

课后作业

2018年9月5日 10:23

作业1 : select count(*) from (select explode(split(words,' '))a1 from word)w1 group by w1.a1;

作业2 : select regexp_replace(info,'[*|&|@|%|#]','')from r1;

嵌套

select regexp_replace(regexp_replace(info,'[*|&|@|%|#]',''),'[.][.],'.')from r1;

作业3 : select regexp_extract(info,'http://(.*)(:)(.*)(:)(.*)',1) from e1;

select * from (select regexp_extract(info,'http://(.*)(:)(.*)(:)(.*)',1)a1 from e1)w1 where
size(split(w1.a1,'[.]'))=3;

```
www.baidu.com
www.sina.com
www.ali.com
www.baidu.com
www.sina.com
www.jd.com
www.yahoo.com
www.google.com
www.badiu.com
www.wangyi.com
www.wangyi.com
www.baidu.com
```

Hive的Join操作

2018年7月27日 13:23

order表数据：

```
1 20180710 P001 20
2 20180710 P002 14
3 20180710 P001 35
4 20180710 P002 40
5 20180710 P001 10
6 20180710 P003 20
7 20180710 P004 12
```

product表数据：

```
P001 xiaomi 2999
P002 huawei 3999
P005 chuizi 4000
```

建表：

```
1 ) create external table order_t (id string,time string,pid string,amount int) row format delimited
fields terminated by ' ' location '/order';
2 ) create external table product_t (pid string,name string,price int) row format delimited fields
terminated by ' ' location '/product';
```

查询：

```
select * from product_t join order_t on product_t.pid=order_t.pid;
```

| | | | | | | |
|------|--------|------|---|----------|------|----|
| P001 | xiaomi | 2999 | 1 | 20180710 | P001 | 20 |
| P002 | huawei | 3999 | 2 | 20180710 | P002 | 14 |
| P001 | xiaomi | 2999 | 3 | 20180710 | P001 | 35 |
| P002 | huawei | 3999 | 4 | 20180710 | P002 | 40 |
| P001 | xiaomi | 2999 | 5 | 20180710 | P001 | 10 |

inner join

```
select * from product_t inner join order_t on product_t.pid=order_t.pid;
```

| | | | | | | |
|------|--------|------|---|----------|------|----|
| P001 | xiaomi | 2999 | 1 | 20180710 | P001 | 20 |
| P002 | huawei | 3999 | 2 | 20180710 | P002 | 14 |
| P001 | xiaomi | 2999 | 3 | 20180710 | P001 | 35 |
| P002 | huawei | 3999 | 4 | 20180710 | P002 | 40 |
| P001 | xiaomi | 2999 | 5 | 20180710 | P001 | 10 |

left join

select * from product_t left join order_t on product_t.pid=order_t.pid;

| | | | | | | |
|------|--------|------|------|----------|------|------|
| P001 | xiaomi | 2999 | 1 | 20180710 | P001 | 20 |
| P001 | xiaomi | 2999 | 3 | 20180710 | P001 | 35 |
| P001 | xiaomi | 2999 | 5 | 20180710 | P001 | 10 |
| P002 | huawei | 3999 | 2 | 20180710 | P002 | 14 |
| P002 | huawei | 3999 | 4 | 20180710 | P002 | 40 |
| P005 | chuizi | 4000 | NULL | NULL | NULL | NULL |

right join

select * from product_t right join order_t on product_t.pid=order_t.pid;

| | | | | | | |
|------|--------|------|---|----------|------|----|
| P001 | xiaomi | 2999 | 1 | 20180710 | P001 | 20 |
| P002 | huawei | 3999 | 2 | 20180710 | P002 | 14 |
| P001 | xiaomi | 2999 | 3 | 20180710 | P001 | 35 |
| P002 | huawei | 3999 | 4 | 20180710 | P002 | 40 |
| P001 | xiaomi | 2999 | 5 | 20180710 | P001 | 10 |
| NULL | NULL | NULL | 6 | 20180710 | P003 | 20 |
| NULL | NULL | NULL | 7 | 20180710 | P004 | 12 |

Full outer join

select * from product_t full outer join order_t on product_t.pid=order_t.pid;

| | | | | | | |
|------|--------|------|------|----------|------|------|
| P001 | xiaomi | 2999 | 5 | 20180710 | P001 | 10 |
| P001 | xiaomi | 2999 | 3 | 20180710 | P001 | 35 |
| P001 | xiaomi | 2999 | 1 | 20180710 | P001 | 20 |
| P002 | huawei | 3999 | 4 | 20180710 | P002 | 40 |
| P002 | huawei | 3999 | 2 | 20180710 | P002 | 14 |
| NULL | NULL | NULL | 6 | 20180710 | P003 | 20 |
| NULL | NULL | NULL | 7 | 20180710 | P004 | 12 |
| P005 | chuizi | 4000 | NULL | NULL | NULL | NULL |

left semi join

select * from product_t left semi join order_t on product_t.pid=order_t.pid;

这种join解决的是exist in (是否存在) 的问题

a表里哪些数据在b表中出现过

| | | |
|------|--------|------|
| P001 | xiaomi | 2999 |
| P002 | huawei | 3999 |

Hive解决数据倾斜问题

2018年7月19日 20:46

概述

什么是数据倾斜以及数据倾斜是怎么产生的？

简单来说数据倾斜就是数据的key的分化严重不均，造成一部分数据很多，一部分数据很少的局面。

举个 word count 的入门例子，它的map 阶段就是形成（“aaa”,1）的形式，然后在reduce 阶段进行 value 相加，得出 “aaa” 出现的次数。若进行 word count 的文本有100G，其中 80G 全部是 “aaa” 剩下 20G 是其余单词，那就会形成 80G 的数据量交给一个 reduce 进行相加，其余 20G 根据 key 不同分散到不同 reduce 进行相加的情况。如此就造成了数据倾斜，临床反应就是 reduce 跑到 99%然后一直在原地等着 那 80G 的reduce 跑完。

如此一来 80G 的 aaa 将发往同一个 reducer，由此就可以知道 reduce 最后 1% 的工作在等什么了。

为什么说数据倾斜与业务逻辑和数据量有关？

从另外角度看数据倾斜，其本质还是在单台节点在执行那一部分数据reduce任务的时候，由于数据量大，跑不动，造成任务卡住。若是这台节点机器内存够大，CPU、网络等资源充足，跑 80G 左右的数据量和跑10M 数据量所耗时间不是很大差距，那么也就不存在问题，倾斜就倾斜吧，反正机器跑的动。所以机器配置和数据量存在一个合理的比例，一旦数据量远超机器的极限，那么不管每个key的数据如何分布，总会有一个key的数据量超出机器的能力，造成 reduce 缓慢甚至卡顿。

业务逻辑造成的数据倾斜会多很多，日常使用过程中，容易造成数据倾斜的原因可以归纳为几点：

- 1) group by
- 2) distinct count(distinct xx)
- 3) join

如何处理group by的数据倾斜问题

1、调优参数

```
set hive.groupby.skewindata=true;
```

hive.groupby.skewindata=true：数据倾斜时负载均衡，当选项设定为true，**生成的查询计划会有两个MRJob**。第一个MRJob中，Map的输出结果集合会随机分布到Reduce中，每个Reduce做部分聚合操作，并输出结果，这样处理的结果是相同的GroupBy Key有可能被分发到不同的Reduce中，从而达到负载均衡的目的；第二个MRJob再根据预处理的数据结果按照GroupBy Key分布到Reduce中（这个过程可以保证相同的GroupBy Key被分布到同一个Reduce中），最后完成最终的聚合操作。

由上面可以看出起到至关重要的作用的其实是第二个参数的设置，它使计算变成了两个mapreduce，先在第一个中在shuffle过程partition时随机给key打标记，使每个key随机均匀分布到各个reduce上计算，但是这样只能完成部分计算，因为相同key没有分配到相同reduce上，所以需要第二次的mapreduce,这次就回归正常shuffle,但是数据分布不均匀的问题在第一次mapreduce已经有了很大的改善，因此基本解决数据倾斜。

Hive优化

2018年7月19日 21:16

1) map side join

mapJoin的主要意思就是，当链接的两个表是一个比较小的表和一个特别大的表的时候，我们把比较小的table直接放到内存中去，然后再对比较大的表格进行map操作。join就发生在map操作的时候，每当扫描一个大的table中的数据，就要去查看小表的数据，哪条与之相符，继而进行连接。**这里的join并不会涉及reduce操作**。map端join的优势就是在于没有shuffle，在实际的应用中，我们这样设置：

```
set hive.auto.convert.join=true;
```

此外，hive有一个参数：hive.mapjoin.smalltable.filesize，默认值是25mb（其中一个表大小小于25mb时，自动启用mapjoin）

2) join语句优化

优化前

```
select m.cid,u.id form order m join customer u on m.cid=u.id where m.dt=' 20160801' ;
```

优化后

```
select m.cid,u.id from (select cid from order where dt=' 20160801' )m  
join customer u on m.cid = u.id
```

注意：Hive在做join时，小表写在前（左边）。

3) group by 优化

```
hive.groupby.skewindata=true
```

如果group by过程出现倾斜，应该设置为true

4) count distinct 优化

优化前

```
select count(distinct id )from tablename
```

优化后

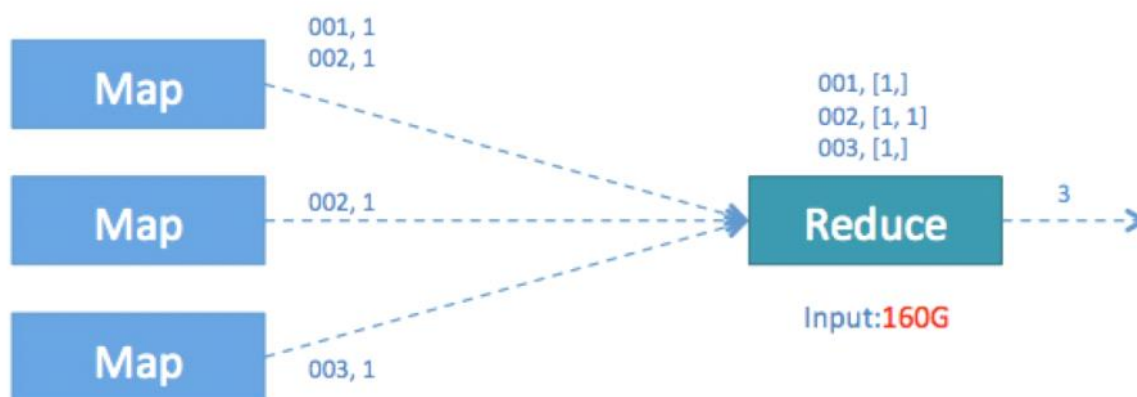
```
select count(*) from (select distinct id from tablename)tmp;
```

日常统计场景中，我们经常会一段时期内的字段进行消重并统计数量，SQL语句类似于

```
SELECT COUNT( DISTINCT id ) FROM TABLE_NAME WHERE ...;
```

这条语句是从一个表的符合WHERE条件的记录中统计不重复的id的总数。

该语句转化为MapReduce作业后执行示意图如下，图中还列出了我们实验作业中Reduce阶段的数据规模：



由于引入了DISTINCT，因此在Map阶段无法利用combine对输出结果消重，必须将id作为Key输出，在Reduce阶段再对来自于不同Map Task、相同Key的结果进行消重，计入最终统计值。

我们看到作业运行时的Reduce Task个数为1，对于统计大数据量时，这会导致最终Map的全部输出由单个的ReduceTask处理。这唯一的Reduce Task需要Shuffle大量的数据，并且进行排序聚合等处理，这使得它成为整个作业的IO和运算瓶颈。

经过上述分析后，我们尝试显式地增大Reduce Task个数来提高Reduce阶段的并发，使每一个Reduce Task的数据处理量控制在2G左右。具体设置如下：

```
set mapred.reduce.tasks=100
```

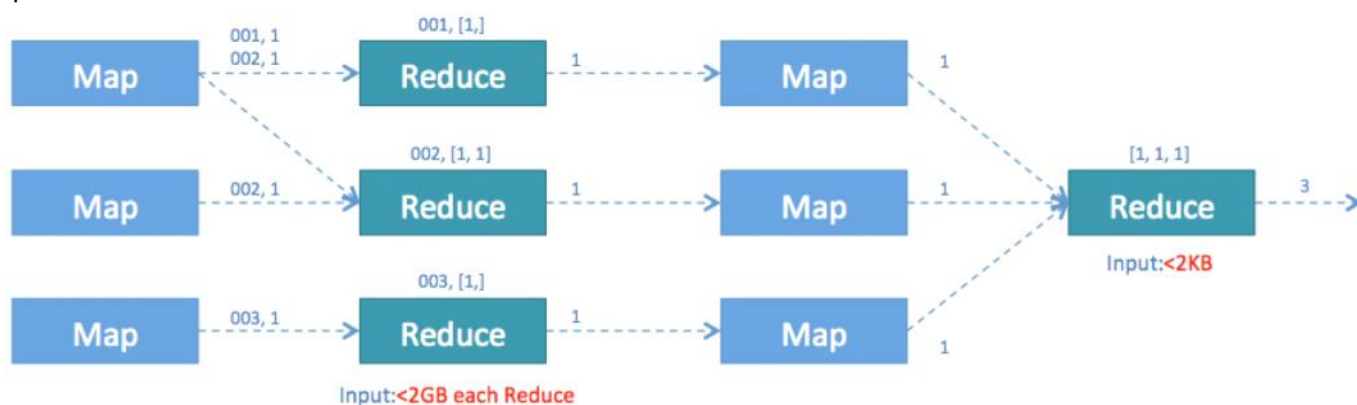
调整后我们发现这一参数并没有影响实际Reduce Task个数，Hive运行时输出“Number of reduce tasks determined at compile time: 1”。

原因是Hive在处理COUNT这种“全聚合(full aggregates)”计算时，它会忽略用户指定的Reduce Task数，而强制使用1。

所以我们只能采用变通的方法来绕过这一限制。我们利用Hive对嵌套语句的支持，**将原来一个MapReduce作业转换为两个作业**，在第一阶段选出全部的非重复id，在第二阶段再对这些已消重的id进行计数。这样在第一阶段我们可以通过增大Reduce的并发数，并发处理Map输出。在第二阶段，由于id已经消重，因此COUNT(*)操作在Map阶段不需要输出原id数据，只输出一个合并后的计数即可。这样即使第二阶段Hive强制指定一个Reduce Task，极少量的Map输出数据也不会使单一的Reduce Task成为瓶颈。改进后的SQL语句如下：

```
SELECT COUNT(*) FROM (SELECT DISTINCT id FROM TABLE_NAME WHERE ... ) t;
```

这一优化使得在同样的运行环境下，优化后的语句执行只需要原语句20%左右的时间。优化后的MapReduce作业流如下：



5) 调整切片数 (map任务数)

Hive底层自动对小文件做了优化，用了CombineTextInputFormat，将做个小文件切片合成一个切片。合成完之后的切片大小，如果>mapred.max.split.size 的大小，就会生成一个新的切片。
mapred.max.split.size 默认是128MB

```
set mapred.max.split.size=134217728 ( 128MB)
```

6) JVM重利用

```
set mapred.job.reuse.jvm.num.tasks=20(默认是1个)
```

JVM重用是hadoop调优参数的内容，对hive的性能具有非常大的影响，特别是对于很难避免小文件的场景或者task特别多的场景，这类场景大多数执行时间都很短。这时JVM的启动过程可能会造成相当大的开销，尤其是执行的job包含有成千上万个task任务的情况。

JVM重用可以使得一个JVM进程在同一个JOB中重新使用N次后才会销毁。

7) 启用严格模式

在hive里面可以通过严格模式防止用户执行那些可能产生意想不到的不好的效果的查询,从而保护hive的集群。

用户可以通过 `set hive.mapred.mode=strict` 来设置严格模式, 改成`unstrict`则为非严格模式。

在严格模式下, 用户在运行如下query的时候会报错:

①分区表的查询没有使用分区字段来限制

②使用了`order by` 但没有使用`limit`语句。(如果不使用`limit`, 会对查询结果进行全局排序, 消耗时间长)

③产生了笛卡尔积

当用户写代码将表的别名写错的时候会引起笛卡尔积, 例如

```
SELECT *
```

```
FROM origindb.promotion__campaign c
```

```
JOIN origindb.promotion__campaignex ce
```

```
ON c.id = c.id
```

```
limit 1000
```

```
CliDriver update main thread name to 9a962abc-afea-470f-9738-dceda72ff1fd
16/08/29 11:38:23 INFO CliDriver: CliDriver update main thread name to 9a962abc-afea-470f-9738-d
ceda72ff1fd

Logging initialized using configuration in jar:file:/opt/meituan/versions/mthive-0.13-package/li
b/hive-common-0.13.1.jar!/hive-log4j.properties
FAILED: SemanticException [Error 10052]: In strict mode, cartesian product is not allowed. If yo
u really want to perform the operation, set hive.mapred.mode=nonstrict
query fails
```

8) 关闭推测执行

因为在测试环境下我们都把应用程序跑通了, 如果还加上推测执行, 如果有一个数据分片本来就会发生数据倾斜, 执行执行时间就是比其他的时间长, 那么hive就会把这个执行时间长的job当作运行失败, 继而又产生一个相同的job去运行, 后果可想而知。可通过如下设置关闭推测执行:

```
set mapreduce.map.speculative=false
```

```
set mapreduce.reduce.speculative=false
```

```
set hive.mapred.reduce.tasks.speculative.execution=false
```

Hive的分桶表

2016年1月30日 11:44

如何使用分桶表

1.创建带桶的 table :

```
create table teacher(class string,name string) clustered by (class) into 3 buckets row format  
delimited fields terminated by ' ';
```

2.开启分桶机制 :

```
set hive.enforce.bucketing=true;
```

3.往表中插入数据 :

```
insert overwrite table teacher select * from tmp; //需要提前准备好temp , 从temp查询数据写入到  
teacher
```

注 : teacher是一个分桶表 , 对于分桶表 , 不允许以外部文件方式导入数据 , 只能从另外一张表数据导入。

temp文件数据样例 :

```
java zhang  
web wang  
java zhao  
java qin  
web liu  
web zheng  
ios li  
linux chen  
ios yang  
ios duan  
linux ma  
linux xu  
java wen  
web wu
```

作用及原理

分桶的原理是根据指定的列的计算hash值模余分桶数量后将数据分开存放。方便数据抽样

```
select * from teacher tablesample(bucket 1 out of 3 on class);
```

注：分桶语法—TABLESAMPLE(BUCKET x OUT OF y)

y必须是table总bucket数的倍数或者因子。hive根据y的大小，决定抽样的比例。

例如：table总共分了3份，当y=3时，抽取($3/3=$)1个bucket的数据，当y=6时，抽取($3/6=$)1/2个bucket的数据。

x表示从哪个bucket开始抽取。

例如：table总bucket数为3，tablesample(bucket 3 out of 3)，表示总共抽取 ($3/3=$) 1个bucket的数据，抽取第3个bucket的数据。

再例如：table总bucket数为32，tablesample(bucket 3 out of 16)，表示总共抽取 ($32/16=$) 2个bucket的数据，分别为第3个bucket和第 ($3+16=$) 19个bucket的数据。

查询第一个桶里数据，并返回一半的数据：

```
select * from bucketed_user tablesample(bucket 1 out of 6 on id);
```

Sqoop安装及指令

2016年1月20日 18:41

Sqoop介绍

sqoop是Apache 提供的工具

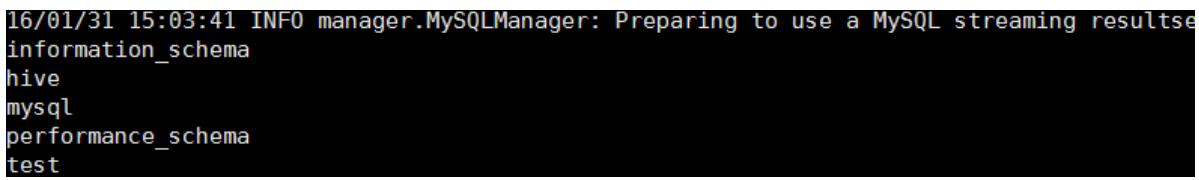
用于hdfs和关系型数据库之间数据的导入和导出

可以从hdfs导出数据到关系型数据库，也可以从关系型数据库导入数据到hdfs。

实现步骤：

- 1.准备sqoop安装包，官网地址：<http://sqoop.apache.org>
- 2.配置jdk环境变量和Hadoop的环境变量。因为sqoop在使用是会去找环境变量对应的路径，从而完整工作。
- 3.sqoop解压即可使用（前提是环境变量都配好了）
- 4.需要将要连接的数据库的驱动包加入sqoop的lib目录下（本例中用的是mysql数据库）
- 5.利用指令操作sqoop

Sqoop基础指令（在Sqoop的bin目录下执行下列指令）

| 说明 | 指令示例 |
|---------------|--|
| 查看mysql所有数据库 | <pre>sh sqoop list-databases --connect jdbc:mysql://192.168.150.138:3306/ -username root -password root</pre>  |
| 查看指定数据库下的所有表 | <pre>sh sqoop list-tables --connect jdbc:mysql://hadoop02:3306/hive -username root -password root</pre> |
| 关系型数据库==>hdfs | <p>实现步骤：</p> <p>1.现在mysql数据库的test数据下建立一张tabx表，并插入测试数据</p> |

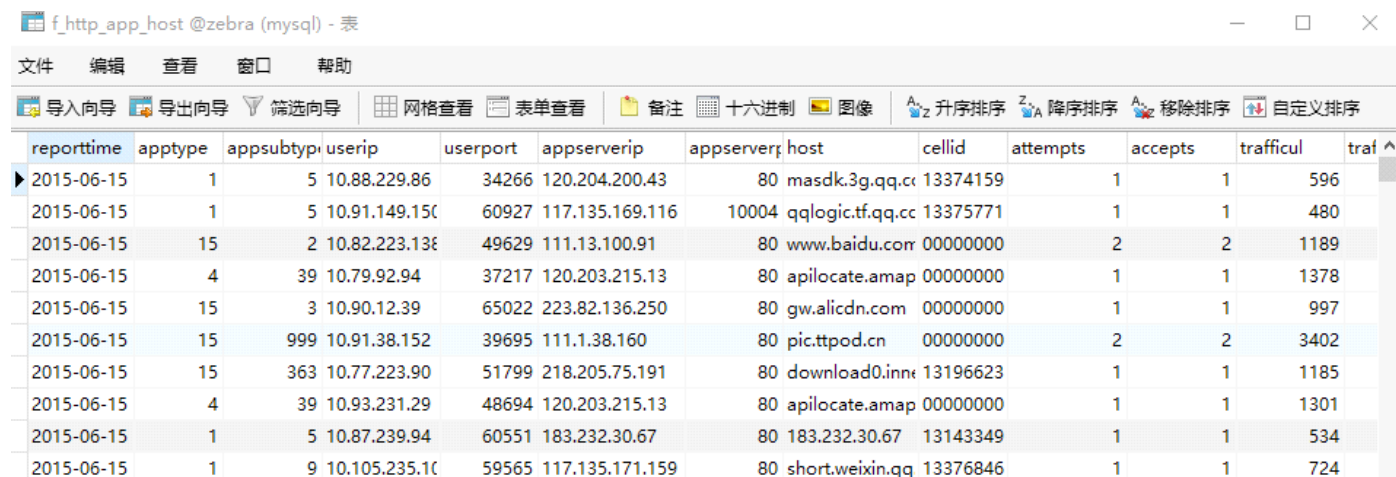
| | |
|--|---|
| sh sqoop import -help (查看import的帮助指令) | <p>建表：create table tabx (id int,name varchar(20));</p> <p>插入：insert into tabx (id,name) values (1,'aaa'),(2,'bbb'),(3,'ccc') , (1,'ddd'),(2,'eee'),(3,'fff');</p> <p>2.进入到sqoop的bin目录下，执行导入语句</p> <p>导入：</p> <p>sh sqoop import --connect jdbc:mysql://192.168.150.138:3306/test --username root --password root --table tabx --target-dir '/sqoop/tabx' --fields-terminated-by ' ' -m 1;</p> |
| hdfs==>关系型数据库 | <p>执行：sh sqoop export --connect jdbc:mysql://192.168.150.138:3306/test --username root --password root --export-dir '/sqoop/tabx/part-m-00000' --table taby -m 1 --fields-terminated-by ' '</p> <p>注：sqoop只能导出数据，不能自动建表。所以在导出之前，要现在mysql数据库里建好对应的表</p> |

Zebra业务回顾

2016年1月31日 16:50

zebra业务回顾

zebra项目最开始阶段会对日志文件进行分析统计，针对apptype,userip等20个字段做了统计，然后把最后的结果落地到数据库里。这张表相当于总表（f_http_app_host）



| reporttime | apptype | appsubtyp | userip | userport | appserverip | appserver | host | cellid | attempts | accepts | trafficul | traf |
|------------|---------|-----------|---------------|----------|-----------------|-----------|------------------|----------|----------|---------|-----------|------|
| 2015-06-15 | 1 | 5 | 10.88.229.86 | 34266 | 120.204.200.43 | 80 | masdk.3g.qq.c | 13374159 | 1 | 1 | 596 | |
| 2015-06-15 | 1 | 5 | 10.91.149.150 | 60927 | 117.135.169.116 | 10004 | qqlogic.tf.qq.cc | 13375771 | 1 | 1 | 480 | |
| 2015-06-15 | 15 | 2 | 10.82.223.138 | 49629 | 111.13.100.91 | 80 | www.baidu.com | 00000000 | 2 | 2 | 1189 | |
| 2015-06-15 | 4 | 39 | 10.79.92.94 | 37217 | 120.203.215.13 | 80 | apilocate.amap | 00000000 | 1 | 1 | 1378 | |
| 2015-06-15 | 15 | 3 | 10.90.12.39 | 65022 | 223.82.136.250 | 80 | gw.alicdn.com | 00000000 | 1 | 1 | 997 | |
| 2015-06-15 | 15 | 999 | 10.91.38.152 | 39695 | 111.1.38.160 | 80 | pic.ttpod.cn | 00000000 | 2 | 2 | 3402 | |
| 2015-06-15 | 15 | 363 | 10.77.223.90 | 51799 | 218.205.75.191 | 80 | download0.inne | 13196623 | 1 | 1 | 1185 | |
| 2015-06-15 | 4 | 39 | 10.93.231.29 | 48694 | 120.203.215.13 | 80 | apilocate.amap | 00000000 | 1 | 1 | 1301 | |
| 2015-06-15 | 1 | 5 | 10.87.239.94 | 60551 | 183.232.30.67 | 80 | 183.232.30.67 | 13143349 | 1 | 1 | 534 | |
| 2015-06-15 | 1 | 9 | 10.105.235.10 | 59565 | 117.135.171.159 | 80 | short.weixin.qq | 13376846 | 1 | 1 | 724 | |

在企业里做到这步并没有结束，因为后续还要做数据分析，可能会针对此表进行多个维度的查询和统计，比如：

- 1.应用欢迎度
- 2.各网站表现
- 3.小区Http上网能力
- 4.小区上网洗好

所以我们可以根据以上四个维度，建立对应的表，并从f_http_app_host 取出对应的数据，然后做统计，最后交给前端做数据可视化的工作。比如下图是针对应用受欢迎程度的数据可视化图：

下图展示了前10名最受欢迎应用，是根据每个应用产生的总量来统计的（一般来说，流量越大，用户越多）



一、应用欢迎度表说明

| D_H_HTTP_APPTYPE(应用欢迎度) | | | |
|-------------------------|------------|----------|-------|
| 序号 | 字段 | 字段类型 | 描述 |
| 0 | hourid | datetime | 小时时间片 |
| 1 | appType | int | 应用大类 |
| 2 | appSubtype | int | 应用小类 |
| 3 | attempts | int(20) | 尝试次数 |
| 4 | accepts | int(20) | 接受次数 |
| 5 | succRatio | int(20) | 尝试成功率 |
| 6 | trafficUL | int(20) | 上行流量 |
| 7 | trafficDL | int(20) | 下行流量 |

| | | | |
|----|---------------|---------|---------|
| 8 | totalTraffic | int(20) | 总流量 |
| 9 | retranUL | int(20) | 重传上行报文数 |
| 10 | retranDL | int(20) | 重传下行报文数 |
| 11 | retranTraffic | int(20) | 重传报文数据 |
| 12 | failCount | int(20) | 延时失败次数 |
| 13 | transDelay | int(20) | 传输时延 |

建表语句：

```
create table D_H_HTTP_APPTYPE(
    hourid datetime,
    apptype int,
    appsubtype int,
    attempts bigint,
    accepts bigint,
    succratio double,
    trafficul bigint,
    trafficdl bigint,
    totaltraffic bigint,
    retranul bigint,
    retrandl bigint,
    retrantraffic bigint,
    failcount bigint,
    transdelay bigint
);
```

业务说明

2016年10月25日 20:10

数据以 | 分割后，每个数据的含义（仅展示项目里用到的字段数据）

| 下标位置 | 字段标识 | 数据类型 | 字段释义 |
|------|------------------|------|--|
| 16 | Cell ID | byte | UE所在小区的ECI |
| 18 | App Type Code | byte | 业务类型编码，参见附录D XDR类型编码定义 |
| 23 | App Sub-type | byte | 应用小类 根据集团定义的识别规则识别出来的小类，参见《中国移动数据流量DPI识别能力规范》。 集团未定义的各厂家根据自己的DPI进行识别 |
| 26 | USER_IP | byte | 终端用户的IPv4地址，如无则填全F |
| 28 | User Port | byte | 用户的四层端口号 |
| 30 | App Server IP | byte | 访问服务器的IPv4地址，如无则填全F |
| 32 | App Server Port | byte | 访问的服务器的端口 |
| 58 | HOST | char | 访问域名 |
| 19 | ProcureStartTime | long | 请求起始时间 |
| 20 | ProcureEndTime | long | 请求结束时间 |
| 22 | App Type | byte | 应用大类更多信息参见《中国移动数据流量DPI识别能力规范》 |
| 33 | UL Data | byte | 上行流量 |
| 34 | DL Data | byte | 下行流量 |
| 39 | RetranUL | byte | 上行TCP重传报文数 |
| 40 | RetranDL | byte | 下行TCP重传报文数 |
| 54 | HTTP/WAP事务状态 | byte | HTTP/WAP2.0层的响应码，参见附录A 状态编码 |

应用大类App Type

| 序号 | 业务类型 | 业务说明 |
|----|------|--|
| 1 | 即时通信 | 互联网消息即时收发业务，如：QQ、飞信等 |
| 2 | 阅读 | 向用户提供在线或离线阅读服务的业务，如：手机阅读、熊猫阅读等 |
| 3 | 微博 | 微博业务，如：移动微博、新浪微博等 |
| 4 | 导航 | 提供浏览、查询、导航等功能的电子地图类业务，如：谷歌地图、高德导航等 |
| 5 | 视频 | 向用户提供音视频内容的直播、分享和下载服务的网站和应用（不包括传统意义上基于P2P技术的视频业务），如：优酷、手机电视等 |
| 6 | 音乐 | 提供音乐在线欣赏和下载服务的网站和应用，如：咪咕音乐、QQ音乐等 |
| 7 | 应用商店 | 提供应用程序、音乐、图书等内容浏览、下载及购买服务的业务，如：Mobile Market、AppStore等 |

| | | |
|----|--------|--|
| 8 | 游戏 | 基于客户端或者网页的游戏业务：QQ游戏、开心农场等 |
| 9 | 支付 | 电子商务类业务，如：手机支付、支付宝、网银等 |
| 10 | 动漫 | 提供动漫在线欣赏和下载服务的网站和应用，如：手机动漫、爱看动漫等 |
| 11 | 邮箱 | 电子邮箱业务，如：139邮箱、QQ邮箱等 |
| 12 | P2P业务 | 基于P2P技术的资源共享业务，包括下载和视频两部分，前者如：迅雷、eMule等，后者如：迅雷看看、PPLive等 |
| 13 | VoIP业务 | 互联网语音通信业务，如：Skype、Uucall等 |
| 14 | 彩信 | 彩信业务 |
| 15 | 浏览下载 | 基于HTTP、WAP、FTP等的普通浏览和下载业务 |
| 16 | 财经 | 金融资讯、股票证券类业务，如：手机商界、大智慧等 |
| 17 | 安全杀毒 | 提供网络安全服务的应用，如：360安全卫士、麦咖啡等；以及网络恶意流量，如：病毒、攻击等 |
| 18 | 其他业务 | |

应用小类 App sub-type

DPI设备子业务识别能力要求（部分）

| 业务类型 | 子业务 | | | |
|------|-----|-----------|-----|-------|
| | 序号 | 子业务名称 | 优先级 | 备注 |
| 即时通信 | 1 | 飞聊 | 必选 | 自有业务 |
| | 2 | 飞信 | 必选 | |
| | 3 | Gtalk | 必选 | 互联网业务 |
| | 4 | MSN | 必选 | |
| | 5 | QQ | 必选 | |
| | 6 | TM | 必选 | |
| | 7 | 阿里旺旺 | 必选 | |
| | 8 | 米聊 | 必选 | |
| | 9 | 微信 | 必选 | |
| | 10 | 人人桌面 | 必选 | |
| | 11 | AOL AIM | 可选 | |
| | 12 | Gadu_Gadu | 可选 | |
| | 13 | go聊 | 可选 | |
| | 14 | ICQ | 可选 | |
| | 15 | IMVU | 可选 | |
| | 16 | Lava-Lava | 可选 | |
| | 17 | NetChat | 可选 | |
| | 18 | Paltalk | 可选 | |
| | 19 | PowWow | 可选 | |
| | 20 | TeamSpeak | 可选 | |
| | 21 | Trillian | 可选 | |

| | | | | |
|--|----|---------|----|--|
| | 22 | VZOchat | 可选 | |
| | 23 | Xfire | 可选 | |
| | 24 | 百度Hi | 可选 | |
| | 25 | 都秀 | 可选 | |
| | 26 | 陌陌 | 可选 | |
| | 27 | 天翼Live | 可选 | |
| | 28 | 翼聊 | 可选 | |
| | 29 | 网易泡泡 | 可选 | |
| | 30 | 新浪UC | 可选 | |
| | 31 | 新浪UT | 可选 | |
| | 32 | 雅虎通 | 可选 | |

| 业务字段 | 字段说明 |
|------|------|
| | |

业务字段处理逻辑

```

HttpAppHost hah=new HttpAppHost();
hah.setReportTime(reportTime);
//上网小区的id
hah.setCellid(data[16]);
//应用类
hah.setAppType(Integer.parseInt(data[22]));
//应用子类
hah.setAppSubtype(Integer.parseInt(data[23]));
//用户ip
hah.setUserIP(data[26]);
//用户port
hah.setUserPort(Integer.parseInt(data[28]));
//访问的服务ip
hah.setAppServerIP(data[30]);
//访问的服务port
hah.setAppServerPort(Integer.parseInt(data[32]));
//域名
hah.setHost(data[58]);

int appTypeCode=Integer.parseInt(data[18]);
String transStatus=data[54];
int appTypeCode=Integer.parseInt(data[18]);
String transStatus=data[54];

//业务逻辑处理
if(hah.getCellid()==null||hah.getCellid().equals("")){
    hah.setCellid("000000000");
}
if(appTypeCode==103){
    hah.setAttempts(1);
}
if(appTypeCode==103
&&"10,11,12,13,14,15,32,33,34,35,36,37,38,48,49,50,51,52,53,54,55,199,200,201,202,203,204,205
,206,302,304,306".contains(transStatus)){
    hah.setAccepts(1);
}else{
    hah.setAccepts(0);
}

```

```

}
if(appTypeCode == 103){
    hah.setTrafficUL(Long.parseLong(data[33]));
}
if(appTypeCode == 103){
    hah.setTrafficDL(Long.parseLong(data[34]));
}
if(appTypeCode == 103){
    hah.setRetranUL(Long.parseLong(data[39]));
}
if(appTypeCode == 103){
    hah.setRetranDL(Long.parseLong(data[40]));
}
if(appTypeCode==103){
    hah.setTransDelay(Long.parseLong(data[20]) - Long.parseLong(data[19]));
}
CharSequence key=hah.getReportTime() + "|" + hah.getAppType() + "|" + hah.getAppSubtype() +
    "|" + hah.getUserIP() + "|" + hah.getUserPort() + "|" + hah.getAppServerIP() + "|" +
    hah.getAppServerPort() + "|" + hah.getHost() + "|" + hah.getCellid();
if(map.containsKey(key)){
    HttpAppHost mapHah=map.get(key);
    mapHah.setAccepts(mapHah.getAccepts()+hah.getAccepts());
    mapHah.setAttempts(mapHah.getAttempts()+hah.getAttempts());
    mapHah.setTrafficUL(mapHah.getTrafficUL()+hah.getTrafficUL());
    mapHah.setTrafficDL(mapHah.getTrafficDL()+hah.getTrafficDL());
    mapHah.setRetranUL(mapHah.getRetranUL()+hah.getRetranUL());
    mapHah.setRetranDL(mapHah.getRetranDL()+hah.getRetranDL());
    mapHah.setTransDelay(mapHah.getTransDelay()+hah.getTransDelay());
    map.put(key, mapHah);
}else{
    map.put(key, hah);
}
}

```

五、zebra业务说明

zebra项目最开始阶段会对日志文件进行分析统计，然后把最后的结果落地到数据库里。

这张表相当于总表

f_http_app_host @zebra (mysql) - 表

| reporttime | apptype | appsubtype | userip | userport | appserverip | appserverport | host | cellid | attempts | accepts | trafficul | traf |
|------------|---------|------------|---------------|----------|-----------------|---------------|------------------|----------|----------|---------|-----------|------|
| 2015-06-15 | 1 | 5 | 10.88.229.86 | 34266 | 120.204.200.43 | 80 | masdk.3g.qq.cc | 13374159 | 1 | 1 | 596 | |
| 2015-06-15 | 1 | 5 | 10.91.149.150 | 60927 | 117.135.169.116 | 10004 | qqlogic.tf.qq.cc | 13375771 | 1 | 1 | 480 | |
| 2015-06-15 | 15 | 2 | 10.82.223.138 | 49629 | 111.13.100.91 | 80 | www.baidu.com | 00000000 | 2 | 2 | 1189 | |
| 2015-06-15 | 4 | 39 | 10.79.92.94 | 37217 | 120.203.215.13 | 80 | apilocate.amap | 00000000 | 1 | 1 | 1378 | |
| 2015-06-15 | 15 | 3 | 10.90.12.39 | 65022 | 223.82.136.250 | 80 | gw.alicdn.com | 00000000 | 1 | 1 | 997 | |
| 2015-06-15 | 15 | 999 | 10.91.38.152 | 39695 | 111.1.38.160 | 80 | pic.ttpod.cn | 00000000 | 2 | 2 | 3402 | |
| 2015-06-15 | 15 | 363 | 10.77.223.90 | 51799 | 218.205.75.191 | 80 | download0.inn | 13196623 | 1 | 1 | 1185 | |
| 2015-06-15 | 4 | 39 | 10.93.231.29 | 48694 | 120.203.215.13 | 80 | apilocate.amap | 00000000 | 1 | 1 | 1301 | |
| 2015-06-15 | 1 | 5 | 10.87.239.94 | 60551 | 183.232.30.67 | 80 | 183.232.30.67 | 13143349 | 1 | 1 | 534 | |
| 2015-06-15 | 1 | 9 | 10.105.235.10 | 59565 | 117.135.171.159 | 80 | short.weixin.qq | 13376846 | 1 | 1 | 724 | |

建表语句

```

create table F_HTTP_APP_HOST(
    reporttime datetime,
    apptype int,
    appsubtype int,
    userip varchar(20),
    userport int,
    appserverip varchar(20),
    appserverport int,

```

```

    host varchar(255),
    cellid varchar(20),
    attempts bigint,
    accepts bigint,
    trafficul bigint,
    trafficdl bigint,
    retransmit bigint,
    retransmitdl bigint,
    failcount bigint,
    transdelay bigint
);

```

后期可能会根据统计出来的数据，进行业务拆分。形成几个不同的维度进行查询：

1.应用欢迎度

2.各网站表现

3.小区Http上网能力

4.小区上网质量好

应用欢迎度表说明

| D_H_HTTP_APPTYPE(应用欢迎度) | | | |
|-------------------------|----------------|----------|---------|
| 序号 | 字段 | 字段类型 | 描述 |
| 0 | hourid | datetime | 小时时间片 |
| 1 | appType | int | 应用大类 |
| 2 | appSubtype | int | 应用小类 |
| 3 | attempts | int(20) | 尝试次数 |
| 4 | accepts | int(20) | 接受次数 |
| 5 | succRatio | int(20) | 尝试成功率 |
| 6 | trafficUL | int(20) | 上行流量 |
| 7 | trafficDL | int(20) | 下行流量 |
| 8 | totalTraffic | int(20) | 总流量 |
| 9 | retransUL | int(20) | 重传上行报文数 |
| 10 | retransDL | int(20) | 重传下行报文数 |
| 11 | retransTraffic | int(20) | 重传报文数据 |
| 12 | failCount | int(20) | 延时失败次数 |
| 13 | transDelay | int(20) | 传输时延 |

建表语句：

```

create table D_H_HTTP_APPTYPE(
    hourid datetime,
    apptype int,
    appsubtype int,
    attempts bigint,
    accepts bigint,
    succratio bigint,
    trafficul bigint,
    trafficdl bigint,
    totaltraffic bigint,
    retransmit bigint,
    retransmitdl bigint,
    retransmittraffic bigint,
    failcount bigint,
    transdelay bigint
);

```

各网站的表现表

| D_H_HTTP_HOST（各网站的表现） | | | |
|-----------------------|---------------|-------------|---------|
| 序号 | 字段 | 字段类型 | 描述 |
| 0 | hourid | datetime | 小时时间片 |
| 1 | host | varchar(50) | 域名 |
| 2 | appServerIP | varchar(20) | 服务器IP |
| 3 | attempts | int(20) | 尝试次数 |
| 4 | accepts | int(20) | 接受次数 |
| 5 | succRatio | int(20) | 尝试成功率 |
| 6 | trafficUL | int(20) | 上行流量 |
| 7 | trafficDL | int(20) | 下行流量 |
| 8 | totalTraffic | int(20) | 总流量 |
| 9 | retranUL | int(20) | 重传上行报文数 |
| 10 | retranDL | int(20) | 重传下行报文数 |
| 11 | retranTraffic | int(20) | 重传报文数据 |
| 12 | failCount | int(20) | 延时失败次数 |
| 13 | transDelay | int(20) | 传输时延 |

建表语句：

#创建各网站表现表

```
create table D_H_HTTP_HOST(  
    hourid datetime,  
    host varchar(255),  
    appserverip varchar(20),  
    attempts bigint,  
    accepts bigint,  
    succratio bigint,  
    trafficul bigint,  
    trafficdl bigint,  
    totaltraffic bigint,  
    retranul bigint,  
    retrandl bigint,  
    retrantraffic bigint,  
    failcount bigint,  
    transdelay bigint  
);
```

小区HTTP上网能力表

| D_H_HTTP_CELLID（小区HTTP上网能力） | | | |
|-----------------------------|---------------|----------|---------|
| 序号 | 字段 | 字段类型 | 描述 |
| 0 | hourid | datetime | 小时时间片 |
| 1 | cellid | varchar | 小区ID |
| 2 | attempts | int(20) | 尝试次数 |
| 3 | accepts | int(20) | 接受次数 |
| 4 | succRatio | int(20) | 尝试成功率 |
| 5 | trafficUL | int(20) | 上行流量 |
| 6 | trafficDL | int(20) | 下行流量 |
| 7 | totalTraffic | int(20) | 总流量 |
| 8 | retranUL | int(20) | 重传上行报文数 |
| 9 | retranDL | int(20) | 重传下行报文数 |
| 10 | retranTraffic | int(20) | 重传报文数据 |
| 11 | failCount | int(20) | 延时失败次数 |
| 12 | transDelay | int(20) | 传输时延 |

创建小区HTTP上网能力表

```
create table D_H_HTTP_CELLID(  
    hourid datetime,  
    cellid varchar(20),  
    attempts bigint,  
    accepts bigint,  
    succratio bigint,  
    trafficul bigint,  
    trafficdl bigint,  
    totaltraffic bigint,  
    retranul bigint,  
    retrandl bigint,  
    retrantraffic bigint,  
    failcount bigint,  
    transdelay bigint  
);
```

```

hourid datetime,
cellid varchar(20),
attempts bigint,
accepts bigint,
succratio bigint,
trafficul bigint,
trafficdl bigint,
totaltraffic bigint,
retranul bigint,
retrandl bigint,
retrantraffic bigint,
failcount bigint,
transdelay bigint
);

```

小区上网喜好表

| D_H_HTTP_CELLID_HOST（小区上网喜好） | | | |
|------------------------------|---------------|-------------|---------|
| 序号 | 字段 | 字段类型 | 描述 |
| 0 | hourid | datetime | 小时时间片 |
| 1 | cellid | varchar | 小区ID |
| 2 | host | varchar(50) | 域名 |
| 3 | attempts | int(20) | 尝试次数 |
| 4 | accepts | int(20) | 接受次数 |
| 5 | succRatio | int(20) | 尝试成功率 |
| 6 | trafficUL | int(20) | 上行流量 |
| 7 | trafficDL | int(20) | 下行流量 |
| 8 | totalTraffic | int(20) | 总流量 |
| 9 | retranUL | int(20) | 重传上行报文数 |
| 10 | retranDL | int(20) | 重传下行报文数 |
| 11 | retranTraffic | int(20) | 重传报文数据 |
| 12 | failCount | int(20) | 延时失败次数 |
| 13 | transDelay | int(20) | 传输时延 |

#创建小区上网喜好表

```

create table D_H_HTTP_CELLID_HOST(
    hourid datetime,
    cellid varchar(20),
    host varchar(255),
    attempts bigint,
    accepts bigint,
    succratio bigint,
    trafficul bigint,
    trafficdl bigint,
    totaltraffic bigint,
    retranul bigint,
    retrandl bigint,
    retrantraffic bigint,
    failcount bigint,
    transdelay bigint
);

```


web应用展示zebra业务

2015年3月29日 22:43

此应用只针对 **应用受欢迎度表 (D_H_HTTP_APPTYPE 表)** 做统计

实现步骤：

1. 数据库建表
2. 插入测试数据
3. 搭建web应用
4. 启动tomcat服务器
5. 浏览测试。在浏览器输入：http://localhost:端口号/项目名 即可访问

最后的展示效果：



Hive实现Zebra

2016年1月20日 13:22

实现流程：

使用flume收集数据 --> 落地到hdfs系统中 --> 创建hive的外部表管理hdfs中收集到的日志 --> 利用hql处理zebra的业务逻辑 --> 使用sqoop技术将hdfs中处理完成的数据导出到mysql中

flume组件工作说明：

flume在收集日志的时候，按天为单位进行收集。hive在处理的时候，按天作为分区条件，继而对每天的日志进行统计分析。最后，hive将统计分析的结果利用sqoop导出到关系型数据库里，然后做数据可视化的相关工作。

对于时间的记录，一种思路是把日志文件名里的日志信息拿出来，第二种思路是flume在收集日志时，将当天的日期记录下来。我们用第二种思路。

Flume配置

配置示例：

```
a1.sources=r1
a1.channels=c1
a1.sinks=s1

a1.sources.r1.type=spooldir
a1.sources.r1.spoolDir=/home/zebra
a1.sources.r1.interceptors=i1
a1.sources.r1.interceptors.i1.type=timestamp

a1.sinks.s1.type=hdfs
a1.sinks.s1.hdfs.path=hdfs://192.168.150.137:9000/zebra/reportTime=%Y-%m-%d
a1.sinks.s1.hdfs.fileType=DataStream
a1.sinks.s1.hdfs.rollInterval=30
a1.sinks.s1.hdfs.rollSize=0
a1.sinks.s1.hdfs.rollCount=0

a1.channels.c1.type=memory

a1.sources.r1.channels=c1
a1.sinks.s1.channel=c1
```

将待处理的日志文件上传到/home/zebra下，最终，这个文件会被01虚拟机收集到，最后落到hdfs上。

注意：在上传日志文件的时候，不要在/root/work/data/flumedata目录下通过rz上传，因为rz是连续传输文件，这样会使得flume在处理时报错，错误为正在处理的日志文件大小被修改，所以最好是先把日志上传到linux的其他目录下，然后通过mv 指令移动到/root/work/data/flumedata目录下

Hive组件工作流程（建表语句不用写，重在了解整个ETL过程，这个过程很重要）

使用hive，创建zebra数据库

执行：create database zebra;

执行：use zebra;

然后建立分区，再建立表

详细建表语句：

```
create EXTERNAL table zebra (a1 string,a2 string,a3 string,a4 string,a5 string,a6 string,a7
string,a8 string,a9 string,a10 string,a11 string,a12 string,a13 string,a14 string,a15
string,a16 string,a17 string,a18 string,a19 string,a20 string,a21 string,a22 string,a23
string,a24 string,a25 string,a26 string,a27 string,a28 string,a29 string,a30 string,a31
string,a32 string,a33 string,a34 string,a35 string,a36 string,a37 string,a38 string,a39
string,a40 string,a41 string,a42 string,a43 string,a44 string,a45 string,a46 string,a47
string,a48 string,a49 string,a50 string,a51 string,a52 string,a53 string,a54 string,a55
string,a56 string,a57 string,a58 string,a59 string,a60 string,a61 string,a62 string,a63
string,a64 string,a65 string,a66 string,a67 string,a68 string,a69 string,a70 string,a71
```

```
string,a72 string,a73 string,a74 string,a75 string,a76 string,a77 string) partitioned by
(reportTime string) row format delimited fields terminated by '|' stored as textfile location
'/zebra';
```

增加分区操作

执行：ALTER TABLE zebra add PARTITION (reportTime='2018-09-06') location '/zebra/reportTime=2018-09-06';

执行查询，看是否能查出数据

可以通过抽样语法来检验：select * from zebra TABLESAMPLE (1 ROWS);

```
Mobile Safari/534.30 text/html http://detail.m.tmall.com/item.htm?id=403029774
78&pos=1 cookie2=1f79b50a70a5b5c623c7904336806420; t=15d42b44dee4ae0a412405c059
Fr4zjaEkT98e1%2Fg9rN1nBMQUUuR%2FpAxDov07f5ednNQjnqIxEN0WI1YD 0 0 0
88 11 93287887015639365 6 1
09649427375 1 15 155 0 10.83.106.110 55697
48 14600 1400 1 0 1 3 6 200 51
d526334abae83a1368e200e7e7290be%26ttid%3D600000%2540taobao_android_4.9.0&ttid=&token=
roid 4.3; zh-cn; vivo X3L Build/JLS36C) AppleWebKit/534.30 (KHTML, like Gecko) Version
SIONID=2BE2F826770EA5C0E2B298A581D72AF5; isg=28E892BDBBD711DD1988D030D50D0554; uc1=co
CLz3%2F65A% 0 0 0 0 3 0 217
016 11 93287887015639366 6 1
09649435581 1 15 3 0 10.83.106.110 55714
49 14600 1400 1 0 1 3 6 200 56
login.m.taobao.com Mozilla/5.0 (Linux; U; Android 4.3; zh-cn; vivo X3L Bu
xt/html http://login.taobao.com/member/logout.jhtml?loginURL=http%3A%2F%2Flogin
/RXTsCAAd9oCsBM0mh0; _m_unitapi_v_=1409121915930; ALIPAYJSESSIONID=101f32dc35e44f9f92a
1409644839533; _m_h5_tk_enc=5bf6a44 0 0 0 0
34 11 93287887015639367 6 1
09649434007 1 15 3 0 10.98.67.79 55087
51 65535 1400 1 0 1 3 6 200 67
```

清洗数据，从原来的77个字段变为23个字段

建表语句：

```
create table dataclear(reporttime string,appType bigint,appSubtype bigint,userIp
string,userPort bigint,appServerIP string,appServerPort bigint,host string,cellid
string,appTypeCode bigint,interruptType String,transStatus bigint,trafficUL bigint,trafficDL
```

```
bigint, retranUL bigint, retranDL bigint, procedureStartTime bigint, procedureEndTime bigint)row  
format delimited fields terminated by '|';
```

从zebra表里导出数据到dataclear表里 (23个字段的值)

建表语句：

```
insert overwrite table dataclear select concat(reportTime,'  
'00:00:00'), a23, a24, a27, a29, a31, a33, a59, a17, a19, a68, a55, a34, a35, a40, a41, a20, a21 from zebra;
```

处理业务逻辑，得到dataproc表

建表语句：

```
create table dataproc (reporttime string, appType bigint, appSubtype bigint, userIp  
string, userPort bigint, appServerIP string, appServerPort bigint, host string, cellid  
string, attempts bigint, accepts bigint, trafficUL bigint, trafficDL bigint, retranUL  
bigint, retranDL bigint, failCount bigint, transDelay bigint)row format delimited fields  
terminated by '|';
```

根据业务规则，做字段处理

建表语句：

```
insert overwrite table dataproc select  
reporttime, appType, appSubtype, userIp, userPort, appServerIP, appServerPort, host,  
if(cellid == '', "000000000", cellid), if(appTypeCode == 103, 1, 0), if(appTypeCode == 103 and  
find_in_set(transStatus, "10, 11, 12, 13, 14, 15, 32, 33, 34, 35, 36, 37, 38, 48, 49, 50, 51, 52, 53, 54, 55, 199, 200  
, 201, 202, 203, 204, 205, 206, 302, 304, 306") != 0 and interruptType == 0, 1, 0), if(apptypeCode ==  
103, trafficUL, 0), if(apptypeCode == 103, trafficDL, 0), if(apptypeCode == 103, retranUL, 0),  
if(apptypeCode == 103, retranDL, 0), if(appTypeCode == 103 and transStatus == 1 and interruptType  
== 0, 1, 0), if(appTypeCode == 103, procedureEndTime - procedureStartTime, 0) from dataclear;
```

查询关心的信息，以应用受欢迎程度表为例：

建表语句：

```
create table D_H_HTTP_APPTYPE(hourid string, appType int, appSubtype int, attempts bigint, accepts
bigint, succRatio double, trafficUL bigint, trafficDL bigint, totalTraffic bigint, retranUL
bigint, retranDL bigint, retranTraffic bigint, failCount bigint, transDelay bigint) row format
delimited fields terminated by ' ';
```

根据总表dataproc,按条件做聚合以及字段的累加

建表语句：

```
insert overwrite table D_H_HTTP_APPTYPE select
reporttime,apptype,appsubtype,sum(attempts),sum(accepts),round(sum(accepts)/sum(attempts),2),
sum(trafficUL),sum(trafficDL),sum(trafficUL)+sum(trafficDL),sum(retranUL),sum(retranDL),sum(retra
nUL)+sum(retranDL),sum(failCount),sum(transDelay)from dataproc group by
reporttime,apptype,appsubtype;
```

18.查询前5名受欢迎app

```
select hourid, apptype, sum(totalTraffic) as tt from D_H_HTTP_APPTYPE group by hourid, apptype
sort by tt desc limit 5;
```

Sqoop组件工作流程：

将Hive表导出到Mysql数据库中，然后通过web应用程序+echarts做数据可视化工作。

实现步骤（讲完sqoop后，作为课后作业）：

- 1.在mysql建立对应的表
- 2.利用sqoop导出d_h_http_apptype表

导出语句：

```
sh sqoop export --connect jdbc:mysql://192.168.150.138:3306/zebra --username root --password  
root --export-dir '/user/hive/warehouse/zebra.db/d_h_http_apptype/000000_0' --table  
D_H_HTTP_APPTYPE -m 1 --fields-terminated-by '|'
```

Hive JDBC

2016年1月20日 19:24

Hive的jdbc编程

hive实现了jdbc接口，所以可以通过java代码操作。但是实际应用中用的不多，一般都是在HDFS储存的文件基础上建立外部表来进行查询处理。所以jdbc了解一下即可。

实现步骤：

1. 在服务器端开启HiveServer服务

```
./hive --service hiveserver2 & （以后台线程启动）
```

2. 创建本地工程，导入jar包

导入hive\lib目录下的hive-jdbc-1.2.0-standalone.jar

导入hadoop-2.7.1\share\hadoop\common下的hadoop-common-2.7.1.jar

3. 编写jdbc代码执行



代码示例：

```
/*  
  
 * 连接和查询  
  
 */  
  
@Test  
  
public void testConnectAndQuery() throws Exception {  
  
    //注册数据库驱动，用的hive的jdbc，驱动名固定写死  
  
    Class.forName("org.apache.hive.jdbc.HiveDriver");  
  
    //如果用的是hive2服务，则写jdbc:hive2，后面跟上hive服务器的ip以及端口号，端口号默  
    认是10000
```



```

Connection conn =

DriverManager.getConnection("jdbc:hive2://192.168.234.21:10000/park","root","root"

);

Statement stat = conn.createStatement();

ResultSet rs = stat.executeQuery("select * from stu");

while(rs.next()){

    String name = rs.getString("name");

    System.out.println(name);

}

stat.close();

conn.close();

}

/*

* 利用executeUpdate ( ) 方法是实现建表的创建、插入数据及删除表

*/

@Test

public void testInsert() throws Exception{

    Class.forName("org.apache.hive.jdbc.HiveDriver");

    Connection conn =

    DriverManager.getConnection("jdbc:hive2://192.168.234.21:10000/park","root","r

oot");

    Statement stat = conn.createStatement();

    //executeUpdate可用于：创建表,向表中插入数据以及删除表

    stat.executeUpdate("insert into table stu values(2,'rose')");

```

```
stat.executeUpdate("create table stu2(id int,name string) row format delimited  
  
fields terminated by ' '");  
  
stat.executeUpdate("drop table stu2");  
  
  
  
stat.close();  
  
stat.close();  
  
}
```

Hive特点

2015年3月29日 16:51

针对海量数据的高性能查询和分析系统

由于 Hive 的查询是通过 MapReduce 框架实现的，而 MapReduce 本身就是为实现针对海量数据的高性能处理而设计的。所以 Hive 天然就能高效的处理海量数据。

与此同时，Hive 针对 HiveQL 到 MapReduce的翻译进行了大量的优化，从而保证了生成的 MapReduce 任务是高效的。在实际应用中，Hive 可以高效的对 TB 甚至 PB级的数据进行处理。

类SQL的查询语言

HiveQL 和 SQL 非常类似，所以一个熟悉SQL 的用户基本不需要培训就可以非常容易的使用 Hive 进行很复杂的查询。

HiveQL 灵活的可扩展性(Extendibility)

除了 HiveQL 自身提供的能力，用户还可以自定义其使用的数据类型、也可以用任何语言自定义 mapper 和 reducer 脚本，还可以自定义函数(普通函数、聚集函数)等。这就赋予了 HiveQL 极大的可扩展性。用户可以利用这种可扩展性实现非常复杂的查询。

高扩展性(Scalability)和容错性

Hive本身并没有执行机制，用户查询的执行是通过 MapReduce 框架实现的。由于MapReduce 框架本身具有高度可扩展(计算能力随 Hadoop 机群中机器的数量增加而线性增加)和高容错的特点，所以 Hive也相应具有这些特点。

与 Hadoop 其他产品完全兼容

Hive 自身并不存储用户数据，而是通过接口访问用户数据。这就使得 Hive 支持各种数据源和数据格式。例如，它支持处理 HDFS 上的多种文件格式 (TextFile、SequenceFile 等)，还支持处理 HBase 数据库。用户也完全可以实现自己的驱动来增加新的数据源和数据格式。一种理想的应用模型是将数据存储在 HBase 中实现实时访问，而用 Hive 对 HBase 中的数据进行批量分析。