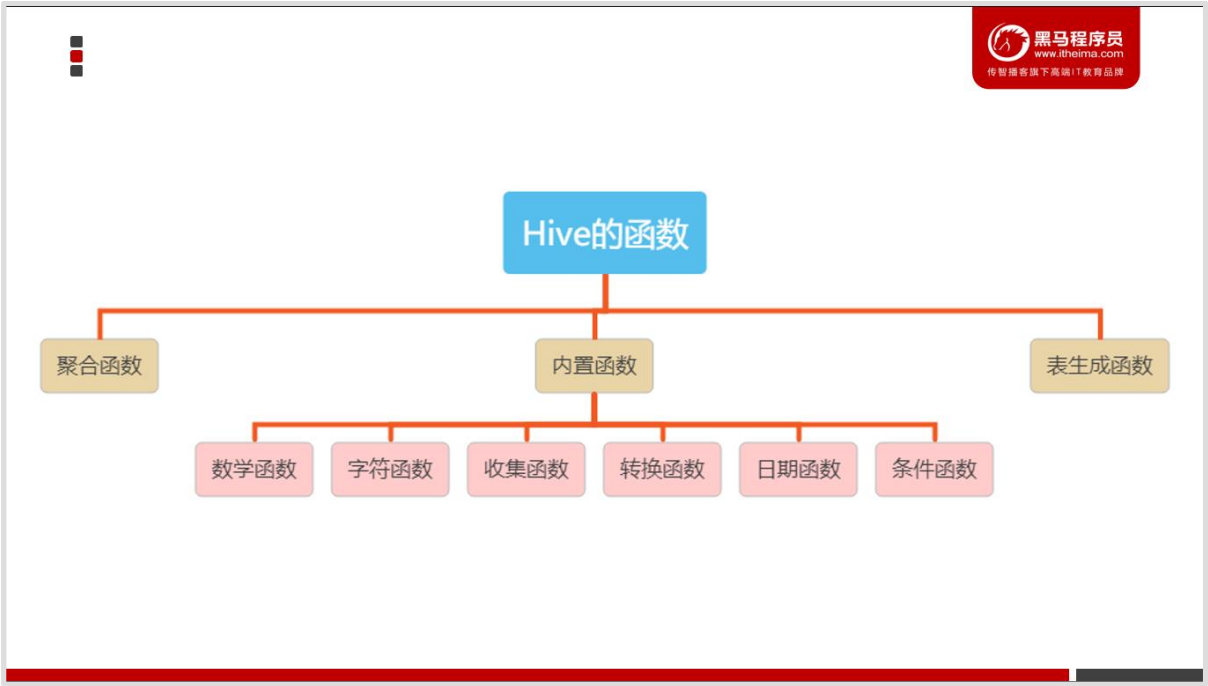
1. Hive函数

Hive的函数分为三类： 聚合函数、内置函数，表生成函数，聚合函数之前已经学习过了，接下来学习内置函数和表生成函数.

1.1. Hive的内置函数



1.1.1. 数学函数

1. 1. 1. 1. **取整函数**: round

语法: round(double a)

返回值: BIGINT

说明:返回double类型的整数值部分（遵循四舍五入）

举例：

```
hive> select round(3.1415926);
3
```

1.1.1.2. **指定精度取整函数**: round

语法: round(double a, int d)

返回值: DOUBLE

说明:返回指定精度d的double类型

举例：

```
hive> select round(3.1415926,4);
3.1416
```

1.1.1.3. **向下取整函数**: floor

语法: floor(double a)

返回值: BIGINT

说明:返回等于或者小于该double变量的最大的整数

举例：

```
hive> select floor(3.1415926);
3
```

1.1.1.4. **向上取整函数**: ceil

语法: ceil(double a)

返回值: BIGINT

说明:返回等于或者大于该double变量的最小的整数

举例：

```
hive> select ceil(3.1415926);
4
```

1.1.1.5. **取随机数函数**: rand

语法: rand(),rand(int seed)

返回值: double

说明:返回一个0到1范围内的随机数。如果指定种子seed，则会返回固定的随机数

举例：

```
hive> select rand();
0.5577432776034763

hive> select rand();
0.6638336467363424

hive> select rand(100);
0.7220096548596434

hive> select rand(100);
0.7220096548596434
```

1.1.1.6. 幂运算函数: pow

语法: pow(double a, double p)

返回值: double

说明:返回a的p次幂

举例：

```
hive> select pow(2,4) ;
16.0
```

1.1.1.7. 绝对值函数: abs

语法: abs(double a) abs(int a)

返回值: double int

说明:返回数值a的绝对值

举例：

```
hive> select abs(-3.9);
3.9
hive> select abs(10.9);
10.9
```

1.1.2. 字符串函数

1.1.2.1. 字符串长度函数：length

语法: length(string A)

返回值: int

说明：返回字符串A的长度

举例：

```
hive> select length('abcedfg')l;
7
```

1.1.2.2. 字符串反转函数：reverse

语法: reverse(string A)

返回值: string

说明：返回字符串A的反转结果

举例：

```
hive> select reverse(abcedfg'  );
gfdecba
```

1.1.2.3. **字符串连接函数：**concat

语法: concat(string A, string B…)

返回值: string

说明：返回输入字符串连接后的结果，支持任意个输入字符串

举例：

```
hive> select concat( 'abc' ,'def' ,'gh' );;
abcdefgh
```

1.1.2.4. **字符串连接函数-带分隔符：**concat_ws

语法: concat_ws(string SEP, string A, string B…)

返回值: string

说明：返回输入字符串连接后的结果，SEP表示各个字符串间的分隔符

举例：

```
hive> select concat_ws(',','abc','def','gh');
abc,def,gh
```

1.1.2.5. **字符串截取函数：**substr,substring

语法: substr(string A, int start),substring(string A, int start)

返回值: string

说明：返回字符串A从start位置到结尾的字符串

举例：

```
hive> select substr('abcde',3);
cde
hive> select substring('abcde',3);
cde
hive>select substr('abcde',-1);
e
```

1.1.2.6. **字符串截取函数：**substr,substring

语法: substr(string A, int start, int len),substring(string A, intstart, int len)

返回值: string

说明：返回字符串A从start位置开始，长度为len的字符串

举例：

```
hive> select substr('abcde',3,2);
cd
hive> select substring('abcde',3,2);
cd
hive>select substring('abcde',-2,2);
de
```

1.1.2.7. 字符串转大写函数：upper,ucase

语法: upper(string A) ucase(string A)

返回值: string

说明：返回字符串A的大写格式

举例：

```
hive> select upper('abSEd');
ABSED
hive> select ucase('abSEd');
ABSED
```

1.1.2.8. 字符串转小写函数：lower,lcase

语法: lower(string A) lcase(string A)

返回值: string

说明：返回字符串A的小写格式

举例：

```
hive> select lower('abSEd');
absed
hive> select lcase('abSEd');
absed
```

1.1.2.9. 去空格函数：trim

语法: trim(string A)

返回值: string

说明:去除字符串两边的空格

举例：

```
hive> select trim(' abc ');
```

```
abc
```

1.1.2.10. **左边去空格函数：**ltrim

语法: ltrim(string A)

返回值: string

说明：去除字符串左边的空格

举例：

```
hive> select ltrim(' abc ');
abc
```

1.1.2.11. **右边去空格函数：**rtrim

语法: rtrim(string A)

返回值: string

说明：去除字符串右边的空格

举例：

```
hive> select rtrim(' abc ');
abc
```

1.1.2.12. **正则表达式替换函数：**regexp_replace

语法: regexp_replace(string A, string B, string C)

返回值: string

说明：将字符串A中的符合java正则表达式B的部分替换为C。注意，在有些情况下要使用转义字符,类似oracle中的regexp_replace函数。

举例：

```
hive> select regexp_replace('foobar', 'oo|ar', '');
fb
```

1.1.2.13. URL**解析函数：**parse_url

语法: parse_url(string urlString, string partToExtract [, stringkeyToExtract])

返回值: string

说明：返回URL中指定的部分。partToExtract的有效值为：HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, and USERINFO.

举例：

```
hive> select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1',
'HOST');
facebook.com

hive> select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1',
'PATH');
/path1/p.php

hive> select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1',
'QUERY','k1');
v1
```

1.1.2.14. **分割字符串函数**: split

语法: split(string str, stringpat)

返回值: array

说明:按照pat字符串分割str，会返回分割后的字符串数组

举例：

```
hive> select split('abtcdtef','t');
["ab","cd","ef"]
```

1.1.3. 日期函数

1.1.3.1. **获取当前**UNIX**时间戳函数**:unix_timestamp

语法: unix_timestamp()

返回值: bigint

说明:获得当前时区的UNIX时间戳

举例：

```
hive> select unix_timestamp();
1323309615
```

1.1.3.2. UNIX**时间戳转日期函数**:from_unixtime

语法: from_unixtime(bigint unixtime[, string format])

返回值: string

说明:转化UNIX时间戳（从1970-01-01 00:00:00 UTC到指定时间的秒数）到当前时区的时间格式

举例：

```
hive> select from_unixtime(1323308943,'yyyyMMdd');
20111208
```

1.1.3.3. **日期转**UNIX**时间戳函数**:unix_timestamp

语法: unix_timestamp(string date)

返回值: bigint

说明:转换格式为"yyyy-MM-ddHH:mm:ss"的日期到UNIX时间戳。如果转化失败，则返回0。

举例：

```
hive> select unix_timestamp('2011-12-07 13:01:03');
1323234063
```

1.1.3.4. **指定格式日期转**UNIX**时间戳函数**:unix_timestamp

语法: unix_timestamp(string date, string pattern)

返回值: bigint

说明:转换pattern格式的日期到UNIX时间戳。如果转化失败，则返回0。

举例：

```
hive> select unix_timestamp('20111207 13:01:03','yyyyMMddHH:mm:ss');
1323234063
```

1.1.3.5. **日期时间转日期函数**:to_date

语法: to_date(string timestamp)

返回值: string

说明:返回日期时间字段中的日期部分。

举例：

```
hive> select to_date('2011-12-08 10:03:01');
2011-12-08
```

1.1.3.6. **日期转年函数**: year

语法: year(string date)

返回值: int

说明:返回日期中的年。

举例：

```
hive> select year('2011-12-08 10:03:01');
2011
hive> select year('2012-12-08');
2012
```

1.1.3.7. **日期转月函数**: month

语法: month (string date)

返回值: int

说明:返回日期中的月份。

举例：

```
hive> select month('2011-12-08 10:03:01');
12
hive> select month('2011-08-08');
8
```

1.1.3.8. **日期转天函数**: day

语法: day (string date)

返回值: int

说明:返回日期中的天。

举例：

```
hive> select day('2011-12-08 10:03:01');
8
hive> select day('2011-12-24');
24
```

同样的，还有 hour,minute,second函数，分别是获取小时，分钟和秒，使用方式和以上类似，这里就不再讲述。

1.1.3.9. **日期转周函数**:weekofyear

语法: weekofyear (string date)

返回值: int

说明:返回日期在当前的周数。

举例：

```
hive> select weekofyear('2011-12-08 10:03:01');
49
```

1.1.3.10. **日期比较函数**: datediff

语法: datediff(string enddate, string startdate)

返回值: int

说明:返回结束日期减去开始日期的天数。

举例：

```
hive> select datediff('2012-12-08','2012-05-09');
213
```

1.1.3.11. **日期增加函数**: date_add

语法: date_add(string startdate, int days)

返回值: string
说明:返回开始日期startdate增加days天后的日期。

举例：

```
hive> select date_add('2012-12-08',10);
2012-12-18
```

1.1.3.12. **日期减少函数**: date_sub

语法: date_sub (string startdate, int days)
返回值: string
说明:返回开始日期startdate减少days天后的日期。

举例：

```
hive> select date_sub('2012-12-08',10);
2012-11-28
```

1.1.4. 条件函数

1.1.4.1. if**函数**: if

语法: if(boolean testCondition, T valueTrue, T valueFalseOrNull)

返回值: T
说明: 当条件testCondition为TRUE时，返回valueTrue；否则返回valueFalseOrNull

举例：

```
hive> select if(1=2,100,200) ;
200
hive> select if(1=1,100,200) ;
100
```

1.1.4.2. **条件判断函数：**CASE

语法: CASE a WHEN b THEN c [WHEN d THEN e]* [ELSE f] END
返回值: T
说明：如果a等于b，那么返回c；如果a等于d，那么返回e；否则返回f

举例：

```
hive> select case 100 when 50 then 'tom' when 100 then 'mary'else 'tim' end ;
mary
hive> select case 200 when 50 then 'tom' when 100 then 'mary'else 'tim' end ;
tim
```

1.1.4.3. 条件判断函数：CASE

语法: CASE WHEN a THEN b [WHEN c THEN d]* [ELSE e] END
返回值: T
说明：如果a为TRUE,则返回b；如果c为TRUE，则返回d；否则返回e

举例：

```
hive> select case when 1=2 then 'tom' when 2=2 then 'mary' else'tim' end ;
mary
hive> select case when 1=1 then 'tom' when 2=2 then 'mary' else'tim' end ;
tom
```

1.1.5. 转换函数

hive有两个类型转换函数。

1，cast()函数。
公式：
cast(表达式 as 数据类型)

cast函数，可以将"20190607"这样类型的时间数据转化成int类型数据。
cast("20190607" as int)

select cast('2017-06-12' as date) filed;

1.1.6. Hive的行转列

1.1.6.1. 介绍

1、行转列是指多行数据转换为一个列的字段。
2、Hive行转列用到的函数：
concat(str1,str2,...) --字段或字符串拼接
concat_ws(sep, str1,str2) --以分隔符拼接每个字符串
collect_set(col) --将某字段的值进行去重汇总，产生array类型字段

1.1.6.2. 测试数据:

字段: deptno ename

```
20  SMITH
30  ALLEN
30  WARD
20  JONES
30  MARTIN
30  BLAKE
10  CLARK
20  SCOTT
10  KING
30  TURNER
20  ADAMS
30  JAMES
20  FORD
10  MILLER
```

1.1.6.3. 操作步骤

1:建表

```
create table emp(
deptno int,
ename string
) row format delimited fields terminated by '\t';
```

2:插入数据：

```
load data local inpath "/opt/data/emp.txt" into table emp;
```

3:转换

```
select deptno,concat_ws("|",collect_set(ename)) as ems from emp group by deptno;
```

行转列，COLLECT_SET(col)：函数只接受基本数据类型，它的主要作用是将某字段的值进行去重汇总，产生array类型字段。

4:结果查看

```
+---------+-----------------------------------------+--+
| deptno  |                   ems                   |
+---------+-----------------------------------------+--+
| 10      | CLARK|KING|MILLER                      |
| 20      | SMITH|JONES|SCOTT|ADAMS|FORD          |
| 30      | ALLEN|WARD|MARTIN|BLAKE|TURNER|JAMES  |
+---------+-----------------------------------------+--+
```

1.2. Hive的表生成函数

1.2.1. explode函数

explode(col)：将hive一列中复杂的array或者map结构拆分成多行。

explode(ARRAY)　列表中的每个元素生成一行

explode(MAP)　map中每个key-value对，生成一行，key为一列，value为一列

数据：

```
10　CLARK|KING|MILLER
20　SMITH|JONES|SCOTT|ADAMS|FORD
30　ALLEN|WARD|MARTIN|BLAKE|TURNER|JAMES
```

建表：

```
create table emp(
deptno int,
names array<string>
)
row format delimited fields terminated by '\t'
collection items terminated by '|';
```

插入数据

```
load data local inpath "/server/data/hivedatas/emp3.txt" into table emp;
```

查询数据

```
select * from emp;
```

```
0: jdbc:hive2://node3:10000> select * from emp;
+-------------+-----------------------------------------------------+--+
| emp.deptno  |                      emp.names                      |
+-------------+-----------------------------------------------------+--+
| 10          | ["CLARK","KING","MILLER"]                           |
| 20          | ["SMITH","JONES","SCOTT","ADAMS","FORD"]            |
| 30          | ["ALLEN","WARD","MARTIN","BLAKE","TURNER","JAMES"]  |
+-------------+-----------------------------------------------------+--+
3 rows selected (2.129 seconds)
```

使用explode查询

```
select explode(names) as name from emp;
```

```
0: jdbc:hive2://node3:10000> select explode(names) as name from emp;
+----------+--+
|   name   |
+----------+--+
| CLARK    |
| KING     |
| MILLER   |
| SMITH    |
| JONES    |
| SCOTT    |
| ADAMS    |
| FORD     |
| ALLEN    |
| WARD     |
| MARTIN   |
| BLAKE    |
| TURNER   |
| JAMES    |
+----------+--+
```

1.2.2. LATERAL VIEW侧视图

LATERAL VIEW

用法：LATERAL VIEW udtf(expression) tableAlias AS columnAlias

解释：用于和split, explode等UDTF一起使用，它能够将一列数据拆成多行数据，在此基础上可以对拆分后的数据进行聚合。

列转行

```
select deptno,name from emp lateral view explode(names) tmp_tb as name;
```

```
0: jdbc:hive2://node3:10000> select deptno,name from emp lateral view explode(names) tmp_tb as name;
+---------+---------+--+
| deptno  |  name   |
+---------+---------+--+
| 10      | CLARK   |
| 10      | KING    |
| 10      | MILLER  |
| 20      | SMITH   |
| 20      | JONES   |
| 20      | SCOTT   |
| 20      | ADAMS   |
| 20      | FORD    |
| 30      | ALLEN   |
| 30      | WARD    |
| 30      | MARTIN  |
| 30      | BLAKE   |
| 30      | TURNER  |
| 30      | JAMES   |
+---------+---------+--+
```

1.2.3. Reflect函数

reflect函数可以支持在sql中调用java中的自带函数

1.2.3.1. 使用java.lang.Math当中的Max求两列中最大值

```
--创建hive表
create table test_udf(col1 int,col2 int) row format delimited fields terminated by
',';

--准备数据  test_udf.txt
1,2
4,3
6,4
7,5
5,6

--加载数据

load data local inpath '/root/hivedata/test_udf.txt'    into table test_udf;

--使用java.lang.Math当中的Max求两列当中的最大值
select reflect("java.lang.Math","max",col1,col2) from test_udf;
```

1.2.3.2. 不同记录执行不同的java内置函数

```
--创建hive表
create table test_udf2(class_name string,method_name string,col1 int , col2 int)
row format delimited fields terminated by ',';

--准备数据  test_udf2.txt
java.lang.Math,min,1,2
java.lang.Math,max,2,3

--加载数据
load data local inpath '/root/hivedata/test_udf2.txt' into table test_udf2;

--执行查询
select reflect(class_name,method_name,col1,col2) from test_udf2;
```

1.3. Hive的开窗函数

1.3.1. 窗口函数(一)

NTILE,ROW_NUMBER,RANK,DENSE_RANK

1.3.1.1. 数据准备

```
cookie1,2018-04-10,1
cookie1,2018-04-11,5
cookie1,2018-04-12,7
cookie1,2018-04-13,3
cookie1,2018-04-14,2
cookie1,2018-04-15,4
cookie1,2018-04-16,4
cookie2,2018-04-10,2
cookie2,2018-04-11,3
cookie2,2018-04-12,5
cookie2,2018-04-13,6
cookie2,2018-04-14,3
cookie2,2018-04-15,9
cookie2,2018-04-16,7
```

```
CREATE TABLE itcast_t2 (
cookieid string,
createtime string,    --day
pv INT
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
stored as textfile;

-- 加载数据：
load data local inpath '/root/hivedata/itcast_t2.dat' into table itcast_t2;
```

1.3.1.2. ROW_NUMBER

ROW_NUMBER()　从1开始，按照顺序，生成分组内记录的序列

```
SELECT
  cookieid,
  createtime,
  pv,
  ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY pv desc) AS rn
  FROM itcast_t2;
```

1.3.1.3. RANK 和 DENSE_RANK

RANK() 生成数据项在分组中的排名，排名相等会在名次中留下空位

DENSE_RANK() 生成数据项在分组中的排名，排名相等会在名次中不会留下空位

```
SELECT
cookieid,
createtime,
pv,
RANK() OVER(PARTITION BY cookieid ORDER BY pv desc) AS rn1,
DENSE_RANK() OVER(PARTITION BY cookieid ORDER BY pv desc) AS rn2,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY pv DESC) AS rn3
FROM itcast_t2
WHERE cookieid = 'cookie1';
```

1.3.2. Hive分析窗口函数(2) SUM,AVG,MIN,MAX

1.3.2.1. 数据准备

```
--建表语句:
create table itcast_t1(
cookieid string,
createtime string,    --day
pv int
) row format delimited
fields terminated by ',';

--加载数据：
load data local inpath '/root/hivedata/itcast_t1.dat' into table itcast_t1;

cookie1,2018-04-10,1
cookie1,2018-04-11,5
cookie1,2018-04-12,7
cookie1,2018-04-13,3
cookie1,2018-04-14,2
cookie1,2018-04-15,4
cookie1,2018-04-16,4

--开启智能本地模式
SET hive.exec.mode.local.auto=true;
```

1.3.2.2. SUM（结果和ORDER BY相关,默认为升序）

```
select cookieid,createtime,pv,
sum(pv) over(partition by cookieid order by createtime) as pv1
from itcast_t1;

select cookieid,createtime,pv,
sum(pv) over(partition by cookieid order by createtime rows between
unbounded preceding and current row) as pv2
from itcast_t1;

select cookieid,createtime,pv,
sum(pv) over(partition by cookieid) as pv3
from itcast_t1;  --如果每天order  by排序语句  默认把分组内的所有数据进行sum操作
```

```
select cookieid,createtime,pv,
sum(pv) over(partition by cookieid order by createtime rows between 3
preceding and current row) as pv4
from itcast_t1;

select cookieid,createtime,pv,
sum(pv) over(partition by cookieid order by createtime rows between 3
preceding and 1 following) as pv5
from itcast_t1;

select cookieid,createtime,pv,
sum(pv) over(partition by cookieid order by createtime rows between current
row and unbounded following) as pv6
from itcast_t1;

--pv1: 分组内从起点到当前行的pv累积 ,如 ,11号的pv1=10号的pv+11号的pv, 12号=10
号+11号+12号
--pv2: 同pv1
--pv3: 分组内(cookie1)所有的pv累加
--pv4: 分组内当前行+往前3行，如，11号=10号+11号， 12号=10号+11号+12号，
                                13号=10号+11号+12号+13号， 14号=11号+12号+13号
+14号
--pv5: 分组内当前行+往前3行+往后1行，如，14号=11号+12号+13号+14号+15号
=5+7+3+2+4=21
--pv6: 分组内当前行+往后所有行，如，13号=13号+14号+15号+16号=3+2+4+4=13，
                                14号=14号+15号+16号=2+4+4=10

/*
- 如果不指定rows between,默认为从起点到当前行;
- 如果不指定order by，则将分组内所有值累加;
- 关键是理解rows between含义,也叫做window子句：
  - preceding：往前
  - following：往后
  - current row：当前行
  - unbounded：起点
  - unbounded preceding 表示从前面的起点
  - unbounded following：表示到后面的终点
 */
```

1.3.2.3. AVG，MIN，MAX

AVG,MIN,MAX和SUM用法一样

```
select cookieid,createtime,pv,
avg(pv) over(partition by cookieid order by createtime rows between
unbounded preceding and current row) as pv2
from itcast_t1;

select cookieid,createtime,pv,
max(pv) over(partition by cookieid order by createtime rows between
unbounded preceding and current row) as pv2
from itcast_t1;

select cookieid,createtime,pv,
min(pv) over(partition by cookieid order by createtime rows between
unbounded preceding and current row) as pv2
from itcast_t1;
```

1.3.3. Hive分析窗口函数(3)

LAG,LEAD,FIRST_VALUE,LAST_VALUE

1.3.3.1. 准备数据

```
cookie1,2018-04-10 10:00:02,url2
cookie1,2018-04-10 10:00:00,url1
cookie1,2018-04-10 10:03:04,1url3
cookie1,2018-04-10 10:50:05,url6
cookie1,2018-04-10 11:00:00,url7
cookie1,2018-04-10 10:10:00,url4
cookie1,2018-04-10 10:50:01,url5
cookie2,2018-04-10 10:00:02,url22
cookie2,2018-04-10 10:00:00,url11
cookie2,2018-04-10 10:03:04,1url33
cookie2,2018-04-10 10:50:05,url66
cookie2,2018-04-10 11:00:00,url77
cookie2,2018-04-10 10:10:00,url44
cookie2,2018-04-10 10:50:01,url55
```

```
CREATE TABLE itcast_t4 (
cookieid string,
createtime string,   --页面访问时间
url STRING           --被访问页面
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
stored as textfile;

--加载数据：
load data local inpath '/root/hivedata/itcast_t4.dat' into table itcast_t4;
```

1.3.3.2. LAG

LAG(col,n,DEFAULT) 用于统计窗口内往上第n行值第一个参数为列名，第二个参数为往上第n行（可选，默认为1），第三个参数为默认值（当往上第n行为NULL时候，取默认值，如不指定，则为NULL）

```
SELECT cookieid,
  createtime,
  url,
  ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
  LAG(createtime,1,'1970-01-01 00:00:00') OVER(PARTITION BY cookieid ORDER BY createtime) AS last_1_time,
  LAG(createtime,2) OVER(PARTITION BY cookieid ORDER BY createtime) AS last_2_time
  FROM itcast_t4;

 --last_1_time: 指定了往上第1行的值，default为'1970-01-01 00:00:00'
                        cookie1第一行，往上1行为NULL,因此取默认值 1970-01-01 00:00:00
                        cookie1第三行 ,往上1行值为第二行值 ,2015-04-10 10:00:02
                        cookie1第六行 ,往上1行值为第五行值 ,2015-04-10 10:50:01
--last_2_time: 指定了往上第2行的值，为指定默认值
                          cookie1第一行，往上2行为NULL
                          cookie1第二行，往上2行为NULL
                          cookie1第四行，往上2行为第二行值，2015-04-10 10:00:02
                          cookie1第七行，往上2行为第五行值，2015-04-10 10:50:01
```

1.3.3.3. LEAD

与LAG相反LEAD(col,n,DEFAULT) 用于统计窗口内往下第n行值第一个参数为列名，第二个参数为往下第n行（可选，默认为1），第三个参数为默认值（当往下第n行为NULL时候，取默认值，如不指定，则为NULL）

```
SELECT cookieid,
createtime,
url,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
LEAD(createtime,1,'1970-01-01 00:00:00') OVER(PARTITION BY cookieid ORDER
BY createtime) AS next_1_time,
LEAD(createtime,2) OVER(PARTITION BY cookieid ORDER BY createtime) AS
next_2_time
FROM itcast_t4;
```

1.3.3.4. FIRST_VALUE

取分组内排序后，截止到当前行，第一个值

```
SELECT cookieid,
createtime,
url,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
FIRST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime) AS first1

FROM itcast_t4;
```

1.3.3.5. LAST_VALUE

取分组内排序后，截止到当前行，最后一个值

```
SELECT cookieid,
createtime,
url,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
LAST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime) AS last1

FROM itcast_t4;
```

如果想要取分组内排序后最后一个值，则需要变通一下：

```
SELECT cookieid,
  createtime,
  url,
  ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
  LAST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime) AS
last1,
  FIRST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime DESC)
AS last2
  FROM itcast_t4
  ORDER BY cookieid,createtime;
```

1.3.3.6. **特别注意**order by

如果不指定ORDER BY，则进行排序混乱，会出现错误的结果

```
SELECT cookieid,
createtime,
url,
FIRST_VALUE(url) OVER(PARTITION BY cookieid) AS first2

FROM itcast_t4;
```

1.4. Hive自定义函数

1.4.1. 概述

Hive 自带了一些函数，比如：max/min等，但是数量有限，自己可以通过自定义UDF来方便的扩展。
当Hive提供的内置函数无法满足你的业务处理需要时，此时就可以考虑使用用户自定义函数（UDF：user-defined function）。
根据用户自定义函数类别分为以下三种：

1、UDF（User-Defined-Function）

一进一出

2、UDAF（User-Defined Aggregation Function）

聚集函数，多进一出

类似于：count/max/min

3、UDTF（User-Defined Table-Generating Functions）

一进多出

如lateral view explore()

1.4.2. 自定义UDF

编程步骤：

（1）继承org.apache.hadoop.hive.ql.UDF

（2）需要实现evaluate函数；evaluate函数支持重载；

注意事项:

（1）UDF必须要有返回类型，可以返回null，但是返回类型不能为void；

（2）UDF中常用Text/LongWritable等类型，不推荐使用java类型；

1.4.2.1. 简单

1.4.2.2. 第一步：创建maven java 工程，导入jar包

```
<dependencies>
    <dependency>
        <groupId>org.apache.hive</groupId>
        <artifactId>hive-exec</artifactId>
        <version>2.7.5</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
        <version>2.7.5</version>
    </dependency>
</dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.0</version>
```

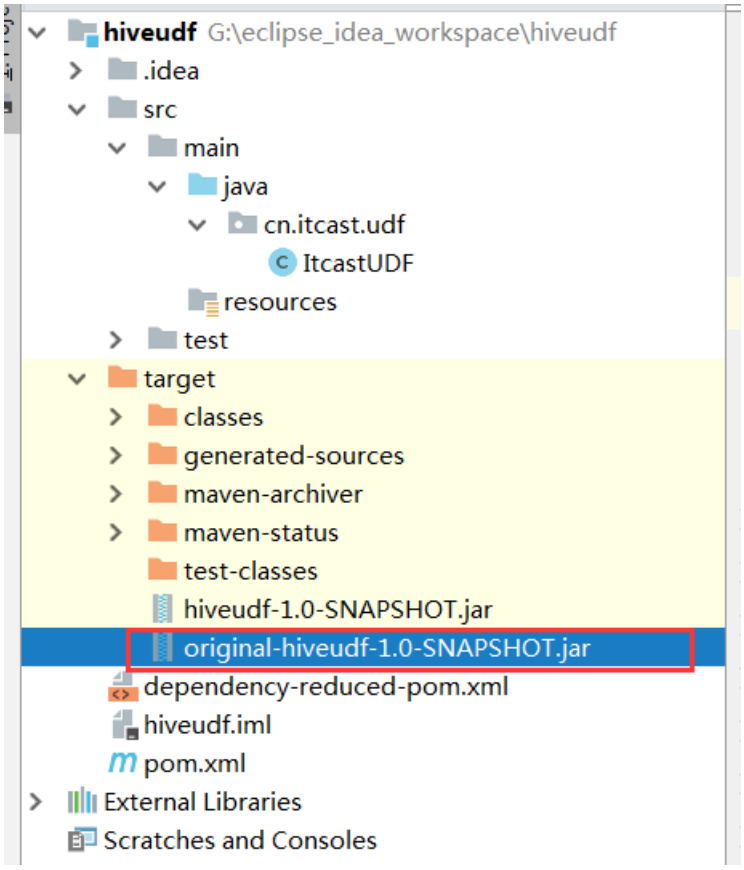
```
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

1.4.2.3. 第二步：开发java类继承UDF，并重载evaluate 方法

```
public class MyUDF    extends UDF{
    public Text evaluate(final Text s) {
        if (null == s) {
            return null;
        }
        //返回大写字母
        return new Text(s.toString().toUpperCase());

    }
}
```

1.4.2.4. 第三步：将我们的项目打包，并上传到hive的lib目录下



1.4.2.5. 第四步：添加我们的jar包

重命名我们的jar包名称

```
cd /export/server/hive-2.7.5/lib
mv original-day_10_hive_udf-1.0-SNAPSHOT.jar my_upper.jar
```

hive的客户端添加我们的jar包

```
add jar /export/server/hive-2.7.5/lib/my_upper.jar;
```

1.4.2.6. 第五步：设置函数与我们的自定义函数关联

```
create temporary function my_upper as 'cn.itcast.udf.ItcastUDF';
```

1.4.2.7. 第六步：使用自定义函数

```
select my_upper('abc');
```

1.4.3. 自定义UDTF

1.4.3.1. 需求

自定义一个UDTF，实现将一个任意分隔符的字符串切割成独立的单词,例如:

源数据：

"zookeeper,hadoop,hdfs,hive,MapReduce"

目标数据:

zookeeper
hadoop
hdfs
hive
MapReduce

1.4.3.2. 代码实现

```
import org.apache.hadoop.hive.ql.exec.UDFArgumentException;
import org.apache.hadoop.hive.ql.metadata.HiveException;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDTF;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspectorFactory;
import org.apache.hadoop.hive.serde2.objectinspector.StructObjectInspector;
import
org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspector
Factory;

import java.util.ArrayList;
import java.util.List;
import java.util.function.ObjDoubleConsumer;

public class MyUDTF extends GenericUDTF {
    private final transient Object[] forwardListObj = new Object[1];

    @Override
    public StructObjectInspector initialize(StructObjectInspector argOIs) throws
UDFArgumentException {
        //设置列名的类型
        List<String> fieldNames = new ArrayList<>();
        //设置列名
        fieldNames.add("column_01");
```

```
        List<ObjectInspector> fieldOIs = new ArrayList<ObjectInspector>()  ;//检查器列表

        //设置输出的列的值类型
        fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);

        return ObjectInspectorFactory.getStandardStructObjectInspector(fieldNames, fieldOIs);

    }

    @Override
    public void process(Object[] objects) throws HiveException {
        //1:获取原始数据
        String args = objects[0].toString();
        //2:获取数据传入的第二个参数，此处为分隔符
        String splitKey = objects[1].toString();
        //3.将原始数据按照传入的分隔符进行切分
        String[] fields = args.split(splitKey);
        //4:遍历切分后的结果，并写出
        for (String field : fields) {
            //将每一个单词添加值对象数组
            forwardListObj[0] = field;
            //将对象数组内容写出
            forward(forwardListObj);
        }

    }

    @Override
    public void close() throws HiveException {

    }
}
```

1.4.3.3. 添加我们的jar包

将打包的jar包上传到node3主机/export/server/hive-2.7.5/lib目录,并重命名我们的jar包名称

```
cd /export/server/hive-2.7.5/lib
mv original-day_10_hive_udtf-1.0-SNAPSHOT.jar my_udtf.jar
```

hive的客户端添加我们的jar包,将jar包添加到hive的classpath下

```
hive>add jar /export/server/hive-2.7.5/lib/my_udtf.jar
```

1.4.3.4. 创建临时函数与开发后的udtf代码关联

```
hive>create temporary function my_udtf as 'cn.itcast.udf.ItcastUDF';
```

1.4.3.5. 使用自定义udtf函数

```
hive>select myudtf("zookeeper,hadoop,hdfs,hive,MapReduce",",") word;
```