用户画像手册

| 方法 | 案例 | 实践

从用户标签指标体系设计->数据分析->数据开发->ETL->打通服务层->画像产品化->数据赋能->业务提升，打通全流程，提供端到端的解决方案

这是一本关于用户画像解决方案的 PDF 电子书

用户画像的核心在于分析、了解用户，使数据的价值最终走出数据仓库，应用到业务系统和营销系统中，驱动营收增长。本手册讲解了如何从用户数据收集、到建模开发、到产品化、再到业务系统，驱动业务增长。这版先更新 V1.1 版本

如果看完本份手册后，对用户画像解决方案细节感兴趣小伙伴可移步这里来深入研究 ：）

目录

1. 画像简介

1 表结构设计

表结构设计的重点是要考虑存储哪些信息、如何存储（数据分区）、如何应用（如何抽取标签）这三方面问题。

不同业务背景有不同的设计方式,这里提供两种设计思路:① 每日全量数据的表结构;② 每日增量数据的表结构。

Hive 需要对输入进行全盘扫描来满足查询条件，通过使用分区可以优化查询。对于用户标签这种日加工的数据，随着时间推移，分区数量的变动也是均匀的。

每日全量数据，即该表的日期分区中记录着截止到当天为止的全量用户数据。例如:select count(*) from userprofile where data='20180701'，这条语句查询的是 userprofile 这个表截止到 20180701 日为止全量用户数据。日全量数据的优势是方便查询，缺点是不便于探查更细粒度的用户行为。

每日增量数据，即该表的日期分区中记录着当日的用户行为数据，例如同样是 select count(*) from userprofile where data='20180701'，这条语句查询的是 userprofile 这个表在 20180701 日记录的当日用户行为数据。日增量数据可视为 ODS 层的用户行为画像，在应用时还需要基于该增量数据做进一步的建模加工。

下面介绍日全量数据表结构设计

日全量数据表中，每天对应的日期分区中插入截止到当天为止的全量数据，用户使用查询时，只需查询最近一天即可获得最新全量数据。下面以一个具体的日全量表结构例子来做说明。

```
CREATE TABLE dw.userprofile_tag_userid (
tagid STRING COMMENT 'tagid',
userid STRING COMMENT 'userid',
tagweight STRING COMMENT 'tagweight',
reserve STRING COMMENT '预留' )
PARTITIONED BY (data_date STRING COMMENT '数据日期',tagtype STRING COMMENT '标签主题分类')
```

这里 tagid 表示标签名称，userid 表示用户 id，tagweight 表示标签权重，reserve 表示一个预留字段。分区方式为（日期+标签主题）分区，设置两个分区字段更便于开发和查询数据。该表结构下的标签权重仅考虑统计类型标签的权重，如：历史购买金额标签对应的权重为金额数量，用户近 30 日访问天数为对应的天数，该权重值的计算未考虑较为复杂的用户行为次数、行为类型、行为距今时间等复杂情况。

2 标签类型

从对用户打标签的方式来看，一般分为三种类型：1、统计类的标签；2、规则类的标签、3、机器学习挖掘类的标签。下面我们介绍这三种类型标签的区别:

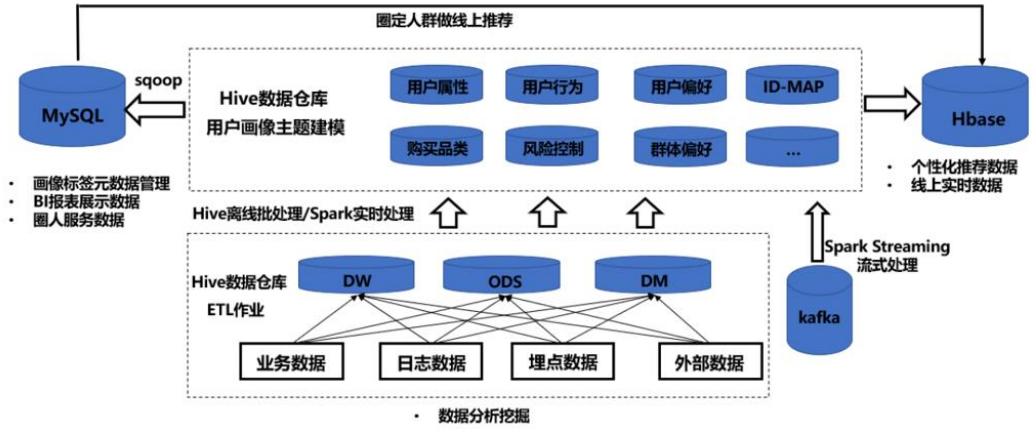
- 统计类的标签：这类标签是最为基础也最为常见的标签类型，例如对于某个用户来说，他的性别、年龄、城市、星座、近 7 日活跃时长、近 7 日活跃天数、近 7 日活跃次数等字段可以从用户注册数据、用户访问、消费类数据中统计得出。该类标签构成了用户画像的基础;
- 规则类的标签：该类标签基于用户行为及确定的规则产生。例如对平台上"消费活跃"用户这一口径的定义为近 30 天交易次数>=2。在实际开发画像的过程中，由于运营人员对业务更为熟悉、而数据人员对数据的结构、分布、特征更为熟悉，因此

规则类标签的规则确定由运营人员和数据人员共同协商确定；

- 机器学习挖掘类的标签：该类标签通过机器学习挖掘产生，应用在对用户的某些属性或某些行为进行预测判断。例如根据一个用户的行为习惯判断该用户是男性还是女性，根据一个用户的消费习惯判断其对某商品的偏好程度。该类标签需要通过算法挖掘产生。

在项目工程实践中，一般统计类和规则类的标签即可以满足应用需求，开发中占有较大比例。机器学习挖掘类标签多用于预测场景，在开发所占比例较小。

3 数据架构



2. 数据指标体系

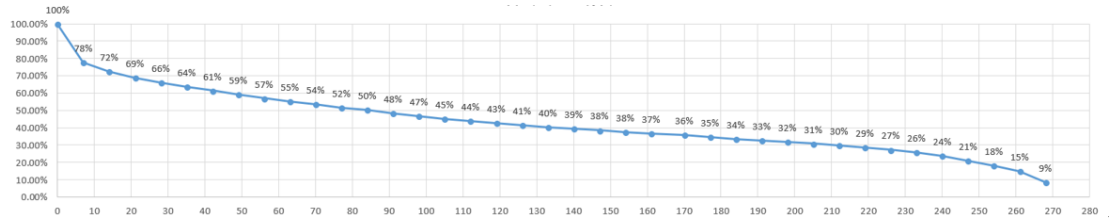
1 用户属性维度

常见用户属性指标包括：用户的年龄、性别安装时间、注册状态、城市、省份、活跃登陆地、历史购买状态、历史购买金额等。用户属性维度的标签建成后可以为客服电话服务、运营人员了解用户基本情况提供帮助

标签名称	标签主题	一级归类	标签类型	是否互斥
男	用户属性	自然性别	分类	互斥
女	用户属性	自然性别	分类	互斥
男	用户属性	购物性别	分类	互斥
女	用户属性	购物性别	分类	互斥
年龄	用户属性	年龄	统计	非互斥
省份	用户属性	地域	统计	非互斥
城市	用户属性	地域	统计	非互斥
注册日期	用户属性	注册日期	统计	非互斥
高	用户属性	手机品牌	分类	互斥
中	用户属性	手机品牌	分类	互斥
低	用户属性	手机品牌	分类	互斥
……	……	……	……	……

开发过程中也会进行数据分析和数据调研

R(最近一次交易时间)	F(历史累计订单量)	M(累计交易金额)		占比
近	高频次	高金额	重要价值用户	2.607%
远	高频次	高金额	重要保持用户	0.693%
近	低频次	高金额	重要发展用户	14.773%
远	低频次	高金额	重要挽留用户	3.927%
近	高频次	低金额	一般价值用户	9.243%
远	高频次	低金额	一般保持用户	2.457%
近	低频次	低金额	一般发展用户	52.377%
远	低频次	低金额	一般挽留用户	13.923%



2 用户行为维度

标签名称	标签主题	一级归类	标签类型	是否互斥
订单评价数	用户行为	订单评价	统计	非互斥
订单好评数	用户行为	订单评价	统计	非互斥
订单差评数	用户行为	订单评价	统计	非互斥
现金券使用次数	用户行为	现金券	统计	非互斥
现金券使用金额	用户行为	现金券	统计	非互斥
近 30 日购买金额	用户行为	近 30 日行为	统计	非互斥
近 30 日购买次数	用户行为	近 30 日行为	统计	非互斥

近 30 日加购物车次数	用户行为	近 30 日行为	统计	非互斥
近 30 日加购物车金额	用户行为	近 30 日行为	统计	非互斥
……	……	……	……	……

3 消费维度

标签名称	标签主题	一级归类	标签类型	是否互斥
鼠标	购买品类	电脑外设	统计	非互斥
键盘	购买品类	电脑外设	统计	非互斥
手写板	购买品类	电脑外设	统计	非互斥
鼠标垫	购买品类	电脑外设	统计	非互斥
摄像头	购买品类	电脑外设	统计	非互斥
休闲鞋	购买品类	男鞋	统计	非互斥
商务鞋	购买品类	男鞋	统计	非互斥
帆布鞋	购买品类	男鞋	统计	非互斥
……	……	……	……	……

4 风控维度

标签名称	标签主题	一级归类	标签类型	是否互斥
退货率	风险控制	退货	统计	非互斥
赔付率	风险控制	退货	统计	非互斥
退货异常	风险控制	退货	统计	非互斥
拒收单数	风险控制	拒收	统计	非互斥
连拒大于 2 单	风险控制	拒收	统计	非互斥
高投诉用户	风险控制	高风险用户	统计	非互斥
高咨询用户	风险控制	高风险用户	统计	非互斥
……	……	……	……	……

3. 标签存储

1 Hive 存储

Hive 存储所有标签相关数据的计算结果集。Hive 存储标签相关的数据涉及到一些表，如标签表、标签聚合表、服务层的表、人群相关的表等

例如标签表的表结构

```
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')
```

标签表中存储的数据

```
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

2 MySQL 存储

MySQL 在画像中可以用来存储标签元数据；结果集校验（标签量级监控、hive 同步到 hbase 的校验、数据波动的校验等）；同步到业务系统中的数据

① 标签元数据

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

② 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]	[illegible]	[illegible]
2018-09-07	usergroup2hba[illegible]	[illegible]	[illegible]
2018-09-07	E[illegible]	[illegible]	[illegible]

③ 同步到业务系统

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

3 hbase 存储

存储线上环境应用数据

```
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");
```

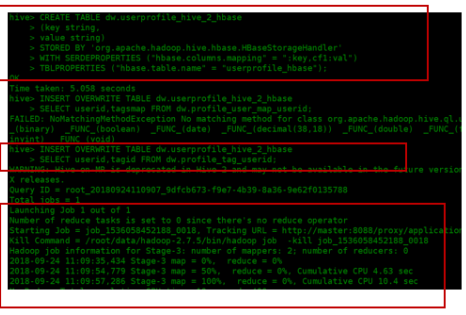
创建一张映射到hbase的hive表

```
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

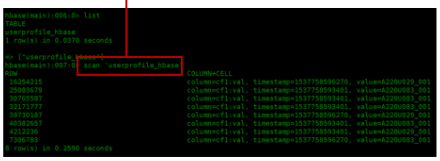
创建映射表



插入测试数据

执行MapReduce作业

查询这张hbase表



4. 标签开发

标签开发相关的工作主要涉及到：统计类标签开发、规则类标签开发、挖掘类标签开发、流式计算类数据开发、用户特征库开发、打通数据服务层

① 统计类/规则类标签开发

需要开发的近30日购买行为这一级类目下的系列标签。具体包括付款订单量（对应标签B220U001_001）、总付款金额（对应标签B220U001_002）、加入购物车次数（对应标签B220U001_003）这3个标签。

```
select 'B220U001_001' as tagid,
       cast(user_id as string) as userid,
       count(distinct order_id) as tagweight     # 付款订单量
  from dw.dw_order_fact     | # 商品订单表
 where pay_status in (1,3)     # 订单状态是已支付
       and to_date(add_time) >= "month_day_ago"    # 付款日期大于等于 30 日前
       and to_date(add_time) <= "start_date_str"   # 付款日期小于等于昨天
union all
select 'B220U001_002' as tagid,
       cast(user_id as string) as userid,
       sum(order_total_amount) as tagweight      #总付款金额
  from dw.dw_order_fact
 where pay_status in (1,3)      # 订单状态是已支付
   and to_date(add_time) >=  "month_day_ago"
   and to_date(add_time) <=  "start_date_str"
union all
select 'B220U001_003' as tagid,
       cast(userid as string) as userid,
       count(distinct eventid) as tagweight     # 加入购物车事件次数
  from ods.ods_event_log       # 行为事件表
 where data_date >= "month_day_ago"
   and data_date <= "start_date_str"
   and eventkey = 'add_to_shoppingbag'     # 行为事件名称为加入购物车
   and  userid is not null      # 用户 id 为非空值
```

首先将需要计算的标签从目标表中抽取出来。

左边段代码中保证每天作业将近30日的标签打在每个有相应行为的用户身上，但要做到增量取用户最新状态，还需要做一层全连接关联。

举个常见的例子来说，某个用户前天购买了3单商品，如果昨天又购买了2单，则今天最新的状态是3+2=5单，替换掉前天3单的权重值，如果昨天没有购买行为，则权重值仍为3。这里通过full outer join全连接的方式当用户有最新状态时，取最新状态，如果没有最新状态则仍保留原来状态的标签。

② 流式计算类数据开发

```
// messages 从kafka获取数据,将数据转为RDD
messages.foreachRDD((rdd, batchTime) => {
  import org.apache.spark.streaming.kafka.HasOffsetRanges
  val offsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges   // 获取偏移量信息
  /**
    * OffsetRange 是对topic name，partition id，fromOffset(当前消费的开始偏移)，untilOffset(当前消费的结束偏移)的封装，
    * *   所以OffsetRange 包含信息有：topic名字，分区Id，开始偏移，结束偏移
    */
  println("==========================> count: " + rdd.map(x => x + "1").count())
 // offsetRanges.foreach(offset => println(offset.topic, offset.partition, offset.fromOffset, offset.untilOffset))
  for (offset <- offsetRanges) {
    // 遍历offsetRanges,里面有多个partition
    println(offset.topic, offset.partition, offset.fromOffset, offset.untilOffset)
    DBs.setupAll()
    // 将partition及对应的untilOffset存到MySQL中
    val saveoffset = DB localTx {
      implicit session =>
       sql"DELETE FROM offsetinfo WHERE topic = ${offset.topic} AND partitionname = ${offset.partition}".update.apply()
       sql"INSERT INTO offsetinfo (topic, partitionname, untilOffset) VALUES (${offset.topic},${offset.partition},${offse
    }
  }
})
```

④ 用户特征库开发

	用户行为日志
cookieid	
goodsid	
siteid	除400、700站
data_date	日分区
visit_cnt	访问次数
cart_cnt	加购次数
fav_cnt	收藏次数
share_cnt	分享次数
checkout_cnt	checkout点击次数
pay_type_cnt	支付类型选择次数
signed_cnt	签到点击次数
reg_cnt	注册次数
trytopay_cnt	支付点击次数
visit_duration	访问时长
paid_cnt	购买次数
order_cnt	下单次数
goods_price	商品单价
goods_impression_cnt	商品曝光次数
goods_click_cnt	商品点击次数
gd_reviews_click_cnt	详情页评论点击次数
goodsdetail_reviews_more_click	详情页评论点击more次数
gd_details_click_cnt	详情页detail点击次数
gd_select_click_cnt	详情页select点击次数
gd_share_click_cnt	详情页分享点击次数
gd_shopgoods_click_cnt	详情页被推荐点击次数
cart_click_cnt	加购点击次数
cart_submit_click_cnt	加购submit点击次数
fav_recommend_cnt	收藏下面推荐商品次数
fav_main_cnt	收藏主商品次数
sizeguide_click_cnt	尺码参考点击次数
gd_shop_tips_click_cnt	详情页购买tips点击次数
gd_brand_click_cnt	详情页品牌点击次数
gd_brand_recommend_cnt	详情页品牌推荐点击次数
gd_coupon_click_cnt	详情页优惠券入口点击次数
gd_coupon_cnt	详情页优惠券领取成功次数

⑤ 打通服务层

在开发完画像后，还需要打通标签数据和各业务系统之间的通路，通过产品化的方式将标签数据应用到业务中去

5. 开发调优

标签开发上线后需要进行迭代，开发调优减少任务的作业时间
开发调优主要包括数据倾斜调优、合并 hive 小文件、使用 spark 缓存、开发中间表等形式

例如针对一些标签读取同样的上游数据，开发对应的中间层表

标签id	标签名称	标签汉语	标签主题	二级标签	来源表	标签类型(tag_type)	维度	作业大致时间	跑数周期	备注
B220U[illegible]_0_001	last 30d buy frequency	近30天购买次数（含退	用户行为		dw.dw_order_fact ods.ods_event_log ods.ods_page_view_log dw.dw_cookie_user_relation	purchase_latest_30_days	userid	7分半	30天	需要对ods.ods_page_view_log ods.ods_enent_log做中间层
B[illegible]001	last 30d buy amount	近30天购买金额（含退拒）	用户行为							
B[illegible]001	last 30d cart frequency	近30天购物车次数	用户行为							
B[illegible]01	last 30d cart abandon num	近30天购物车放弃数	用户行为							
B[illegible]01	last 30d cart submit num	近30天购物车提交商品数	用户行为							
B[illegible]01	last 30d average order price	近30天客单价	用户行为							
B[illegible]1	last 30d active days	近30天活跃天数	用户行为							
B[illegible]1	last 7d buy frequency	近7天购买次数（含退拒）	用户行为		dw.dw_order_fact ods.ods_event_log ods.ods_page_view_log dw.dw_cookie_user_relation	purchase_latest_7_days	userid	4分半	7天	需要对ods.ods_page_view_log ods.ods_enent_log做中间层
B[illegible]1	last 7d buy amount	近7天购买金额（含退拒）	用户行为							
B[illegible]01	last 7d average order price	近7天客单价	用户行为							
B[illegible]01	last 7d cart frequency	近7天购物车次数	用户行为							
B[illegible]01	last 7d cart abandon num	近7天购物车放弃数	用户行为							
B[illegible]01	last 7d cart submit num	近7天购物车提交商品数	用户行为							
[illegible]01	last 7d active days	近7天活跃天数	用户行为							

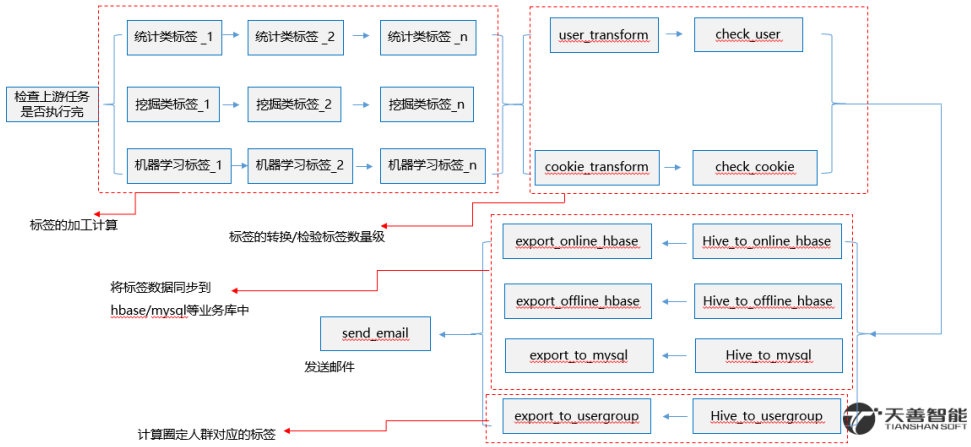
6. ETL 调度

① 调度流中涉及的任务

在用户画像工程化调度中主要涉及到的环节/模块：

- **标签调度：**主要的调度任务，负责每天调度计算用户身上的标签，插入对应的标签表中；
- **标签校验：**分为多个模块。①校验每天插入hive表中的标签数据是否出现异常；②校验同步到hbase、关系数据库中的标签数据是否和hive中标签数量级一致；③校验“圈人功能”中计算出来人群对应的标签是否出现异常波动；
- **数据同步：**将用户标签同步到hbase、MySQL等关系数据库的业务系统中。这算是数据服务层的任务；
- **人群计算：**根据产品端业务人员圈定的用户标签组合，计算对应的人群。计算任务使用MapReduce或spark作业将数据插入到hive中，然后同步到对应的业务系统中
- **通知邮件：**数据插入到hive、hbase或关系数据库后校验标签的数量级或波动情况。如超出正常范围则触发报警邮件

② 工程化调度流模块

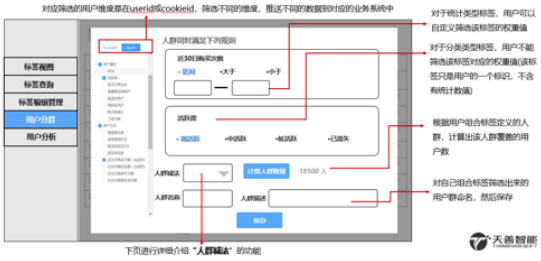


③ 同步到服务层

开发后的标签支持到数据服务层（如push系统、广告系统、外呼系统等），可以选择离线方式或实时方式。

离线方式的推送数据质量较为稳定，对于数据团队的运维压力较小，能满足大多数情况的需求；

实时方式对效率要求较高，数据团队既要保证数据准确性又要保障推送出去数据的质量，运维压力较大。本节主要讲下离线支持到服务层的方式



运营人员在产品端根据标签圈定人群，选择对应的推送系统。

这步操作对应的标签及人群记录存在MySQL中

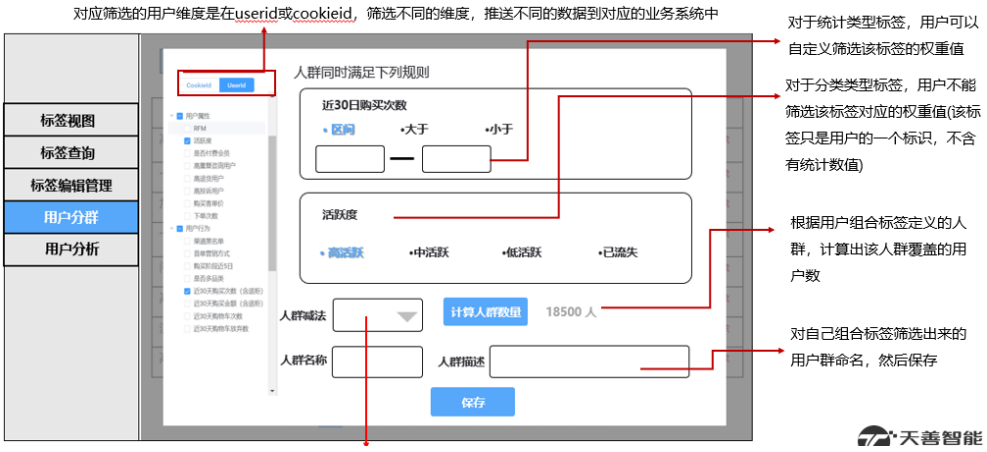
离线调度流每日ETL调度时定时解析MySQL中记录的人群标签规则，跑作业，计算每个业务规则下人群的userid或cookieid，推送到对应的业务系统中

7. 画像产品化

产品化中主要包括标签视图与查询、标签编辑管理、用户人群圈定与推送、多维透视分析等多个模块

标签查询

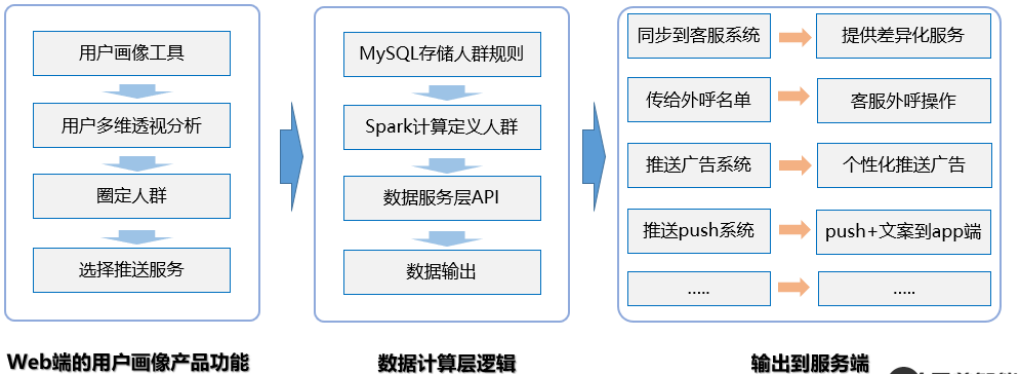
用户分群



数据计算逻辑

自定义人群提供根据现有用户标签设置圈定用户群体的功能，业务人员可利用“***多维透视分析***”功能进行人群的对比分析，通过预计算对该运营规则圈定的人群数量做测算。保存后将生成圈定人群的规则，而后依据该规则生成圈定人群推送到服务端

下面介绍提供产品化服务的调度流程



8. 用户画像应用

用户画像应用范围较广，包括对用户的多维透视分析、圈定用户推送 push、广告位分群展示、短信/邮件营销、客服 vip 服务、外呼召回用户等场景

根据用户标签推送的 AB 组效果分析

发送时间	发送主题	人群	发送量	抵达量	抵达率	总优惠码收入	roi
20180519	[redacted]会场	历史购买过细分类目 -黑名单AB测试 A黑名单数据	237232	129140	54.44%	15411.29	4.77
20180519	[redacted]S会场	历史购买过细分类目 -黑名单AB测试 B非黑名单数据	654940	597481	91.23%	146159.57	16.39
20180523	[redacted] 全场80%off	[redacted]万史下单用户 黑名单	491437	280925	57.16%	65819.78	9.84
20180523	[redacted] 全场80%off	[redacted]历史下单用户 非黑名单	987918	951564	96.32%	446098.36	33.17
20180527	爆发 80%off最后机会	[redacted]历史下单用户 黑名单	505391	289506	57.28%	23039.81	3.35
20180527	[redacted]爆发 80% off 免邮	[redacted]历史下单用户 白名单	990306	954418	96.38%	181139.12	13.44