天善智能
TIANSHAN SOFT

标签数据存储

讲师：watermelon

Hive存储及应用

Hive存储数据的特点

Hive存储所有标签相关数据的计算结果集。

因为跑用户相关的作业计算量非常大，相关作业的执行一般使用mp或spark作为执行引擎，将跑出来的结果写入到HDFS上，这个作业过程不可能在MySQL、hbase、mongodb等其他数据库中实现

Hive存储标签相关的数据涉及到的一些表，包括用户标签表、标签聚合的表、人群计算的表等等，下面会做详细的介绍

用户标签相关的表结构设计—tag表(1)

表结构设计

```
CREATE TABLE `dw.profile_tag_userid`(
`tagid` string COMMENT 'tagid',
`userid` string COMMENT 'userid',
`tagweight` string COMMENT 'tagweight',
`reserve1` string COMMENT '预留1',
`reserve2` string COMMENT '预留2',
`reserve3` string COMMENT '预留3')
COMMENT 'tagid维度userid 用户画像数据'
PARTITIONED BY ( `data_date` string COMMENT '数据日期', `tagtype` string COMMENT '标签主题分类')
```

该表下面记录了标签id、用户id、标签权重等主要字段。按日期和标签主题作为分区。标签主题也作为分区是为了做ETL调度时方便，可以同时计算多个标签插入到该表下面

向hive里面插入几条测试数据，看一下效果

```
--------------------------------插入几条测试数据----------------------------------------
insert into table dw.profile_tag_userid partition(data_date='20180421', tagtype='user_install_days')
values('A220U029_001','25083679','282','','','');
insert into table dw.profile_tag_userid partition(data_date='20180421', tagtype='user_install_days')
values('A220U029_001','7306783','166','','','');
insert into table dw.profile_tag_userid partition(data_date='20180421', tagtype='user_install_days')
values('A220U029_001','4212236','458','','','');
insert into table dw.profile_tag_userid partition(data_date='20180421', tagtype='user_install_days')
values('A220U029_001','39730187','22','','','');
insert into table dw.profile_tag_userid partition(data_date='20180421', tagtype='user_install_days')
values('A220U029_001','16254215','57','','','');
insert into table dw.profile_tag_userid partition(data_date='20180421', tagtype='userid_all_paid_money')
values('A220U083_001','25083679','800.39','','','');
insert into table dw.profile_tag_userid partition(data_date='20180421', tagtype='userid_all_paid_money')
values('A220U083_001','7306783','311.29','','','');
insert into table dw.profile_tag_userid partition(data_date='20180421', tagtype='userid_all_paid_money')
values('A220U083_001','32171777','129.65','','','');
insert into table dw.profile_tag_userid partition(data_date='20180421', tagtype='userid_all_paid_money')
values('A220U083_001','40382657','602.3','','','');
insert into table dw.profile_tag_userid partition(data_date='20180421', tagtype='userid_all_paid_money')
values('A220U083_001','30765587','465.93','','','');
```

用户标签相关的表结构设计—tag表(2)

数据存储

```
hive> select * from dw.profile_tag_userid;
OK
A220U029_001    25083679        282                             20180421        user_install_days
A220U029_001    7306783 166                             20180421        user_install_days
A220U029_001    4212236 458                             20180421        user_install_days
A220U029_001    39730187        22                              20180421        user_install_days
A220U029_001    16254215        57                              20180421        user_install_days
A220U083_001    25083679        800.39                          20180421        userid_all_paid_money
A220U083_001    7306783 311.29                          20180421        userid_all_paid_money
A220U083_001    32171777        129.65                          20180421        userid_all_paid_money
A220U083_001    40382657        602.3                           20180421        userid_all_paid_money
A220U083_001    30765587        465.93                          20180421        userid_all_paid_money
Time taken: 0.713 seconds, Fetched: 10 row(s)
```

通过设置标签类型这个分区字段，可以同时向该表中插入一个用户的不同类型的标签

存储路径

hdfs://master:9000/root/hive/warehouse/dw.db/profile_tag_userid/data_date=20180421/tagtype=userid_all_paid_money

用户标签在userid和cookieid维度各做了一套，同理适用于cookieid维度的表

```
CREATE TABLE `dw.profile_tag_cookieid`(
`tagid` string COMMENT 'tagid',
`cookieid` string COMMENT 'cookieid',
`tagweight` string COMMENT '标签权重',
`reserve1` string COMMENT '预留1',
`reserve2` string COMMENT '预留2',
`reserve3` string COMMENT '预留3')
COMMENT 'tagid维度的用户cookie画像数据'
PARTITIONED BY ( `data_date` string COMMENT '数据日期', `tagtype` string COMMENT '标签主题分类')
```

***总结：*可以建立时间分区+标签类型分区的tag表，用于插入每天用户相关的每一个标签到相应的分区下**

用户标签相关的表结构设计—tag-map表(1)

表结构设计

```
CREATE TABLE `dw.profile_user_map_userid`(
`userid` string COMMENT 'userid',
`tagsmap` map<string,string> COMMENT 'tagsmap',
`reserve1` string COMMENT '预留1',
`reserve2` string COMMENT '预留2')
COMMENT 'userid 用户画像数据'
PARTITIONED BY (`data_date` string COMMENT '数据日期')
```

数据存储

```
    > select * from dw.profile_user_map_userid;
OK
16254215        {"A220U029_001":"57"}                   20180421
25083679        {"A220U029_001":"282","A220U083_001":"800.39"}                  20180421
30765587        {"A220U083_001":"465.93"}               20180421
32171777        {"A220U083_001":"129.65"}               20180421
39730187        {"A220U029_001":"22"}                   20180421
40382657        {"A220U083_001":"602.3"}                20180421
4212236 {"A220U029_001":"458"}                  20180421
7306783 {"A220U029_001":"166","A220U083_001":"311.29"}                  20180421
Time taken: 0.434 seconds, Fetched: 8 row(s)
```

这里将一个用户身上的所有标签进行聚合

用户标签的聚合在userid和cookieid维度各做了一套，同理适用于cookieid维度的表

```
CREATE TABLE `dw.profile_user_map_cookieid`(
`cookieid` string COMMENT 'tagid',
`tagsmap` map<string,string> COMMENT 'cookieid',
`reserve1` string COMMENT '预留1',
`reserve2` string COMMENT '预留2')
COMMENT 'cookie 用户画像数据'
PARTITIONED BY (`data_date` string COMMENT '数据日期')
```

用户标签相关的表结构设计—tag-map表(2)

总结：从dw.profile_tag_userid 到 dw.profile_user_map_userid，实际是将同一个用户的多个标签聚合在一起。在tag表中，通过设置标签类型这一分区字段的形式，将每个用户身上的多个标签存储在了不同的分区下面。通过tag-map表，将一个用户的全部标签聚合在了一起。

标签聚合的执行命令

```
  insert overwrite table dw.profile_user_map_userid  partition(data_date="20180910")
  select userid,
         str_to_map(concat_ws(',',collect_set(concat(tagid,':',tagweight)))) as tagsmap,
         '',
         ''
    from dw.profile_tag_userid
   where data_date="20180910"
group by userid
```

标签聚合的执行过程

```
   > insert overwrite table dw.profile_user_map_userid  partition(data_date="20180421") select userid,str_to_map(concat_ws(',',collect_set(concat(t
agid,':',tagweight)))) as tagsmap,'','' from dw.profile_tag_userid where data_date="20180421" group by userid;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark
, tez) or using Hive 1.X releases.
Query ID = root_20180924001127_53cbe0ef-2e53-4ff1-8390-a278bf085333
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1536058452188_0011, Tracking URL = http://master:8088/proxy/application_1536058452188_0011/
Kill Command = /root/data/hadoop-2.7.5/bin/hadoop job  -kill job_1536058452188_0011
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2018-09-24 00:11:43,927 Stage-1 map = 0%,  reduce = 0%
2018-09-24 00:11:57,480 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 4.13 sec
2018-09-24 00:12:03,454 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 8.87 sec
2018-09-24 00:12:12,047 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 12.46 sec
MapReduce Total cumulative CPU time: 12 seconds 460 msec
Ended Job = job_1536058452188_0011
Loading data to table dw.profile_user_map_userid partition (data_date=20180421)
```

用户人群相关表设计

```
CREATE TABLE `dw.profile_usergroup_tag`(
`userid` string,
`tagsmap` map<string,string>,
`reserve1` string,
`reserve2` string)
PARTITIONED BY (`data_date` string, `target` string)
```

用户人群表主要记录了用户的id、人群名称id以及推送到的业务系统

```
val tablename="dw.profile_usergroup_tag"

val dataset = spark.sql(
  s"""
     |    select t1.userid,t2.order_sn,t3.tel
     |      from ${tablename} t1
     | inner join dw.paid_order_fact t2
     |         on t1.userid = t2.user_id
     | inner join dw.order_user_info t3
     |         on t2.order_id = t3.order_id
     | where t1.data_date = '${data_date}'
     |   and t1.target = "100000207486"
     | group by t1.userid,t3.tel
     | having t3.tel <> ''
  """.stripMargin)
```

这里通过人群表t1 join了t2订单表，得到用户的订单编号，然后通过t2订单表join用户收货信息表t3，得到用户的 手机号；

这样通过圈定人群可以得到一批运营用户及他们的手机号，可以推送给外呼中心进行外呼操作

MySQL存储及应用

MySQL存储画像相关数据类型

MySQL在画像中存储的数据主要包括以下几种类型：

- 画像标签的元数据（元数据在后面讲画像产品化的时候回提到）；
- 结果集的校验（标签量级的监控、Hive同步到hbase的校验、数据波动的校验）；
- 同步到业务系统中的数据（业务方使用的数据库，如客服系统使用的MySQL，使用sqoop同步相关的数据）；

画像标签的元数据管理

元数据信息读取的数据

tag_id	tag_chinese_name	tag_theme	level_1_id	level_1_name	level_2_id	level_2_name	tag_type	develop_type	update_type	developer	sub_developer	idtype
A121H013_002	积分试用会员	用户属性	13	是否付费会员(CO	(Null)		1	2	日	AAA	BBB	cookieid
A121H013_003	赠送会员	用户属性	13	是否付费会员(CO	(Null)		1	2	日	AAA	BBB	cookieid
A121H013_004	补偿会员	用户属性	13	是否付费会员(CO	(Null)		1	2	日	AAA	BBB	cookieid
A121H013_005	历史会员	用户属性	13	是否付费会员(CO	(Null)		1	2	日	AAA	BBB	cookieid
A121H013_006	非会员	用户属性	13	是否付费会员(CO	(Null)		1	2	日	AAA	BBB	cookieid
A121H030_001	未注册	用户属性	10030	注册状态	(Null)		1	2	日	AAA	BBB	cookieid
A121H030_002	已注册	用户属性	10030	注册状态	(Null)		1	2	日	AAA	BBB	cookieid
B120H008_003	首单新人价商品	用户行为	8	首单营销方式(CO	(Null)		1	2	日	AAA	BBB	cookieid
B120H008_004	首单优惠券	用户行为	8	首单营销方式(CO	(Null)		1	2	日	AAA	BBB	cookieid
B120H008_005	首单新人专享优惠商	用户行为	8	首单营销方式(CO	(Null)		1	2	日	AAA	BBB	cookieid
B120H008_006	首单红包	用户行为	8	首单营销方式(CO	(Null)		1	2	日	AAA	BBB	cookieid
B120U008_001	首单正常购买	用户行为	2008	首单营销方式	(Null)		1	2	日	AAA	BBB	userid
B120U008_002	首单免费礼物	用户行为	2008	首单营销方式	(Null)		1	2	日	AAA	BBB	userid
B120U008_003	首单新人价商品	用户行为	2008	首单营销方式	(Null)		1	2	日	AAA	BBB	userid
B120U008_004	首单优惠券	用户行为	2008	首单营销方式	(Null)		1	2	日	AAA	BBB	userid

标签的元数据维护着标签的id、名称、主题、一级二级分类、标签描述等信息

结果集校验

Hive作业完成后，每个标签量级/覆盖率的监控

id	tagid	date	tag_count	tag_total_per	tag_dau_per	t
86	A121H002_003	2018-04-21	581391334	0.0193901	0.030789	
87	A111H001_002	2018-04-21	508867844	0.169713	0.723889	
89	A121H002_002	2018-04-21	1460253600	0.487011	0.657123	
90	B220H026_001	2018-04-21	283956377	0.947052	1.00701	
91	A121H031_002	2018-04-21	25371635	0.846173	0.681642	
92	A121H030_002	2018-04-21	996368533	0.3323	0.51521	
93	A121H030_001	2018-04-21	184326924	0.614752	0.491804	
94	A111H041_002	2018-04-21	315144932	0.105104	0.129659	
95	A111H041_001	2018-04-21	319472833	0.106548	0.0742948	
96	A220H029_001	2018-04-21	28396377	0.947052	1.00701	
97	A121H002_009	2018-04-21	238660590	0.079596	0.0429698	

当日该标签覆盖的用户量

当日该标签覆盖的用户占当日活跃用户的比例

当日该标签与昨日相比的波动比例

Hive同步到hbase后，数据校验

date	service_type	hive_count	hbase_count
2018-09-09	addresstag	[illegible]	[illegible]
2018-09-08	userprofile	[illegible]	[illegible]
2018-09-08	[illegible]	[illegible]	[illegible]
2018-09-08	addresstag	[illegible]	[illegible]
2018-09-07	userprofile	[illegible]	[illegible]
2018-09-07	usergroup2hba[illegible]r	[illegible]	[illegible]
2018-09-07	usergroup2hba[illegible]r	[illegible]	[illegible]
2018-09-07	E[illegible]	[illegible]	[illegible]

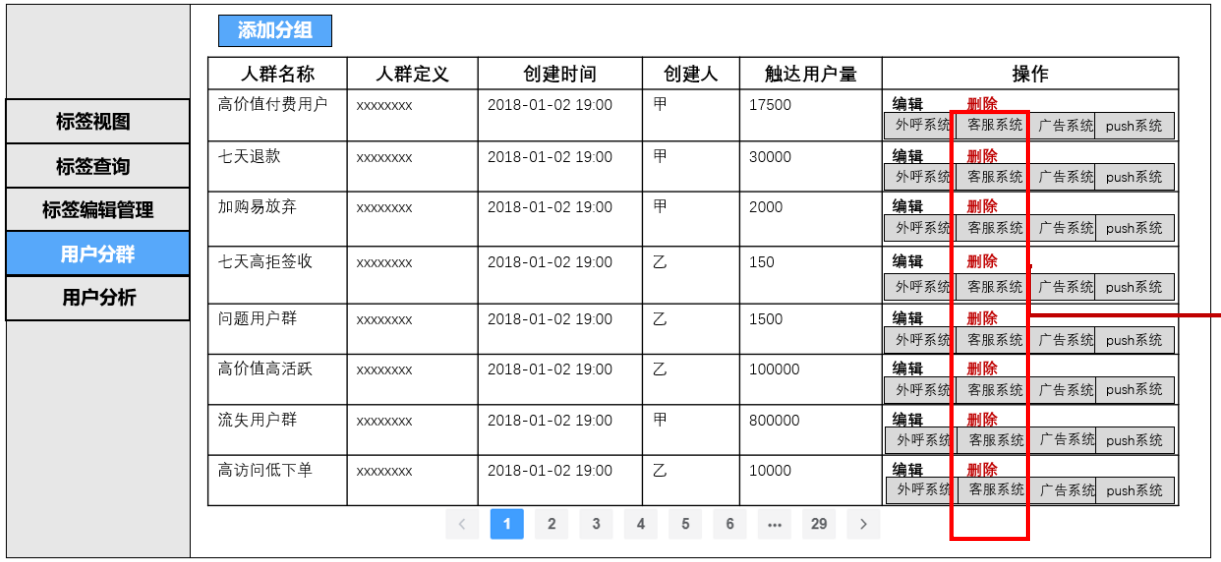
存放数据校验标志位

process_date	stage	state	is_online
2018-08-17	Cluster	0	2
2018-08-18	Cluster	1	2
2018-08-19	Cluster	0	2
2018-08-20	Cluster	0	2
2018-08-21	Cluster	0	2
2018-08-22	Cluster	0	2
2018-08-23	Cluster	0	2
2018-08-24	Cluster	0	2
2018-08-25	Cluster	0	2

放置标志位用于判断某些任务是否需要继续执行

天善智能 TIANSHAN SOFT

同步到业务系统中



在用户画像产品化章节中，当运营人员圈定用户后，需要将该批待运营的用户推送到对应的业务系统中；

不同的业务系统读取的数据库不全都一样，比如说“客服系统”中读取的数据库是关系型数据库

这里，通过sqoop把hive中的数据同步到对应的MySQL库表中

```
os.system("sqoop export --connect jdbc:mysql://xxx.xx.23.142:3307/userprofile --username userprofile
--password userprofile --table tag_tmp_userid --export-dir
hdfs://master:9000/user/hive/warehouse/dw.db/dw_profile_user_tag_service/data_date=20180901/business=push
--input-fields-terminated-by '\001'")
```

同步后的存储结果

cookie_user_id	tagsmap	reserve	reserve1
10005353	{'A121U013_0_006': '', 'B121U031_0_002': '10', 'A120U014_0_001': '3'}		
10005433	{'B121U031_0_001': '1', 'A121U013_0_006': '', 'A120U014_0_001': '2'}		
10005439	{'A121U013_0_005': '', 'B121U031_0_002': '6', 'A120U014_0_001': '2'}		
10005447	{'A121U013_0_006': '', 'B121U031_0_002': '4', 'A120U015_0_001': '0.4', 'A120U014_0_001':		
10005493	{'B121U031_0_002': '2', 'A120U014_0_001': '2', 'A121U013_0_006': ''}		
10005499	{'B121U031_0_002': '3', 'A120U014_0_001': '3', 'A121U013_0_006': ''}		
10005671	{'A121U013_0_006': '', 'B121U031_0_002': '1', 'A120U014_0_001': '2'}		
10005707	{'A121U013_0_006': '', 'B121U031_0_002': '1', 'A120U014_0_001': '2'}		
10005911	{'A121U013_0_006': '', 'B121U031_0_002': '1', 'A120U014_0_001': '2'}		
10005941	{'A121U013_0_005': '', 'B121U031_0_002': '2', 'A120U014_0_001': '2'}		
10005955	{'A121U013_0_004': '', 'B121U031_0_002': '6', 'A120U014_0_001': '2'}		
10005983	{'A121U013_0_004': '', 'B121U031_0_002': '10', 'A120U014_0_001': '3'}		
10006011	{'B121U031_0_001': '1', 'A121U013_0_005': '', 'A120U015_0_001': '0.5555555555555556'}		

这里可以写一个Python脚本，把对应的hive数据同步到MySQL库表下面

Hbase的存储及应用

同步Hive数据到Hbase

```
hive --auxpath
$HIVE_HOME/lib/zookeeper-3.4.6.jar,$HIVE_HOME/lib/hive-hbase-handler-2.3.3.jar,$HIVE_HOME/lib/hbase-ser
ver-1.1.1.jar --hiveconf hbase.zookeeper.quorum=master,node-1,node-2
```

启动hive

```
CREATE TABLE dw.userprofile_hive_2_hbase
(key string,
value string)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:val")
TBLPROPERTIES ("hbase.table.name" = "userprofile_hbase");
```

创建一张映射到hbase的hive表

```
INSERT OVERWRITE TABLE dw.userprofile_hive_2_hbase
SELECT userid,tagid FROM dw.profile_tag_userid;
```

向该映射表插入测试数据

执行过程

创建映射表

```
hive> CREATE TABLE dw.userprofile_hive_2_hbase
    > (key string,
    > value string)
    > STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
    > WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:val")
    > TBLPROPERTIES ("hbase.table.name" = "userprofile_hbase");
OK
Time taken: 5.058 seconds
```

插入测试数据

```
hive> INSERT OVERWRITE TABLE dw.userprofile_hive_2_hbase
    > SELECT userid,tagsmap FROM dw.profile_user_map_userid;
FAILED: NoMatchingMethodException No matching method for class org.apache.hadoop.hive.ql.u
_(binary)  _FUNC_(boolean)  _FUNC_(date)  _FUNC_(decimal(38,18))  _FUNC_(double)  _FUNC_(
invint)  FUNC (void)
hive> INSERT OVERWRITE TABLE dw.userprofile_hive_2_hbase
    > SELECT userid,tagid FROM dw.profile_tag_userid;
WARNING: Hive on MR is deprecated in Hive 2 and may not be available in the future versio
X releases.
Query ID = root_20180924110907_9dfcb673-f9e7-4b39-8a36-9e62f0135788
Total jobs = 1
```

执行MapReduce作业

```
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1536058452188_0018, Tracking URL = http://master:8088/proxy/applicatio
Kill Command = /root/data/hadoop-2.7.5/bin/hadoop job  -kill job_1536058452188_0018
Hadoop job information for Stage-3: number of mappers: 2; number of reducers: 0
2018-09-24 11:09:35,434 Stage-3 map = 0%,  reduce = 0%
2018-09-24 11:09:54,779 Stage-3 map = 50%,  reduce = 0%, Cumulative CPU 4.63 sec
2018-09-24 11:09:57,286 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 10.4 sec
```

查询这张hbase表

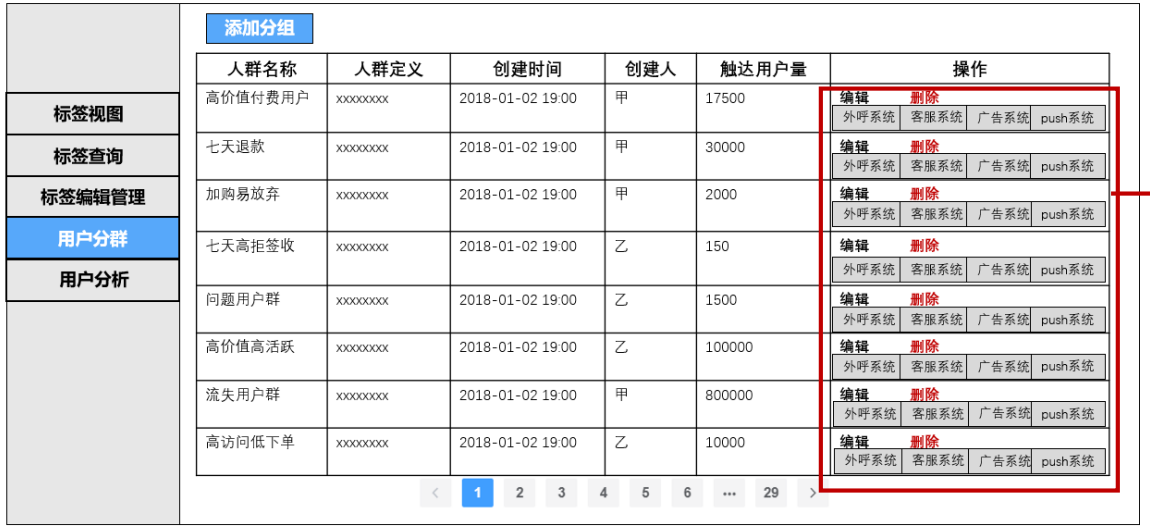
```
hbase(main):006:0> list
TABLE
userprofile_hbase
1 row(s) in 0.0370 seconds

=> ["userprofile_hbase"]
hbase(main):007:0> scan 'userprofile_hbase'
ROW                                    COLUMN+CELL
 16254215                              column=cf1:val, timestamp=1537758596270, value=A220U029_001
 25083679                              column=cf1:val, timestamp=1537758593401, value=A220U083_001
 30765587                              column=cf1:val, timestamp=1537758593401, value=A220U083_001
 32171777                              column=cf1:val, timestamp=1537758593401, value=A220U083_001
 39730187                              column=cf1:val, timestamp=1537758596270, value=A220U029_001
 40382657                              column=cf1:val, timestamp=1537758593401, value=A220U083_001
 4212236                               column=cf1:val, timestamp=1537758593401, value=A220U029_001
 7306783                               column=cf1:val, timestamp=1537758596270, value=A220U083_001
8 row(s) in 0.2590 seconds
```

天善智能 TIANSHAN SOFT

用户标签同步到Hbase在工程上应用场景

在画像产品化章节中有讲到圈人服务，业务方根据规则圈定人群后进一步通过分析明确该人群是其要运营的人群后，将该人群推送到相应的业务系统中



选择将用户推送到哪个业务系统，有的业务系统使用hbase提供服务。如广告系统、push消息系统等

工程上调度流程

运营人员根据规则圈出的人群及对应的标签集存储到MySQL → Spark作业读取MySQL标签集信息，计算对应的人群 → 将计算后得到人群存储到对应的分区表 → 每天/实时将该分区表数据同步到hbase

这个调度流程可以做成离线模式（走ETL，做T+1模式），也可以是实时模式，实时计算出对应用户群，推到hbase

工程开发上注意

因为灌入到hbase中的数据一般直接应用到线上，反馈到用户那里。所以在hive数据同步hbase数据的时候，需要做一些校验机制来保障结果的准确性，防止在同步数据的过程中出现问题（比如hive中数据5000万条，同步到hbase后才1000万条）

在开发过程中，可以尝试两种解决方案：

1、hive到hbase同步数据后，现在hbase中建立一个temp临时表，然后校验hbase的这个临时表和对应hive表的数量差异，如果在可接受范围内，则将hbase的该临时表进行重命名为正式表；

2、hive到hbase同步数据后，直接将数据写入正式表，同时在hbase中建立一张状态表，用于标志状态位。当校验hbase的这张正式表和hive的数量差异在可接受范围内时，写入对应的状态表中。接口请求时，只读取状态位这张表中，最近日期的那张表（所以如果hbase的数据同步异常，不会写入状态表中，也不会影响线上数据的读取；）

为什么用不同数据库存储标签数据

不同数据库存储的数据

当然不同的公司有不同的技术选型，也不定都用下面介绍的这三种数据库存储，或者也有用mongodb、redis等存储相关标签数据的。

这里只分享一种在工程上可行的实现方式，在遇到同类问题时可以提供相应的解决思路

Hive：跑spark作业或MapReduce作业，处理大量的数据集时使用；

MySQL：存储一些数量级较少的标签。MySQL的读写不用跑mapreduce作业，对于小量的数据读写速度很快。用于存储元数据、标签量级的监控、一些表加工结果的状态位、业务系统中读取的一些数据；

Hbase：存储线上推荐给用户的实时性较强的数据