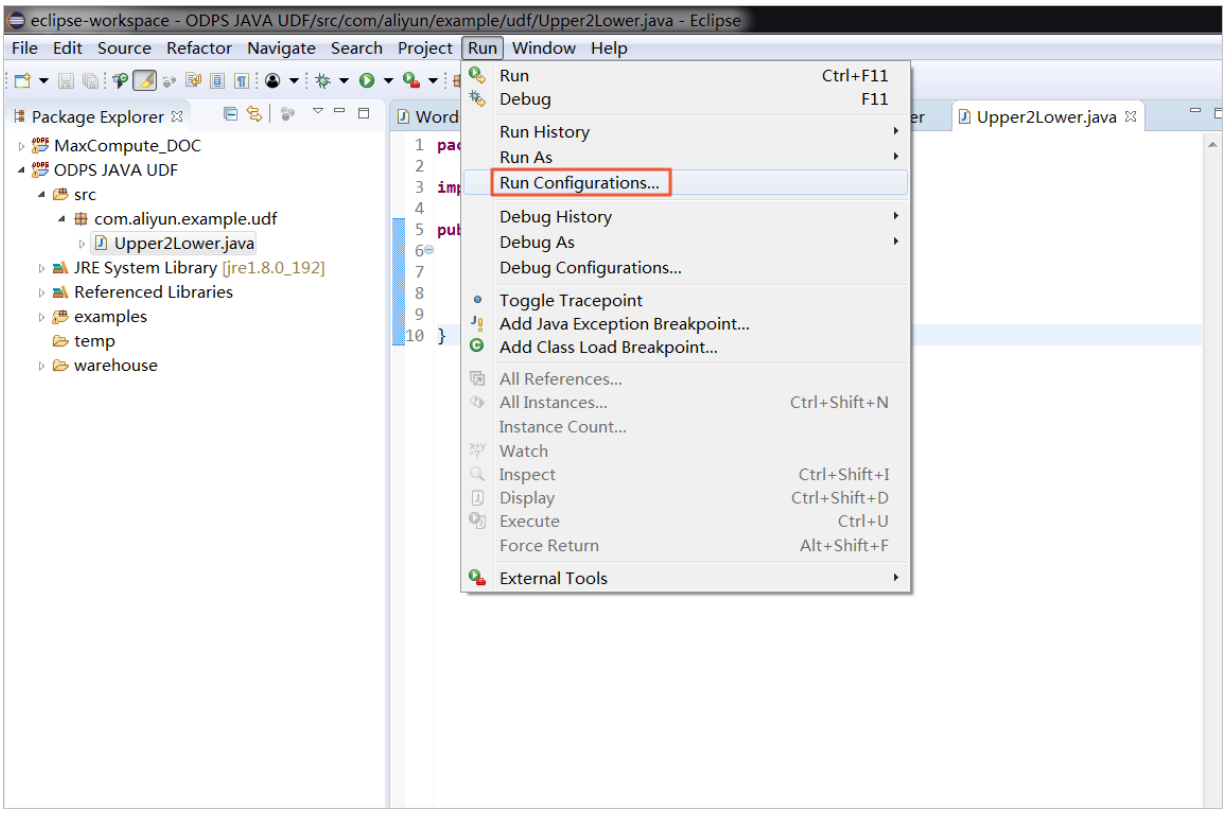
测试JAVA UDF代码

为了测试JAVA UDF代码，我们可以首先在MaxCompute上存放一些大写字母作为输入数据。您可以利用odpscmd客户端使用SQL语句`create table upperABC(upper string);`新建一个名为upperABC的测试表格。

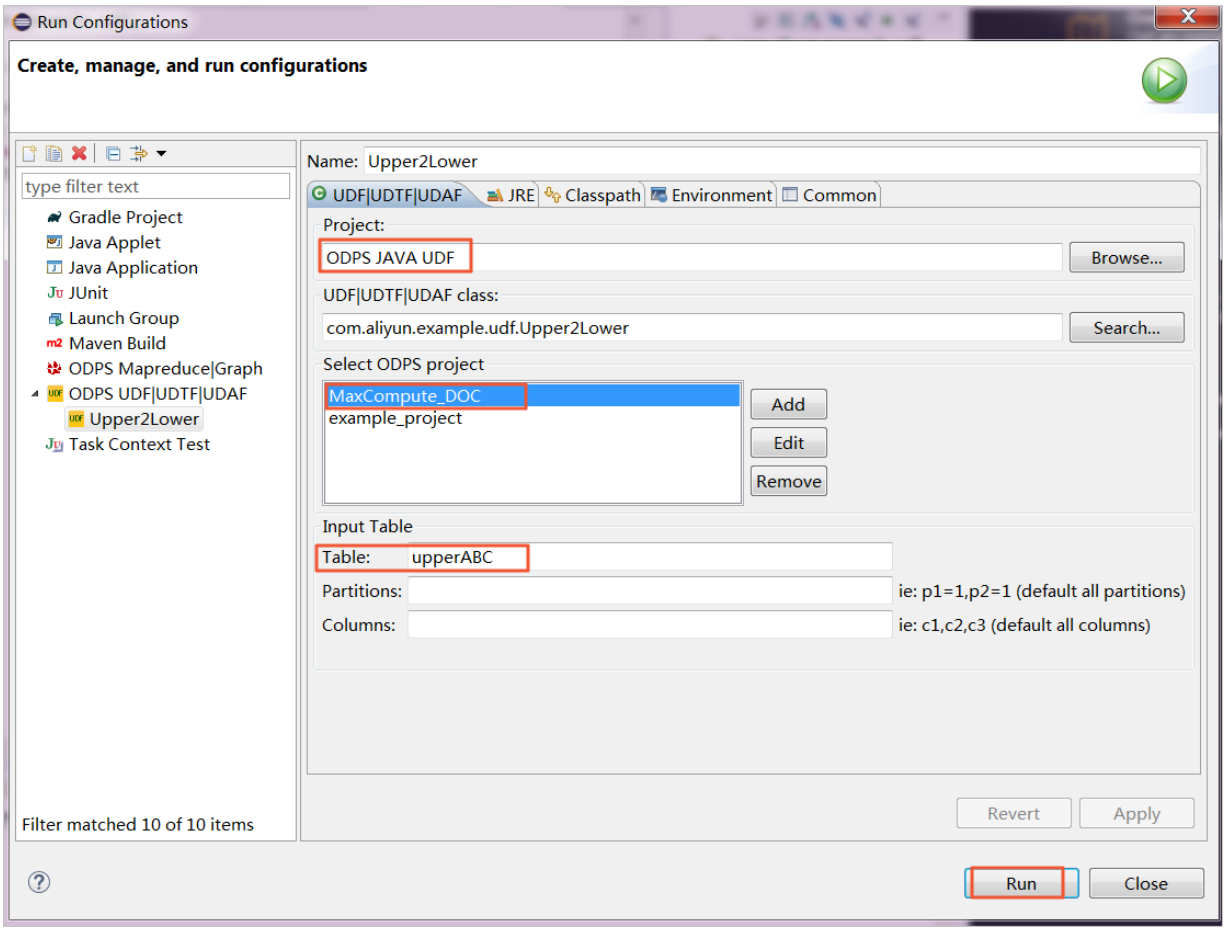
```
odps@ MaxCompute_DOC>create table upperABC(upper string) ;

ID = 20181214094323883gb23j292
OK
```

使用SQL语句`insert into upperABC values('ALIYUN');`在表格中插入测试用的大写字母“ALIYUN”。

完成测试数据准备后，点击**Run > Run Configurations**配置测试参数。



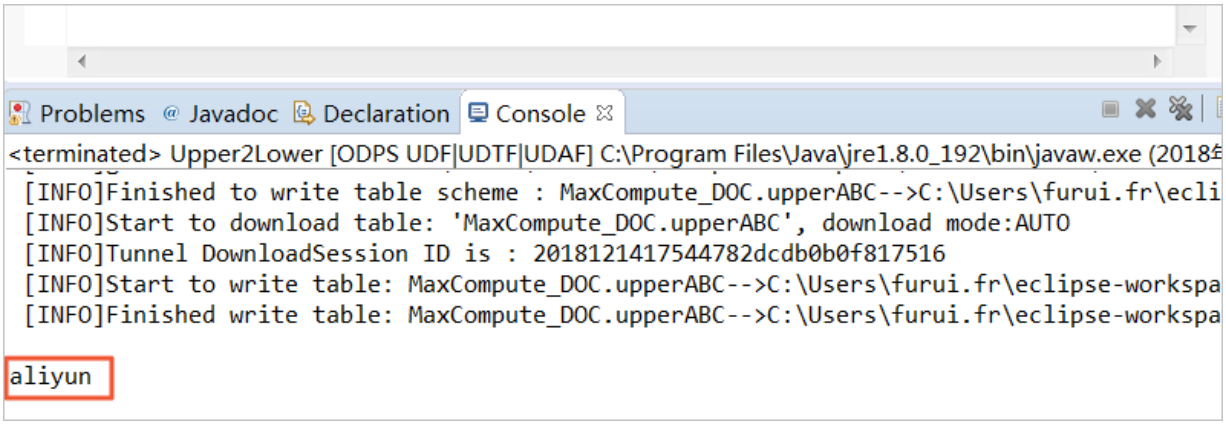
配置测试参数：**Project**一栏中填写我们创建的JAVA ODPS Project名称，**Select ODPS project**中填写您的MaxCompute项目名称（请注意与odpscmd客户端当前连接的MaxCompute项目保持一致），**Table**一栏填写我们刚才创建的测试表格名称。完成配置后点击**Run**进行测试。



您可以在下方的**Console**中看到测试结果，如下图所示。

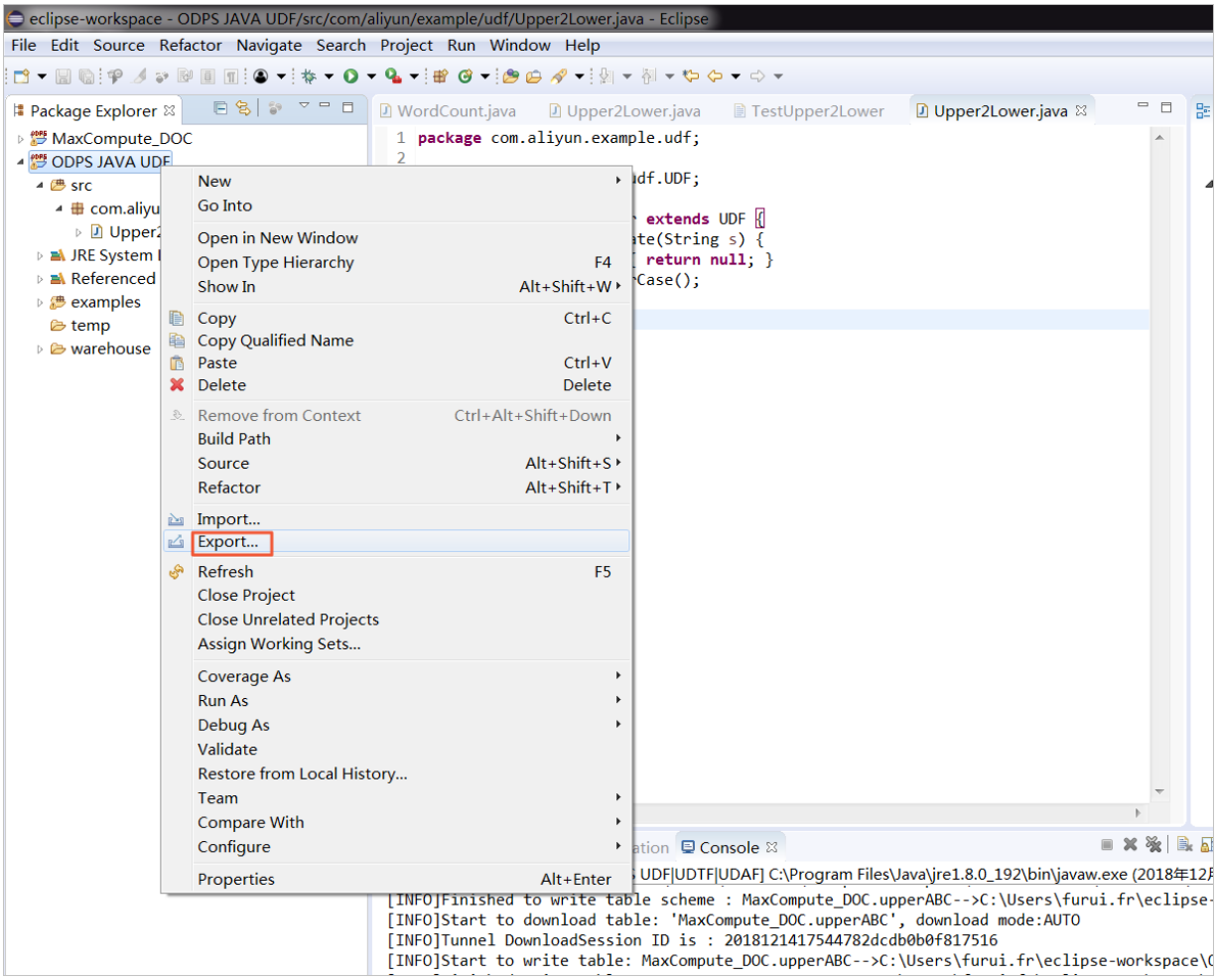
说明：

测试结果只是Eclipse获取表格中的数据后在本地转换的结果，并不代表MaxCompute中的数据已经转换为小写的aliyun了。

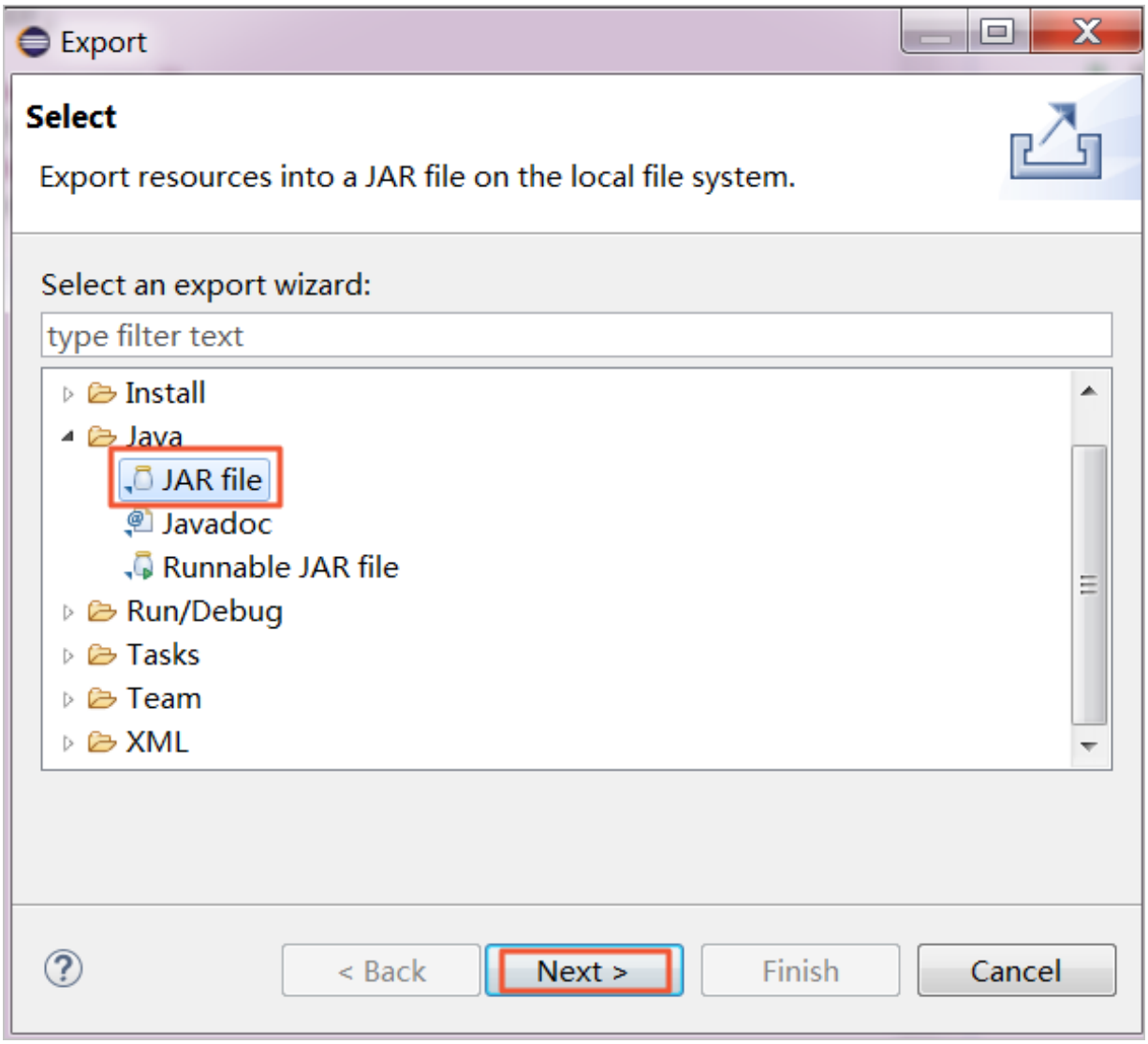


导出Jar包

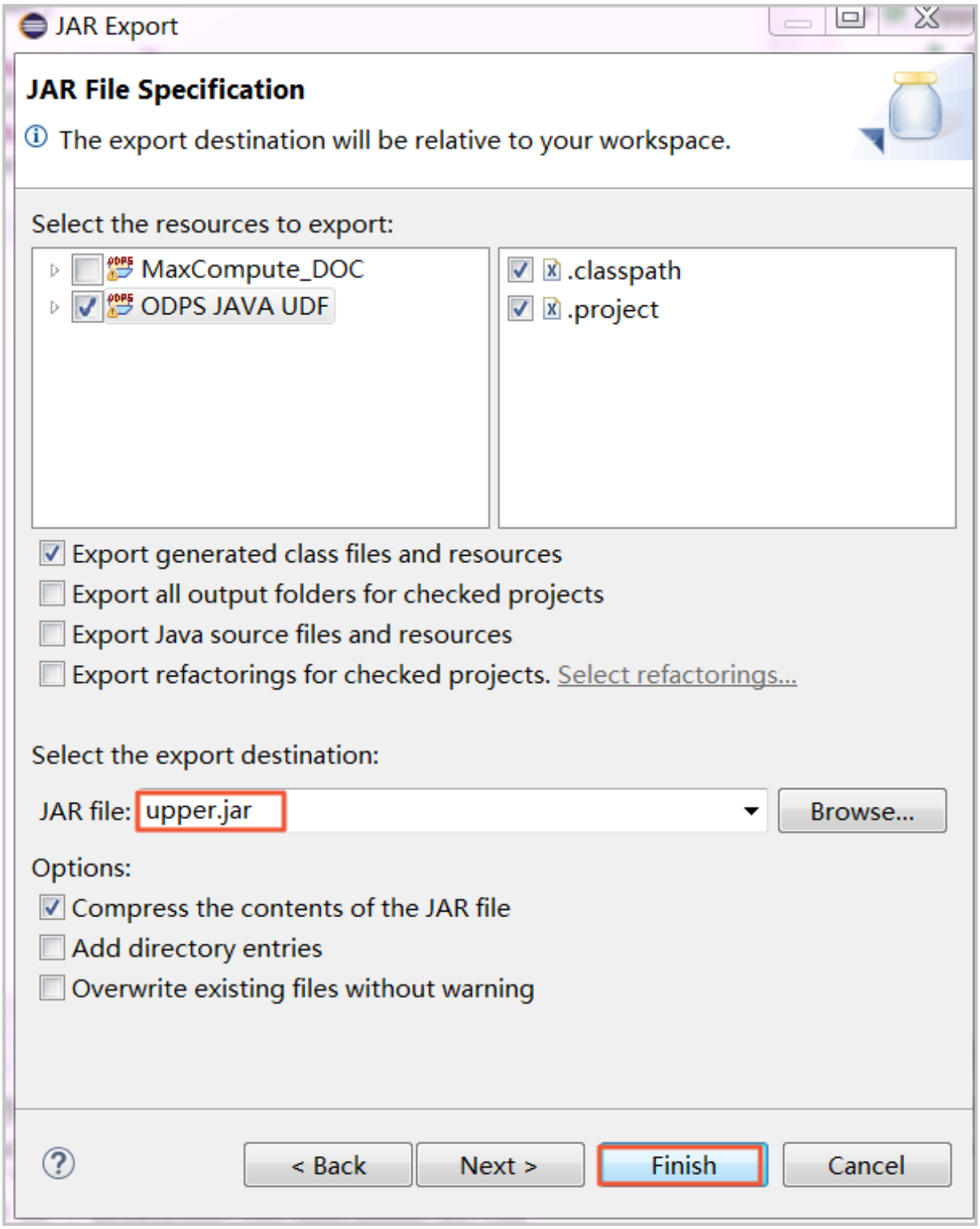
确定测试结果正确后，就可以开始正式使用JAVA UDF了。首先您需要在左侧您新建的ODPS Project上右键单击，选择**Export**。



在弹框中选择**JAR file**，点击**Next**。

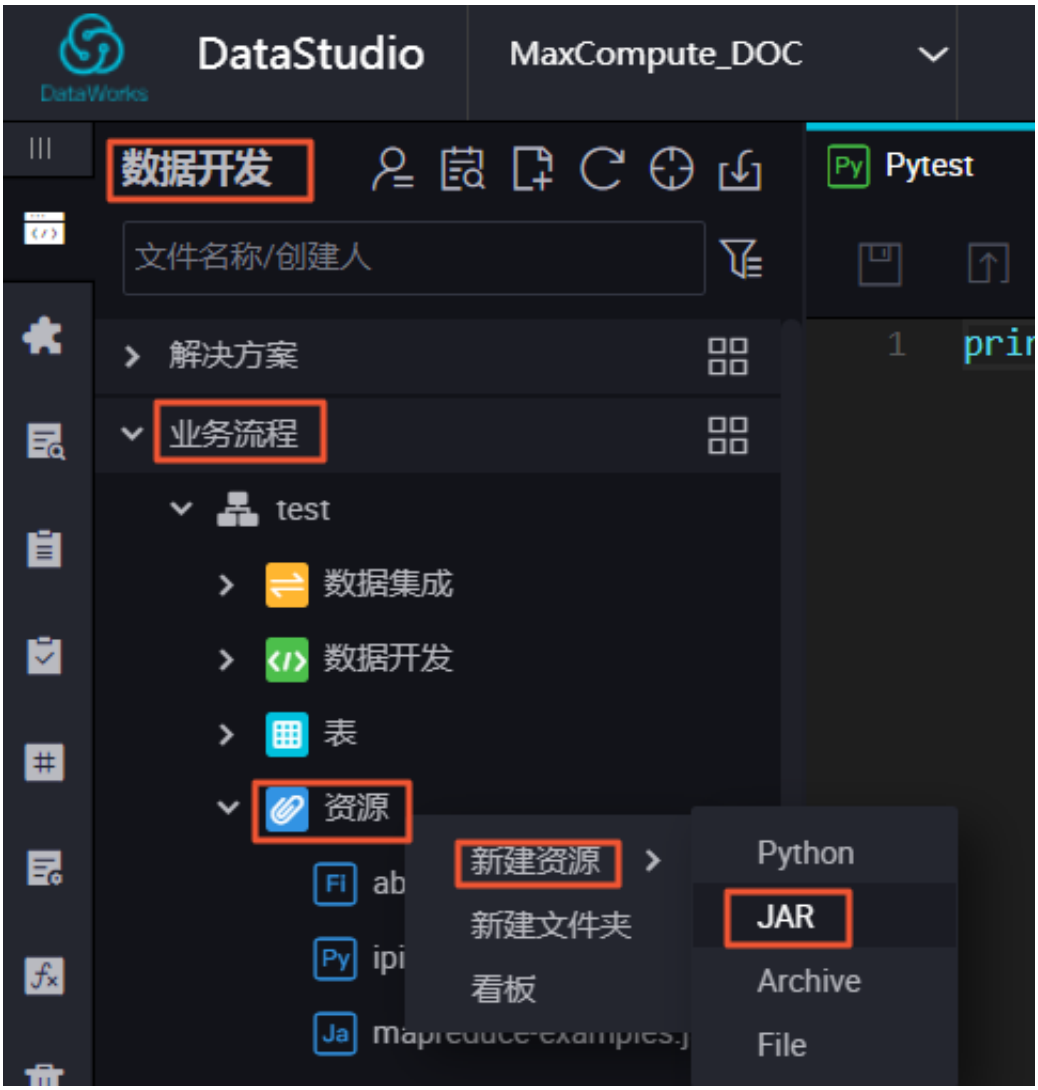


在对话框中JAR file处填写Jar包名称，单击**Finish**即可导出至当前workspace目录下。

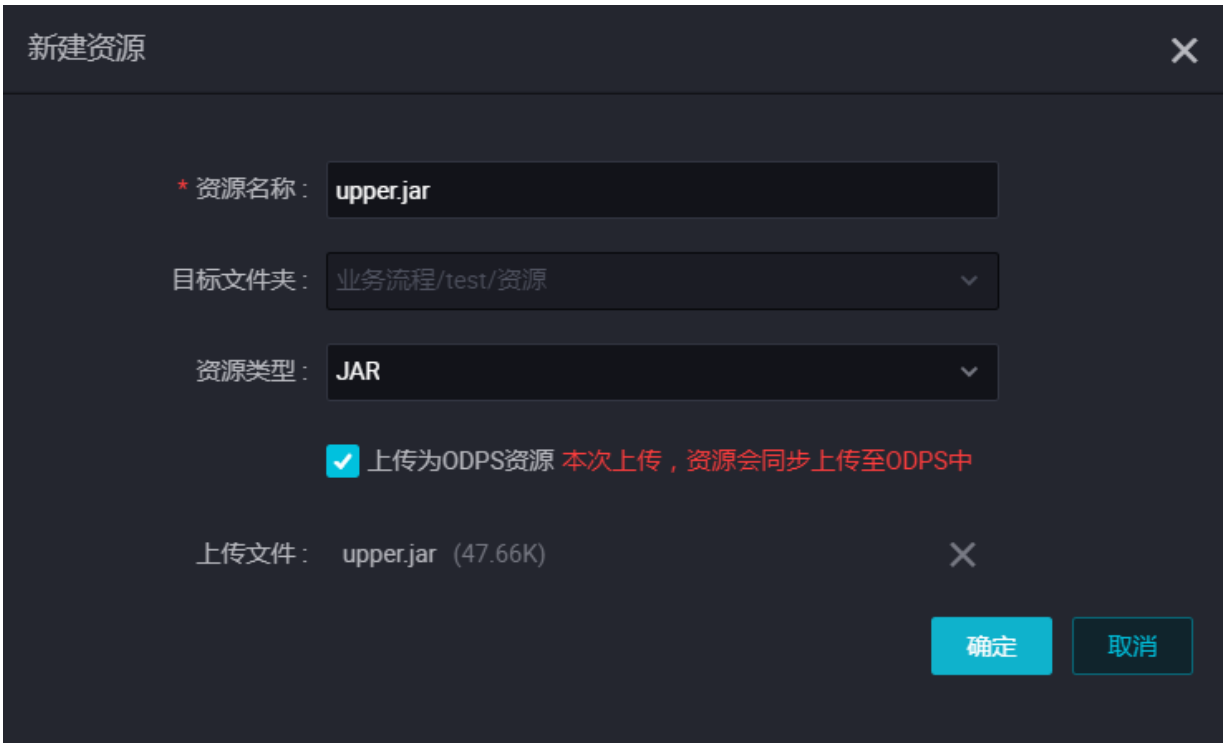


使用DataWorks引用Jar包

登录您的DataWorks控制台，进入同一个项目（本例中为项目MaxCompute_DOC）的数据开发页面。选择业务流程 > 资源 > 新建资源 > **JAR**，新建一个*JAR*类型资源。



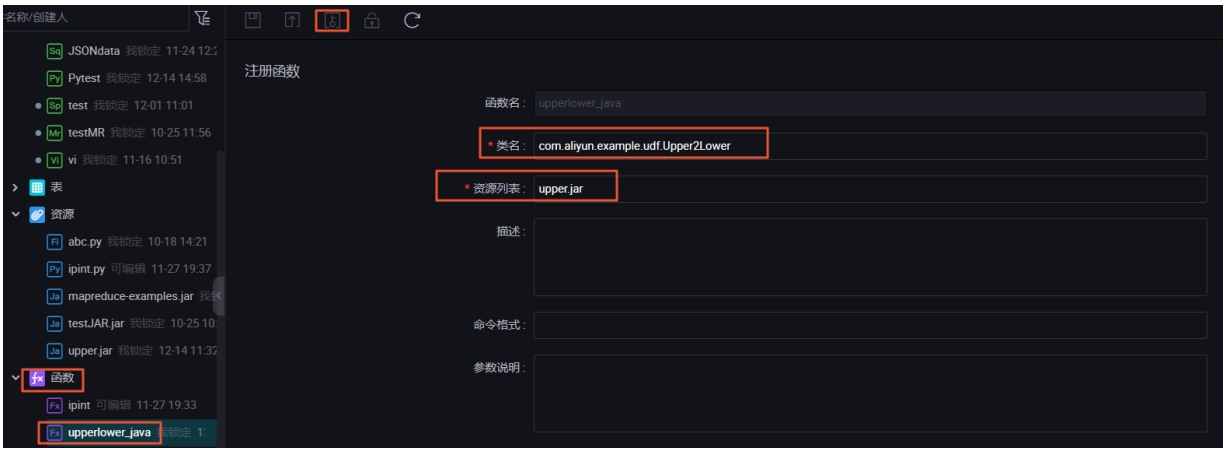
在弹窗中上传您刚导出的Jar资源。



刚才只是将Jar资源上传到DataWorks，接下来您需要点击进入Jar资源，点击**提交并解锁**（**提交**）按钮将资源上传至 MaxCompute。

完成上传后，您可以在odpscmd客户端使用**`list resources`**命令查看您上传的Jar资源。

现在Jar资源已经存在于您的MaxCompute项目中了，接下来您需要点击**业务流程 > 函数 > 新建函数**，新建一个Jar资源对应的函数，本例中命名函数名称为upperlower_java。函数的**资源列表**即为刚上传的Jar资源名称。完成后，依次点击**保存**和**提交并解锁**（**提交**）。



完成提交后，您可以在odpscmd客户端使用**`list functions`**命令查看您注册的函数。到此，您的JAVA UDF函数upperlower_java已经可用了。

JAVA UDF函数结果验证

打开您的odpscmd命令行界面，运行`select upperlower_java('ABCD') from dual;`命令，可以观察到该JAVA UDF已经可以转换字母的大小写了，函数运行正常。