ALIBABA CLOUD

阿里云

大数据计算服务

最佳实践

文档版本：20220601

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 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.SQL

1.1. 与标准SQL的主要区别及解决方法

本文为您列举MaxCompute SQL与标准SQL的区别及常见问题解决方法。

MaxCompute SQL与标准SQL的基本区别

主要区别	问题现象	解决方法
应用场景	不支持事务（不支持Commit和Rollback，不推荐使用INSERT INTO）。	建议代码具备幂等性，支持重新执行。推荐您使用INSERT OVERWRITE写数据。
	不支持索引和主键约束。	无。
	部分字段不支持默认值或默认函数。	如果字段有默认值，您可以在数据写入时自行赋值。MaxCompute支持在创建表时，对BIGINT、DOUBLE、BOOLEAN和STRING类型的字段添加默认值。
	不支持自增字段。	无。
表分区	单表最多支持6万个分区。超过6万个分区会报错。	选择合适的分区列，减少分区数。
	一次查询输入的分区不能超过1万个，否则会报错。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。	解决方法请参见执行INSERT INTO或INSERT OVERWRITE操作时，报错a single instance cannot output data to more than 10000 partitions，如何解决？。
精度	DOUBLE类型存在精度问题。	不建议直接使用等于号（=）关联两个DOUBLE字段。建议将两个数相减，如果差距小于一个预设的值，则认为两个数是相同的。例如 `ABS(a1-a2)<0.000000001` 。
	虽然MaxCompute支持高精度类型DECIMAL，但是有更高精度的要求。	如果有更高的精度要求，您可以先把数据存储为STRING类型，然后使用UDF实现对应的计算。
数据类型转换	出现各种预期外的错误，代码维护问题。	如果有2个不同的字段类型需要执行JOIN操作，建议您先转换字段类型再执行JOIN操作。
	日期类型和字符串的隐式转换。	如果在需要传入日期类型的函数中传入一个字符串，字符串和日期类型根据 `yyyy-mm-dd hh:mi:ss` 格式进行转换。

DDL与DML的区别及解决方法

主要区别	问题现象	解决办法
表结构	不能修改分区列列名，只能修改分区列对应的值。	解决方案请参见分区和分区列的区别是什么？。
	支持增加列，但是不支持删除列及修改列的数据类型。	解决方案请参见如何修改列的数据类型？和如何删除列？。
INSERT	MaxCompute SQL需要在INSERT INTO或INSERT OVERWRITE后加关键字TABLE。	无。
	数据插入表的字段映射不是根据SELECT的别名执行，而是根据SELECT字段的顺序和表中字段的顺序执行映射。	无。
UPDATE和DELETE	仅支持对Transactional表执行UPDATE和DELETE语句。	解决方案请参见如何删除MaxCompute表或分区中的数据？和如何更新MaxCompute表或分区中的数据？。
SELECT	MaxCompute SQL最多支持6张小表的MAPJOIN，并且连续JOIN的表不能超过16张。	解决方案请参见在执行JOIN操作时，报错Maximum 16 join inputs allowed，如何解决？。

主要区别	问题现象	解决办法
IN和NOT IN	IN、NOT IN、EXIST和NOT EXIST，后面的子查询返回的分区数据量不能超过1000条。	解决方案请参见在执行MaxCompute SQL过程中，使用NOT IN后面接子查询，子查询返回的结果是上万级别的数据量，但当IN和NOT IN后面的子查询返回的是分区时，返回的数量上限为1000。在必须使用NOT IN的情况下，该如何实现此查询？。如果业务上已经保证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。	解决方案请参见LIMIT限制输出行数（number）。
	需要查询的结果数据条数很多。	解决方案请参见使用SQLTask执行SQL查询时，如果查询结果条数大于限制的1000条，该如何获取所有数据？。
MAPJOIN	JOIN不支持笛卡尔积。	JOIN必须要用ON关键字设置关联条件。 如果有一些小表要作为广播表，需要使用MAPJOIN HINT。
ORDER BY	ORDER BY需要配合LIMIT N使用。	如果希望执行大数据量的排序任务，甚至是全表排序任务，可以增大N值。解决方案请参见MaxCompute查询得到的数据是根据什么排序的？。
UNION ALL	参与UNION ALL运算的所有表必须列数一致，否则会报错。	参与UNION ALL运算的所有列的数据类型、列个数和列名称必须完全一致。
	UNION ALL需要再嵌套一层子查询。	无。

1.2. MaxCompute SQL示例解析

本文为您介绍MaxCompute SQL常见使用场景，让您快速掌握SQL的写法。

准备数据集

本文以emp表和dept表为示例数据集。您可以自行在MaxCompute项目上创建表并上传数据。数据导入请参见概述。

下载emp表数据文件和dept表数据文件。

创建emp表。

```
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。

```
CREATE TABLE IF NOT EXISTS dept (
  DEPTNO BIGINT,
  DNAME STRING,
  LOC STRING);
```

SQL示例

- 示例1：查询员工人数大于零的所有部门。

  为了避免数据量太大，此场景下建议您使用JOIN子句。

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

- 示例2：查询薪金比SMITH高的所有员工。

此场景为MAPJOIN的典型场景。

```
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;
```

- 示例3：查询所有员工的姓名及其直接上级的姓名。

  此场景为等值连接。

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

- 示例4：查询基本薪金大于1500的所有工作。

  此场景下需要使用HAVING子句。

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

- 示例5：查询在每个部门工作的员工数量、平均工资和平均服务期限。

  此场景为使用内建函数的典型场景。

```
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`;
```

- 示例6：查询每个部门的薪水前3名的人员的姓名以及其排序。

  此场景为典型的Top N场景。

```
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;
```

- 示例7：查询每个部门的人数以及该部门中办事员（CLERK）人数的占比。

```
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;
```

注意事项

- 使用GROUP BY时，SELECT部分必须是分组项或聚合函数。
- 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，可以使用MAPJOIN替代。
- 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操作执行。

1.3. 不兼容SQL重写

本文为您介绍如何修改不兼容SQL。

背景信息

MaxCompute 2.0完全拥抱开源生态，支持更多的语言功能，拥有更快的运行速度。但是MaxCompute 2.0会执行更严格的语法检测，一些在旧版本编译器下正常执行的不严谨的语法在MaxCompute 2.0下执行会报错。

group.by.with.star

说明：即 `select * …group by…` 语句。

- MaxCompute 2.0版本中，要求Group By列表是源表中所有的列，否则执行报错。
- 旧版MaxCompute中，即使Group By列表不覆盖源表中所有的列，也支持 `select * from group by key` 语法。

示例

- 场景1：Group By Key中不包含所有列。
  - 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
    ```

  - 正确写法

    ```
    select distinct key from t;
    ```

- 场景2：group by key包含所有列。
  - 如下写法不推荐。

    ```
    select * from t group by key, value; -- t has columns key and value
    ```

  - 虽然MaxCompute2.0不会报错，但推荐改为如下。

    ```
    select distinct key, value from t;
    ```

bad.escape

说明：错误的escape序列问题。

按照MaxCompute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。但\01、\0001也被当作\001处理了。

这种方式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` 后面再加数字，如 `\0001 - \0009` 或 `\00001` 的写法可能会返回错误。

MaxCompute 2.0会解决此问题，对脚本中错误的序列进行修改。

- 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

说明：如果创建表时列名重复，MaxCompute 2.0将会报错。

示例

- 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
```

- 正确写法

```
create table t (a BIGINT, b BIGINT);
```

string.join.double

说明：写JOIN条件时，等号的左右两边分别是STRING和DOUBLE类型。

- 旧版MaxCompute会把两边都转成BIGINT类型，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。
- MaxCompute 2.0会与Hive兼容转为DOUBLE类型。

示例

- 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);
```

window.ref.prev.window.alias

说明：Window Function引用同级select List中的其他Window Function Alias的问题。

示例

- 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

说明：select列表里面允许用户使用星号（*）代表选择某张表的全部列，但星号（*）后面不允许加 `alias` ，即使星号（*）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

- 错误写法

```
select * as alias from table_test;
```

- 报错信息

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

- 正确改法

```
select * from table_test;
```

agg.having.ref.prev.agg.alias

说明：有Having子句的情况下，select List可以出现前面Aggregate Function Alias的问题。

示例

- 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子句，旧版MaxCompute并未报错，MaxCompute 2.0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

说明：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

- 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
```

在子查询 `order by v.uv desc` 中增加 `limit` 。

另外，MaxCompute 1.0对于view的检查不够严格。比如在一个不需要检查 `limit` 的Project（odps.sql.validate.orderby.limit=false）中，创建了一个View。

```
create view table_view as select id from table_view order by id;
```

若访问此View：

```
select * from table_view;
```

MaxCompute 1.0不会报错，而MaxCompute 2.0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

说明：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。

MaxCompute 2.0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

- 不推荐写法

```
select _c0 from (select count(*) from table_name) t;
```

- 建议改法：

```
select c from (select count(*) c from table_name) 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。MaxCompute 2.0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 table_name 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 table_name 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，MaxCompute 2.0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 table_test
order by id;
```

正确改法：

```
select id, id id2
from table_name
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 table_name
where not_exist_col in (select id from table_name 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，MaxCompute 2.0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 table_name
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 perf
ormance.
```

MaxCompute 2.0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发送心跳而导致超时。

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 table_name;
in task M1_Stg1:
    Data source: meta_dev.table_name
    TS: alias: table_name
      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MaxCompute 2.0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 table_name;
```

报错信息：

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

除了上述的错误，还可能遇到overflow错误，比如：

错误写法：

```
select if(false, 1/0, 1.0)
from table_name;
```

报错信息：

```
FAILED: ODPS-0130071:[1,19] Semantic analysis exception - encounter runtime exception while evaluating function /, detailed mess
age: 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，MaxCompute 2.0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。

MaxCompute 2.0 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, mem
ory 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。
```

MaxCompute 2.0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.Aggr
egateCallValidator$AggregateCallValidationException: Invalid argument type - The first argument of WM_CONCAT must be constant st
ring.
```

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，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MaxCompute 2.0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

- 错误写法：

  ```
  select id id2 from table_name 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

说明：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

```
insert overwrite table srcpt partition(pt) select id, 'pt1' from table_name;
```

会被转化成

```
insert overwrite table srcpt partition(pt='pt1') select id from table_name;
```

如果用户指定的分区值不合法，比如错误地使用了'${bizdate}'，MaxCompute 2.0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 table_name 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

说明：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。

MaxCompute 2.0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

```
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。

- 正确写法

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

1.4. 导出SQL的运行结果

本文将通过示例，为您介绍导出MaxCompute SQL计算结果的方法。

> **说明** 本文中所有SDK部分仅以Java举例。

概述

您可以通过以下方法导出SQL的运行结果：

- 如果数据比较少，请使用SQL Task得到全部的查询结果。
- 如果需要导出某个表或者分区，请使用Tunnel直接导出查询结果。
- 如果SQL比较复杂，请使用Tunnel和SQL相互配合导出查询结果。
- DataWorks 可以方便地帮您运行SQL，同步数据，并支持定时调度，配置任务依赖的功能。
- 开源工具DataX可以帮助您方便地把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

SQLTask使用SDK方法，直接调用MaxCompute SQL的接口SQLTask.getResult(i)，可以很方便地运行SQL并获得其返回结果。使用方法请参见SQLTask。

使用SQLTask时，请注意：

- SQLTask.getResult(i)用于导出SELECT查询结果，不适用于导出**show tables;**等其他MaxCompute命令操作结果。
- SELECT语句返回给客户端的数据条数可以通过READ_TABLE_MAX_ROW进行设置，详情请参见项目空间操作。
- SELECT语句最多返回1万条数据至客户端。即如果在客户端（包括SQLTask）直接执行SELECT语句，相当于在SELECT语句最后加了Limit N。

  如果使用**CREATE TABLE XX AS SELECT**或者**INSERT INTO/OVERWRITE TABLE**把结果固化到具体的表里则不受此限制。

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。

此处提供一个Tunnel命令行导出数据的简单示例，Tunnel SDK的编写适用于Tunnel命令行无法支持的场景，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配合Tunnel方式导出

SQLTask不能处理超过1万条数据，而Tunnel方式可以，两者可以互补，因此可以基于两者实现超过1万条数据的导出。

代码实现的示例如下。

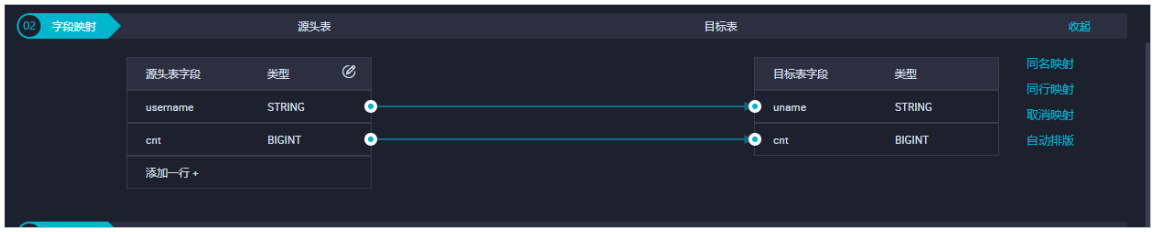
```
private static final String accessId = "userAccessId";
    private static final String accessKey = "userAccessKey";
    private static final String endPoint = "http://service.cn-shanghai.maxcompute.aliyun.com/api";
    private static final String project = "userProject";
    private static final String sql = "userSQL";
    private static final String table = "Tmp_" + UUID.randomUUID().toString().replace("-", "_");//用随机字符串作为临时表的名称。
    private static final Odps odps = getOdps();
    public static void main(String[] args) {
        System.out.println(table);
        runSql();
        tunnel();
    }
    /*
     * 下载SQLTask的结果。
     * */
    private static void tunnel() {
        TableTunnel tunnel = new TableTunnel(odps);
        try {
            DownloadSession downloadSession = tunnel.createDownloadSession(
                    project, table);
            System.out.println("Session Status is : "
                    + downloadSession.getStatus().toString());
            long count = downloadSession.getRecordCount();
            System.out.println("RecordCount is: " + count);
            RecordReader recordReader = downloadSession.openRecordReader(0,
                    count);
            Record record;
            while ((record = recordReader.read()) != null) {
                consumeRecord(record, downloadSession.getSchema());
            }
            recordReader.close();
        } catch (TunnelException e) {
            e.printStackTrace();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }
    /*
     * 保存数据。
     * 数据量少时直接打印后拷贝也是可行的。实际场景可以用Java.io写到本地文件，或者写到远端存储上保存起来。
     * */
    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。
     * 此处保存数据的生命周期为1天，即使删除步骤出了问题，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;
    }
```

DataWorks数据同步方式导出

DataWorks支持运行SQL并配置数据同步任务，以完成数据生成和导出需求。

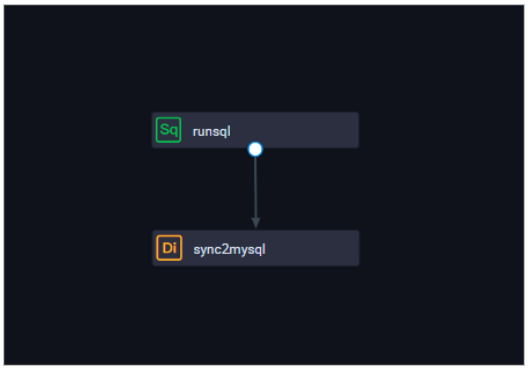
1. 登录DataWorks控制台。
2. 在左侧导航栏，单击**工作空间列表**。
3. 单击相应工作空间后的**进入数据开发**。
4. 新建业务流程。
   i. 右键单击**业务流程**，选择**新建业务流程**
   ii. 输入**业务名称**。
   iii. 单击**新建**。
5. 创建SQL节点。
   i. 右键单击业务流程，选择**新建 > MaxCompute > ODPS SQL**。
   ii. 填写**节点名称**为runsql，单击**提交**。
   iii. 配置ODPS SQL节点，配置完成后单击**保存**。
6. 创建数据同步节点。
   i. 右键单击业务流程，选择**新建 > 数据集成 > 离线同步**。
   ii. 填写**节点名称**为sync2mysql，单击**提交**。
   iii. 选择数据来源以及去向。

   iv. 配置字段映射。



   v. 配置通道控制。

   vi. 单击**保存**。
7. 将数据同步节点和ODPS SQL节点连线配置成依赖关系，ODPS SQL节点作为数据的产出节点，数据同步节点作为数据的导出节点。



8. 工作流调度配置完成后（可以直接使用默认配置），单击**运行**。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
```

9. 执行如下SQL语句查看数据同步的结果。

```
select count(*) from result_in_db;
```

1.5. 分区剪裁合理性评估

本文为您介绍如何评估分区剪裁合理性。

背景信息

MaxCompute分区表是指在创建表时指定分区空间，即指定表内的几个字段作为分区列。使用数据时，如果指定了需要访问的分区名称，则只会读取相应的分区，避免全表扫描，提高处理效率，降低费用。

分区剪裁是指对分区列指定过滤条件，使得SQL执行时只用读取表分区的部分数据，避免全表扫描引起的数据错误及资源浪费。但是分区失效的情况会经常发生。

本文将从以下两方面介绍分区剪裁：

- 判断分区剪裁是否生效。
- 分区剪裁失效的场景分析。

判断分区剪裁是否生效

通过EXPLAIN命令查看SQL执行计划，用于判断SQL中的分区剪裁是否生效。

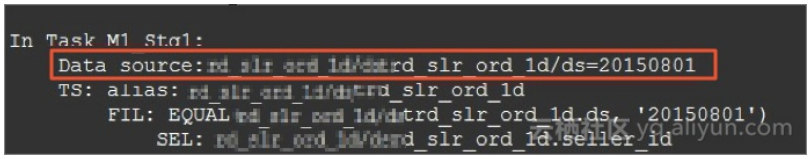
- 分区剪裁未生效。

```
explain
select seller_id
from xxxxx_trd_slr_ord_1d
where ds=rand();
```

从执行计划中可见，SQL读取了表的1344个分区，即该表的所有分区。

- 分区剪裁生效

```
explain
select seller_id
from xxxxx_trd_slr_ord_1d
where ds='20150801';
```



从执行计划中可见，SQL只读取了表的20150801分区。

分区剪裁失效的场景分析

- 自定义函数导致分区剪裁失效

  当分区剪裁的条件中使用了用户自定义函数（或者部分内建函数）时，分区剪裁失效。所以，对于分区值的限定，如果使用了非常规函数，建议您使用Explain命令查看执行计划，确定分区剪裁是否生效。

```
explain
select ...
from xxxxx_base2_brd_ind_cw
where ds = concat(SPLIT_PART(bi_week_dim(' ${bdp.system.bizdate}'), ',', 1), SPLIT_PART(bi_week_dim(' ${bdp.system.bizdate}'),
',', 2))
```

  > 说明 UDF已支持分区裁剪，详情请参见WHERE子句（where_condition）文中的说明。

- 使用JOIN时分区剪裁失效

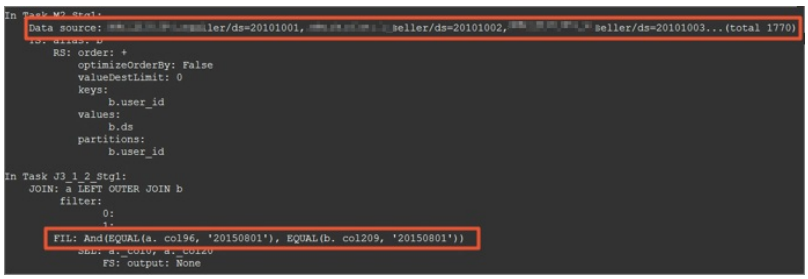
  在SQL语句中使用JOIN进行关联时：

  - 如果分区剪裁条件放在WHERE子句中，则分区剪裁会生效。
  - 如果分区剪裁条件放在ON子句中，从表的分区剪裁会生效，主表则不会生效。

  下面针对三种JOIN具体说明：

  - LEFT OUTER JOIN
    - 分区剪裁条件均放在ON子句中

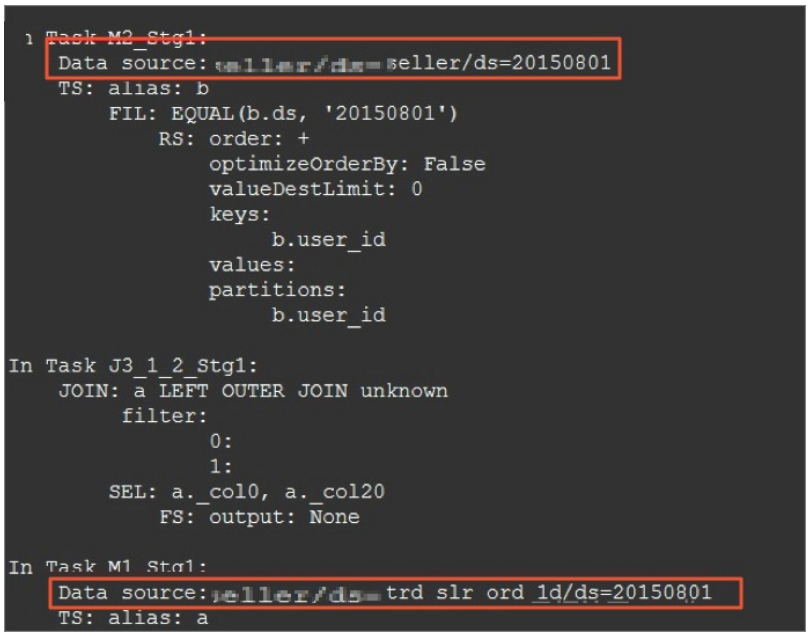
```
set odps.sql.allow.fullscan=true;
explain
select a.seller_id
    ,a.pay_ord_pbt_1d_001
from xxxxx_trd_slr_ord_1d a
left outer join
     xxxxx_seller b
on a.seller_id=b.user_id
and a.ds='20150801'
and b.ds='20150801';
```



    由上图可见，左表进行了全表扫描，只有右表的分区裁剪有效果。

- 分区剪裁条件均放在WHERE子句中

```
set odps.sql.allow.fullscan=true;
explain
select a.seller_id
    ,a.pay_ord_pbt_1d_001
from xxxxx_trd_slr_ord_1d a
left outer join
    xxxxx_seller b
on a.seller_id=b.user_id
where a.ds='20150801'
and b.ds='20150801';
```



由上图可见，两张表的分区裁剪都有效果。

- RIGHT OUTER JOIN

  与LEFT OUTER JOIN类似，如果分区剪裁条件放在ON子句中则只有RIGHT OUTER JOIN的左表生效。如果分区剪裁条件放在WHERE中，则两张表都会生效。

- FULL OUTER JOIN

  分区剪裁条件只有都放在WHERE子句中才会生效，放在ON子句中都不会生效。

注意事项

- 分区剪裁如果失效影响较大，且不容易发现。因此，建议在代码提交时关注分区剪裁失效问题。
- 自定义函数中使用分区剪裁时，需要修改类或者在SQL语句前设置 `set odps.sql.udf.ppr.deterministic = true;` 。详情请参见WHERE子句（where_condition）。

1.6. 分组取出每组数据的前N条

本文将为您介绍如何对数据进行分组，取出每组数据的前N条数据。

示例数据

empno	ename	job	sal
7369	SMITH	CLERK	800.0
7876	SMITH	CLERK	1100.0
7900	JAMES	CLERK	950.0
7934	MILLER	CLERK	1300.0
7499	ALLEN	SALESMAN	1600.0
7654	MARTIN	SALESMAN	1250.0

empno	ename	job	sal
7844	TURNER	SALESMAN	1500.0
7521	WARD	SALESMAN	1250.0

实现方法

您可以通过以下两种方法实现：

- 取出每条数据的行号，再用WHERE语句进行过滤。

```
SELECT * FROM (
  SELECT empno
  , ename
  , sal
  , job
  , ROW_NUMBER() OVER (PARTITION BY job ORDER BY sal) AS rn
  FROM emp
) tmp
WHERE rn < 10;
```

- 使用UDTF实现Split函数。

  详情请参见MaxCompute 学习计划中最后的示例。此方法可以更迅速地判断当前的序号，如果当前序号已经超过指定的条数（例如10条），便不再处理，从而提高计算效率。

1.7. 多行数据合并为一行数据

本文为您介绍，如何使用SQL实现多行数据合并为一行数据。

示例数据

class	gender	name
1	M	LiLei
1	F	HanMM
1	M	Jim
1	F	HanMM
2	F	Kate
2	M	Peter

使用示例

- 示例1：将class相同的names合并为一行，并对names去重。去重操作可通过嵌套子查询实现。

```
SELECT class, wm_concat(distinct ',', name) FROM students GROUP BY class;
```

> **说明** `wm_concat` 是字符拼接函数，详情请参见WM_CONCAT。

输出结果如下。

class	names
1	LiLei,HanMM,Jim
2	Kate,Peter

- 示例2：统计不同class对应的男女人数。

```
SELECT
class
,SUM(CASE WHEN gender = 'M' THEN 1 ELSE 0 END) AS cnt_m
,SUM(CASE WHEN gender = 'F' THEN 1 ELSE 0 END) AS cnt_f
FROM students
GROUP BY class;
```

输出结果如下。

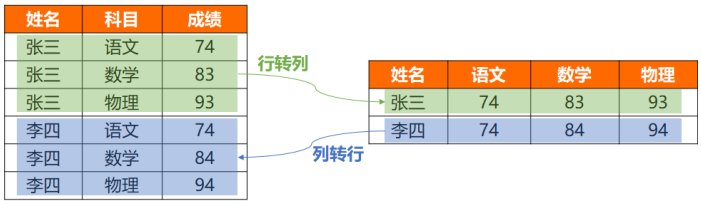
class	cnt_m	cnt_f
1	2	2
2	1	1

1.8. 行转列及列转行最佳实践

本文基于示例为您介绍如何使用SQL实现行转列、列转行需求。

背景信息

行转列与列转行的示意图如下。



- 行转列

  将多行数据转换成一行显示，或将一列数据转换成多列显示。

- 列转行

  将一行数据转换成多行显示，或将多列数据转换成一列显示。

示例数据

为便于理解后续代码示例，本文为您提供源数据，并基于源数据提供相关转换示例。

- 创建用于实现**行转列**的源表并插入数据，命令示例如下。

```
create table rowtocolumn (name string, subject string, result bigint);
insert into table rowtocolumn values
('张三' , '语文' , 74),
('张三' , '数学' , 83),
('张三' , '物理' , 93),
('李四' , '语文' , 74),
('李四' , '数学' , 84),
('李四' , '物理' , 94);
```

- 创建用于实现**列转行**的源表并插入数据，命令示例如下。

```
create table columntorow (name string, chinese bigint, mathematics bigint, physics bigint);
insert into table columntorow values
('张三' , 74, 83, 93),
('李四' , 74, 84, 94);
```

行转列示例

您可以通过如下两种方法实现行转列：

- 方法一：使用 `case when` 表达式，灵活提取各科目（subject）的值作为单独的列，命令示例如下。

```
select name as 姓名,
       max(case subject when '语文' then result end) as 语文,
       max(case subject when '数学' then result end) as 数学,
       max(case subject when '物理' then result end) as 物理
from rowtocolumn
group by name;
```

  返回结果如下。

```
+--------+------------+------------+------------+
| 姓名   | 语文       | 数学       | 物理       |
+--------+------------+------------+------------+
| 张三   | 74         | 83         | 93         |
| 李四   | 74         | 84         | 94         |
+--------+------------+------------+------------+
```

- 方法二：借助MaxCompute提供的内建函数实现，先基于CONCAT和WM_CONCAT函数合并科目和成绩为一列，然后通过KEYVALUE函数解析科目（subject）的值作为单独的列。命令示例如下。

```
select name as 姓名,
       keyvalue(subject, '语文') as 语文,
       keyvalue(subject, '数学') as 数学,
       keyvalue(subject, '物理') as 物理
from(
     select name, wm_concat(';',concat(subject,':',result))as subject
     from rowtocolumn
     group by name);
```

返回结果如下。

```
+--------+------------+------------+------------+
| 姓名   | 语文       | 数学       | 物理       |
+--------+------------+------------+------------+
| 张三   | 74         | 83         | 93         |
| 李四   | 74         | 84         | 94         |
+--------+------------+------------+------------+
```

列转行示例

您可以通过如下两种方法实现列转行：

- 方法一：使用 `union all` ，将各科目（chinese、mathematics、physics）整合为一列，命令示例如下。

```
--解除order by必须带limit的限制，方便列转行SQL命令对结果按照姓名排序。
set odps.sql.validate.orderby.limit=false;
--列转行SQL。
select name as 姓名, subject as 科目, result as 成绩
from(
     select name, '语文' as subject, chinese as result from columntorow
     union all
     select name, '数学' as subject, mathematics as result from columntorow
     union all
     select name, '物理' as subject, physics as result from columntorow)
order by name;
```

返回结果如下。

```
+--------+--------+------------+
| 姓名   | 科目   | 成绩       |
+--------+--------+------------+
| 张三   | 语文   | 74         |
| 张三   | 数学   | 83         |
| 张三   | 物理   | 93         |
| 李四   | 语文   | 74         |
| 李四   | 数学   | 84         |
| 李四   | 物理   | 94         |
+--------+--------+------------+
```

- 方法二：借助MaxCompute提供的内建函数实现，先基于CONCAT函数拼接各科目和成绩，然后基于TRANS_ARRAY和SPLIT_PART函数逐层拆解科目和成绩作为单独的列。命令示例如下。

```
select name as 姓名,
       split_part(subject,':',1) as 科目,
       split_part(subject,':',2) as 成绩
from(
       select trans_array(1,';',name,subject) as (name,subject)
       from(
            select name,
        concat('语文',':',chinese,';','数学',':',mathematics,';','物理',':',physics) as subject
            from columntorow)tt)tx;
```

返回结果如下。

```
+--------+--------+------------+
| 姓名   | 科目   | 成绩       |
+--------+--------+------------+
| 张三   | 语文   | 74         |
| 张三   | 数学   | 83         |
| 张三   | 物理   | 93         |
| 李四   | 语文   | 74         |
| 李四   | 数学   | 84         |
| 李四   | 物理   | 94         |
+--------+--------+------------+
```

1.9. MaxCompute SQL中的关联操作

本文为您介绍MaxCompute SQL中的关联（JOIN）操作。

概述

JOIN类型如下所示。

类型	说明
INNER JOIN	输出符合关联条件的数据。
LEFT JOIN	输出左表的所有记录，以及右表中符合关联条件的数据。右表中不符合关联条件的行，输出NULL。
RIGHT JOIN	输出右表的所有记录，以及左表中符合关联条件的数据。左表中不符合关联条件的行，输出NULL。
FULL JOIN	输出左表和右表的所有记录，对于不符合关联条件的数据，未关联的另一侧输出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，如果右表中不存在符合关联条件的数据，则输出左表。

SQL语句中，同时存在JOIN和WHERE子句时，如下所示。

```
(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}
```

计算顺序如下：

1. 子查询中的WHERE子句（即 `{subquery_where_condition}` ）。
2. JOIN子句中的关联条件（即 `{on_condition}` ）。
3. JOIN结果集中的WHERE子句（即 `{where_condition}` ）。

因此，对于不同的JOIN类型，过滤条件在 `{subquery_where_condition}` 、 `{on_condition}` 和 `{where_condition}` 中时，查询结果可能一致，也可能不一致。详情请参见场景说明。

示例数据

- 表A

  建表语句如下。

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

  示例数据如下。

key	ds
1	20180101
2	20180101
2	20180102

- 表B

  建表语句如下。

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

  示例数据如下。

key	ds
1	20180101
3	20180101
2	20180102

- 表A和表B的笛卡尔乘积如下。

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

场景说明

- INNER JOIN

  INNER JOIN对左右表执行笛卡尔乘积，然后输出满足ON表达式的行。

  结论：过滤条件在 `{subquery_where_condition}` 、 `{on_condition}` 和 `{where_condition}` 中时，查询结果是一致的。

  - 情况1：过滤条件在子查询 `{subquery_where_condition}` 中。

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

    结果如下。

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

  - 情况2：过滤条件在JOIN的关联条件 `{on_condition}` 中。

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

    笛卡尔积结果为9条，满足关联条件的结果只有1条，如下。

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

- 情况3：过滤条件在JOIN结果集的WHERE子句中。

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

笛卡尔积的结果为9条，满足关联条件的结果有3条，如下。

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

对上述结果执行JOIN结果集中的过滤条件 `A.ds='20180101' and B.ds='20180101'` ，结果只有1条，如下。

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

- LEFT JOIN

LEFT JOIN对左右表执行笛卡尔乘积，输出满足ON表达式的行。对于左表中不满足ON表达式的行，输出左表，右表输出NULL。

结论：过滤条件在 `{subquery_where_condition}` 、 `{on_condition}` 和 `{where_condition}` 中时，查询结果不一致。

- 左表的过滤条件在 `{subquery_where_condition}` 和 `{where_condition}` 中时，查询结果是一致的。
- 右表的过滤条件在 `{subquery_where_condition}` 和 `{on_condition}` 中时，查询结果是一致的。

- 情况1：过滤条件在子查询 `{subquery_where_condition}` 中。

```
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;
```

结果如下。

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

- 情况2：过滤条件在JOIN的关联条件 `{on_condition}` 中。

```
SELECT A.*, B.*
FROM A LEFT 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，满足关联条件的结果只有1条。左表输出剩余不满足关联条件的两条记录，右表输出NULL。

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

- 情况3：过滤条件在JOIN结果集的WHERE子句中。

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

笛卡尔积的结果为9条，满足ON条件的结果有3条。

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

对上述结果执行JOIN结果集中的过滤条件 `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

- RIGHT JOIN

  RIGHT JOIN和LEFT JOIN是类似的，只是左右表的区别。

  - 过滤条件在 `{subquery_where_condition}` 、 `{on_condition}` 和 `{where_condition}` 时，查询结果不一致。
  - 右表的过滤条件，在 `{subquery_where_condition}` 和 `{where_condition}` 中时，查询结果一致。
  - 左表的过滤条件，放在 `{subquery_where_condition}` 和 `{on_condition}` 中时，查询结果一致。

- FULL JOIN

  FULL JOIN对左右表执行笛卡尔乘积，然后输出满足关联条件的行。对于左右表中不满足关联条件的行，输出有数据表的行，无数据的表输出NULL。

  结论：过滤条件在 `{subquery_where_condition}` 、 `{on_condition}` 和 `{where_condition}` 时，查询结果不一致。

  - 情况1：过滤条件在子查询 `{subquery_where_condition}` 中。

```
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;
```

结果如下。

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

  - 情况2：过滤条件在JOIN的关联条件 `{on_condition}` 中。

```
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，满足关联条件的结果只有1条。对于左表不满足关联条件的两条记录输出左表数据，右表输出NULL。对于右表不满足关联条件的两条记录输出右表数据，左表输出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

- 情况3：过滤条件在JOIN结果集的WHERE子句中。

```
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，满足关联条件的结果有3条。

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

对于不满足关联条件的表输出数据，另一表输出NULL。

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

对上述结果执行JOIN结果集中的过滤条件 `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

- LEFT SEMI JOIN

LEFT SEMI JOIN将左表的每一条记录，和右表进行匹配。如果匹配成功，则输出左表。如果匹配不成功，则跳过。由于只输出左表，所以JOIN后的WHERE条件中不涉及右表。

结论：过滤条件在 `{subquery_where_condition}` 、 `{on_condition}` 和 `{where_condition}` 中时，查询结果是一致的。

- 情况1：过滤条件在子查询{subquery_where_condition}中。

```
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;
```

结果如下。

a.key	a.ds
1	20180101

- 情况2：过滤条件在JOIN的关联条件 `{on_condition}` 中。

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

结果如下。

a.key	a.ds
1	20180101

- 情况3：过滤条件在JOIN结果集的WHERE子句中。

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

符合关联条件的结果如下。

a.key	a.ds
1	20180101

对上述结果执行JOIN结果集中的过滤条件 `A.ds='20180101'` ，结果如下。

a.key	a.ds
1	20180101

- LEFT ANTI JOIN

LEFT ANTI JOIN将左表的每一条记录，和右表进行匹配。如果右表中的记录不匹配，则输出左表。由于只输出左表，所以JOIN后的WHERE条件中不能涉及右表。LEFT ANTI JOIN常常用来实现NOT EXISTS语义。

结论：过滤条件在 `{subquery_where_condition}` 、 `{on_condition}` 和 `{where_condition}` 中时，查询结果不一致。

- 左表的过滤条件在 `{subquery_where_condition}` 和 `{where_condition}` 中时，查询结果是一致的。
- 右表的过滤条件在 `{subquery_where_condition}` 和 `{on_condition}` 中时，查询结果是一致的。
- 情况1：过滤条件在子查询 `{subquery_where_condition}` 中。

```
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;
```

结果如下。

a.key	a.ds
2	20180101

- 情况2：过滤条件在JOIN的关联条件 `{on_condition}` 中。

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

结果如下。

a.key	a.ds
2	20180101
2	20180102

- 情况3：过滤条件在JOIN结果集的WHERE子句中。

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

左表中符合关联条件的数据如下。

a.key	a.ds
2	20180101
2	20180102

对上述结果执行JOIN结果集中的过滤条件 `A.ds='20180101'` ，结果如下。

a.key	a.ds
2	20180101

注意事项

- INNER JOIN/LEFT SEMI JOIN左右表的过滤条件不受限制。
- LEFT JOIN/LEFT ANTI JOIN左表的过滤条件需放在 `{subquery_where_condition}` 或 `{where_condition}` 中，右表的过滤条件需放在 `{subquery_where_condition}` 或 `{on_condition}` 中。
- RIGHT JOIN和LEFT JOIN相反，右表的过滤条件需放在 `{subquery_where_condition}` 或 `{where_condition}` 中，左表的过滤条件需放在 `{subquery_where_condition}` 或 `{on_condition}` 。
- FULL OUTER JOIN的过滤条件只能放在 `{subquery_where_condition}` 中。

2.数据迁移

本文为您介绍数据迁移的最佳实践，包含将其他业务平台的业务数据或日志数据迁移至MaxCompute，或将MaxCompute的数据迁移至其它业务平台。

背景信息

传统关系型数据库不适合处理海量数据，如果您的数据存放在传统的关系型数据库且数据量庞大时，可以将数据迁移至MaxCompute。

MaxCompute为您提供了完善的数据迁移方案以及多种经典的分布式计算模型，能够快速地解决海量数据存储和计算问题，有效降低企业成本。

DataWorks为MaxCompute提供一站式的数据集成、数据开发、数据管理和数据运维等功能。其中：数据集成为您提供稳定高效和弹性伸缩的数据同步平台。

最佳实践合集

- 迁移其它业务平台的业务数据至MaxCompute：
  - 跨项目迁移MaxCompute数据，详情请参见跨项目迁移MaxCompute数据。
  - 迁移Hadoop数据至MaxCompute，详情请参见迁移Hadoop数据至MaxCompute、迁移Hadoop数据至MaxCompute最佳实践（视频）。数据迁移和脚本迁移遇到的问题及解决方案请参见迁移自建Hadoop数据至MaxCompute实践。
  - 迁移Oracle数据至MaxCompute，详情请参见迁移Oracle数据至MaxCompute。
  - 迁移消息队列for Apache Kafka集群数据至MaxCompute，详情请参见迁移消息队列for Apache Kafka数据至MaxCompute。
  - 迁移Kafka集群数据至MaxCompute，详情请参见迁移Kafka数据至MaxCompute。
  - 迁移Elasticsearch集群数据至MaxCompute，详情请参见迁移Elasticsearch数据至MaxCompute。
  - 迁移RDS数据至MaxCompute，详情请参见迁移RDS数据至MaxCompute。
  - 迁移OSS的JSON数据至MaxCompute，详情请参见迁移OSS的JSON数据至MaxCompute。
  - 迁移MongoDB的JSON数据至MaxCompute，详情请参见迁移MongoDB的JSON数据至MaxCompute。
  - 迁移ECS上自建的MySQL数据库中的数据至MaxCompute，详情请参见迁移ECS自建MySQL数据库至MaxCompute。
- 迁移其它业务平台的日志数据至MaxCompute：
  - 通过Tunnel迁移日志数据至MaxCompute，详情请参见通过Tunnel迁移日志数据至MaxCompute。
  - 通过DataHub迁移日志数据至MaxCompute，详情请参见通过DataHub迁移日志数据至MaxCompute。
  - 通过DataWorks迁移日志数据至MaxCompute，详情请参见通过DataWorks迁移日志数据至MaxCompute。
  - 通过LogHub迁移日志数据至MaxCompute，详情请参见通过LogHub迁移日志数据至MaxCompute。
- 迁移MaxCompute数据至其它业务平台：
  - 迁移MaxCompute数据至OSS，详情请参见迁移MaxCompute数据至OSS。
  - 迁移MaxCompute数据至OTS，详情请参见迁移MaxCompute数据至OTS。

MaxCompute处理业务数据和日志数据后，可以通过Quick BI快速地以可视化方式展现数据处理结果，详情请参见基于MaxCompute的大数据BI分析。

2.1. 数据迁移

本文为您介绍数据迁移的最佳实践，包含将其他业务平台的业务数据或日志数据迁移至MaxCompute，或将MaxCompute的数据迁移至其它业务平台。

背景信息

传统关系型数据库不适合处理海量数据，如果您的数据存放在传统的关系型数据库且数据量庞大时，可以将数据迁移至MaxCompute。

MaxCompute为您提供了完善的数据迁移方案以及多种经典的分布式计算模型，能够快速地解决海量数据存储和计算问题，有效降低企业成本。

DataWorks为MaxCompute提供一站式的数据集成、数据开发、数据管理和数据运维等功能。其中：数据集成为您提供稳定高效和弹性伸缩的数据同步平台。

最佳实践合集

- 迁移其它业务平台的业务数据至MaxCompute：
  - 跨项目迁移MaxCompute数据，详情请参见跨项目迁移MaxCompute数据。
  - 迁移Hadoop数据至MaxCompute，详情请参见迁移Hadoop数据至MaxCompute、迁移Hadoop数据至MaxCompute最佳实践（视频）。数据迁移和脚本迁移遇到的问题及解决方案请参见迁移自建Hadoop数据至MaxCompute实践。
  - 迁移Oracle数据至MaxCompute，详情请参见迁移Oracle数据至MaxCompute。
  - 迁移消息队列for Apache Kafka集群数据至MaxCompute，详情请参见迁移消息队列for Apache Kafka数据至MaxCompute。
  - 迁移Kafka集群数据至MaxCompute，详情请参见迁移Kafka数据至MaxCompute。
  - 迁移Elasticsearch集群数据至MaxCompute，详情请参见迁移Elasticsearch数据至MaxCompute。
  - 迁移RDS数据至MaxCompute，详情请参见迁移RDS数据至MaxCompute。
  - 迁移OSS的JSON数据至MaxCompute，详情请参见迁移OSS的JSON数据至MaxCompute。
  - 迁移MongoDB的JSON数据至MaxCompute，详情请参见迁移MongoDB的JSON数据至MaxCompute。

- 迁移ECS上自建的MySQL数据库中的数据至MaxCompute，详情请参见迁移ECS自建MySQL数据库至MaxCompute。

- 迁移其它业务平台的日志数据至MaxCompute：
  - 通过Tunnel迁移日志数据至MaxCompute，详情请参见通过Tunnel迁移日志数据至MaxCompute。
  - 通过DataHub迁移日志数据至MaxCompute，详情请参见通过DataHub迁移日志数据至MaxCompute。
  - 通过DataWorks迁移日志数据至MaxCompute，详情请参见通过DataWorks迁移日志数据至MaxCompute。
  - 通过LogHub迁移日志数据至MaxCompute，详情请参见通过LogHub迁移日志数据至MaxCompute。
- 迁移MaxCompute数据至其它业务平台：
  - 迁移MaxCompute数据至OSS，详情请参见迁移MaxCompute数据至OSS。
  - 迁移MaxCompute数据至OTS，详情请参见迁移MaxCompute数据至OTS。

MaxCompute处理业务数据和日志数据后，可以通过Quick BI快速地以可视化方式展现数据处理结果，详情请参见基于MaxCompute的大数据BI分析。

2.2. MaxCompute跨项目迁移

本文为您介绍如何配置相同区域下不同的MaxCompute项目，以及如何实现数据迁移。

前提条件

请您首先完成教程《搭建互联网在线运营分析平台》的全部步骤，详情请参见业务场景与开发流程。

背景信息

本文使用的被迁移的原始项目为教程《搭建互联网在线运营分析平台》中的bigdata_DOC项目，您需要再创建一个迁移目标项目，用于存放原始项目的表、资源、配置和数据。

操作步骤

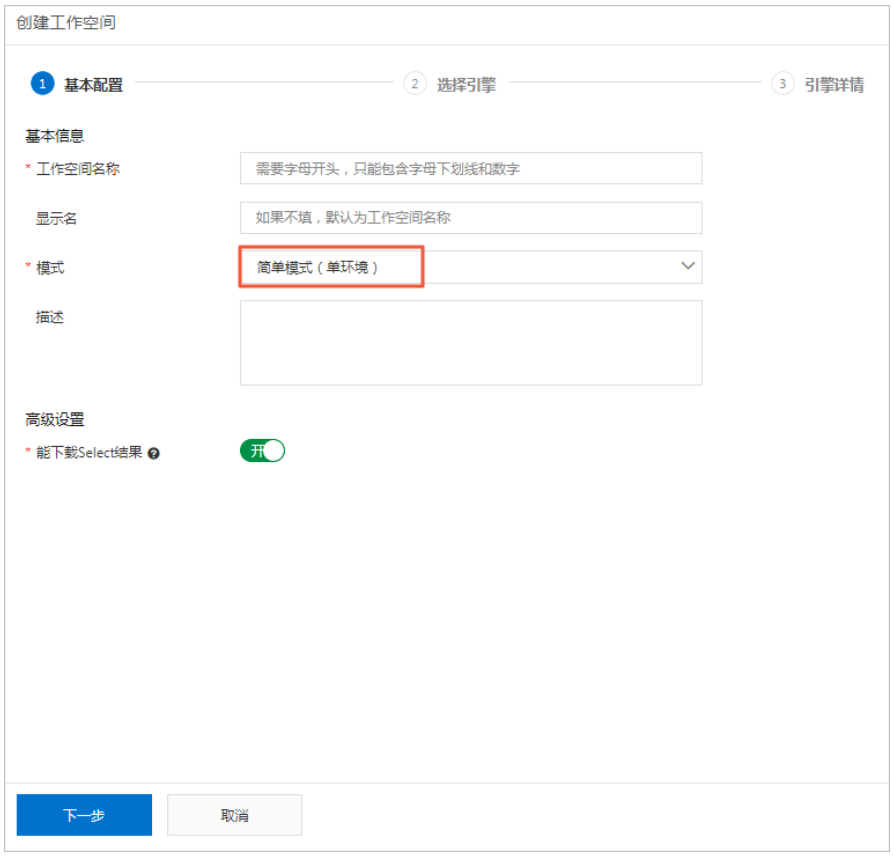
1. 创建迁移目标项目

   本文的MaxCompute项目即DataWorks的工作空间。

   i. 登录DataWorks控制台，单击左侧导航栏中的**工作空间列表**。

   ii. 选择区域为**华东1（杭州）**，单击**创建工作空间**。

iii. 填写**创建工作空间**对话框中的**基本配置**，单击**下一步**。



分类	参数	描述
基本信息	**工作空间名称**	工作空间名称的长度需要在3~23个字符，以字母开头，且只能包含字母、下划线（_）和数字。
	显示名	显示名不能超过23个字符，只能字母、中文开头，仅包含中文、字母、下划线（_）和数字。
	模式	工作空间模式是DataWorks新版推出的新功能，分为**简单模式**和**标准模式**： ■ **简单模式**：指一个DataWorks工作空间对应一个MaxCompute项目，无法设置开发和生产环境，只能进行简单的数据开发，无法对数据开发流程以及表权限进行强控制。 ■ **标准模式**：指一个DataWorks工作空间对应两个MaxCompute项目，可以设置开发和生产两种环境，提升代码开发规范，并能够对表权限进行严格控制，禁止随意操作生产环境的表，保证生产表的数据安全。 详情请参见简单模式和标准模式的区别。
	描述	对创建的工作空间进行简单描述。
高级设置	**能下载select结果**	控制数据开发中查询的数据结果是否能够下载，如果关闭无法下载select的数据查询结果。此参数在工作空间创建完成后可以在工作空间配置页面进行修改，详情可参考文档：安全设置。

由于原始项目bigdata_DOC为简单模式，为方便起见，本文中DataWorks工作空间模式也为**简单模式（单环境）**。

工作空间名称全局唯一，建议您使用易于区分的名称，本例中使用的名称为clone_test_doc。

iv. 选择**计算引擎服务**为MaxCompute、按量付费，单击**下一步**。

v. 配置**引擎详情**，单击**创建工作空间**。

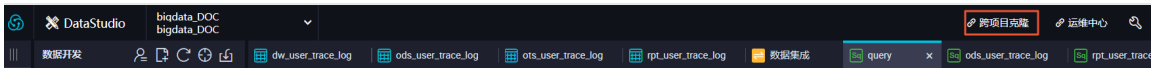
分类	参数	描述
MaxCompute	**实例显示名称**	实例显示名称不能超过27个字符，仅支持字母、中文开头，仅包含中文、字母、下划线和数字。
	MaxCompute项目名称	默认与DataWorks工作空间的名称一致。
	MaxCompute访问身份	开发环境的MaxCompute访问身份默认为**任务负责人**，不可以修改。 生产环境的MaxCompute访问身份包括**阿里云主账号**和**阿里云子账号**。
	Quota组切换	Quota用来实现计算资源和磁盘配额。

2. 跨项目克隆

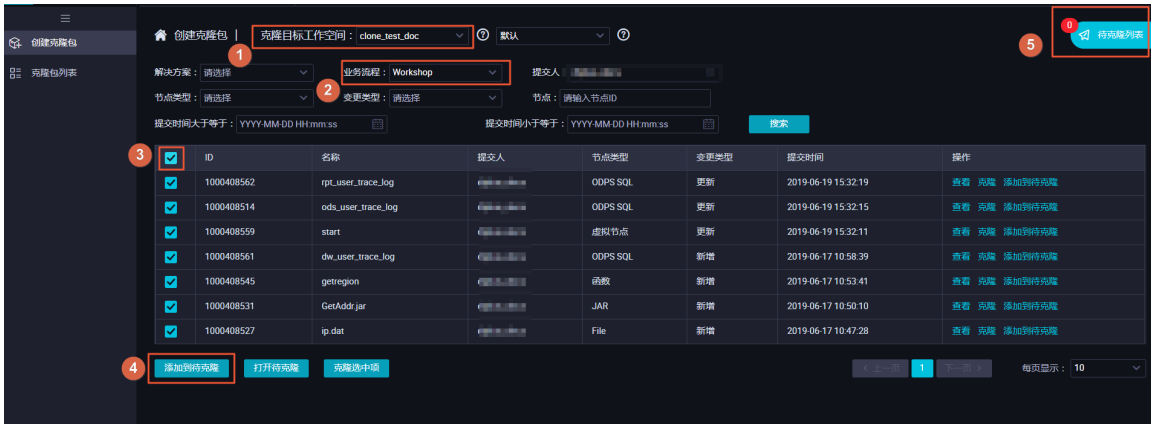
您可以通过**跨项目克隆**功能将原始项目bigdata_DOC的节点配置和资源复制到当前项目，详情请参见跨项目克隆实践。

> 说明
> - 跨项目克隆无法复制表结构与数据。
> - 跨项目克隆无法复制组合节点，需要您手动创建。

i. 单击原始项目bigdata_DOC右上角的**跨项目克隆**，跳转至相应的克隆页面。



ii. 选择**克隆目标工作空间**为clone_test_doc，**业务流程**为您需要克隆的业务流程Workshop，勾选所有节点，单击**添加到待克隆**后单击右侧的**待克隆列表**。



iii. 单击**全部克隆**，将选中的节点克隆至工作空间clone_test_DOC。

iv. 切换至您新建的项目，检查节点是否已完成克隆。

3. 新建数据表

跨项目克隆功能无法克隆您的表结构，因此您需要手动新建表。

- 对于非分区表，建议使用如下语句迁移表结构。

```
create table table_name as select * from 源库MaxCompute项目.表名 ;
```

- 对于分区表，建议使用如下语句迁移表结构。

```
create table table_name partitioned by (分区列 string);
```

新建表后请将表提交到生产环境。更多建表信息，请参见新建数据表。

4. 数据同步

跨项目克隆功能无法复制原始项目的数据到新项目，因此您需要手动同步数据，本文中仅同步表rpt_user_trace_log的数据。

i. 新建数据源。

a. 在**数据集成**页面，单击左侧导航栏上的**数据源**。

b. 在**数据源管理**页面，单击右上角**新增数据源**，并选择**MaxCompute(ODPS)**。

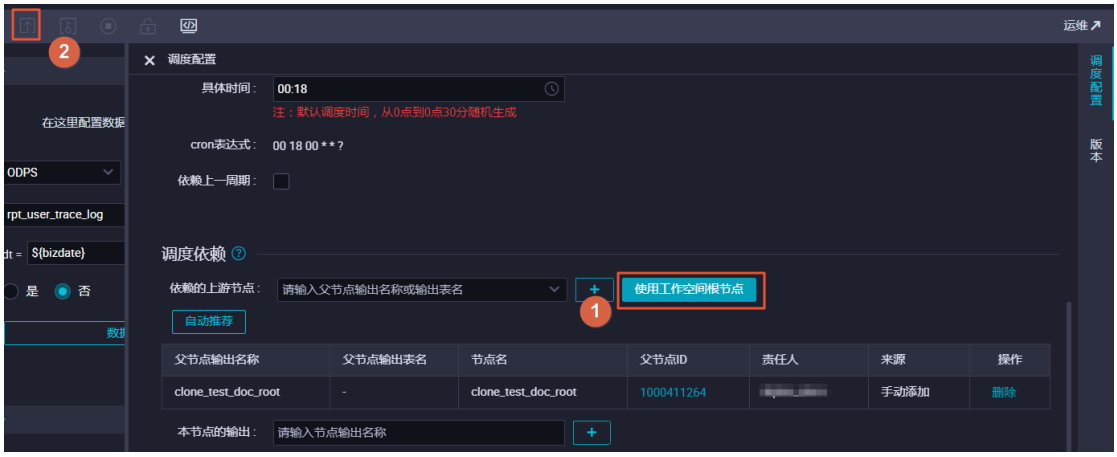
c. 填写您的**数据源名称**、**ODPS项目名称**、**AccessKey ID**、**AccessKey Secret**等信息，单击**完成**，详情请参见配置MaxCompute数据源。

ii. 创建数据同步任务。

a. 在**数据开发**页面右键单击您克隆的业务流程**Workshop**下的**数据集成**，选择**新建 > 离线同步**。

b. 编辑您新建的数据同步任务节点，填写参数如下图所示。其中数据源bigdata_DOC是您的原始项目，数据源**odps_first**代表您当前的新建项目，表名是您需要同步数据的表rpt_user_trace_log。完成后单击**调度配置**。

c. 单击**使用工作空间根节点**后，提交数据同步任务。



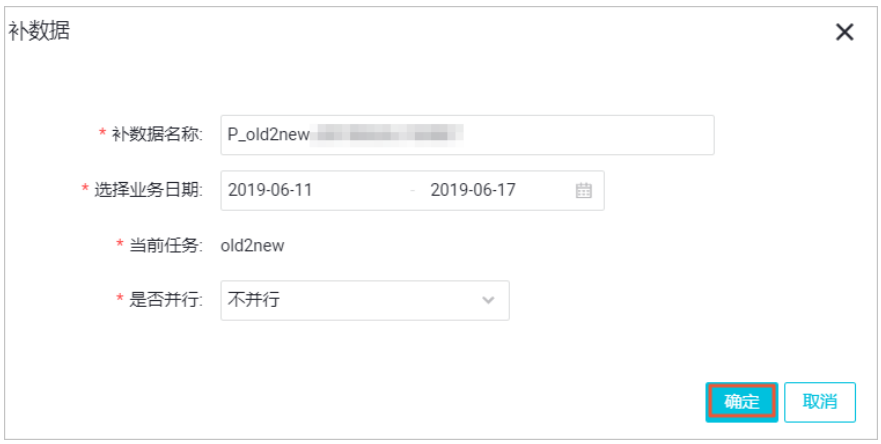
iii. 补数据

a. 单击左上角的图标，选择**全部产品 > 运维中心**。

b. 单击左侧导航栏中的**周期任务运维 > 周期任务**。

c. 右键单击您的数据同步任务，选择**补数据 > 当前节点**。

d. 本例中，需要补数据的日期分区为2019年6月11日到17日，您可以直接选择业务日期，进行多个分区的数据同步。完成设置后，单击**确定**。



e. 在**周期任务运维 > 补数据实例**页面，您可以查看补数据实例任务运行状态，显示**运行成功**则说明完成数据同步。

iv. 验证结果

您可以在**业务流程 > 数据开发**中新建**ODPS SQL**类型节点，执行如下语句查看数据是否完成同步。

```
select * from rpt_user_trace_log where dt BETWEEN '20190611' and '20190617';
```

2.3. Hadoop数据迁移MaxCompute最佳实践

本文为您介绍如何通过DataWorks数据同步功能，迁移HDFS数据至MaxCompute，或从MaxCompute迁移数据至HDFS。无论您使用Hadoop还是Spark，均可以与MaxCompute进行双向同步。

前提条件

- 开通MaxCompute并创建项目。

  本文以在华东1（杭州）地域创建项目bigdata_DOC为例。详情请参见开通MaxCompute和DataWorks。

- 搭建Hadoop集群。

  进行数据迁移前，您需要保证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，详情请参见创建集群。

  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，地域为华东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）。

步骤一：数据准备

1. Hadoop集群创建测试数据。
   i. 通过阿里云账号登录阿里云E-MapReduce控制台。
   ii. 在EMR控制台界面，选择目标项目并新建作业doc。本例中Hive建表语句如下。 关于EMR上新建作业更多信息请参见作业编辑。

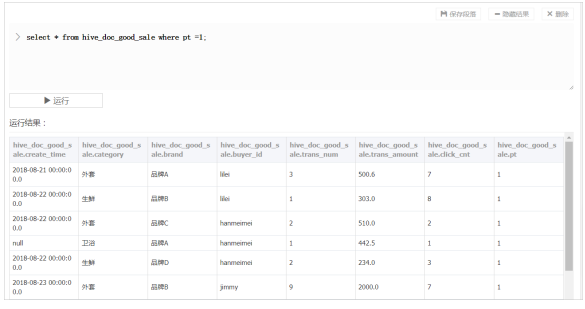
```
CREATE TABLE IF NOT
EXISTS hive_doc_good_sale(
   create_time timestamp,
   category STRING,
   brand STRING,
   buyer_id STRING,
   trans_num BIGINT,
   trans_amount DOUBLE,
   click_cnt BIGINT
   )
   PARTITIONED BY (pt string) ROW FORMAT
DELIMITED FIELDS TERMINATED BY ',' lines terminated by '\n';
```

   iii. 单击**运行**，出现 `Query executed successfully` 提示，则说明成功在EMR Hadoop集群上创建了表hive_doc_good_sale。

   iv. 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) ;
```

   v. 完成插入数据后，您可以执行 `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以用于迁移。



2. 利用DataWorks新建目标表。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 在**工作空间列表**页面，单击相应工作空间后的**数据开发**。
   iv. 在**数据开发**页面，右键单击目标工作流程，选择**新建 > MaxCompute > 表**。
   v. 在弹出的**新建表**对话框中，填写**表名**，并单击**提交**。

   > 说明
   > - 表名必须以字母开头，不能包含中文或特殊字符。
   > - 如果绑定多个实例，则需要选择MaxCompute引擎实例。

   vi. 在表的编辑页面，选择**DDL模式**。
   vii. 在**DDL模式**对话框中输入建表语句，单击**生成表结构**，并确认操作。本示例的建表语句如下所示。

```
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Hive数据类型与MaxCompute数据类型的映射，当前数据映射关系请参见数据类型映射表。

   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见安装并配置MaxCompute客户端。

```
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;
   > ```

   viii. 单击**提交到生产环境**，完成表的创建。
   ix. 完成建表后，单击左侧导航栏中的**表管理**，即可查看当前创建的MaxCompute表。

步骤二：数据同步

1. 新建自定义资源组。

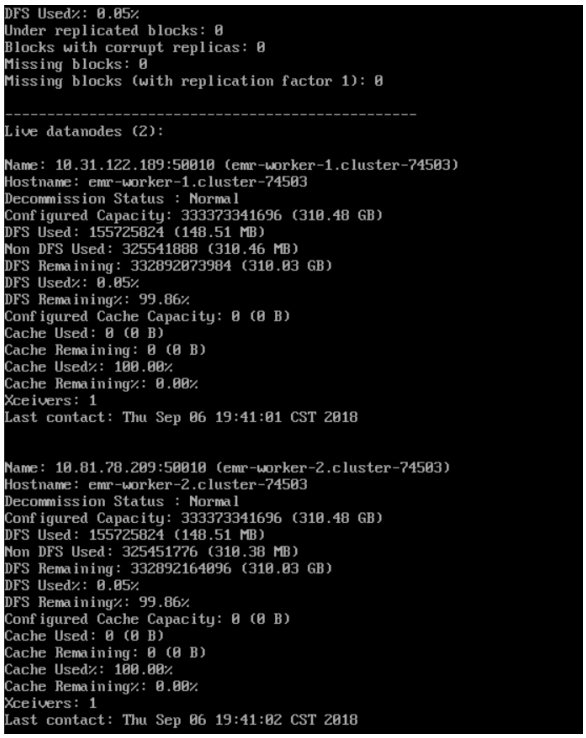
   由于MaxCompute项目所处的网络环境与Hadoop集群中的数据节点（data node）网络通常不可达，您可以通过自定义资源组的方式，将DataWorks的同步任务运行在Hadoop集群的Master节点上（Hadoop集群内Master节点和数据节点通常可达）。

   i. 查看Hadoop集群数据节点 。

      a. 登录EMR控制台，单击**集群管理**。

      b. 选择集群名称，并在左侧导航栏上单击**主机列表**。

      您也可以通过单击上图中Master节点的ECS ID，进入ECS实例详情页。然后单击远程连接进入ECS，执行 `hadoop dfsadmin -report` 命令查看data node。



```
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
```

> **说明** 本示例的data node只具有内网地址，很难与DataWorks默认资源组互通，所以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。

ii. 新建任务资源组。

a. 在DataWorks控制台上，进入**数据集成 > 自定义资源组管理**页面，单击右上角的**新增自定义资源组**。

> **说明** 目前仅专业版及以上版本方可使用此入口。

b. 添加服务器时，需要输入ECS UUID和机器IP等信息（对于经典网络类型，需要输入服务器名称。对于专有网络类型，需要输入服务器UUID）。目前仅DataWorks V2.0华东2（上海）支持添加经典网络类型的调度资源，对于其他地域，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需要填写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）。

c. 添加服务器后，需要保证master node与DataWorks网络可达。如果您使用的是ECS服务器，需要设置服务器安全组。

- 如果您使用的内网IP互通，请参见ECS自建数据库的安全组配置。
- 如果您使用的是公网IP，可以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）。

d. 完成上述步骤后，按照提示安装自定义资源组agent。当前状态显示为**可用**时，则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，执行`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。

2. 新建数据源。

DataWorks新建工作空间后，默认数据源odps_first。因此只需要添加Hadoop集群数据源。更多详情请参见配置HDFS数据源。

i. 进入**数据集成**页面，单击左侧导航栏中的**数据源**。

ii. 在**数据源管理**页面，单击右上角的**新增数据源** 。

iii. 在**新增数据源**页面中，选择数据源类型为**HDFS**。

iv. 填写HDFS数据源的各配置项。

配置	说明
数据源名称	数据源名称必须以字母、数字、下划线组合，且不能以数字和下划线开头。
数据源描述	对数据源进行简单描述，不得超过80个字符。
适用环境	可以选择**开发**或**生产**环境。 **说明** 仅标准模式工作空间会显示此配置。
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` 。 本实验中的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。

v. 完成配置后，单击**测试连通性**。

vi. 测试连通性通过后，单击**完成**。

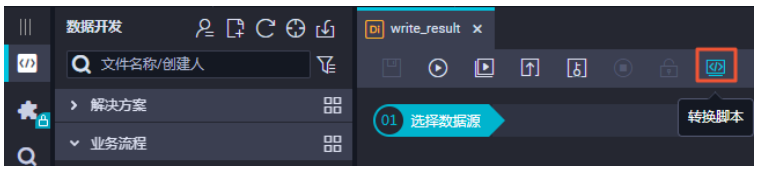
> **说明** 如果EMR Hadoop集群设置网络类型为专有网络，则不支持连通性测试。

3. 配置数据同步任务 。

i. 在**数据开发**页面的左侧菜单栏顶部，单击图标，选择**数据集成 > 离线同步**。

ii. 在**新建节点**对话框中，输入**节点名称**和**目标文件夹**，单击**提交**。

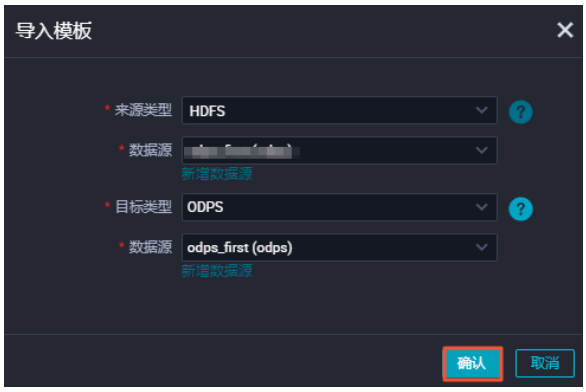
iii. 成功创建数据同步节点后，单击工具栏中的**转换脚本**按钮。



iv. 单击**提示**对话框中的**确认**，即可进入脚本模式进行开发。

v. 单击工具栏中的**导入模板**按钮。

vi. 在**导入模板**对话框中，选择**来源类型**、**数据源**、**目标类型**及**数据源**，单击**确认**。



vii. 新建同步任务完成后，通过导入模板已生成了基本的读取端配置。

此时您可以继续手动配置数据同步任务的读取端数据源，以及需要同步的表信息等。本示例的代码如下所示，更多详情请参见HDFS Reader。

```
{
  "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://47.100.XX.XXX: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",
        }
      }
    },
    "type": "job",
    "version": "1.0"
}
```

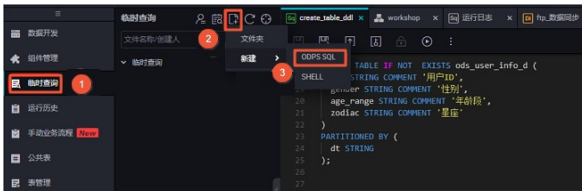
其中，path参数为数据在Hadoop集群中存放的位置。您可以在登录Master Node后，执行 `hdfs dfs -ls /user/hive/warehouse/hive_doc_good_sale` 命令确认。对于分区表，您可以不指定分区，DataWorks数据同步会自动递归到分区路径。

```
[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
```

viii. 完成配置后，单击**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，可以通过日志进行排查。

步骤三：查看结果

1. 在DataStudio页面的左侧导航栏，单击**临时查询**。
2. 选择**新建 > ODPS SQL**。



3. 选择**新建 > ODPS SQL**。
4. 编写并执行SQL语句，查看导入hive_doc_good_sale的数据。

   SQL语句如下所示：

```
--查看是否成功写入MaxCompute。
select * from hive_doc_good_sale where pt=1;
```

> **说明** 您也可以在odpscmd命令行工具中输入 `select * FROM hive_doc_good_sale where pt =1;` ，查询表结果。

如果您想实现MaxCompute数据迁移至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": {
```

```
    "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://47.100.XX.XX:9000",
      "writeMode": "append",
      "fieldDelimiter": ",",
      "encoding": "UTF-8",
      "fileType": "text"
      }
    },
    "setting": {
      "errorLimit": {
        "record": "1000"
    },
    "speed": {
      "throttle": false,
      "concurrent": 1,
      "mbps": "1",
    }
    }
  },
  "type": "job",
  "version": "1.0"
  }
```

说明 您需要参见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，手动复制同步过去的文件。

2.4. Oracle数据迁移MaxCompute最佳实践

本文将为您介绍如何通过DataWorks数据集成，迁移Oracle上的数据至MaxCompute。

前提条件

- 准备DataWorks环境
  i. 开通MaxCompute和DataWorks。
  ii. 开通DataWorks。
  iii. 在DataWorks上完成创建业务流程，本例使用DataWorks简单模式。详情请参见创建业务流程。
- 准备Oracle环境

本文中的Oracle安装在云服务器ECS上，ECS具体配置如下。为了让网络互通，您需要给ECS配置公网IP，并且配置ECS的安全组规则放通Oracle数据库的常用端口1521。关于ECS安全组配置详情请参见修改安全组规则。

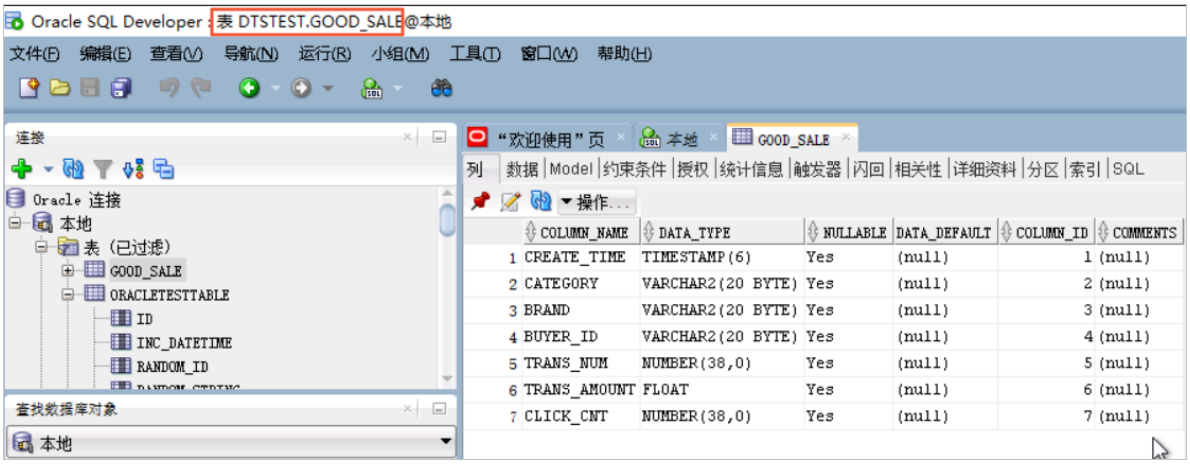
如上图所示，本文中的ECS规格为**ecs.c5.xlarge**，使用专有网络，区域为华东1（杭州）。

背景信息

本文需要使用DataWorks Oracle Reader读取Oracle中的测试数据，详情请参见Oracle Reader。

准备Oracle测试数据

1. 进入Oracle图形化操作界面，新建表DTSTEST.GOOD_SALE，主要包括create_time、category、brand、buyer_id、trans_num、trans_amount、click_cnt这7列。



2. 插入测试数据，本文中手动插入数据如下。

```
insert into good_sale values('28-12月-19','厨具','品牌A','hanmeimei','6','80.6','4');
insert into good_sale values('21-12月-19','生鲜','品牌B','lilei','7','440.6','5');
insert into good_sale values('29-12月-19','衣服','品牌C','lily','12','351.9','9');
commit;
```

3. 完成数据插入之后，执行如下语句查看表数据。

```
select * from good_sale;
```

通过DataWorks将数据从Oracle迁移至MaxCompute

1. 登录DataWorks控制台。
2. 在DataWorks上创建目标表。用以接收从Oracle迁移的数据。
   i. 右键单击已创建的**业务流程**，选择**新建 > MaxCompute > 表**。
   ii. 在**新建表**页面，选择引擎类型并输入**表名**。
   iii. 在表的编辑页面，单击**DDL模式**。
   iv. 在**DDL模式**对话框，输入建表语句，单击**生成表结构**。

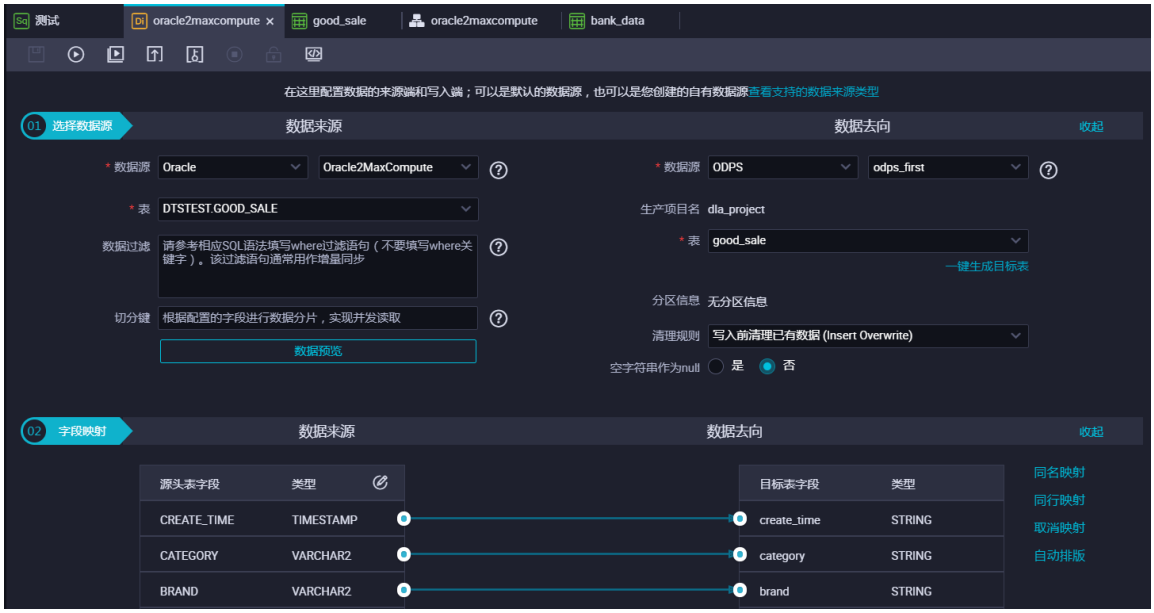
```
CREATE TABLE good_sale
(
    create_time     string,
    category        string,
    brand           string,
    buyer_id        string,
    trans_num       bigint,
    trans_amount    double,
    click_cnt       bigint
) ;
```

   在建表过程中，需要考虑Oracle数据类型与MaxCompute数据类型的映射，Oracle Reader支持的数据类型请参见类型转换列表。
   v. 单击**提交到生产环境**。
3. 新建Oracle数据源，详情请参见配置Oracle数据源。
4. 创建离线同步节点。

i.

ii.

iii. 成功创建数据同步节点后，选择**数据源**为您刚刚添加的Oracle数据源，**表**为您刚刚创建的测试表格，选择**同名映射**。其他参数保持默认配置。



iv. 单击图标运行代码。

v. 您可以在**运行日志**查看运行结果。

验证结果

1. 右键单击业务流程，选择**新建 > MaxCompute > ODPS SQL**。
2. 在**新建节点**对话框中输入**节点名称**，并单击**提交**。
3. 在ODPS SQL节点编辑页面输入如下语句。

```
--查看是否成功写入MaxCompute。
select * from good_sale;
```

4. 单击图标运行代码。
5. 您可以在**运行日志**查看运行结果。

2.5. Kafka数据迁移MaxCompute最佳实践

本文为您介绍如何使用DataWorks数据集成，将Kafka集群上的数据迁移至MaxCompute。

前提条件

- 开通MaxCompute和DataWorks。
- 在DataWorks上完成创建业务流程，本例使用DataWorks简单模式。详情请参见创建业务流程。
- 搭建Kafka集群

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

  本文使用的EMR Kafka版本信息如下：

  - EMR版本：EMR-3.12.1
  - 集群类型：Kafka
  - 软件信息：Ganglia 3.7.2，ZooKeeper 3.4.12，Kafka 2.11-1.0.1，Kafka-Manager 1.3.3.16

  Kafka集群使用专有网络，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。

背景信息

Kafka是一款分布式发布与订阅的消息中间件，具有高性能、高吞量的特点被广泛使用，每秒能处理上百万的消息。Kafka适用于流式数据处理，主要应用于用户行为跟踪、日志收集等场景。

一个典型的Kafka集群包含若干个生产者（Producer）、Broker、消费者（Consumer）以及一个Zookeeper集群。Kafka集群通过Zookeeper管理自身集群的配置并进行服务协同。

Topic是Kafka集群上最常用的消息的集合，是一个消息存储逻辑概念。物理磁盘不存储Topic，而是将Topic中具体的消息按分区（Partition）存储在集群中各个节点的磁盘上。每个Topic可以有多个生产者向它发送消息，也可以有多个消费者向它拉取（消费）消息。

每个消息被添加到分区时，会分配一个Offset（偏移量，从0开始编号），是消息在一个分区中的唯一编号。

步骤一：准备Kafka数据

您需要在Kafka集群创建测试数据。为保证您可以顺利登录EMR集群Header主机，以及保证MaxCompute和DataWorks可以顺利和EMR集群Header主机通信，请您首先配置EMR集群Header主机安全组，放行TCP 22及TCP 9092端口。

1. 登录EMR集群Header主机地址。
   i. 进入EMR Hadoop控制台。
   ii. 在顶部导航栏，单击**集群管理**。
   iii. 在显示的页面，找到您需要创建测试数据的集群，进入集群详情页。
   iv. 在集群详情页面，单击主机列表，确认EMR集群Header主机地址，并通过SSH连接远程登录。
2. 创建测试Topic。

   执行如下命令创建测试所使用的Topic testkafka。

   ```
   kafka-topics.sh --zookeeper emr-header-1:2181/kafka-1.0.1 --partitions 10 --replication-factor 3 --topic testkafka  --create
   ```
3. 写入测试数据。

   执行如下命令，可以模拟生产者向Topic testkafka中写入数据。由于Kafka用于处理流式数据，您可以持续不断的向其中写入数据。为保证测试结果，建议写入10条以上的数据。

   ```
   kafka-console-producer.sh --broker-list emr-header-1:9092 --topic testkafka
   ```

   您可以同时再打开一个SSH窗口，执行如下命令，模拟消费者验证数据是否已成功写入Kafka。当数据写入成功时，您可以看到已写入的数据。

   ```
   kafka-console-consumer.sh --bootstrap-server emr-header-1:9092 --topic testkafka --from-beginning
   ```

步骤二：在DataWorks上创建目标表

在DataWorks上创建目标表用以接收Kafka数据。

1. 进入**数据开发**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 单击相应工作空间后的**数据开发**。
2. 右键单击**业务流程**，选择**新建 > MaxCompute > 表**。
3. 在弹出的**新建表**对话框中，填写**表名**，并单击**提交**。

   > 说明
   > - 表名必须以字母开头，不能包含中文或特殊字符。
   > - 如果绑定多个实例，则需要选择MaxCompute引擎实例。
4. 在表的编辑页面，选择**DDL模式**。
5. 在**DDL模式**对话框中，输入如下建表语句，单击**生成表结构**。

   ```
   CREATE TABLE testkafka
   (
    key             string,
    value           string,
    partition1      string,
    timestamp1      string,
    offset          string,
    t123            string,
    event_id        string,
    tag             string
   ) ;
   ```

   其中的每一列，对应于DataWorks数据集成Kafka Reader的默认列：
   - __key__表示消息的key。
   - __value__表示消息的完整内容 。
   - __partition__表示当前消息所在分区。
   - __headers__表示当前消息headers信息。
   - __offset__表示当前消息的偏移量。
   - __timestamp__表示当前消息的时间戳。

您还可以自主命名，详情参见Kafka Reader。

6. 单击**提交到生产环境**并**确认**。

步骤三：同步数据

1. 新建独享数据集成资源组。

   由于当前DataWorks的默认资源组无法完美支持Kafka插件，您需要使用独享数据集成资源组完成数据同步。详情请参见新增和使用独享数据集成资源组。

2. 新建数据集成节点。

   i.

   ii.

3.

4.

5. 配置脚本，示例代码如下。

```
{
    "type": "job",
    "steps": [
        {
            "stepType": "kafka",
            "parameter": {
                "server": "47.xxx.xxx.xxx:9092",
                "kafkaConfig": {
                    "group.id": "console-consumer-83505"
                },
                "valueType": "ByteArray",
                "column": [
                    "__key__",
                    "__value__",
                    "__partition__",
                    "__timestamp__",
                    "__offset__",
                    "'123'",
                    "event_id",
                    "tag.desc"
                ],
                "topic": "testkafka",
                "keyType": "ByteArray",
                "waitTime": "10",
                "beginOffset": "0",
                "endOffset": "3"
            },
            "name": "Reader",
            "category": "reader"
        },
        {
            "stepType": "odps",
            "parameter": {
                "partition": "",
                "truncate": true,
                "compress": false,
                "datasource": "odps_first",
                "column": [
                    "key",
                    "value",
                    "partition1",
                    "timestamp1",
                    "offset",
                    "t123",
                    "event_id",
                    "tag"
                ],
                "emptyAsNull": false,
                "table": "testkafka"
            },
            "name": "Writer",
            "category": "writer"
        }
    ],
    "version": "2.0",
    "order": {
        "hops": [
            {
                "from": "Reader",
                "to": "Writer"
            }
        ]
    },
    "setting": {
        "errorLimit": {
            "record": ""
        },
        "speed": {
            "throttle": false,
            "concurrent": 1,
        }
    }
}
```

您可以通过在Header主机上使用**kafka-consumer-groups.sh --bootstrap-server emr-header-1:9092 --list**命令查看group.id参数，及消费者的Group名称。

- 命令示例

```
kafka-consumer-groups.sh  --bootstrap-server emr-header-1:9092  --list
```

- 返回结果

```
_emr-client-metrics-handler-group
console-consumer-69493
console-consumer-83505
console-consumer-21030
console-consumer-45322
console-consumer-14773
```

以console-consumer-83505为例，您可以根据该参数在Header主机上使用**kafka-consumer-groups.sh --bootstrap-server emr-header-1:9092 --describe --group console-consumer-83505**命令确认beginOffset及endOffset参数。

- 命令示例。

```
kafka-consumer-groups.sh --bootstrap-server emr-header-1:9092 --describe --group console-consumer-83505
```

- 返回结果

```
TOPIC                          PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG        CONSUMER-ID
HOST                           CLIENT-ID
testkafka                      6          0               0               0          -
-                              -
test                           6          3               3               0          -
-                              -
testkafka                      0          0               0               0          -
-                              -
testkafka                      1          1               1               0          -
-                              -
testkafka                      5          0               0               0          -
-                              -
```

6. 配置调度资源组。
   i. 在节点编辑页面的右侧导航栏，单击**调度配置**。
   ii. 在**资源属性**区域，选择**调度资源组**为您创建的独享数据集成资源组。

   > **说明** 如果您需要将Kafka的数据周期性（例如每小时）写入MaxCompute，您可以使用beginDateTime及endDateTime参数，设置数据读取的时间区间为1小时，然后每小时调度一次数据集成任务。详情请参见Kafka Reader。

7. 单击图标运行代码。
8. 您可以在**运行日志**查看运行结果。

后续步骤

您可以新建一个ODPS SQL节点运行SQL语句，查看从Kafka同步数据至MaxCompute是否成功。详情请参见使用临时查询运行SQL语句（可选）。

2.6. 将消息队列for Apache Kafka的数据迁移至MaxCompute

本文介绍如何使用DataWorks数据集成，将消息队列for Apache Kafka集群上的数据迁移至MaxCompute，方便您对离线数据进行分析加工。

前提条件

- 消息队列 for Apache Kafka集群运行正常。本文以部署在华东1（杭州）地域的集群为例。
- 开通 MaxCompute。
- 开通 DataWorks 。
- 创建MaxCompute项目。本文以在华东1（杭州）地域创建名为bigdata_DOC的项目为例。

步骤一：准备Kafka数据

1. 登录消息队列 for Apache Kafka控制台创建Topic和Consumer Group，分别命名为testkafka和console-consumer。详情请参见步骤三：创建资源。
   本示例中， console-consumer将用于消费testkafka中的数据。
2. 向testkafka中写入数据。由于消息队列 for Apache Kafka用于处理流式数据，您可以持续不断地向其中写入数据。为保证测试结果，建议写入10条以上的数据。您可以直接在控制台使用发送消息功能来写入数据，也可以使用消息队列 for Apache Kafka的SDK收发消息。详情参见使用SDK收发消息。
3. 为验证写入数据是否生效，您可以在控制台查询消息，查看之前写入Topic中的数据。详情请参见查询消息。

步骤二：在DataWorks上创建目标表

您需要在DataWorks上创建目标表，以保证MaxCompute可以接收消息队列for Apache Kafka的数据。

1. 进入**数据开发**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 单击相应工作空间后的**数据开发**。
2. 右键单击**业务流程**，选择**新建 > MaxCompute > 表**。
3. 在**新建表**页面，选择引擎类型并输入**表名**。
4. 在表的编辑页面，单击**DDL模式**。
5. 在**DDL模式**对话框，输入如下建表语句，单击**生成表结构**。

```
CREATE TABLE testkafka
(
 key            string,
 value          string,
 partition1     string,
 timestamp1     string,
 offset         string,
 t123           string,
 event_id       string,
 tag            string
) ;
```

   建表语句中的每一列对应DataWorks数据集成Kafka Reader的默认列：
   - key：表示消息的 Key。
   - value：表示消息的完整内容 。
   - partition：表示当前消息所在分区。
   - headers：表示当前消息 headers 信息。
   - offset：表示当前消息的偏移量。
   - timestamp：表示当前消息的时间戳。

   您还可以自主命名，详情参见Kafka Reader。
6. 单击**提交到生产环境**并**确认**。

步骤三：同步数据

1. 新增和使用自定义数据集成资源组。此处创建的ECS实例将用以完成数据集成任务。
2. 新建数据集成节点。
   i.
   ii.
3.
4.
5.
6. 配置脚本，示例代码如下。

```
{
    "type": "job",
    "steps": [
        {
            "stepType": "kafka",
            "parameter": {
                "server": "47.xxx.xxx.xxx:9092",
                "kafkaConfig": {
                    "group.id": "console-consumer"
                },
                "valueType": "ByteArray",
                "column": [
                    "__key__",
                    "__value__",
                    "__partition__",
                    "__timestamp__",
                    "__offset__",
                    "'123'",
                    "event_id",
                    "tag.desc"
                ],
                "topic": "testkafka",
                "keyType": "ByteArray",
                "waitTime": "10",
                "beginOffset": "0",
                "endOffset": "3"
            },
            "name": "Reader",
            "category": "reader"
        },
        {
            "stepType": "odps",
            "parameter": {
                "partition": "",
                "truncate": true,
                "compress": false,
                "datasource": "odps_first",
                "column": [
                    "key",
                    "value",
                    "partition1",
                    "timestamp1",
                    "offset",
                    "t123",
                    "event_id",
                    "tag"
                ],
                "emptyAsNull": false,
                "table": "testkafka"
            },
            "name": "Writer",
            "category": "writer"
        }
    ],
    "version": "2.0",
    "order": {
        "hops": [
            {
                "from": "Reader",
                "to": "Writer"
            }
        ]
    },
    "setting": {
        "errorLimit": {
            "record": ""
        },
        "speed": {
            "throttle": false,
            "concurrent": 1
        }
    }
}
```

7. 配置调度资源组。
   i. 在右侧导航栏，单击**调度配置**。
   ii. 在**资源属性**区域，选择**调度资源组**为步骤 1 中创建的自定义资源组。
8. 单击图标运行代码。
9. 您可以在**运行日志**查看运行结果。

后续步骤

您可以新建一个ODPS SQL节点运行SQL语句，查看从Kafka同步数据至MaxCompute是否成功。详情请参见使用临时查询运行SQL语句（可选）。

2.7. Elasticsearch数据迁移至MaxCompute

本文为您介绍如何通过DataWorks数据同步功能，迁移阿里云Elasticsearch集群上的数据至MaxCompute。

前提条件

- 已开通MaxCompute服务。

  开通指导，详情请参见开通MaxCompute。
- 已开通DataWork服务。

  开通指导，详情请参见开通DataWorks。
- 在DataWorks上已完成创建业务流程。

  本例使用DataWorks简单模式，详情请参见创建业务流程。
- 已搭建阿里云Elasticsearch集群。

  进行数据迁移前，您需要保证自己的阿里云Elasticsearch集群环境正常。搭建阿里云Elasticsearch集群的详细过程，请参见快速入门。

  本示例中阿里云Elasticsearch的具体配置如下：
  - 地域：华东2（上海）
  - 可用区：上海可用区B
  - 版本：5.5.3 with Commercial Feature

背景信息

Elasticsearch是一个基于Lucene的搜索服务器，它提供了一个多用户分布式的全文搜索引擎。Elasticsearch是遵从Apache开源条款的一款开源产品，是当前主流的企业级搜索引擎。

阿里云Elasticsearch提供Elasticsearch 5.5.3 with Commercial Feature、6.3.2 with Commercial Feature、6.7.0 with Commercial Feature及商业插件X-pack服务，致力于数据分析、数据搜索等场景服务。在开源Elasticsearch基础上提供企业级权限管控、安全监控告警、自动报表生成等功能。

操作步骤

1. 在Elasticsearch上创建源表。详情请参见通过DataWorks将MaxCompute数据同步至Elasticsearch。
2. 在MaxCompute上创建目标表。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 在**工作空间列表**页面，单击相应工作空间后的**数据开发**。
   iv. 在**数据开发**页面，右键单击目标工作流程，选择**新建 > MaxCompute > 表**。
   v. 在弹出的**新建表**对话框中，填写**表名**，并单击**提交**。

      > 说明
      > - 表名必须以字母开头，不能包含中文或特殊字符。
      > - 如果绑定多个实例，则需要选择MaxCompute引擎实例。

   vi. 在表的编辑页面，单击**DDL模式**。

vii. 在**DDL模式**对话框，输入如下建表语句，单击**生成表结构**。

```
create table elastic2mc_bankdata
(
age             string,
job             string,
marital         string,
education       string,
default         string,
housing         string,
loan            string,
contact         string,
month           string,
day of week     string
);
```

viii. 单击**提交到生产环境**。

3. 同步数据。

i.

ii.

iii.

iv.

v.

vi. 配置脚本。

示例代码如下。代码释义请参见Elasticsearch Reader。

```
{
    "type": "job",
    "steps": [
        {
            "stepType": "elasticsearch",
            "parameter": {
                "retryCount": 3,
                "column": [
                    "age",
                    "job",
                    "marital",
                    "education",
                    "default",
                    "housing",
                    "loan",
                    "contact",
                    "month",
                    "day_of_week",
                    "duration",
                    "campaign",
                    "pdays",
                    "previous",
                    "poutcome",
                    "emp_var_rate",
                    "cons_price_idx",
                    "cons_conf_idx",
                    "euribor3m",
                    "nr_employed",
                    "y"
                ],
                "scroll": "1m",
                "index": "es_index",
                "pageSize": 1,
                "sort": {
                    "age": "asc"
},
                "type": "elasticsearch",
                "connTimeOut": 1000,
                "retrySleepTime": 1000,
                "endpoint": "http://es-cn-xxxx.xxxx.xxxx.xxxx.com:9200",
                "password": "xxxx",
                "search": {
                    "match_all": {}
                },
                "readTimeOut": 5000,
                "username": "xxxx"
```

```
            },
            "name": "Reader",
            "category": "reader"
        },
        {
            "stepType": "odps",
            "parameter": {
                "partition": "",
                "truncate": true,
                "compress": false,
                "datasource": "odps_first",
                "column": [
                    "age",
                    "job",
                    "marital",
                    "education",
                    "default",
                    "housing",
                    "loan",
                    "contact",
                    "month",
                    "day_of_week",
                    "duration",
                    "campaign",
                    "pdays",
                    "previous",
                    "poutcome",
                    "emp_var_rate",
                    "cons_price_idx",
                    "cons_conf_idx",
                    "euribor3m",
                    "nr_employed",
                    "y"
                ],
                "emptyAsNull": false,
                "table": "elastic2mc_bankdata"
            },
            "name": "Writer",
            "category": "writer"
        }
    ],
    "version": "2.0",
    "order": {
        "hops": [
            {
                "from": "Reader",
                "to": "Writer"
            }
        ]
    },
    "setting": {
        "errorLimit": {
            "record": "0"
        },
        "speed": {
            "throttle": false,
            "concurrent": 1,
            "dmu": 1
        }
    }
}
```

> **说明** 您可以在创建的阿里云Elasticsearch集群的**基本信息**中，查看公网地址和公网端口信息。

vii. 单击图标运行代码。

viii. 您可以在**运行日志**查看运行结果。

4. 查看结果。
   i. 右键单击业务流程，选择**新建 > MaxCompute > ODPS SQL**。
   ii. 在**新建节点**对话框中输入**节点名称**，并单击**提交**。

iii. 在ODPS SQL节点编辑页面输入如下语句。

```
SELECT * FROM elastic2mc_bankdata;
```

iv. 单击图标运行代码。

v. 您可以在**运行日志**查看运行结果。

2.8. RDS迁移至MaxCompute实现动态分区

本文为您介绍如何使用DataWorks数据集成同步功能自动创建分区，动态地将RDS中的数据迁移至MaxCompute大数据计算服务。

前提条件

- 准备DataWorks环境
  i. 开通MaxCompute和DataWorks。
  ii. 开通DataWorks。
  iii. 在DataWorks上完成创建业务流程，本例使用DataWorks简单模式。详情请参见创建业务流程。
- 新增数据源
  - 新增MySQL数据源作为数据来源，详情请参见配置MySQL数据源。
  - 新增MaxCompute数据源作为目标数据源接收RDS数据，详情请参见配置MaxCompute数据源。

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。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。

1. 登录DataWorks控制台。
2. 在MaxCompute上创建目标表。
   i.
   ii. 单击相应工作空间后的**数据开发**。
   iii. 右键单击已创建的**业务流程**，选择**新建 > MaxCompute > 表**。
   iv. 在**新建表**页面，选择引擎类型并输入**表名**。
   v. 在表的编辑页面，单击**DDL模式**。
   vi. 在**DDL模式**对话框，输入如下建表语句，单击**生成表结构**。

```
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
);
```

   vii. 单击**提交到生产环境**。
3. 新建离线同步节点。
   i.
   ii.

iii. 选择数据来源和数据去向。

4. 配置分区参数。

i. 在右侧导航栏上，单击**调度配置**。

ii. 在**基础属性**区域，设置**参数**。参数值默认为系统自带的时间参数 `${bizdate}`，格式为yyyymmdd。

> 说明 默认参数值与**数据去向**中的**分区信息**值对应。调度执行迁移任务时，目标表的分区值会被自动替换为任务执行日期的前一天，默认情况下，您会在当前执行前一天的业务数据，这个日期也叫做业务日期。如果您需要使用当天任务运行的日期作为分区值，则需自定义参数值。

自定义参数设置：用户可以自主选择某一天和格式配置，如下所示：

- 后N年：`$[add_months(yyyymmdd,12*N)]`
- 前N年：`$[add_months(yyyymmdd,-12*N)]`
- 前N月：`$[add_months(yyyymmdd,-N)]`
- 后N周：`$[yyyymmdd+7*N]`
- 后N月：`$[add_months(yyyymmdd,N)]`
- 前N周：`$[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
> - 使用中括号（[]）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调度参数概述。

5. 单击图标运行代码。

6. 您可以在**运行日志**查看运行结果。

补数据实验

如果您的数据中存在大量运行日期之前的历史数据，需要实现自动同步和自动分区。您可以通过DataWorks的**运维中心**，选择当前的同步数据节点，使用**补数据**功能实现。

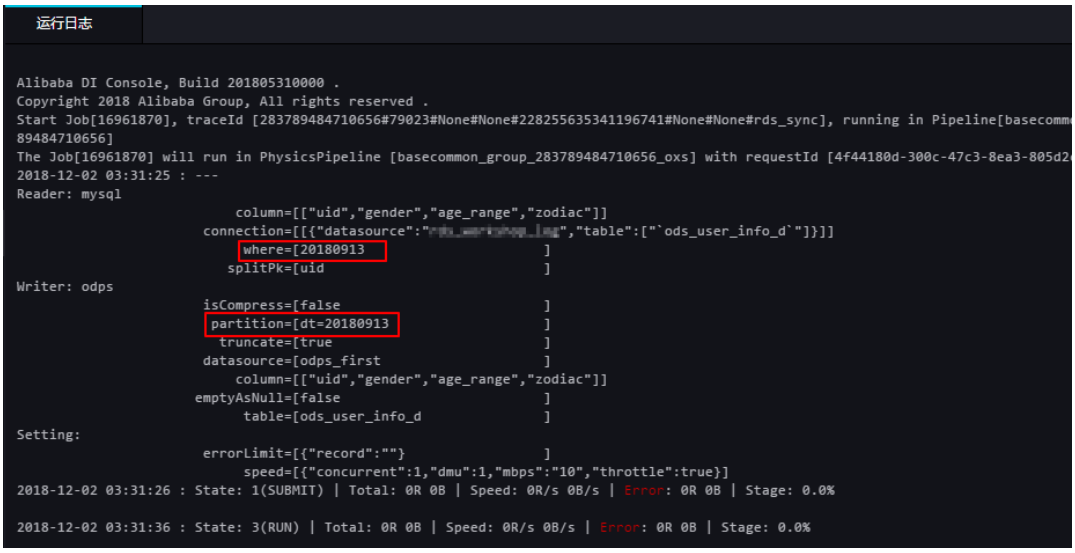
1. 在RDS端按照日期筛选出历史数据。

   您可以在同步节点**数据来源**区域设置**数据过滤**条件。

2. 执行补数据操作。详情请参见执行补数据并查看补数据实例

3. 在运行的日志中查看对RDS数据的抽取结果。

   从运行结果中可以看到MaxCompute已自动创建分区。



4. 运行结果验证。在MaxCompute客户端执行如下命令，查看数据写入情况。

```
SELECT count(*) from ods_user_info_d where dt = 20180913;
```

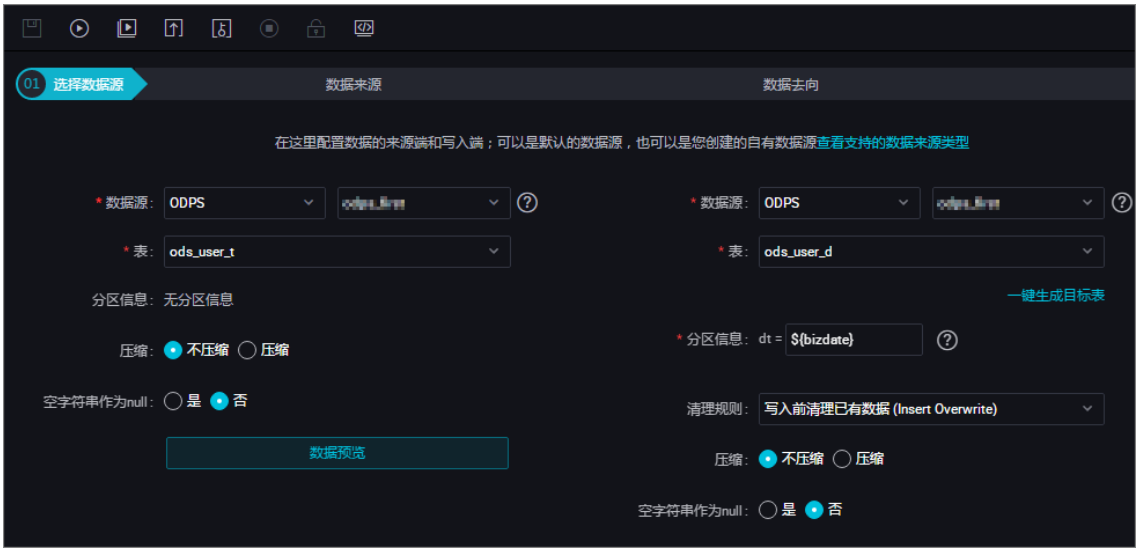
Hash实现非日期字段分区

如果您的数据量较大，或没有按照日期字段对第一次全量的数据进行分区，而是按照省份等非日期字段分区，则此时数据集成操作将不能实现自动分区。这种情况下，您可以按照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。

1. 将数据全量同步到MaxCompute的一个临时表，创建一个SQL脚本节点。执行如下命令。

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

2. 创建同步任务的节点mysql_to_odps，即简单的同步任务。将RDS数据全量同步到MaxCompute，无需设置分区。

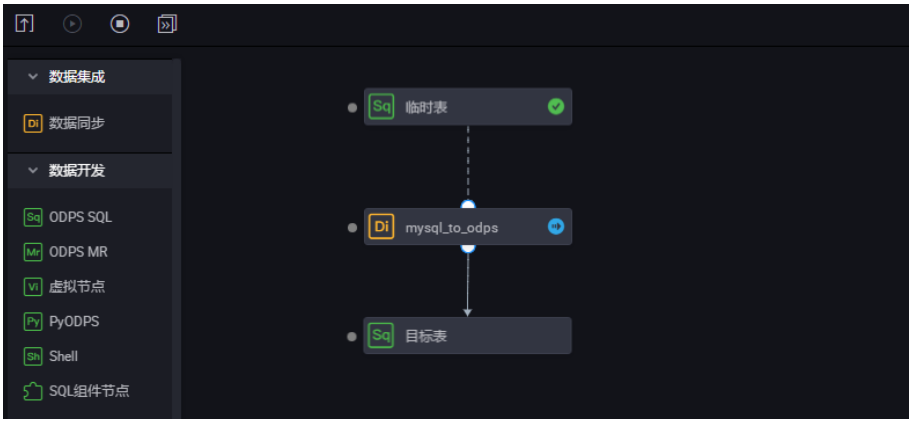


3. 使用SQL语句进行动态分区到目标表，命令如下。

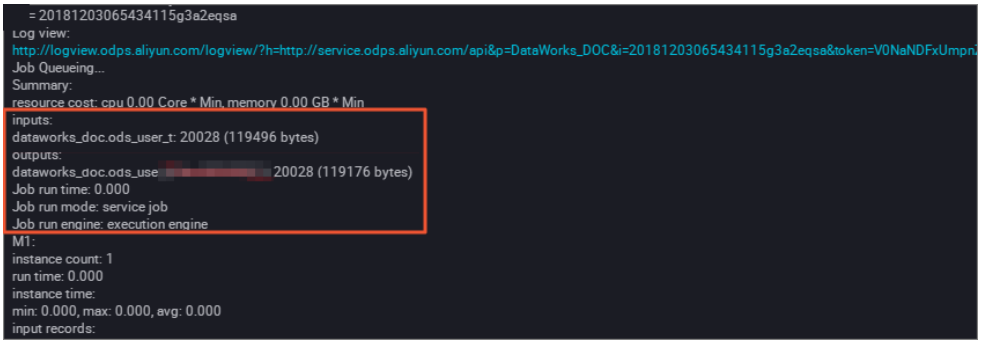
```
drop table if exists ods_user_d;
//创建一个ODPS分区表（最终目的表）。
    CREATE TABLE ods_user_d (
    uid STRING,
        gender STRING,
        age_range STRING,
        zodiac STRING
)
PARTITIONED BY (
    dt STRING
);
//执行动态分区SQL，按照临时表的字段dt自动分区，dt字段中相同的数据值，会按照这个数据值自动创建一个分区值。
//例如dt中有些数据是20181025，会自动在ODPS分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中您可以通过SQL语句完成数据同步。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. 将三个节点配置成一个工作流，按顺序执行。



5. 查看执行过程。您可以重点观察最后一个节点的动态分区过程。



6. 运行结果验证。在MaxCompute客户端执行如下命令，查看数据写入情况。

```
SELECT count(*) from ods_user_d where dt = 20180913;
```

2.9. JSON数据从MongoDB迁移至MaxCompute

本文为您介绍如何通过DataWorks的数据集成功能，将从MongoDB提取的JSON字段迁移至MaxCompute。

前提条件

- 开通MaxCompute和DataWorks。
- 开通DataWorks。
- 在DataWorks上完成创建业务流程，本例使用DataWorks简单模式。详情请参见创建业务流程。

在MongoDB上准备测试数据

1. 账号准备。

   在数据库内新建用户，用于DataWorks添加数据源。本示例执行如下命令。

   ```
   db.createUser({user:"bookuser",pwd:"123456",roles:["root"]})
   ```

   新建用户名为bookuser，密码为123456，权限为root。

2. 数据准备。

   将数据上传至MongoDB数据库。本示例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
               }
   ```

3. 在MongoDB的DMS控制台，本示例使用的数据库为admin，集合为userlog。执行如下命令，查看已上传的数据。

```
db.userlog.find().limit(10)
```

通过DataWorks将JSON数据从MongoDB迁移至MaxCompute

1. 登录DataWorks控制台。
2. 在DataWorks上创建目标表。用以接收从MongoDB迁移的数据。
   i. 右键单击已创建的**业务流程**，选择**新建 > MaxCompute > 表**。
   ii. 在**新建表**页面，选择引擎类型并输入**表名**。
   iii. 在表的编辑页面，单击**DDL模式**。
   iv. 在**DDL模式**对话框，输入建表语句，单击**生成表结构**。

   ```
   create table mqdata (mqdata string);
   ```

   v. 单击**提交到生产环境**。
3. 新增MongoDB数据源，详情请参见配置MongoDB数据源。
4. 创建离线同步节点。
   i.
   ii.
   iii.
   iv.
   v.

vi. 输入如下脚本。

```
{
    "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.String" //非一层子属性以最终获取的类型为准。假如您选取的JSON字段为一级字段，例如本例中的expensive，则直接填写string即可。
                }
              ],
            "collectionName": "userlog"   //集合名称。
            },
        "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,
            }
            }
        }
```

vii. 单击图标运行代码。

viii. 您可以在**运行日志**查看运行结果。

验证结果

1. 右键单击业务流程，选择**新建 > MaxCompute > ODPS SQL**。
2. 在**新建节点**对话框中输入**节点名称**，并单击**提交**。
3. 在ODPS SQL节点编辑页面输入如下语句。

```
SELECT * from mqdata;
```

4. 单击图标运行代码。
5. 您可以在**运行日志**查看运行结果。

2.10. JSON数据从OSS迁移至MaxCompute

本文为您介绍如何通过DataWorks数据集成，将JSON数据从OSS迁移至MaxCompute，并使用MaxCompute内置字符串函数GET_JSON_OBJECT提取JSON信息。

前提条件

- 开通MaxCompute和DataWorks。
- 开通DataWorks。
- 在DataWorks上完成创建业务流程，本例使用DataWorks简单模式。详情请参见创建业务流程。
- 将JSON文件重命名为后缀为 `.txt` 的文件，并上传至OSS。本文中OSS Bucket地域为华东2（上海）。示例文件如下。

```
{
    "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JSON数据从OSS迁移至MaxCompute

1. 新增OSS数据源。详情请参见配置OSS数据源。
2. 在DataWorks上新建数据表，用于存储迁移的JSON数据。
   i.
   ii. 在**新建表**页面，选择引擎类型并输入**表名**。
   iii. 在表的编辑页面，单击**DDL模式**。
   iv. 在**DDL模式**对话框，输入如下建表语句，单击**生成表结构**。

   ```
   create table mqdata (mq_data string);
   ```

   v. 单击**提交到生产环境**。
3. 新建离线同步节点。
   i.
   ii.
   iii.
   iv.
   v.

vi. 修改JSON代码后，单击按钮。

示例代码如下。

```
{
    "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,
        }
    }
}
```

结果验证

1. 新建ODPS SQL节点。
   i. 右键单击业务流程，选择**新建 > MaxCompute > ODPS SQL**。
   ii. 在**新建函数**对话框中，输入**函数名称**，单击**提交**。

iii. 在ODPS SQL节点编辑页面输入如下语句。

```
--查询表mq_data数据。
SELECT * from mqdata;
--获取JSON文件中的EXPENSIVE值。
SELECT GET_JSON_OBJECT(mqdata.MQdata,'$.expensive') FROM mqdata;
```

iv. 单击图标运行代码。

v. 您可以在**运行日志**查看运行结果。

2.11. MaxCompute数据迁移至OTS

本文为您介绍如何将MaxCompute数据迁移至表格存储OTS（Table Store）。

前提条件

- 开通MaxCompute和DataWorks。
- 开通DataWorks。
- 在DataWorks上完成创建业务流程，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

1. 在DataWorks上创建表。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 单击相应工作空间后的**数据开发**。
   iv. 右键单击已创建的**业务流程**，选择**新建 > MaxCompute > 表**。
   v. 在**新建表**页面，选择引擎类型并输入**表名**。
   vi. 在表的编辑页面，单击**DDL模式**。
   vii. 在**DDL模式**对话框，输入如下建表语句，单击**生成表结构**。

```
create table Transs
(name    string,
id    bigint,
gender    string);
```

   viii. 单击**提交到生产环境**。
2. 为表Transs导入数据。
   i. 在**数据开发**页面，单击图标。
   ii. 在**数据导入向导**对话框，至少输入3个字母来搜索需要导入数据的表，单击**下一步**。
   iii. **选择数据导入方式**为**上传本地数据**，单击**选择文件**后的**浏览...**。选择本地数据文件，配置导入信息。
   示例数据如下。

```
qwe,145,F
asd,256,F
xzc,345,M
rgth,234,F
ert,456,F
dfg,12,M
tyj,4,M
bfg,245,M
nrtjeryj,15,F
rwh,2344,M
trh,387,F
srjeyj,67,M
saerh,567,M
```

   iv. 单击**下一步**。
   v. 选择目标表字段与源字段的匹配方式。
   vi. 单击**导入数据**。
3. 在表格存储控制台上创建表。
   i. 登录表格存储控制台，创建实例。详情请参见创建实例。
   ii. 创建数据表Trans。详情请参见创建数据表。
4. 在DataWorks中新增数据源。
   i.

ii. 在左侧导航栏，单击**工作空间列表**。

iii.

iv.

v. 单击右上角**新增数据源**，并选择数据类型为**ODPS**。

vi. 在**新增ODPS数据源**对话框中配置参数，并单击**完成**。详情请参见配置MaxCompute数据源。

vii. 新增OTS数据源，详情请参见配置OTS数据源。

5. 配置MaxCompute（ODPS）Reader和表格存储（OTS）Writer。

i.

ii.

iii.

iv.

v.

vi. 修改JSON代码后，单击图标。

代码如下。

```
{
    "type": "job",
    "steps": [
        {
            "stepType": "odps",
            "parameter": {
                "partition": [],
                "datasource": "odps_first",
                "column": [
                    "name",
                    "id",
                    "gender"
                ],
                "table": "Transs"
            },
            "name": "Reader",
            "category": "reader"
        },
        {
            "stepType": "ots",
            "parameter": {
                "datasource": "Transs",
                "column": [
                    {
                        "name": "Gender",
                        "type": "STRING"
                    }
                ],
                "writeMode": "UpdateRow",
                "table": "Trans",
                "primaryKey": [
                    {
                        "name": "Name",
                        "type": "STRING"
                    },
                    {
                        "name": "ID",
                        "type": "INT"
                    }
                ]
            },
            "name": "Writer",
            "category": "writer"
        }
    ],
    "version": "2.0",
    "order": {
        "hops": [
            {
                "from": "Reader",
                "to": "Writer"
            }
        ]
    },
    "setting": {
        "errorLimit": {
            "record": "0"
        },
        "speed": {
            "throttle": false,
            "concurrent": 1,
            "dmu": 1
        }
    }
}
```

6. 在表格存储控制台中查看新增的表数据。
   i. 登录表格存储控制台。
   ii. 在左侧导航栏上，单击**全部实例**。
   iii. 单击实例名称进入**实例管理**页面。在**数据表列表**区域，单击要查看的数据表名称。

iv. 单击顶部**数据管理**页签，查看新增的表数据。

2.12. MaxCompute数据迁移至OSS

本文为您介绍如何使用DataWorks的数据同步功能将MaxCompute数据迁移至对象存储OSS（Object Storage Service）。

前提条件

- 开通MaxCompute和DataWorks。
- 开通DataWorks。
- 在DataWorks上完成创建业务流程，本例使用DataWorks简单模式。详情请参见创建业务流程。

操作步骤

1. 在DataWorks上创建表。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 单击相应工作空间后的**数据开发**。
   iv. 右键单击已创建的**业务流程**，选择**新建 > MaxCompute > 表**。
   v. 在**新建表**页面，选择引擎类型并输入**表名**。
   vi. 在表的编辑页面，单击**DDL模式**。
   vii. 在**DDL模式**对话框，输入如下建表语句，单击**生成表结构**。

```
create table Transs
(name    string,
id    string,
gender    string);
```

   viii. 单击**提交到生产环境**。
2. 为表Transs导入数据。
   i. 在**数据开发**页面，单击图标。
   ii. 在**数据导入向导**对话框，至少输入3个字母来搜索需要导入数据的表，单击**下一步**。
   iii. **选择数据导入方式**为**上传本地数据**，单击**选择文件**后的**浏览...**。选择本地数据文件，配置导入信息。
   示例数据如下。

```
qwe,145,F
asd,256,F
xzc,345,M
rgth,234,F
ert,456,F
dfg,12,M
tyj,4,M
bfg,245,M
nrtjeryj,15,F
rwh,2344,M
trh,387,F
srjeyj,67,M
saerh,567,M
```

   iv. 单击**下一步**。
   v. 选择目标表字段与源字段的匹配方式。
   vi. 单击**导入数据**。
3. 在OSS控制台上创建表。
   i. 登录OSS控制台，创建Bucket。详情请参见创建存储空间。
   ii. 上传文件*qwee.csv*至OSS。详情请参见上传文件。

> **说明** 请确保*qwee.csv*文件中的字段与表Transs的字段完全一致。

4. 在DataWorks上新增数据源。
   i.
   ii.
   iii.
   iv. 在左侧导航栏上，单击**数据源**，进入**数据源管理**页面。
   v. 单击右上角**新增数据源**，并选择数据类型为**ODPS**。

vi. 在**新增ODPS数据源**对话框中配置参数，并单击**完成**。详情请参见配置MaxCompute数据源。

vii. 新增OSS数据源，详情请参见配置OSS数据源。

5. 配置MaxCompute（ODPS）Reader和对象存储（OSS）Writer。

i.

ii.

iii.

iv.

v.

vi. 修改JSON代码后，单击图标。

示例代码如下。

```
{
    "order":{
        "hops":[
            {
                "from":"Reader",
                "to":"Writer"
            }
        ]
    },
    "setting":{
        "errorLimit":{
            "record":"0"
        },
        "speed":{
            "concurrent":1,
            "dmu":1,
            "throttle":false
        }
    },
    "steps":[
        {
            "category":"reader",
            "name":"Reader",
            "parameter":{
                "column":[
                    "name",
                    "id",
                    "gender"
                ],
                "datasource":"odps_first",
                "partition":[],
                "table":"Transs"
            },
            "stepType":"odps"
        },
        {
            "category":"writer",
            "name":"Writer",
            "parameter":{
                "datasource":"Trans",
                "dateFormat":"yyyy-MM-dd HH:mm:ss",
                "encoding":"UTF-8",
                "fieldDelimiter":",",
                "fileFormat":"csv",
                "nullFormat":"null",
                "object":"qweee.csv",
                "writeMode":"truncate"
            },
            "stepType":"oss"
        }
    ],
    "type":"job",
    "version":"2.0"
}
```

6. 在OSS控制台中查看新增的表数据。详情请参见下载文件。

2.13. 迁移ECS自建MySQL数据库至MaxCompute

本文为您介绍如何使用独享数据集成资源，将您在ECS上自建的MySQL数据库中的数据，迁移到MaxCompute。

前提条件

- 已拥有至少一个绑定专有网络VPC的ECS（请勿使用经典网络），并在ECS上安装好MySQL数据库，数据库中已创建好用户和测试数据。本文中ECS自建MySQL的测试数据创建语句如下。

```
CREATE TABLE IF NOT EXISTS good_sale(
   create_time timestamp,
   category varchar(20),
   brand varchar(20),
   buyer_id varchar(20),
   trans_num varchar(20),
   trans_amount DOUBLE,
   click_cnt varchar(20)
   );
insert into good_sale values('2018-08-21','coat','brandA','lilei',3,500.6,7),
('2018-08-22','food','brandB','lilei',1,303,8),
('2018-08-22','coat','brandC','hanmeimei',2,510,2),
('2018-08-22','bath','brandA','hanmeimei',1,442.5,1),
('2018-08-22','food','brandD','hanmeimei',2,234,3),
('2018-08-23','coat','brandB','jimmy',9,2000,7),
('2018-08-23','food','brandA','jimmy',5,45.1,5),
('2018-08-23','coat','brandE','jimmy',5,100.2,4),
('2018-08-24','food','brandG','peiqi',10,5560,7),
('2018-08-24','bath','brandF','peiqi',1,445.6,2),
('2018-08-24','coat','brandA','ray',3,777,3),
('2018-08-24','bath','brandG','ray',3,122,3),
('2018-08-24','coat','brandC','ray',1,62,7) ;
```

- 请记录好您的ECS的**私有IP**、**专有网络**和**虚拟交换机**信息。

- ECS上的安全组已放通MySQL数据库所使用的端口（默认为3306），详情请参见添加安全组规则，请记录好您的安全组名称。

- 已成功创建DataWorks工作空间。本文使用DataWorks简单模式工作空间，计算引擎为MaxCompute。请保证您的ECS与DataWorks工作空间处于同一个地域，创建方法请参见创建工作空间。
- 已完成独享数据集成资源的购买，并且绑定了ECS所在的专有网络VPC。请注意独享资源组必须与ECS同一可用区，详情请参见新增和使用独享数据集成资源组。完成绑定后，您可以在**资源组列表**查看到您的独享资源组。

- 在**专有网络绑定**处查看**专有网络**、**交换机**和**安全组**信息是否和ECS一致。

背景信息

独享资源可以保障您的数据快速、稳定地传输。您购买的独享数据集成资源和需要访问的数据源（即本文中的ECS自建MySQL数据库）必须在同地域同可用区，且和DataWorks工作空间同地域。

操作步骤

1. 在DataWorks上创建MySQL数据源。
   i. 使用主账号登录DataWorks控制台。

ii. 在**工作空间列表**单击**进入数据集成**。

iii. 在左侧导航栏，单击**数据源**。

iv. 单击**数据源管理**页面右上角的**新增数据源**。

v. 在**新增数据源**页面，单击**MySQL**。

vi. 在**新增MySQL数据源**对话框中，配置各项参数，详情请参见配置MySQL数据源。

本文以**连接串模式**为例，在**JDBC URL**处输入您刚刚记录的ECS私有地址和MySQL的默认端口号3306。

> **说明** 当前VPC环境下的自建MySQL数据源暂不支持**测试连通性**，因此连通性测试失败是正常现象。

vii. 单击相应资源组后的**测试连通性**。

数据同步时，一个任务只能使用一种资源组。您需要在每种资源组上单独测试连通性，以保证同步任务使用的数据集成资源组能够与数据源连通，否则将无法正常执行数据同步任务。详情请参见选择网络连通方案。

viii. 测试连通性通过后，单击**完成**。

2. 创建MaxCompute表。

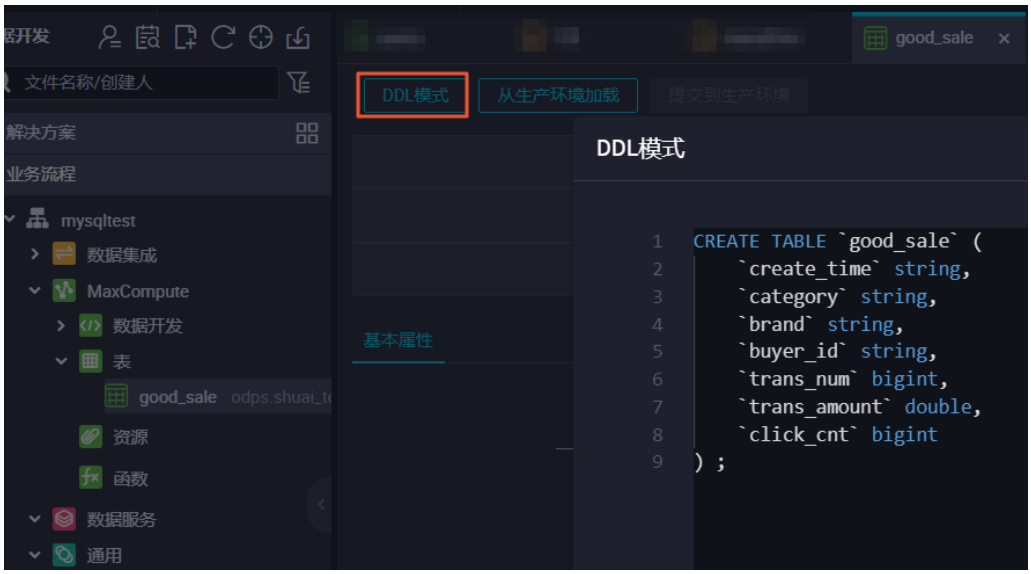
您需要通过DataWorks创建一个表，用于接收来自MySQL的测试数据。

i. 单击左上角的图标，选择**全部产品 > DataStudio（数据开发）**。

ii. 新建一个**业务流程**，详情请参见创建业务流程。

iii. 右键单击新建的业务流程，选择**新建 > MaxCompute > 表**。

iv. 输入您的MaxCompute表名称，本例中使用和MySQL数据库表一样的名称good_sale。单击**DDL模式**后，输入您的建表语句并**生成表结构**。



本例中使用的建表语句如下，请注意数据类型的转换。

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

v. 输入表的**中文名**后，单击**提交到生产环境**，完成MaxCompute表good_sale的创建。

3. 配置数据集成任务。

i. 右键单击业务流程，选择**新建 > 数据集成 > 离线同步**，创建一个数据集成任务。

ii. 选择您的数据来源为您刚添加的MySQl数据源，数据去向为默认MaxCompute数据源odps_first，单击**转换脚本**切换数据集成任务为脚本模式。

此时，如果产生报错或您无法选择数据来源的表，都属于正常现象，直接转换为脚本模式即可。

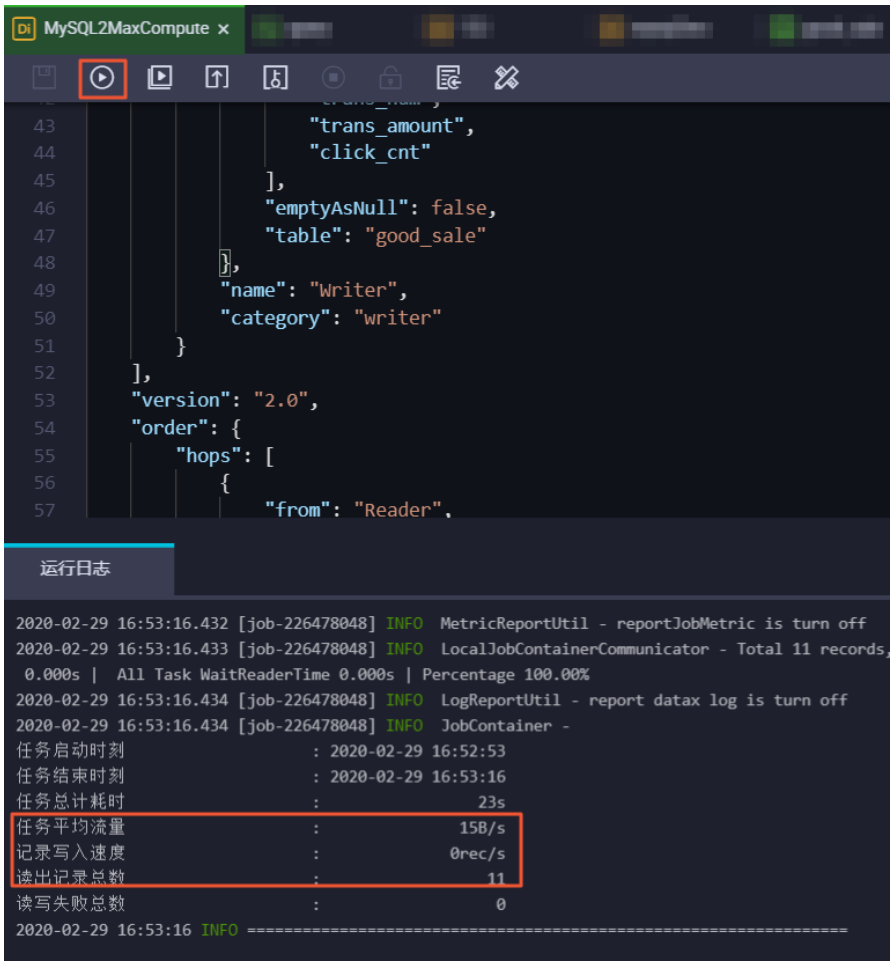
iii. 单击页面右侧的**数据集成资源组配置**，选中已购买的独享资源组。

如果未切换任务资源组为数据集成独享资源，后续您的任务将无法成功运行。

iv. 填写数据集成任务脚本内容如下。

```
{
    "type": "job",
    "steps": [
        {
            "stepType": "mysql",
            "parameter": {
                "column": [//源列名
                    "create_time",
                    "category",
                    "brand",
                    "buyer_id",
                    "trans_num",
                    "trans_amount",
                    "click_cnt"
                ],
                "connection": [
                    {
                        "datasource": "shuai",//源数据源
                        "table": [
                            "good_sale"//源数据库表名，此处必须为方括号数组格式。
                        ]
                    }
                ],
                "where": "",
                "splitPk": "",
                "encoding": "UTF-8"
            },
            "name": "Reader",
            "category": "reader"
        },
        {
            "stepType": "odps",
            "parameter": {
                "partition": "",
                "truncate": true,
                "datasource": "odps_first",//目标数据源
                "column": [//目标列名
                    "create_time",
                    "category",
                    "brand",
                    "buyer_id",
                    "trans_num",
                    "trans_amount",
                    "click_cnt"
                ],
                "emptyAsNull": false,
                "table": "good_sale"//目标表名
            },
            "name": "Writer",
            "category": "writer"
        }
    ],
    "version": "2.0",
    "order": {
        "hops": [
            {
                "from": "Reader",
                "to": "Writer"
            }
        ]
    },
    "setting": {
        "errorLimit": {
            "record": "0"
        },
        "speed": {
            "throttle": false,
            "concurrent": 2
        }
    }
}
```

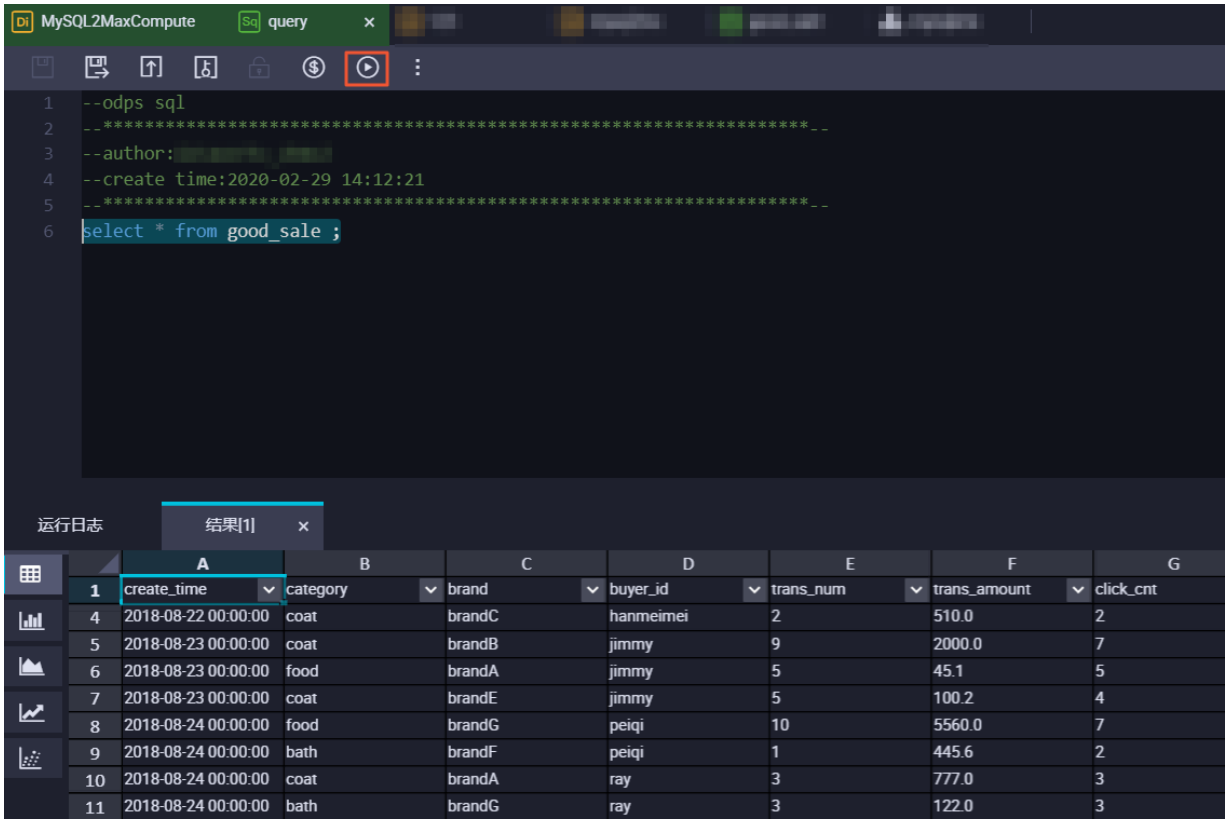
v. 单击运行，您可以在下方的**运行日志**查看数据是否已传输到MaxCompute。



执行结果

您可以新建一个**ODQP SQL**类型的节点，用于查询当前MaxCompute表中的数据。

输入您的查询语句 `select * from good_sale ;` ，单击**运行**，即可看到当前已传入MaxCompute表中的数据。



	A	B	C	D	E	F	G
1	create_time	category	brand	buyer_id	trans_num	trans_amount	click_cnt
4	2018-08-22 00:00:00	coat	brandC	hanmeimei	2	510.0	2
5	2018-08-23 00:00:00	coat	brandB	jimmy	9	2000.0	7
6	2018-08-23 00:00:00	food	brandA	jimmy	5	45.1	5
7	2018-08-23 00:00:00	coat	brandE	jimmy	5	100.2	4
8	2018-08-24 00:00:00	food	brandG	peiqi	10	5560.0	7
9	2018-08-24 00:00:00	bath	brandF	peiqi	1	445.6	2
10	2018-08-24 00:00:00	coat	brandA	ray	3	777.0	3
11	2018-08-24 00:00:00	bath	brandG	ray	3	122.0	3

2.14. Amazon Redshift数据迁移至MaxCompute

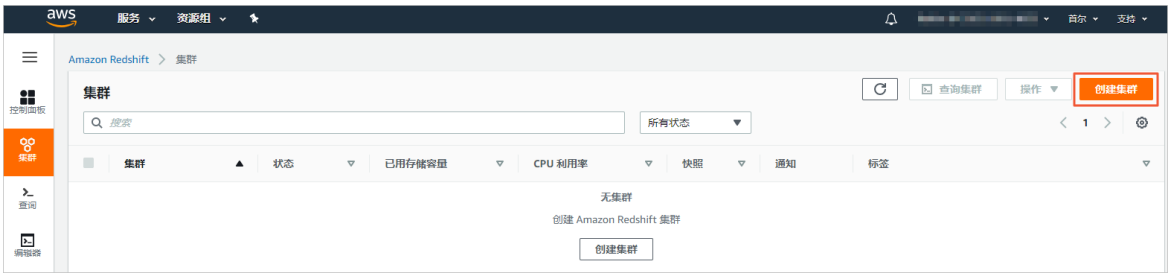
本文为您介绍如何通过公网环境将Amazon Redshift数据迁移至MaxCompute。

前提条件

- 准备Amazon Redshift集群环境及数据环境。

  您可以登录AWS官网，获取创建Redshift集群的详细操作内容，详情请参见Amazon Redshift集群管理指南。

  i. 创建Redshift集群。如果已有Redshift集群，您可以直接使用已有的Redshift集群。

  ii. 在Redshift集群中准备好需要迁移的Amazon Redshift数据。

  假设，已在public schema中准备好了TPC-H数据集。数据集使用MaxCompute 2.0数据类型和Decimal 2.0数据类型。

- 准备MaxCompute的项目环境

  操作详情请参见准备工作。

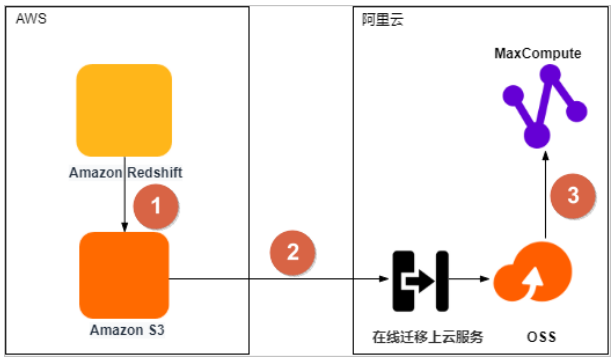
以**新加坡（新加坡）**区域为例，创建作为迁移目标的MaxCompute项目。由于TPC-H数据集使用MaxCompute 2.0数据类型和Decimal 2.0数据类型，因此本文创建的项目为MaxCompute 2.0版本。

- 开通阿里云OSS服务。

  开通阿里云OSS服务详情请参见开通OSS服务。

背景信息

将Amazon Redshift数据迁移至MaxCompute的流程如下。



序号	描述
①	将Amazon Redshift数据导出至Amazon S3数据湖（简称S3）。
②	通过对象存储服务OSS的**在线迁移上云服务**，将数据从S3迁移至OSS。
③	将数据从OSS迁移至同区域的MaxCompute项目中，并校验数据完整性和正确性。

> **说明** 由于Amazon Redshift和MaxCompute之间语法存在很多差异，因此在实际迁移过程中，您需要修改Amazon Redshift上编写的脚本，然后才能在MaxCompute中使用。更多Amazon Redshift和MaxCompute间的语法差异信息，请参见Amazon RedShift到MaxCompute迁移实践指导。

步骤一：将Amazon Redshift数据导出至S3

Amazon Redshift支持IAM角色和临时安全凭证（AccessKey）认证方式。您可以基于这两种认证方式通过Redshift UNLOAD命令将数据导出至S3。将Amazon Redshift数据导出至S3的详细操作内容请参见卸载数据。

两种认证方式的UNLOAD命令格式如下：

- 基于IAM角色的UNLOAD命令

```
-- 通过UNLOAD命令将表customer的内容导出至S3。
UNLOAD ('SELECT * FROM customer')
TO 's3://bucket_name/unload_from_redshift/customer/customer_' --S3 Bucket。
IAM_ROLE 'arn:aws:iam::****:role/MyRedshiftRole'; --角色ARN。
```

- 基于AccessKey的UNLOAD命令

```
-- 通过UNLOAD命令将表customer的内容导出至S3。
UNLOAD ('SELECT * FROM customer')
TO 's3://bucket_name/unload_from_redshift/customer/customer_' --S3 Bucket。
Access_Key_id '<access-key-id>'  --IAM用户的Access Key ID。
Secret_Access_Key '<secret-access-key>'  --IAM用户的Access Key Secret。
Session_Token '<temporary-token>';  --IAM用户的临时访问令牌。
```

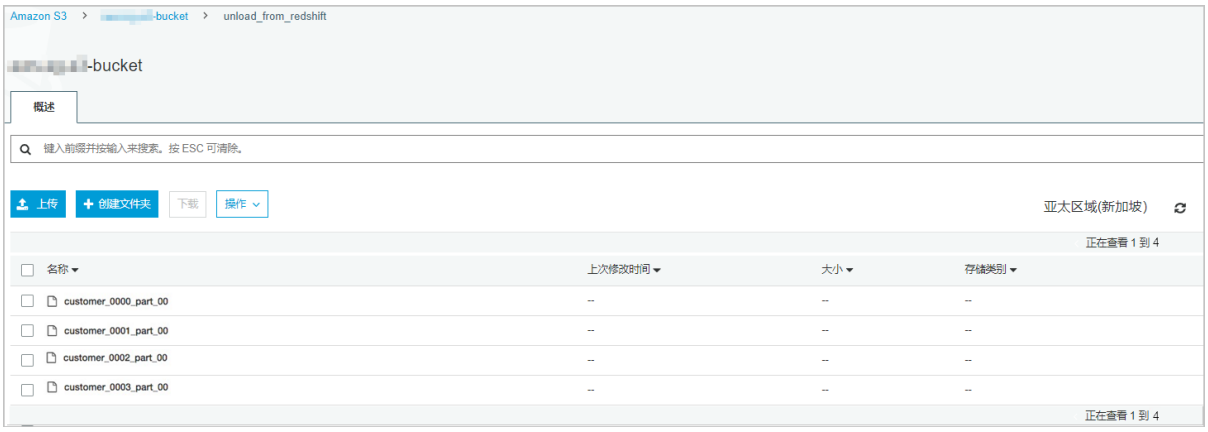
UNLOAD命令导出的数据格式如下：

- 默认格式

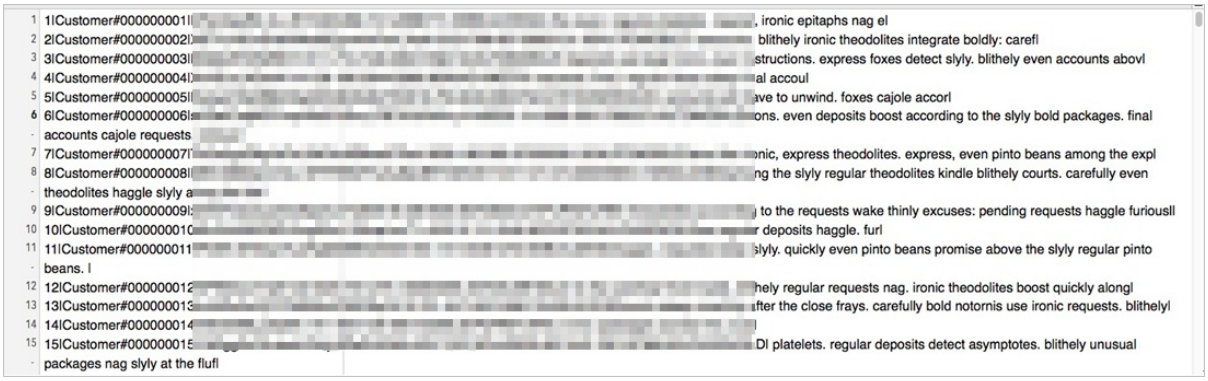
命令示例如下。

```
UNLOAD ('SELECT * FROM customer')
TO 's3://bucket_name/unload_from_redshift/customer/customer_'
IAM_ROLE 'arn:aws:iam::****:role/redshift_s3_role';
```

执行成功后，导出以竖线（|）分隔的文本文件。您可以登录S3控制台，在对应Bucket中查看导出的文本文件。



导出的文本文件格式如下。



- PARQUET格式

  以PARQUET格式导出，便于其它引擎直接读取数据。命令示例如下。

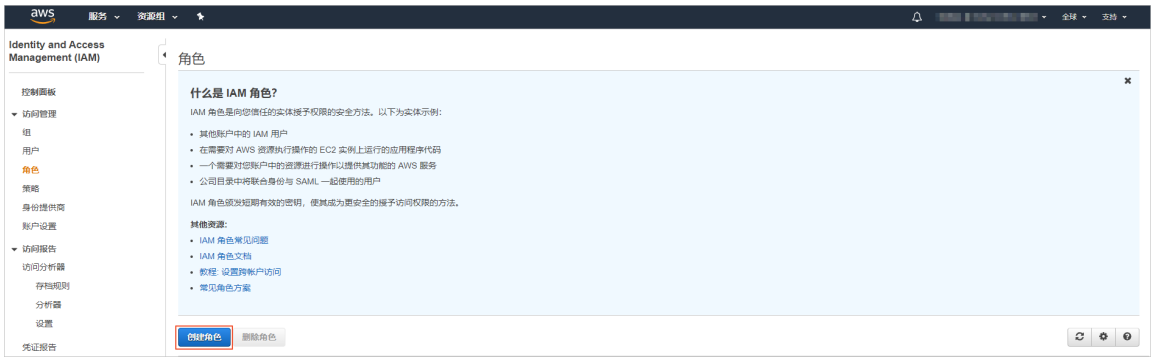
```
UNLOAD ('SELECT * FROM customer')
TO 's3://bucket_name/unload_from_redshift/customer_parquet/customer_'
FORMAT AS PARQUET
IAM_ROLE 'arn:aws:iam::xxxx:role/redshift_s3_role';
```

执行成功后，您可以在对应Bucket中查看导出的文件。PARQUET文件比文本文件更小，数据压缩率更高。

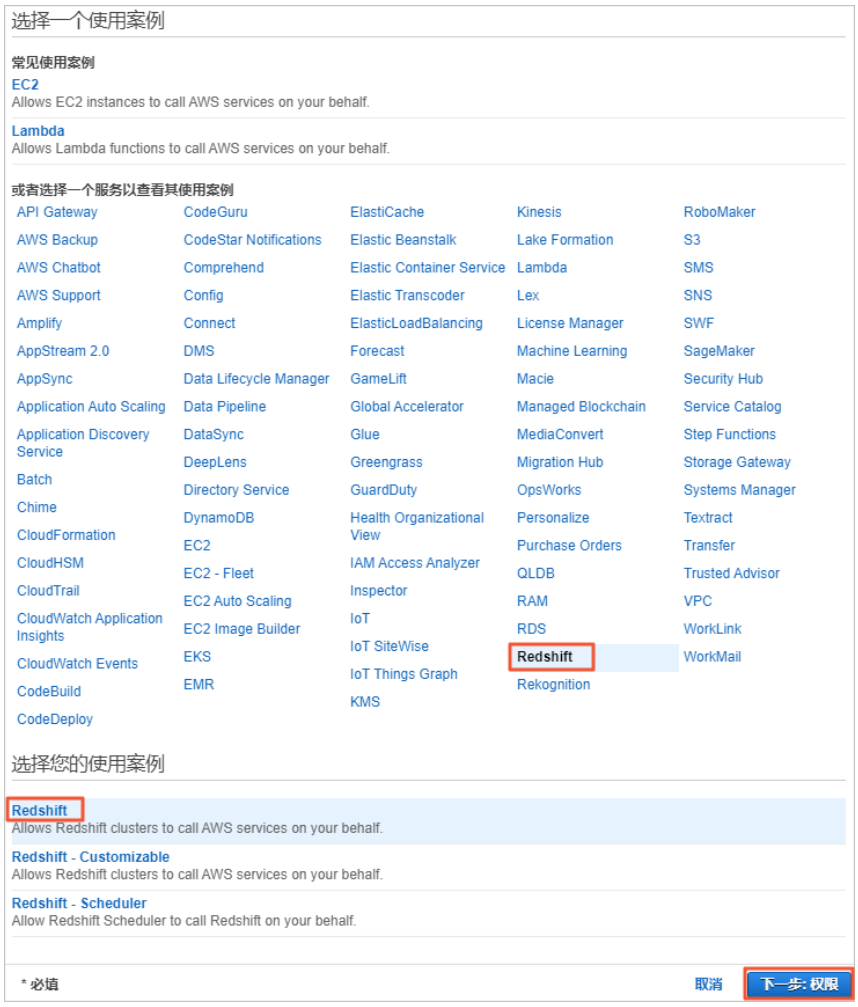
本文以IAM角色认证及导出PARQUET格式为例介绍数据迁移操作。

1. 新建Redshift类型的IAM角色。

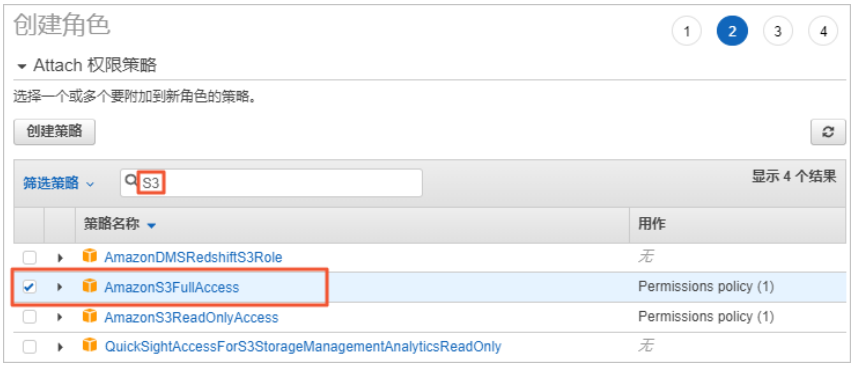
i. 登录IAM控制台，单击**创建角色**。



ii. 在**创建角色**页面的**选择一个使用案例**区域，单击**Redshift**。在**选择您的使用案例**区域单击**Redshift-Customizable**后，单击**下一步：权限**。



2. 添加读写S3的权限策略。在**创建角色**页面的**Attach权限策略**区域，输入**S3**，选中**AmazonS3FullAccess**，单击**下一步：标签**。



3. 为IAM角色命名并完成IAM角色创建。
   i. 单击**下一步：审核**，在**创建角色**页面的**审核**区域，配置**角色名称**和**角色描述**，单击**创建角色**，完成IAM角色创建。

   ii. 返回IAM控制台，在搜索框输入**redshift_s3_role**，单击**redshift_s3_role**角色名称，获取并记录**角色ARN**。
   执行UNLOAD命令迁移数据时会使用**角色ARN**访问S3。

4. 为Redshift集群添加创建好的IAM角色，获取访问S3的权限。
   i. 登录Amazon Redshift控制台，在右上角选择区域为**亚太地区（新加坡）**。
   ii. 在左侧导航栏，单击**集群**，选中已创建好的Redshift集群，在**操作**下拉列表选择**管理IAM角色**。
   iii. 在**管理IAM角色**页面，单击搜索框右侧的▼图标，选择**redshift_s3_role**。单击**添加IAM角色 > 完成**，将具备S3访问权限的**redshift_s3_role**添加至Redshift集群。
5. 将Amazon Redshift数据导出至S3。
   i. 返回Amazon Redshift控制台。

ii. 在左侧导航栏，单击**编辑器**，执行UNLOAD命令将Amazon Redshift数据以PARQUET格式导出至S3的Bucket目录。

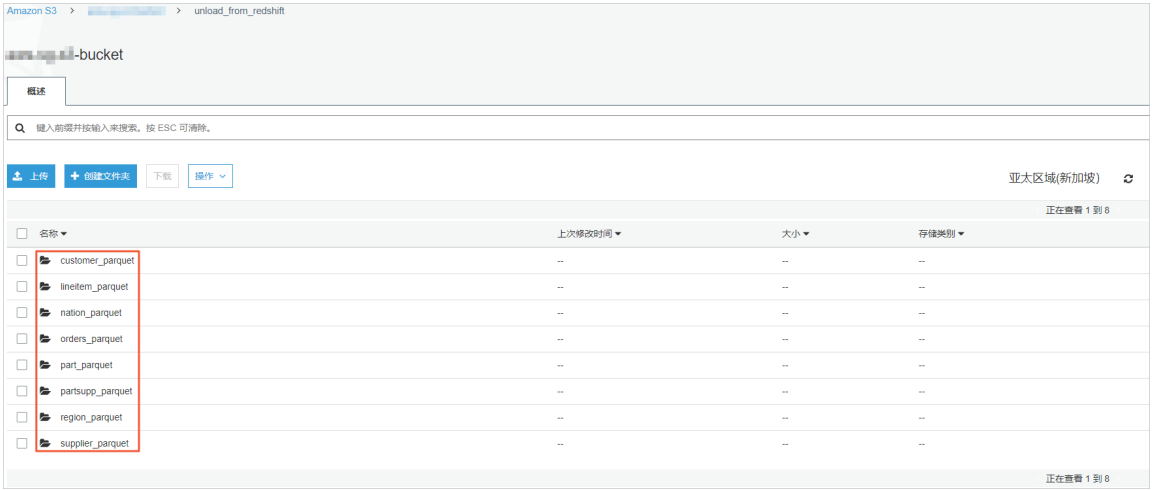
命令示例如下。

```
UNLOAD ('SELECT * FROM customer')
TO 's3://bucket_name/unload_from_redshift/customer_parquet/customer_'
FORMAT AS PARQUET
IAM_ROLE 'arn:aws:iam::xxxx:role/redshift_s3_role';
UNLOAD ('SELECT * FROM orders')
TO 's3://bucket_name/unload_from_redshift/orders_parquet/orders_'
FORMAT AS PARQUET
IAM_ROLE 'arn:aws:iam::xxxx:role/redshift_s3_role';
UNLOAD ('SELECT * FROM lineitem')
TO 's3://bucket_name/unload_from_redshift/lineitem_parquet/lineitem_'
FORMAT AS PARQUET
IAM_ROLE 'arn:aws:iam::xxxx:role/redshift_s3_role';
UNLOAD ('SELECT * FROM nation')
TO 's3://bucket_name/unload_from_redshift/nation_parquet/nation_'
FORMAT AS PARQUET
IAM_ROLE 'arn:aws:iam::xxxx:role/redshift_s3_role';
UNLOAD ('SELECT * FROM part')
TO 's3://bucket_name/unload_from_redshift/part_parquet/part_'
FORMAT AS PARQUET
IAM_ROLE 'arn:aws:iam::xxxx:role/redshift_s3_role';
UNLOAD ('SELECT * FROM partsupp')
TO 's3://bucket_name/unload_from_redshift/partsupp_parquet/partsupp_'
FORMAT AS PARQUET
IAM_ROLE 'arn:aws:iam::xxxx:role/redshift_s3_role';
UNLOAD ('SELECT * FROM region')
TO 's3://bucket_name/unload_from_redshift/region_parquet/region_'
FORMAT AS PARQUET
IAM_ROLE 'arn:aws:iam::xxxx:role/redshift_s3_role';
UNLOAD ('SELECT * FROM supplier')
TO 's3://bucket_name/unload_from_redshift/supplier_parquet/supplier_'
FORMAT AS PARQUET
IAM_ROLE 'arn:aws:iam::xxxx:role/redshift_s3_role';
```

> 说明 **编辑器**支持一次提交多条UNLOAD命令。

iii. 登录S3控制台，在S3的Bucket目录下检查导出的数据。

格式为符合预期的PARQUET格式。

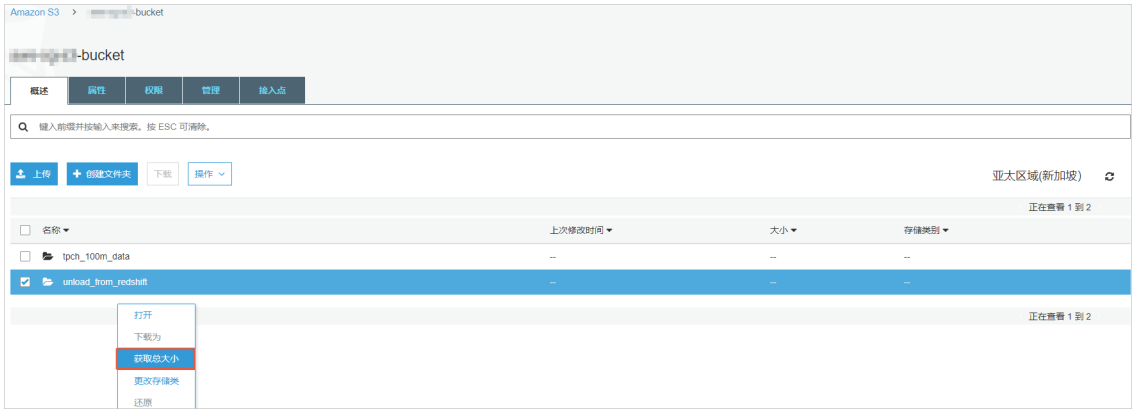


步骤二：将导出至S3的数据迁移至对象存储服务OSS

MaxCompute支持通过OSS的**在线迁移上云服务**，将S3的数据迁移至OSS，详情请参见AWS S3迁移教程。**在线迁移上云服务**处于公测状态，您需要提工单联系客服，并由在线服务团队开通后才可使用。

1. 登录OSS管理控制台，创建保存迁移数据的Bucket，详情请参见创建存储空间。

2. 创建RAM用户并授予相关权限。
   i. 登录RAM访问控制台，创建RAM用户，详情请参见创建RAM用户。
   ii. 选中新创建的用户登录名称，单击**添加权限**，为新创建的RAM用户授予**AliyunOSSFullAccess**（存储空间读写权限）和**AliyunMGWFullAccess**（在线迁移权限），单击**确定 > 完成**。
   iii. 在左侧导航栏，单击**概览**。在**概览**页面的**账号管理**区域，单击**用户登录地址**链接，使用新创建的RAM用户登录阿里云控制台。
3. 在AWS侧准备可编程及访问S3的IAM用户。
   i. 登录S3控制台。
   ii. 在导出的数据目录上单击右键，选择**获取总大小**，获取迁移目录的数据大小和文件个数。
      - 获取总大小

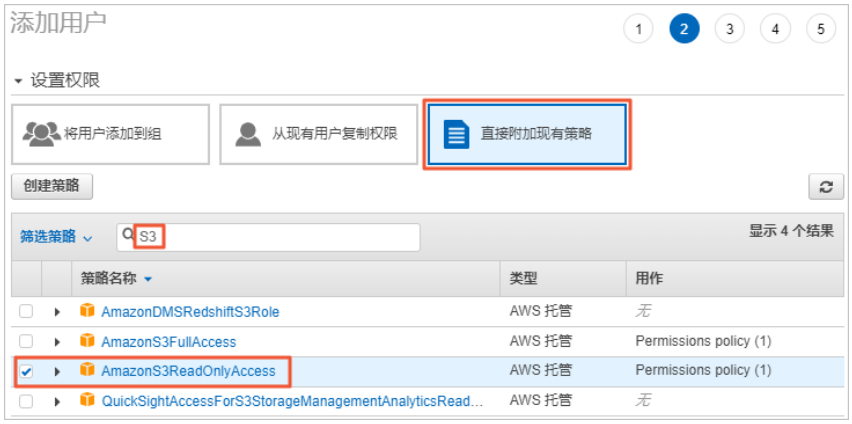


      - 获取迁移目录的数据大小和文件个数

   iii. 登录IAM用户控制台，单击**添加用户**。

iv. 在**添加用户**页面，配置**用户名**。在**选择AWS访问类型**区域，选中**编程访问**，单击**下一步：权限**。

v. 在**添加用户**页面，单击**直接附加现有策略**。在搜索框中输入S3，选中AmazonS3ReadOnlyAccess策略，单击**下一步：标签**。



vi. 单击**下一步：审核 > 创建用户**，完成IAM用户创建并记录密钥信息。

创建在线迁移任务时会使用该密钥信息。

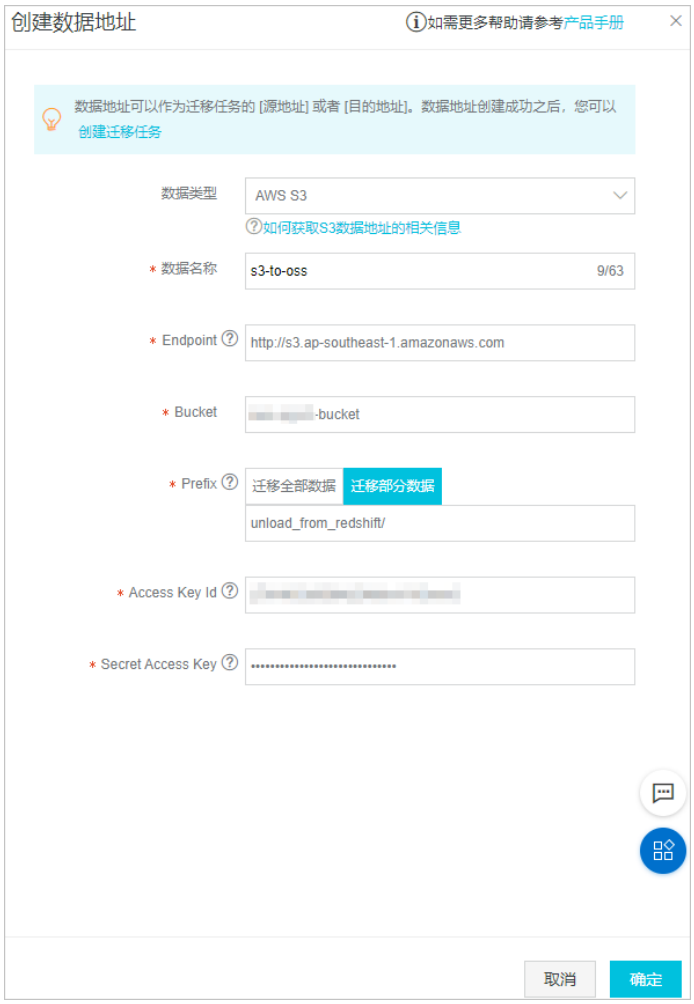
4. 创建在线迁移数据地址。

i. 登录阿里云数据在线迁移控制台。在左侧导航栏，单击**数据地址**。

ii. （可选）如果未开通在线迁移服务，请在弹出的对话框中单击**去申请**。在**在线迁移公测申请**页面，填写信息，并单击**提交**。

iii. 在**管理数据地址**页面，单击**创建数据地址**，配置数据源及目标地址相关参数，单击**确定**。参数详情请参见迁移实施。

- 数据源



> 说明 Access Key ID和Access Key Secret为IAM用户的密钥信息。

- 目标地址

> 说明 Access Key ID和Access Key Secret为RAM用户的密钥信息。

5. 创建在线迁移任务。
   i. 在左侧导航栏，单击**迁移任务**。
   ii. 在**迁移任务列表**页面，单击**创建迁移任务**，配置相关信息后，单击**创建**。参数详情请参见迁移实施。

- **任务配置**

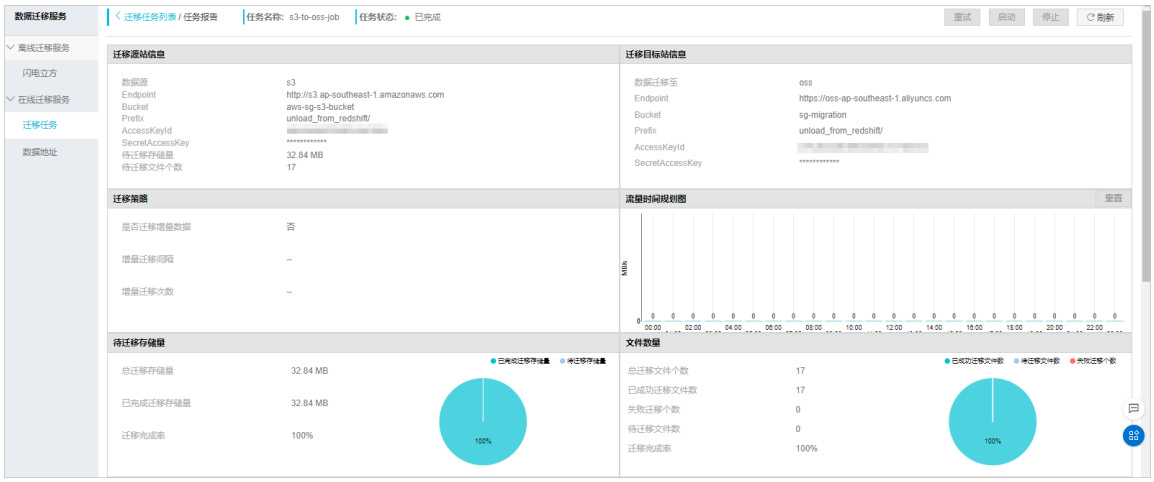

创建迁移任务
如需更多帮助请参考产品手册
任务配置
性能调优
迁移数据地址
* 任务名称
s3-to-oss-job
13/63
如果无可用数据（源/目的）地址，请您先创建数据地址
* 源地址
[s3] s3-to-oss
aws-sg-s3-bucket:unload_from_redshift/
* 目的地址
[oss] oss_data_address
https://oss-ap-southeast-1.aliyuncs.com:sg-migration1/unload_from_redshift/
迁移策略
迁移方式
全量迁移
增量迁移
数据同步
全量数据迁移完成后任务将立即停止，不再对增量数据进行迁移。同任务多次提交全量迁移，仅迁移更新的数据
多版本迁移
不使用
使用
多版本迁移会扫描您源站文件的所有版本，并全部（按顺序）迁移到目的地址。
迁移文件起点时间
迁移全部
指定时间
文件覆盖方式
最后修改时间优先
条件覆盖
全覆盖
不覆盖
对于同名文件，优先判断二者的LastModified，即最后修改时间。
1. 如果源LastModified < 目的LastModified，则此文件将被执行跳过。
2. 如果源LastModified > 目的LastModified，则执行覆盖。
3. 如果源LastModified == 目的LastModified，则继续判断：
- 若二者的Size或Content-Type有其一不相等，则执行覆盖。
- 否则（Size、Content-Type都相等），文件将被执行跳过。
取消
下一步

- **性能调优**

> **说明** **待迁移存储量**和**待迁移文件个数**是您通过S3控制台获取到的待迁移数据大小和文件个数。

iii. 创建的迁移任务会自动运行。请您确认**任务状态**为**已完成**，表示迁移任务成功结束。

iv. 在迁移任务的右侧单击**管理**，查看迁移任务报告，确认数据已经全部迁移成功。



v. 登录OSS管理控制台。

vi. 在左侧导航栏，单击**Bucket列表**。在**Bucket列表**区域单击创建的Bucket。在Bucket页面，单击**文件管理**，查看文件迁移结果。

步骤三：将数据从OSS迁移至同区域的MaxCompute项目

您可以通过MaxCompute的LOAD命令将OSS数据迁移至同区域的MaxCompute项目中。

LOAD命令支持STS认证和AccessKey认证两种方式，AccessKey认证方式需要使用明文AccessKey ID和AccessKey Secret。STS认证方式不会暴露AccessKey信息，具备高安全性。本文以STS认证方式为例介绍数据迁移操作。

1. 在DataWorks的临时查询界面或MaxCompute客户端（odpscmd），使用Redshift集群数据的DDL，创建与迁移数据相对应的表。

   临时查询功能详情请参见使用临时查询运行SQL语句（可选）。命令示例如下。

```
CREATE TABLE customer(
C_CustKey int ,
C_Name varchar(64) ,
C_Address varchar(64) ,
C_NationKey int ,
C_Phone varchar(64) ,
C_AcctBal decimal(13, 2) ,
C_MktSegment varchar(64) ,
C_Comment varchar(120) ,
skip varchar(64)
);
CREATE TABLE lineitem(
L_OrderKey int ,
L_PartKey int ,
L_SuppKey int ,
L_LineNumber int ,
L_Quantity int ,
L_ExtendedPrice decimal(13, 2) ,
L_Discount decimal(13, 2) ,
L_Tax decimal(13, 2) ,
L_ReturnFlag varchar(64) ,
L_LineStatus varchar(64) ,
L_ShipDate timestamp ,
L_CommitDate timestamp ,
L_ReceiptDate timestamp ,
```

```
L_ShipInstruct varchar(64) ,
L_ShipMode varchar(64) ,
L_Comment varchar(64) ,
skip varchar(64)
);
CREATE TABLE nation(
N_NationKey int ,
N_Name varchar(64) ,
N_RegionKey int ,
N_Comment varchar(160) ,
skip varchar(64)
);
CREATE TABLE orders(
O_OrderKey int ,
O_CustKey int ,
O_OrderStatus varchar(64) ,
O_TotalPrice decimal(13, 2) ,
O_OrderDate timestamp ,
O_OrderPriority varchar(15) ,
O_Clerk varchar(64) ,
O_ShipPriority int ,
O_Comment varchar(80) ,
skip varchar(64)
);
CREATE TABLE part(
P_PartKey int ,
P_Name varchar(64) ,
P_Mfgr varchar(64) ,
P_Brand varchar(64) ,
P_Type varchar(64) ,
P_Size int ,
P_Container varchar(64) ,
P_RetailPrice decimal(13, 2) ,
P_Comment varchar(64) ,
skip varchar(64)
);
CREATE TABLE partsupp(
PS_PartKey int ,
PS_SuppKey int ,
PS_AvailQty int ,
PS_SupplyCost decimal(13, 2) ,
PS_Comment varchar(200) ,
skip varchar(64)
);
CREATE TABLE region(
R_RegionKey int ,
R_Name varchar(64) ,
R_Comment varchar(160) ,
skip varchar(64)
);
CREATE TABLE supplier(
S_SuppKey int ,
S_Name varchar(64) ,
S_Address varchar(64) ,
S_NationKey int ,
S_Phone varchar(18) ,
S_AcctBal decimal(13, 2) ,
S_Comment varchar(105) ,
skip varchar(64)
);
```

本文的TPC-H数据集使用MaxCompute 2.0数据类型和Decimal 2.0数据类型，所以创建的项目为MaxCompute 2.0版本。如果您的项目需要设置使用2.0数据类型，请在命令前添加如下语句一起提交执行。

```
setproject odps.sql.type.system.odps2=true;
setproject odps.sql.decimal.odps2=true;
```

2. 创建具备访问OSS权限的RAM角色并授权。详情请参见STS模式授权。
3. 分次执行LOAD命令，将OSS的全部数据加载至创建的MaxCompute表中，并执行SQL命令查看和校验数据导入结果。LOAD命令详情请参见LOAD。

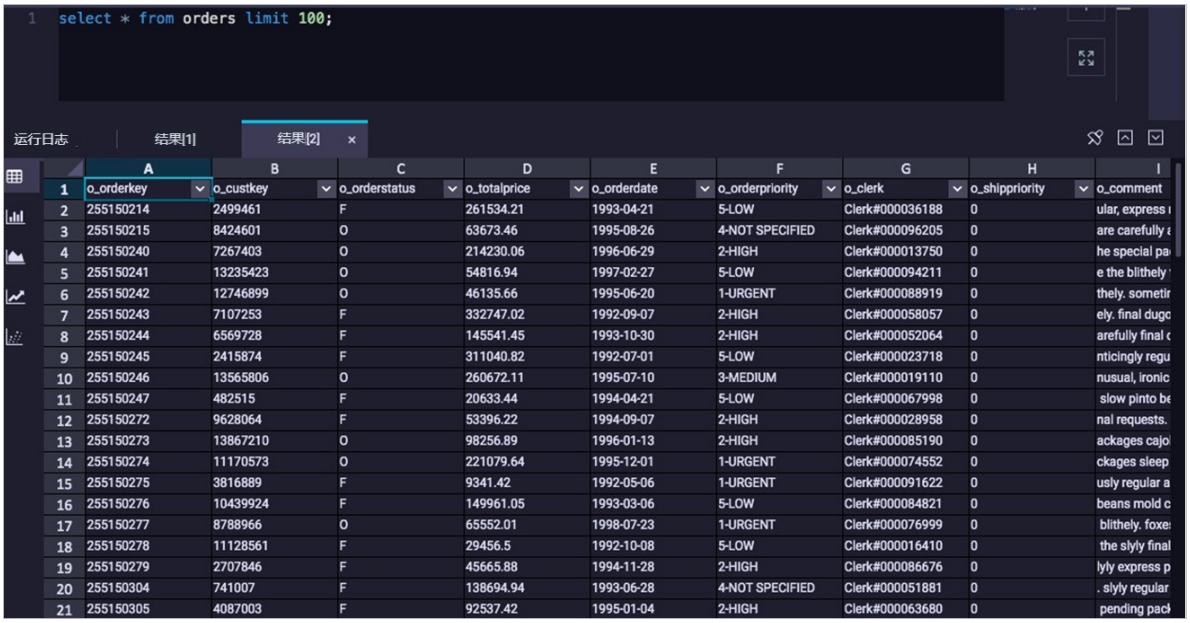
```
LOAD OVERWRITE TABLE orders
FROM  LOCATION 'oss://endpoint/oss_bucket_name/unload_from_redshift/orders_parquet/' --OSS存储空间位置。
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
WITH SERDEPROPERTIES ('odps.properties.rolearn'='acs:ram::xxx:role/xxx_role')
STORED AS PARQUET;
```

> **说明** 如果导入失败，请提工单联系MaxCompute团队处理。

执行如下命令查看和校验数据导入结果。

```
SELECT * FROM orders limit 100;
```

返回结果示例如下。



```
select * from orders limit 100;
```

运行日志 | 结果[1] | 结果[2]

	A	B	C	D	E	F	G	H	I
1	o_orderkey	o_custkey	o_orderstatus	o_totalprice	o_orderdate	o_orderpriority	o_clerk	o_shippriority	o_comment
2	255150214	2499461	F	261534.21	1993-04-21	5-LOW	Clerk#000036188	0	ular, express
3	255150215	8424601	O	63673.46	1995-08-26	4-NOT SPECIFIED	Clerk#000096205	0	are carefully
4	255150240	7267403	O	214230.06	1996-06-29	2-HIGH	Clerk#000013750	0	he special pa
5	255150241	13235423	O	54816.94	1997-02-27	5-LOW	Clerk#000094211	0	e the blithely
6	255150242	12746899	O	46135.66	1995-06-20	1-URGENT	Clerk#000088919	0	thely. sometir
7	255150243	7107253	F	332747.02	1992-09-07	2-HIGH	Clerk#000058057	0	ely. final dugc
8	255150244	6569728	F	145541.45	1993-10-30	2-HIGH	Clerk#000052064	0	arefully final
9	255150245	2415874	F	311040.82	1992-07-01	5-LOW	Clerk#000023718	0	nticingly regu
10	255150246	13565806	O	260672.11	1995-07-10	3-MEDIUM	Clerk#000019110	0	nusual, ironic
11	255150247	482515	F	20633.44	1994-04-21	5-LOW	Clerk#000067998	0	slow pinto be
12	255150272	9628064	F	53396.22	1994-09-07	2-HIGH	Clerk#000028958	0	nal requests.
13	255150273	13867210	O	98256.89	1996-01-13	2-HIGH	Clerk#000085190	0	ackages cajo
14	255150274	11170573	O	221079.64	1995-12-01	1-URGENT	Clerk#000074552	0	ckages sleep
15	255150275	3816889	F	9341.42	1992-05-06	1-URGENT	Clerk#000091622	0	usly regular a
16	255150276	10439924	F	149961.05	1993-03-06	5-LOW	Clerk#000084821	0	beans mold c
17	255150277	8788966	O	65552.01	1998-07-23	1-URGENT	Clerk#000076999	0	blithely. foxe
18	255150278	11128561	F	29456.5	1992-10-08	5-LOW	Clerk#000016410	0	the slyly final
19	255150279	2707846	F	45665.88	1994-11-28	2-HIGH	Clerk#000086676	0	lyly express p
20	255150304	741007	F	138694.94	1993-06-28	4-NOT SPECIFIED	Clerk#000051881	0	. slyly regular
21	255150305	4087003	F	92537.42	1995-01-04	2-HIGH	Clerk#000063680	0	pending pack

4. 通过表的数量、记录的数量和典型作业的查询结果，校验迁移至MaxCompute的数据是否和Redshift集群的数据一致。
   i. 登录Amazon Redshift控制台，在右上角选择区域为**亚太地区（新加坡）**。在左侧导航栏，单击**编辑器**，输入如下命令执行查询操作。

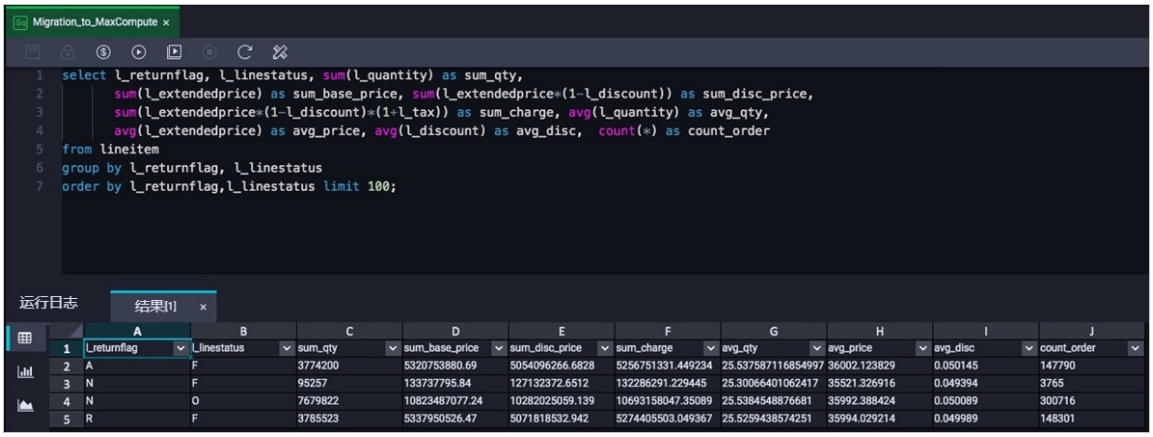
```
SELECT l_returnflag, l_linestatus, SUM(l_quantity) as sum_qty,
SUM(l_extendedprice) AS sum_base_price, SUM(l_extendedprice*(1-l_discount)) AS sum_disc_price,
SUM(l_extendedprice*(1-l_discount)*(1+l_tax)) AS sum_charge, AVG(l_quantity) AS avg_qty,
AVG(l_extendedprice) AS avg_price, AVG(l_discount) AS avg_disc,  COUNT(*) AS count_order
FROM lineitem
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag,l_linestatus;
```

返回结果示例如下。

l_returnflag	l_linestatus	sum_qty	sum_base_price	sum_disc_price	sum_charge	avg_qty
A	F	3774200	5320753880.69	5054096266.6828	5256751331.449234	25
N	F	95257	133737795.84	127132372.6512	132286291.229445	25
N	O	7679822	10823487077.24	10282025059.1390	10693158047.350890	25
R	F	3785523	5337950526.47	5071818532.9420	5274405503.049367	25

ii. 通过DataWorks的临时查询界面或MaxCompute客户端（odpscmd），执行上述命令，验证返回结果是否与Redshift集群的返回结果一致。

返回结果示例如下。



	A	B	C	D	E	F	G	H	I	J
1	l_returnflag	l_linestatus	sum_qty	sum_base_price	sum_disc_price	sum_charge	avg_qty	avg_price	avg_disc	count_order
2	A	F	3774200	5320753880.69	5054096266.6828	5256751331.449234	25.537587116854997	36002.123829	0.050145	147790
3	N	F	95257	133737795.84	127132372.6512	132286291.229445	25.30066401062417	35521.326916	0.049394	3765
4	N	O	7679822	10823487077.24	10282025059.139	10693158047.35089	25.5384548876681	35992.388424	0.050089	300716
5	R	F	3785523	5337950526.47	5071818532.942	5274405503.049367	25.5259438574251	35994.029214	0.049989	148301

2.15. BigQuery数据迁移至MaxCompute

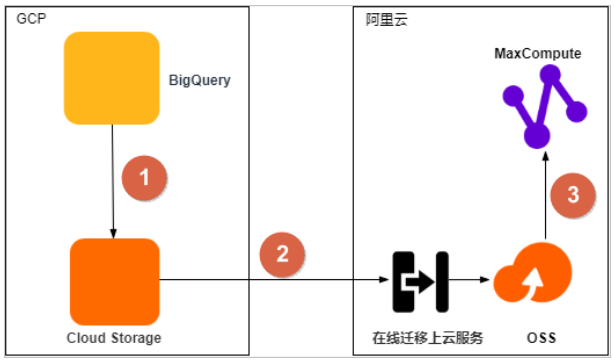
本文为您介绍如何通过公网环境将谷歌云GCP（Google Cloud Platform）的BigQuery数据集迁移至阿里云MaxCompute。

前提条件

类别	平台	要求	参考文档
环境及数据	谷歌云GCP	• 已开通谷歌BigQuery服务，并准备好环境及待迁移的数据集。 • 已开通谷歌Cloud Storage服务，并创建存储分区（Bucket）。	如果您没有相关环境及数据集，可参考如下内容准备： • BigQuery：BigQuery快速入门和创建数据集 • Cloud Storage：Cloud Storage快速入门和创建存储分区
	阿里云	• 已开通MaxCompute、DataWorks服务并创建项目空间。 以**印度尼西亚（雅加达）**区域为例，创建作为迁移目标的MaxCompute项目。 • 已开通对象存储服务OSS并创建存储分区（Bucket）。 • 已开通OSS的在线迁移上云服务。	如果您没有相关环境，可参考如下内容准备： • MaxCompute、DataWorks：准备工作和创建MaxCompute项目 • OSS：开通OSS服务和创建存储空间 • 在线迁移上云服务：提工单或在线申请
账号	谷歌云GCP	已创建具备访问谷歌Cloud Storage权限的IAM用户。	通过JSON使用IAM权限
	阿里云	已创建具备存储空间读写权限和在线迁移权限的RAM用户及RAM角色。	创建RAM用户和STS模式授权
区域	谷歌云GCP	无。	无
	阿里云	开通OSS服务的区域与MaxCompute项目在同一区域。	无

背景信息

将BigQuery数据集迁移至阿里云MaxCompute的流程如下。



序号	描述
①	将BigQuery数据集导出至谷歌Cloud Storage。

序号	描述
②	通过对象存储服务OSS的**在线迁移上云服务**，将数据从谷歌Cloud Storage迁移至OSS。
③	将数据从OSS迁移至同区域的MaxCompute项目中，并校验数据完整性和正确性。

步骤一：将BigQuery数据集导出至谷歌Cloud Storage

您可以使用bq命令行工具执行 `bq extract` 命令，将BigQuery数据集导出至谷歌Cloud Storage。

1. 登录Google Cloud控制台，创建存储迁移数据的分区（Bucket）。操作详情请参见创建存储分区。

2. 使用bq命令行工具，查询TPC-DS数据集中表的DDL脚本并下载至本地设备。操作详情请参见使用INFORMATION_SCHEMA获取表元数据。

   如果您需要了解bq命令行工具，请参见使用bq命令行工具。

   BigQuery不提供诸如 `show create table` 之类的命令来查询表的DDL脚本。BigQuery允许您使用内置的用户自定义函数UDF来查询特定数据集中表的DDL脚本。DDL脚本示例如下。

3. 通过bq命令行工具执行 `bq extract` 命令，将BigQuery数据集中的表依次导出至谷歌Cloud Storage的目标存储分区（Bucket）。相关操作及导出的数据格式和压缩类型详情请参见导出表数据。

   导出命令示例如下。

```
bq extract
--destination_format AVRO
--compression SNAPPY
tpcds_100gb.web_site
gs://bucket_name/web_site/web_site-*.avro.snappy;
```

4. 查看存储分区，检查数据导出结果。

步骤二：将导出至谷歌Cloud Storage的数据迁移至对象存储服务OSS

对象存储服务OSS支持通过**在线迁移上云服务**，将谷歌Cloud Storage的数据迁移至OSS，详情请参见谷歌云GCP迁移教程。**在线迁移上云服务**处于公测状态，您需要提工单联系客服，并由在线服务团队开通后才可使用。

1. 预估需要迁移的数据，包括迁移存储量和迁移文件个数。您可以使用gsutil工具或通过存储日志查看待迁移存储分区（Bucket）的存储量。详情请参见获取存储分区信息。
2. （可选）如果您未创建存储分区，请登录OSS管理控制台，创建保存迁移数据的分区（Bucket），详情请参见创建存储空间。

3. （可选）如果您未创建RAM用户，请创建RAM用户并授予相关权限。
   i. 登录RAM访问控制台，创建RAM用户，详情请参见创建RAM用户。

ii. 选中新创建的用户登录名称，单击**添加权限**，为新创建的RAM用户授予**AliyunOSSFullAccess**（存储空间读写权限）和**AliyunMGWFullAccess**（在线迁移权限），单击**确定 > 完成**。

iii. 在左侧导航栏，单击**概览**。在**概览**页面的**账号管理**区域，单击**用户登录地址**链接，使用新创建的RAM用户登录阿里云控制台。

4. 在GCP侧准备一个以编程方式访问谷歌Cloud Storage的用户。操作详情请参见通过JSON使用IAM权限。

i. 登录IAM用户控制台，选择一个有权限访问BigQuery的用户。在**操作**列，单击 **> 创建密钥**。

ii. 在弹出的对话框，选择**JSON**，单击**创建**。将JSON文件保存至本地设备，并单击**完成**。

iii. 在**创建服务账号**页面，单击**选择角色**，选择**Cloud Storage > Storage Admin**，授予用户访问谷歌Cloud Storage的权限。

5. 创建在线迁移数据地址。

i. 登录阿里云数据在线迁移控制台。在左侧导航栏，单击**数据地址**。

ii. （可选）如果您未开通在线迁移服务，请在弹出的对话框中单击**去申请**。在**在线迁移公测申请**页面，填写信息，并单击**提交**。

> **说明** **在线迁移公测申请**页面的**迁移数据源类型**中如果没有谷歌云GCP，请您选择其中一种数据源类型，并在**备注**中指明实际数据源类型。

iii. 在**管理数据地址**页面，单击**创建数据地址**，配置数据源及目标地址相关参数，单击**确定**。参数详情请参见迁移实施。

- 数据源

> 说明 Key File是步骤4下载的JSON文件。

- 目标地址

> 说明 Access Key ID和Access Key Secret为RAM用户的密钥信息。

6. 创建在线迁移任务。
   i. 在左侧导航栏，单击**迁移任务**。
   ii. 在**迁移任务列表**页面，单击**创建迁移任务**，配置相关信息后，单击**创建**。参数详情请参见迁移实施。

- **任务配置**

- **性能调优**

> **说明** **待迁移存储量**和**待迁移文件个数**是您通过Google Cloud控制台获取到的待迁移数据大小和文件个数。

iii. 创建的迁移任务会自动运行。请您确认**任务状态**为**已完成**，表示迁移任务成功结束。

iv. 在迁移任务的右侧单击**管理**，查看迁移任务报告，确认数据已经全部迁移成功。

v. 登录OSS管理控制台。

vi. 在左侧导航栏，单击**Bucket列表**。在**Bucket列表**区域单击创建的Bucket。在Bucket页面，单击**文件管理**，查看数据迁移结果。

步骤三：将数据从OSS迁移至同区域的MaxCompute项目

您可以通过MaxCompute的LOAD命令将OSS数据迁移至同区域的MaxCompute项目中。

LOAD命令支持STS认证和AccessKey认证两种方式，AccessKey认证方式需要使用明文AccessKey ID和AccessKey Secret。STS认证方式不会暴露AccessKey信息，具备高安全性。本文以STS认证方式为例介绍数据迁移操作。

1. 在DataWorks的临时查询界面或MaxCompute客户端（odpscmd），修改已获取到的BigQuery数据集中表的DDL，适配MaxCompute数据类型，创建与迁移数据相对应的表。

   临时查询功能详情请参见使用临时查询运行SQL语句（可选）。命令示例如下。

```
CREATE OR REPLACE TABLE
`****.tpcds_100gb.web_site`
(
  web_site_sk INT64,
  web_site_id STRING,
  web_rec_start_date STRING,
  web_rec_end_date STRING,
  web_name STRING,
  web_open_date_sk INT64,
  web_close_date_sk INT64,
  web_class STRING,
  web_manager STRING,
  web_mkt_id INT64,
  web_mkt_class STRING,
  web_mkt_desc STRING,
  web_market_manager STRING,
  web_company_id INT64,
  web_company_name STRING,
  web_street_number STRING,
  web_street_name STRING,
  web_street_type STRING,
  web_suite_number STRING,
  web_city STRING,
  web_county STRING,
  web_state STRING,
  web_zip STRING,
  web_country STRING,
  web_gmt_offset FLOAT64,
  web_tax_percentage FLOAT64
)
Modify the INT64 and
FLOAT64 fields to obtain the following DDL script:
CREATE
TABLE IF NOT EXISTS <your_maxcompute_project>.web_site_load
(
web_site_sk BIGINT,
web_site_id STRING,
web_rec_start_date STRING,
web_rec_end_date STRING,
web_name STRING,
web_open_date_sk BIGINT,
web_close_date_sk BIGINT,
web_class STRING,
web_manager STRING,
web_mkt_id BIGINT,
web_mkt_class STRING,
web_mkt_desc STRING,
web_market_manager STRING,
web_company_id BIGINT,
web_company_name STRING,
web_street_number STRING,
web_street_name STRING,`
web_street_type STRING,
web_suite_number STRING,
web_city STRING,
web_county STRING,
web_state STRING,
web_zip STRING,
web_country STRING,
web_gmt_offset DOUBLE,
web_tax_percentage DOUBLE
);
```

BigQuery和MaxCompute数据类型的对应关系如下。

BigQuery数据类型	MaxCompute数据类型
INT64	BIGINT
FLOAT64	DOUBLE
NUMERIC	DECIMAL、DOUBLE
BOOL	BOOLEAN

BigQuery数据类型	MaxCompute数据类型
STRING	STRING
BYTES	VARCHAR
DATE	DATE
DATETIME	DATETIME
TIME	DATETIME
TIMESTAMP	TIMESTAMP
STRUCT	STRUCT
GEOGRAPHY	STRING

2. （可选）如果您未创建RAM角色，请创建具备访问OSS权限的RAM角色并授权。详情请参见STS模式授权。
3. 执行LOAD命令，将OSS的全部数据加载至创建的MaxCompute表中，并执行SQL命令查看和校验数据导入结果。LOAD命令一次只能加载一张表，有多个表时，需要执行多次。LOAD命令详情请参见LOAD。

```
LOAD OVERWRITE TABLE web_site
FROM  LOCATION 'oss://oss-<your_region_id>-internal.aliyuncs.com/bucket_name/tpc_ds_100gb/web_site/' --OSS存储空间位置。
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
WITH SERDEPROPERTIES ('odps.properties.rolearn'='<Your_RoleARN>','mcfed.parquet.compression'='SNAPPY')
STORED AS AVRO;
```

> **说明** 如果导入失败，请提工单联系MaxCompute团队处理。

执行如下命令查看和校验数据导入结果。

```
SELECT * FROM web_site;
```

返回结果示例如下。

	web_site_sk	web_site_id	web_rec_start_date	web_rec_end_date	web_name	web_open_date_sk	web_close_date_sk	web_class	web_manager	web_mkt_id	web_mkt_class	web_mkt_desc	web_market_manager	web_company_id	web_company_na
1	16	AAAAA...	1997-08-16	1999-08-16	site_2	2450679	2447029	Unkno...	Herbert H...	6	Hard new ...	Basic othe...	Albert Leung	6	cally
2	17	AAAAA...	1999-08-17	2001-08-15	site_2	2450679	2447029	Unkno...	Charles C...	5	New, differ...	Basic othe...	Keith Frazier	6	cally
3	18	AAAAA...	2001-08-16	\N	site_2	2450679	2447029	Unkno...	Charles C...	5	Broad, ne...	Basic othe...	Keith Frazier	6	cally
4	1	AAAAA...	1997-08-16	\N	site_0	2450807	\N	Unkno...	Ronald Sh...	4	Grey lines ...	Well simila...	Joe George	6	cally
5	2	AAAAA...	1997-08-16	2000-08-15	site_0	2450798	2443498	Unkno...	Tommy J...	6	Completely...	Lucky pas...	David Myers	4	ese
6	3	AAAAA...	2000-08-16	\N	site_0	2450798	2443498	Unkno...	Tommy J...	3	Completely...	Particular, ...	David Myers	4	ese
7	19	AAAAA...	1997-08-16	\N	site_3	2450654	\N	Unkno...	Jimmy Pope	2	Leaders m...	Home new...	Eldon Snow	5	anti
8	4	AAAAA...	1997-08-16	1999-08-16	site_0	2450781	2447131	Unkno...	Harold Wi...	5	As strong ...	Deeply sm...	James Harris	5	anti
9	5	AAAAA...	1999-08-17	2001-08-15	site_0	2450781	2447131	Unkno...	Harold Wi...	5	Wide, final ...	Deeply sm...	Edward George	1	ought
10	6	AAAAA...	2001-08-16	\N	site_0	2450781	2447131	Unkno...	Harold Wi...	5	Wide, final ...	Thin roles ...	John Sheppard	3	ought
11	20	AAAAA...	1997-08-16	2000-08-15	site_3	2450645	2446995	Unkno...	Peter Cas...	4	Just popul...	Formerly l...	Kelvin Lynch	3	pri
12	21	AAAAA...	2000-08-16	\N	site_3	2450645	2446995	Unkno...	Adam Sto...	5	As private ...	Formerly l...	Jarvis Allen	4	pri
13	22	AAAAA...	1997-08-16	1999-08-16	site_3	2450628	2443328	Unkno...	Lewis Wolf	5	Severe, us...	Individual ...	James Bernard	2	able
14	23	AAAAA...	1999-08-17	2001-08-15	site_3	2450628	2443328	Unkno...	Jason Silva	4	Severe, us...	Individual ...	Jeffrey Martin	2	able
15	24	AAAAA...	2001-08-16	\N	site_3	2450628	2443328	Unkno...	John Ward	4	Severe, us...	Individual ...	Philip Sampson	2	able
16	7	AAAAA...	1997-08-16	\N	site_1	2450756	\N	Unkno...	Moses Hi...	4	New home...	Enough s...	William Reyes	4	ese
17	8	AAAAA...	1997-08-16	2000-08-15	site_1	2450747	2447097	Unkno...	Charles P...	1	Only, temp...	Physical wi...	Gerald Craft	5	anti
18	9	AAAAA...	2000-08-16	\N	site_1	2450747	2447097	Unkno...	Marshall ...	6	Labour car...	Physical wi...	Scott Bryant	1	anti
19	10	AAAAA...	1997-08-16	1999-08-16	site_1	2450730	2443430	Unkno...	Dwight A...	3	New, impo...	Companie...	Frank Cooper	2	able
20	11	AAAAA...	1999-08-17	2001-08-15	site_1	2450730	2443430	Unkno...	Dwight A...	3	Right effor...	Companie...	Casey Banks	2	able
21	12	AAAAA...	2001-08-16	\N	site_1	2450730	2443430	Unkno...	Richard F...	3	Right effor...	Acres see ...	Casey Banks	2	able
22	13	AAAAA...	1997-08-16	\N	site_2	2450705	\N	Unkno...	John Tho...	2	About rura...	Political af...	James Brewer	5	anti
23	14	AAAAA...	1997-08-16	2000-08-15	site_2	2450696	2443396	Unkno...	Robert Ar...	5	Necessary,...	Associatio...	Gilbert Chapman	5	anti
24	15	AAAAA...	2000-08-16	\N	site_2	2450696	2443396	Unkno...	James Au...	5	Necessary,...	Associatio...	Zachery Oneil	5	anti

4. 通过表的数量、记录的数量和典型作业的查询结果，校验迁移至MaxCompute的数据是否和BigQuery的数据一致。

2.16. 日志数据迁移至MaxCompute

2.16.1. 概述

日常工作中，企业通常会对实时日志数据进行开发。其中：日志数据来源可以为ECS、容器、移动端、开源软件、网站服务或JavaScript。本文为您介绍如何通过Tunnel、DataHub、LogHub以及DataWorks数据集成将日志数据迁移至MaxCompute。

方案	说明	适用场景
Tunnel	通过MaxCompute的Tunnel功能，将日志数据上传至MaxCompute。 详情请参见通过Tunnel迁移日志数据至MaxCompute。	Tunnel主要用于批量上传数据至离线表，适用于离线计算的场景。

方案	说明	适用场景
DataHub	DataHub数据迁移功能通过Connector实现。DataHub Connector可以将DataHub中的流式数据同步至MaxCompute。您只需要向DataHub中写入数据，并在DataHub中配置同步功能，便可以在MaxCompute中使用这些数据。 详情请参见通过DataHub迁移日志数据至MaxCompute。	此方法多用于公测和自研。DataHub用于实时上传数据，主要适用于流式计算场景。 数据上传后会保存到实时表，后续会在几分钟内通过定时任务的形式同步到MaxCompute离线表，供离线计算使用。
LogHub	日志服务SLS提供数据投递功能，将LogHub实时采集的日志投递至MaxCompute。 详情请参见通过LogHub迁移日志数据至MaxCompute。	此方法适用于海量数据场景，对数据量不设上限，同时支持行、列和TEXTFILE等各种存储格式以及用户自定义分区配置。
DataWorks数据集成	通过DataWorks配置离线同步节点和同步任务将日志数据同步至MaxCompute。 详情请参见通过DataWorks数据集成迁移日志数据至MaxCompute。	此方法为定时任务，配置一次可以多次执行同步任务。

2.16.2. 通过Tunnel迁移日志数据至MaxCompute

本文为您介绍如何通过Tunnel上传日志数据至MaxCompute。

前提条件

- 安装MaxCompute客户端，详情请参见安装并配置MaxCompute客户端。
- 将日志数据保存至本地。本文使用的示例数据为loghub.csv。

背景信息

Tunnel是MaxCompute的批量上传数据工具，适用于离线计算场景。Tunnel详细信息请参见Tunnel使用说明。

操作步骤

1. 在MaxCompute客户端（odpscmd）执行如下命令创建表*loghub*，用于存储上传的日志数据。

```
--打开新类型数据开关，此命令需要和SQL语句一起提交。
set odps.sql.type.system.odps2=true;
--创建表loghub。
CREATE TABLE loghub
(
client_ip    STRING ,
receive_time STRING ,
topic STRING,
id STRING,
name VARCHAR(32),
salenum STRING
);
```

2. 执行如下命令将日志数据上传至MaxCompute。

```
Tunnel u D:\loghub.csv loghub;
```

上述命令中需要指定如下两个参数：

- D:\loghub.csv：本地日志数据文件存储路径。
- loghub：MaxCompute中存储日志数据的表名。

> **说明** 使用Tunnel数据不支持通配符或正则表达式。如果您想使用通配符或正则表达式上传数据，详情请参见通过LogHub迁移日志数据至MaxCompute。

3. 执行如下命令查询数据是否成功导入至表中。

```
SELECT * FROM loghub;
```

返回结果如下，表示导入成功。

2.16.3. 通过DataHub迁移日志数据至MaxCompute

本文为您介绍如何通过DataHub迁移日志数据至MaxCompute。

前提条件

授权访问MaxCompute的账号已开通以下权限：

- MaxCompute中项目的CreateInstance权限。
- MaxCompute中表的查看、修改和更新权限。

授权操作详情请参见权限列表。

背景信息

DataHub是阿里云流式数据（Streaming Data）的处理平台。数据上传至DataHub后保存在实时表里，后续会在5分钟内通过定时任务的形式同步到MaxCompute离线表里，供离线计算使用。

您只需要创建DataHub Connector，指定相关配置，即可创建将DataHub中流式数据定期归档到MaxCompute的同步任务。

操作步骤

1. 在MaxCompute客户端（odpscmd）执行如下语句创建表，用于存储DataHub同步的日志数据。示例语句如下。

```
CREATE TABLE test(f1 string, f2 string, f3 double) partitioned by (ds string);
```

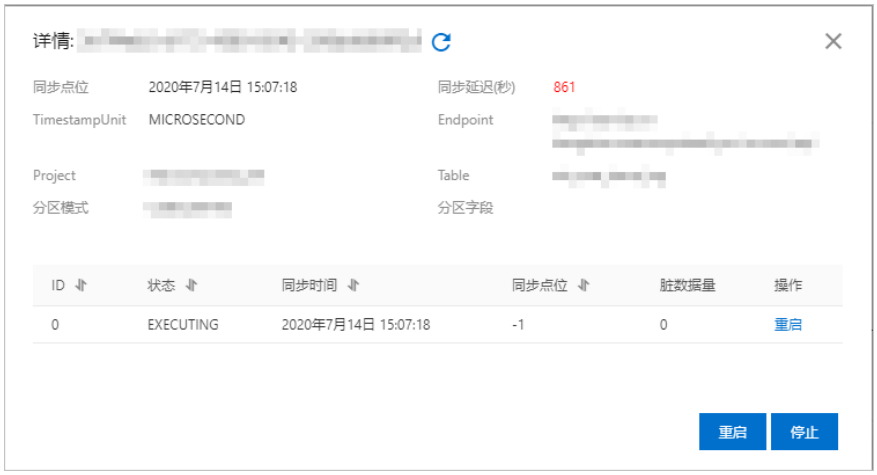
2. 在DataHub上创建项目。
   i. 登录DataHub控制台，在左上角选择地域。
   ii. 在左侧导航栏，单击**项目管理**。
   iii. 在**项目列表**区域，单击右上角**新建项目**。
   iv. 在**新建项目**页面，填写**名称**及**描述**，单击**创建**。
3. 创建Topic。
   i. 在**项目列表**区域，单击项目右侧的**查看**。
   ii. 在项目详情页面，单击右上角**新建Topic**，进入**新建Topic**页面，填写配置参数。

iii. 单击**下一步**，完善Topic信息。

> 说明
> - Schema与MaxCompute表为对应关系，DataHub Topic Schema的字段名称、类型和顺序必须与MaxCompute表字段完全一致，如果三个条件中的任意一个不满足，则Connector创建失败。
> - 支持将TUPLE和BLOB类型的DataHub Topic同步到MaxCompute表中。
> - 系统默认最多可以创建20个Topic。如果需要创建更多，请提交工单申请。
> - DataHub Topic的Owner或Creator账号有操作Connector的权限，包括创建和删除等。

4. 向新建的Topic中写入数据。
   i. 单击新建Topic右侧的**查看**。
   ii. 在Topic详情页面，单击右上角**同步**。
   iii. 在**新建Connector**页面，单击**MaxCompute**，配置相关参数后，单击**创建**。
5. 查看Connector详细信息。
   i. 在左侧导航栏，单击**项目管理**。
   ii. 在**项目列表**区域，单击项目右侧的**查看**。
   iii. 在**Topic列表**区域，单击Topic右侧的**查看**。
   iv. 在Topic详情页面，单击**Connector**页签，查看创建好的Connector。
   v. 单击Connector右侧的**查看**，即可查看Connector详细信息。

   DataHub默认每五分钟或当数据达到60 MB时强行向MaxCompute离线表中投递一次数据。**同步点位**代表已经同步数据的条数。



6. 执行如下语句测试日志数据是否投递成功。

```
SELECT * FROM test;
```

返回结果如下，表示投递成功。

2.16.4. 通过LogHub迁移日志数据至MaxCompute

本文为您介绍如何通过LogHub的直接投递功能迁移日志数据至MaxCompute。

前提条件

- 执行本操作的账号为主账号。
- 已开通日志服务LogHub，详情请参见开通日志服务。

背景信息

日志服务LogHub提供将实时采集的日志数据投递至MaxCompute的功能。如果启用该功能，日志服务后台会定时把写入到该日志库内的日志数据投递到MaxCompute对应的表中，方便您对数据进行后续加工。

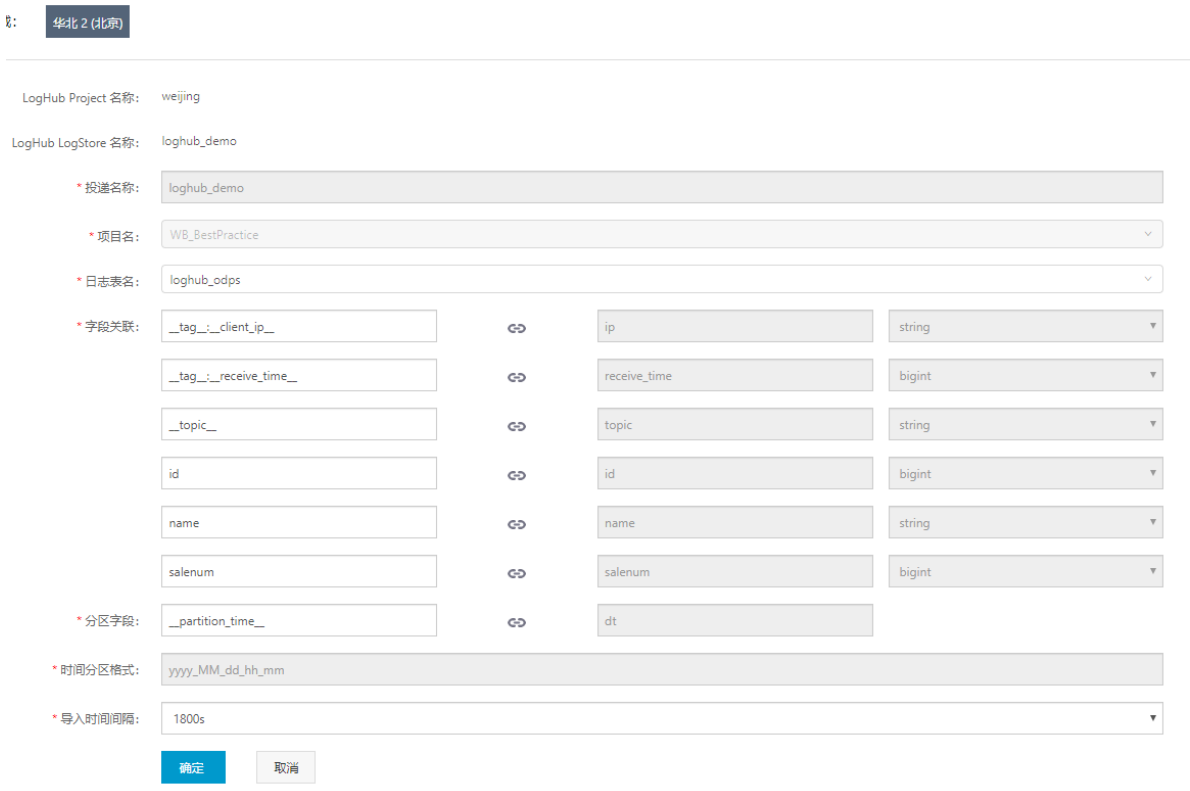
操作步骤

1. 在MaxCompute客户端（odpscmd）执行如下语句创建表loghub，用于接收日志服务投递数据。

```
CREATE TABLE loghub
(
id       BIGINT,
name     STRING,
salenum  BIGINT
) PARTITIONED BY (ds string);
```

2. 在LogHub中创建Project和Logstore。
   i. 登录日志服务控制台，在**Project列表**区域，单击右上角**创建Project**，填写配置信息创建新的Project。配置参数说明请参见创建Project。
   ii. 单击已经创建的Project，进入日志管理页面。
   iii. 单击搜索文本框后的**+**新建Logstore，并填写配置参数。配置参数说明请参见创建Logstore。

   iv. 单击Logstore的名称，即可查看Logstore的日志数据。
3. 选择Logstore下的**数据处理 > 导出 > MaxCompute（原ODPS）**。
4. 在**MaxCompute（原ODPS）投递管理**页面，单击右上角**开启投递**，并在确认框中单击**直接投递**。
5. 在**LogHub —— 数据投递**页面配置相关参数。



配置参数说明如下。

参数	语义
投递名称	自定义一个投递的名称，方便后续管理。

参数	语义
项目名	MaxCompute表所在的项目名称，该项默认为新创建的Project，对于已经创建的Project，可以在下拉框中选择。
日志表名	MaxCompute表名称，请输入自定义新建的MaxCompute表名称或者选择已有的MaxCompute表。 **说明** MaxCompute表至少包含一个数据列和一个分区列。
字段关联	按顺序左边填写与MaxCompute表数据列映射的日志服务字段名称，右边填写或者选择MaxCompute表的普通字段名称及字段类型。 **说明** 日志服务数据的一个字段最多允许映射到一个MaxCompute表的列（数据列或分区列），不支持字段冗余，同一个字段名第二次使用时其投递的值为null，如果null出现在分区列会导致数据无法被投递。
分区字段	按顺序左边填写与MaxCompute表分区列映射的日志服务字段名称，右边填写或者选择MaxCompute表的普通字段名称及字段类型。 **注意** • 投递MaxCompute是批量任务，需谨慎设置分区列及其类型。保证一个同步任务内处理的数据分区数小于512个。用作分区列的字段值不能为空或包括正斜线（/）等MaxCompute保留字段。 • MaxCompute单表有分区数目6万的限制，分区数超出后无法再写入数据，所以日志服务导入MaxCompute表至多支持3个分区列。需谨慎选择自定义字段作为分区列，保证其值可枚举。
分区时间格式	`__partition_time__` 输出的日期格式。例如 `yyyy-MM-dd` 。
导入时间间隔	MaxCompute数据投递间隔，默认1800，单位：秒（s）。

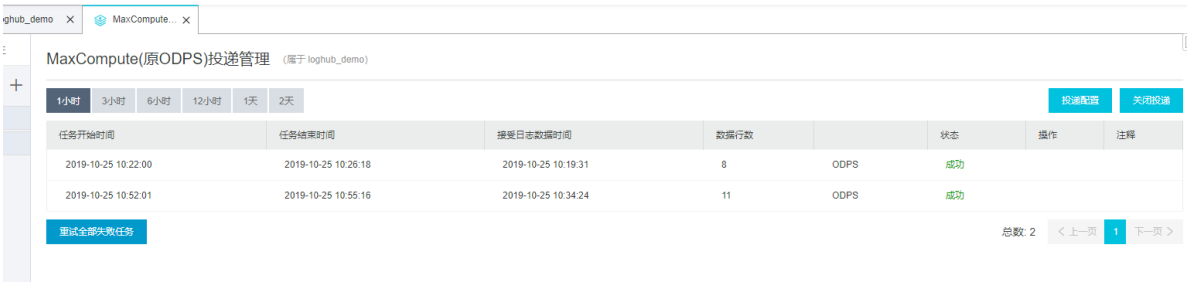
说明

- 不同Logstore的数据不能导入到同一个MaxCompute表中，否则会造成分区冲突或丢失数据。
- 海外Region不支持通过LogHub将数据投递至MaxCompute，可以使用DataWorks进行数据投递。详情请参见通过DataWorks数据集成迁移日志数据至MaxCompute。

6. 配置完成后，单击**确定**，启动投递。

   启动投递功能后，日志服务后台会定期启动离线投递任务。

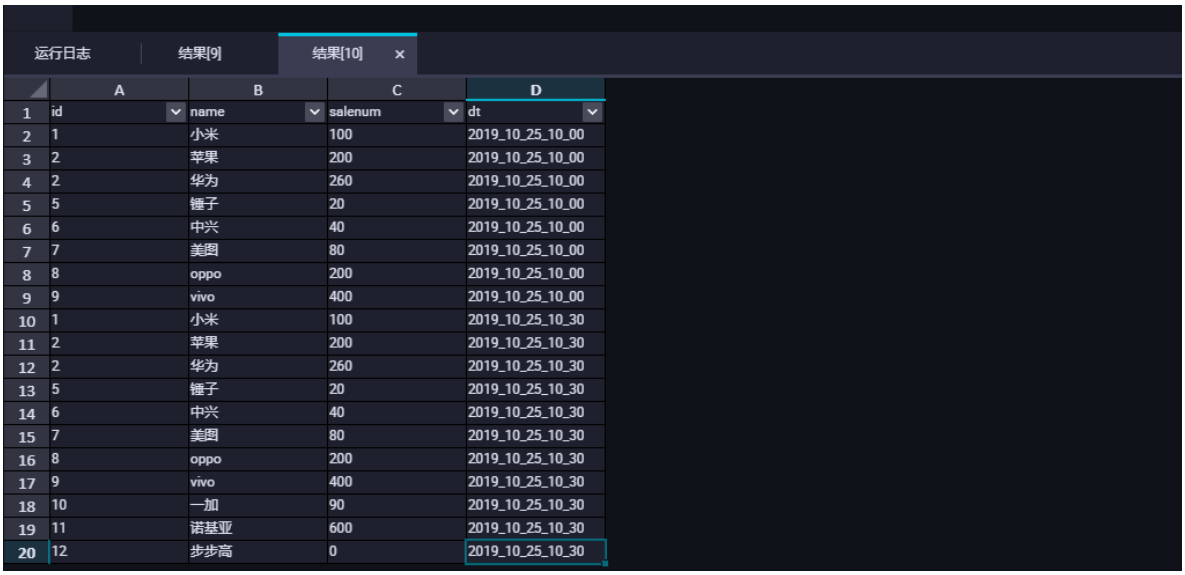
7. 在**提示**对话框，单击**确定**，返回日志服务控制台查看投递运行状态。



8. 执行如下语句在MaxCompute中查询表数据，检查日志投递结果。

```
SELECT id,name,salenum,dt FROM loghub WHERE substr(dt,1,18) = '2019_10_25';
```

返回结果如下，说明投递成功。


运行日志 结果[9] 结果[10]

	A	B	C	D
1	id	name	salenum	dt
2	1	小米	100	2019_10_25_10_00
3	2	苹果	200	2019_10_25_10_00
4	2	华为	260	2019_10_25_10_00
5	5	锤子	20	2019_10_25_10_00
6	6	中兴	40	2019_10_25_10_00
7	7	美图	80	2019_10_25_10_00
8	8	oppo	200	2019_10_25_10_00
9	9	vivo	400	2019_10_25_10_00
10	1	小米	100	2019_10_25_10_30
11	2	苹果	200	2019_10_25_10_30
12	2	华为	260	2019_10_25_10_30
13	5	锤子	20	2019_10_25_10_30
14	6	中兴	40	2019_10_25_10_30
15	7	美图	80	2019_10_25_10_30
16	8	oppo	200	2019_10_25_10_30
17	9	vivo	400	2019_10_25_10_30
18	10	一加	90	2019_10_25_10_30
19	11	诺基亚	600	2019_10_25_10_30
20	12	步步高	0	2019_10_25_10_30

2.16.5. 通过DataWorks数据集成迁移日志数据至MaxCompute

本文为您介绍如何通过数据集成功能同步LogHub数据至MaxCompute。

背景信息

日志服务支持以下数据同步场景：

- 跨地域的LogHub与MaxCompute等数据源的数据同步。
- 不同阿里云账号下的LogHub与MaxCompute等数据源间的数据同步。
- 同一阿里云账号下的LogHub与MaxCompute等数据源间的数据同步。
- 公共云与金融云账号下的LogHub与MaxCompute等数据源间的数据同步。

以B账号进入数据集成配置同步任务，将A账号的LogHub数据同步至B账号的MaxCompute为例，跨阿里云账号的特别说明如下：

1. 使用A账号的AccessKey ID和AccessKey Secret创建LogHub数据源。

   此时B账号可以同步A账号下所有日志服务项目的数据。

2. 使用A账号下的RAM用户A1的AccessKey ID和AccessKey Secret创建LogHub数据源。
   - A账号为RAM用户A1赋予日志服务的通用权限，即 `AliyunLogFullAccess` 和 `AliyunLogReadOnlyAccess` ，详情请参见创建RAM用户及授权。
   - A账号给RAM用户A1赋予日志服务的自定义权限。

     主账号A进入**RAM控制台 > 权限管理 > 权限策略管理**页面，单击**新建授权策略**。

     相关的授权请参见访问控制RAM和RAM子用户访问。

     根据下述策略进行授权后，B账号通过RAM用户A1只能同步日志服务project_name1以及project_name2的数据。

```
{
"Version": "1",
"Statement": [
{
"Action": [
"log:Get*",
"log:List*",
"log:CreateConsumerGroup",
"log:UpdateConsumerGroup",
"log:DeleteConsumerGroup",
"log:ListConsumerGroup",
"log:ConsumerGroupUpdateCheckPoint",
"log:ConsumerGroupHeartBeat",
"log:GetConsumerGroupCheckPoint"
],
"Resource": [
"acs:log:*:*:project/project_name1",
"acs:log:*:*:project/project_name1/*",
"acs:log:*:*:project/project_name2",
"acs:log:*:*:project/project_name2/*"
],
"Effect": "Allow"
}
]
}
```

新建LogHub数据源

1. 登录DataWorks控制台，单击相应工作空间后的**进入数据集成**。
2. 单击左侧导航栏中的**数据源**，即可跳转至**工作空间管理 > 数据源管理**页面。
3. 在**数据源管理**页面，单击右上角的**新增数据源**。
4. 在**新增数据源**对话框中，选择数据源类型为**LogHub**。
5. 填写**新增LogHub数据源**对话框中的配置。

参数	描述
数据源名称	数据源名称必须以字母、数字、下划线组合，且不能以数字和下划线开头。
数据源描述	对数据源进行简单描述，不得超过80个字符。
LogHub Endpoint	LogHub的Endpoint，格式为 `http://example.com` 。详情请参见服务入口。
Project	输入项目名称。
AccessKey ID	访问密钥中的AccessKey ID，您可以进入控制台的**用户信息管理**页面进行复制。
AccessKey Secret	访问密钥中的AccessKey Secret，相当于登录密码。

6. 单击**测试连通性**。
7. 连通性测试通过后，单击**完成**。

新建离线同步节点

1. 在**数据源**页面，单击左上角的图标，选择**全部产品 > DataStudio（数据开发）**。
2. 在**数据开发**页面，鼠标悬停至图标，单击**业务流程**。
3. 在**新建业务流程**对话框中，输入**业务流程名称**和**描述**，单击**新建**。
4. 展开业务流程，右键单击**数据集成**，选择**新建 > 离线同步**。
5. 在**新建节点**对话框中，输入**节点名称**，并选择**目标文件夹**。
6. 单击**提交**，进入离线节点编辑页面。

通过向导模式配置同步任务

1. 在离线节点编辑页面，选择数据来源。

参数	描述
数据源	输入LogHub数据源的名称。
Logstore	导出增量数据的表的名称。该表需要开启Stream，可以在建表时开启，或者使用UpdateTable接口开启。
日志开始时间	数据消费的开始时间位点，为时间范围（左闭右开）的左边界，为yyyyMMddHHmmss格式的时间字符串（例如20180111013000）。该参数可以和DataWorks的调度时间参数配合使用。
日志结束时间	数据消费的结束时间位点，为时间范围（左闭右开）的右边界，为yyyyMMddHHmmss格式的时间字符串（例如20180111013010）。该参数可以和DataWorks的调度时间参数配合使用。
批量条数	一次读取的数据条数，默认为256。

> **说明** 您可以进行数据预览，此处仅选择LogHub中的几条数据展现在预览框。由于您在进行同步任务时，会指定开始时间和结束时间，会导致预览结果和实际的同步结果不一致。

2. 选择MaxCompute数据源及目标表。
3. 选择字段的映射关系。
4. 在**通道控制**中配置作业速率上限和脏数据检查规则。
5. 确认当前节点的配置无误后，单击左上角的**保存**。
6. 运行离线同步节点。

   您可以通过以下两种方式运行离线同步节点：

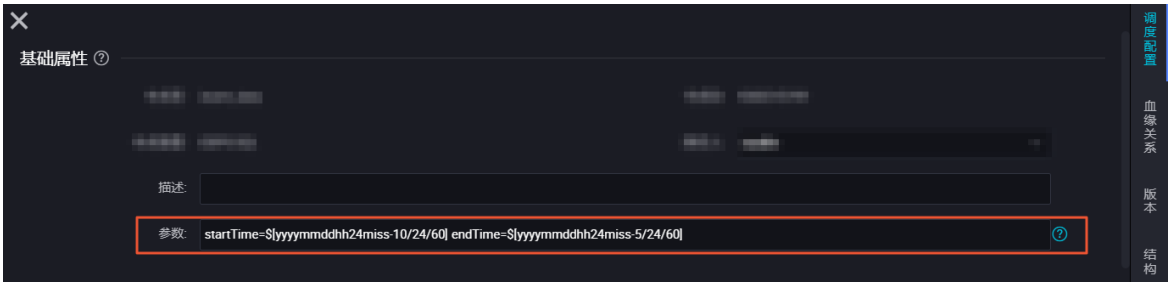
   - 直接运行（一次性运行）

     单击节点编辑页面工具栏中的**运行**图标，直接在页面运行。

     > **说明** 运行之前需要配置自定义参数的具体取值。

   - 调度运行

     单击节点编辑页面工具栏中的**提交**图标，提交离线同步节点至调度系统，调度系统会根据配置的属性，从第2天开始自动定时运行。



如上图所示，设置开始时间为系统前10分钟，结束时间为系统前5分钟：startTime=$[yyyymmddhh24miss-10/24/60]
endTime=$[yyyymmddhh24miss-5/24/60]。

如上图所示，设置离线同步节点的调度周期为分钟，从00:00~23:59每5分钟调度一次。

通过脚本模式配置离线同步节点

1. 成功创建离线同步节点后，单击工具栏中的**转换脚本**。

2. 单击**提示**对话框中的**确认**，即可进入脚本模式进行开发。
3. 单击工具栏中的**导入模板**。

4. 在**导入模板**对话框中，选择从来源端的LogHub数据源同步至目标端的ODPS数据源的导入模板，单击**确认**。
5. 导入模板后，根据自身需求编辑代码，示例脚本如下。

```
{
"type": "job",
"version": "1.0",
"configuration": {
"reader": {
"plugin": "loghub",
"parameter": {
"datasource": "loghub_lzz",//数据源名，需要和您添加的数据源名一致。
"logstore": "logstore-ut2",//目标日志库的名字，LogStore是日志服务中日志数据的采集、存储和查询单元。
"beginDateTime": "${startTime}",//数据消费的开始时间位点，为时间范围（左闭右开）的左边界。
"endDateTime": "${endTime}",//数据消费的结束时间位点，为时间范围（左闭右开）的右边界。
"batchSize": 256,//一次读取的数据条数，默认为256。
"splitPk": "",
"column": [
"key1",
"key2",
"key3"
]
}
},
"writer": {
"plugin": "odps",
"parameter": {
"datasource": "odps_first",//数据源名，需要和您添加的数据源名一致。
"table": "ok",//目标表名。
"truncate": true,
"partition": "",//分区信息。
"column": [//目标列名。
"key1",
"key2",
"key3"
]
}
},
"setting": {
"speed": {
"mbps": 8,//作业速率上限。
"concurrent": 7//并发数。
}
}
}
}
```

3.数据开发

3.1. 构造测试数据

当您需要研究某类型数据的SQL处理方法，或验证功能实现逻辑是否符合预期，或需要在某些场合演示功能时，可以通过构造测试数据支撑功能验证及演示。本文为您介绍构造数据的方法，仅供参考。

背景信息

通常，先有数据才会有基于数据的应用，但当需要将应用运用到其他场景时，没有数据无法进行功能验证或演示。

企业的真实数据经过脱敏后也可以应用于功能验证及演示，但是数据脱敏操作很复杂，脱敏过重会导致数据无法使用，脱敏过轻会出现数据泄露风险，最安全的方式就是构造测试数据。在构造测试数据前，您需要梳理清楚业务场景及对数据的特殊要求，例如表的记录数、列字段类型、列字段的枚举值等。

构造测试数据的业务场景包含如下两种：

- 单一业务场景：例如咨询或验证SQL处理数据的方法，只需要构造单张表并通过 `insert … values` 、 `values table` 、 `select ... from values` 或 `union all` 操作插入少量数据即可。更多构造小表信息，请参见构造小表。更多语法信息，请参见VALUES。
- 复杂业务场景：需要构造一个业务系统，相对于单一业务场景，存在多个表，且表之间存在关联关系。该场景下构造数据分为两个阶段：
  i. 构造维度表：通过笛卡尔积构造大表。更多构造大表信息，请参见构造大表。
  ii. 构造事实表：基于维度表构造事实表，此阶段实现复杂，本文不展开赘述，如果您想深入了解，请填写钉钉群申请表单加入钉钉群进行咨询。

构造小表

在MaxCompute上创建表后，构造少量测试数据的方法如下：

- （推荐）方法一：通过 `insert … values` 操作向表中插入数据。命令示例如下。

```
--创建分区表srcp。
create table if not exists srcp (key string,value bigint) partitioned by (p string);
--向分区表srcp添加分区。
alter table srcp add if not exists partition (p='abc');
--向表srcp的指定分区abc中插入数据。
insert into table srcp partition (p='abc') values ('a',1),('b',2),('c',3);
--查询表srcp。
select * from srcp where p='abc';
--返回结果如下。
+------------+------------+------------+
| key        | value      | p          |
+------------+------------+------------+
| a          | 1          | abc        |
| b          | 2          | abc        |
| c          | 3          | abc        |
+------------+------------+------------+
```

- （推荐）方法二：通过 `values table` 操作向表中插入数据。命令示例如下。

```
-创建分区表srcp。
create table if not exists srcp (key string,value bigint) partitioned by (p string);
--向表srcp中插入数据。values (...), (...) t(a, b)相当于定义了一个名为t，列为a和b，数据类型分别为STRING和BIGINT的表。列的类型需要从values列表中推导。
insert into table srcp partition (p) select concat(a,b), length(a)+length(b),'20170102' from values ('d',4),('e',5),('f',6) t(a,b);
--查询表srcp。
select * from srcp where p='20170102';
--返回结果如下。
+------------+------------+------------+
| key        | value      | p          |
+------------+------------+------------+
| d4         | 2          | 20170102   |
| e5         | 2          | 20170102   |
| f6         | 2          | 20170102   |
+------------+------------+------------+
```

- 方法三：通过 `from values` 或 `union all` 操作构造表并插入数据。命令示例如下。

```
with t as (select 1 c union all select 2 c) select * from t;
--等价于如下语句。
select * from values (1), (2) t(c);
--返回结果如下。
+------------+
| c          |
+------------+
| 1          |
| 2          |
+------------+
```

构造大表

您需要先构造小表，然后通过 `mapJoin` 对小表使用笛卡尔积方式基于随机值或有序值构造大量数据。

数据种类从本质上可以分为如下两种：

- 序列值：有序的数列，使用窗口函数函数构造，该类型数据可以定义为主键。
- 枚举值：少数的一些代码值，例如数值、金额，这些枚举值主要使用随机函数RAND构造。

命令示例如下。

```
--构造一张小表，插入序列值。
create table za1 as
select c0 from values
 (1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),(12),(13),(14),(15)
,(16),(17),(18),(19),(20),(21),(22),(23),(24),(25),(26),(27),(28),(29),(30)
,(31),(32),(33),(34),(35),(36),(37),(38),(39),(40),(41),(42),(43),(44),(45)
,(46),(47),(48),(49),(50),(51),(52),(53),(54),(55),(56),(57),(58),(59),(60)
,(61),(62),(63)
t(c0);
--查看za1表中构造的数据。
select * from za1;
+------+
| c0   |
+------+
| 1    |
| 2    |
| 3    |
| 4    |
| 5    |
| 6    |
| 7    |
| 8    |
| 9    |
| 10   |
| 11   |
| 12   |
| 13   |
| 14   |
| 15   |
| 16   |
| 17   |
| 18   |
| 19   |
| 20   |
| 21   |
| 22   |
| 23   |
| 24   |
| 25   |
| 26   |
| 27   |
| 28   |
| 29   |
| 30   |
| 31   |
| 32   |
| 33   |
| 34   |
| 35   |
| 36   |
| 37   |
| 38   |
| 39   |
| 40   |
```

```
| 41   |
| 42   |
| 43   |
| 44   |
| 45   |
| 46   |
| 47   |
| 48   |
| 49   |
| 50   |
| 51   |
| 52   |
| 53   |
| 54   |
| 55   |
| 56   |
| 57   |
| 58   |
| 59   |
| 60   |
| 61   |
| 62   |
| 63   |
+------+
--您也可以通过SELECT TRANSFORM插入序列值。
create table za1 as select transform('for i in `seq 1 63`; do echo $i; done') using 'sh' as (data);
--使用ROW_NUMBER函数生成有序值构造大表。生成的数据是一个有序的从1000000-1039689的序列。
create table zb1 as
select *
  from(
select /*+mapjoin(t2,t3)*/
       1000000 + row_number() over(partition by 1)-1 as c0
  from za1 t1
  join za1 t2
  join(select c0 from za1 limit 10)t3
)t;
--查看zb1表中构造的数据。
select * from zb1;
+------+
| c0   |
+------+
| 1000000    |
| 1000001    |
| 1000002    |
| 1000003    |
|   ...      |
| 1039688    |
| 1039689    |
+------+
--使用RAND函数生成随机值构造大表。c2列生成的数据是相对均匀的1~1000的值。
create table zb2 as
select *
  from(
select /*+mapjoin(t2,t3)*/
       1000000 + row_number() over(partition by 1)-1 as c0
      ,1617120000 as c1
      ,cast(round(rand()*999,0) as bigint)+1 as c2
  from za1 t1
  join za1 t2
  join(select c0 from za1 limit 10)t3
  )t;
--查看zb2表中构造的数据。
select * from zb2;
+------------+------+------------+
| c0         | c1   | c2         |
+------------+------+------------+
| 1000000    | 1617120000 | 1          |
| 1000001    | 1617120000 | 800        |
| 1000002    | 1617120000 | 835        |
| 1000003    | 1617120000 | 108        |
| ...        |  ...       |  ...       |
| 1039687    | 1617120000 | 4          |
| 1039688    | 1617120000 | 577        |
| 1039689    | 1617120000 | 33         |
+------------+------+------------+
```

当您有特殊数据要求，例如文本型枚举值或日期值时，可参考如下命令构造数据。

```
--构造枚举值（文本）。
with za as (
select * from values
 (1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),(12),(13),(14),(15)
,(16),(17),(18),(19),(20),(21),(22),(23),(24),(25),(26),(27),(28),(29),(30)
,(31),(32),(33),(34),(35),(36),(37),(38),(39),(40),(41),(42),(43),(44),(45)
,(46),(47),(48),(49),(50),(51),(52),(53),(54),(55),(56),(57),(58),(59),(60)
,(61),(62),(63)
t(c0)
)
,ta as (
select * from values ('zhangsan',4),('lisi',5),('wangmazi',6) t(a,b))
select k,a,b,c
  from(
select 100 + row_number() over(partition by 1)-1 as k
      ,cast(round(rand()*3,0) as bigint)+3 as c
  from za
 limit 5
)tb join ta on ta.b=tb.c;
--返回结果如下。
+------------+---+------+------------+
| k          | a | b    | c          |
+------------+---+------+------------+
| 101        | lisi | 5    | 5       |
| 102        | wangmazi | 6    | 6   |
| 104        | lisi | 5    | 5       |
+------------+---+------+------------+
--构造日期时间。
with za as (
select * from values
 (1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),(12),(13),(14),(15)
,(16),(17),(18),(19),(20),(21),(22),(23),(24),(25),(26),(27),(28),(29),(30)
,(31),(32),(33),(34),(35),(36),(37),(38),(39),(40),(41),(42),(43),(44),(45)
,(46),(47),(48),(49),(50),(51),(52),(53),(54),(55),(56),(57),(58),(59),(60)
,(61),(62),(63)
t(c0)
)
select k
,from_unixtime(1617120000) as t
,from_unixtime(1617120000
+3600000 * c )
 as b
,c
  from(
select 100 + row_number() over(partition by 1)-1 as k
      ,cast(round(rand()*3,0) as bigint)+3 as c
  from za
 limit 3
)tb;
--返回结果如下。
+------------+------------+------------+------------+
| k          | t          | b          | c          |
+------------+------------+------------+------------+
| 100        | 2021-03-31 00:00:00 | 2021-08-03 00:00:00 | 3          |
| 101        | 2021-03-31 00:00:00 | 2021-10-25 08:00:00 | 5          |
| 102        | 2021-03-31 00:00:00 | 2021-12-06 00:00:00 | 6          |
+------------+------------+------------+------------+
```

参考文档

更多关于构造数据的详细信息，请参见MaxCompute造数据详解。

3.2. 日期数据格式转换：STRING、TIMESTAMP、DATETIME互相转换

本文为您介绍STRING、TIMESTAMP与DATETIME类型数据间的转换方法，帮助您在实际业务处理过程中，快速对标找到合适的日期转换方法，提升业务处理效率。

日期数据格式转换常见场景如下：

STRING转换为TIMESTAMP

- 应用场景

  将STRING类型数据转换为TIMESTAMP类型（格式为 `yyyy-mm-dd hh:mi:ss.ff3` ）的日期值。

- 实现方法

  使用CAST函数进行转换。

- 使用限制

  输入的STRING类型数据的格式至少要满足 `yyyy-mm-dd hh:mi:ss` 要求。

- 使用示例
  - 示例1：使用CAST函数，将STRING类型数据 `2009-07-01 16:09:00` 转换为TIMESTAMP类型。命令示例如下。

    ```
    --返回2009-07-01 16:09:00.000。
    select cast('2009-07-01 16:09:00' as timestamp);
    ```

  - 示例2：**错误**使用CAST函数的命令示例如下。

    ```
    --返回NULL。输入数据格式不满足要求。至少要包含yyyy-mm-dd hh:mi:ss格式。
    select cast('2009-07-01' as timestamp);
    ```

STRING转换为DATETIME

- 应用场景

  将STRING类型数据转换为DATETIME类型（格式为 `yyyy-mm-dd hh:mi:ss` ）的日期值。

- 实现方法
  - 方法一：使用CAST函数进行转换。
  - 方法二：使用TO_DATE函数进行转换。
- 使用限制
  - 使用CAST函数时，输入的STRING类型数据的格式必须要满足 `yyyy-mm-dd hh:mi:ss` 要求。
  - 使用TO_DATE函数时，需要指定format参数的取值为 `yyyy-mm-dd hh:mi:ss` 。
- 使用示例
  - 示例1：使用CAST函数，将STRING类型数据 `2009-07-01 16:09:00` 转换为DATETIME类型。命令示例如下。

    ```
    --返回2009-07-01 16:09:00。
    select cast('2009-07-01 16:09:00' as datetime);
    ```

  - 示例2：使用TO_DATE函数，指定format参数，将STRING类型数据 `2009-07-01 16:09:00` 转换为DATETIME类型。命令示例如下。

    ```
    --返回2009-07-01 16:09:00。
    select to_date('2009-07-01 16:09:00','yyyy-mm-dd hh:mi:ss');
    ```

  - 示例3：**错误**使用CAST函数的命令示例如下。

    ```
    --返回NULL。输入数据格式不满足要求。必须为yyyy-mm-dd hh:mi:ss格式。
    select cast('2009-07-01' as datetime);
    ```

  - 示例4：**错误**使用TO_DATE函数的命令示例如下。

    ```
    --返回NULL。输入数据格式不满足要求。必须为yyyy-mm-dd hh:mi:ss格式。
    select to_date('2009-07-01','yyyy-mm-dd hh:mi:ss');
    ```

TIMESTAMP转换为STRING

- 应用场景

  将TIMESTAMP类型（格式为 `yyyy-mm-dd hh:mi:ss.ff3` ）的日期值转换为STRING类型。

- 实现方法
  - 方法一：使用CAST函数进行转换。
  - 方法二：使用TO_CHAR函数按照format参数指定的格式进行转换。

- 使用示例
  - 示例1：使用CAST函数，将TIMESTAMP类型数据 `2009-07-01 16:09:00` 转换为STRING类型。为构造TIMESTAMP类型数据，总共需要使用2次CAST函数。命令示例如下。

```
--返回2009-07-01 16:09:00。
select cast(cast('2009-07-01 16:09:00' as timestamp) as string);
```

  - 示例2：使用TO_CHAR函数，将TIMESTAMP类型数据 `2009-07-01 16:09:00` 转换为STRING类型。为构造TIMESTAMP类型数据，需要使用到1次CAST函数。命令示例如下。

```
--返回2009-07-01 16:09:00。
select to_char(cast('2009-07-01 16:09:00' as timestamp),'yyyy-mm-dd hh:mi:ss');
```

TIMESTAMP转换为DATETIME

- 应用场景

  将TIMESTAMP类型（格式为 `yyyy-mm-dd hh:mi:ss.ff3` ）的日期值转换为DATETIME类型（格式为 `yyyy-mm-dd hh:mi:ss` ）的日期值。

- 实现方法
  - 方法一：使用CAST函数进行转换。
  - 方法二：使用TO_DATE函数进行转换。
- 使用限制

  使用TO_DATE函数时，需要指定format参数的取值为 `yyyy-mm-dd hh:mi:ss` 。

- 使用示例
  - 示例1：使用CAST函数，将TIMESTAMP类型数据 `2009-07-01 16:09:00` 转换为DATETIME类型。为构造TIMESTAMP类型数据，总共需要使用2次CAST函数。命令示例如下。

```
--返回2009-07-01 16:09:00。
select cast(cast('2009-07-01 16:09:00' as timestamp) as datetime);
```

  - 示例2：使用TO_DATE函数，指定format参数，将TIMESTAMP类型数据 `2009-07-01 16:09:00` 转换为DATETIME类型。为构造TIMESTAMP类型数据，需要使用到1次CAST函数。命令示例如下。

```
--返回2009-07-01 16:09:00。
select to_date(cast('2009-07-01 16:09:00' as timestamp),'yyyy-mm-dd hh:mi:ss');
```

DATETIME转换为TIMESTAMP

- 应用场景

  将DATETIME类型（格式为 `yyyy-mm-dd hh:mi:ss` ）的日期值转换为TIMESTAMP类型（格式为 `yyyy-mm-dd hh:mi:ss.ff3` ）的日期值。

- 实现方法

  使用CAST函数进行转换。

- 使用示例

  使用CAST函数，将DATETIME类型的日期值转换为TIMESTAMP类型。为构造DATETIME类型数据，需要使用到1次GETDATE函数。命令示例如下。

```
--返回2021-10-14 10:21:47.939。
select cast(getdate() as timestamp);
```

DATETIME转换为STRING

- 应用场景

  将DATETIME类型（格式为 `yyyy-mm-dd hh:mi:ss` ）的日期值转换为STRING类型。

- 实现方法
  - 方法一：使用CAST函数进行转换。
  - 方法二：使用TO_CHAR函数按照format参数指定的格式进行转换。
- 使用示例
  - 示例1：使用CAST函数，将DATETIME类型的日期值转换为STRING类型。为构造DATETIME类型数据，需要使用到1次GETDATE函数。命令示例如下。

```
--返回2021-10-14 10:21:47。
select cast(getdate() as string);
```

- 示例2：使用TO_CHAR函数，将DATETIME类型的日期值转换为指定格式的STRING类型。为构造DATETIME类型数据，需要使用到1次GETDATE函数。命令示例如下。

```
--返回2021-10-14 10:21:47。
select to_char (getdate(),'yyyy-mm-dd hh:mi:ss');
--返回2021-10-14。
select to_char (getdate(),'yyyy-mm-dd');
--返回2021。
select to_char (getdate(),'yyyy');
```

3.3. 基于MaxCompute UDF将IPv4或IPv6地址转换为归属地

随着大数据平台发展，现已可以处理多类型的非结构化、半结构化数据，其中将IP地址转换为归属地是常见的一种场景。本文为您介绍如何通过MaxCompute UDF实现将IPv4或IPv6地址转换为归属地。

前提条件

请确认已满足如下条件：

- 已安装MaxCompute客户端。

  更多安装并配置MaxCompute客户端信息，请参见安装并配置MaxCompute客户端。

- 已安装MaxCompute Studio、已连接MaxCompute项目、已创建MaxCompute Java Module。

  更多操作信息，请参见安装MaxCompute Studio、管理项目连接和创建MaxCompute Java Module。

背景信息

要实现将IPv4或IPv6地址转换为归属地，必须要有IP地址库，您需要下载IP地址库文件并以资源形式上传至MaxCompute项目。开发MaxCompute UDF，并基于IP地址库文件注册函数，从而在SQL语句中调用函数将IP地址转换为归属地。

注意事项

本文提供的IP地址库文件，仅供验证该最佳实践使用，请您结合实际业务情况，自行维护IP地址库文件。

操作流程

基于MaxCompute UDF将IPv4或IPv6地址转换为归属地的操作流程如下：

1. 步骤一：上传IP地址库文件

   将IP地址库文件作为资源上传至MaxCompute项目，后续创建的MaxCompute UDF会依赖此资源。

2. 步骤二：编写MaxCompute UDF

   在IntelliJ IDEA上编写MaxCompute UDF代码。

3. 步骤三：注册MaxCompute UDF

   将MaxCompute UDF注册为函数。

4. 步骤四：调用MaxCompute UDF转换IP地址为归属地

   在SQL语句中调用注册好的函数将IP地址转换为归属地。

步骤一：上传IP地址库文件

1. 下载IP地址库文件至本地，解压得到ipv4.txt和ipv6.txt，并放置于MaxCompute客户端的安装目录 `...\odpscmd_public\bin` 下。

   本文提供的IP地址库文件，仅供验证该最佳实践使用，请您结合实际业务情况，自行维护IP地址库文件。

2. 登录MaxCompute客户端，进入目标MaxCompute项目。

3. 执行 `add file` 命令，将ipv4.txt和ipv6.txt以File类型资源上传至MaxCompute项目。

   命令示例如下。

   ```
   add file ipv4.txt -f;
   add file ipv6.txt -f;
   ```

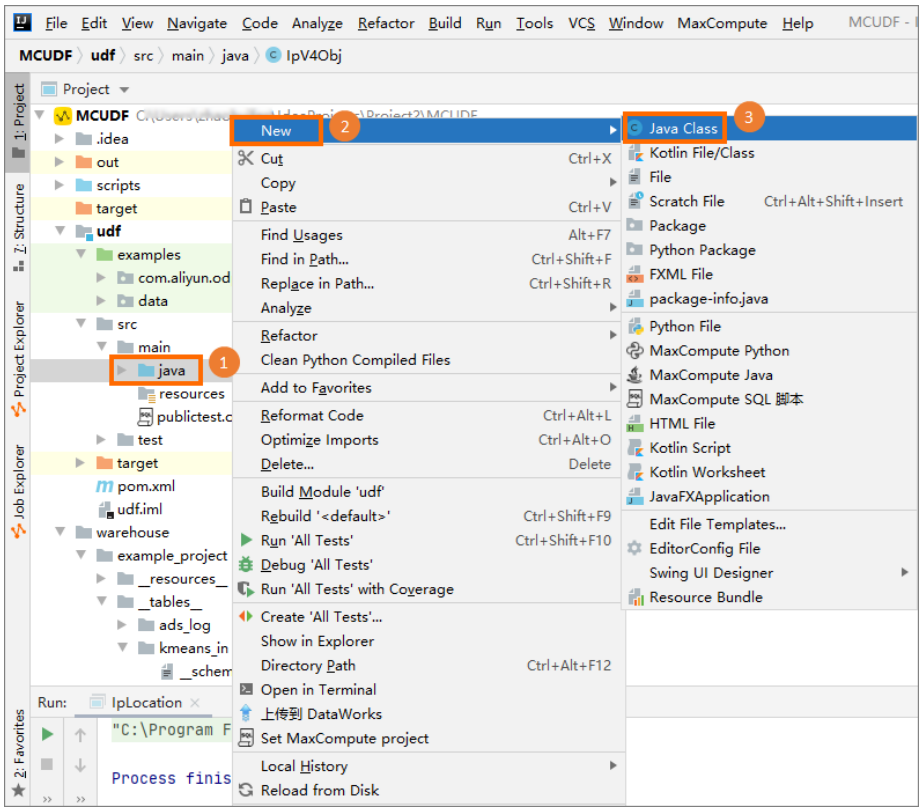
   更多添加资源信息，请参见添加资源。

步骤二：编写MaxCompute UDF

1. 创建Java Class对象。

   后续步骤中编写的MaxCompute UDF代码会用到此处创建的Java Class。

i. 进入IntelliJ IDEA界面，在**Project**区域，右键单击Module的源码目录（即**src > main > java**），选择**new > Java Class**。



ii. 在**New Java Class**对话框，输入Class名称，按下Enter键并在代码编辑区域输入代码。

您需要依次创建3个Java Class对象，Class名称及对应代码如下，代码可直接复制使用，无需修改。

- **IpUtils**

```
package com.aliyun.odps.udf.utils;
import java.math.BigInteger;
import java.net.Inet4Address;
import java.net.Inet6Address;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Arrays;
public class IpUtils {
    /**
     * 将字符串形式的ip地址转换为long
     *
     * @param ipInString
     * 字符串形式的ip地址
     * @return 返回long形式的ip地址
     */
    public static long StringToLong(String ipInString) {
        ipInString = ipInString.replace(" ", "");
        byte[] bytes;
        if (ipInString.contains(":"))
            bytes = ipv6ToBytes(ipInString);
        else
            bytes = ipv4ToBytes(ipInString);
        BigInteger bigInt = new BigInteger(bytes);
//        System.out.println(bigInt.toString());
        return bigInt.longValue();
    }
    /**
     * 将字符串形式的ip地址转换为long
     *
     * @param ipInString
     * 字符串形式的ip地址
     * @return bigint的string形式的ip地址
     */
    public static String StringToBigIntString(String ipInString) {
        ipInString = ipInString.replace(" ", "");
```

```
        byte[] bytes;
        if (ipInString.contains(":"))
            bytes = ipv6ToBytes(ipInString);
        else
            bytes = ipv4ToBytes(ipInString);
        BigInteger bigInt = new BigInteger(bytes);
        return bigInt.toString();
    }
    /**
     * 将整数形式的ip地址转换为字符串形式
     *
     * @param ipInBigInt
     * 整数形式的ip地址
     * @return 字符串形式的ip地址
     */
    public static String BigIntToString(BigInteger ipInBigInt) {
        byte[] bytes = ipInBigInt.toByteArray();
        byte[] unsignedBytes = Arrays.copyOfRange(bytes, 1, bytes.length);
        // 去除符号位
        try {
            String ip = InetAddress.getByAddress(unsignedBytes).toString();
            return ip.substring(ip.indexOf('/') + 1).trim();
        } catch (UnknownHostException e) {
            throw new RuntimeException(e);
        }
    }
    /**
     * ipv6地址转有符号byte[17]
     */
    private static byte[] ipv6ToBytes(String ipv6) {
        byte[] ret = new byte[17];
        ret[0] = 0;
        int ib = 16;
        boolean comFlag = false;// ipv4混合模式标记
        if (ipv6.startsWith(":"))// 去掉开头的冒号
            ipv6 = ipv6.substring(1);
        String groups[] = ipv6.split(":");
        for (int ig = groups.length - 1; ig > -1; ig--) {// 反向扫描
            if (groups[ig].contains(".")) {
                // 出现ipv4混合模式
                byte[] temp = ipv4ToBytes(groups[ig]);
                ret[ib--] = temp[4];
                ret[ib--] = temp[3];
                ret[ib--] = temp[2];
                ret[ib--] = temp[1];
                comFlag = true;
            } else if ("".equals(groups[ig])) {
                // 出现零长度压缩，计算缺少的组数
                int zlg = 9 - (groups.length + (comFlag ? 1 : 0));
                while (zlg-- > 0) {// 将这些组置0
                    ret[ib--] = 0;
                    ret[ib--] = 0;
                }
            } else {
                int temp = Integer.parseInt(groups[ig], 16);
                ret[ib--] = (byte) temp;
                ret[ib--] = (byte) (temp >> 8);
            }
        }
        return ret;
    }
    /**
     * IPv4地址转有符号byte[5]
     */
    private static byte[] ipv4ToBytes(String ipv4) {
        byte[] ret = new byte[5];
        ret[0] = 0;
        // 先找到ip地址字符串中.的位置
        int position1 = ipv4.indexOf(".");
        int position2 = ipv4.indexOf(".", position1 + 1);
        int position3 = ipv4.indexOf(".", position2 + 1);
        // 将每个.之间的字符串转换成整型
        ret[1] = (byte) Integer.parseInt(ipv4.substring(0, position1));
        ret[2] = (byte) Integer.parseInt(ipv4.substring(position1 + 1,
                position2));
        ret[3] = (byte) Integer.parseInt(ipv4.substring(position2 + 1
```

```
            ret[3] = (byte) Integer.parseInt(ipv4.substring(position2 + 1,
                    position3));
            ret[4] = (byte) Integer.parseInt(ipv4.substring(position3 + 1));
            return ret;
        }
        /**
         * @param ipAdress ipv4或ipv6字符串
         * @return 4:ipv4, 6:ipv6, 0:地址不对
         * @throws Exception
         */
        public static int isIpV4OrV6(String ipAdress) throws Exception {
            InetAddress address = InetAddress.getByName(ipAdress);
            if (address instanceof Inet4Address)
                return 4;
            else if (address instanceof Inet6Address)
                return 6;
            return 0;
        }
        /*
         * 验证ip是否属于某个IP段
         *
         * ipSection ip段（以'-'分隔）
         *
         * ip 所验证的ip号码
         */
        public static boolean ipExistsInRange(String ip, String ipSection) {
            ipSection = ipSection.trim();
            ip = ip.trim();
            int idx = ipSection.indexOf('-');
            String beginIP = ipSection.substring(0, idx);
            String endIP = ipSection.substring(idx + 1);
            return getIp2long(beginIP) <= getIp2long(ip)
                    && getIp2long(ip) <= getIp2long(endIP);
        }
        public static long getIp2long(String ip) {
            ip = ip.trim();
            String[] ips = ip.split("\\.");
            long ip2long = 0L;
            for (int i = 0; i < 4; ++i) {
                ip2long = ip2long << 8 | Integer.parseInt(ips[i]);
            }
            return ip2long;
        }
        public static long getIp2long2(String ip) {
            ip = ip.trim();
            String[] ips = ip.split("\\.");
            long ip1 = Integer.parseInt(ips[0]);
            long ip2 = Integer.parseInt(ips[1]);
            long ip3 = Integer.parseInt(ips[2]);
            long ip4 = Integer.parseInt(ips[3]);
            long ip2long = 1L * ip1 * 256 * 256 * 256 + ip2 * 256 * 256 + ip3 * 256
                    + ip4;
            return ip2long;
        }
        public static void main(String[] args) {
            System.out.println(StringToLong("2002:7af3:f3be:ffff:ffff:ffff:ffff:ffff"));
            System.out.println(StringToLong("54.38.72.63"));
        }
        private class Invalid{
            private Invalid()
            {
            }
        }
}
```

- IpV4Obj

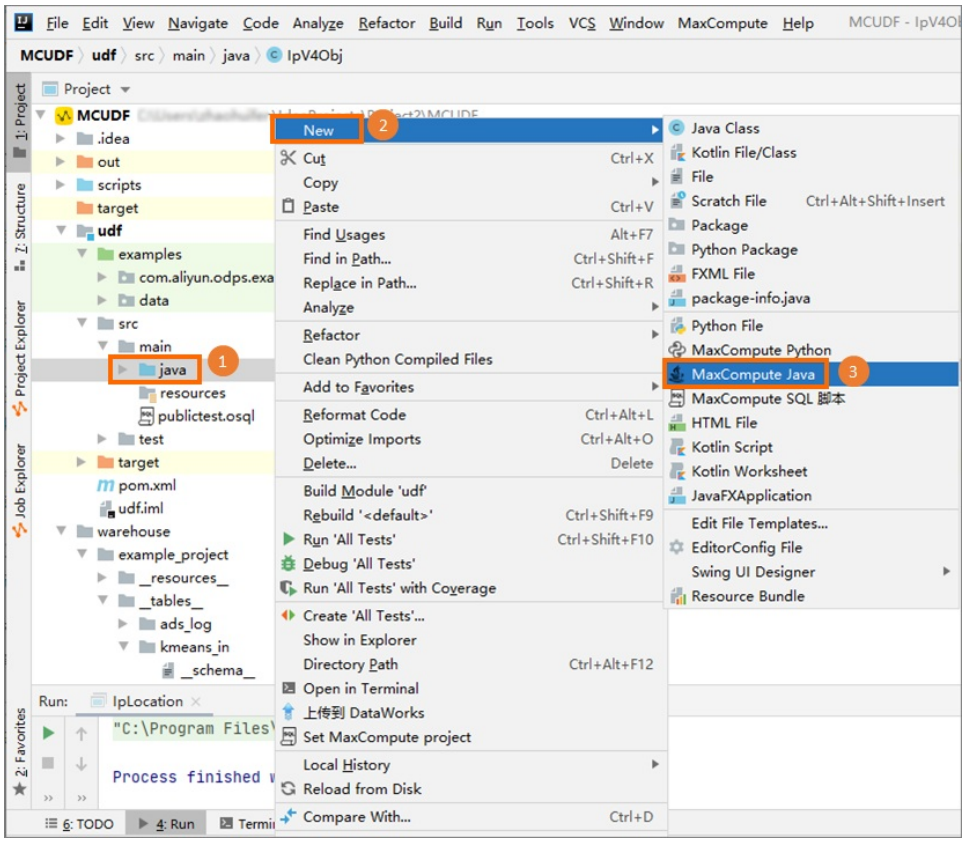
```
package com.aliyun.odps.udf.objects;
public class IpV4Obj {
    public long startIp ;
    public long endIp ;
    public String city;
    public String province;
    public IpV4Obj(long startIp, long endIp, String city, String province) {
        this.startIp = startIp;
        this.endIp = endIp;
        this.city = city;
        this.province = province;
    }
    @Override
    public String toString() {
        return "IpV4Obj{" +
                "startIp=" + startIp +
                ", endIp=" + endIp +
                ", city='" + city + '\'' +
                ", province='" + province + '\'' +
                '}';
    }
    public void setStartIp(long startIp) {
        this.startIp = startIp;
    }
    public void setEndIp(long endIp) {
        this.endIp = endIp;
    }
    public void setCity(String city) {
        this.city = city;
    }
    public void setProvince(String province) {
        this.province = province;
    }
    public long getStartIp() {
        return startIp;
    }
    public long getEndIp() {
        return endIp;
    }
    public String getCity() {
        return city;
    }
    public String getProvince() {
        return province;
    }
}
```

- IpV6Obj

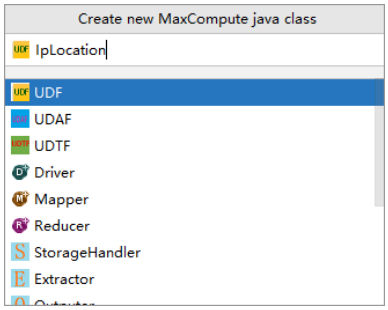
```
package com.aliyun.odps.udf.objects;
public class IpV6Obj {
    public String startIp ;
    public String endIp ;
    public String city;
    public String province;
    public String getStartIp() {
        return startIp;
    }
    @Override
    public String toString() {
        return "IpV6Obj{" +
                "startIp='" + startIp + '\'' +
                ", endIp='" + endIp + '\'' +
                ", city='" + city + '\'' +
                ", province='" + province + '\'' +
                '}';
    }
    public IpV6Obj(String startIp, String endIp, String city, String province) {
        this.startIp = startIp;
        this.endIp = endIp;
        this.city = city;
        this.province = province;
    }
    public void setStartIp(String startIp) {
        this.startIp = startIp;
    }
    public String getEndIp() {
        return endIp;
    }
    public void setEndIp(String endIp) {
        this.endIp = endIp;
    }
    public String getCity() {
        return city;
    }
    public void setCity(String city) {
        this.city = city;
    }
    public String getProvince() {
        return province;
    }
    public void setProvince(String province) {
        this.province = province;
    }
}
```

2. 编写MaxCompute UDF代码。

i. 在Project区域，右键单击Module的源码目录（即src > main > java），选择new > MaxCompute Java。



ii. 在Create new MaxCompute java class对话框，单击UDF并填写Name后，按Enter键并在代码编写区域输入代码。



例如Java Class名称为IpLocation。代码内容如下，代码可直接复制使用，无需修改。

```
package com.aliyun.odps.udf.udfFunction;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.UDF;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.utils.IpUtils;
import com.aliyun.odps.udf.objects.IpV4Obj;
import com.aliyun.odps.udf.objects.IpV6Obj;
import java.io.*;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;
public class IpLocation extends UDF {
    public static IpV4Obj[] ipV4ObjsArray;
    public static IpV6Obj[] ipV6ObjsArray;
    public IpLocation() {
        super();
    }
    @Override
    public void setup(ExecutionContext ctx) throws UDFException, IOException {
        //IPV4
        if(ipV4ObjsArray==null)
        {
```

```
        {
            BufferedInputStream bufferedInputStream = ctx.readResourceFileAsStream("ipv4.txt");
            BufferedReader br = new BufferedReader(new InputStreamReader(bufferedInputStream));
            ArrayList<IpV4Obj> ipV4ObjArrayList=new ArrayList<>();
            String line = null;
            while ((line = br.readLine()) != null) {
                String[] f = line.split("\\|", -1);
                if(f.length>=5)
                {
                    long startIp = IpUtils.StringToLong(f[0]);
                    long endIp = IpUtils.StringToLong(f[1]);
                    String city=f[3];
                    String province=f[4];
                    IpV4Obj ipV4Obj = new IpV4Obj(startIp, endIp, city, province);
                    ipV4ObjArrayList.add(ipV4Obj);
                }
            }
            br.close();
            List<IpV4Obj> collect = ipV4ObjArrayList.stream().sorted(Comparator.comparing(IpV4Obj::getStartIp)).collect(
Collectors.toList());
            ArrayList<IpV4Obj> basicIpV4DataList=(ArrayList)collect;
            IpV4Obj[] ipV4Objs = new IpV4Obj[basicIpV4DataList.size()];
            ipV4ObjsArray = basicIpV4DataList.toArray(ipV4Objs);
        }
        //IPV6
        if(ipV6ObjsArray==null)
        {
            BufferedInputStream bufferedInputStream = ctx.readResourceFileAsStream("ipv6.txt");
            BufferedReader br = new BufferedReader(new InputStreamReader(bufferedInputStream));
            ArrayList<IpV6Obj> ipV6ObjArrayList=new ArrayList<>();
            String line = null;
            while ((line = br.readLine()) != null) {
                String[] f = line.split("\\|", -1);
                if(f.length>=5)
                {
                    String startIp = IpUtils.StringToBigIntString(f[0]);
                    String endIp = IpUtils.StringToBigIntString(f[1]);
                    String city=f[3];
                    String province=f[4];
                    IpV6Obj ipV6Obj = new IpV6Obj(startIp, endIp, city, province);
                    ipV6ObjArrayList.add(ipV6Obj);
                }
            }
            br.close();
            List<IpV6Obj> collect = ipV6ObjArrayList.stream().sorted(Comparator.comparing(IpV6Obj::getStartIp)).collect(
Collectors.toList());
            ArrayList<IpV6Obj> basicIpV6DataList=(ArrayList)collect;
            IpV6Obj[] ipV6Objs = new IpV6Obj[basicIpV6DataList.size()];
            ipV6ObjsArray = basicIpV6DataList.toArray(ipV6Objs);
        }
    }
    public String evaluate(String ip){
        if(ip==null||ip.trim().isEmpty()||!(ip.contains(".")||ip.contains(":")))
        {
            return null;
        }
        int ipV4OrV6=0;
        try {
            ipV4OrV6= IpUtils.isIpV4OrV6(ip);
        } catch (Exception e) {
            return null;
        }
        //如果是IPv4
        if(ipV4OrV6==4)
        {
            int i = binarySearch(ipV4ObjsArray, IpUtils.StringToLong(ip));
            if(i>=0)
            {
                IpV4Obj ipV4Obj = ipV4ObjsArray[i];
                return ipV4Obj.city+","+ipV4Obj.province;
            }else{
                return null;
            }
        }else if(ipV4OrV6==6)//如果是IPv6
        {
            int i = binarySearchIPV6(ipV6ObjsArray, IpUtils.StringToBigIntString(ip));
```

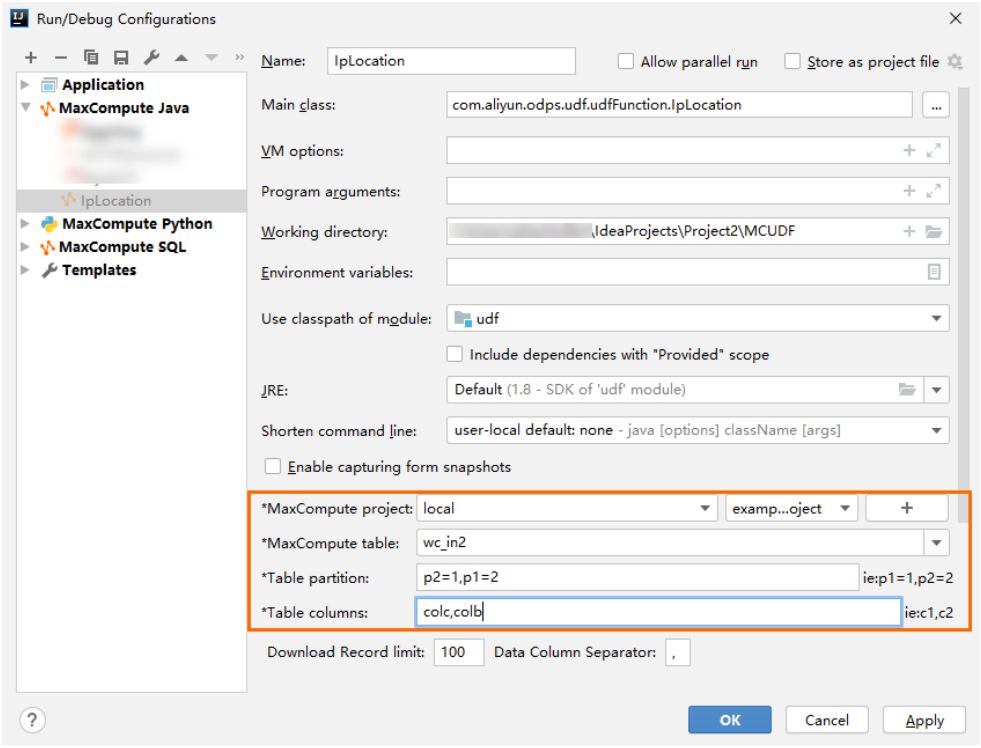
```
            if(i>=0)
            {
                IpV6Obj ipV6Obj = ipV6ObjsArray[i];
                return ipV6Obj.city+","+ipV6Obj.province;
            }else{
                return null;
            }
        }else{//如果不符合IPv4或IPv6格式
            return null;
        }
    }
    @Override
    public void close() throws UDFException, IOException {
        super.close();
    }
    private static int binarySearch(IpV4Obj[] array,long ip){
        int low=0;
        int hight=array.length-1;
        while (low<=hight)
        {
            int middle=(low+hight)/2;
            if((ip>=array[middle].startIp)&&(ip<=array[middle].endIp))
            {
                return middle;
            }
            if (ip < array[middle].startIp)
                hight = middle - 1;
            else {
                low = middle + 1;
            }
        }
        return -1;
    }
    private static int binarySearchIPV6(IpV6Obj[] array,String ip){
        int low=0;
        int hight=array.length-1;
        while (low<=hight)
        {
            int middle=(low+hight)/2;
            if((ip.compareTo(array[middle].startIp)>=0)&&(ip.compareTo(array[middle].endIp)<=0))
            {
                return middle;
            }
            if (ip.compareTo(array[middle].startIp) < 0)
                hight = middle - 1;
            else {
                low = middle + 1;
            }
        }
        return -1;
    }
    private class Invalid{
        private Invalid()
        {
        }
    }
}
```

3. 调试MaxCompute UDF，确保代码可以运行成功。

   更多调试操作，请参见通过本地运行调试UDF。

   i. 右键单击编写完成的MaxCompute UDF脚本，选择**Run**。

ii. 在Run/Debug Configurations对话框，配置下图红框所示运行参数，单击OK。

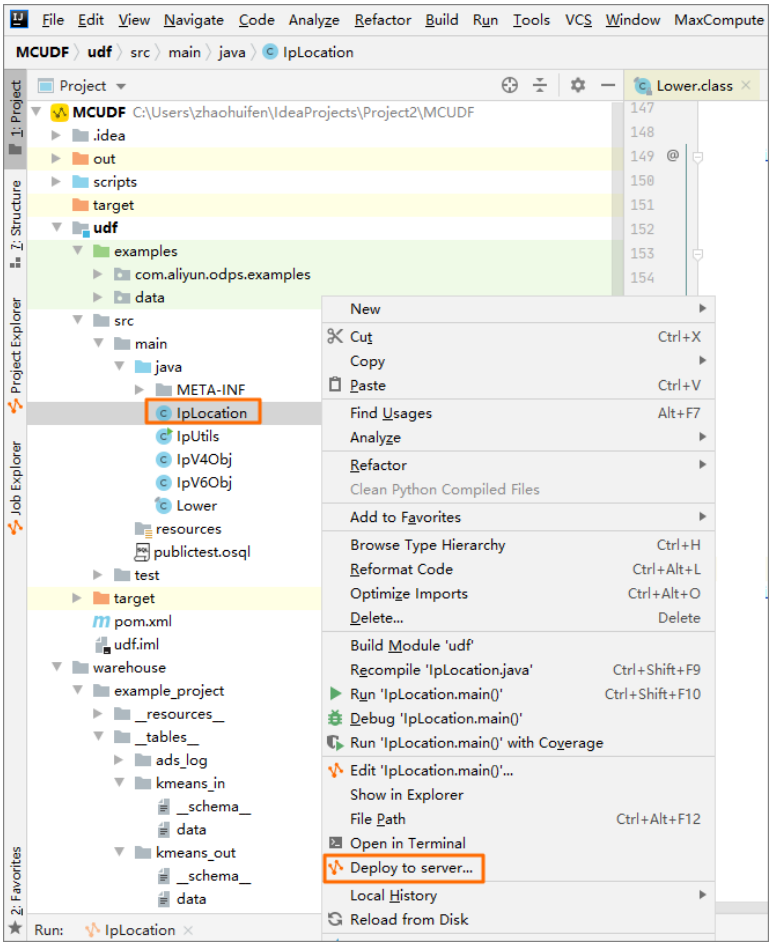


返回无报错，说明代码运行成功，即可继续执行后续步骤。如有报错，请按照Intellij IDEA报错信息处理。

> 说明 运行参数可参照图示数据填写。

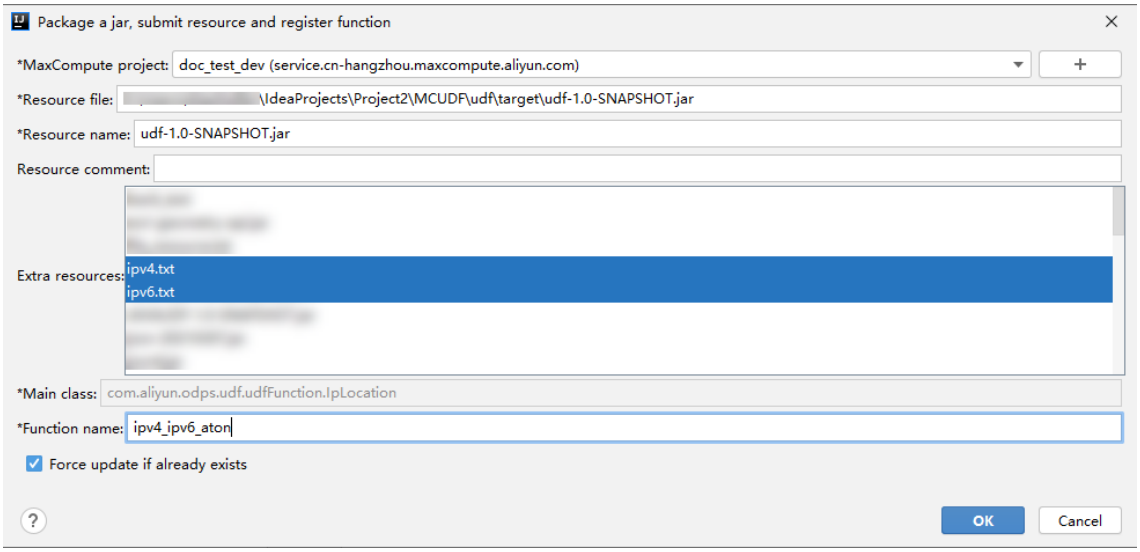
步骤三：注册MaxCompute UDF

1. 右键单击已经编译成功的MaxCompute UDF脚本，选择Deploy to server...。



2. 在Package a jar, submit resource and register function对话框中，配置参数信息。

   更多参数解释，请参见打包、上传及注册。



   Extra resources必须选中步骤一中上传的IP地址库文件ipv4.txt和ipv6.txt。假设注册好的函数名称为ipv4_ipv6_aton。

步骤四：调用MaxCompute UDF转换IP地址为归属地

1. 登录MaxCompute客户端。
2. 执行SQL SELECT语句调用MaxCompute UDF将IPv4或IPv6地址转换为归属地。

   命令示例如下。

   - 转换IPv4地址为归属地

```
select ipv4_ipv6_aton('116.11.34.15');
```

返回结果如下。

```
北海市,广西壮族自治区
```

- 转换IPv6地址为归属地

```
select ipv4_ipv6_aton('2001:0250:080b:0:0:0:0:0');
```

返回结果如下。

```
保定市,河北省
```

3.4. IntelliJ IDEA Java UDF开发最佳实践

IntelliJ IDEA是Java语言的集成开发环境，可以帮助您快速的开发Java程序。本文为您详细介绍如何使用IntelliJ IDEA的插件MaxCompute Studio进行Java UDF开发，实现大写字母转换为小写字母。

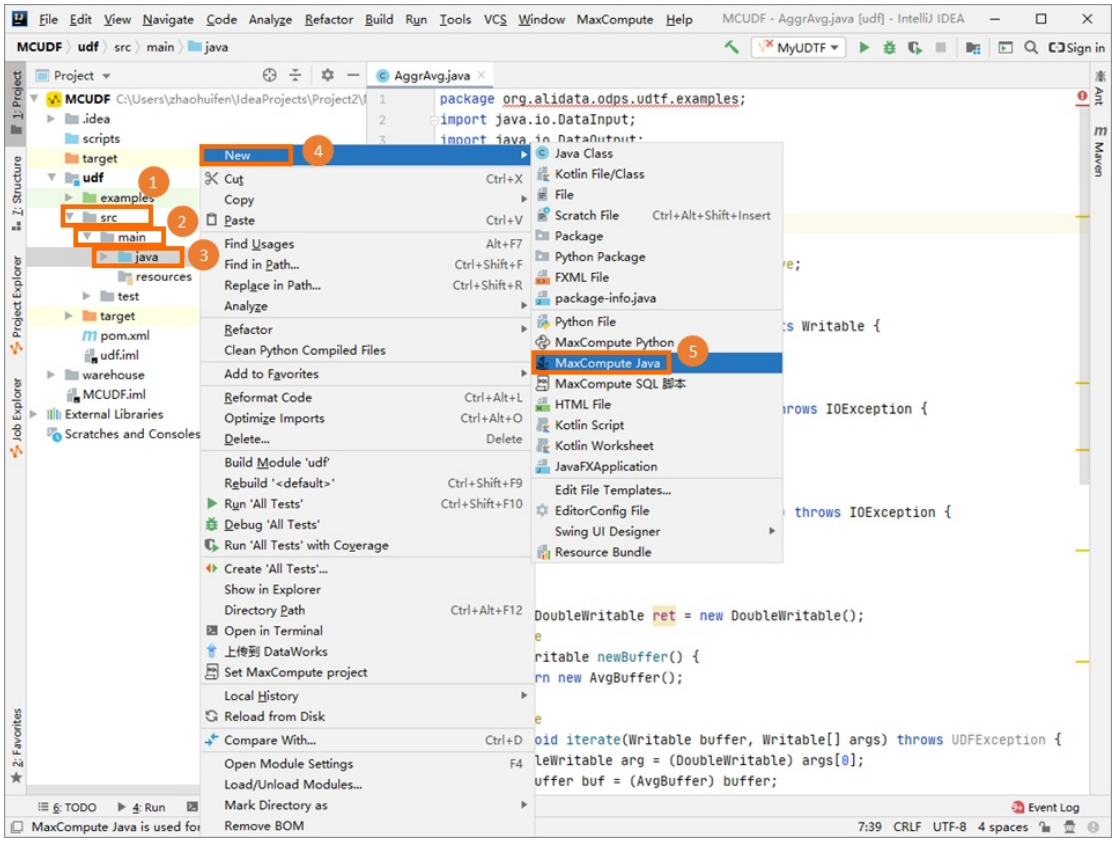
前提条件

请确认已在IntelliJ IDEA上完成如下准备工作：

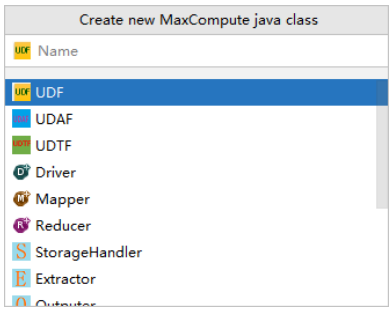
1. 安装MaxCompute Studio
2. 创建MaxCompute项目连接
3. 创建MaxCompute Java Module

操作步骤

1. 编写Java UDF。
   i. 在**Project**区域，右键单击Module的源码目录（即**src > main > java**），选择**new > MaxCompute Java**。

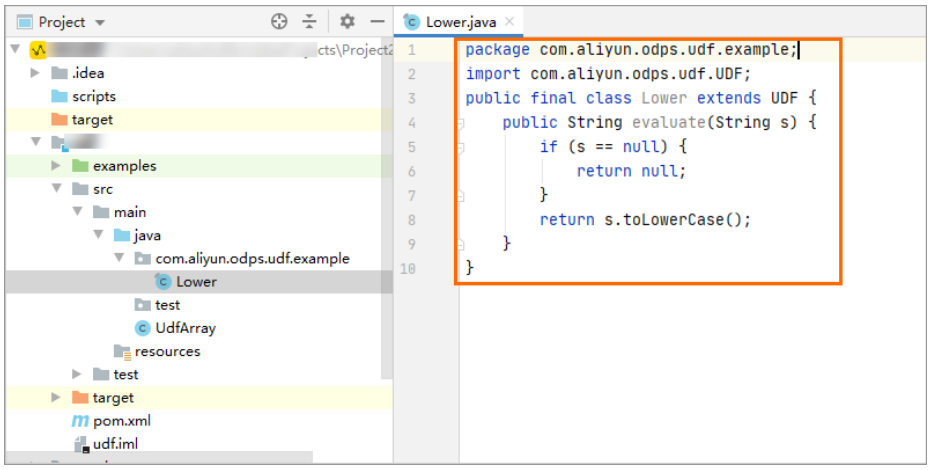


ii. 在**Create new MaxCompute java class**对话框，单击**UDF**并填写**Name**后，按Enter键。例如Java Class名称为Lower。



Name为创建的MaxCompute Java Class名称。如果还没有创建Package，在此处填写packagename.classname，会自动生成Package。

iii. 在代码编写区域写入如下代码。



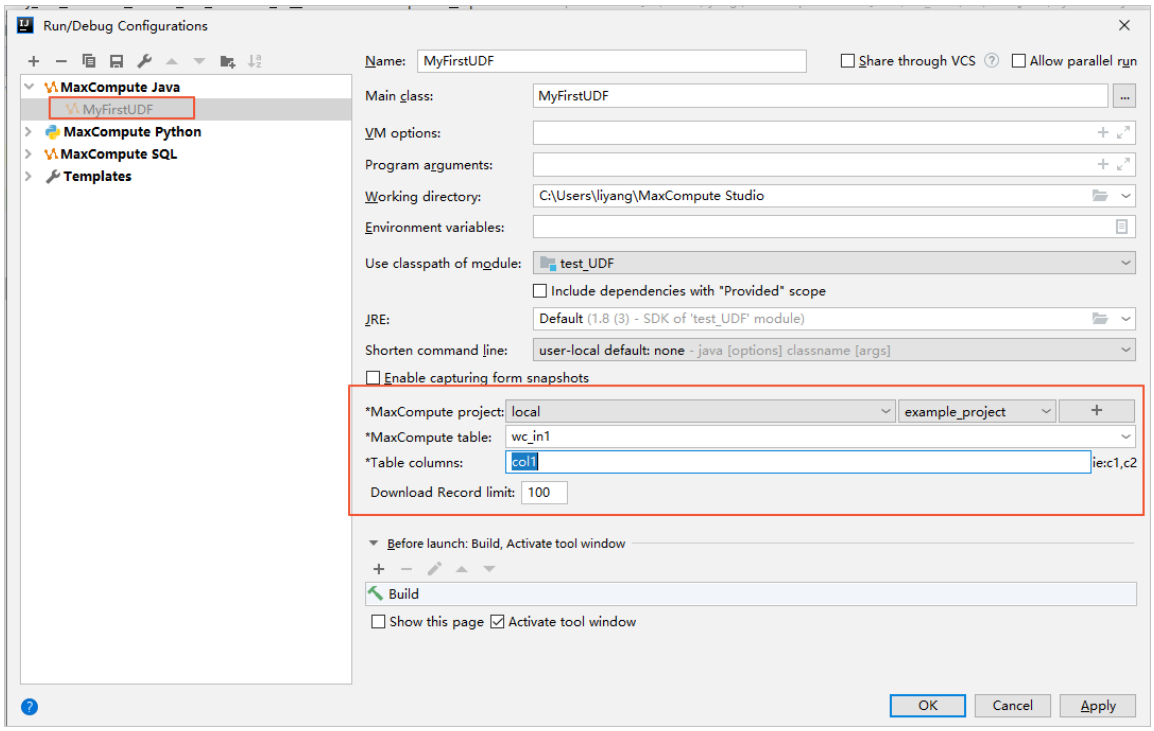
UDF代码示例如下。

```
package <packagename>;
import com.aliyun.odps.udf.UDF;
public final class Lower extends UDF {
    public String evaluate(String s) {
        if (s == null) {
           return null;
        }
           return s.toLowerCase();
    }
}
```

2. 调试UDF，确保可以运行成功。

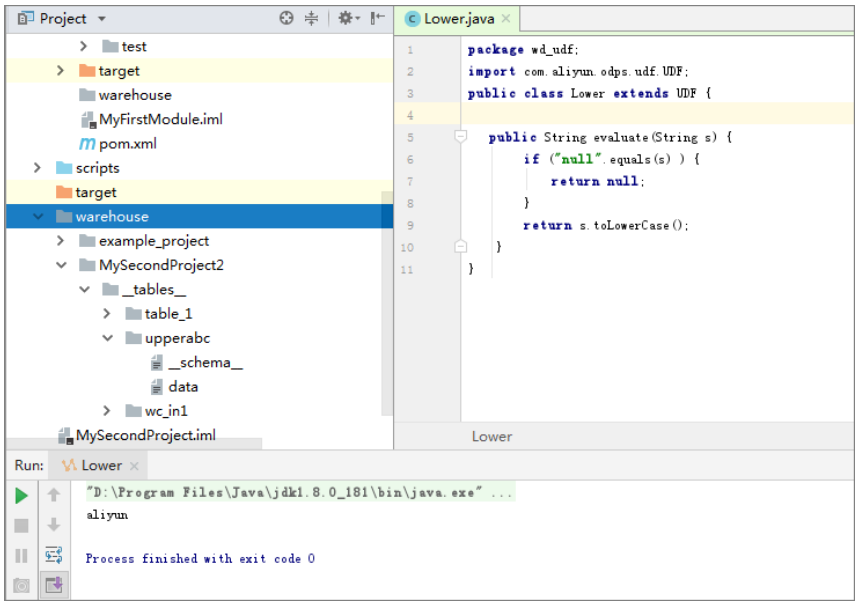
i. 在*Java*目录下，右键单击编写完成的Java脚本，选择**Run**。

ii. 在Run/Debug Configurations对话框，配置运行参数。



- MaxCompute project：UDF运行使用的MaxCompute空间。本地运行时选择**local**。
- MaxCompute table：UDF运行时需要使用的MaxCompute表的名称。
- Table columns：UDF运行时需要使用的MaxCompute表的列信息。

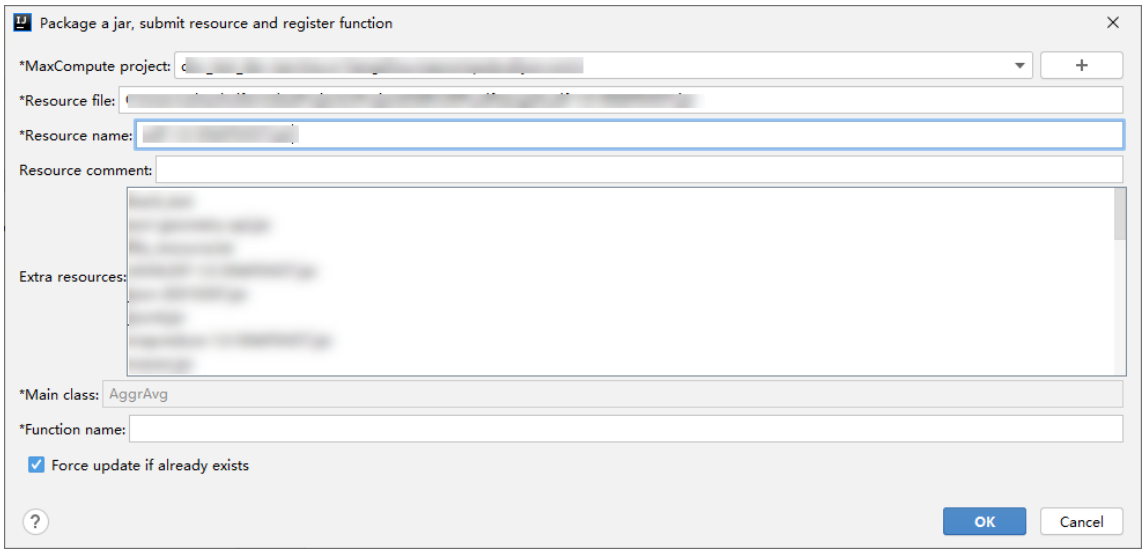
iii. 单击**OK**，运行结果如下图。



3. 注册MaxCompute UDF。

i. 在UDF Java文件上单击右键，选择**Deploy to server...**。

ii. 在Package a jar, submit resource and register function对话框，配置如下参数。

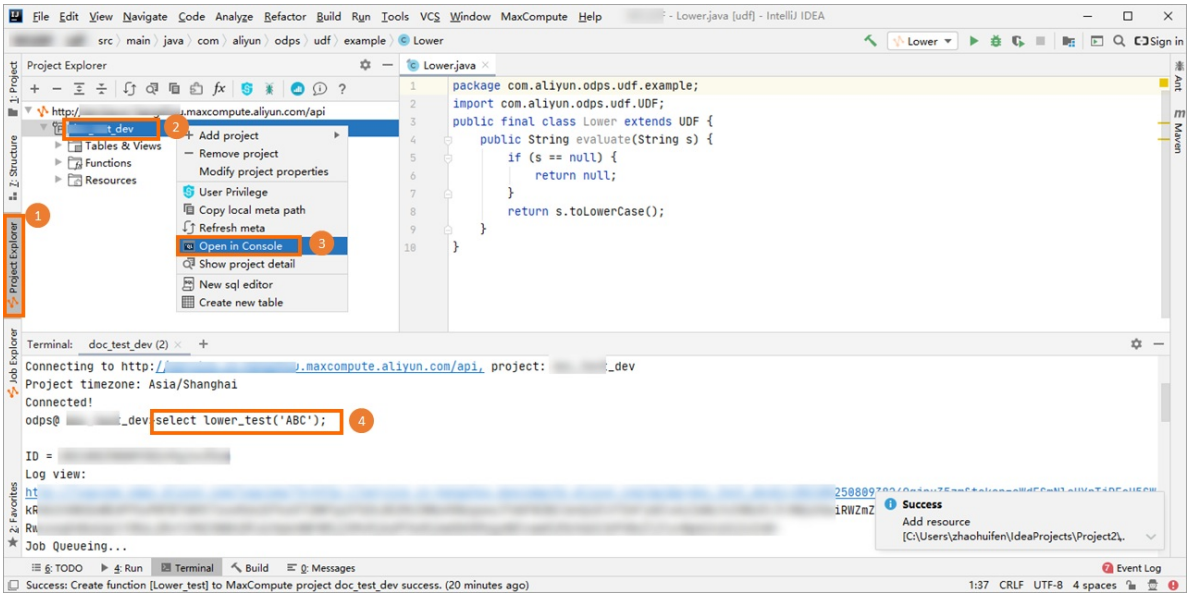


- **MaxCompute project**：UDF所在的MaxCompute项目名称。由于UDF本身是在连接的MaxCompute项目下编写的，此处保持默认值即可。
- **Resource file**：UDF依赖的资源文件路径。此处保持默认值即可。
- **Resource name**：UDF依赖的资源。此处保持默认值即可。
- **Function name**：注册的函数名称，即后续SQL中调用的UDF名称。例如Lower_test。

iii. 单击**OK**，完成UDF注册。

4. 调用UDF。

在左侧导航栏单击**Project Explore**，在目标MaxCompute项目上单击右键，选择**Open in Console**并在Console区域输入调用UDF的SQL语句，按Enter键运行即可。



SQL语句示例如下。

```
select Lower_test('ALIYUN');
```

返回结果如下。表明使用Intellij IDEA上开发的Java UDF函数Lower_test已经可用了。

```
+------------+
| _c0        |
+------------+
| aliyun     |
+------------+
1 records (at most 10000 supported) fetched by instance tunnel.
```

3.5. 使用MaxCompute分析IP来源最佳实践

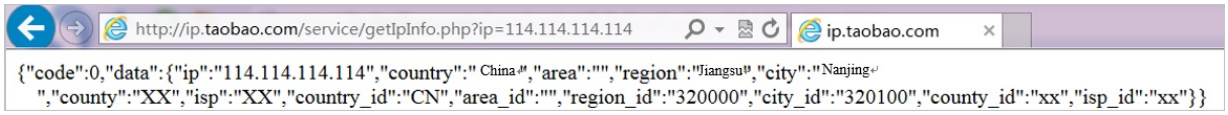
本文为您介绍如何使用MaxCompute分析IP来源，包括下载、上传IP地址库数据、编写UDF函数和编写SQL四个步骤。

前提条件

- 开通MaxCompute和DataWorks。
- 开通DataWorks。
- 在DataWorks上完成创建业务流程，本例使用DataWorks简单模式。详情请参见创建业务流程。

背景信息

淘宝IP地址库的查询接口为IP地址字串，使用示例如下。



在MaxCompute中禁止使用HTTP请求，因此目前可以通过如下三种方式实现在MaxCompute中查询IP：

- 用SQL将数据下载至本地，再发起HTTP请求查询。

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

- 下载IP地址库至本地，再进行查询。

  > **说明** 效率低下，且不利于数据仓库等产品分析使用。

- 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地址库需要自己定期维护。

下载IP地址库数据

1. 获取地址库数据。本文提供示例IP地址库数据UTF-8格式的不完整的地址库demo。
2. 下载示例地址库数据至本地，示例如下。

   ```
   0,16777215,"0.0.0.0","0.255.255.255","","","内网IP","内网IP","内网IP"
   16777216,16777471,"1.0.0.0","1.0.0.255","澳大利亚","","","",""
   16777472,16778239,"1.0.1.0","1.0.3.255","中国","福建省","福州市","","电信"
   ```

   示例数据说明如下：

   - 数据格式为UTF-8。
   - 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。IP地址段为整数形式，以便计算IP是否属于这个网段。

   > **说明** 如果您需要使用自己的IP地址，请自行下载IP地址库，具体的下载地址和使用方式请参见MaxCompute中实现IP地址归属地转换。

上传IP地址库数据

1. 在MaxCompute客户端执行如下语句，创建表ipresource存放IP地址库数据。

   ```
   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. 执行如下Tunnel命令，上传本地示例IP地址库数据至表ipresource。

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

   上述命令中，*D:/ipdata.txt.utf8*为IP地址库数据本地存放路径。更多命令说明请参见Tunnel命令。

   您可以执行如下语句验证数据是否上传成功。

   ```
   --查询表中数据条数。
   select count(*) from ipresource;
   ```

3. 执行如下SQL语句，查看表ipresource前10条的样本数据。

```
select * from ipresource limit 10;
```

返回结果如下。

```
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地址转化为整数类型的IP地址，本示例使用DataWorks的PyODPS完成。详情请参见创建PyODPS 2节点。

1. 进入**数据开发**页面。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 单击相应工作空间后的**数据开发**。
2. 新建Python资源。
   i. 右键单击业务流程，选择**新建 > MaxCompute > 资源 > Python**。
   ii. 在**新建资源**对话框中，填写**资源名称**，并勾选**上传为ODPS资源**，单击**新建**。
   iii. 在Python资源中输入如下代码后，单击图标。

```
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
```

   iv. 单击图标。
3. 新建函数。
   i. 右键单击已创建的业务流程，选择**新建 > MaxCompute > 函数**。
   ii. 在**新建函数**对话框中，输入**函数名称**，单击**新建**。

> **说明** 如果绑定了多个MaxCompute引擎，则需要选择**MaxCompute引擎实例**。

iii. 在函数的编辑页面，配置各项参数。

参数	描述
函数类型	选择函数类型，包括**数学运算函数**、**聚合函数**、**字符串处理函数**、**日期函数**、**窗口函数**和**其他函数**。
MaxCompute引擎实例	默认不可以修改。
函数名	UDF函数名，即SQL中引用该函数所使用的名称。需要全局唯一，且注册函数后不支持修改。
责任人	默认显示。
类名	实现UDF的主类名，必填。 **说明** 当资源类型为Python时，类名格式为Python资源名称.类名（资源名称中的.py无需填写）。
资源列表	完整的文件名称，支持模糊匹配查找本工作空间中已添加的资源，必填。 多个文件之间，使用英文逗号（,）分隔。
描述	针对当前UDF作用的简单描述。
命令格式	该UDF的具体使用方法示例，例如`test`。
参数说明	支持输入的参数类型以及返回参数类型的具体说明。
返回值	返回值，例如1，非必填项。
示例	函数中的示例，非必填项。

4. 单击工具栏中的图标。
5. 提交函数。
   i. 单击工具栏中的图标。
   ii. 在**提交新版本**对话框中，输入**备注**。
   iii. 单击**确认**。

在SQL中使用UDF函数分析IP来源

1. 右键单击业务流程，选择**新建 > MaxCompute > ODPS SQL**。
2. 在**新建节点**对话框中输入**节点名称**，并单击**提交**。
3. 在ODPS SQL节点编辑页面，输入如下语句。

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

4. 单击图标运行代码。
5. 您可以在**运行日志**查看运行结果。

3.6. 解决DataWorks 10 MB文件限制问题最佳实践

本文为您介绍如何解决在DataWorks上执行MapReduce作业时，大于10 MB的JAR和资源文件不能上传至DataWorks的问题，方便您使用调度功能定期执行MapReduce作业。

前提条件

请下载并安装MaxCompute客户端，详情请参见安装并配置MaxCompute客户端。

操作步骤

1. 在MaxCompute客户端上执行如下命令上传大于10 MB的资源。

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

2. 通过MaxCompute客户端上传的资源，在DataWorks左侧资源列表中不显示。因此需要执行如下命令查看资源列表，确认上传是否成功。

```
--查看资源。
list resources;
```

3. 减小JAR文件。DataWorks执行MapReduce作业的时候，需要在本地执行，所以保留Main函数即可。

```
jar
-resources test_mr.jar,test_ab.jar --资源在客户端注册后直接引用。
-classpath test_mr.jar --减小JAR文件策略：在gateway上提交包含Main函数的Mapper和Reducer，不需要提交额外的三方依赖，其他都可以放在resour
ces com.aliyun.odps.examples.mr.test_mr wc_in wc_out中。
```

3.7. 实现指定用户访问特定UDF最佳实践

本文为您介绍如何实现将资源（表、UDF等）设置为仅能被指定的用户访问。此方法涉及数据的加密解密算法，属于数据安全管控范畴。

前提条件

您需要提前安装MaxCompute客户端，以实现指定UDF被指定用户访问的操作。详情请参见安装并配置MaxCompute客户端。

背景信息

设置用户访问权限的常见方法有如下几种：

- Package方案，通过打包授权进行权限精细化管控。

  Package用于解决跨项目空间的数据共享及资源授权问题。通过Package授予用户开发者角色后，用户拥有所有权限，风险不可控。详情请参见基于Package跨项目访问资源。

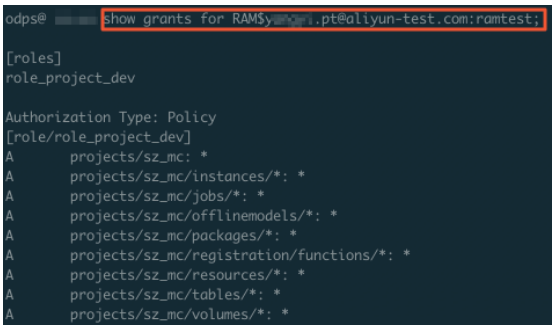
  - 下图为DataWorks开发者角色的权限。

```
odps@ [redacted]:desc role role_project_dev;

Authorization Type: Policy
A       projects/sz_mc: *
A       projects/sz_mc/instances/*: *
A       projects/sz_mc/jobs/*: *
A       projects/sz_mc/offlinemodels/*: *
A       projects/sz_mc/packages/*: *
A       projects/sz_mc/registration/functions/*: *
A       projects/sz_mc/resources/*: *
A       projects/sz_mc/tables/*: *
A       projects/sz_mc/volumes/*:
```

  由上图可见，开发者角色对工作空间中的Package、Functions、Resources和Table默认有全部权限，不符合权限配置的要求。

- 下图为通过DataWorks添加子账号并赋予开发者角色的权限。



由此可见，通过打包授权和DataWorks默认的角色都不能满足特定用户访问指定UDF的需求。例如，授予子账号 `RAM$xxxxx.pt@aliyun-test.com:ramtest` 开发者角色，则默认该子账号拥有当前工作空间中全部对象的所有操作权限，详情请参见用户授权。

- 在DataWorks中新建角色进行权限管控。

  在DataWorks工作配置页面的**MaxCompute高级配置**页面，可以对自定义用户角色进行权限管控。在该页面只能针对某个表或项目进行授权，不能对资源和UDF进行授权。

  > **说明** 更多有关DataWorks工作空间的MaxCompute属性介绍，请参见MaxCompute高级配置。

- Role Policy结合Project Policy实现指定用户访问指定UDF。

  通过Policy可以精细化地管理具体用户对特定资源的具体权限粒度。

  > **说明** 为了安全起见，建议初学者使用测试项目来验证Policy。

因此您可以通过Policy方案实现特定UDF被指定用户访问：

- 如果您不想让其他用户访问工作空间内具体的资源，在DataWorks中添加数据开发者权限后，再根据Role Policy的操作，在MaxCompute客户端将其配置为拒绝访问权限。
- 如果您需要指定用户访问指定资源，在DataWorks中添加数据开发者权限后，再根据Project Policy的操作，在MaxCompute客户端将其配置为允许访问权限。

操作步骤

1. 创建默认拒绝访问UDF的角色。
   i. 在客户端输入如下命令创建角色denyudfrole。

   ```
   create role denyudfrole;
   ```

   ii. 创建Policy授权文件，如下所示。

   ```
   {
   "Version": "1", "Statement"
   [{
   "Effect":"Deny",
   "Action":["odps:Read","odps:List"],
   "Resource":"acs:odps:*:projects/sz_mc/resources/getaddr.jar"
   },
   {
   "Effect":"Deny",
   "Action":["odps:Read","odps:List"],
   "Resource":"acs:odps:*:projects/sz_mc/registration/functions/getregion"
   }
    ] }
   ```

   iii. 设置Role Policy。

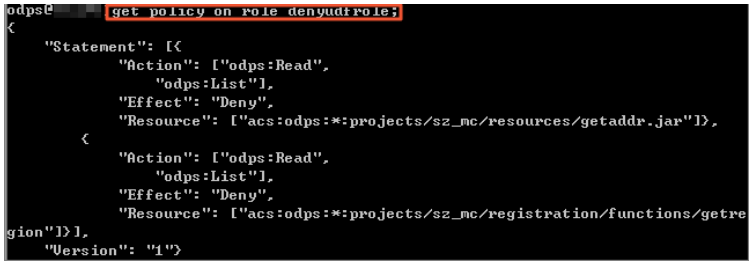
   在客户端执行如下命令，设置Role Policy文件的存放路径。

   ```
   put policy /Users/yangyi/Desktop/role_policy.json on role denyudfrole;
   ```

iv. 在客户端执行如下命令查看Role Policy。

```
get policy on role denyudfrole;
```

返回结果如下。

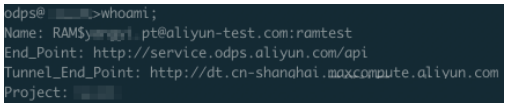


v. 在客户端执行如下命令添加子账号至role denyudfrole。

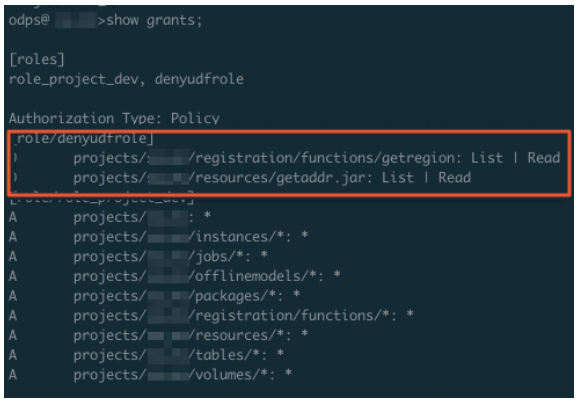
```
grant denyudfrole to RAM$xxxx.pt@aliyun-test.com:ramtest;
```

2. 验证拒绝访问UDF的角色是否创建成功。

i. 登录客户端输入 `whoami;` 确认角色。

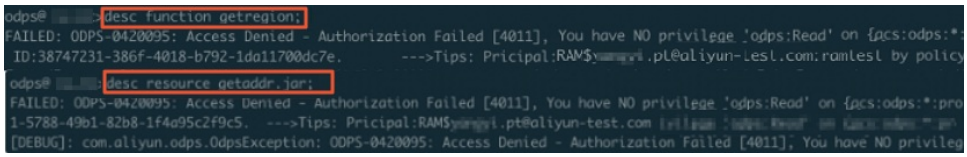


ii. 通过 `show grants;` 查看当前登录用户权限。



通过查询发现该RAM子账号有两个角色，一个是role_project_dev（即DataWorks默认的开发者角色），另一个是刚自定义创建的denyudfrole。

iii. 验证自建UDF以及依赖的包的权限。



通过上述验证发现，该子账号在拥有DataWorks开发者角色的前提下，没有自建UDF（getregion）的读权限。您还需要结合Project Policy来实现该UDF只能被指定的用户访问。

3. 配置Project Policy。

i. 编写Policy。

```
{
"Version": "1", "Statement":
[{
"Effect":"Allow",
"Principal":"RAM$yangyi.pt@aliyun-test.com:yangyitest",
"Action":["odps:Read","odps:List","odps:Select"],
"Resource":"acs:odps:*:projects/sz_mc/resources/getaddr.jar"
},
{
"Effect":"Allow",
 "Principal":"RAM$xxxx.pt@aliyun-test.com:yangyitest",
"Action":["odps:Read","odps:List","odps:Select"],
"Resource":"acs:odps:*:projects/sz_mc/registration/functions/getregion"
}] }
```

ii. 设置Role Policy。

在客户端执行如下命令，设置Role Policy文件的存放路径。

```
put policy /Users/yangyi/Desktop/project_policy.json;
```

iii. 在客户端执行如下命令查看Role Policy。

```
get policy;
```

返回结果如下。

```
odps@ ** get policy;
{
    "Statement": [{
            "Action": ["odps:Read",
                "odps:List",
                "odps:Select"],
            "Effect": "Allow",
            "Principal": ["RAM$**i.pt@aliyun-test.com:yangyitest"],
            "Resource": ["acs:odps:*:projects/sz_mc/resources/getaddr.jar"]},
        {
            "Action": ["odps:Read",
                "odps:List",
                "odps:Select"],
            "Effect": "Allow",
            "Principal": ["RAM$yangyi.pt@aliyun-test.com:yangyitest"],
            "Resource": ["acs:odps:*:projects/sz_mc/registr**
    "Version": "1"}
```

iv. 通过 `whoami;` 和 `show grants;` 进行验证。

```
odps@ ** whoami;
Name: RAM$**.pt@aliyun-test.com:yangyitest
End_Point: http://service.odps.aliyun.com/api
Tunnel_End_Point: http://dt.cn-shanghai.maxcompute.aliyun.com
Project: sz_mc
odps@ ** show grants;

[roles]
role_project_dev

Authorization Type: Policy
[role/role_project_dev]
A       projects/sz_mc: *
A       projects/sz_mc/instances/*: *
A       projects/sz_mc/jobs/*: *
A       projects/sz_mc/offlinemodels/*: *
A       projects/sz_mc/packages/*: *
A       projects/sz_mc/registration/functions/*: *
A       projects/sz_mc/resources/*: *
A       projects/sz_mc/tables/*: *
A       projects/sz_mc/volumes/*: *
[user/RAM$yangyi.pt@aliyun-test.com:yangyitest]
A       projects/sz_mc/registration/functions/getregion: List | Read | Select
A       projects/sz_mc/resources/getaddr.jar: List | Read | Select
```

v. 运行SQL任务，查看是否仅指定的RAM子账号能够查看指定的UDF和依赖的包。

- 指定的RAM子账号查看指定的UDF。

```
odps@ [redacted]>select getregion('172.[redacted]');

ID = 2019011409[redacted]=2
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.com/api&p=sz_[redacted]
U0ODA2MTU2Myx7IlN0YXRl[redacted]/hZCJdLCJFZmZlY3QiOiJBbGxv[redacted]
biI6IjEifQ==
Job Queueing.
--------------------------------------------------------------------------------
                STAGES        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  BACKU
M1_job_0 .................    TERMINATED      1          1        0        0
--------------------------------------------------------------------------------
STAGES: 01/01    [=================================>>] 100%  ELAPSED TIME: 24.34 s
--------------------------------------------------------------------------------
Summary:
resource cost: cpu 0.27 Core * Min, memory 0.53 GB * Min
inputs:
outputs:
Job run time: 18.000
Job run mode: fuxi job
Job run engine: execution engine
M1:
        instance count: 1
        run time: 18.000
        instance time:
                min: 16.000, max: 16.000, avg: 16.000
        input records:
        output records:
                AdhocSink1: 1  (min: 1, max: 1, avg: 1)


+------------+
| _c0        |
+------------+
| [美国，美国，，] |
+------------+
```

- 查看依赖包。

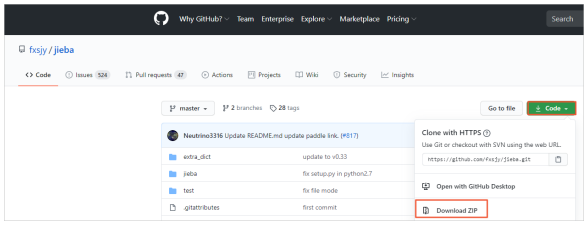
```
odps@ [redacted]>desc resource getaddr.jar;
Name                    getaddr.jar
Owner                   ALIYUN$[redacted].pt@aliyun-test.com
Type                    JAR
Comment                 IDE RESOURCE UPDATE TO ODPS /home/admin/oxs-base-biz-phoenix/temp/4d3efccz0sl9eiirin53o5n4/getaddr.jar
CreatedTime             2018-05-24 19:51:16
LastModifiedTime        2018-05-24 19:51:16
LastUpdator
Size                    1353716
Md5sum                  770497a9f605e09e198cb166cec7fa08
```

3.8. PyODPS节点实现结巴中文分词

本文为您介绍如何使用DataWorks的PyODPS类型节点，借助开源结巴中文分词包实现对中文字段的分词并写入新的表，以及如何通过闭包函数使用自定义词典进行分词。

前提条件

- 请首先确保您已经完成DataWorks工作空间的创建，本示例使用绑定多个MaxCompute计算引擎的**简单模式**工作空间，详情请参见创建工作空间。
- 请在GitHub下载开源结巴分词中文包。



背景信息

PyODPS集成了MaxCompute的Python SDK。您可以在DataWorks的PyODPS节点上直接编辑Python代码，并使用MaxCompute的Python SDK。关于PyODPS节点的详情，请参见PyODPS节点。

本文主要为您介绍如何使用PyODPS节点实现结巴中文分词。

- 借助开源包实现结巴中文分词
- 自定义的词典实现结巴分词

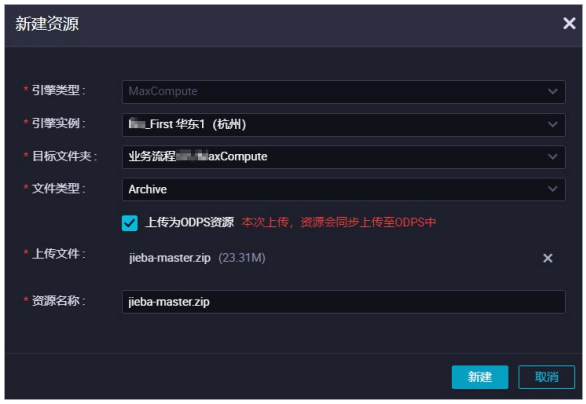
> **注意** 本文的操作仅作为代码示例，不建议用于实际的生产环境。

借助开源包实现结巴中文分词

1. 创建业务流程。
   i. 登录DataWorks控制台。
   ii. 在左侧导航栏，单击**工作空间列表**。
   iii. 选择工作空间所在地域后，单击相应工作空间后的**进入数据开发**。
   iv. 鼠标悬停至图标，单击**业务流程**。
   v. 在**新建业务流程**对话框中，输入**业务名称**和**描述**，单击**新建**。

   > 注意 业务名称必须是大小写字母、中文、数字、下划线（_）以及小数点（.），且不能超过128个字符。

2. 上传jieba-master.zip包。
   i. 展开新建的业务流程下的**MaxCompute**，右键单击**资源**，选择**新建 > Archive**。
   ii. 在**新建资源**对话框中，配置各项参数后，单击**新建**。



参数	描述
引擎类型	从下拉列表中选择该资源所在的计算引擎。 **说明** 如果您所在的工作空间仅绑定一个实例，则不会显示该参数。
引擎实例	任务所绑定的MaxCompute引擎名称。
目标文件夹	默认当前所在文件夹的路径，您可以进行修改。
文件类型	此处选择**Archive**类型。 **说明** 如果该资源包已经在MaxCompute（ODPS）客户端上传过，请取消勾选**上传为ODPS资源**，否则上传会报错。
上传文件	单击**点击上传**，在本地选择已下载的文件jieba-master.zip后，单击**打开**。
资源名称	资源的名称，无需和上传的文件名保持一致，但需要符合以下规范： ■ 资源名称仅包含中文、字母、数字、英文句号（.）、下划线（_）和短划线（-）。 ■ 资源类型为**Archive**时，资源名称和文件名的后缀必须一致，且后缀名包含.zip、.tgz、.tar.gz或.tar。

   iii. 在工具栏中，单击图标。
   iv. 在**提交新版本**对话框中，输入**变更描述**，单击**确定**。
3. 创建测试数据表。
   i. 展开新建的业务流程下的**MaxCompute**，右键选择**表 > 新建表**。
   ii. 在**新建表**对话框中，输入**表名**，单击**新建**。

   > 说明 本示例表名示例为jieba_test。

iii. 单击**DDL模式**，输入以下建表DDL语句。

```
CREATE TABLE jieba_test (
    `chinese` string,
    `content` string
);
```

> **说明** 本教程准备了两列测试数据，您在后续开发过程中可以选择一列进行分词。

iv. 在弹出的对话框中，单击**确定**。

v. 在**基本属性**区域，输入表的**中文名**，单击**提交到生产环境**。

vi. 在**提交生产确认**对话框中，选中**我已悉知风险，确认提交**，单击**确认**。

4. 以同样的方式创建存放测试结果的数据表**jieba_result**，DDL语句如下所示。

```
CREATE TABLE jieba_result (
    `chinese` string
);
```

> **说明** 本例仅对测试数据的chinese列进行分词处理，因此结果表仅有一列。

5. 单击测试数据下载分词测试数据。

6. 上传测试数据：

i. 在**数据开发**页面，单击图标。

ii. 在**数据导入向导**对话框中，输入需要导入数据的测试表jieba_test并选中，单击**下一步**。

iii. 单击**浏览**，上传您下载至本地的*jieba_test.csv*文件，单击**下一步**。

iv. 选中**按名称匹配**，单击**导入数据**。

7. 创建PyODPS 2节点。

i. 展开业务流程中的**MaxCompute**，右键单击**数据开发**，选择**新建 > PyODPS 2**。

ii. 在**新建节点**对话框中，输入**节点名称**，并选择**目标文件夹**，单击**提交**。

> **说明**
> - 节点名称必须是大小写字母、中文、数字、下划线（_）和小数点（.），且不能超过128个字符。
> - 本示例的**节点名称**为word_split。

iii. 在word_split节点，输入下述PyODPS代码。

```
def test(input_var):
    import jieba
    import sys
    reload(sys)
    sys.setdefaultencoding('utf-8')
    result=jieba.cut(input_var, cut_all=False)
    return "/ ".join(result)
hints = {
    'odps.isolation.session.enable': True
}
libraries =['jieba-master.zip']  #引用您的jieba-master.zip压缩包。
iris = o.get_table('jieba_test').to_df()  #引用您的jieba_test表中的数据。
example = iris.chinese.map(test).execute(hints=hints, libraries=libraries)
print(example)  #查看分词结果，分词结构为MAP类型数据。
abci=list(example)   #将分词结果转为list类型的数据。
i = 0
for i in range(i,len(abci)):
    pq=str(abci[i])
    o.write_table('jieba_result',[pq])  #通过循环，逐条写入数据至结果表jieba_result中。
    i+=1
else:
    print("done")
```

iv. 单击工具栏中的图标。

v. 单击工具栏中的图标，在**参数**对话框中，从**调度资源组**下拉列表选择需要使用的资源组，单击**确定**。

> **说明** **调度资源组**详细说明，请参见DataWorks资源组概述。

vi. 在页面下方的**运行日志**区域，查看结巴分词程序的运行结果。

```
                                    chinese
0   数据库/ 备份/ 是/ 为/ 数据库/ 提供/ 连续/ 数据保护/ 低成本/ 的/ 备份/ 服务
1         数据库/ 备份/ 拥有/ 一套/ 完整/ 的/ 数据备份/ 和/ 数据恢复/ 解决方案
2  可以/ 通过/ 简单/ 的/ 配置/ 实现/ 数据库/ 全量/ 备份/ 增量/ 备份/ 以及...
3       为了/ 节省成本/ ，/ 可以/ 选择/ 多种/ OSS/ 存储/ 类型/ 进行/ 存储
4  在/ 进行/ 数据恢复/ 时/ ，/ 可以/ 使用/ 存储/ 的/ 增量/ 备份/ 实现/ ...
5  为了/ 降低/ 在/ 故障/ 发生/ 后/ 数据/ 丢失/ ，/ 数据库/ 备份/ DBS/...
6              出于/ 安全/ 合规/ 要求/ ，/ 部分/ 数据/ 需要/ 长期/ 保存
7  作为/ 完整/ 数据库/ 灾备/ 方案/ ，/ 除了/ 要/ 有/ 本地/ 数据库/ 备份/...
8        数据库/ 备份/ DBS/ 提供数据/ 全量/ 备份/ 增量/ 备份/ 和/ 数据恢复

done

2022-03-14 15:05:39 INFO ================================================================
2022-03-14 15:05:39 INFO Exit code of the Shell command 0
2022-03-14 15:05:39 INFO --- Invocation of Shell command completed ---
2022-03-14 15:05:39 INFO Shell run successfully!
```

8. 创建和运行ODPS SQL节点。
   i. 展开业务流程中的**MaxCompute**，右键单击**数据开发**，选择**新建 > ODPS SQL**。
   ii. 在**新建节点**对话框中，输入**节点名称**，并选择**目标文件夹**，单击**提交**。

   > **说明** 节点名称必须是大小写字母、中文、数字、下划线（_）和小数点（.），且不能超过128个字符。

   iii. 在节点的编辑页面，输入以下SQL语句。

   ```
   select * from jieba_result;
   ```

   iv. 单击工具栏中的图标。

   v. 单击工具栏中的图标，在**参数**对话框中，从**调度资源组**下拉列表选择需要使用的资源组，单击**确定**。

   > **说明** 调度资源组详细说明，请参见DataWorks资源组概述。

   vi. 在**MaxCompute计算成本估计**对话框中，确认**预估费用**后，单击**运行**。
   vii. 在页面下方的**运行日志**区域，查看运行结果。

自定义的词典实现结巴分词

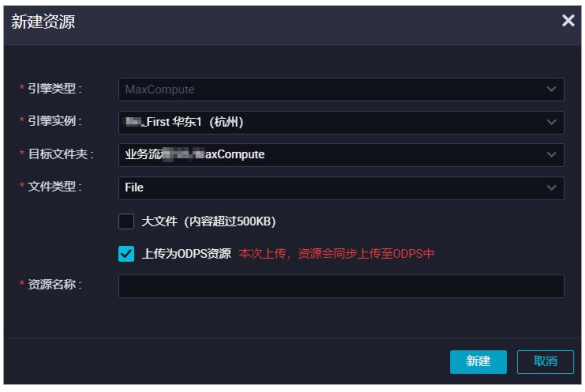
如果开源结巴分词的词库无法满足您的需求，您可以选择使用自定义的词典。

PyODPS自定义函数可以读取上传至MaxCompute的资源（表资源或文件资源）。此时，自定义函数需要写为闭包函数或Callable类。如果您需要引用复杂的自定义函数，则可以使用DataWorks的注册MaxCompute函数功能，详情请参见注册MaxCompute函数。

本文以使用闭包函数的方式，引用上传至MaxCompute的资源文件（即自定义词典）key_words.txt。

> **说明** 本示例的资源文件名为key_words.txt。

1. 展开业务流程中的**MaxCompute**，右键单击**资源**，选择**新建 > File**。
2. 在**新建资源**对话框中，配置各项参数，单击**确定**。



参数	描述
引擎类型	从下拉列表中选择该资源所在的计算引擎。 > **说明** 如果您所在的工作空间仅绑定一个实例，则不会显示该参数。

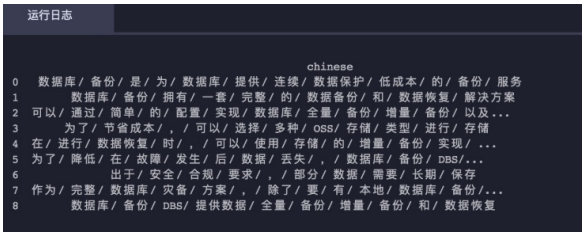
参数	描述
引擎实例	任务所绑定的MaxCompute引擎名称。
目标文件夹	默认当前所在文件夹的路径，您可以进行修改。
文件类型	此处选择**File**类型。 **说明** 如果您从本地上传词典文件至DataWorks，则文件必须为UTF-8编码格式。
上传文件	单击**点击上传**，在本地选择文件key_words.txt后，单击**打开**。
资源名称	资源名称仅包含中文、字母、数字、英文句号（.）、下划线（_）和短划线（-）。

3. 在key_words.txt资源编辑页面，输入自定义词典的内容。格式如下。
   - 一个词占一行。
   - 每一行包括词、词频（可省略）和词性（可省略），使用空格分隔，且不可以颠倒顺序。
4. 单击工具栏中的图标提交资源。
5. 创建一个**PyODPS 2**节点，并输入如下代码。

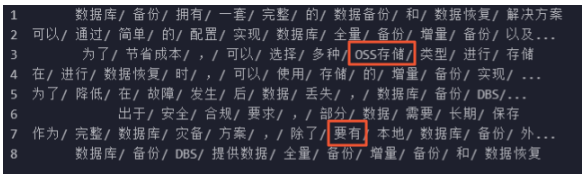
```
def test(resources):
    import jieba
    import sys
    reload(sys)
    sys.setdefaultencoding('utf-8')
    fileobj = resources[0]
    def h(input_var):#在嵌套函数h()中，执行词典加载和分词。
        import jieba
        jieba.load_userdict(fileobj)
        result=jieba.cut(input_var, cut_all=False)
        return "/ ".join(result)
    return h
hints = {
    'odps.isolation.session.enable': True
}
libraries =['jieba-master.zip']  #引用您的jieba-master.zip压缩包。
iris = o.get_table('jieba_test').to_df()  #引用您的jieba_test表中的数据。
file_object = o.get_resource('key_words.txt') #get_resource()引用odps资源。
example = iris.chinese.map(test, resources=[file_object]).execute(hints=hints, libraries=libraries) #map调用函数，并传递resour
ces参数。
print(example)  #查看分词结果，分词结构为MAP类型数据。
abci=list(example)    #将分词结果转为list类型数据。
for i in range(i,len(abci)):
    pq=str(abci[i])
    o.write_table('jieba_result',[pq])  #通过循环，逐条写入数据至结果表jieba_result中。
    i+=1
else:
    print("done")
```

6. 运行代码，对比引用自定义词典前后的结果。

   引用自定义词典前结果如下。



   引用自定义词典后结果如下。



3.9. PyODPS节点实现避免将数据下载到本地

本文为您介绍PyODPS如何避免将数据下载到本地。

背景信息

PyODPS提供了多种方便下载数据到本地的方法。因此，在设备允许的情况下，可以把数据下载到本地处理，然后再上传至MaxCompute。但是这种操作非常低效，数据下载到本地进行处理，无法使用MaxCompute的大规模并行能力。当数据量大于10 MB时，不建议进行本地数据处理。常见的将数据下载到本地的操作如下：

- Head、Tail和To_pandas方法的调用。通常，可以调用 `head、tail` 方法返回少量数据进行数据探查，当数据量较大时，建议调用Persisit方法，将数据直接保存在MaxCompute表中。详情请参见执行。
- 在表或SQL实例上直接执行Open_reader方法获取表数据。当数据量大时，建议使用PyODPS DataFrame（从MaxCompute表创建）和MaxCompute SQL来处理数据，以替代本地数据处理这种比较低效的方式。

示例代码

将一份JSON串数据按Key-Value对展开成一行，示例代码如下。

- 本地测试，通过 `head()` 方法返回少量数据进行测试。

```
In [12]: df.head(2)
                json
0  {"a": 1, "b": 2}
1  {"c": 4, "b": 3}
In [14]: from odps.df import output
In [16]: @output(['k', 'v'], ['string', 'int'])
    ...: def h(row):
    ...:     import json
    ...:     for k, v in json.loads(row.json).items():
    ...:         yield k, v
    ...:
In [21]: df.apply(h, axis=1).head(4)
   k  v
0  a  1
1  b  2
2  c  4
3  b  3
```

- 线上生产，通过 `persist()` 方法将结果存回MaxCompute表。

```
In [14]: from odps.df import output
In [16]: @output(['k', 'v'], ['string', 'int'])
    ...: def h(row):
    ...:     import json
    ...:     for k, v in json.loads(row.json).items():
    ...:         yield k, v
    ...:
In [21]: df.apply(h, axis=1).persist('my_table')
```

3.10. Spark On MaxCompute访问Phoenix数据

本文主要为您介绍使用Spark连接Phoenix，并将HBase中的数据写入到MaxCompute的实践方案。

背景信息

Phoenix是HBase提供的SQL层，主要为了解决高并发、低延迟、简单查询等场景。为了满足用户在Spark On MaxCompute环境下访问Phoenix的数据需求，本文从Phoenix表的创建与数据写入，再到IDEA上的Spark代码编写以及DataWorks上代码的冒烟测试，完整的描述了Spark On MaxCompute访问Phoenix的数据实践方案。

前提条件

在实践之前，您需要提前做好以下准备工作：

- 已开通MaxCompute服务并创建MaxCompute项目。详情请参见开通MaxCompute服务和创建MaxCompute项目。
- 已开通DataWorks服务。详情请参见DataWorks购买指导。
- 已开通HBase服务，详情请参见HBase购买指导。

> **说明** 本实践内容是以HBase 1.1版本为例。实际开发中，您也可以配套其他HBase版本。

- 已下载并安装Phoenix 4.12.0版本。详情请参见HBase SQL(Phoenix) 4.x使用说明。

> **说明** HBase 1.1版本对应的Phoenix版本为4.12.0，实际开发过程中需要注意版本对应关系。

- 已开通专有网络VPC，并配置了HBase集群安全组和白名单。详情请参见专有网络连接方案。

> **说明** 本实践中，HBase是在VPC的网络环境下，所以安全组开放端口为2181、10600、16020，对应MaxCompute IP的白名单为 `100.104.0.0/16` 。

操作步骤

1. 进入Phoenix的 `bin` 目录，执行如下示例命令启动Phoenix客户端。

```
./sqlline.py hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com:2181,hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com:2181,hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com:2181
```

> **说明** `hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com:2181,hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com:2181,hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com:2181` 为ZooKeeper的连接地址。您可以通过登录HBase控制台，在HBase集群实例详情页的**数据库连接**页面获取ZooKeeper的连接地址。

2. 在Phoenix客户端，执行如下语句创建表users，并插入数据。

```
CREATE TABLE IF NOT EXISTS users(
id UNSIGNED_INT,
username char(50),
password char(50)
CONSTRAINT my_ph PRIMARY KEY (id));
UPSERT INTO users(id,username,password) VALUES (1,'kongxx','Letmein');
```

> **说明** Phoenix语法详情，请参见HBase SQL(Phoenix) 入门。

3. 在Phoenix客户端，执行如下语句查看users表的数据。

```
select * from users;
```

4. 在IDEA编译工具编写Spark代码逻辑并打包。
   i. 使用Scala编程语言编写Spark代码逻辑进行测试

   在IDEA中按照对应的Pom文件配置本地开发环境。您可以先使用公网连接地址进行测试，待代码逻辑验证成功后再调整代码示例中spark.hadoop.odps.end.point参数内容。公网连接地址请通过登录HBase控制台，在HBase集群实例详情页的**数据库连接**页面获取。具体代码如下：

```
package com.phoenix
import org.apache.hadoop.conf.Configuration
import org.apache.spark.sql.SparkSession
import org.apache.phoenix.spark._
/**
  * 本示例适用于Phoenix 4.x版本。
  */
object SparkOnPhoenix4xSparkSession {
  def main(args: Array[String]): Unit = {
    //HBase集群的ZooKeeper连接地址。
    val zkAddress = hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com:2181,hb-2zecxg2ltnpeg8me4-master*-***.hbase.
rds.aliyuncs.com:2181,hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com:2181
    //Phoenix侧的表名。
    val phoenixTableName = users
    //Spark侧的表名。
    val ODPSTableName = users_phoenix
    val sparkSession = SparkSession
      .builder()
      .appName("SparkSQL-on-MaxCompute")
      .config("spark.sql.broadcastTimeout", 20 * 60)
      .config("spark.sql.crossJoin.enabled", true)
      .config("odps.exec.dynamic.partition.mode", "nonstrict")
      // 需设置spark.master为local[N]才能直接运行，N为并发数。
      //.config("spark.master", "local[4]")
      .config("spark.hadoop.odps.project.name", "***")
      .config("spark.hadoop.odps.access.id", "***")
      .config("spark.hadoop.odps.access.key", "***")
      //.config("spark.hadoop.odps.end.point", "http://service.cn.maxcompute.aliyun.com/api")
      .config("spark.hadoop.odps.end.point", "http://service.cn-beijing.maxcompute.aliyun-inc.com/api")
      .config("spark.sql.catalogImplementation", "odps")
      .getOrCreate()
    var df = sparkSession.read.format("org.apache.phoenix.spark").option("table", phoenixTableName).option("zkUrl",zkAdd
ress).load()
    df.show()
    df.write.mode("overwrite").insertInto(ODPSTableName)
  }
}
```

对应的POM文件如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at
    http://www.apache.org/licenses/LICENSE-2.0
  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <properties>
        <spark.version>2.3.0</spark.version>
        <cupid.sdk.version>3.3.8-public</cupid.sdk.version>        <scala.version>2.11.8</scala.version>
        <scala.binary.version>2.11</scala.binary.version>
        <phoenix.version>4.12.0-HBase-1.1</phoenix.version>
    </properties>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>Spark-Phonix</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <dependencies>
        <dependency>
            <groupId>org.jpmml</groupId>
            <artifactId>pmml-model</artifactId>
            <version>1.3.8</version>
        </dependency>
        <dependency>
            <groupId>org.jpmml</groupId>
```

```
            <artifactId>pmml-evaluator</artifactId>
            <version>1.3.10</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_${scala.binary.version}</artifactId>
            <version>${spark.version}</version>
            <scope>provided</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.scala-lang</groupId>
                    <artifactId>scala-library</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>org.scala-lang</groupId>
                    <artifactId>scalap</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-sql_${scala.binary.version}</artifactId>
            <version>${spark.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-mllib_${scala.binary.version}</artifactId>
            <version>${spark.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming_${scala.binary.version}</artifactId>
            <version>${spark.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.aliyun.odps</groupId>
            <artifactId>cupid-sdk</artifactId>
            <version>${cupid.sdk.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.aliyun.phoenix</groupId>
            <artifactId>ali-phoenix-core</artifactId>
            <version>4.12.0-AliHBase-1.1-0.8</version>
            <exclusions>
                <exclusion>
                    <groupId>com.aliyun.odps</groupId>
                    <artifactId>odps-sdk-mapred</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>com.aliyun.odps</groupId>
                    <artifactId>odps-sdk-commons</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>com.aliyun.phoenix</groupId>
            <artifactId>ali-phoenix-spark</artifactId>
            <version>4.12.0-AliHBase-1.1-0.8</version>
            <exclusions>
                <exclusion>
                    <groupId>com.aliyun.phoenix</groupId>
                    <artifactId>ali-phoenix-core</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
```

```
                <version>2.4.3</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <minimizeJar>false</minimizeJar>
                            <shadedArtifactAttached>true</shadedArtifactAttached>
                            <artifactSet>
                                <includes>
                                    <!-- Include here the dependencies you
                                        want to be packed in your fat jar -->
                                    <include>*:*</include>
                                </includes>
                            </artifactSet>
                            <filters>
                                <filter>
                                    <artifact>*:*</artifact>
                                    <excludes>
                                        <exclude>META-INF/*.SF</exclude>
                                        <exclude>META-INF/*.DSA</exclude>
                                        <exclude>META-INF/*.RSA</exclude>
                                        <exclude>**/log4j.properties</exclude>
                                    </excludes>
                                </filter>
                            </filters>
                            <transformers>
                                <transformer
                                        implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer">
                                    <resource>reference.conf</resource>
                                </transformer>
                                <transformer
                                        implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer">
                                    <resource>META-INF/services/org.apache.spark.sql.sources.DataSourceRegister</resourc
e>
                                </transformer>
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>net.alchim31.maven</groupId>
                <artifactId>scala-maven-plugin</artifactId>
                <version>3.3.2</version>
                <executions>
                    <execution>
                        <id>scala-compile-first</id>
                        <phase>process-resources</phase>
                        <goals>
                            <goal>compile</goal>
                        </goals>
                    </execution>
                    <execution>
                        <id>scala-test-compile-first</id>
                        <phase>process-test-resources</phase>
                        <goals>
                            <goal>testCompile</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

ii. 在IDEA中将代码以及依赖文件打成JAR包，并通过MaxCompute客户端上传至MaxCompute项目环境中。详情请参见添加资源。

> **说明** 由于DatadWork界面方式上传JAR包有50 MB的限制，因此采用MaxCompute客户端上传JAR包。

5. 在DataWorks上进行冒烟测试。

i. 按照如下建表语句，在DataWorks上创建MaxCompute表。详情请参见创建MaxCompute表。

```
CREATE TABLE IF NOT EXISTS users_phoenix
(
    id       INT   ,
    username STRING,
    password STRING
) ;
```

ii. 在DataWorks上，选择对应的MaxCompute项目环境，将上传的JAR包添加到数据开发环境中。详情请参见创建JAR资源。

iii. 新建ODPS Spark，并设置任务参数。详情请参见创建ODPS Spark节点。

提交Spark任务的配置参数如下图所示。

对应的spark.hadoop.odps.cupid.vpc.domain.list参数如下所示，请您根据个人HBase集群节点进行配置。

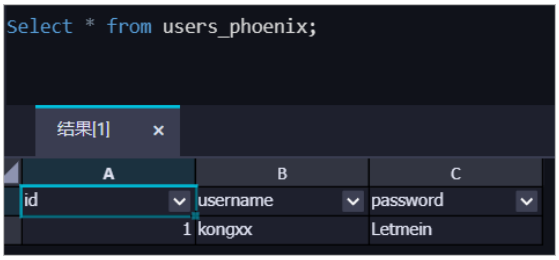
```
{
  "regionId":"cn-beijing",
  "vpcs":[
    {
      "vpcId":"vpc-2zeaeq21********0exox",
      "zones":[
        {
          "urls":[
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":2181
            },
  {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":2181
            },
  {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":2181
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16000
            },
  {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16000
            },
  {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16000
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
          ]
        }
      ]
    }
  ]
}
```


iv. 单击图标开始冒烟测试。

6. 冒烟测试成功后，在临时查询节点中执行如下查询语句。

```
select * from users_phoenix;
```

可以看到数据已经写入MaxCompute的表中。



3.11. Spark on MaxCompute如何访问HBase

本文为您介绍Spark on MaxCompute访问云数据库HBase的配置方法。

背景信息

Spark on MaxCompute可以访问位于阿里云VPC内的实例（ECS、HBase、RDS等）。MaxCompute底层网络和外网默认是隔离的，Spark on MaxCompute提供了一种方案通过配置 `spark.hadoop.odps.cupid.vpc.domain.list` 来访问阿里云的VPC网络环境的HBase。HBase标准版和增强版（Lindorm）的配置不同，详情如下。

- Spark on MaxCompute访问阿里云HBase标准版。
- Spark on MaxCompute访问阿里云HBase增强版（Lindorm）。

前提条件

在实践之前，您需要提前做好以下准备工作：

- 已开通MaxCompute服务并创建MaxCompute项目。详情请参见开通MaxCompute服务和创建MaxCompute项目。
- 已开通DataWorks服务。详情请参见DataWorks购买指导。
- 已开通HBase服务，详情请参见HBase购买指导。
- 已开通专有网络VPC，并配置了HBase集群安全组和白名单。详情请参见专有网络连接方案。

> 说明
> - HBase标准版安全组开放端口为2181、10600、16020，对应MaxCompute IP的白名单为 `100.104.0.0/16` 。
> - HBase增强版（Lindorm）版安全组开放端口为30020、10600、16020，对应MaxCompute IP的白名单为 `100.104.0.0/16` 。

Spark on MaxCompute访问阿里云HBase标准版

1. 在HBase客户端，执行如下语句创建HBase表。

```
create 'test','cf'
```

> 说明 更多HBase使用命令，请参见HBase Shell使用介绍。

2. 在IDEA编译工具编写Spark代码逻辑并打包。

i. 使用Scala编程语言，按如下代码示例编写Spark代码逻辑。

```
object App {
  def main(args: Array[String]) {
    val spark = SparkSession
      .builder()
      .appName("HbaseTest")
      .config("spark.sql.catalogImplementation", "odps")
      .config("spark.hadoop.odps.end.point","http://service.cn.maxcompute.aliyun.com/api")
      .config("spark.hadoop.odps.runtime.end.point","http://service.cn.maxcompute.aliyun-inc.com/api")
      .getOrCreate()
    val sc = spark.sparkContext
    val config = HBaseConfiguration.create()
    //HBase集群的ZooKeeper连接地址。
    val zkAddress = "hb-2zecxg2ltnpeg8me4-master*-***:2181,hb-2zecxg2ltnpeg8me4-master*-***:2181,hb-2zecxg2ltnpeg8me4-ma
ster*-***:2181"
    config.set(HConstants.ZOOKEEPER_QUORUM, zkAddress);
    val jobConf = new JobConf(config)
    jobConf.setOutputFormat(classOf[TableOutputFormat])
    //HBase表名。
    jobConf.set(TableOutputFormat.OUTPUT_TABLE,"test")
    try{
      import spark._
      //将MaxCompute表中数据写入HBase表。以下查询MaxCompute表语句以常量为例，实际开发环境需替换。
      spark.sql("select '7', 88 ").rdd.map(row => {
        val name= row(0).asInstanceOf[String]
        val id = row(1).asInstanceOf[Integer]
        val put = new Put(Bytes.toBytes(id))
        put.addColumn(Bytes.toBytes("cf"), Bytes.toBytes(id), Bytes.toBytes(name))
        (new ImmutableBytesWritable, put)
      }).saveAsHadoopDataset(jobConf)
    } finally {
      sc.stop()
    }
  }
}
```

> **说明** 您可以通过登录HBase控制台，在HBase集群实例详情页的**数据库连接**页面获取ZooKeeper的连接地址。

对应的HBase依赖文件如下。

```
<dependency>
      <groupId>org.apache.hbase</groupId>
      <artifactId>hbase-mapreduce</artifactId>
      <version>2.0.2</version>
    </dependency>
     <dependency>
      <groupId>com.aliyun.hbase</groupId>
      <artifactId>alihbase-client</artifactId>
      <version>2.0.5</version>
    </dependency>
```

ii. 在IDEA中将代码以及依赖文件打成JAR包，并通过MaxCompute客户端上传至MaxCompute项目环境中。详情请参见添加资源。

> **说明** 由于DatadWork界面方式上传JAR包有50 MB的限制，因此采用MaxCompute客户端上传JAR包。

3. 在DataWorks上创建ODPS Spark节点并配置。

i. 在DataWorks上，选择对应的MaxCompute项目环境，将上传的JAR包添加到数据开发环境中。详情请参见创建JAR资源。

ii. 新建ODPS Spark，并设置任务参数。详情请参见创建ODPS Spark节点。

提交Spark任务的配置参数如下图所示。

对应的spark.hadoop.odps.cupid.vpc.domain.list参数如下所示，请您根据个人HBase集群节点进行配置。

```
{
  "regionId":"cn-beijing",
  "vpcs":[
    {
      "vpcId":"vpc-2zeaeq21mb1dmkqh0exox",
      "zones":[
        {
          "urls":[
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":2181
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16000
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":2181
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16000
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":2181
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16000
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-cor*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-cor*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-cor*-***.hbase.rds.aliyuncs.com",
              "port":16020
            }
          ]
        }
      ]
    }
  ]
}
```

Spark on MaxCompute访问阿里云HBase增强版（Lindorm）

1. 在HBase客户端，执行如下语句创建HBase表。

```
create 'test','cf'
```

> 说明 更多HBase使用命令，请参见HBase Shell使用介绍。

2. 在IDEA编译工具编写Spark代码逻辑并打包。

i. 使用Scala编程语言，按照如下示例编写Spark代码逻辑。

```
object McToHbase {
  def main(args: Array[String]) {
    val spark = SparkSession
      .builder()
      .appName("spark_sql_ddl")
      .config("spark.sql.catalogImplementation", "odps")
      .config("spark.hadoop.odps.end.point","http://service.cn.maxcompute.aliyun.com/api")
      .config("spark.hadoop.odps.runtime.end.point","http://service.cn.maxcompute.aliyun-inc.com/api")
      .getOrCreate()
      val sc = spark.sparkContext
    try{
      //将MaxCompute表中数据写入HBase表。以下查询MaxCompute表语句以常量为例，实际开发环境需替换。
      spark.sql("select '7', 'long'").rdd.foreachPartition { iter =>
        val config = HBaseConfiguration.create()
        //ZooKeeper集群的连接地址(VPC内网地址)
        config.set("hbase.zookeeper.quorum", "<ZooKeeper连接地址>:30020");
        import spark._
        //HBase用户名和密码
        config.set("hbase.client.username", "<用户名>");
        config.set("hbase.client.password", "<密码>");
        //HBase表名
        val tableName = TableName.valueOf( "test")
        val conn = ConnectionFactory.createConnection(config)
        val table = conn.getTable(tableName);
        val puts = new util.ArrayList[Put]()
        iter.foreach(
          row => {
            val id = row(0).asInstanceOf[String]
            val name = row(1).asInstanceOf[String]
            val put = new Put(Bytes.toBytes(id))
            put.addColumn(Bytes.toBytes("cf"), Bytes.toBytes(id), Bytes.toBytes(name))
            puts.add(put)
            table.put(puts)
          }
        )
      }
  } finally {
    sc.stop()
  }
  }
}
```

> **说明** 您可以通过登录HBase控制台，在HBase集群实例详情页的**数据库连接**页面获取ZooKeeper的连接地址以及HBase用户名和密码。

对应的HBase依赖文件如下。

```
<dependency>
      <groupId>com.aliyun.hbase</groupId>
      <artifactId>alihbase-client</artifactId>
      <version>2.0.8</version>
    </dependency>
```

ii. 在IDEA中将代码以及依赖文件打成JAR包，并通过MaxCompute客户端上传至MaxCompute项目环境中。详情请参见添加资源。

> **说明** 由于DatadWork界面方式上传JAR包有50 MB的限制，因此采用MaxCompute客户端上传JAR包。

3. 在DataWorks上创建ODPS Spark节点并配置。
   i. 在DataWorks上，选择对应的MaxCompute项目环境，将上传的JAR包添加到数据开发环境中。详情请参见创建JAR资源。
   ii. 新建ODPS Spark，并设置任务参数。详情请参见创建ODPS Spark节点。

   提交Spark任务的配置参数如下图所示。

对应的spark.hadoop.odps.cupid.vpc.domain.list参数如下所示，请您根据个人HBase集群节点进行配置。

```
{
  "regionId":"cn-beijing",
  "vpcs":[
    {
      "vpcId":"vpc-2zeaeq21mb1dmkqh0exox",
      "zones":[
        {
          "urls":[
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":30020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16000
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":30020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16000
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":30020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16000
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-cor*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-cor*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
            {
              "domain":"hb-2zecxg2ltnpeg8me4-cor*-***.hbase.rds.aliyuncs.com",
              "port":16020
            },
             {"domain":"172.*.*.10","port":16000}
          ]
        }
      ]
    }
  ]
}
```


说明 172.*.*.10为HBase增强版Java API访问地址。必须采用IP的形式，您可以在所属服务器使用 `ping` 命令获取。

3.12. MaxCompute在电商场景中如何进行漏斗模型分析

本文以电商场景为例，为您介绍如何使用MaxCompute进行离线计算以及漏斗模型展示。

背景信息

漏斗模型是通过产品各项数据的转化率来判断产品运营情况的工具。转化漏斗则是通过各阶段数据的转化，来判断产品在哪一个环节出了问题，然后不断优化产品。电商漏斗模型主要展示用户购买商品的路径，即从浏览商品到支付订单的每一个环节的转化。本文将展示从用户浏览、点击、购买环节做漏斗分析及展示。

前提条件

- 已开通日志服务SLS。详情请参见日志服务SLS购买指导。
- 已开通MaxCompute服务并创建MaxCompute项目。详情请参见开通MaxCompute服务和创建MaxCompute项目。
- 已开通DataWorks服务。详情请参见DataWorks购买指导。
- 已开通Quick BI服务。详情请参见Quick BI购买指导。

操作步骤

1. 通过日志服务采集日志数据。

   日志服务采集日志数据，详情请参见数据采集概述。本文以测试数据为例演示，如需下载测试数据，请您单击测试数据下载。

2. 将采集的日志数据迁移至MaxCompute。

   采集的日志数据迁移至MaxCompute，详情请参见日志数据迁移至MaxCompute。

3. 使用MaxCompute构建离线计算数据模型。

   i. 按照如下建表语句，在DataWorks数据开发界面创建数据引入层（ODS）表ods_user_trace_data。

```
--以dt作为时间分区，单位为天。
CREATE TABLE IF NOT EXISTS ods_user_trace_data
(
    md5                     STRING COMMENT '用户uid的md5值前8位',
    uid                     STRING COMMENT '用户uid',
    ts                      BIGINT COMMENT '用户操作时间戳',
    ip                      STRING COMMENT 'ip地址',
    status                  BIGINT COMMENT '服务器返回状态码',
    bytes                   BIGINT COMMENT '返回给客户端的字节数',
    device_brand            STRING COMMENT '设备品牌',
    system_type             STRING COMMENT '系统类型，Android、IOS、ipad、Windows_phone',
    customize_event         STRING COMMENT '自定义事件：登录/退出/购买/注册/点击/后台/切换用户/浏览/评论',
    use_time                BIGINT COMMENT 'APP单次使用时长，当事件为退出、后台、切换用户时有该项',
    customize_event_content STRING COMMENT '用户关注内容信息，在customize_event为浏览和评论时，包含该列'
)
PARTITIONED BY
(
    dt STRING
);
```

   > 说明
   > - 以上建表语句中的字段是根据测试数据构建的。如何在DataWorks创建表，请参见创建ODPS SQL节点。
   > - 更多ODS说明，请参见数据引入层（ODS）。

   ii. 按照如下建表语句，在DataWorks数据开发界面创建明细粒度事实层（DWD）表dw_user_trace_data。

```
--以dt作为时间分区，单位为天。
CREATE TABLE IF NOT EXISTS dw_user_trace_data
(
    uid                     STRING COMMENT '用户uid',
    device_brand            STRING COMMENT '设备品牌',
    system_type             STRING COMMENT '系统类型，Android、IOS、ipad、Windows_phone',
    customize_event         STRING COMMENT '自定义事件：登录/退出/购买/注册/点击/后台/切换用户/浏览/评论',
    use_time                BIGINT COMMENT 'APP单次使用时长，当事件为退出、后台、切换用户时有该项',
    customize_event_content STRING COMMENT '用户关注内容信息，在customize_event为浏览和评论时，包含该列'
)
PARTITIONED BY
(
    dt STRING
);
```

   > 说明 更多DWD说明，请参见明细粒度事实层（DWD）。

iii. 按照如下建表语句，在DataWorks数据开发界面创建数据应用层（ADS）表rpt_user_trace_data。

```
 --以dt作为时间分区，单位为天。
CREATE TABLE IF NOT EXISTS rpt_user_trace_data
(
    browse      STRING COMMENT '浏览量',
    click       STRING COMMENT '点击量',
    purchase    STRING COMMENT '购买量',
    browse_rate STRING COMMENT '浏览转化率',
    click_rate  STRING COMMENT '点击转化量'
)
PARTITIONED BY
(
    dt STRING
);
```

> 说明 更多ADS说明，请参见数仓分层。

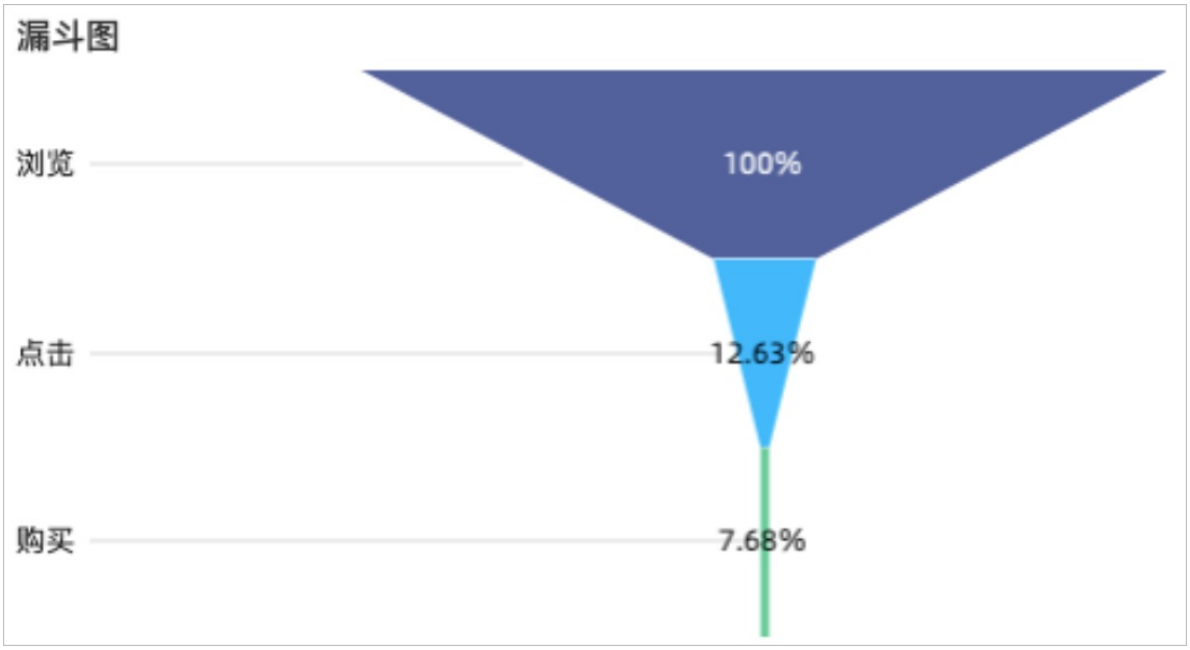
iv. 按照如下SQL语句，在DataWorks数据开发界面编写业务代码逻辑。

```
insert OVERWRITE table rpt_user_trace_data PARTITION (dt=${bdp.system.bizdate})
SELECT browse as 浏览量
      ,click as 点击量
      ,purchase as 购买量
      ,concat(round((click/browse)*100,2),'%') as 点击转化率
      ,concat(round((purchase/click)*100,2),'%') as 购买转化率
from
(SELECT dt,count(1) browse from dw_user_trace_data where customize_event='browse'
 and dt = ${bdp.system.bizdate} group by dt) a
left JOIN
(select dt,count(1) click from dw_user_trace_data where customize_event='click'
 and dt = ${bdp.system.bizdate} group by dt) b
on a.dt=b.dt
left JOIN
(select dt,count(1) purchase from dw_user_trace_data where customize_event='purchase'
and dt = ${bdp.system.bizdate} group by dt)c
on  a.dt=c.dt
;
```

> 说明 用户路径是从浏览到点击再到购买。各个环节的转化率为从当一个页面进入下一页面的人数比率，例如点击转换率=进入点击页面的人数/浏览页面人数。

4. 数据可视化展示。

通过Quick BI创建网站用户分析画像的仪表板，实现该数据表的可视化，详情请参见云数据源MaxCompute和漏斗图。



3.13. MaxCompute如何访问Hologres

本文为您介绍MaxCompute如何访问Hologres。

背景信息

Hologres是阿里云实时交互式分析产品。Hologres具备高并发地实时写入和查询数据的能力，同时支持数据无需迁移就能高性能加速分析MaxCompute数据，通过联邦分析Hologres实时数据与MaxCompute离线数据，实现离线实时一体化的数据仓库产品解决方案。您可以使用MaxCompute和Hologres的组合方案，来满足大规模离线分析、实时运营分析、交互式查询等多业务场景。

- MaxCompute SQL外部表的方式访问Hologres，详情请参见MaxCompute SQL外部表方式访问Hologres。
- MaxCompute Spark方式访问Hologres，如下所示。
  - MaxCompute Spark访问Hologres：Local提交模式。
  - MaxCompute Spark访问Hologres：Cluster提交模式。
  - MaxCompute Spark访问Hologres：DataWorks提交模式。

前提条件

- 已准备MaxCompute相应的环境。
  - 已开通MaxCompute服务并创建Hologres外部表的目标MaxCompute项目。

    开通MaxCompute服务以及创建MaxCompute项目的操作指导，请参见开通MaxCompute和创建MaxCompute项目。
  - 已安装MaxCompute客户端。

    安装MaxCompute客户端的操作指导，请参见安装并配置MaxCompute客户端。
  - 已搭建MaxCompute Spark开发环境。

    本实践的MaxCompute Spark开发环境搭建在Linux操作系统下，使用的是Spark2.4.5发布包。具体的操作指导，请参见搭建MaxCompute Spark开发环境。
- 已开通DataWorks服务。

  开通DataWorks服务的操作指导，请参见开通DataWorks。
- 已开通Hologres服务并连接HoloWeb。

  开通Hologres并连接HoloWeb的操作指导，请参见开通Hologres和连接HoloWeb。
- 已下载PostgreSQL的JDBC驱动包。

  本实践使用的是 `postgresql-42.2.16.jar` 驱动包，并且将该JAR包存放在Linux的*/home/postgreSQL*路径下。您可以通过PostgreSQL驱动官网下载。

MaxCompute SQL外部表方式访问Hologres

1. 在Hologres管理控制台，选择目标实例并创建Hologres数据库 `mc_db_holo` 。

   创建Hologres数据库的操作指导，请参见创建Hologres数据库。

2. 在**HoloWeb**开发界面的 `mc_db_holo` 数据库下，执行如下语句创建Hologres表 `mc_sql_holo` 并插入数据。

   创建Hologres表的操作指导，请参见创建Hologres表。

```
CREATE TABLE mc_sql_holo(
        id INTEGER,
        name TEXT
);
INSERT INTO mc_sql_holo VALUES
        (1,'zhangsan'),
        (2,'lisi'),
        (3,'wangwu')
;
```

3. 在RAM访问控制台创建RAM角色 `AliyunOdpsHoloRole` 并修改信任策略配置内容。

   创建RAM角色并修改信任策略配置内容的操作指导，请参见创建RAM角色。

> **说明** 本实践创建的RAM角色可信实体类型为**阿里云账号**。

4. 添加 `AliyunOdpsHoloRole` RAM角色至Hologres实例并授权。

   添加RAM角色至Hologres实例并授权的操作指导，请参见添加RAM角色至Hologres实例并授权。

5. 在MaxCompute客户端，按照如下语句创建Hologres外部表 `mc_externaltable_holo` 。

```
create external table if not exists mc_externaltable_holo
(
    id int ,
    name string
)
stored by 'com.aliyun.odps.jdbc.JdbcStorageHandler'
with serdeproperties (
  'odps.properties.rolearn'='acs:ram::13969******5947:role/aliyunodpsholorole')
LOCATION 'jdbc:postgresql://hgprecn-cn-2r42******-cn-hangzhou-internal.hologres.aliyuncs.com:80/mc_db_holo?currentSchema=pub
lic&useSSL=false&table=mc_sql_holo/'
TBLPROPERTIES (
  'mcfed.mapreduce.jdbc.driver.class'='org.postgresql.Driver',
  'odps.federation.jdbc.target.db.type'='holo',
  'odps.federation.jdbc.colmapping'='id:id,name:name'
);
```

> **说明** 创建外部表参数说明，请参见Hologres外部表。

6. 创建完成后，在MaxCompute客户端执行如下语句，获取Hologres外部表的数据。

```
set odps.sql.split.hive.bridge=true;
set odps.sql.hive.compatible=true;
set odps.sql.jobconf.odps2=true;
set odps.sql.jobconf.odps2.enforce=true;
select * from mc_externaltable_holo limit 10;
```

> **说明** SET操作的属性说明，详情请参见SET操作。

结果如下所示：

```
+----+----------+
| id | name     |
+----+----------+
| 1  | zhangsan |
| 2  | lisi     |
| 3  | wangwu   |
+----+----------+
```

7. 在MaxCompute客户端执行如下语句，写数据至Hologres外部表。

```
set odps.sql.split.hive.bridge=true;
set odps.sql.hive.compatible=true;
insert into mc_externaltable_holo values (4,'alice');
```

8. 在**HoloWeb**开发界面，查询Hologres表 `mc_sql_holo` 中数据。


```
select * from mc_sql_holo;
```

MaxCompute Spark访问Hologres：Local提交模式

1. 在**HoloWeb**开发界面的 `mc_db_holo` 数据库下，执行如下语句创建Hologres表 `mc_jdbc_holo` 。

   创建Hologres表的操作指导，请参见创建Hologres表。

```
CREATE TABLE mc_jdbc_holo(
        id INTEGER,
        name TEXT
);
```

2. 在Linux操作系统的*/home/pythoncode*路径下，新建Python文件 `holo_local.py` 。

   Python脚本内容如下所示：

```
from pyspark.sql import SparkSession
spark = SparkSession \
            .builder \
            .appName("Spark_local") \
            .config("spark.eventLog.enabled","false") \
            .getOrCreate()
jdbcDF = spark.read.format("jdbc"). \
            options(
            url='jdbc:postgresql://hgprecn-cn-2r42******-cn-hangzhou.hologres.aliyuncs.com:80/mc_db_holo',
            dbtable='mc_jdbc_holo',
            user='LTAI5tJE8fuerxXdPPB****',
            password='Okr2kbBKueR3uRaHaBiUHw4r6****',
            driver='org.postgresql.Driver').load()
jdbcDF.printSchema()
```

   脚本内容参数说明如下所示：

   - url：使用Spark的JDBC连接方式，驱动为 `postgresql` 。
     - hgprecn-cn-2r42******-cn-hangzhou.hologres.aliyuncs.com:80：Hologres实例的公网访问域名。获取方式，请参见实例配置。
     - mc_db_holo：连接的Hologres数据库名称。本实践命名为mc_db_holo。
   - dbtable：Hologres源表名称。本实践命名为mc_jdbc_holo。
   - user：阿里云账号或RAM用户的AccessKey ID。您可以进入AccessKey管理页面获取AccessKey ID。
   - password：AccessKey ID对应的AccessKey Secret。您可以进入AccessKey管理页面获取AccessKey Secret。
   - driver：PostgreSQL驱动，固定值为org.postgresql.Driver。

3. 在Linux系统任意目录下，使用**spark-submit**提交本地作业。

```
spark-submit --master local --driver-class-path /home/postgreSQL/postgresql-42.2.16.jar --jars /home/postgreSQL/postgresql-4
2.2.16.jar /home/pythoncode/holo_local.py
```

查看Spark打印日志，打印Schema信息与Hologres中创建的 `mc_jdbc_holo` 表一致，即访问成功。

```
22/04/02 16:41:39 INFO StateStoreCoordinatorRef: Registered StateStoreCoordinator endpoint
root
 |-- id: integer (nullable = true)
 |-- name: string (nullable = true)
```

MaxCompute Spark访问Hologres：Cluster提交模式

1. 在**HoloWeb**开发界面的 `mc_db_holo` 数据库下，执行如下语句创建Hologres表 `mc_jdbc_holo` 。

   创建Hologres表的操作指导，请参见创建Hologres表。

   ```
   CREATE TABLE mc_jdbc_holo(
           id INTEGER,
           name TEXT
   );
   ```

2. 在Linux操作系统的*/home/pythoncode*路径下，新建Python文件 `holo_yarncluster.py` 。

   Python脚本内容如下所示：

   ```
   from pyspark.sql import SparkSession
   spark = SparkSession \
               .builder \
               .appName("Spark_yarn") \
               .getOrCreate()
   jdbcDF = spark.read.format("jdbc"). \
               options(
               url='jdbc:postgresql://hgprecn-cn-2r42******-cn-hangzhou-internal.hologres.aliyuncs.com:80/mc_db_holo',
               dbtable='mc_jdbc_holo',
               user='LTAI5tJE8fuerxXdPPB****',
               password='Okr2kbBKueR3uRaHaBiUHw4r6****',
               driver='org.postgresql.Driver').load()
   jdbcDF.printSchema()
   ```

   脚本内容参数说明如下所示：

   - url：使用Spark的JDBC连接方式，驱动为postgresql。
     - hgprecn-cn-2r42******-cn-hangzhou-internal.hologres.aliyuncs.com:80：Hologres实例的经典网络访问域名。获取方式，请参见实例配置。
     - mc_db_holo：连接的Hologres数据库名称。本实践命名为mc_db_holo。
   - dbtable：Hologres源表名称。本实践命名为mc_jdbc_holo。
   - user：阿里云账号或RAM用户的AccessKey ID。您可以进入AccessKey管理页面获取AccessKey ID。
   - password：AccessKey ID对应的AccessKey Secret。您可以进入AccessKey管理页面获取AccessKey Secret。
   - driver：PostgreSQL驱动，固定值为org.postgresql.Driver。

3. 配置MaxCompute Spark客户端解压目录*/home/spark2.4.5/spark-2.4.5-odps0.33.2/conf*下的spark-defaults.conf文件。

   ```
   #需配置以下配置项
   spark.hadoop.odps.project.name = <MaxCompute_Project_Name>
   spark.hadoop.odps.end.point = <Endpoint>
   spark.hadoop.odps.runtime.end.point = <VPC_Endpoint>
   spark.hadoop.odps.access.id = <AccessKey_ID>
   spark.hadoop.odps.access.key = <AccessKey_Secret>
   spark.hadoop.odps.cupid.trusted.services.access.list = <Hologres_Classic_Network>
   #以下内容保持不变
   spark.master = yarn-cluster
   spark.driver.cores = 2
   spark.driver.memory = 4g
   spark.dynamicAllocation.shuffleTracking.enabled = true
   spark.dynamicAllocation.shuffleTracking.timeout = 20s
   spark.dynamicAllocation.enabled = true
   spark.dynamicAllocation.maxExecutors = 10
   spark.dynamicAllocation.initialExecutors = 2
   spark.executor.cores = 2
   spark.executor.memory = 8g
   spark.eventLog.enabled = true
   spark.eventLog.overwrite = true
   spark.eventLog.dir = odps://admin_task_project/cupidhistory/sparkhistory
   spark.sql.catalogImplementation = hive
   spark.sql.sources.default = hive
   ```

   配置文件参数说明如下所示：

   - MaxCompute_Project_Name：待访问MaxCompute项目的名称。

此处为MaxCompute项目名称，非工作空间名称。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。

- AccessKey_ID：具备目标MaxCompute项目访问权限的AccessKey ID。

  您可以进入AccessKey管理页面获取AccessKey ID。
- AccessKey_Secret：AccessKey ID对应的AccessKey Secret。

  您可以进入AccessKey管理页面获取AccessKey Secret。
- Endpoint：MaxCompute项目所属地域的外网Endpoint。

  各地域的外网Endpoint信息，请参见各地域Endpoint对照表（外网连接方式）。
- VPC_Endpoint：MaxCompute项目所属地域的VPC网络的Endpoint。

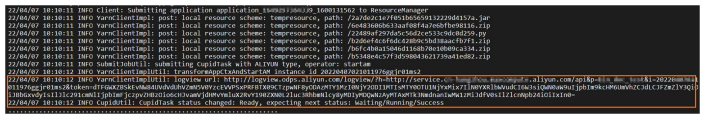
  各地域的VPC网络Endpoint信息，请参见各地域Endpoint对照表（阿里云VPC网络连接方式）。
- Hologres_Classic_Network：Hologres经典网络类型。配置此项主要是为了在MaxCompute安全运行沙箱环境中，开启到对应Hologres实例的网络策略，否则MaxCompute集群无法访问外部服务。

4. 在Linux系统任意目录下，使用**spark-submit**提交作业。

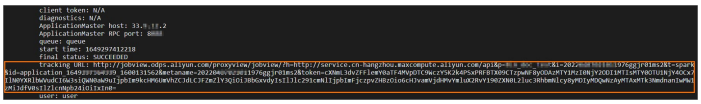
```
spark-submit --master yarn-cluster --driver-class-path /home/postgreSQL/postgresql-42.2.16.jar --jars /home/postgreSQL/postgresql-42.2.16.jar /home/pythoncode/holo_yarncluster.py
```

提交后可以查看Spark打印日志，作业正常完成会打印作业的Logview以及Spark-UI的Jobview链接地址，可供开发者进一步诊断作业。

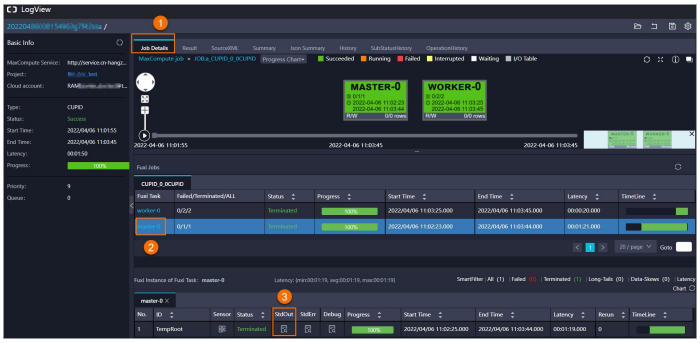
- 作业的Logview链接地址。



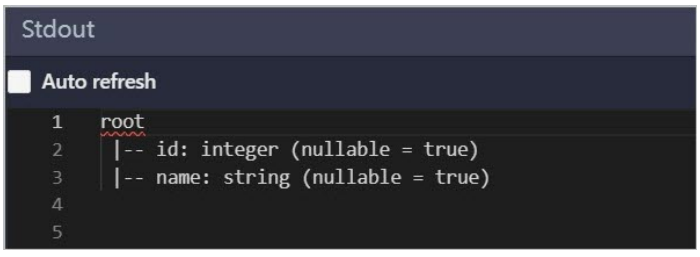
- Spark-UI的Jobview链接地址。



5. 打开Logview链接，如果作业执行状态为**success**，选择**Job Details > master-0 > StdOut**，查看`jdbcDF.printSchema()`的返回结果。



查看**Stdout**，打印的Schema信息与Hologres中创建的`mc_jdbc_holo`表一致，即访问成功。



> **说明** 您也可以打开Spark-UI的Jobview链接地址，进行作业的查看与诊断。

MaxCompute Spark访问Hologres：DataWorks提交模式

1. 在**HoloWeb**开发界面的`mc_db_holo`数据库下，执行如下语句创建Hologres表`mc_jdbc_holo`。

   创建Hologres表的操作指导，请参见创建Hologres表。

```
CREATE TABLE mc_jdbc_holo(
        id INTEGER,
        name TEXT
);
```

2. 配置MaxCompute Spark客户端解压目录*/home/spark2.4.5/spark-2.4.5-odps0.33.2/conf*下的spark-defaults.conf文件。

```
#需配置以下配置项
spark.hadoop.odps.project.name = <MaxCompute_Project_Name>
spark.hadoop.odps.end.point = <Endpoint>
spark.hadoop.odps.runtime.end.point = <VPC_Endpoint>
spark.hadoop.odps.access.id = <AccessKey_ID>
spark.hadoop.odps.access.key = <AccessKey_Secret>
spark.hadoop.odps.cupid.trusted.services.access.list = <Hologres_Classic_Network>
#以下内容保持不变
spark.master = yarn-cluster
spark.driver.cores = 2
spark.driver.memory = 4g
spark.dynamicAllocation.shuffleTracking.enabled = true
spark.dynamicAllocation.shuffleTracking.timeout = 20s
spark.dynamicAllocation.enabled = true
spark.dynamicAllocation.maxExecutors = 10
spark.dynamicAllocation.initialExecutors = 2
spark.executor.cores = 2
spark.executor.memory = 8g
spark.eventLog.enabled = true
spark.eventLog.overwrite = true
spark.eventLog.dir = odps://admin_task_project/cupidhistory/sparkhistory
spark.sql.catalogImplementation = hive
spark.sql.sources.default = hive
```

配置文件参数说明如下所示：

- MaxCompute_Project_Name：待访问MaxCompute项目的名称。

  此处为MaxCompute项目名称，非工作空间名称。您可以登录MaxCompute控制台，左上角切换地域后，即可在**项目管理**页签查看到具体的MaxCompute项目名称。

- AccessKey_ID：具备目标MaxCompute项目访问权限的AccessKey ID。

  您可以进入AccessKey管理页面获取AccessKey ID。

- AccessKey_Secret：AccessKey ID对应的AccessKey Secret。

  您可以进入AccessKey管理页面获取AccessKey Secret。

- Endpoint：MaxCompute项目所属地域的外网Endpoint。

  各地域的外网Endpoint信息，请参见各地域Endpoint对照表（外网连接方式）。

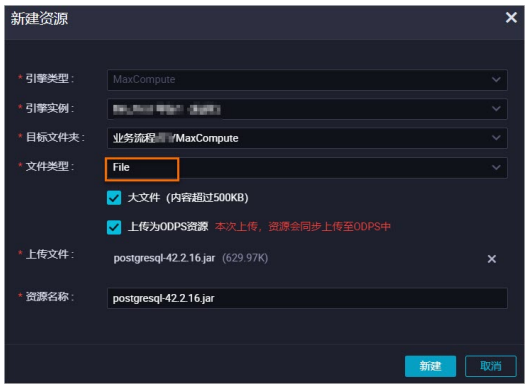
- VPC_Endpoint：MaxCompute项目所属地域的VPC网络的Endpoint。

  各地域的VPC网络Endpoint信息，请参见各地域Endpoint对照表（阿里云VPC网络连接方式）。

- Hologres_Classic_Network：Hologres经典网络类型。配置此项主要是为了在MaxCompute安全运行沙箱环境中，开启到对应Hologres实例的网络策略，否则MaxCompute集群无法访问外部服务。

3. 登录DataWorks控制台。
4. 在左侧导航栏，单击**工作空间列表**。
5. 在**工作空间列表**页面，单击相应工作空间后的**数据开发**。
6. 新建PostgreSQL JDBC资源以及ODPS Spark节点。

i. 在目标业务流程下，右键选择**MaxCompute > 资源 > File**，在弹出的**新建资源**对话框，上传PostgreSQL JDBC的JAR包文件后，单击**新建**。



> 说明
> - DataWorks业务流程创建，详情请参见创建业务流程。
> - DataWorks上的MaxCompute资源创建，详情请参见创建MaxCompute资源。

ii. 在目标业务流程下，右键选择**MaxCompute > 资源 > 新建 > Python**，在**新建资源**对话框，填写**资源名称**后，单击**新建**。

本实践将**资源名称**命名为`read_holo.py`。

iii. 按照如下脚本内容，编写`read_holo.py`，并单击。

```
from pyspark.sql import SparkSession
spark = SparkSession \
            .builder \
            .appName("Spark") \
            .getOrCreate()
jdbcDF = spark.read.format("jdbc"). \
            options(
            url='jdbc:postgresql://hgprecn-cn-2r42******-cn-hangzhou-internal.hologres.aliyuncs.com:80/mc_db_holo',
            dbtable='mc_jdbc_holo',
            user='LTAI5tJE8fuerxXdPPB****',
            password='Okr2kbBKueR3uRaHaBiUHw4r6****',
            driver='org.postgresql.Driver').load()
jdbcDF.printSchema()
```

脚本内容参数说明如下所示：

- url：使用Spark的JDBC连接方式，驱动为`postgresql`。
  - hgprecn-cn-2r42******-cn-hangzhou.hologres.aliyuncs.com:80：Hologres实例的公网访问域名。获取方式，请参见实例配置。
  - mc_db_holo：连接的Hologres数据库名称。本实践命名为mc_db_holo。
- dbtable：Hologres源表名称。本实践命名为mc_jdbc_holo。
- user：阿里云账号或RAM用户的AccessKey ID。您可以进入AccessKey管理页面获取AccessKey ID。
- password：AccessKey ID对应的AccessKey Secret。您可以进入AccessKey管理页面获取AccessKey Secret。
- driver：PostgreSQL驱动，固定值为org.postgresql.Driver。

iv. 在目标业务流程下，右键选择**MaxCompute > 数据开发 > 新建 > ODPS Spark**，在**新建节点**对话框，填写**节点名称**后，单击**提交**。

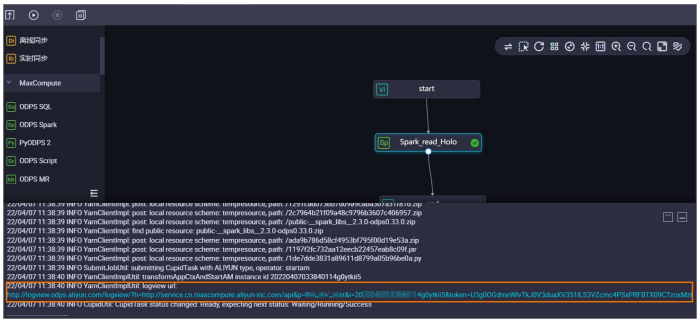
v. 按照下图指引，配置 `spark_read_holo` 。

- 配置项：`spark.hadoop.odps.cupid.trusted.services.access.list` 。
- 配置项取值：`hgprecn-cn-2r42******-cn-hangzhou-internal.hologres.aliyuncs.com:80` 。Hologres经典网络类型。

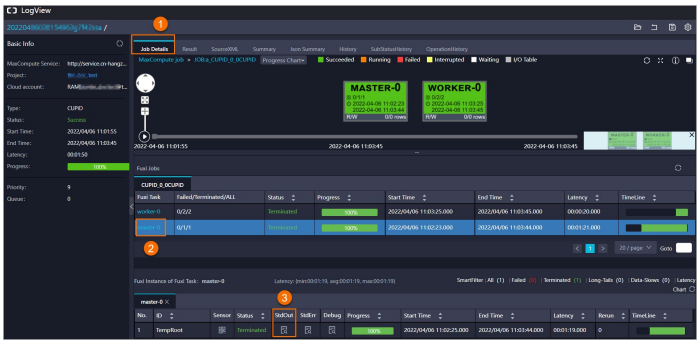
> **说明** 配置此项主要是为了在MaxCompute安全运行沙箱环境中，开启到对应Hologres实例的网络策略，否则MaxCompute集群无法访问外部服务。

7. 在目标业务流程画板，右键选择**spark_read_holo > 运行节点**。

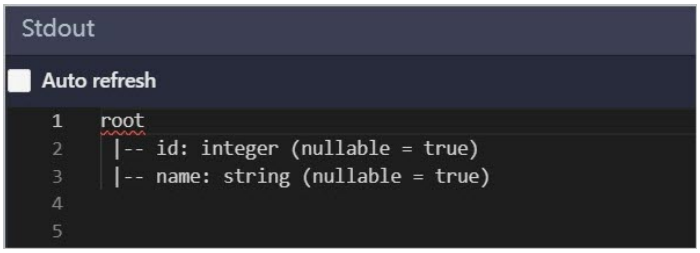
   作业运行后，将会打印作业日志，其中包含MaxCompute作业的诊断信息、Logview链接地址、Spark-UI的Jobview链接地址等。



8. 打开Logview链接，如果作业执行状态为**success**，选择**Job Details > master-0 > StdOut**，查看 `jdbcDF.printSchema()` 的返回结果。



查看**Stdout**，打印的Schema信息与Hologres中创建的 `mc_jdbc_holo` 表一致，即访问成功。



说明 您也可以打开Spark-UI的Jobview链接地址，进行作业的查看与诊断。

4.计算优化

4.1. SQL调优

本文为您介绍常见的SQL问题以及优化示例。

数据倾斜优化

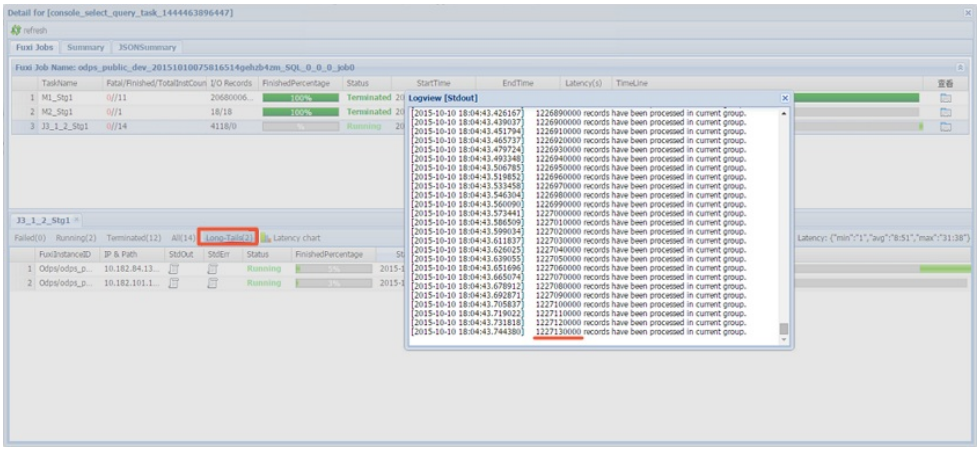
数据倾斜产生的根本原因是少数Worker处理的数据量远远超过其他Worker处理的数据量，因此少数Worker的运行时长远远超过其他Worker的平均运行时长，导致整个任务运行时间超长，造成任务延迟。

- **Join操作导致的数据倾斜**

  Join操作导致数据倾斜的原因是Join on的Key分布不均匀。假设大表A和小表B执行Join操作，运行如下语句。

  ```
  SELECT * FROM A JOIN B ON A.value = B.value;
  ```

  复制该语句的Logview链接并打开，双击执行Join操作的作业，可以看到此时在[Long-tails]区域存在长尾现象，表示数据已经倾斜了。

  您可通过如下方法进行优化：

  - 由于表B是个小表并且没有超过512 MB，您可将上述语句优化为MapJoin语句后执行，语句如下。

    ```
    SELECT /* + MAPJOIN(B) */ * FROM A JOIN B ON A.value = B.value;
    ```

  - 将倾斜的Key用单独的逻辑来处理。假设两边的Key中有大量NULL数据导致了倾斜，则需要在Join前先过滤掉NULL数据或者补上随机数，然后再进行Join，示例如下。

    ```
    SELECT * FROM A JOIN B ON CASE WHEN A.value IS NULL THEN CONCAT('value',RAND() ) ELSE A.value END = B.value;
    ```

    在实际场景中，如果您发现已经数据倾斜，但无法获取导致数据倾斜的Key信息，可以使用如下方法查看数据倾斜。

    ```
    --执行如下语句产生数据倾斜。
    SELECT * FROM a JOIN b ON a.key=b.key;
    --您可以执行如下SQL，查看Key的分布，判断执行Join操作时是否会有数据倾斜。
    SELECT left.key, left.cnt * right.cnt FROM
    (select key, count(*) AS cnt FROM a GROUP BY key) LEFT
    JOIN
    (SELECT key, COUNT(*) AS cnt FROM b GROUP BY key) RIGHT
    ON left.key=right.key;
    ```

- **Group By倾斜**

  造成Group By倾斜的原因是Group By的Key分布不均匀。

  假设表A内有两个字段（Key, Value），表内的数据量较大且Key值分布不均匀，运行语句如下所示。

  ```
  SELECT key, COUNT(value) FROM A GROUP BY key;
  ```

  当表中的数据足够大时，您会在Logview中发现长尾现象。解决此问题，您需要在执行SQL前设置防倾斜的参数，设置语句为`set odps.sql.groupby.skewindata=true`。

- **错误使用动态分区造成的数据倾斜**

  动态分区SQL时，在MaxCompute中会默认增加一个Reduce，用来将相同分区的数据合并在一起。此操作可以：

  - 减少MaxCompute系统产生的小文件，使后续处理更快速。
  - 避免一个Worker输出文件很多时占用内存过大。

  如果引入的Reduce导致分区数据倾斜，则会发生长尾。因为相同的数据最多只会有10个Worker处理，所以数据量大，则会发生长尾，示例如下。

```
INSERT OVERWRITE TABLE A2 PARTITION(dt) SELECT SPLIT_PART(value,'\t',1) AS field1, SPLIT_PART(value,'\t',2) AS field2, dt FROM
A WHERE dt='20151010';
```

这种情况下，不建议使用动态分区，优化语句如下。

```
INSERT OVERWRITE TABLE A2 PARTITION (dt='20151010')
SELECT
SPLIT_PART(value,'\t',1) as field1,
SPLIT_PART(value,'\t',2) as field2
FROM A
WHERE dt='20151010';
```

数据倾斜优化的详情请参见其它计算长尾调优。

窗口函数优化

如果SQL语句中使用了窗口函数，通常每个窗口函数会形成一个Reduce作业。如果窗口函数较多，会消耗过多的资源。您可以对符合下述条件的窗口函数进行优化：

- 窗口函数在OVER关键字后面要完全相同，要有相同的分组和排序条件。
- 多个窗口函数在同一层SQL中执行。

符合上述2个条件的窗口函数会合并为一个Reduce执行。SQL示例如下所示。

```
SELECT
RANK()OVER(PARTITION BY A ORDER BY B desc) AS RANK,
ROW_NUMBER()OVER(PARTITION BY A ORDER BY B desc) AS row_num
FROM MyTable;
```

子查询优化

子查询如下所示。

```
SELECT * FROM table_a a WHERE a.col1 IN (SELECT col1 FROM table_b b WHERE xxx);
```

当此语句中的table_b子查询返回的col1的个数超过1000个时，系统会报错为 `records returned from subquery exceeded limit of 1000` 。此时您可以使用Join语句来代替，如下所示。

```
SELECT a.* FROM table_a a JOIN (SELECT DISTINCT col1 FROM table_b b WHERE xxx) c ON (a.col1 = c.col1)
```

> **说明**
> - 如果没有使用DISTINCT关键字，而子查询表c返回的结果中有相同的col1的值，可能会导致a表的结果数变多。
> - DISTINCT关键字会导致查询在同一个Worker中执行。如果子查询数据量较大，会导致查询比较慢。
> - 如果业务上已经确保子查询中col1列值无重复，您可以删除DISTINCT关键字，以提高性能。

Join语句优化

当两个表进行Join操作时，建议在如下位置使用WHERE子句：

- 主表的分区限制条件可以写在WHERE子句中（最好先用子查询过滤）。
- 主表的WHERE子句建议写在SQL语句最后。
- 从表分区限制条件不要写在WHERE子句中，建议写在ON条件或者子查询中。

示例如下。

```
SELECT * FROM A JOIN (SELECT * FROM B WHERE dt=20150301)B ON B.id=A.id WHERE A.dt=20150301;
SELECT * FROM A JOIN B ON B.id=A.id WHERE B.dt=20150301; --不建议使用。此语句会先执行Join操作后进行分区裁剪，导致数据量变大，性能下降。
SELECT * FROM (SELECT * FROM A WHERE dt=20150301)A JOIN (SELECT * FROM B WHERE dt=20150301)B ON B.id=A.id;
```

4.2. JOIN长尾优化

本文为您介绍执行SQL时JOIN阶段常见的数据倾斜场景以及对应的解决办法。

背景信息

MaxCompute SQL在JOIN阶段会将JOIN Key相同的数据分发到同一个Instance上进行处理。如果某个Key上的数据量比较多，会导致该Instance执行时间比其他Instance执行时间长。执行日志中该JOIN Task的大部分Instance都已执行完成，但少数几个Instance一直处于执行中，这种现象称之为长尾。

数据量倾斜导致长尾的现象比较普遍，严重影响任务的执行时间，尤其是在双十一等大型活动期间，长尾程度比平时更为严重。例如，某些大型店铺的浏览量远远超过一般店铺的浏览量，当用浏览日志数据和卖家维表关联时，会按照卖家ID进行分发，导致某个Instance处理的数据量远远超过其他Instance，而整个任务会因为这个长尾的Instance无法结束。

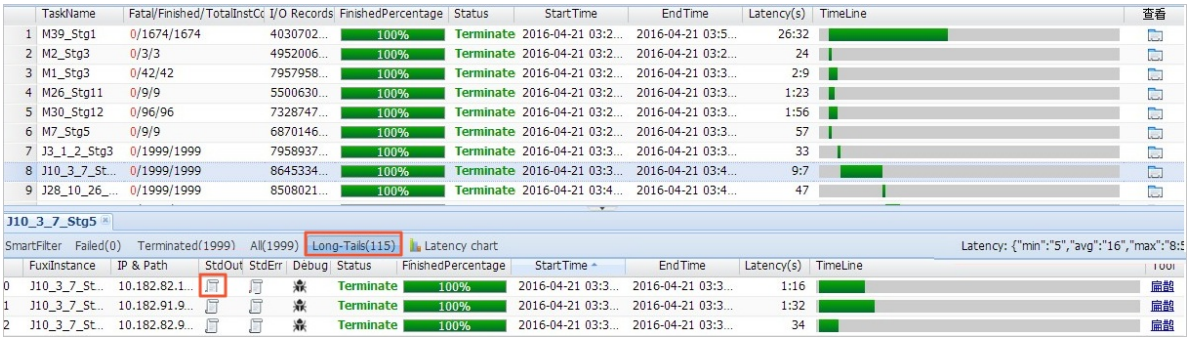
您可以从以下四个方面进行长尾处理考虑：

- 如果两张表里有一张大表和一张小表，可以考虑使用MAP JOIN，对小表进行缓存，具体的语法和说明请参见SELECT语法。
- 如果两张表都比较大，就需要先尽量去重。
- 从业务上考虑，寻找两个大数据量的Key执行笛卡尔积的原因，从业务上进行优化。
- 小表LEFT JOIN大表，直接LEFT JOIN较慢。先将小表和大表进行MAP JOIN，得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（Key倾斜程度与表的膨胀大小成正比）。然后小表再和这个中间表进行LEFT JOIN，这样操作的效果等于小表LEFT JOIN大表。

查看数据倾斜

执行如下步骤查看JOIN是否发生数据倾斜：

1. 打开SQL执行时产生的Logview日志，查看每个Fuxi Task的详细执行信息。**Long-Tails(115)**表示有115个长尾。



2. 单击**FuxiInstance**后的图标，查看**StdOut**中Instance读入的数据量。

   例如，`Read from 0 num:52743413 size:1389941257` 表示JOIN输入读取的数据量是1389941257行。如果Long-Tails中Instance读取的数据量远超过其它Instance读取的数据量，则表示是因为数据量导致长尾。

常见场景及解决方案

- MAP JOIN方案：JOIN倾斜时，如果某路输入比较小，可以采用MAP JOIN避免分发引起的长尾。

  MAP JOIN的原理是将JOIN操作提前到Map端执行，这样可以避免因为分发Key不均匀导致数据倾斜。MAP JOIN使用限制如下：

  - MAP JOIN使用时，JOIN中的从表比较小才可用。所谓从表，即LEFT OUTER JOIN中的右表，或者RIGHT OUTER JOIN中的左表。
  - MAP JOIN使用时，对小表的大小有限制，默认小表读入内存后的大小不能超过512 MB。用户可以通过如下语句加大内存，最大为8192 MB。

    ```
    set odps.sql.mapjoin.memory.max=8192
    ```

    > 说明 odps.sql.mapjoin.memory.max设置过大可能导致OOM（Out Of Memory）内存溢出，请您谨慎操作。

  MAP JOIN的使用方法非常简单，在SQL语句中`SELECT`后加上`/*+ mapjoin(b) */`即可，其中b代表小表（或者是子查询）的别名。举例如下。

  ```
  select   /*+ mapjoin(b) */
             a.c2
           ,b.c3
  from
           (select   c1
                    ,c2
            from     t1         ) a
  left outer join
           (select   c1
                    ,c3
            from     t2         ) b
  on       a.c1 = b.c1;
  ```

- JOIN因为热点值导致长尾

  如果是因为热点值导致长尾，并且JOIN的输入比较大无法用MAP JOIN，可以先将热点Key取出，对于主表数据用热点Key切分成热点数据和非热点数据两部分分别处理，最后合并。以淘宝的浏览量日志表关联商品维表取商品属性为例：

  i. 取出热点Key

     将浏览量大于50000的商品ID取出到临时表。

```
insert   overwrite table topk_item PARTITION (ds = '${bizdate}')
select   item_id
from
        (select   item_id
                 ,count(1) as cnt
         from     dwd_tb_log_pv_di
         where    ds = '${bizdate}'
         and      url_type = 'ipv'
         and      item_id is not null
         group by item_id
         ) a
where    cnt >= 50000;
```

ii. 取出非热点数据。

将主表*（sdwd_tb_log_pv_di）*和热点key表*（topk_item）*外关联后通过条件 `b1.item_id is null` ，取出关联不到的数据即非热点商品的日志数据，此时需要用MAP JOIN。再用非热点数据关联商品维表，因为已经排除了热点数据，不会存在长尾。

```
select   ...
from
        (select   *
         from     dim_tb_itm
         where    ds = '${bizdate}'
         ) a
right outer join
        (select   /*+ mapjoin(b1) */
                  b2.*
         from
                 (select   item_id
                  from     topk_item
                  where    ds = '${bizdate}'
                  ) b1
         right outer join
                 (select   *
                  from     dwd_tb_log_pv_di
                  where    ds = '${bizdate}'
                  and      url_type = 'ipv'
                  ) b2
         on       b1.item_id = coalesce(b2.item_id,concat("tbcdm",rand())
         where    b1.item_id is null
         ) l
on       a.item_id = coalesce(l.item_id,concat("tbcdm",rand());
```

iii. 取出热点数据。

将主表*（sdwd_tb_log_pv_di）*和热点Key表*（topk_item）*内关联，此时需要用MAP JOIN，取到热点商品的日志数据。同时，需要将商品维表*（dim_tb_itm）*和热点Key表*（topk_item）*内关联，取到热点商品的维表数据，然后将第一部分数据外关联第二部分数据，因为第二部分只有热点商品的维表，数据量比较小，可以用MAP JOIN避免长尾。

```
select   /*+ mapjoin(a) */
         ...
from
        (select   /*+ mapjoin(b1) */
                  b2.*
         from
                 (select   item_id
                  from     topk_item
                  where    ds = '${bizdate}'
                  )b1
         join
                 (select   *
                  from     dwd_tb_log_pv_di
                  where    ds = '${bizdate}'
                  and      url_type = 'ipv'
                  and      item_id is not null
                  ) b2
         on       (b1.item_id = b2.item_id)
         ) l
left outer join
        (select   /*+ mapjoin(a1) */
                  a2.*
         from
                 (select   item_id
                  from     topk_item
                  where    ds = '${bizdate}'
                  ) a1
         join
                 (select   *
                  from     dim_tb_itm
                  where    ds = '${bizdate}'
                  ) a2
         on       (a1.item_id = a2.item_id)
         ) a
on       a.item_id = l.item_id;
```

iv. 将步骤2和步骤3的数据通过 `union all` 合并后即得到完整的日志数据，并且关联了商品的信息。

- 通过设置odps.sql.skewjoin参数解决长尾问题。

  此方法简单方便，但是如果倾斜的值发生变化需要修改代码重新执行命令，且变化无法提前预知。另外，如果倾斜值较多也不方便在参数中设置，需要根据实际情况选择拆分代码或者参数设置。参数设置的操作步骤如下：

  i. 开启功能。

  ```
  set odps.sql.skewjoin=true
  ```

  ii. 设置倾斜的Key及对应的值。

  ```
  set odps.sql.skewinfo=skewed_src:(skewed_key) [("skewed_value")]
  ```

  其中：skewed_key代表倾斜的列，skewed_value代表倾斜列上的倾斜值。

- 通过SkewJoin Hint避免热值倾斜。SkewJoin Hint详情请参见SKEWJOIN HINT。
  - **使用方法**

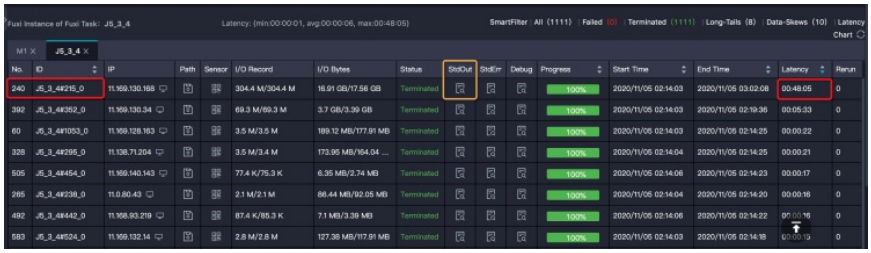
    ```
    --方法1：Hint表名（注意Hint的是表的alias）。
    select /*+ skewjoin(a) */ * from T0 a join T1 b on a.c0 = b.c0 and a.c1 = b.c1;
    --方法2：Hint表名和认为可能产生倾斜的列，例如表a的c0和c1列存在数据倾斜。
    select /*+ skewjoin(a(c0, c1)) */ * from T0 a join T1 b on a.c0 = b.c0 and a.c1 = b.c1 and a.c2 = b.c2;
    --方法3：Hint表名和列，并提供发生倾斜的key值。如果是STRING类型，需要加上引号。例如(a.c0=1 and a.c1="2")和(a.c0=3 and a.c1="4")的值都存在数据倾斜。
    select /*+ skewjoin(a(c0, c1)((1, "2"), (3, "4"))) */ * from T0 a join T1 b on a.c0 = b.c0 and a.c1 = b.c1 and a.c2 = b.c2;
    ```

    > 说明 方法3直接指定值的处理效率比方法1和方法2（不指定值）高。

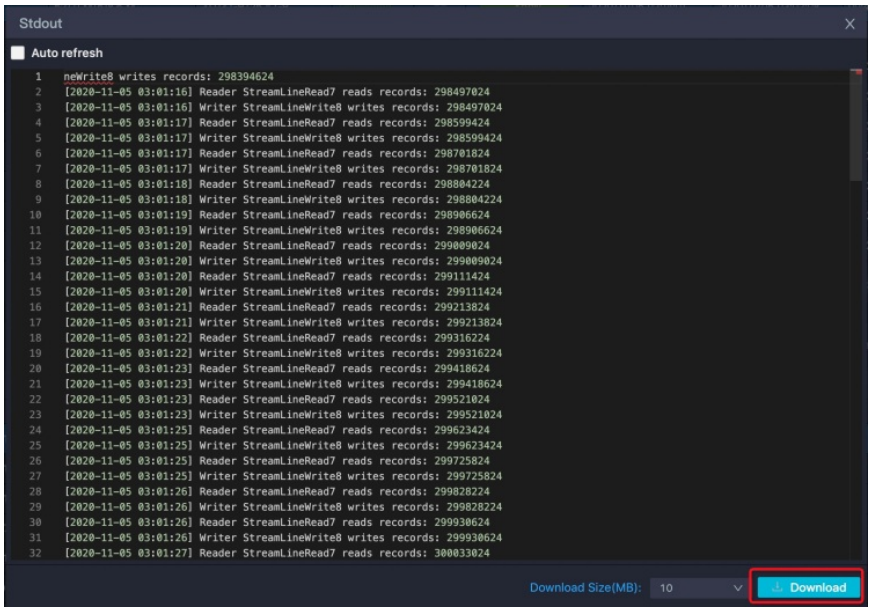
  - **定位倾斜Join**

以如下Logview为例，耗时最长的Fuxi Task为J5_3_4，定位倾斜Join的步骤如下。

a. 单击J5_3_4，在Fuxi Instance页面观察J5的Instances，可以看到J5_3_4#215_0的耗时最长，且I/O record和I/O Bytes远远大于其它Instance。

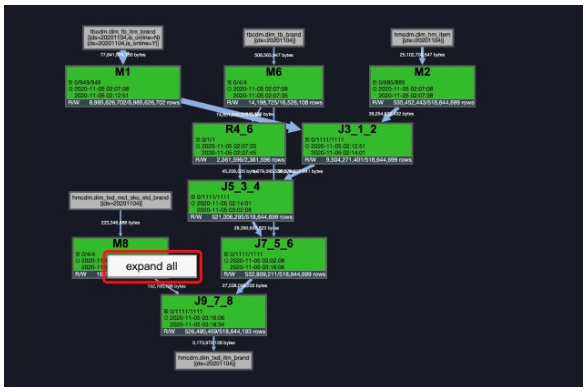


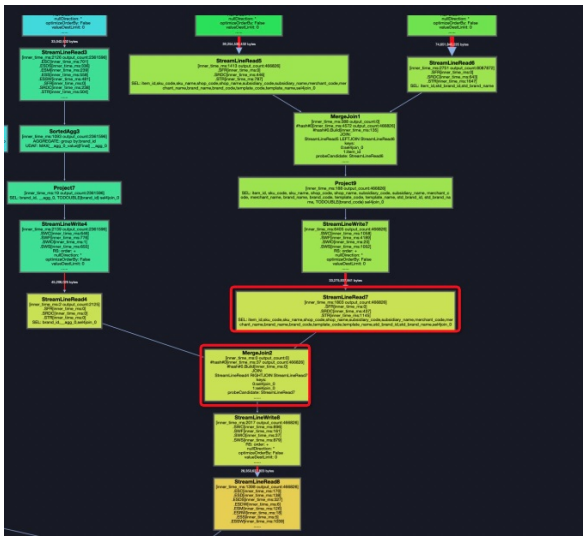
b. 此时，可以认为J5节点发生了数据倾斜，继续需要确定倾斜的是哪个Join。单击倾斜Instance的**StdOut**和任一个不倾斜节点的**StdOut**，查看内容。通常，网页无法显示所有**StdOut**的内容，您可以单击**Download**，查看全部内容。



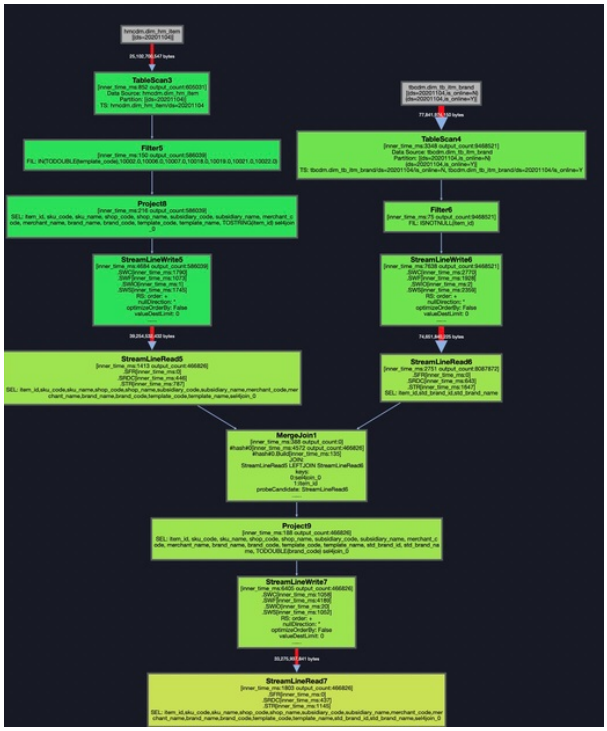
c. Download的内容如下图所示，可以看到倾斜Instance的StreamLineRead7的record count远远大于非倾斜Instance，因此可以认为StreamLineWrite7到SreamLineRead7的这个Shuffle发生了倾斜。

d. 在DAG热点图界面，单击鼠标右键选择**expland all**，即可找到StreamLineWrite7和StreamLineRead7。





e. 可以看到MergeJoin2发生了倾斜，倾斜的是MergeJoin2的StreamLineRead7这一路。此时，进一步追溯StreamLineRead7的输入节点，可以看到是dim_hm_item和dim_tb_itm_brand Join之后的结果，再与StreamLineRead4（输入是dim_tb_brand表）进行Join，即MergeJoin2。



f. 此时根据这些表名，在SQL语句中定位，可以发现是Left Outer Join发生了倾斜，而倾斜的是t1表，在SQL中添加 `/*+ skewjoin(t1) */` 即可。

```
        t1.sku_code,
        t1.sku_name,
        t1.brand_code,
        t1.brand_name,
        t2.std_brand_id,
        t2.std_brand_name
    from hmcdm.dim_hm_item t1
    left outer join tbcdm.dim_tb_itm_brand t2
    on cast(t1.item_id as string)=cast(t2.item_id as string)
    and t2.ds=max_pt('tbcdm.dim_tb_itm_brand')
    where t1.ds='20201104'
    and t1.template_code in (
        10002,
        10006,
        10007,
        10018,
        10019,
        10021,
        10022
    )
) t1
left outer join(
    select brand_id as std_brand_id, max(brand_name) as std_brand_name
    from tbcdm.dim_tb_brand
    where ds=max_pt('tbcdm.dim_tb_brand')
    and is_standard_brand='Y'
    and cast(brand_id as bigint)>0
    group by brand_id
) t21
on t1.brand_code=t21.std_brand_id
left outer join(
    select
```

4.3. 其它计算长尾调优

除了Join之外还有其它计算长尾现象产生，本文将为您介绍典型的长尾问题的场景及其解决方案。

长尾问题是分布式计算中最常见的问题之一。造成长尾问题的主要原因是数据分布不均，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。

为了避免一个Worker单独运行大量的工作，需要把工作分给多个Worker去执行。

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：

- 对SQL进行改写，添加随机数，把长Key进行拆分。举例如下。

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

不考虑Combiner，Mapper会Shuffle到Reducer上，然后Reducer再做Count操作。对应的执行计划是 Mapper > Reducer。但是如果对长尾的Key再进行一次工作再分配，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Mapper>Reducer>Reducer。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** 如果数据的长尾并不严重，用此方法人为地增加一次Reducer的过程，最终的消耗时间可能反而更长。

- 通过设置系统参数优化长尾问题。

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

此设置为通用性的优化策略，无法针对具体的业务进行分析，得出的结果不一定是最优的。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，将长Key进行拆分的策略已经不生效了。此场景下，您可以考虑通过其它方式解决。

解决办法

```
--原始SQL，不考虑uid为空的情况。
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;
```

该方法是把Distinct改成了普通的Count，这样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，系统又能执行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，会在最后启用一个Reduce，对数据进行整理，所以如果使用动态分区写入数据时有倾斜，就会发生长尾。
- 一般情况下，滥用动态分区功能也是产生这类长尾的一个常见原因。

解决办法

如果已经确定需要把数据写入某个具体分区，则可以在插入数据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。通过Combiner，减少Mapper Shuffle到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。例如，在运行任务的时候，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_S
tg2_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_S
tg2_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_S
tg2_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_S
tg2_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，运行一样的数据，然后取运行结束较早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。

- 实际数据可能包含非常多的噪音。例如，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在计算时长尾。类似的情况还有根据xxid进行关联的时候，需要考虑关联字段是否存在为空的情况。
- 一些特殊的业务情况。例如，ISV的操作记录，在数据量、行为方式上会和普通个人有很大的区别。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，尽量不要使用常量字段做Distribute by字段来实现全排序。

4.4. 长周期指标的计算优化方案

本文为您介绍如何对长周期指标的计算进行优化。

实验背景

电子商务公司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。这些指标需要根据一段时间内的累积数据进行计算。

通常，这些指标的计算方式为从日志明细表中查询数据进行计算。例如，运行如下SQL语句计算商品最近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。下文不再重复说明。

当每天的日志量很大时，SELECT操作需要大量的Map Instance，运行上面的代码需要的Map Instance个数太多，甚至会超过99999个Instance的限制个数，导致Map Task无法顺利执行。

实验目的

在不影响性能的情况下计算长周期的指标。

影响性能的问题根源是多天汇总数据量过大，建议您使用构建临时表的方式对每天的数据进行轻度汇总，这样可以去掉很多重复数据，减少数据量。

实验方案

1. 构建中间表，每天汇总一次。

   对于上述示例，构建 `item_id+visitior_id` 粒度的日汇总表，记作A。

   ```
   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。

   ```
   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

上述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数据的时候，都需要读取N个分区的数据。

您可以通过增量累计方式计算长周期指标的方式，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。

场景示例

计算最近1天店铺商品的老买家数。老买家是指过去一段时间有购买的买家（例如过去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A，记录买家和店铺的购买关系、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。
- 计算老买家数时，判断最近一次购买时间是否是30天之内，从而做到最大程度上的数据关系对去重，减少计算输入数据量。

5.作业诊断

5.1. 通过Logview诊断慢作业

在实际业务开发过程中，企业通常要求作业能在期望的时间节点前产出结果，并根据结果做进一步决策，这就需要作业开发人员及时关注作业运行状态，识别并优化慢作业。您可以通过MaxCompute的Logview功能诊断慢作业。本文为您介绍导致出现慢作业的原因及如何查看慢作业并提供对应的解决措施。

背景信息

MaxCompute提供的Logview功能会记录作业的全部运行阶段日志信息，为查看和调式作业提供指导依据。您可以通过作业运行结果中的**Log View**处获取Logview链接。MaxCompute提供了2个版本的Logview功能，推荐您使用Logview 2.0，页面加载速度、设计风格更优。更多Logview 2.0信息，请参见Logview 2.0。

出现慢作业的常见情形如下：

- 计算资源不足

  当MaxCompute项目的计费模式为包年包月时，如果某一时间段内提交的作业数量较多或小文件过多，会导致购买的计算资源（CU）全部占满，作业变为排队状态。

- 数据倾斜

  当处理的数据量很大或某些作业专门用于处理特定数据时，会导致出现大部分作业已经运行结束但某些作业迟迟不结束的长尾（Long-Tails）情况。

- 代码逻辑问题

  当SQL或UDF逻辑低效，或没有使用最优的参数设定时，会导致出现Fuxi Task的运行时间很长，但Fuxi Task下每个Fuxi Instance的运行时间都很均匀的情况。作业、Fuxi Task、Fuxi Instance间的关系请参见作业详情。

计算资源不足

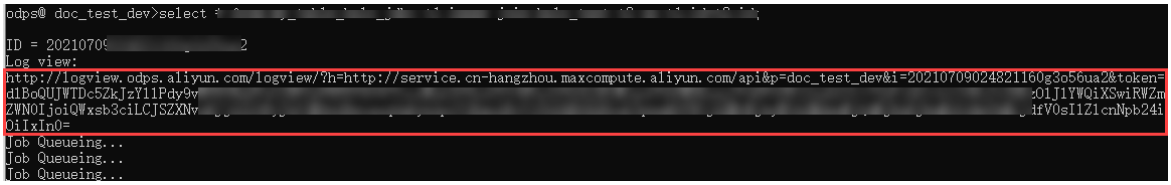
问题现象

提交作业后，有如下两种表现：

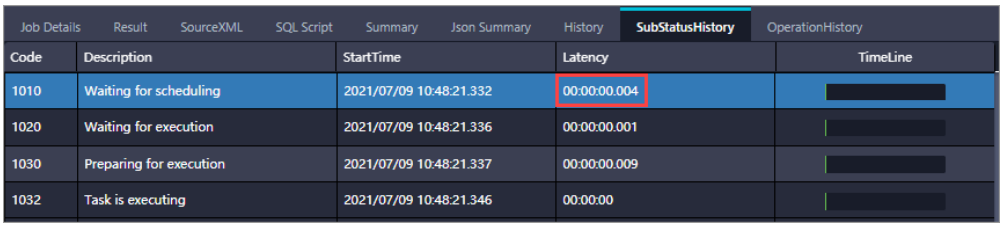
- 现象一：作业一直显示 `Job Queueing...` 状态。

  可能因其他作业占用了资源组的资源，出现作业排队情况。您可以通过如下步骤查看作业等待时长：

  i. 在作业运行结果信息中获取Logview链接并在浏览器中打开。

  ii. 在**SubStatusHistory**页签的**Description**列查看Waiting for scheduling对应的**Latency**值，即为等待时长。

Code	Description	StartTime	Latency	TimeLine
1010	Waiting for scheduling	2021/07/09 10:48:21.332	00:00:00.004	
1020	Waiting for execution	2021/07/09 10:48:21.336	00:00:00.001	
1030	Preparing for execution	2021/07/09 10:48:21.337	00:00:00.009	
1032	Task is executing	2021/07/09 10:48:21.346	00:00:00	

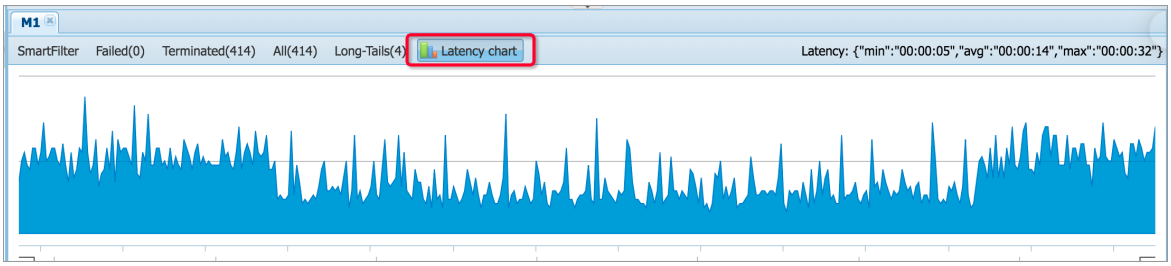
- 现象二：作业有进展但执行速度慢。

  提交作业后，由于所需计算资源较多，当前的资源组不能同时启动所有的Fuxi Instance，作业虽然有进度但执行不快。您可以通过如下步骤查看作业执行状态：

  i. 在作业运行结果信息中获取Logview链接并在浏览器中打开。

  ii. 在**Job Details**页签的Fuxi Instance区域，单击**Latency Chart**，查看作业运行状态图。

下图展示的是一个资源充足的作业的运行状态，蓝色部分的低端是平齐的，表明所有的Fuxi Instance几乎在同一时间启动。



下图展示的是一个资源不充足的作业的运行状态，图形整体呈现阶梯向上的形态，表明作业的Fuxi Instance是逐步调度起来的。

产生原因

针对上述现象，您可以按照如下操作确认产生原因：

1. 进入MaxCompute管家。
2. 在左侧导航栏，单击**配额**。

3. 在**包年包月配额组**区域，单击MaxCompute项目对应的配额组。
4. 在**资源消耗**页签的**预留CU资源使用趋势**图中，单击计算资源使用量最高的点，记录时间点。

5. 在左侧导航栏，单击**作业**后，在右侧单击**作业管理**。
6. 在作业管理页面，根据记录的时间点填写**日期范围**，在**作业状态**下拉列表选择**Running**，单击**确定**。
7. 在作业列表区域，单击**CPU使用占比（%）**后的图标，将作业按照CPU使用量占比降序排列。
   - 如果某个作业占用CPU特别大，单击作业**操作**列的Logview，在Fuxi Instance区域查看**I/O Bytes**。当**I/O Bytes**只有1 MB或几十KB，且作业的并行度很高（多个Fuxi Instance）时，表明读取表的小文件过多，需要合并小文件或调整并行度。

- 如果CPU占比均匀说明同时提交了多个大作业，占满了计算资源。您需要增购计算资源或将作业使用按量计费资源运行。

解决措施

- 合并小文件：

  合并小文件，请参见小文件优化及作业诊断常见问题。

- 调整并行度：

  MaxCompute的并行度会根据输入的数据量和作业复杂度自动推测执行，一般不需要调节，理想情况并行度越大速度处理越快。但是对于包年包月资源组，资源组可能会占满，导致作业都在等待资源，作业运行变慢。您可以通过odps.stage.mapper.split.size、odps.stage.reducer.num、odps.stage.joiner.num或odps.stage.num参数调整并行度。更多参数信息，请参见SET操作。

- 购买计算资源：

  增购计算资源，请参见升级。

- 使用按量计费资源：

  购买按量计费规格，并通过MaxCompute管家设置包年包月项目使用按量计费资源。更多操作信息，请参见包年包月项目使用按量计费资源。

数据倾斜

问题现象

Fuxi Task中大多数Fuxi Instance都已经结束了，但是部分Fuxi Instance迟迟不结束，出现长尾（Long-Tails）情况。

您可以在Logview的**Job Details**页签的Fuxi Instance区域，单击**Long-Tails**，查看出现长尾情况的Fuxi Instance。

产生原因

Fuxi Instance处理的数据多或Fuxi Instance负责处理特殊数据。

解决措施

数据倾斜的详细解决措施，请参见数据倾斜优化。

代码逻辑问题

问题现象

提交作业后，有如下两种表现：

- 现象一：数据膨胀。Fuxi Task的输出数据量比输入数据量大很多。

  输入、输出数据量可通过Fuxi Task区域的**I/O Record**和**I/O Bytes**参数获取。如下图所示，1 GB的数据经过处理变为了1 TB，一个Fuxi Instance处理1 TB的数据，会降低运行效率。

SQL_0_0_0_job_0 | SQL_0_0_0_merge

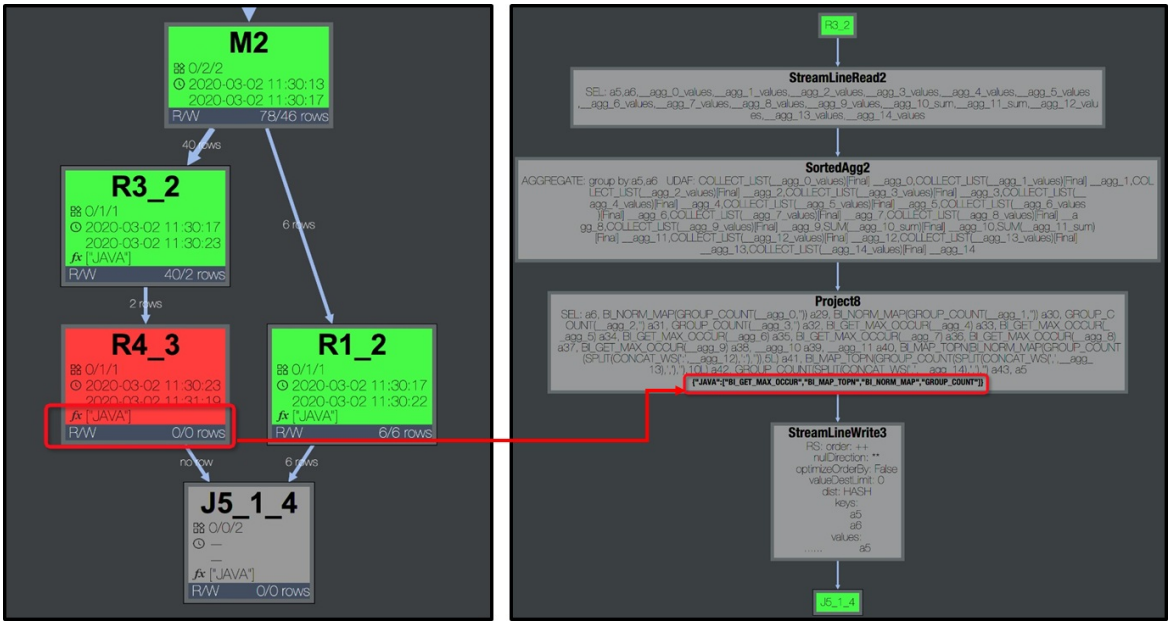
Fuxi Task	Failed/Terminated/ALL	I/O Record	I/O Bytes	Status	Sensor
M1	0/10,054/10,054	793.6 M/1.3 T	1.95 TB/14.88 TB	Terminated	
R2_1	0/4,000/4,000(+2 backups)	1.3 T/8.8 G	14.88 TB/189.14 GB	Terminated	
R3_2	0/4,000/4,000	8.8 G/5.6 M	189.14 GB/113.22 GB	Terminated	

- 现象二：UDF执行效率低。

  某个Fuxi Task执行效率低，且该Fuxi Task中有UDF。当UDF执行超时报错 `Fuxi job failed - WorkerRestart errCode:252,errMsg:kInstanceMonitorTimeout, usually caused by bad udf performance` 。您可以通过如下步骤查看UDF的位置及执行速度：

  i. 在作业运行结果信息中获取Logview链接并在浏览器中打开。

  ii. 在作业执行图区域，双击运行速度慢或运行失败的Fuxi Task，在Operator算子图中查看UDF的位置。如下图所示。



iii. 在Fuxi Instance区域，单击StdOut，查看UDF的执行速度。

通常，`Speed(records/s)`在百万或者十万级别。

```
[2021-07-12 15:35:16] Table reader TableScan1 has read 3 records
[2021-07-12 15:35:16] Table writer AdhocSink1 has produced 7 records
[2021-07-12 15:35:16] Table writer AdhocSink1 has produced 7 records
[2021-07-12 15:35:16] Table writer cursor AdhocSink1 process data elapsed time(ms): 50.636;
CursorId                    OutputCount       InnerTime(ms)    Speed(records/s)          Others
AdhocSink1                            7                 137                  50 {"ByteEncodingCount":3,"ByteEncodingInBytes":21,"CompressionInBytes":536,
"CompressionLatencyUs":13180,"CompressionOutBytes":519,"EncodingCount":5,"EncodingInBytes":96,"EncodingLatencyUs":30,"EncodingOutBytes":20,"IOBlockingLatencyUs":6,"IOCount":7,
"WriterCount":1,"WriterInclLatencyUs":97}
GlobalInit                            0                 458                   0 {"LoadAndParseIR":{"T":3500},"OperatorCreation":{"T":455302}}
TableFunctionScan1                    7                   0               14285
TableScan1                            3                  55                  54 {"DecodingCount":2,"DecodingInBytes":4,"DecodingLatencyUs":1,"DecodingOutBytes":12,
"DecomplatencyUs":2,"IOBlockingLatencyUs":1255,"IOCount":3,"ReadRowCount":3,"ReaderCount":1,"ReaderInclLatencyUs":3570,"RequestedRowCount":3,"VectorBatchConversionLatencyUs":25}
[2021-07-12 15:35:16]   ----------  End Of Task: M1#0   ----------
```

产生原因

- 现象一：实际业务处理逻辑导致出现数据膨胀，需要确认业务逻辑是否存在问题。
- 现象二：UDF代码逻辑不合理，需要调整代码逻辑。

解决措施

- 现象一的解决措施：确认业务逻辑是否存在问题，如果存在问题请修改；如果不存在问题，请通过odps.stage.mapper.split.size、odps.stage.reducer.num、odps.stage.joiner.num或odps.stage.num参数调整并行度。更多参数信息，请参见SET操作。
- 现象二的解决措施：检查并修改UDF代码逻辑，优先使用内建函数实现，如果内建函数无法满足业务需求，再考虑使用UDF。更多内建函数信息，请参见内建函数。

6.成本优化

6.1. 成本优化概述

本文介绍了成本优化的流程。

MaxCompute的成本优化是一个持续不断的过程。由于大数据的动态性和不断变化的性质，企业用户成本优化的活动应该持续不断的进行。您可以参考以下流程进行优化：

1. 在使用MaxCompute之前，建议您详细了解付费策略以及预估自己需要使用的资源，选择适合您的付费方式。详情请参见选择付费方式。
2. 在使用过程中，可以从计算、存储、上传和下载这几个方面进行优化，以减少成本。详情请参见计算成本优化、存储成本优化、数据上传下载成本优化。
3. 及时查看账单，对账单中的异常点进行分析和优化。详情请参见成本追踪。

6.2. 选择付费方式

本文为您介绍如何根据实际情况选择付费方式以降低使用成本。

MaxCompute的计费策略

MaxCompute提供了两种计费方式：

- 包年包月：计算资源是包月或者包年的，存储和下载资源是按实际使用量计费。
- 按量计费：存储、计算和下载资源都是按实际使用量计费。

详细的计费策略请参见计费项与计费方式概述。MaxCompute提供了TCO工具和成本预估实践两种选型工具帮助您分析如何选择付费方式。

TCO工具

您可以使用如下TCO工具进行费用预估：

- MaxCompute报价速算器：适用包年包月方式。您可通过输入上传和下载的数据大小以及需要的计算资源自动地计算月成本。
- CostSQL方法：适用按量计费方式。
  - 您在实际生产环境中，即正式上线一个分析SQL前，可以通过Cost SQL命令估算该SQL作业的费用。详情请参见计量预估。
  - 如果您使用的开发工具为Intellij IDEA，可以在提交SQL脚本时自动估算费用。详情请参见开发及提交SQL脚本。
  - 如果您使用的开发工具为DataWorks，也可以进行费用预估。详情请参见计算费用（按量计费）。

> 说明
> - 部分SQL运算不支持费用预估。例如，外部表参与计算的SQL。
> - 此功能仅为费用预估，实际费用以最终出账为准。

成本预估实践

成本预估实践为您提供了一些成本预估的案例和技巧，您可以根据实际情况，参考案例选择最经济的付费方式。

- **处理1 TB数据的付费方式。**

  经过相关测试，对于费用我们给出如下预估信息供您参考。

付费方式	用户类型	相应速度	每月预估费用
包年包月	密集型计算	分钟级别	24000元
	密集型存储	小时级别	7500元
按量计费	9000元（复杂度为1，每天运行1次的月价格）		

  包年包月方式下根据您对CPU资源的要求，有如下两种建议：

  - 密集型计算：适用于CPU资源要求比较高的场景。使用160 CU资源运行1 TB数据，响应速度为分钟级别，资源费用为每月24000元左右。
  - 密集型存储：如果对于计算的响应时间要求不高，推荐您使用包年包月密集型存储，使用的计算资源为50 CU左右，响应速度为小时级别，费用为每月7500元左右。

  如果您选择按量计费，按照基础的复杂度1来计算，对于1 TB的数据的单次计算资源费用大约为300元/天，一个月为9000元。按量计费是按照次数计费的，如果多次进行1 TB数据的计算，其费用也会成倍增加。

  对于刚开始上云的企业，建议先开通按量计费，然后将数据进行POC测试（即针对客户具体应用的验证性测试），计算自己的任务大概需要消耗多少Worker，通过Worker数推算CU数量，这样就能大概估算出最终需要购买资源的数量。

- **Hadoop用户上云迁移的付费方式。**

  某个Hadoop集群可能有1个管控节点以及5台计算节点，每台机器32核，相当于是32个CPU，5台计算节点就是160个CPU，对应标准的官方报价是每个月24000元（此价格未包含折扣或者优惠）。

MaxCompute无需考虑管控节点，比Hive性能快80%，且免运维，为您节省成本。

- **混合使用的付费方式。**
  - 包年包月模式进行生产业务（小时级ETL）+按量计费模式进行非周期任务或即席查询。

    对周期性高密度计算作业使用包年包月模式，对非周期性的大规模数据处理作业使用按量计费模式。按量计费模式下可以不存储数据，通过读取其它账号下的表获取数据，从而可以节省数据存储费用。不同账号下跨表计算需要通过授权来实现，详细请参见创建角色（项目级别）。
  - 包年包月模式进行非周期任务或即席查询+按量计费模式进行生产业务（天级别ETL）。

    企业为了解决因为日常数据测试引起的费用不可控的问题，可以把数据测试和非周期任务放在固定资源组，通过MaxCompute管家为开发组和BI组配置不同的二级资源。生产作业如果只是每天处理一次，可以放在按量计费资源组。

调整付费方式

如果您选择了包年包月模式的服务，但因为业务变化导致数据量急剧变化，资源不够使用或者空余，您可以进行升配或者降配，详情请参见升级和降配。

此外您还可以灵活地选择和转变付费方式。例如，从包年包月转换成按量计费，或从按量计费转换成包年包月，详细请参见转换计费方式。

> **说明** 请合理评估计算作业性能与时间的关系，避免转换成包年包月模式后，由于购买的CU数量少，导致延长作业计算周期，达不到预期后又转回按量计费模式。

6.3. 计算成本优化

本文为您介绍如何通过对SQL作业和MapReduce作业的优化减少计算成本。

您可以在计算前对计算成本进行预估，控制计算成本。详细的预估方法，请参见TCO工具。也可以配置消费预警，预防意料之外的高额费用。预警方法设置请参见消费监控告警消费控制。如果计算成本过高，您可以参考下面的方法进行优化，以控制计算成本。

SQL作业计算成本控制

对于SQL计算作业，大部分费用较高的SQL都是由全表扫描引起的。另外，调度频繁也会引起SQL作业费用的增加，调度频繁可能会产生任务的堆积，在后付费的情况下会造成排队现象，如果任务多又出现了排队，那么第二天的账单就会异常。通过如下策略进行SQL作业计算成本控制：

- 避免频繁调度。MaxCompute是批量计算的服务，距离实时的计算服务还是存在一定距离的。如果间隔时间变短，计算频率增加，再加上使用SQL的不良习惯就会导致计算费用飙升，产生费用较高的账单。所以请尽量避免频繁调度，如果要进行频繁调度请通过CostSQL等方式预估一下SQL的开销到底有多大，不然会造成较大预估外的开销。
- 控制全表扫描。您可以通过以下几种策略来控制全表扫描问题：
  - 设置参数关闭全表扫描功能。目前支持Session级别和Project级别的控制。

    ```
    --禁止session 级别全表扫描。
    set odps.sql.allow.fullscan=false;
    --禁止project级别全表扫描。
    SetProject odps.sql.allow.fullscan=false;
    ```
  - 使用列剪裁。在读数据的时候，只读取查询中需要用到的列，而忽略其他列，避免使用 `SELECT *` 引起全表扫描。

    ```
    SELECT a,b FROM T WHERE e < 10;
    ```

    其中，T包含5个列 `(a,b,c,d,e)` ，列c，d将会被忽略，只会读取a，b，e列。
  - 使用分区剪裁。分区剪裁是指对分区列指定过滤条件，使得只读取表的部分分区数据，避免全表扫描引起的错误及资源浪费。

    ```
    SELECT a,b FROM T WHERE partitiondate='2017-10-01';
    ```
  - SQL关键字的优化。计费的SQL关键字包括：JOIN、GROUP BY、ORDER BY、DISTINCT、INSERT INTO。您可以根据以下建议进行优化：
    - 在进行JOIN的时候，一定要先进行分区剪裁再进行JOIN，不然的话就可能会先做全表扫描。分区裁剪失效请参考分区剪裁失效的场景分析。

- 减少FULL OUTER JOIN 的使用，改为UNION ALL。

```
SELECT COALESCE(t1.id, t2.id) AS id, SUM(t1.col1) AS col1
 , SUM(t2.col2) AS col2
FROM (
 SELECT id, col1
 FROM table1
) t1
FULL OUTER JOIN (
 SELECT id, col2
 FROM table2
) t2
ON t1.id = t2.id
GROUP BY COALESCE(t1.id, t2.id);
--可以优化为如下语句。
SELECT t.id, SUM(t.col1) AS col1, SUM(t.col2) AS col2
FROM (
 SELECT id, col1, 0 AS col2
 FROM table1
 UNION ALL
 SELECT id, 0 AS col1, col2
 FROM table2
) t
GROUP BY t.id;
```

- 在UNION ALL内部尽可能不使用GROUP BY，改为在外层统一GROUP BY。

```
SELECT t.id, SUM(t.val) AS val
FROM (
 SELECT id, SUM(col3) AS val
 FROM table3
 GROUP BY id
 UNION ALL
 SELECT id, SUM(col4) AS val
 FROM table4
 GROUP BY id
) t
GROUP BY t.id;
可以优化为----------------------------
SELECT t.id, SUM(t.val) AS val
FROM (
 SELECT id, col3 AS val
 FROM table3
 UNION ALL
 SELECT id, col4 AS val
 FROM table4
) t
GROUP BY t.id;
```

- 临时导出的数据如果需要排序，尽量在导出后使用Excel等工具进行排序，避免使用ORDER BY。
- 尽量避免使用DISTINCT关键字，改为多套一层GROUP BY。

```
SELECT COUNT(DISTINCT id) AS cnt
FROM table1;
可以优化为----------------------------
SELECT COUNT(1) AS cnt
FROM (
 SELECT id
 FROM table1
 GROUP BY id
) t;
```

- 尽量避免使用INSERT INTO方式写入数据，可以考虑增加一个分区字段。通过降低SQL复杂度，来节省SQL的费用。

- 避免使用运行查询的方式预览表数据。如果您想预览表数据，可以使用表预览的方式查看数据，而不会产生费用。如果您使用DataWorks，在**数据地图**页面，可以预览表以及查看表的详情，具体方法请参见查看表详情。如果您使用MaxCompute Studio，双击表就可以进行表数据预览。
- 计算时合理的选择工具。由于MaxCompute的查询响应是分钟级，不适合直接用于前端查询，计算出的结果数据同步到外部存储中保存，对于大部分用户来说，关系型数据库是最优先的选择。轻度计算推荐使用MaxCompute，重度计算（即直接出最终结果。前端展示时，不做任何判断、聚合、关联字典表、甚至不带WHERE条件）推荐使用RDS等关系型数据库。

MapReduce作业计算成本控制

通过如下策略进行MapReduce作业计算成本控制：

- 设置合理的参数。

- split size

  Map默认的split size是256 MB，split size的大小决定了Map个数多少，如果用户的代码逻辑比较耗时，Map需要较长时间结束，可以通过 `JobConf#setSplitSize` 方法适当调小 `split size` 。然而 `split size` 也不宜设置太小，否则会占用过多的计算资源。

- MapReduce Reduce Instance

  单个job默认Reduce Instance个数为Map Instance个数的1/4，用户设置作为最终的Reduce Instance个数，范围[0, 2000]，数量越多，计算时消耗越多，成本越高，应合理设置。

- MapReduce减少中间环节

  如果有多个MapReduce作业之间有关联关系，前一个作业的输出是后一个作业的输入，可以考虑采用Pipeline的模式，将多个串行的MapReduce作业合并为一个，这样可以用更少的作业数量完成同样的任务。一方面减少中间表造成的多余磁盘IO，提升性能；另一方面减少作业数量使调度更加简单，增强流程的可维护性，具体使用方法请参见Pipeline示例。

- 对输入表列裁剪

  对于列数特别多的输入表，Map阶段处理只需要其中的某几列，可以通过在添加输入表时明确指定输入的列，减少输入量。例如只需要c1，c2列，可以参考如下设置。

```
InputUtils.addTable(TableInfo.builder().tableName("wc_in").cols(new String[]{"c1","c2"}).build(), job);
```

  设置后，在Map中读取到的Record就只有c1，c2列，如果之前是使用列名获取Record数据，不会有影响，而用下标获取的需要注意这个变化。

- 避免资源重复读取

  资源的读取尽量放置到Setup阶段读取，避免资源多次读取的性能损失，另外系统也有64次读取的限制，资源的读取请参见使用资源示例。

- 减少对象构造开销

  对于Map、Reduce阶段每次都会用到的Java对象，避免在Map/Reduce函数里构造，可以放到Setup阶段，避免多次构造产生的开销。

```
{
    ...
    Record word;
    Record one;
    public void setup(TaskContext context) throws IOException {
      // 创建一次就可以，避免在map中每次重复创建。
      word = context.createMapOutputKeyRecord();
      one = context.createMapOutputValueRecord();
      one.set(new Object[]{1L});
    }
    ...
}
```

- 合理使用Combiner

  如果Map的输出结果中有很多重复的Key，可以合并后输出，Combiner后可以减少网络带宽传输和一定Shuffle的开销。如果Map输出本来就没有多少重复的，就不要用Combiner，用了反而可能会有一些额外的开销。Combiner实现的是和Reducer相同的接口，例如一个WordCount程序的Combiner可以定义如下。

```
/**
   * A combiner class that combines map output by sum them.
   */
  public static class SumCombiner extends ReducerBase {
    private Record count;
    @Override
    public void setup(TaskContext context) throws IOException {
      count = context.createMapOutputValueRecord();
    }
    @Override
    public void reduce(Record key, Iterator<Record> values, TaskContext context)
        throws IOException {
      long c = 0;
      while (values.hasNext()) {
        Record val = values.next();
        c += (Long) val.get(0);
      }
      count.set(0, c);
      context.write(key, count);
    }
  }
```

- 合理选择Partition Column或自定义Partitioner

  合理选择Partition Columns，可以使用 `JobConf#setPartitionColumns` 这个方法进行设置（默认是Key Schema定义的Column），设置后数据将按照指定的列计算HASH值分发到Reduce中， 避免数据倾斜导致作业长尾现象，如有必要也可以选择自定义Partitioner，自定义Partitioner的使用方法如下。

```
import com.aliyun.odps.mapred.Partitioner;
public static class MyPartitioner extends Partitioner {
@Override
public int getPartition(Record key, Record value, int numPartitions) {
  // numPartitions即对应reducer的个数
  // 通过该函数决定map输出的key value去往哪个reducer。
  String k = key.get(0).toString();
  return k.length() % numPartitions;
}
}
```

在*jobconf*里进行设置如下。

```
jobconf.setPartitionerClass(MyPartitioner.class)
```

需要在*jobconf*里明确指定Reducer的个数。

```
jobconf.setNumReduceTasks(num)
```

- 合理使用JVM内存参数

  过于追求调优，把MapReduce任务内存设置过大也会造成成本上升。标准配置是 `1 Core 4G ，odps.stage.reducer.jvm.mem=4006` ，当CPU与内存比超过 `1:4` 时，对应的费用也会大幅升高。

6.4. 存储成本优化

本文从数据分区、表生命周期和定期删除表3个方面为您介绍如何优化存储成本。

对于存储优化而言，有三个关键点：

- 合理地进行数据分区。
- 设置合理的表生命周期。
- 定期地删除废表。

合理设置数据分区

MaxCompute将分区列的每个值作为一个分区。您可以指定多级分区，即将表的多个字段作为表的分区，分区之间的关系类似多级目录的关系。在使用数据时如果指定了需要访问的分区名称，则只会读取相应的分区，避免全表扫描，提高处理效率，降低费用。

- 假如最小统计周期为天，建议采用日期作为分区字段。每天将数据迁移到指定分区，再读取指定分区的数据进行下游统计。
- 假如最小统计周期为小时，建议采用日期+小时作为分区字段。每小时将数据迁移到指定分区，再读取指定分区的数据进行下游统计。如果小时调度的统计任务也按天分区，数据每小时追加，则每小时将多读取大量的无用数据，增加不必要的费用。

您可以根据实际的业务情况选择分区字段，除了日期和时间，也可以使用其他的枚举值个数相对固定的字段，例如渠道、国家和省份地市。或者使用时间和其他字段共同作为分区字段。一般而言，推荐使用二级分区，因为最大的单表最多只支持6万个分区。

合理设置表生命周期

您可以根据数据本身的使用情况，在创建表时对表设置生命周期，MaxCompute会及时删除超过生命周期的数据，达到节省存储空间的目的。

例如，创建一张生命周期为100天的表。如果这张表或者分区的最后修改时间超过了100天将会被删掉。

```
CREATE TABLE test3 (key boolean) PARTITIONED BY (pt string, ds string) LIFECYCLE 100;
```

生命周期最小单位是分区，所以一个分区表中，如果部分分区达到了生命周期的阈值，那么这些分区会被直接删掉，未达到生命周期阈值的分区不受影响。

已经创建的表可以通过如下命令修改生命周期。详情请参见生命周期操作。

```
ALTER TABLE table_name SET lifecycle days;
```

删除废表

建议您定期地删除访问跨度大（即长期不会访问）的废表，因为这些表的意义并不大，会极大的浪费存储资源，例如：

- 3个月内没有被访问的表。
- 一张表是非分区表，同时最近1个月内没有被访问。
- 存储为0KB的表，即没有存储的表。

6.5. 数据上传下载成本优化

本文为您介绍如何优化数据上传和下载的同步成本。

- 尽可能使用经典网络和VPC网络

  您可以使用内部网络（经典网络或VPC）实现零成本数据导入和导出。网络设置详情请参见Endpoint。

- 合理利用ECS的公共下载资源

  如果您的ECS使用包月资源，可以使用Tunnel等数据同步工具，将MaxCompute数据同步到ECS，然后下载到本地。详情请参见导出SQL的运行结果。

- Tunnel文件上传优化

  小文件会消耗更多计算资源，建议当文件量积累较大时一次性上传。例如，调用Tunnel SDK时，缓存达到64 MB提交一次。

- 合理预估VPC带宽

  当数据在IDC机房时，如果您需要通过专线同步数据到MaxCompute，请预估带宽，平衡数据同步与带宽之间的成本。例如，50 TB数据上云，同步1天，预估需要5 GB带宽。带宽的计算方式为 `50（TB）×1024（GB）×8（bit）/（24（小时）×3600（秒））=4.7 GB/s` 。

6.6. 成本追踪

本文介绍如何追踪成本的消耗，优化资源的使用以及减少费用。

阿里云为您提供了三个成本管理工具：

- 账单明细：您可以在阿里云的费用中心看到。
- 使用记录：它会记录每条SQL的使用，复杂度、计量时间以及一天24小时的存储情况和下行流量等明细。
- 命令行工具：您可以通过命令行工具还原之前操作，查看如何产生了所谓的“贵SQL”。

账单明细

建议您定期查看账单，及时优化使用成本。您可以通过控制台查看账单明细。包年包月的出账时间是次日的12点，按量计费的出账时间是次日的9点。账单明细的查看方法，请参见查看账单详情。

使用记录

当您在账单里面发现某一个项目的计费可能突然在某一天达到了几千元，是平常账单的多倍，类似这样的异常情况，需要查看它的明细。您可以下载使用记录，查看异常记录的详细情况。使用记录的下载请参见查看账单详情。

对于存储费用，按照小时推送计量信息，1天24次。计算存储价格时需要将字节数相加并计算一个24小时的平均值，之后再按照阶梯定价的公式进行计算最终得到存储价格。

计量信息是以每一条任务的结束时间为准，如果某条任务的结束时间是第二天凌晨，那么这条任务的计量时间就会计入第二天，不会计入第一天。

对于下载费用，内网也就是经典网络的下行流量是不收费的，上行流量也是不收费的。只有使用公网的时候，下行流量才会计费。

命令行

当发现异常的SQL时，可以通过命令行还原当时的情景。

- 通过使用记录或者 `show p;` 命令获取异常数据的InstanceId，使用 `wait InstanceId` 命令获取其Logview，即SQL的详细日志，并将logview打印出来查看执行时的具体情况。

  > 说明 目前只能获取7天之内的logview信息，更早的信息无法获得。

- 使用 `desc instance instid` 将SQL显示在控制台里面。

6.7. 计费命令参考

本文介绍常用的计费命令。

语法表达式	用途	是否收费	样例
TUNNEL DOWNLOAD	下载数据（经典网络）	否	`TUNNEL DOWNLOAD table_name e:/table_name.txt;` 配置经典网络 `Endpoint:http://dt.cn-shanghai.maxcompute.aliyun-inc.com` Endpoint的配置，请参见Endpoint。
TUNNEL DOWNLOAD	下载数据（公网）	收费	`TUNNEL DOWNLOAD table_name e:/table_name.txt;` 配置外网网络`Endpoint:http://dt.cn-shanghai.maxcompute.aliyun.com` Endpoint的配置，请参见Endpoint。

TUNNEL UPLOAD	上传数据	否	`TUNNEL UPLOAD e:/table_name.txt table_name;`
COST SQL	费用预估	否	`COST SQL SELECT * FROM table_name;`
INSERT OVERWRITE...SELECT	数据更新	收费	`INSERT OVERWRITE TABLE table_name PARTITION (sale_date='20180122') SELECT shop_name, customer_id, total_price FROM sale_detail;`
DESC TABLE	查看表信息	否	`DESC table_name;`
DROP TABLE	删除表及表数据	否	`DROP TABLE if exists table_name;`
CREATE TABLE	创建表	否	`CREATE TABLE if not exists table_name (key string ,value bigint) PARTITIONED BY(p string);`
CREATE TABLE...SELECT	创建表	收费	`CREATE TABLE if not exists table_name AS SELECT * FROM a_tab;`
INSERT INTO TABLE...VALUES	快速插入常量数据	否	`INSERT INTO TABLE table_name partition (p) (key,p) VALUES ('d','20170101'), ('e','20170101'), ('f','20170101');`
INSERT INTO TABLE...SELECT	插入数据	收费	`INSERT INTO TABLE table_name SELECT shop_name, customer_id, total_price FROM sale_detail;`
SELECT UDF [NOT COUNT or All] FROM TABLE	查询表数据	收费	`SELECT sum(a) FROM table_name;`
SET FLAG	会话设置	否	`SET odps.sql.allow.fullscan=true;`
JAR MR	运行MapReduce作业	收费	`JAR -l com.aliyun.odps.mapred.example.WordCount wc_in wc_out`
ADD JAR/FILE/ARCHIVE/TABLE	注册资源	否	`ADD jar data\resources\mapreduce-examples.jar -f;`
DROP JAR/FILE/ARCHIVE/TABLE	删除资源	否	`DROP RESOURCE sale.res`
LIST RESOURCES	查看资源列表	否	`LIST RESOURCES;`
GET RESOURCES	下载资源	否	`GET RESOURCES odps-udf-examples.jar d:\;`
CREATE FUNCTIONS	注册函数	否	`CREATE FUNCTION test_lower ;`
DROP FUNCTIONS	删除函数	否	`DROP FUNCTION test_lower;`
LIST FUNCTIONS	查看函数列表	否	`LIST FUNCTIONS;`
ALTER TABLE...DROP PARTITION	删除表分区	否	`ALTER TABLE user DROP if exists partition(region='hangzhou',dt='20150923');`

TRUNCATE TABLE	删除非分区表数据	否	`TRUNCATE TABLE table_name;`
CREATE EXTERNAL TABLE	创建外表	否	`CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_csv_external… LOCATION 'oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/Demo/'`
SELECT [EXTERNAL] TABLE	读取外表	收费	`SELECT recordId, patientId, direction FROM ambulance_data_csv_external WHERE patientId > 25;`
SHOW TBALES	列出当前项目空间下所有的表	否	`SHOW TABLES;`
SHOW PARTITIONS table_name	列出一张表中的所有分区	否	`SHOW PARTITIONS <table_name>`
SHOW INSTANCE/SHOW P	返回由当前用户创建的实例信息	否	`SHOW INSTANCES/SHOW P`
WAIT INSTANCE	返回指定实例Logview	否	`WAIT 20131225123302267gk3u6k4y2`
STATUS INSTANCE	返回指定实例的状态	否	`STATUS 20131225123302267gk3u6k4y2`
KILL INSTANCE	停止您指定的实例	否	`KILL 20131225123302267gk3u6k4y2`

6.8. MaxCompute账单分析最佳实践

如果您想了解费用的分布情况或确保在使用MaxCompute产品产生的费用不超出预期时，您可以通过获取MaxCompute账单进行分析，为资源使用率最大化及降低成本提供有效支撑。本文为您介绍如何通过用量明细表分析MaxCompute的费用分布情况。

背景信息

MaxCompute是一款大数据分析平台，其计算资源的计费方式分为包年包月和按量计费两种。MaxCompute每天以项目为维度进行计费，账单会在第二天06:00前生成。更多MaxCompute计量计费信息，请参见计费项与计费方式概述。

MaxCompute会在数据开发阶段或者产品上线前发布账单波动（通常情况下为消费增长）信息。您可以自助分析账单波动情况，再对MaxCompute项目的作业进行优化。您可以在阿里云费用中心下载阿里云所有收费产品的用量明细。更多获取和下载账单操作，请参见查看账单详情。

> **说明** 通常您需要使用阿里云账号查看账单详情。如果RAM用户需要查看账单信息，需要阿里云账号选择策略并对RAM用户授权，详情请参见费用中心RAM配置策略。

上传用量明细数据至MaxCompute

1. 使用MaxCompute客户端（odpscmd）按如下示例语句创建表maxcomputefee。

```
DROP TABLE IF EXISTS maxcomputefee;
CREATE TABLE IF NOT EXISTS maxcomputefee
(
    projectid STRING COMMENT '项目编号'
    ,feeid STRING COMMENT '计费信息编号'
    ,type STRING COMMENT '数据分类，包括Storage、ComputationSQL、DownloadEx等'
    ,storage BIGINT COMMENT '存储（Byte）'
    ,endtime STRING COMMENT '结束时间'
    ,computationsqlinput BIGINT COMMENT 'SQL/交互式分析读取量（Byte）'
    ,computationsqlcomplexity DOUBLE COMMENT 'SQL复杂度'
    ,uploadex BIGINT COMMENT '公网上行流量Byte'
    ,download BIGINT COMMENT '公网下行流量Byte'
    ,cu_usage DOUBLE COMMENT 'MR计算时*second'
    ,input_ots BIGINT COMMENT '访问OTS的数据输入量'
    ,input_oss BIGINT COMMENT '访问OSS的数据输入量'
    ,starttime STRING COMMENT '开始时间'
    ,source_type STRING COMMENT '计算资源'
    ,source_id STRING COMMENT 'DataWorks调度任务ID'
);
```

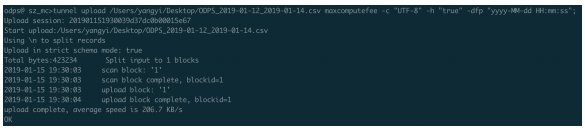
账单明细字段说明如下：

- 项目编号：当前账号或RAM用户对应的阿里云账号的MaxCompute项目列表。
- 计量信息编号：以存储、计算、上传和下载的任务ID为计费信息编号，SQL为InstanceID，上传和下载为Tunnel SessionId。
- 数据分类：Storage（存储）、ComputationSql（计算）、UploadIn（内网上传）、UploadEx（外网上传）、DownloadIn（内网下载）、DownloadEx（外网下载）。按照计费规则只有红色部分为实际计费项目。
- 存储（Byte）：每小时读取的存储量，单位为Byte。
- 开始时间或结束时间：按照实际作业执行时间进行计量，只有存储是按照每个小时取一次数据。
- SQL/交互式分析读取量（Byte）：SQL计算项，每一次SQL执行时SQL的Input数据量，单位为Byte。
- SQL复杂度：SQL的复杂度，为SQL计费因子之一。
- 公网上行流量（Byte）或公网下行流量（Byte）：分别为外网上传或下载的数据量，单位为Byte。
- MR/Spark作业计算（Core*Second）：MapReduce或Spark作业的计算时单位为`Core*Second`，需要转换为计算时Hour。
- SQL读取量_访问OTS（Byte）、SQL读取量_访问OSS（Byte）：外部表实施收费后的读取数据量，单位为Byte。
- 计算资源规格（按量计费）：计量信息所属项目的计算资源规格，若值为NULL，则表示按量计费标准版；若值为OdpsDev，则表示按量计费开发者版。
- 计算资源规格（包年包月）：计量信息所属项目的计算资源规格。若值为NULL，则表示包年包月标准版；若值为OdpsPlus160CU150TB、OdpsPlus320CU300TB或OdpsPlus600CU500TB，则分别表示存储密集型160套餐、存储密集型320套餐或存储密集型600套餐。
- DataWorks调度任务ID：作业在DataWorks上的调度节点ID。若值NULL，则表示非DataWorks调度节点提交的作业；若值为一串数字ID，则表示作业对应的DataWorks调度节点ID。您可以在DataWorks对应的项目中使用该ID搜索到具体任务。

2. 使用Tunnel上传数据。

   在上传CSV文件时，您需确保CSV文件中的列数、数据类型必须与表maxcomputefee的列数、数据类型保持一致，否则会导入失败。

```
tunnel upload ODPS_2019-01-12_2019-01-14.csv maxcomputefee -c "UTF-8" -h "true" -dfp "yyyy-MM-dd HH:mm:ss";
```



> **说明**
> - Tunnel的配置详情请参见Tunnel命令。
> - 您也可以通过DataWorks的数据导入功能来执行此操作，具体请参见数据集成导入数据。

3. 执行如下语句验证数据。

```
SELECT * FROM maxcomputefee limit 10;
```

通过SQL分析账单数据

1. 分析SQL费用。云上用户使用MaxCompute，95%的用户通过SQL即可满足需求，SQL也在消费增长中占很大比例。

> **说明** 一次SQL计算费用 = 计算输入数据量×SQL复杂度×单价（0.3元/GB）

```
--分析SQL消费，按照sqlmoney排行。
SELECT  to_char(endtime,'yyyymmdd') as ds,feeid as instanceid
        ,projectid
        ,computationsqlcomplexity  --复杂度
        ,SUM((computationsqlinput / 1024 / 1024 / 1024)) as computationsqlinput  --数据输入量（GB）
        ,SUM((computationsqlinput / 1024 / 1024 / 1024)) * computationsqlcomplexity * 0.3 AS sqlmoney
FROM    maxcomputefee
WHERE   TYPE = 'ComputationSql'
AND to_char(endtime,'yyyymmdd') >= '20190112'
GROUP BY to_char(endtime,'yyyymmdd'),feeid
         ,projectid
         ,computationsqlcomplexity
ORDER BY sqlmoney DESC
LIMIT   10000
;
```

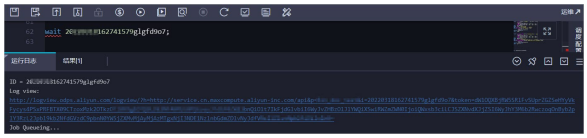
查询结果如下。

	A	B	C	D	E	F
1	ds	instanceid	projectid	computationsqlcor	computationsqlinp	sqlmoney
2	20190114	20190113220731203g		1.5	939.2598592275754	422.6669366524089
3	20190113	20190112211606543g		1.5	939.0618489095941	422.5778320093173
4	20190114	20190113214053411g		1.5	797.8894629115239	359.0502583101857
5	20190113	20190112204652265g		1.5	797.6614840412512	358.94766781856305
6	20190114	20190113212053799g		1.5	700.2501222227706	315.1125550024677
7	20190113	20190112202519334g		1.5	700.0137415388599	315.006183692486 96
8	20190114	20190113222308193g		1.0	750.0447742938995	225.01343228816987
9	20190113	20190112212657773g		1.0	749.9437991976738	224.98313975930213
10	20190114	20190113173351510g		1.0	546.8404815010726	164.05214445032178

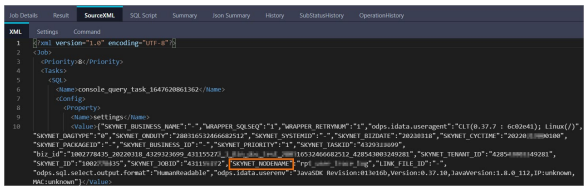
根据查询结果可以得到以下结论：

- 大作业可以减小数据读取量、降低复杂度、优化费用成本。
- 可以按照ds字段（按照天）进行汇总，分析某个时间段内的SQL消费金额走势。例如利用本地Excel或Quick BI等工具绘制折线图等方式，更直观的反应作业的趋势。
- 根据执行结果可以定位到需要优化的点，方法如下：
  a. 通过查询的instanceid，获取目标实例运行日志的Logview地址。

     在MaxCompute客户端（odpscmd）或DataWorks中执行 `wait <instanceid>;` 命令，查看instanceid的运行日志。

  b. 在浏览器中打开Logview的URL地址，在Logview页面的SourceXML页签，获取该实例的SKYNET_NODENAME。

     > **说明**
     > - 关于Logview的介绍，详情请参见使用Logview 2.0查看Job运行信息。
     > - 如果获取不到SKYNET_NODENAME或SKYNET_NODENAME无值，您可以在SQL Script页签获取代码片段后，在DataWorks上通过搜索代码片段，获取目标节点进行优化。详情请参见DataWorks代码搜索。

  c. 在DataWorks中，搜索查询到的SKYNET_NODENAME，对目标节点进行优化。

2. 分析作业增长趋势。通常费用的增长是由于重复执行或调度属性配置不合理造成作业量暴涨。

```
--分析作业增长趋势。
SELECT   TO_CHAR(endtime,'yyyymmdd') AS ds
        ,projectid
        ,COUNT(*) AS tasknum
FROM    maxcomputefee
WHERE   TYPE = 'ComputationSql'
AND     TO_CHAR(endtime,'yyyymmdd') >= '20190112'
GROUP BY TO_CHAR(endtime,'yyyymmdd')
         ,projectid
ORDER BY tasknum DESC
LIMIT   10000
;
```

执行结果如下。

	A	B	C
1	ds	projectid	tasknum
2	20190112		311
3	20190113		304
4	20190114		282

从执行结果可以看出12~14日提交到MaxCompute且执行成功的作业数的波动趋势。

3. 分析存储费用。

> **说明** 存储费用的计费规则相对复杂。明细中是按每个小时取一次得出的数据。按照MaxCompute存储计费规则，会先整体24小时求和，再将平均之后的值进行阶梯收费。详情请参见存储费用。

```
--分析存储费用。
SELECT  t.ds
        ,t.projectid
        ,t.storage
        ,CASE    WHEN t.storage < 0.5 THEN 0.01
                 WHEN t.storage >= 0.5 AND t.storage <= 10240 THEN t.storage*0.0072
                 WHEN t.storage > 10240 AND t.storage <= 102400 THEN (10240*0.0072+(t.storage-10240)*0.006)
                 WHEN t.storage > 102400  THEN (10240*0.0072+(102400-10240)*0.006+(t.storage-102400)*0.004)
        END storage_fee
FROM    (
            SELECT  to_char(starttime,'yyyymmdd') as ds
                    ,projectid
                    ,SUM(storage/1024/1024/1024)/24 AS storage
            FROM    maxcomputefee
            WHERE   TYPE = 'Storage'
            and to_char(starttime,'yyyymmdd') >= '20190112'
            GROUP BY to_char(starttime,'yyyymmdd')
                     ,projectid
        ) t
ORDER BY storage_fee DESC
;
```

执行结果如下。

	A	B	C	D
1	ds	projectid	storage	storage_fee
2	20190113	[redacted]	8467.066039182557	73.31215472913347
3	20190112	[redacted]	4527.7404568577185	40.22181983760483
4	20190114	[redacted]	2672.068578289938	24.63417605763548

根据执行结果可以分析得出如下结论：

- 存储在12日有一个增长的过程，但在14日出现降低。
- 存储优化，建议为表设置生命周期，删除长期不使用的临时表等。

4. 分析下载费用。

   对于外网或者跨区域的数据下载，MaxCompute将按照下载的数据量进行计费。

   > 说明 一次下载费用=下载数据量×单价（0.8元/GB）。

```
--分析下载消费明细。
SELECT  TO_CHAR(starttime,'yyyymmdd') AS ds
        ,projectid
        ,SUM((download/1024/1024/1024)*0.8) AS download_fee
FROM    maxcomputefee
WHERE   type = 'DownloadEx'
AND     TO_CHAR(starttime,'yyyymmdd') >= '20190112'
GROUP BY TO_CHAR(starttime,'yyyymmdd')
         ,projectid
ORDER BY download_fee DESC
;
```

   按照执行结果也可以分析出某个时间段内的下载费用走势。另外可以通过 `tunnel show history` 命令查看具体历史信息，具体命令详见Tunnel命令。

5. 分析MapReduce作业消费。

   > 说明 MapReduce作业当日计算费用 = 当日总计算时×单价（0.46元）

```
--分析MapReduce作业消费。
SELECT  TO_CHAR(starttime,'yyyymmdd') AS ds
        ,projectid
        ,(cu_usage/3600)*0.46 AS mr_fee
FROM    maxcomputefee
WHERE   type = 'MapReduce'
AND     TO_CHAR(starttime,'yyyymmdd') >= '20190112'
GROUP BY TO_CHAR(starttime,'yyyymmdd')
         ,projectid
         ,cu_usage
ORDER BY mr_fee DESC
;
```

6. 分析外部表作业（OTS和OSS）。

> 说明 一次SQL外部表计算费用 = 计算输入数据量×单价（0.03元/GB）

```
--分析OTS外部表SQL作业消费。
SELECT  TO_CHAR(starttime,'yyyymmdd') AS ds
        ,projectid
        ,(computationsqlinput/1024/1024/1024)*1*0.03 AS ots_fee
FROM    maxcomputefee
WHERE   type = 'ComputationSqlOTS'
AND     TO_CHAR(starttime,'yyyymmdd') >= '20190112'
GROUP BY TO_CHAR(starttime,'yyyymmdd')
         ,projectid
         ,computationsqlinput
ORDER BY ots_fee DESC
;
--分析OSS外部表SQL作业消费。
SELECT  TO_CHAR(starttime,'yyyymmdd') AS ds
        ,projectid
        ,(computationsqlinput/1024/1024/1024)*1*0.03 AS oss_fee
FROM    maxcomputefee
WHERE   type = 'ComputationSqlOSS'
AND     TO_CHAR(starttime,'yyyymmdd') >= '20190112'
GROUP BY TO_CHAR(starttime,'yyyymmdd')
         ,projectid
         ,computationsqlinput
ORDER BY oss_fee DESC
;
```

7. 分析Lightning查询费用。

> 说明 一次Lightning查询费用 = 查询输入数据量×单价（0.03元/GB）

```
SELECT  to_char(endtime,'yyyymmdd') as ds,feeid as instanceid
        ,projectid
        ,computationsqlcomplexity
        ,SUM((computationsqlinput / 1024 / 1024 / 1024)) as computationsqlinput
        ,SUM((computationsqlinput / 1024 / 1024 / 1024)) * computationsqlcomplexity * 0.03 AS sqlmoney
FROM    maxcomputefee
WHERE   TYPE = 'LightningQuery'
--AND to_char(endtime,'yyyymmdd') >= '20190112'
GROUP BY to_char(endtime,'yyyymmdd'),feeid
         ,projectid
         ,computationsqlcomplexity
ORDER BY sqlmoney DESC
LIMIT   10000
;
```

8. 分析Spark计算费用。

> 说明 Spark作业当日计算费用 = 当日总计算时×单价（0.66元/计算时）

```
--分析Spark作业消费。
SELECT  TO_CHAR(starttime,'yyyymmdd') AS ds
        ,projectid
        ,(cu_usage/3600)*0.66 AS mr_fee
FROM    maxcomputefee
WHERE   type = 'spark'
AND     TO_CHAR(starttime,'yyyymmdd') >= '20190112'
GROUP BY TO_CHAR(starttime,'yyyymmdd')
         ,projectid
         ,cu_usage
ORDER BY mr_fee DESC
;
```

更多信息

如果您想了解更多关于费用成本优化的文章，请参见成本优化概述。

6.9. 统计MaxCompute TOPN费用账号及耗时作业

数据开发者在使用MaxCompute开发过程中，需要统计MaxCompute项目中账号的费用以及作业的耗时情况，助力合理规划和调整作业。本文为您介绍如何通过MaxCompute元数据（Information Schema）统计TOP费用账号及耗时作业，同时通过钉钉推送到客户群。

背景信息

通常，数据开发者会通过DataWorks标准模式使用MaxCompute，MaxCompute会在Information Schema中记录所有作业的执行账号为同一个主账号，只有小部分的作业执行账号为RAM用户。此时数据开发者会关注如何统计各个账号的费用和耗时作业。MaxCompute提供如下方案解决这两个问题：

- 账号费用：您可以通过账单详情中的用量明细来查询，但是这种方式无法将用量明细归属到对应的RAM用户。Information Schema视图中的TASKS_HISTORY会记录MaxCompute项目内已完成的作业详情，且保留近14天数据。您可以将TASKS_HISTORY中的数据备份到指定MaxCompute项目中，基于该数据统计TOP费用账号。
- 耗时作业：您可以通过TASKS_HISTORY中的数据统计TOP耗时作业。

更多关于Information Schema的功能及使用限制，请参见Information Schema概述。

统计MaxCompute TOPN费用账号及耗时作业的流程如下：

1. 步骤一：获取Information Schema服务。
2. （可选）步骤二：对除Project Owner外的用户授权。
3. 步骤三：下载并备份元数据。
4. 步骤四：创建统计TOPN费用账号及耗时作业。
5. 步骤五：创建钉钉群机器人并推送TOPN费用账号及耗时作业信息。
6. 步骤六：配置上下游节点调度属性并运行节点。

步骤一：获取Information Schema服务

自2020年12月1日起，对于新创建的MaxCompute项目，MaxCompute默认提供Information Schema相关的元数据视图，您无需手工安装Information Schema权限包。

对于存量MaxCompute项目，在您开始使用Information Schema服务前，需要以项目所有者（Project Owner）或具备Super_Administrato管理角色的RAM用户身份安装Information Schema权限包，获得访问项目元数据的权限。更多为用户授权管理角色操作信息，请参见将角色赋予用户。安装方式有如下两种：

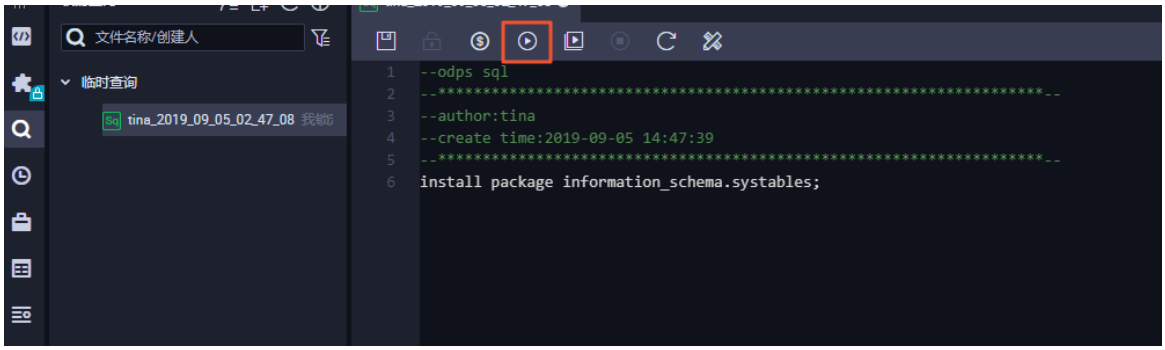
- 登录MaxCompute客户端，执行如下命令：

```
install package Information_Schema.systables;
```

- 登录DataWorks控制台，进入**临时查询**界面。更多临时查询操作详情，请参见使用临时查询运行SQL语句（可选）。执行如下命令：

```
install package Information_Schema.systables;
```

执行示例如下。



> **说明** 如果统计多个MaxCompute项目的元数据，您需要分别对各个MaxCompute项目安装Information Schema权限包。然后把各个MaxCompute项目的元数据的备份数据插入到同一个表中做集中统计分析。

（可选）步骤二：对除Project Owner外的用户授权

Information Schema的视图包含了项目级别的所有用户数据，默认项目所有者可以查看。如果项目内其他用户或角色需要查看，需要进行授权，请参见基于Package跨项目访问资源。

授权语法如下。

```
grant <actions> on package Information_Schema.systables to user <user_name>;
grant <actions> on package Information_Schema.systables to role <role_name>;
```

- actions：待授予的操作权限，取值为Read。
- user_name：已添加至项目中的阿里云账号或RAM用户。

  您可以通过MaxCompute客户端执行 `list users;` 命令获取用户账号。
- role_name：已添加至项目中的角色。

您可以通过MaxCompute客户端执行 `list roles;` 命令获取角色名称。

授权示例如下。

```
grant read on package Information_Schema.systables to user RAM$Bob@aliyun.com:user01;
```

步骤三：下载并备份元数据

在MaxCompute项目上创建元数据备份表，并定时将元数据写入备份表中。以MaxCompute客户端为例，操作流程如下：

1. 登录MaxCompute客户端，执行如下命令创建元数据备份表。

```
--project_name为MaxCompute项目名称。
create table if not exists <project_name>.information_history
(
    task_catalog STRING
    ,task_schema STRING
    ,task_name STRING
    ,task_type STRING
    ,inst_id STRING
    ,`status` STRING
    ,owner_id STRING
    ,owner_name STRING
    ,result STRING
    ,start_time DATETIME
    ,end_time DATETIME
    ,input_records BIGINT
    ,output_records BIGINT
    ,input_bytes BIGINT
    ,output_bytes BIGINT
    ,input_tables STRING
    ,output_tables STRING
    ,operation_text STRING
    ,signature STRING
    ,complexity DOUBLE
    ,cost_cpu DOUBLE
    ,cost_mem DOUBLE
    ,settings STRING
    ,ds STRING
);
```

2. 进入DataWorks数据开发界面，创建ODPS SQL节点（information_history）并配置定时调度，用于定时将数据写入备份表information_history。完成后单击左上角图标保存。

   创建ODPS SQL节点操作，请参见创建ODPS SQL节点。

   ODPS SQL节点运行的命令示例如下：

```
--project_name为MaxCompute项目名称。
use <project_name>;
insert into table <project_name>.information_history select * from information_schema.tasks_history where ds ='${datetime1}'
;
```

   `${datetime1}` 为DataWorks的调度参数，您需要在ODPS SQL节点右侧，单击**调度配置**，在**基础属性**区域配置**参数**值为 `datetime1=${yyyymmdd}` 。

   > **说明** 如果需要同时对多个MaxCompute项目的元数据进行统计分析，您可以创建多个ODPS SQL节点，将这些MaxCompute项目的元数据写入到同一张数据备份表中。

步骤四：创建统计TOPN费用账号及耗时作业

TASKS_HISTORY视图中的settings会记录上层调度或用户传入的信息，以JSON格式存储。包含的具体信息有：useragent、bizid、skynet_id和skynet_nodename。您可以通过settings字段定位到创建作业的RAM用户信息。因此您可以基于备份数据表计算TOPN费用账号及耗时作业。操作流程如下：

1. 登录MaxCompute客户端，创建一张RAM用户明细表user_ram，记录需要统计的账号及账号ID。

   命令示例如下：

```
create table if not exists <project_name>.user_ram
(
    user_id STRING
    ,user_name STRING
);
```

2. 创建一张统计账号费用的明细表cost_topn，记录TOPN费用账号明细。

命令示例如下：

```
create table if not exists <project_name>.cost_topn
(
    cost_sum DECIMAL(38,5)
    ,task_owner STRING
)
partitioned by
(
    ds STRING
);
```

3. 建一张统计耗时作业的明细表time_topn，记录TOPN耗时作业明细。

   命令示例如下：

```
create table if not exists <project_name>.time_topn
(
    inst_id STRING
    ,cost_time BIGINT
    ,task_owner STRING
)
partitioned by
(
    ds STRING
);
```

4. 进入DataWorks数据开发界面，创建ODPS SQL节点（topn）并配置定时调度，用于定时将cost_topn表中统计的数据写入user_ram表。完成后单击左上角图标保存。

   创建ODPS SQL节点操作，请参见创建ODPS SQL节点。

   ODPS SQL节点运行的命令示例如下：

```
--开启2.0数据类型开关。2.0数据类型详情，请参见2.0数据类型版本。
set odps.sql.decimal.odps2=true;
--将元数据写入cost_topn、time_topn表。user_id为账号ID。您可以在个人信息个人信息页面查看账号ID。
insert into table <project_name>.cost_topn partition (ds = '${datetime1}')
select
nvl(cost_sum,0) cost_sum
        ,case when a.task_owner='<user_id>' or a.task_owner='<user_id>' or a.task_owner='<user_id>' then b.user_name
              else a.task_owner
         end task_owner
from    (
            select  inst_id
                    ,owner_name
                    ,task_type
                    ,a.input_bytes
                    ,a.cost_cpu
                    ,a.status
                    ,case    when a.task_type = 'SQL' then cast(a.input_bytes/1024/1024/1024 * a.complexity * 0.3 as DECIMAL
(18,5) )
                             when a.task_type = 'SQLRT' then cast(a.input_bytes/1024/1024/1024 * a.complexity * 0.3 as DECIM
AL(18,5) )
                             when a.task_type = 'CUPID' and a.status='Terminated'then cast(a.cost_cpu/100/3600 * 0.66 as DEC
IMAL(18,5) )
                             else 0
                     end cost_sum
                    ,a.settings
                    ,get_json_object(settings, "$.SKYNET_ONDUTY") owner
                    ,case    when get_json_object(a.settings, "$.SKYNET_ONDUTY") is null then owner_name
                             else get_json_object(a.settings, "$.SKYNET_ONDUTY")
                     end task_owner
            from    information_history
            where   ds = '${datetime1}'
        ) a
left join <project_name>.user_ram b
on      a.task_owner = b.user_id;
insert into table <project_name>.time_topn partition(ds = '${datetime1}')
select  inst_id
        ,cost_time
        ,case    when a.task_owner='<user_id>' or a.task_owner='<user_id>' or a.task_owner='<user_id>' then b.user_name
                 else a.task_owner
         end task_owner
from    (
            select  inst_id
                    ,task_type
                    ,status
                    ,datediff(a.end_time, a.start_time, 'ss') AS cost_time
                    ,case    when get_json_object(a.settings, "$.SKYNET_ONDUTY") is null then owner_name
                             else get_json_object(a.settings, "$.SKYNET_ONDUTY")
                     end task_owner
            from    <project_name>.information_history a
            where   ds = '${datetime1}'
        ) a
left join <project_name>.user_ram b
on      a.task_owner = b.user_id
;
```

> **说明** 示例中的 `task_type = 'SQL'` 表示SQL作业，`task_type = 'SQLRT'` 表示查询加速作业，`task_type = 'CUPID'` 表示Spark作业。如果需要统计其他计费作业，例如MapReduce、Lightning（交互式分析）、Mras，您可以按照计费公式添加相应代码行。计费详情，请参见计算费用（按量计费）。

`${datetime1}` 为DataWorks的调度参数，您需要在ODPS SQL节点右侧，单击**调度配置**，在**基础属性**区域配置参数值为 `datetime1=${yyyymmdd}` 。

步骤五：创建钉钉群机器人并推送TOPN费用账号及耗时作业信息

以PC端为例，创建钉钉群机器人并推送TOPN费用账号及耗时作业信息的操作流程如下：

1. 创建钉钉群机器人。
   i. 选择目标钉钉群，单击右上角的图标。
   ii. 在**群设置**面板，单击**智能群助手**。
   iii. 在**智能群助手**面板，单击**添加机器人**。
   iv. 在**群机器人**对话框的**添加机器人**区域，单击图标。

v. 在**群机器人**对话框，单击**自定义**机器人。

vi. 在**机器人详情**对话框，单击**添加**。

vii. 在**添加机器人**对话框，编辑机器人信息。

属性名称	设置规则
头像	单击头像右下角的图标来编辑头像。
机器人名字	输入机器人名字。
安全设置	完成必要的安全设置（至少选择1种），勾选**我已阅读并同意《自定义机器人服务及免责条款》**，单击**完成**。 安全设置有3种方式： ■ **自定义关键词**：最多可以设置10个关键词。 ■ **加签**：勾选**加签**可以获取到机器人的密钥。 ■ **IP地址（段）**：只有来自IP地址范围内的请求才会被正常处理。

viii. 在**添加机器人**对话框，复制生成的**Webhook**地址。单击**完成**。

> **注意** 请保管好此Webhook地址，不要公布在外部网站上，泄露后会有安全风险。

2. 通过Intellij IDEA创建Maven项目并编译推送钉钉群消息的Java程序，编译完成后生成JAR包。

   Intellij IDEA操作详情，请单击Intellij IDEA工具界面右上角的**Help**获取。

   i. 配置Pom依赖。

   Pom依赖如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>DingTalk_Information</groupId>
    <artifactId>DingTalk_Information</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>com.aliyun.odps</groupId>
            <artifactId>odps-sdk-core</artifactId>
            <version>0.35.5-public</version>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.15</version>
            <exclusions>
                <exclusion>
                    <groupId>com.sun.jmx</groupId>
                    <artifactId>jmxri</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>com.sun.jdmk</groupId>
                    <artifactId>jmxtools</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>javax.jms</groupId>
                    <artifactId>jms</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>com.aliyun</groupId>
            <artifactId>alibaba-dingtalk-service-sdk</artifactId>
            <version>1.0.1</version>
        </dependency>
        <dependency>
            <groupId>com.aliyun.odps</groupId>
            <artifactId>odps-jdbc</artifactId>
            <version>3.0.1</version>
            <classifier>jar-with-dependencies</classifier>
        </dependency>
    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>2.4.1</version>
                <configuration>
                    <!-- get all project dependencies -->
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                    <!-- MainClass in mainfest make a executable jar -->
                    <archive>
                        <manifest>
                            <mainClass>com.alibaba.sgri.message.test</mainClass>
                        </manifest>
                    </archive>
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id>
                        <!-- bind to the packaging phase -->
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

ii. 开发Java程序并生成JAR包topn_new.jar。

Java代码示例如下：

```
package com.alibaba.sgri.message;
import java.io.IOException;
import java.util.concurrent.atomic.AtomicInteger;
import com.aliyun.odps.Instance;
import com.aliyun.odps.Odps;
import com.aliyun.odps.OdpsException;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.data.ResultSet;
import com.aliyun.odps.task.SQLTask;
import com.dingtalk.api.DefaultDingTalkClient;
import com.dingtalk.api.DingTalkClient;
import com.dingtalk.api.request.OapiRobotSendRequest;
import com.dingtalk.api.response.OapiRobotSendResponse;
import com.taobao.api.ApiException;
public class test {
    public static void main(String[] args) throws ApiException {
        if (args.length < 1) {
            System.out.println("请输入日期参数");
            System.exit(0);
        }
        System.out.println("开始读取数据");
        DingTalkClient client = new DefaultDingTalkClient(
                "https://oapi.dingtalk"
                        + ".com/robot/send?access_token=<机器人Webhook地址>\n");
        OapiRobotSendRequest request = new OapiRobotSendRequest();
        request.setMsgtype("markdown");
        OapiRobotSendRequest.Markdown markdown = new OapiRobotSendRequest.Markdown();
        //这里的日期作为参数
        markdown.setText(getContent(args[0]));
        markdown.setTitle("作业消费TOPN");
        request.setMarkdown(markdown);
        OapiRobotSendResponse response = client.execute(request);
        System.out.println("消息发送成功");
    }
    /**
     * 读取ODPS,获取要发送的数据
     */
    public static String getContent(String day) {
```

```
        Odps odps = createOdps();
        StringBuilder sb = new StringBuilder();
        try {
            //==================这是费用账号====================
            String costTopnSql = "select sum(cost_sum)cost_sum,task_owner from cost_topn where ds='" + day + "' " + "gro
up by task_owner order by cost_sum desc limit 5;";
            Instance costInstance = SQLTask.run(odps, costTopnSql);
            costInstance.waitForSuccess();
            ResultSet costTopnRecords = SQLTask.getResultSet(costInstance);
            sb.append("<font color=#FF0000 size=4>").append("费用账号TOPN(").append(day).append(
                    ")[按照阿里云按量付费计算]").append("</font>").append("\n\n");
            AtomicInteger costIndex = new AtomicInteger(1);
            costTopnRecords.forEach(item -> {
                sb.append(costIndex.getAndIncrement()).append(".").append("账号:");
                sb.append("<font color=#2E64FE>").append(item.getString("task_owner")).append("\n\n").append("</font>");
                sb.append("  ").append(" ").append("消费:").append("<font color=#2E64FE>").append(item.get("cost_sum"))
                        .append("元").append(
                        "</font>").append("\n\n")
                        .append("</font>");
            });
            //==================这是耗时作业====================
            String timeTopnSql = "select * from time_topn where ds='" + day + "' ORDER BY cost_time DESC limit 5;";
            Instance timeInstance = SQLTask.run(odps, timeTopnSql);
            timeInstance.waitForSuccess();
            ResultSet timeTopnRecords = SQLTask.getResultSet(timeInstance);
            sb.append("<font color=#FF8C00 size=4>").append("耗时作业TOPN(").append(day).append(")")
                    .append("\n\n").append("</font>");
            AtomicInteger timeIndex = new AtomicInteger(1);
            timeTopnRecords.forEach(item -> {
                sb.append(timeIndex.getAndIncrement()).append(".").append("作业:");
                sb.append("<font color=#2E64FE>").append(item.getString("inst_id")).append("\n\n").append("</font>");
                sb.append("   ").append("账号:").append("<font color=#2E64FE>").append(item.getString("task_owner")).appe
nd("\n\n").append("</font>");
                sb.append("   ").append("耗时:").append("<font color=#2E64FE>").append(item.get("cost_time"))
                        .append("秒").append(
                        "</font>").append("\n\n");
            });
        } catch (OdpsException | IOException e) {
            e.printStackTrace();
        }
        return sb.toString();
    }
    /**
     * 创建ODPS
     */
    public static Odps createOdps() {
        String project = "<project_name>";
        String access_id = "<AccessKey_id>";
        String access_key = "<AccessKey_Secret>";
        String endPoint = "http://service.odps.aliyun.com/api";
        Account account = new AliyunAccount(access_id, access_key);
        Odps odps = new Odps(account);
        odps.setEndpoint(endPoint);
        odps.setDefaultProject(project);
        return odps;
    }
}
```

> 说明 自定义钉钉群机器人开发API，请参见机器人开发。

iii. 上传生成的topn_new.jar包为MaxCompute资源。

上传MaxCompute资源操作，请参见创建MaxCompute资源。

3. 创建Shell节点（dingsend），引用topn_new.jar包并配置定时调度。

创建Shell节点操作，请参见Shell节点。

Shell节点运行的命令示例如下：

```
java -jar  topn_new.jar  $1
```

`$1` 为DataWorks的调度参数，您需要在Shell节点右侧，单击**调度配置**，在**基础属性**区域配置**参数**值为 `${yyyymmdd}` 。

步骤六：配置上下游节点调度属性并运行节点

在业务流程面板将information_history、topn和dingsend节点连线形成依赖关系，并配置每个节点的重跑属性和依赖的上游节点。配置完成后在节点上单击右键，选择**运行节点**即可。

依赖关系配置，请参见配置同周期调度依赖。

节点上下游配置，请参见配置节点上下文。

效果展示

钉钉群推送内容效果如下，仅供参考。

相关文档

- MaxCompute账单分析最佳实践
- 查看账单详情
- 在DataWorks标准模式下统计个人账号使用资源情况
- 利用InformationSchema与阿里云交易和账单管理API实现MaxCompute费用对账分摊统计

在线支持

如果您在使用MaxCompute的过程中有任何疑问或建议，欢迎填写钉钉群申请表单加入钉钉群进行反馈。

7.安全管理

7.1. MaxCompute项目设置RAM子账号为超级管理员

本文为您介绍在MaxCompute项目中如何将RAM子账号设置为超级管理员，并提供了超级管理员在成员管理、权限管理等方面的使用建议。

背景信息

日常工作中，为了保障数据安全，通常主账号为特定人员管理，使用MaxCompute的大部分用户都只持有RAM子账号。但是项目的所有者（Owner）只能为主账号，且MaxCompute的很多权限管理需要项目所有者才可以操作（例如项目级别Flag的设置、Package跨项目资源共享配置等），因此您需要一个拥有超级管理员权限的RAM子账号。

MaxCompute新增了内置的管理角色Super_Administrator，拥有项目内所有类型资源的全部权限以及项目的管理类权限，具体权限说明请参见角色规划与管理。

Super_Administrator角色可以由项目所有者授权给RAM子账号。RAM子账号获得该角色后，即可替代项目所有者对项目执行各种管理操作，包括常用的项目级别Flag设置以及所有资源权限的管理。

设置方法

建议您为有权限创建项目的RAM子账号指派Super_Administrator角色，这样该账号在管理DataWorks工作空间的同时，也可以管理DataWorks工作空间对应的MaxCompute项目。

> **说明**
> - 为RAM子账号授予创建项目的权限方法请参见给RAM子账号授权DataWorks相关管理权限。
> - 建议明确该RAM子账号持有人的责任，一个RAM子账号对应一个开发者，避免账号共用，以便更好地保障数据安全。
> - 一个项目中只能为一个RAM子账号指派Super_Administrator角色。您可以为其他需要有基本管理权限的账号可以赋予Admin角色权限。

选定RAM子账号后，使用该RAM子账号创建项目。此时项目所有者依然是主账号，主账号可以通过以下方式将Super_Administrator角色授权给RAM子账号：

- 通过MaxCompute客户端授权。

  假设主账号用户bob@aliyun.com是项目project_a的所有者，Allen是bob@aliyun.com中的RAM子账号。

  i. 使用主账号执行如下命令进行授权。

  ```
  --打开项目project_a。
  use project_a;
  --为项目project_a添加RAM子账号Allen。
  add user ram$bob@aliyun.com:Allen;
  --为RAM子账号Allen授权Super_Administrator角色权限。
  grant super_administrator TO ram$bob@aliyun.com:Allen;
  --为RAM子账号Allen授权Admin角色权限。
  grant admin TO ram$bob@aliyun.com:Allen;
  ```

  ii. 使用被授权RAM子账号执行如下命令查看账号自身权限。如果返回值中有Super_Administrator角色则说明赋权成功。

  ```
  show grants;
  ```

- 通过DataWorks授权。

  i. 登录DataWorks，进入工作空间配置。

  ii. （可选）添加RAM子账号为项目成员。如果RAM子账号已经为项目成员，可以忽略此步骤。

  a. 在**工作空间配置**页面左侧导航栏上，单击**成员管理**，进入**成员管理**页面。

  b. 单击右上角的**添加成员**。

  c. 在**添加成员**页面，从**待添加账号**列表中选择需要添加的组织成员显示在**已添加账号**列表中。

  > **说明** 单击**添加成员**对话框中的**刷新**，即可将RAM中当前阿里云主账号下存在的RAM子账号同步至**待添加账号**列表中。

  d. 勾选角色并单击**确定**，完成成员添加。

  iii. 为RAM子账号授权Super_Administrator角色。

  a. 在**工作空间配置**页面左侧导航栏上，单击**MaxCompute高级配置**。

  b. 在**MaxCompute高级配置**页面的左侧导航栏上，单击**自定义用户角色**。

c. 单击需要授权角色后的**成员管理**，从**待添加账号**列表中选择需要添加的组织成员显示在**已添加账号**列表中。

d. 单击**确定**，完成账号授权。

iv. 使用被授权RAM子账号执行如下命令查看账号自身权限。如果返回值中有Super_Administrator角色则说明赋权成功。

```
show grants;
```

使用说明

- 成员管理
  - MaxCompute支持云账号（主账号）和RAM子账号。为了更好的保障数据安全，建议项目中添加的成员均为此项目所有者的RAM子账号。

    主账号可以控制RAM子账号，在人员转岗离职等场景下主账号可以注销或更新对应的RAM子账号以保证数据安全。

    > **说明** 如果通过DataWorks进行项目成员管理，只能添加项目所有者的RAM子账号作为项目成员。

  - RAM子账号只能通过主账号添加。即使拥有Super_Administrator角色的超级管理员，也需要主账号先创建RAM子账号后，才可以将创建成功的RAM子账号添加到项目中。
  - 建议您只添加需要在当前项目进行数据开发（即会在当前项目执行作业）的用户为项目成员。对于有数据交互业务需求的用户，通过Package方式进行跨项目资源共享，避免将过多用户添加至项目内而增加了成员管理的复杂度。
  - 员工转岗或离职，需要先将其持有的RAM子账号在项目中移除，然后再通知项目所有者注销RAM子账号。如果是拥有Super_Administrator角色的RAM子账号持有者转岗或离职，则需要由主账号执行移除、注销账号操作。
- 权限管理
  - 建议通过角色进行权限管理，即权限和角色（Role）关联，角色（Role）和用户（User）关联。
  - 建议实施最小够用原则，避免权限过大造成安全隐患。
  - 跨项目使用数据时，建议通过Package方式实现，避免资源提供方增加成员管理成本，只需要管理Package即可。

  > **说明** 拥有Super_Administrator角色的RAM子账号本身已经拥有项目所有资源的查询和操作权限，所以无须再给自身授权。

- 权限审计

  可以通过MaxCompute的元数据服务提供的视图进行权限审计。详情请参见元数据视图列表。
- 成本管理

  关于成本管理，请参见查看账单详情。对于RAM子账号，需要主账号给RAM子账号赋予费用中心相关权限才可以进行账单数据的查询。授权方法请参见为RAM角色授权，所需权限如下：
  - AliyunBSSFullAccess：管理费用中心的权限。
  - AliyunBSSReadOnlyAccess：费用中心的访问、只读权限。
  - AliyunBSSOrderAccess：费用中心查看订单、支付订单及取消订单的权限。

  > **说明** 费用中心相关权限与MaxCompute项目的Super_Administrator角色无关联，需要单独授予给用户。

- 资源使用管理
  - 如果您使用MaxCompute包年包月计算资源，则可以通过MaxCompute管家对已经使用的计算资源进行查看、对所有计算资源进行管理。详情请参见MaxCompute管家。

- 如果您使用MaxCompute按量计费计算资源，则可以通过MaxCompute的元数据服务提供的相关视图对已经使用的计算资源进行查看。例如，TASKS_HISTORY可以查看详细的审计作业执行情况，包括时间、Job内容、资源消耗等信息，详情请参见TASKS_HISTORY。

  > 说明 元数据服务提供的视图只保留最近15天的数据。如果需要更长时间的数据，建议您定期自行读取数据并保存。

7.2. 基于Policy对具备内置角色的用户进行权限管理

用户被赋予MaxCompute内置的角色后，会具备内置角色相应的权限，例如用户被赋予开发角色则具备表、资源等的操作权限。但实际业务场景中，需要对此类用户的操作权限进行更精细化的管理，例如不允许删除重要表。本文基于案例为您介绍如何通过Policy对具备内置角色权限的用户进行权限管理。

前提条件

已安装MaxCompute客户端。更多安装MaxCompute客户端操作，请参见安装并配置MaxCompute客户端。

背景信息

当用户已经被赋予内置角色，且需要对用户的权限进行更精细化管理时，推荐您使用Policy权限管控机制，来解决ACL权限管控机制无法精细化管理此类权限的问题。

Policy权限管控机制是一种基于角色的权限管理方式，允许或禁止角色操作（例如读、写）项目中的对象（例如表），将角色赋予用户后，即可管控用户权限。更多Policy授权或撤销授权语法信息，请参见Policy权限管控。

Policy授权

假设RAM用户Alice已具备MaxCompute项目的开发角色，现需要禁止Alice删除以tb_开头的所有表。Alice账号名为Alice，所属阿里云账号为Bob@aliyun.com。

Policy授权操作需要由项目所有者、具备Super_Administrator或Admin角色的用户执行。Policy授权操作流程如下：

1. 登录MaxCompute客户端。
2. 执行 `create role` 命令创建角色delete_test。

   命令示例如下。

   ```
   create role delete_test;
   ```

   更多创建角色信息，请参见角色规划与管理。
3. 基于Policy权限管控机制，执行 `grant` 命令为角色delete_test授权，禁止删除以tb_开头的所有表。

   命令示例如下。

   ```
   grant drop on table tb_* to role delete_test privilegeproperties("policy" = "true", "allow"="false");
   ```

   更多Policy授权语法格式信息，请参见Policy授权（Grant）。
4. 执行 `grant` 命令将delete_test角色赋予Alice。

   命令示例如下。

   ```
   grant delete_test to ram$bob@aliyun.com:Alice;
   ```

   如果不清楚账号名称，可以在MaxCompute客户端执行 `list users;` 命令获取。更多将角色赋予用户信息，请参见角色规划与管理。
5. 执行 `show grants` 命令查看Alice的权限信息。

   命令示例如下。

   ```
   show grants for ram$bob@aliyun.com:Alice;
   ```

   返回结果如下。

```
[roles]
role_project_admin, delete_test                                  --Alice已被赋予delete_test角色。
Authorization Type: Policy                                       --授权方式为Policy。
[role/delete_test]
D       projects/mcproject_name/tables/tb_*: Drop                --不允许（D表示Deny）删除项目中以tb_开头的所有表。
[role/role_project_admin]
A       projects/mcproject_name: *
A       projects/mcproject_name/instances/*: *
A       projects/mcproject_name/jobs/*: *
A       projects/mcproject_name/offlinemodels/*: *
A       projects/mcproject_name/packages/*: *
A       projects/mcproject_name/registration/functions/*: *
A       projects/mcproject_name/resources/*: *
A       projects/mcproject_name/tables/*: *
A       projects/mcproject_name/volumes/*: *
Authorization Type: ObjectCreator
AG      projects/mcproject_name/tables/local_test: All
AG      projects/mcproject_name/tables/mr_multiinout_out1: All
AG      projects/mcproject_name/tables/mr_multiinout_out2: All
AG      projects/mcproject_name/tables/ramtest: All
AG      projects/mcproject_name/tables/wc_in: All
AG      projects/mcproject_name/tables/wc_in1: All
AG      projects/mcproject_name/tables/wc_in2: All
AG      projects/mcproject_name/tables/wc_out: All
```

更多查看用户权限信息，请参见通过MaxCompute SQL查询权限信息。

6. 以Alice身份登录MaxCompute客户端，执行 `drop table` 命令尝试删除以tb_开头的表。

   命令示例如下。

   ```
   drop table tb_test;
   ```

   返回结果如下。表明权限控制生效。如果删除成功，表明权限控制不生效，请仔细确认是否已经正确执行上述步骤。

   ```
   FAILED: Catalog Service Failed, ErrorCode: 50, Error Message: ODPS-0130013:Authorization exception - Authorization Failed [4
   011],
   You have NO privilege 'odps:Drop' on {acs:odps:*:projects/mcproject_name/tables/tb_test}.
   Explicitly denied by policy.
   Context ID:85efa8e9-40da-4660-bbfd-b503dfa64c0a. --->Tips: Pricipal:RAM$bob@aliyun.com:Alice; Deny by policy
   ```

撤销Policy授权

基于Policy授权中的案例，假设以tb_开头的所有表在实际业务中已不再需要，允许Alice执行删除操作。

撤销Policy授权操作需要由项目所有者、具备Super_Administrator或Admin角色的用户执行。撤销Policy授权方式如下：

- **撤销为角色授予的权限，保留角色**

  操作流程如下：

  i. 登录MaxCompute客户端。

  ii. 基于Policy权限管控机制，执行 `revoke` 命令撤销Policy授权，允许delete_test角色删除以tb_开头的所有表。

  命令示例如下。

  ```
  revoke drop on table tb_* from role delete_test privilegeproperties("policy" = "true", "allow"="false");
  ```

  更多撤销Policy授权语法格式信息，请参见撤销Policy授权（Revoke）。

  iii. 执行 `show grants` 命令查看Alice的权限信息。命令示例如下。

  ```
  show grants for ram$bob@aliyun.com:Alice;
  ```

  返回结果如下。

```
[roles]
role_project_admin, delete_test                                    --保留delete_test角色。
Authorization Type: Policy                                         --Policy授权信息已撤销。
[role/role_project_admin]
A       projects/mcproject_name: *
A       projects/mcproject_name/instances/*: *
A       projects/mcproject_name/jobs/*: *
A       projects/mcproject_name/offlinemodels/*: *
A       projects/mcproject_name/packages/*: *
A       projects/mcproject_name/registration/functions/*: *
A       projects/mcproject_name/resources/*: *
A       projects/mcproject_name/tables/*: *
A       projects/mcproject_name/volumes/*: *
Authorization Type: ObjectCreator
AG      projects/mcproject_name/tables/local_test: All
AG      projects/mcproject_name/tables/mr_multiinout_out1: All
AG      projects/mcproject_name/tables/mr_multiinout_out2: All
AG      projects/mcproject_name/tables/ramtest: All
AG      projects/mcproject_name/tables/tb_test: All
AG      projects/mcproject_name/tables/wc_in: All
AG      projects/mcproject_name/tables/wc_in1: All
AG      projects/mcproject_name/tables/wc_in2: All
AG      projects/mcproject_name/tables/wc_out: All
```

更多查看用户权限信息，请参见通过MaxCompute SQL查询权限信息。

iv. 以Alice身份登录MaxCompute客户端，执行 `drop table` 命令尝试删除以tb_开头的表。

命令示例如下。

```
drop table tb_test;
```

返回OK，表明撤销Policy授权成功。

- **收回赋予用户的角色，如果不再需要角色，可删除角色**

操作流程如下：

i. 登录MaxCompute客户端。

ii. 执行 `revoke` 命令收回授予Alice的delete_test角色。

命令示例如下。

```
revoke delete_test from ram$bob@aliyun.com:Alice;
```

更多收回赋予用户的角色信息，请参见角色规划与管理。

iii. 执行 `show grants` 命令查看Alice的权限信息。命令示例如下。

```
show grants for ram$bob@aliyun.com:Alice;
```

返回结果如下。

```
[roles]
role_project_admin                                         --已删除delete_test角色。
Authorization Type: Policy
[role/role_project_admin]
A       projects/mcproject_name: *
A       projects/mcproject_name/instances/*: *
A       projects/mcproject_name/jobs/*: *
A       projects/mcproject_name/offlinemodels/*: *
A       projects/mcproject_name/packages/*: *
A       projects/mcproject_name/registration/functions/*: *
A       projects/mcproject_name/resources/*: *
A       projects/mcproject_name/tables/*: *
A       projects/mcproject_name/volumes/*: *
Authorization Type: ObjectCreator
AG      projects/mcproject_name/tables/local_test: All
AG      projects/mcproject_name/tables/mr_multiinout_out1: All
AG      projects/mcproject_name/tables/mr_multiinout_out2: All
AG      projects/mcproject_name/tables/ramtest: All
AG      projects/mcproject_name/tables/wc_in: All
AG      projects/mcproject_name/tables/wc_in1: All
AG      projects/mcproject_name/tables/wc_in2: All
AG      projects/mcproject_name/tables/wc_out: All
```

iv. 以Alice身份登录MaxCompute客户端，执行 `drop table` 命令尝试删除以tb_开头的表。

命令示例如下。

```
drop table tb_test;
```

返回OK，表明撤销Policy授权成功。

v. （可选）执行 `drop role` 命令删除delete_test角色。

命令示例如下。

```
drop role delete_test;
```

返回OK，表明角色删除成功。更多删除角色信息，请参见角色规划与管理。

8.资源管理

8.1. 资源规划及规格选型

MaxCompute资源包含存储资源和计算资源（CU）两种，存储资源用于存储表或资源（Resource），计算资源用于运行作业。为实现以最低的费用，满足业务对存储资源、计算资源的需求，企业必须要根据自身情况合理规划资源。本文为您介绍如何规划存储、计算资源，并选择合适的产品规格，仅供参考。

背景信息

最佳资源规划方案可以达到如下目的：

- 存储资源

  资源充足，既能够存储当前的所有存量数据，也能够存储未来一段时间的增量数据。

- 计算资源
  - 资源充足，且无浪费，能够满足所有计算作业的资源需求。
  - 不同优先级的作业可以互不干扰，优先保证高优先级的作业获取到足够的计算资源。
  - 当某些作业需要处理庞大的数据量且耗费计算资源较多时，可以同时确保其他作业能获取到计算资源且不会阻塞运行。
  - 能够满足不同时段的差异化资源需求，根据不同项目的资源使用情况按照时段划分资源，提高资源利用率。

存储资源规划及规格选型

- **存储资源规划频率**

  建议每半年评估一次计存比、存储容量。当您重新评估后发现需要变更规格或转换计费方式时，请参见升级和降配或转换计费方式。

  > **说明** 计存比是计算资源和实际存储数量（单位为TB）的比值。即计存比=计算资源/实际存储数量。例如项目的计算资源为50 CU，实际存储数据量为10 TB，则计存比为50 CU/10 TB=5。

- **规格选型建议**

  基于MaxCompute的产品规格情况，推荐您按照如下情形选择规格：
  - 当预估企业未来一段时间内的数据存储量较大（100 TB以上），计算作业少（计存比小于1.5）时，推荐选择包年包月套餐版。
  - 当预估企业未来一段时间内的数据存储量很小（500 GB以下），计算作业仅涉及SQL、PyODPS类型时，推荐选择按量计费开发者版。
  - 其他情况下，企业可以使用灵活的存储资源：
    - 如果预估项目使用的计算资源数量稳定，推荐选择包年包月标准版。
    - 如果希望计算资源数量不受限制（资源需求不稳定），推荐选择按量计费标准版。

  > **说明** 更多预估计算资源信息，请参见计算资源规划及规格选型。

- **存储资源规划依据**

  您可以根据计存比、存储容量选择产品规格。MaxCompute各规格的计存比和存储容量情况如下：
  - 包年包月套餐版：计存比是固定的，计存比为1左右。
  - 包年包月标准版：计存比不固定。存储资源池为共享型，按需存储。
  - 按量计费标准版：计存比不固定，存储资源池为共享型，按需存储。
  - 按量计费开发者版：计存比不固定。存储容量限制为500 GB。

- **存储资源计费方式**

  对于存储资源，MaxCompute提供两种计费方式：
  - 按量计费：该计费方式适用于灵活存储项目。存储资源使用灵活，仅根据存储量计算费用。计算资源不受存储限制，可单独进行规划。
  - 包年包月套餐：该计费方式适用于存储量大、计算作业较少的项目。同时包年包月套餐的计算资源数量是固定的，无法应对计算资源需求量猛增的情况。例如日常的数据批量处理作业可以正常运行，但在大型活动期间的数据批量处理作业会出现严重阻塞。

计算资源规划及规格选型

- **计算资源规划频率**

  建议每半年评估一次计存比及存储容量，并调整计算资源数量。当您重新评估后发现需要变更规格或转换计费方式时，请参见升级和降配或转换计费方式。

- **规格选型建议**
  - 如果您的项目处于开发测试阶段，推荐选择按量计费标准版。
  - 如果您的项目已完成开发，正式进入上线阶段，推荐选择包年包月标准版，购买包年包月模式的计算资源。需要购买的计算资源数量需要根据按照计存比规划计算资源或按照项目实际消耗的计算资源数量规划计算资源进行预估。

- **计算资源规划依据**

- 方式一：按照计存比规划计算资源

  先预估数据存储量及计存比，然后再预估最低需要购买的计算资源数量。

- 方式二：按照项目实际消耗的计算资源数量规划计算资源

  在项目正式上线前或在项目正式上线运行一小段时间之后，评估计算资源消耗的CPU总和，然后再根据每天所有作业必须在哪个时间段运行完成，计算消耗费用最少的最佳计算资源数量。

- **计算资源计费方式**

  对于计算资源，MaxCompute提供两种计费方式：

  - 按量计费：该计费方式适用于处于测试阶段的项目。该阶段消耗的计算资源数量不多，采用按量计费方式成本更低。
  - 包年包月：该计费方式适用于已完成开发进入上线阶段的项目，包年包月模式下购买的预留计算资源数量是固定的，不会在公共资源池抢占计算资源，能保障作业的顺利运行。同时有预留和非预留计算资源时，会优先使用预留计算资源。

按照计存比规划计算资源

按照计存比规划计算资源的流程如下：

1. 预估存储容量。

   存储容量=当前数据存储量+每月预估数据增量×月数。

   在项目上线之后就可以得到当前数据存储量。每月预估数据增量需要在项目上线两到三个月后，根据每月增量之和除以月数得到。如果还要考虑未来数据中台会承载更多业务、每月数据增量会变大等因素，可以将当前计算得到的每月预估数据增量值乘以倍数。

2. 预估计存比。

   按照项目开发测试阶段、以及上线运行一两个月的情况，预估计存比。

   计存比一般规划为2~10，后续根据规划的计存比及预估存储容量购买计算资源：

   - 如果项目每天运行的批量处理作业很多，且SQL程序计算复杂度高，计存比推荐规划为10。
   - 如果项目每天运行的数据批量处理作业比较少，且SQL程序计算复杂度不高，计存比推荐规划为2。
   - 如果项目每天运行的数据批量处理作业适中，SQL程序计算复杂度也适中，计存比可以规划为2~10之间的值。

3. 预估计算资源数量。

   根据预估的存储容量、计存比值预估计算资源数量。计算资源数量=预估存储容量×预估计存比。

   > **说明** 按照计存比预估项目会消耗的计算资源数量时，存储容量、计存比都属于预估项，不够精准。因此，该方法要求项目的技术负责人需要拥有较多的项目实施经验，能够在每一步预估都尽可能保证准确。

按照项目实际消耗的计算资源数量规划计算资源

按照项目实际消耗的计算资源数量规划计算资源的流程如下：

1. 查看项目的CPU消耗量。

   Information Schema提供的TASKS_HISTORY视图记录了MaxCompute项目中所有计算作业消耗的计算资源情况。您可以通过MaxCompute客户端查询TASKS_HISTORY视图中的cost_cpu字段，基于该字段进一步计算消耗的资源数量。查询TASKS_HISTORY视图的命令示例如下：

   ```
   select * from information_schema.tasks_history where ...;
   ```

   cost_cpu代表MaxCompute计算作业的CPU消耗量。例如10 core运行5s，cost_cpu为10（CPU核数消耗量）×100（1 core×s）×5（任务运行时间，单位为秒）=5000。

2. 规划计算资源数量。

   i. 计算账户下所有项目平均每天运行所有作业消耗的cost_cpu总和。

   假设某个MaxCompute项目平均每天会运行1000个作业，这些作业消耗的cost_cpu分别是W1~W1000。计算W1~W1000的和即可得到平均每天运行所有作业消耗的cost_cpu总和，假设总和为Wz。

   > **说明** 通常，数据中台一般会划分如下6个MaxCompute项目：
   >
   > - ods_dev：贴源层开发测试项目
   > - ods_prod：贴源层生产项目
   > - cdm_dev：公共层开发测试项目
   > - cdm_prod：公共层生产项目
   > - ads_dev：应用层开发测试项目
   > - ads_prod：应用层生产项目
   >
   > 该情况下，您需要计算上述6个项目的所有计算作业的cost_cpu总和。

   ii. 基于计算得到的cost_cpu总和进行换算。

   因为cost_cpu按照秒统计，对于实际项目评估过于精细，通常将cost_cpu除以100、然后再除以3600，得到`core×h`（CPU核数消耗量×小时）。这样方便评估实际项目在规定时间段内运行完所有作业需要的最少计算资源数量。假设该值为W。

iii. 确认作业需要在每天的哪些时间段完成运行。

例如，客户要求在00:00:00~06:00:00间所有数据批量处理作业必须完成运行。即每天能够运行的总时长是6个小时。本文假设所有作业必须在N个小时内完成运行。

iv. 预估需要购买的计算资源最小值。

基于前两步得到的值（W和N）进行运算：W（CPU核数消耗量×小时）/N（作业运行时长N个小时），即可得到MaxCompute项目需要购买的计算资源数量的最小值。

计算W/N的前提是数据处理作业的cost_cpu很稳定，而且在这N个小时内，所有作业都随时在运行，不存在任何空闲的计算资源。但是，实际项目可能会因为某些原因导致作业运行时间延长（例如参与计算的数据量增加），相当于W会变大。同时，由于通过DataWorks或Dataphin运行调度作业还会产生很多延迟时间、作业获取计算资源也会耽误很多时间，这部分延迟时间会加大作业之间运行的时间间隔，真正用于运行作业的时间会小于N。W/N的分母实际变大、分子实际变小，进而变相地要求增加计算资源，以便让作业获取更多资源进而运行更加快速。因此，建议您在W/N结果基础上增加一倍。

购买计算资源后，您可以通过MaxCompute管家设置配额组，进一步对计算资源进行合理规划，充分利用计算资源。相关资源规划实践案例，请参见包年包月资源分时配额或包年包月资源隔离。

参考文档

如需了解更多延伸信息，请参见资源规划管理及评估。

8.2. 包年包月资源分时配额

本文为您介绍如何对包年包月计算资源按照时间段设置资源配额，协助您更快上手使用分时配额功能，提高计算资源使用率。

背景信息

在数据开发过程中，不同项目（开发、生产或分析）使用计算资源的时段不同。例如，数据仓库团队有生产、开发和业务分析项目，夜间是生产项目的计算高峰期；白天是开发和业务分析项目的计算高峰期。即不同项目在不同的时段对计算资源的需求量不同，且日常峰谷时间段比较固定。

MaxCompute管家之前提供的自定义配额组，只能对计算资源进行切分，但无法按照时段进行切分。自2020年7月23日开始，各个区域的MaxCompute管家陆续升级至新版本，新版本支持分时配额功能，可以对预留计算资源（预留CU）按时段切分，满足不同时段的差异化资源需求，并对计算资源进行隔离（生产、开发或分析），避免项目之间相互干扰，同时更大化提高计算资源使用率。

使用限制

- 仅项目空间Owner可以设置分时配额功能。
- 仅包年包月预留计算资源支持设置分时配额，非预留计算资源不支持设置分时配额。

使用说明

分时配额功能使用注意点如下。

操作类型	注意事项
开启分时配额	• 开启分时配额功能后，当前区域下的所有配额组都开启分时配额功能，但默认配额组不支持设置分时配额。 • 开启分时配额功能后，默认只有一个分时时间段（00:00:00~23:59:59）。最多支持设置3个分时时间段。所有配额组都支持设置同一个分时时间段，即所有配额组的分时时间段相同。 • 只支持设置整点分时时间段，例如00:00:00~07:00:00，最后一个分时时间段截止时刻必须为23:59:59。 • 如果某个配额组不需要设置分时时间段，设置每个分时时间段的预留最小或最大CU数量一样即可。 • 同一个分时时间段，所有配额组的预留CU最小值总和等于购买的CU数量。
关闭分时配额	如果设置的分时时间段不合理或需要调整，您可以单击**设置分时**。在**设置分时**对话框，单击**关闭分时**。配额组的预留CU量为关闭前的时间段所设置的CU量。详情请参见设置配额组 > 设置分时。
修改分时配额	不支持直接在已设置的分时上修改分时时间段，您需要先**关闭分时**，再**开启分时**，配置新分时时间段，并修改各个自定义配额组中不同时间段的预留CU量。

使用案例

场景

例如，数据仓库团队在华北2（北京）区域购买了包年包月预留计算资源100 CU，该团队有生产、开发和分析项目。

分析各项目资源使用高峰

- 生产项目：计算高峰时间段为00:00:00~08:00:00，计算资源使用量较高，其余时间会有小部分小时任务且时间要求不高。
- 开发项目：计算高峰时间段为09:00:00~23:00:00，计算资源使用量较高。
- 分析项目：计算高峰时间段为09:00:00~23:00:00，计算比较分散，但也有些固定周期调度任务需要保障。

上述时间段分布比较固定，因此可以将配额组切分为2个时段00:00:00~08:00:00和08:00:00~23:59:59。生产、开发、分析项目对计算资源的需求量也各有差异，因此可以自定义2个配额组，外加默认配额组共3个配额组对计算资源进行分配。

设计配额组

- 分时时间段1：00:00:00~08:00:00

- 分时时间段2：08:00:00~23:59:59

配额组名称	分时时间段1预留CU Min值	分时时间段1预留CU Max值	分时时间段2预留CU Min值	分时时间段2预留CU Max值	备注
默认配额组	80	100	20	100	生产项目配额。
开发配额组	10	10	30	50	分时时间段1预留一些计算资源用于临时开发测试场景。若计算资源空闲，当默认配额组下的资源需求量超过80 CU时也可以使用。
分析配额组	10	10	50	100	分时时间段1预留一些计算资源用于临时获取数据场景。若计算资源空闲，当默认配额组下的资源需求量超过80 CU时也可以使用。

> 说明 默认配额组不支持自定义编辑，预留CU值为扣除自定义配额组设置的CU数量后剩余的CU量。

设置步骤

1. 进入MaxCompute管家页面。
   i. 登录MaxCompute控制台，在左上角选择地域。

   ii. 单击**管家**页签，即可进入MaxCompute管家页面。
2. 在左侧导航栏，单击**配额**。在**配额**页面，单击右上角**设置分时**。
3. 在**设置分时**对话框，按照需要增加时间段，单击**开启分时**。
4. 在**配额**页面，单击右上角**新建配额组**新建开发和分析配额组。单击默认配额组右侧的**修改**修改配额组，对每个配额组预留CU不同时段的Min和Max值进行设置。例如，分析项目的配额组设置如下。

> 说明 非预留CU最大配额即包年包月里的非预留计算资源，详情请参见包年包月非预留计算资源。

5. 在左侧导航栏，单击**项目**。在**项目**页面的**包年包月项目**页签，单击生产、开发或分析项目右侧的**修改**。
6. 在**修改配额组信息**面板，从**配额组**下拉列表，选择相应的配额组，单击**执行**，即可完成配置。

8.3. 包年包月资源隔离

对于使用MaxCompute的企业，通常会根据业务的差异性来创建多个MaxCompute项目进行数据隔离。由于每个MaxCompute项目对计算资源（CU）的需求不一致，需要对CU按照项目进行隔离，以便实现CU使用率最大化。本文为您介绍如何通过MaxCompute管家隔离MaxCompute包年包月CU。

背景信息

您可以通过MaxCompute管家的配额组功能对购买的包年包月CU进行隔离。MaxCompute管家支持的配额组如下：

- 默认预付费Quota：购买包年包月计算资源后，系统默认创建的配额组。该配额组不支持修改。升级或降配时，对应的CU量都在该配额组中进行增减。
- 自定义配额组：在自定义的配额组中，**预留CU最小配额**和**非预留CU最大配额**的值将从默认预付费Quota配额组对应的值里进行扣减。**预留CU最大配额**可以设置为购买的预留CU量。其中：
  - **预留CU最小配额**：为最小保障CU量。
  - **预留CU最大配额**：可使用的最大CU量。当有多个配额组且配置了**预留CU最小配额<预留CU最大配额**时，一旦有配额组的CU为空闲状态时，其他配额组可以占用空闲CU。该实现会导致所有配额组在当前账号当前区域内都是共享资源组。

所有配额组的**预留CU最小配额**相加等于购买的预留CU量；所有配额组的**非预留CU最大配额**相加等于购买的非预留CU量。

使用案例

场景

数据仓库团队使用MaxCompute进行开发、分析、挖掘的业务大致为：数据仓库开发和生产、运营分析需求、算法挖掘。基于不同的业务创建不同的MaxCompute项目：

- 数据仓库开发和生产：按照数据仓库模型分层划分MaxCompute项目，分为数据仓库开发项目和数据仓库生产项目。
- 运营分析需求：根据业务需求创建不同部门专用的MaxCompute项目，获取日常数据并进行分析。
- 算法挖掘：根据作业周期特点划分MaxCompute项目，分为算法开发项目和算法生产项目。

根据前期业务评估结果，购买的预留CU量为1000 CU，非预留CU量为600 CU。现在需要将这些CU合理地进行隔离分配，以便能最大化提升CU使用率。

划分CU

划分CU时，您可以关注如下内容：

- 高保障MaxCompute项目主要配置预留CU，非预留CU可作为加持资源。
- **预留CU最小配额**要根据实际配置避免滥用。
- 对于非高保障，优先级也不高但是可能请求大量CU的MaxCompute项目，建议控制对应配额组的**预留CU最大配额**，以免影响其他配额组。
- 对于平均占用CU时间较长的MaxCompute项目可以设置独立配额组，同时控制对应配额组的**预留CU最大配额**。
- 对时效性要求不高，CU占用频率高的MaxCompute项目可以考虑设置**非预留CU最大配额**。
- 您可以根据实际情况结合分时功能实现CU利用最大化。更多分时信息，请参见设置配额组 > 设置分时。
- 默认预付费Quota配额组不支持修改CU配额，如果您不希望出现某些项目发起的作业可能会占用所有CU量的情况，可以考虑默认预付费Quota配额组不关联项目。由于默认预付费Quota配额组的**预留CU最小配额**不能为0，可以保留1 CU，然后设置其他配额组中的**预留CU最小配额<预留CU最大配额**，其他配额组依然能占用这1 CU。

设计配额组

配额组	特点	预留CU最小配额	预留CU最大配额	非预留CU最大配额	关联项目	分时设置
数据仓库生产配额组	高保障，有较为突出的CU请求波峰时段，小时作业CU请求较平稳。	• 时段1：600 • 时段2：300	• 时段1：700 • 时段2：700	0	数据仓库生产项目	• 时段1：00:00:00~09:00:00 • 时段2：09:00:00~23:59:59
数据仓库开发配额组	高保障，有较为突出的CU请求波谷时段。	• 时段1：20 • 时段2：100	• 时段1：100 • 时段2：100	0	数据仓库开发项目	• 时段1：00:00:00~09:00:00 • 时段2：09:00:00~23:59:59
算法生产配额组	高保障，部分作业的CPU占用量较高且运行时间较长。	300	300	400	算法生产项目	00:00:00~23:59:59
算法开发配额组	高保障，部分作业的CPU占用量较高且运行时间较长。	50	50	0	算法开发项目	00:00:00~23:59:59

配额组	特点	预留CU最小配额	预留CU最大配额	非预留CU最大配额	关联项目	分时设置
运营分析配额组	使用的人员多，需要一定的CU保障，有较为突出的CU请求波谷时段，会有一些大作业长时间占用资源。	• 时段1：30 • 时段2：250	• 时段1：300 • 时段2：250	200	运营分析项目	• 时段1：00:00:00~09:00:00 • 时段2：09:00:00~23:59:59

考虑到上述业务的特点，对应配额组的**预留CU最大配额**都进行了限制，避免严重影响其他配额组的最低保障值。在MaxCompute管家上设计配额组时，默认预付费Quota配额组不关联MaxCompute项目，但**预留CU最小配额**又必须大于0，可以保留1 CU，将数据仓库开发项目的**预留CU最小配额**减1。

设置步骤

1. 设置分时时段，增加00:00:00~09:00:00、09:00:00~23:59:59两个时段，操作详情请参见设置配额组 > 设置分时。
2. 基于配额组规划，新增配额组，设置数据仓库生产配额组、数据仓库开发配额组、算法生产配额组、算法开发配额组、运营分析配额组。操作详情请参见设置配额组 > 新建配额组。
3. 将项目关联对应的配额组，默认预付费Quota配额组不关联项目。操作详情请参见修改项目配额组。

参考文档

随着业务的不断变化，配额组的划分也需要随之改变，所以您有必要随时监控配额组的使用情况，以便及时对配额组进行调整。您可以通过阿里云监控服务的**MaxCompute-包年包月Quota组资源**指标进行监控，详情请参见监控报警。

您还可以结合MaxCompute更多的资源管理功能实现更精细的资源管理，例如包年包月项目使用按量计费资源和包年包月项目作业优先级。

包年包月资源分时配额最佳实践请参见包年包月资源分时配额。