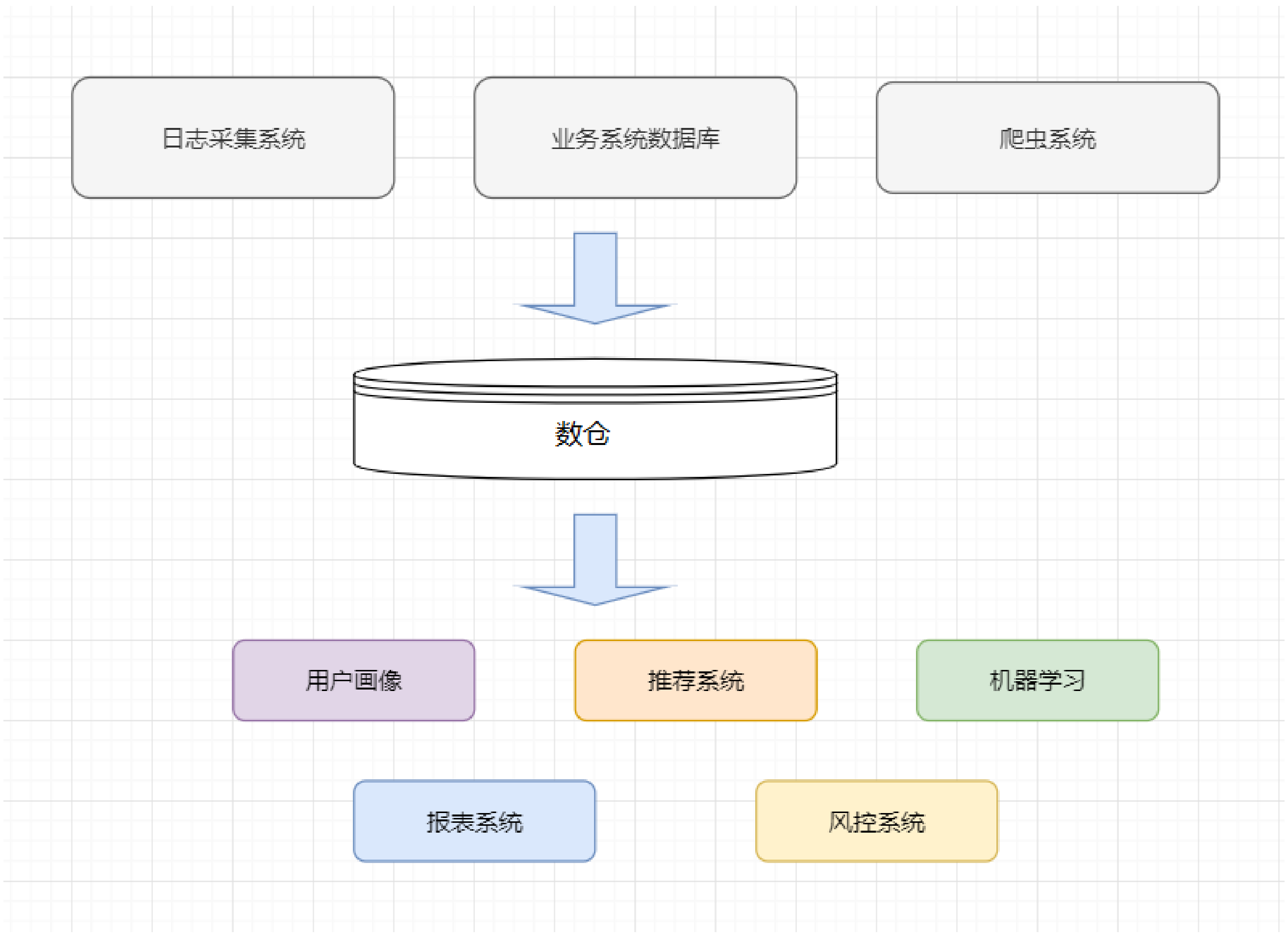
一、数仓的数据准备与采集

数据仓库概念

> 数据仓库 （ Data Warehouse ）， 是为企业所有决策制定过程，提供所有系统数据支持的战略集合。 通过对数据仓库中数据的分析，可以帮助企业，改进业务流程、控制成本、提高产品质量等。

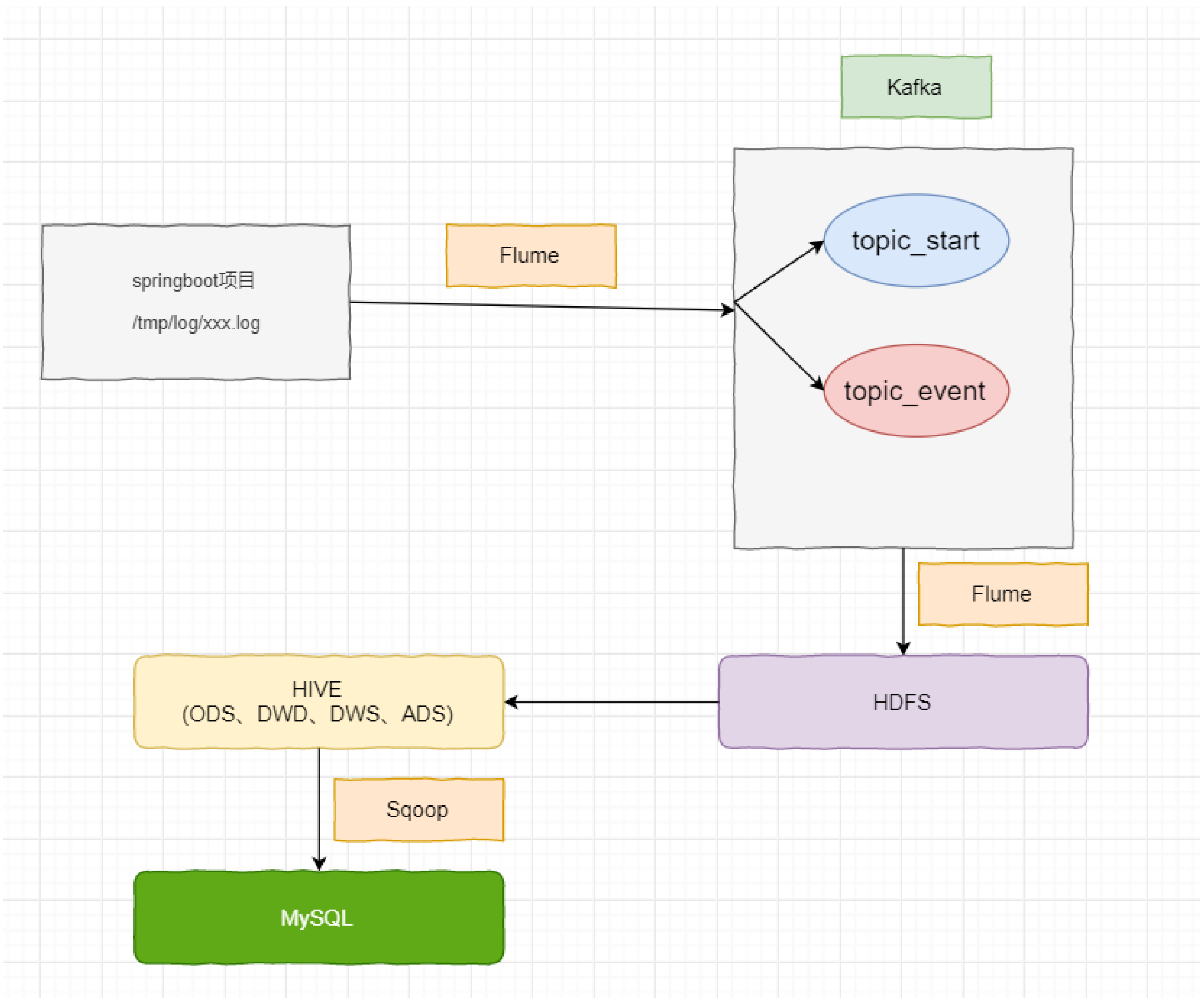
> 数据仓库，并不是数据的最终目的地，而是为数据最终的目的地做好准备。这些准备包括对数据的：清洗，转义，分类，重组，合并，拆分，统计等等。



项目需求以及架构设计

```
一、项目需求
    1.数据采集平台搭建
    2.实现用户行为数据仓库的分层搭建
    3.实现业务数据仓库的分层搭建
    4.针对数据仓库中的数据进行，留存、转换率、GMV、复购率、活跃等报表分析
```

系统数据流程设计以及技术选型



数据生成模块

启动日志数据

标签	含义
entry	入口： push=1， widget=2， icon=3， notification=4, lockscreen_widget =5
open_ad_type	开屏广告类型: 开屏原生广告=1, 开屏插屏广告=2
action	状态：成功=1 失败=2
loading_time	加载时长：计算下拉开始到接口返回数据的时间，（开始加载报0，加载成功或加载失败才上报时间）
detail	失败码（没有则上报空）
extend1	失败的message（没有则上报空）
en	日志类型start

```
{
    "action":"1",
    "ar":"MX",
    "ba":"HTC",
    "detail":"",
    "en":"start",
    "entry":"2",
    "extend1":"",
    "g":"43R2SEQX@gmail.com",
    "hw":"640*960",
    "l":"en",
    "la":"20.4",
    "ln":"-99.3",
    "loading_time":"2",
    "md":"HTC-2",
    "mid":"995",
    "nw":"4G",
    "open_ad_type":"2",
    "os":"8.1.2",
    "sr":"B",
    "sv":"V2.0.6",
    "t":"1561472502444",
    "uid":"995",
    "vc":"10",
    "vn":"1.3.4"
}
```

事件日志数据

埋点数据基本格式

- 公共字段：基本所有安卓手机都包含的字段
- 业务字段：埋点上报的字段，有具体的业务类型

```
1540934156385|{
    "ap": "app",   //项目数据来源 app pc
    "cm": {  //公共字段
        "mid": "",  // (String) 设备唯一标识
        "uid": "",  // (String) 用户标识
        "vc": "1",  // (String) versionCode，程序版本号
        "vn": "1.0",  // (String) versionName，程序版本名
        "l": "zh",  // (String) language系统语言
        "sr": "",  // (String) 渠道号，应用从哪个渠道来的。
        "os": "7.1.1",  // (String) Android系统版本
        "ar": "CN",  // (String) area区域
        "md": "BBB100-1",  // (String) model手机型号
        "ba": "blackberry",  // (String) brand手机品牌
        "sv": "V2.2.1",  // (String) sdkVersion
        "g": "",  // (String) gmail

        "hw": "1620x1080",  // (String) heightXwidth，屏幕宽高
        "t": "1506047606608",  // (String) 客户端日志产生时的时间
        "nw": "WIFI",  // (String) 网络模式
        "ln": 0,  // (double) lng经度
        "la": 0  // (double) lat 纬度
    },
        "et": [
            {
                "ett": "1506047605364",  //客户端事件产生时间
                "en": "display",  //事件名称
                "kv": {  //事件结果，以key-value形式自行定义
                    "goodsid": "236",
                    "action": "1",
                    "extend1": "1",
                    "place": "2",
                    "category": "75"
                }
            },{
                "ett": "1552352626835",
                "en": "active_background",
                "kv": {
                     "active_source": "1"
                }
            }
        ]
    }
}
```

下面是各个埋点日志格式

商品列表页

事件名称：loading

标签	**含义**
action	动作：开始加载=1，加载成功=2，加载失败=3
loading_time	**加载时长：计算下拉开始到接口返回数据的时间，（开始加载报0，加载成功或加载失败才上报时间）**
loading_way	加载类型：1-读取缓存，2-从接口拉新数据 （加载成功才上报加载类型）
extend1	**扩展字段 Extend1**
extend2	扩展字段 Extend2
type	**加载类型：自动加载=1，用户下拽加载=2，底部加载=3（底部条触发点击底部提示条/点击返回顶部加载）**
type1	加载失败码：把加载失败状态码报回来（报空为加载成功，没有失败）

商品点击

事件标签：display

标签	含义
action	动作：曝光商品=1，点击商品=2，
goodsid	商品ID（服务端下发的ID）
place	顺序（第几条商品，第一条为0，第二条为1，如此类推）
extend1	曝光类型：1 - 首次曝光 2-重复曝光
category	分类ID（服务端定义的分类ID）

商品详情页

事件标签：newsdetail

标签	含义
entry	页面入口来源：应用首页=1、push=2、详情页相关推荐=3
action	动作：开始加载=1，加载成功=2（pv），加载失败=3, 退出页面=4
goodsid	商品ID（服务端下发的ID）
show_style	商品样式：0、无图、1、一张大图、2、两张图、3、三张小图、4、一张小图、5、一张大图两张小图
news_staytime	页面停留时长：从商品开始加载时开始计算，到用户关闭页面所用的时间。若中途用跳转到其它页面了，则暂停计时，待回到详情页时恢复计时。或中途划出的时间超过10分钟，则本次计时作废，不上报本次数据。如未加载成功退出，则报空。
loading_time	加载时长：计算页面开始加载到接口返回数据的时间 （开始加载报0，加载成功或加载失败才上报时间）
type1	加载失败码：把加载失败状态码报回来（报空为加载成功，没有失败）
category	分类ID（服务端定义的分类ID）

广告

事件名称：ad

标签	含义
entry	入口：商品列表页=1 应用首页=2 商品详情页=3
action	动作：请求广告=1 取缓存广告=2 广告位展示=3 广告展示=4 广告点击=5
content	状态：成功=1 失败=2
detail	失败码（没有则上报空）
source	广告来源:admob=1 facebook=2 ADX（百度）=3 VK（俄罗斯）=4
behavior	用户行为： 主动获取广告=1 被动获取广告=2
newstype	Type: 1- 图文 2-图集 3-段子 4-GIF 5-视频 6-调查 7-纯文 8-视频+图文 9-GIF+图文 0-其他
show_style	内容样式：无图(纯文字)=6 一张大图=1 三站小图+文=4 一张小图=2 一张大图两张小图+文=3 图集+文 = 5 一张大图+文=11 GIF大图+文=12 视频(大图)+文 = 13 来源于详情页相关推荐的商品，上报样式都为0（因为都是左文右图）

消息通知

事件标签：notification

标签	含义
action	动作：通知产生=1，通知弹出=2，通知点击=3，常驻通知展示（不重复上报，一天之内只报一次）=4
type	通知id：预警通知=1，天气预报（早=2，晚=3），常驻=4
ap_time	客户端弹出时间
content	备用字段

用户前台活跃

事件标签: active_foreground

标签	含义
push_id	推送的消息的id，如果不是从推送消息打开，传空
access	1.push 2.icon 3.其他

用户后台活跃

事件标签: active_background

标签	含义
active_source	1=upgrade,2=download(下载),3=plugin_upgrade

评论

事件标签：comment

标签	含义
comment_id	评论表
userid	用户id
p_comment_id	父级评论id(为0则是一级评论,不为0则是回复)
content	评论内容
addtime	创建时间
other_id	评论的相关id
praise_count	点赞数量
reply_count	回复数量

标签 含义 push_id 推送的消息的id，如果不是从推送消息打开，传空 access 1.push 2.icon 3.其他

标签 含义 active_source 1=upgrade,2=download(下载),3=plugin_upgrade

收藏

事件标签：favorites

标签	含义
id	主键
course_id	商品id
userid	用户ID
add_time	创建时间

点赞

事件标签：praise

标签	含义
id	主键id
userid	用户id
target_id	点赞的对象id
type	点赞类型 1问答点赞 2问答评论点赞 3 文章点赞数4 评论点赞
add_time	添加时间

错误日志

事件标签：error

标签	含义
errorBrief	错误摘要
errorDetail	错误详情

搭建日志数据生成模块

参考：log-collector项目

```
1.项目生成对应的jar包，上传linux服务器
2.通过java -jar xx.jar命令运行jar包
3.查看/tmp/logs下的日志文件
```

Flume采集日志数据到Kafka

1.编写flume脚本，将数据采集到kafka

```
#a1是agent的名称，a1中定义了一个叫r1的source，如果有多个，使用空格间隔
a1.sources = r1

a1.channels = c1 c2
```

```
#组名名.属性名=属性值
a1.sources.r1.type=TAILDIR
a1.sources.r1.filegroups=f1
#读取/tmp/logs/app-yyyy-mm-dd.log ^代表以xxx开头$代表以什么结尾 .代表匹配任意字符
#+代表匹配任意位置
a1.sources.r1.filegroups.f1=/tmp/logs/^app.+.log$

#定义拦截器
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type = com.baizhi.interceptor.MyInterceptor$Builder

#定义ChannelSelector
a1.sources.r1.selector.type = multiplexing
a1.sources.r1.selector.header = topic
a1.sources.r1.selector.mapping.topic_start = c1
a1.sources.r1.selector.mapping.topic_event = c2


#定义chanel
a1.channels.c1.type=org.apache.flume.channel.kafka.KafkaChannel
a1.channels.c1.kafka.bootstrap.servers=hadoop10:9092
a1.channels.c1.kafka.topic=topic_start
a1.channels.c1.parseAsFlumeEvent=false

a1.channels.c2.type=org.apache.flume.channel.kafka.KafkaChannel
a1.channels.c2.kafka.bootstrap.servers=hadoop10:9092
a1.channels.c2.kafka.topic=topic_event
a1.channels.c2.parseAsFlumeEvent=false

#连接组件 同一个source可以对接多个channel，一个sink只能从一个channel拿数据!
a1.sources.r1.channels=c1 c2
```

2.在kafka中创建两个topic

```
kafka-topics.sh --zookeeper hadoop10:2181 --create --topic topic_start --replication-factor 1 --
partitions 1
kafka-topics.sh --zookeeper hadoop10:2181 --create --topic topic_event --replication-factor 1 --
partitions 1
```

3.自定义flume的拦截器

```
package com.baizhi.interceptor;

import org.apache.commons.lang.StringUtils;
import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.interceptor.Interceptor;
```

```java
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.List;

public class MyInterceptor implements Interceptor {

    public void initialize() {

    }


    public Event intercept(Event event) {
        byte[] body = event.getBody();
        String content = new String(body, Charset.forName("UTF-8"));

        /**
         * 判断字符串是否为null
         */
        if(StringUtils.isBlank(content)){
            return null;
        }
        boolean flag = true;
        if(content.contains("\"en\":\"start\"")){
            flag = ETLUtil.validStartLog(content);
            event.getHeaders().put("topic","topic_start");
        }else {
            flag = ETLUtil.validEventLog(content);
            event.getHeaders().put("topic","topic_event");
        }
        if(!flag){
            return null;
        }
        return event;
    }

    private List<Event> results = new ArrayList<>();

    public List<Event> intercept(List<Event> list) {
        results.clear();
        for (Event event : list) {
            Event result = intercept(event);
            if(result != null){
                results.add(result);
            }
        }
        return results;
    }

    public void close() {

    }
    public static class Builder implements Interceptor.Builder{

        public Interceptor build() {
            return new MyInterceptor();
        }

        public void configure(Context context) {

        }
    }

}
```

```java
package com.baizhi.interceptor;

import org.apache.commons.lang.StringUtils;
import org.apache.commons.lang.math.NumberUtils;

public class ETLUtil {

    public static boolean validStartLog(String content){
        String trimStr = content.trim();
        if(trimStr.startsWith("{")  && trimStr.endsWith("}") ){
            return true;
        }else {
            return false;
        }
    }

    public static boolean validEventLog(String content){
        String trimStr = content.trim();

        String[] str = trimStr.split("\\|");
        if(str.length != 2){
            return false;
        }

        //判断时间戳 长度  纯数字
        if(str[0].length() != 13 || !NumberUtils.isDigits(str[0])){
            return false;
        }

        if(str[1].startsWith("{") && str[1].endsWith("}")){
            return true;
        }

        return false;
    }

}
```

Flume采集kafka日志数据到HDFS

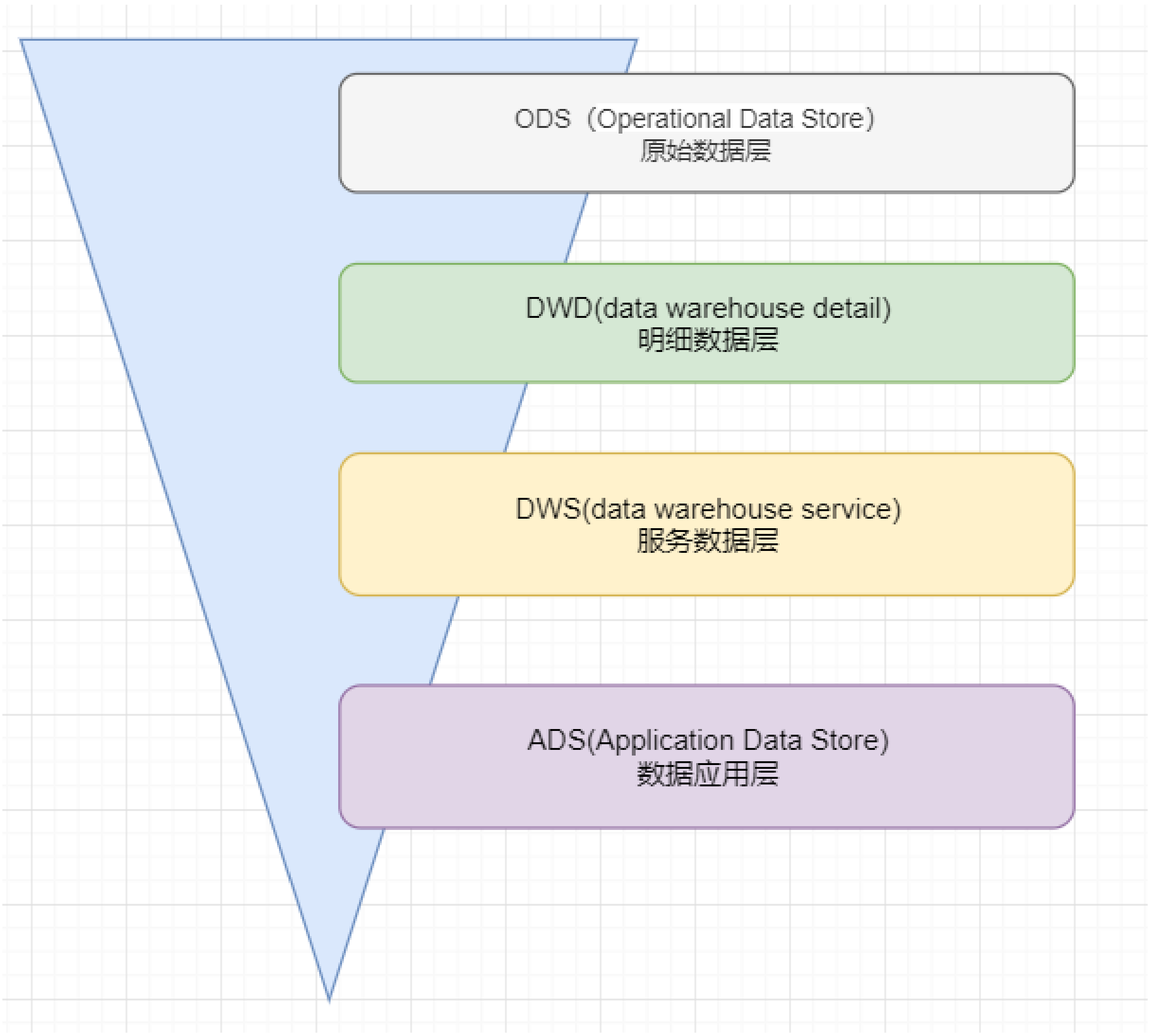
```
#配置文件编写
a1.sources = r1 r2
a1.sinks = k1 k2
a1.channels = c1 c2

#配置source
a1.sources.r1.type=org.apache.flume.source.kafka.KafkaSource
a1.sources.r1.kafka.bootstrap.servers=hadoop10:9092
a1.sources.r1.kafka.topics=topic_start
a1.sources.r1.kafka.consumer.auto.offset.reset=earliest
a1.sources.r1.kafka.consumer.group.id=CG_Start

a1.sources.r2.type=org.apache.flume.source.kafka.KafkaSource
a1.sources.r2.kafka.bootstrap.servers=hadoop10:9092
a1.sources.r2.kafka.topics=topic_event
a1.sources.r2.kafka.consumer.auto.offset.reset=earliest
a1.sources.r2.kafka.consumer.group.id=CG_Event

#配置channel
a1.channels.c1.type=memory
a1.channels.c1.capacity=1000
a1.channels.c1.transactionCapacity=1000

a1.channels.c2.type=memory
a1.channels.c2.capacity=1000
a1.channels.c2.transactionCapacity=1000

#sink
a1.sinks.k1.type = hdfs
#一旦路径中含有基于时间的转义序列，要求event的header中必须有timestamp=时间戳，如果没有需要将
useLocalTimeStamp = true
a1.sinks.k1.hdfs.path = hdfs://hadoop10:9000/origin_data/gmall/log/topic_start/%Y-%m-%d
a1.sinks.k1.hdfs.filePrefix = logstart-
a1.sinks.k1.hdfs.useLocalTimeStamp = true

#文件的滚动
a1.sinks.k1.hdfs.rollInterval = 0
a1.sinks.k1.hdfs.rollSize = 134217700
a1.sinks.k1.hdfs.rollCount = 0
a1.sinks.k1.hdfs.fileType = DataStream

a1.sinks.k2.type = hdfs
a1.sinks.k2.hdfs.path = hdfs://hadoop10:9000/origin_data/gmall/log/topic_event/%Y-%m-%d
a1.sinks.k2.hdfs.filePrefix = logevent-

a1.sinks.k2.hdfs.useLocalTimeStamp = true
```

```
a1.sinks.k2.hdfs.rollInterval = 0
a1.sinks.k2.hdfs.rollSize = 134217700
a1.sinks.k2.hdfs.rollCount = 0
a1.sinks.k2.hdfs.fileType = DataStream


#连接组件
a1.sources.r1.channels=c1
a1.sources.r2.channels=c2
a1.sinks.k1.channel=c1
a1.sinks.k2.channel=c2
```

二、数仓分层

数据分层概念

> 数据仓库为什么分层：
>
> 1. 把复杂问题简单化
>
>   将一个复杂的任务分解成多个步骤来完成，每一层只处理单一的步骤，比较简单、并且方便定位问题
>
> 2. 减少重复开发
>
>   规范数据分层，通过的中间层数据，能够减少极大的重复计算，增加一次计算结果的复用性
>
> 3. 隔离原始数据
>
>   不论是数据的异常还是数据的敏感性，使真实数据与统计数据解耦开


ODS（Operational Data Store）
原始数据层
DWD(data warehouse detail)
明细数据层
DWS(data warehouse service)
服务数据层
ADS(Application Data Store)
数据应用层

```
ODS层：存放原始数据，直接加载原始日志、数据，数据保持原貌不做处理
DWD层：结构和粒度与原始表保持一致，对ODS层数据进行清洗(去除空值，脏数据，超过极限范围的数据)
DWS层：以DWD为基础，进行轻度汇总
ADS层：为各种统计报表提供数据
```

数仓分层思路

1) ODS层 (原始数据层)

```
/origin_data/gmall/log/topic_start/  2020-10-30
                                     2020-10-31
                       ods_start_log 2020-10-30
                                     2020-10-31
-----------------------------------------------------
/origin_data/gmall/log/topic_event   2020-10-30
                                     2020-10-31
                       ods_event_log 2020-10-30
                                     2020-10-31
```

2)DWD层(明细数据层)

	ods_start_log	**ods_event_log**
基本明细解析		dwd_base_event_log
具体的表	dwd_start_log	dwd_display_log、dwd_active_foreground_log dwd_newsdetail_log、dwd_active_background_log dwd_loading_log、dwd_comment_log、dwd_error_log、dwd_ad_log、dwd_praise_log、dwd_notification_log、dwd_favorites

3)DWS层(服务数据层)

```
以DWD为基础，进行轻度汇总。一般聚集到以用户当日，设备当日，商家当日，商品当日等等的粒度
在这层通常会有以某一个维护为线索，组成跨主题的宽表，比如：一个用户的当日的签到数、收藏数、评论数、抽奖数、订阅数、点赞数、浏览商品数、添加购物车数、下单数、支付数、退款数、点击广告数组成的多列表

dwd_start_log            启动日志表

dws_uv_detail_day        每日活跃设备明细表
dws_uv_detail_wk         每周活跃设备明细表
dws_uv_detail_mn         每月活跃设备明细表
dws_new_mid_day          每日新增设备明细表
dws_user_retention_day   每日留存用户明细表
```

4)ADS层(数据应用层)

```
ADS层：为各种统计报表提供数据，也有公司把这层命名为APP层

dws_uv_detail_day        每日活跃设备明细表
dws_uv_detail_wk         每周活跃设备明细表
dws_uv_detail_mn         每月活跃设备明细表

ads_uv_count             日、周、月设备活跃
```

数仓分层之ODS层

> 原始数据层，存放原始数据，直接加载原始日志、数据，数据保持原貌不做处理。

1. 创建数据库

```
create database gmall;
use gmall;
```

说明：如果数据库存在且有数据，需要强制删除时执行：drop database gmall cascade;

2. 创建启动日志表ods_start_log【ods层启动日志表】

```
CREATE EXTERNAL TABLE ods_start_log (line string)
PARTITIONED BY (dt string)

#加载数据
load data inpath '/origin_data/gmall/log/topic_start/2019-12-14' into table gmall.ods_start_log
partition(dt='2019-12-14');
```

3. 创建事件日志表ods_event_log【ods层事件日志表】

```
CREATE EXTERNAL TABLE ods_event_log(line string)
PARTITIONED BY (dt string)
#加载数据
load data inpath '/origin_data/gmall/log/topic_event/2019-12-14' into table gmall.ods_event_log
partition(dt='2019-12-14');
```

数仓分层之DWD层

> 结构和粒度与原始数据表保持一致，对ODS层数据进行清洗（去除空值，脏数据，超过极限范围的数据）

DWD层启动表数据解析

1. 创建启动表【dwd层启动日志表】

```
CREATE EXTERNAL TABLE dwd_start_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`entry` string,
`open_ad_type` string,
`action` string,
`loading_time` string,
`detail` string,
`extend1` string
)
PARTITIONED BY (dt string)
```

2. 向启动表导入数据

```
insert overwrite table dwd_start_log
PARTITION (dt='2019-12-14')
select
    get_json_object(line,'$.mid') mid_id,
    get_json_object(line,'$.uid') user_id,
    get_json_object(line,'$.vc') version_code,
    get_json_object(line,'$.vn') version_name,
    get_json_object(line,'$.l') lang,
    get_json_object(line,'$.sr') source,
    get_json_object(line,'$.os') os,
    get_json_object(line,'$.ar') area,
    get_json_object(line,'$.md') model,
    get_json_object(line,'$.ba') brand,
    get_json_object(line,'$.sv') sdk_version,
    get_json_object(line,'$.g') gmail,
    get_json_object(line,'$.hw') height_width,
    get_json_object(line,'$.t') app_time,
    get_json_object(line,'$.nw') network,
    get_json_object(line,'$.ln') lng,
    get_json_object(line,'$.la') lat,
    get_json_object(line,'$.entry') entry,
    get_json_object(line,'$.open_ad_type') open_ad_type,
    get_json_object(line,'$.action') action,
    get_json_object(line,'$.loading_time') loading_time,
    get_json_object(line,'$.detail') detail,
    get_json_object(line,'$.extend1') extend1
from ods_start_log
where dt='2019-12-14';
```

3. 测试

```
select * from dwd_start_log limit 2;
```

```
get_json_object函数的使用:
https://blog.csdn.net/qq_34105362/article/details/80454697
```

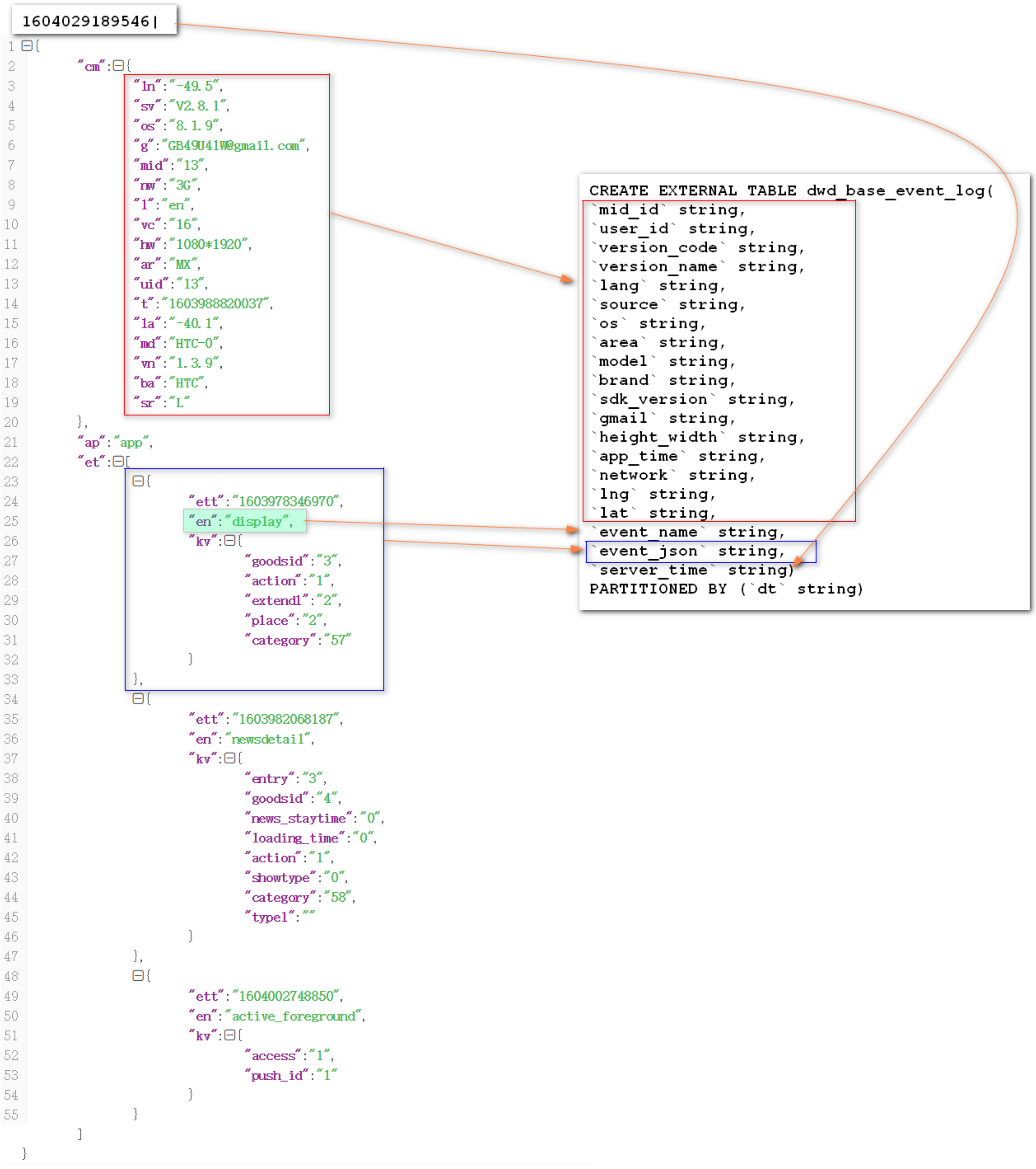
DWD层事件表数据解析

1. 创建基础明细表

> 明细表用于存储ODS层原始表转换过来的明细数据。

```
CREATE EXTERNAL TABLE dwd_base_event_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`event_name` string,
`event_json` string,
`server_time` string)
PARTITIONED BY (`dt` string)
```

说明：其中event_name和event_json用来对应事件名和整个事件。这个地方将原始日志1对多的形式拆分出来了。操作的时候我们需要将原始日志展平，需要用到UDF和UDTF。



```
1604029189546|
{
    "cm":{
        "ln":"-49.5",
        "sv":"V2.8.1",
        "os":"8.1.9",
        "g":"GB49U41W@gmail.com",
        "mid":"13",
        "nw":"3G",
        "l":"en",
        "vc":"16",
        "hw":"1080*1920",
        "ar":"MX",
        "uid":"13",
        "t":"1603988820037",
        "la":"-40.1",
        "md":"HTC-0",
        "vn":"1.3.9",
        "ba":"HTC",
        "sr":"L"
    },
    "ap":"app",
    "et":[
        {
            "ett":"1603978346970",
            "en":"display",
            "kv":{
                "goodsid":"3",
                "action":"1",
                "extend1":"2",
                "place":"2",
                "category":"57"
            }
        },
        {
            "ett":"1603982068187",
            "en":"newsdetail",
            "kv":{
                "entry":"3",
                "goodsid":"4",
                "news_staytime":"0",
                "loading_time":"0",
                "action":"1",
                "showtype":"0",
                "category":"58",
                "type1":""
            }
        },
        {
            "ett":"1604002748850",
            "en":"active_foreground",
            "kv":{
                "access":"1",
                "push_id":"1"
            }
        }
    ]
}
```

```
CREATE EXTERNAL TABLE dwd_base_event_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`event_name` string,
`event_json` string,
`server_time` string)
PARTITIONED BY (`dt` string)
```

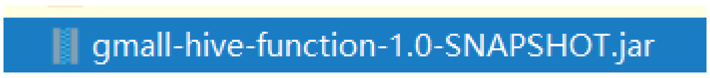
2. 自定义UDF

```
package com.baizhi.udf;

import org.apache.hadoop.hive.ql.exec.UDF;
import org.json.JSONException;
import org.json.JSONObject;

public class BaseFieldUDF extends UDF {

    public String evaluate(String line, String key) throws JSONException {
        String[] str = line.split("\\|");
        if(str.length != 2){
            return "";
        }

        JSONObject object = new JSONObject(str[1].trim());
        String result = "";

        if("et".equals(key)){
            if(object.has(key)){
                result = object.getString(key);
            }
        }else if("st".equals(key)){
            result = str[0].trim();
        }else {
            JSONObject cm = object.getJSONObject("cm");
            if(cm.has(key)){
                result = cm.getString(key);
            }
        }
        return result;
    }


    public static void main(String[] args)throws Exception {
        String s1 = "1604029189469|{\"cm\":
{\"ln\":\"-59.1\",\"sv\":\"V2.5.6\",\"os\":\"8.1.9\",\"g\":\"F68P58GN@gmail.com\",\"mid\":\
"0\",\"nw\":\"3G\",\"l\":\"pt\",\"vc\":\"18\",\"hw\":\"640*960\",\"ar\":\"MX\",\"uid\":\"0\
",\"t\":\"1603938702133\",\"la\":\"-8.7\",\"md\":\"HTC-
13\",\"vn\":\"1.2.2\",\"ba\":\"HTC\",\"sr\":\"Y\"},\"ap\":\"app\",\"et\":
[{\"ett\":\"1603967426797\",\"en\":\"display\",\"kv\":
{\"goodsid\":\"0\",\"action\":\"1\",\"extend1\":\"1\",\"place\":\"0\",\"category\":\"1\"}},
{\"ett\":\"1604016049664\",\"en\":\"newsdetail\",\"kv\":
{\"entry\":\"3\",\"goodsid\":\"1\",\"news_staytime\":\"1\",\"loading_time\":\"0\",\"action\
":\"3\",\"showtype\":\"5\",\"category\":\"55\",\"type1\":\"\"}},
{\"ett\":\"1604006992214\",\"en\":\"ad\",\"kv\":
{\"entry\":\"2\",\"show_style\":\"5\",\"action\":\"3\",\"detail\":\"\",\"source\":\"2\",\"b
ehavior\":\"1\",\"content\":\"1\",\"newstype\":\"9\"}},
{\"ett\":\"1603981343080\",\"en\":\"notification\",\"kv\":
{\"ap_time\":\"1603943309240\",\"action\":\"4\",\"type\":\"1\",\"content\":\"\"}},
{\"ett\":\"1604025178902\",\"en\":\"active_foreground\",\"kv\":
{\"access\":\"1\",\"push_id\":\"3\"}},{\"ett\":\"1603956615958\",\"en\":\"error\",\"kv\":
{\"errorDetail\":\"at cn.lift.dfdfdf.control.CommandUtil.getInfo(CommandUtil.java:67)\\\\n
at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)\\\\n
at java.lang.reflect.Method.invoke(Method.java:606)\\\\n\",\"errorBrief\":\"at
cn.lift.dfdf.web.AbstractBaseController.validInbound(AbstractBaseController.java:72)\"}},
{\"ett\":\"1603939544113\",\"en\":\"favorites\",\"kv\":
{\"course_id\":9,\"id\":0,\"add_time\":\"1603972571283\",\"userid\":7}}]}";

        String s = new BaseFieldUDF().evaluate(s1, "st");
        System.out.println(s);

    }


}
```

3. 自定义UDTF

```java
package com.baizhi.udf;

import org.apache.commons.lang.StringUtils;
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.PrimitiveObjectInspectorFactory;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.List;

public class EventJsonUDTF extends GenericUDTF {

    @Override
    public StructObjectInspector initialize(StructObjectInspector argOIs) throws
UDFArgumentException {
        //1.定义输出数据的列名和类型 (固定格式)
        List<String> fieldNames = new ArrayList<>();
        List<ObjectInspector> fieldOIs = new ArrayList<>();

        //2.添加输出数据的列名和类型
        fieldNames.add("event_name");
        //字符串的固定格式
        fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);

        fieldNames.add("event_json");
        fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);

        return ObjectInspectorFactory.getStandardStructObjectInspector(fieldNames,
fieldOIs);
    }

    //输入1条记录，输出若干条结果
    @Override
    public void process(Object[] objects) throws HiveException {
        String input = objects[0].toString();
        if(StringUtils.isBlank(input)){
            return;
        }else {
            try {
                JSONArray array = new JSONArray(input);
                if(array == null)
                    return;

                for(int i=0;i<array.length();i++){
                    String[] result = new String[2];
                    JSONObject object = array.getJSONObject(i);

                    result[0] = object.getString("en");
                    result[1] = array.getString(i);

                    // 将结果返回
                    forward(result);
                }
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public void close() throws HiveException {

    }
}
```

4. 打包


gmall-hive-function-1.0-SNAPSHOT.jar

5. 将jar上传到HDFS上的/user/hive/jars路径下

6. 创建永久函数与开发好的java class关联

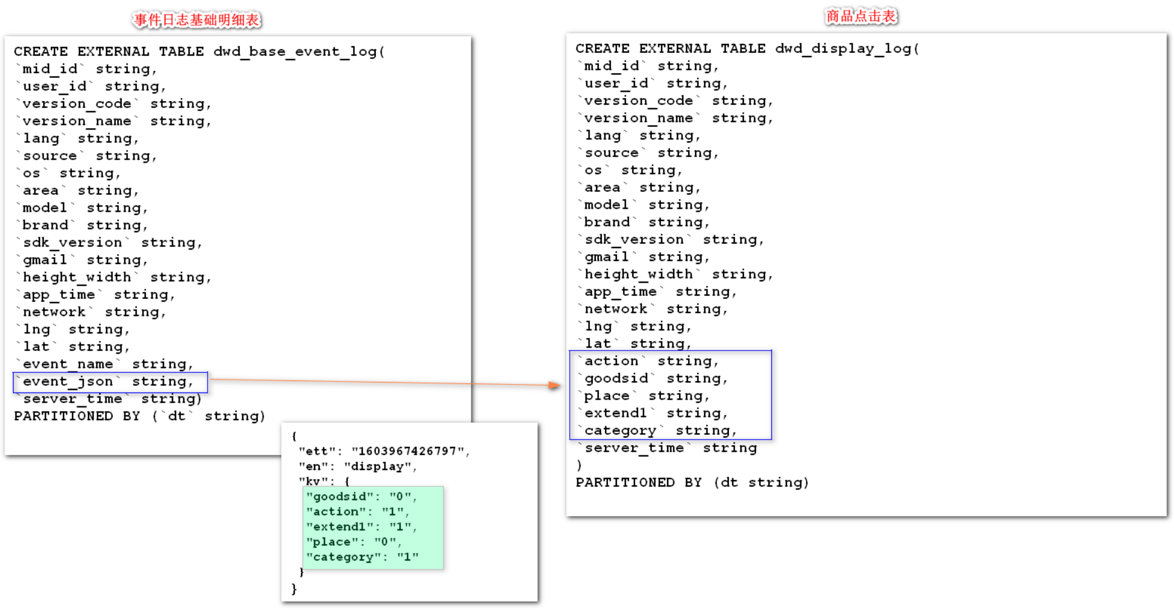
```
create function base_analizer as 'com.baizhi.udf.BaseFieldUDF' using jar
'hdfs://hadoop10:9000/user/hive/jars/gmall-hive-function-1.0-SNAPSHOT.jar';

create function flat_analizer as 'com.baizhi.udf.EventJsonUDTF' using jar
'hdfs://hadoop10:9000/user/hive/jars/gmall-hive-function-1.0-SNAPSHOT.jar';
```

7. 向解析事件日志基础明细表添加数据

```
insert overwrite table dwd_base_event_log partition(dt='2019-10-30')
select
    base_analizer(line,'mid') as mid_id,
    base_analizer(line,'uid') as user_id,
    base_analizer(line,'vc') as version_code,
    base_analizer(line,'vn') as version_name,
    base_analizer(line,'l') as lang,
    base_analizer(line,'sr') as source,
    base_analizer(line,'os') as os,
    base_analizer(line,'ar') as area,

    base_analizer(line,'md') as model,
    base_analizer(line,'ba') as brand,
    base_analizer(line,'sv') as sdk_version,
    base_analizer(line,'g') as gmail,
    base_analizer(line,'hw') as height_width,
    base_analizer(line,'t') as app_time,
    base_analizer(line,'nw') as network,
    base_analizer(line,'ln') as lng,
    base_analizer(line,'la') as lat,
    event_name,
    event_json,
    base_analizer(line,'st') as server_time
from ods_event_log lateral view flat_analizer(base_analizer(line,'et')) tmp_flat as
event_name,event_json
where dt='2019-10-30' and base_analizer(line,'et')<>'';
```

8. DWD层事件表获取

> 将事件日志基础明细表数据分发到各个原始业务表中，思路如下：



```
CREATE EXTERNAL TABLE dwd_base_event_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`event_name` string,
`event_json` string,
`server_time` string)
PARTITIONED BY (`dt` string)
```

```
{
 "ett": "1603967426797",
 "en": "display",
 "kv": {
  "goodsid": "0",
  "action": "1",
  "extend1": "1",
  "place": "0",
  "category": "1"
 }
}
```

```
CREATE EXTERNAL TABLE dwd_display_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`action` string,
`goodsid` string,
`place` string,
`extend1` string,
`category` string,
`server_time` string
)
PARTITIONED BY (dt string)
```

- 商品点击表

```
CREATE EXTERNAL TABLE dwd_display_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,

`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`action` string,
`goodsid` string,
`place` string,
`extend1` string,
`category` string,
`server_time` string
)
PARTITIONED BY (dt string);

insert overwrite table dwd_display_log
PARTITION (dt='2019-12-14')
select
mid_id,
user_id,
version_code,
version_name,
lang,
source,
os,
area,
model,
brand,
sdk_version,
gmail,
height_width,
app_time,
network,
lng,
lat,
get_json_object(event_json,'$.kv.action') action,
get_json_object(event_json,'$.kv.goodsid') goodsid,
get_json_object(event_json,'$.kv.place') place,
get_json_object(event_json,'$.kv.extend1') extend1,
get_json_object(event_json,'$.kv.category') category,
server_time
from dwd_base_event_log
where dt='2019-12-14' and event_name='display';
```

- 商品详情页表

```
CREATE EXTERNAL TABLE dwd_newsdetail_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`entry` string,
`action` string,
`goodsid` string,
`showtype` string,
`news_staytime` string,
`loading_time` string,
`type1` string,
`category` string,
`server_time` string)
PARTITIONED BY (dt string);

insert overwrite table dwd_newsdetail_log
PARTITION (dt='2019-12-14')
select
mid_id,
user_id,
version_code,
version_name,
lang,
source,
os,
area,
model,
brand,
sdk_version,
gmail,
height_width,
app_time,
network,
lng,
lat,
get_json_object(event_json,'$.kv.entry') entry,
get_json_object(event_json,'$.kv.action') action,
get_json_object(event_json,'$.kv.goodsid') goodsid,
get_json_object(event_json,'$.kv.showtype') showtype,
get_json_object(event_json,'$.kv.news_staytime') news_staytime,
get_json_object(event_json,'$.kv.loading_time') loading_time,
get_json_object(event_json,'$.kv.type1') type1,
get_json_object(event_json,'$.kv.category') category,
server_time

from dwd_base_event_log
```

```
where dt='2019-12-14' and event_name='newsdetail';
```

- 商品列表页表

```
CREATE EXTERNAL TABLE dwd_loading_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`action` string,
`loading_time` string,
`loading_way` string,
`extend1` string,
`extend2` string,
`type` string,
`type1` string,
`server_time` string)
PARTITIONED BY (dt string);

insert overwrite table dwd_loading_log
PARTITION (dt='2019-12-14')
select
mid_id,
user_id,
version_code,
version_name,
lang,
source,
os,
area,
model,
brand,
sdk_version,
gmail,
height_width,
app_time,
network,
lng,
lat,
get_json_object(event_json,'$.kv.action') action,
get_json_object(event_json,'$.kv.loading_time') loading_time,
get_json_object(event_json,'$.kv.loading_way') loading_way,
get_json_object(event_json,'$.kv.extend1') extend1,
get_json_object(event_json,'$.kv.extend2') extend2,
get_json_object(event_json,'$.kv.type') type,
get_json_object(event_json,'$.kv.type1') type1,
server_time
from dwd_base_event_log
where dt='2019-12-14' and event_name='loading';
```

- 广告表

```
CREATE EXTERNAL TABLE dwd_ad_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`entry` string,
`action` string,
`content` string,
`detail` string,
`ad_source` string,
`behavior` string,
`newstype` string,
`show_style` string,
`server_time` string)
PARTITIONED BY (dt string);

insert overwrite table dwd_ad_log
PARTITION (dt='2019-12-14')
select
mid_id,
user_id,
version_code,
version_name,
lang,
source,
os,
area,
model,
brand,
sdk_version,
gmail,
height_width,
app_time,
network,
lng,
lat,
get_json_object(event_json,'$.kv.entry') entry,
get_json_object(event_json,'$.kv.action') action,
get_json_object(event_json,'$.kv.content') content,
get_json_object(event_json,'$.kv.detail') detail,
get_json_object(event_json,'$.kv.source') ad_source,
get_json_object(event_json,'$.kv.behavior') behavior,
get_json_object(event_json,'$.kv.newstype') newstype,
get_json_object(event_json,'$.kv.show_style') show_style,
server_time
from dwd_base_event_log
where dt='2019-12-14' and event_name='ad';
```

- 消息通知表

```
CREATE EXTERNAL TABLE dwd_notification_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`action` string,
`noti_type` string,
`ap_time` string,
`content` string,
`server_time` string
)
PARTITIONED BY (dt string);
```

```
insert overwrite table dwd_notification_log
PARTITION (dt='2019-12-14')
select
mid_id,
user_id,
version_code,
version_name,
lang,
source,
os,
area,
model,
brand,
sdk_version,
gmail,
height_width,
app_time,
network,
lng,
lat,
get_json_object(event_json,'$.kv.action') action,
get_json_object(event_json,'$.kv.type') noti_type,
get_json_object(event_json,'$.kv.ap_time') ap_time,
get_json_object(event_json,'$.kv.content') content,
server_time
from dwd_base_event_log
where dt='2019-12-14' and event_name='notification';
```

- 用户前台活跃表

```
CREATE EXTERNAL TABLE dwd_active_foreground_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`push_id` string,
`access` string,
`server_time` string)
PARTITIONED BY (dt string);


insert overwrite table dwd_active_foreground_log
PARTITION (dt='2019-12-14')
select
mid_id,
user_id,
version_code,
version_name,
lang,
source,
os,
area,
model,
brand,
sdk_version,
gmail,
height_width,
app_time,
network,
lng,
lat,
get_json_object(event_json,'$.kv.push_id') push_id,
get_json_object(event_json,'$.kv.access') access,
server_time
from dwd_base_event_log
where dt='2019-12-14' and event_name='active_foreground';
```

- 用户后台活跃表

```
CREATE EXTERNAL TABLE dwd_active_background_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
 `height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`active_source` string,
`server_time` string
)
PARTITIONED BY (dt string);

insert overwrite table dwd_active_background_log
PARTITION (dt='2019-12-14')
select
mid_id,
user_id,
version_code,
version_name,
lang,
source,
os,
area,
model,
brand,
sdk_version,
gmail,
height_width,
app_time,
network,
lng,
lat,
get_json_object(event_json,'$.kv.active_source') active_source,
server_time
from dwd_base_event_log
where dt='2019-12-14' and event_name='active_background';
```

- 评论表

```
CREATE EXTERNAL TABLE dwd_comment_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`comment_id` int,
`userid` int,
`p_comment_id` int,
`content` string,
`addtime` string,
`other_id` int,
`praise_count` int,
`reply_count` int,
`server_time` string
)
PARTITIONED BY (dt string);

insert overwrite table dwd_comment_log
PARTITION (dt='2019-12-14')
select
mid_id,
user_id,
version_code,
version_name,
lang,
source,
os,
area,
model,
brand,
sdk_version,
gmail,
height_width,
app_time,
network,
lng,
lat,
get_json_object(event_json,'$.kv.comment_id') comment_id,
get_json_object(event_json,'$.kv.userid') userid,
get_json_object(event_json,'$.kv.p_comment_id') p_comment_id,
get_json_object(event_json,'$.kv.content') content,
get_json_object(event_json,'$.kv.addtime') addtime,
get_json_object(event_json,'$.kv.other_id') other_id,
get_json_object(event_json,'$.kv.praise_count') praise_count,
get_json_object(event_json,'$.kv.reply_count') reply_count,
server_time
from dwd_base_event_log
where dt='2019-12-14' and event_name='comment';
```

- 收藏表

```
CREATE EXTERNAL TABLE dwd_favorites_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`id` int,
`course_id` int,
`userid` int,
`add_time` string,
`server_time` string
)
PARTITIONED BY (dt string);

insert overwrite table dwd_favorites_log
PARTITION (dt='2019-12-14')
select
mid_id,
user_id,
version_code,
version_name,
lang,
source,
os,
area,
model,
brand,
sdk_version,
gmail,
height_width,
app_time,
network,
lng,
lat,
get_json_object(event_json,'$.kv.id') id,
get_json_object(event_json,'$.kv.course_id') course_id,
get_json_object(event_json,'$.kv.userid') userid,
get_json_object(event_json,'$.kv.add_time') add_time,
server_time
from dwd_base_event_log
where dt='2019-12-14' and event_name='favorites';
```

- 点赞表

```
CREATE EXTERNAL TABLE dwd_praise_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`id` string,
`userid` string,
`target_id` string,
`type` string,
`add_time` string,
`server_time` string
)
PARTITIONED BY (dt string);

insert overwrite table dwd_praise_log
PARTITION (dt='2019-12-14')
select
mid_id,
user_id,
version_code,
version_name,
lang,
source,
os,
area,
model,
brand,
sdk_version,
gmail,
height_width,
app_time,
network,
lng,
lat,
get_json_object(event_json,'$.kv.id') id,
get_json_object(event_json,'$.kv.userid') userid,
get_json_object(event_json,'$.kv.target_id') target_id,
get_json_object(event_json,'$.kv.type') type,
get_json_object(event_json,'$.kv.add_time') add_time,
server_time
from dwd_base_event_log
where dt='2019-12-14' and event_name='praise';
```

- 错误日志表

```
CREATE EXTERNAL TABLE dwd_error_log(
`mid_id` string,
`user_id` string,
`version_code` string,
`version_name` string,
`lang` string,
`source` string,
`os` string,
`area` string,
`model` string,
`brand` string,
`sdk_version` string,
`gmail` string,
`height_width` string,
`app_time` string,
`network` string,
`lng` string,
`lat` string,
`errorBrief` string,
`errorDetail` string,
`server_time` string)
PARTITIONED BY (dt string);

insert overwrite table dwd_error_log
PARTITION (dt='2019-12-14')
select
mid_id,
user_id,
version_code,
version_name,
lang,
source,
os,
area,
model,
brand,
sdk_version,
gmail,
height_width,
app_time,
network,
lng,
lat,
get_json_object(event_json,'$.kv.errorBrief') errorBrief,
get_json_object(event_json,'$.kv.errorDetail') errorDetail,
server_time
from dwd_base_event_log
where dt='2019-12-14' and event_name='error';
```

三、需求一: 用户活跃主题

DWS层

目标：统计当日(DAU)、当周、当月活动的每个设备明细

每日活跃设备明细

- 建表【每日活跃设备明细表】

```
create external table dws_uv_detail_day
(
    `mid_id` string COMMENT '设备唯一标识',
    `user_id` string COMMENT '用户标识',
    `version_code` string COMMENT '程序版本号',
    `version_name` string COMMENT '程序版本名',
    `lang` string COMMENT '系统语言',
    `source` string COMMENT '渠道号',
    `os` string COMMENT '安卓系统版本',
    `area` string COMMENT '区域',
    `model` string COMMENT '手机型号',
    `brand` string COMMENT '手机品牌',
    `sdk_version` string COMMENT 'sdkVersion',
    `gmail` string COMMENT 'gmail',
    `height_width` string COMMENT '屏幕宽高',
    `app_time` string COMMENT '客户端日志产生时的时间',
    `network` string COMMENT '网络模式',
    `lng` string COMMENT '经度',
    `lat` string COMMENT '纬度'
)
partitioned by(dt string)
```

- 导入数据

```
insert overwrite table dws_uv_detail_day
partition(dt='2020-10-30')
select
    mid_id,
    concat_ws('|', collect_set(user_id)) user_id,
    concat_ws('|', collect_set(version_code)) version_code,
    concat_ws('|', collect_set(version_name)) version_name,
    concat_ws('|', collect_set(lang))lang,
    concat_ws('|', collect_set(source)) source,
    concat_ws('|', collect_set(os)) os,
    concat_ws('|', collect_set(area)) area,
    concat_ws('|', collect_set(model)) model,
    concat_ws('|', collect_set(brand)) brand,
    concat_ws('|', collect_set(sdk_version)) sdk_version,
    concat_ws('|', collect_set(gmail)) gmail,
    concat_ws('|', collect_set(height_width)) height_width,
    concat_ws('|', collect_set(app_time)) app_time,
    concat_ws('|', collect_set(network)) network,
    concat_ws('|', collect_set(lng)) lng,
    concat_ws('|', collect_set(lat)) lat
from dwd_start_log
where dt='2020-10-30'
group by mid_id;
```

每周活跃明细

- 建表【每周活跃设备明细表】

```
create external table dws_uv_detail_wk(
    `mid_id` string COMMENT '设备唯一标识',
    `user_id` string COMMENT '用户标识',
    `version_code` string COMMENT '程序版本号',
    `version_name` string COMMENT '程序版本名',
    `lang` string COMMENT '系统语言',
    `source` string COMMENT '渠道号',
    `os` string COMMENT '安卓系统版本',
    `area` string COMMENT '区域',
    `model` string COMMENT '手机型号',
    `brand` string COMMENT '手机品牌',
    `sdk_version` string COMMENT 'sdkVersion',
    `gmail` string COMMENT 'gmail',
    `height_width` string COMMENT '屏幕宽高',
    `app_time` string COMMENT '客户端日志产生时的时间',
    `network` string COMMENT '网络模式',
    `lng` string COMMENT '经度',
    `lat` string COMMENT '纬度',
    `monday_date` string COMMENT '周一日期',
    `sunday_date` string COMMENT  '周日日期'
) COMMENT '活跃用户按周明细'
PARTITIONED BY (`wk_dt` string)
```

- 导入数据

```
set hive.exec.dynamic.partition.mode=nonstrict;

insert overwrite table dws_uv_detail_wk partition(wk_dt)
select
    mid_id,
    concat_ws('|', collect_set(user_id)) user_id,
    concat_ws('|', collect_set(version_code)) version_code,
    concat_ws('|', collect_set(version_name)) version_name,
    concat_ws('|', collect_set(lang)) lang,
    concat_ws('|', collect_set(source)) source,
    concat_ws('|', collect_set(os)) os,
    concat_ws('|', collect_set(area)) area,

    concat_ws('|', collect_set(model)) model,
    concat_ws('|', collect_set(brand)) brand,
    concat_ws('|', collect_set(sdk_version)) sdk_version,
    concat_ws('|', collect_set(gmail)) gmail,
    concat_ws('|', collect_set(height_width)) height_width,
    concat_ws('|', collect_set(app_time)) app_time,
    concat_ws('|', collect_set(network)) network,
    concat_ws('|', collect_set(lng)) lng,
    concat_ws('|', collect_set(lat)) lat,
    date_add(next_day('2020-10-30','MO'),-7),
    date_add(next_day('2020-10-30','MO'),-1),
    concat(date_add(next_day('2020-10-30','MO'),-7), '_' , date_add(next_day('2020-10-
30','MO'),-1))
from dws_uv_detail_day
where dt>=date_add(next_day('2020-10-30','MO'),-7) and dt<=date_add(next_day('2020-10-
30','MO'),-1)
group by mid_id;
```

每月活跃设备明细

- 建表【每月活跃设备明细表】

```
create external table dws_uv_detail_mn(
    `mid_id` string COMMENT '设备唯一标识',
    `user_id` string COMMENT '用户标识',
    `version_code` string COMMENT '程序版本号',
    `version_name` string COMMENT '程序版本名',
    `lang` string COMMENT '系统语言',
    `source` string COMMENT '渠道号',
    `os` string COMMENT '安卓系统版本',
    `area` string COMMENT '区域',
    `model` string COMMENT '手机型号',
    `brand` string COMMENT '手机品牌',
    `sdk_version` string COMMENT 'sdkVersion',
    `gmail` string COMMENT 'gmail',
    `height_width` string COMMENT '屏幕宽高',
    `app_time` string COMMENT '客户端日志产生时的时间',
    `network` string COMMENT '网络模式',
    `lng` string COMMENT '经度',
    `lat` string COMMENT '纬度'
) COMMENT '活跃用户按月明细'
PARTITIONED BY (`mn` string)
```

- 导入数据

```
set hive.exec.dynamic.partition.mode=nonstrict;

insert overwrite table dws_uv_detail_mn partition(mn)
select
    mid_id,
    concat_ws('|', collect_set(user_id)) user_id,
    concat_ws('|', collect_set(version_code)) version_code,
    concat_ws('|', collect_set(version_name)) version_name,
    concat_ws('|', collect_set(lang)) lang,
    concat_ws('|', collect_set(source)) source,
    concat_ws('|', collect_set(os)) os,
    concat_ws('|', collect_set(area)) area,
    concat_ws('|', collect_set(model)) model,
    concat_ws('|', collect_set(brand)) brand,
    concat_ws('|', collect_set(sdk_version)) sdk_version,
    concat_ws('|', collect_set(gmail)) gmail,
    concat_ws('|', collect_set(height_width)) height_width,
    concat_ws('|', collect_set(app_time)) app_time,
    concat_ws('|', collect_set(network)) network,
    concat_ws('|', collect_set(lng)) lng,
    concat_ws('|', collect_set(lat)) lat,
    date_format('2020-10-30','yyyy-MM')
from dws_uv_detail_day
where date_format(dt,'yyyy-MM') = date_format('2020-10-30','yyyy-MM')
group by mid_id;
```

ADS层

目标：当日、当周、当月活跃设备数

- 建表【活跃设备统计结果表】

```
create external table ads_uv_count(
    `dt` string COMMENT '统计日期',
    `day_count` bigint COMMENT '当日用户数量',
    `wk_count`  bigint COMMENT '当周用户数量',
    `mn_count`  bigint COMMENT '当月用户数量',
    `is_weekend` string COMMENT 'Y,N是否是周末,用于得到本周最终结果',
    `is_monthend` string COMMENT 'Y,N是否是月末,用于得到本月最终结果'
) COMMENT '活跃设备数'
row format delimited fields terminated by '\t'
```

- 导入数据

```
insert into table ads_uv_count
select
  '2020-10-30' dt,
   daycount.ct,
   wkcount.ct,
   mncount.ct,
   if(date_add(next_day('2020-10-30','MO'),-1)='2020-10-30','Y','N') ,
   if(last_day('2020-10-30')='2020-10-30','Y','N')
from
(
   select
      '2020-10-30' dt,
       count(*) ct
   from dws_uv_detail_day
   where dt='2020-10-30'
)daycount join
(
   select
     '2020-10-30' dt,
     count (*) ct
   from dws_uv_detail_wk
   where wk_dt=concat(date_add(next_day('2020-10-30','MO'),-7),'_' ,date_add(next_day('2020-10-30','MO'),-1) )
) wkcount on daycount.dt=wkcount.dt
join
(
   select
     '2020-10-30' dt,
     count (*) ct
   from dws_uv_detail_mn
   where mn=date_format('2020-10-30','yyyy-MM')
)mncount on daycount.dt=mncount.dt;
```

四、需求二：用户新增主题

首次联网使用应用的用户。如果一个用户首次打开某APP，那这个用户定义为新增用户；卸载再安装的设备，不会被算作一次新增。新增用户包括日新增用户、周新增用户、月新增用户。

DWS层

- 建表【每日新增设备明细表】

```
create external table dws_new_mid_day
(
    `mid_id` string COMMENT '设备唯一标识',
    `user_id` string COMMENT '用户标识',
    `version_code` string COMMENT '程序版本号',
    `version_name` string COMMENT '程序版本名',
    `lang` string COMMENT '系统语言',
    `source` string COMMENT '渠道号',
    `os` string COMMENT '安卓系统版本',
    `area` string COMMENT '区域',
    `model` string COMMENT '手机型号',
    `brand` string COMMENT '手机品牌',
    `sdk_version` string COMMENT 'sdkVersion',
    `gmail` string COMMENT 'gmail',
    `height_width` string COMMENT '屏幕宽高',
    `app_time` string COMMENT '客户端日志产生时的时间',
    `network` string COMMENT '网络模式',
    `lng` string COMMENT '经度',
    `lat` string COMMENT '纬度',

    `create_date`  string  comment '创建时间'
) COMMENT '每日新增设备信息'
```

- 导入数据

```
insert into table dws_new_mid_day
select
    ud.mid_id,
    ud.user_id ,
    ud.version_code ,
    ud.version_name ,
    ud.lang ,
    ud.source,
    ud.os,
    ud.area,
    ud.model,
    ud.brand,
    ud.sdk_version,
    ud.gmail,
    ud.height_width,
    ud.app_time,
    ud.network,
    ud.lng,
    ud.lat,
    '2020-10-30'
from dws_uv_detail_day ud left join dws_new_mid_day nm on ud.mid_id=nm.mid_id
where ud.dt='2020-10-30' and nm.mid_id is null;
```

ADS层

- 建表【每日新增设备统计结果表】

```
create external table ads_new_mid_count(
`create_date`   string comment '创建时间',
`new_mid_count` BIGINT comment '新增设备数量'
) COMMENT '每日新增设备信息数量'
row format delimited
fields terminated by '\t'
```

- 导入数据

```
insert into table ads_new_mid_count
select
create_date,
count(*)
from dws_new_mid_day
where create_date='2020-10-30'
group by create_date;
```

五、需求三：用户留存主题

留存用户：某段时间内的新增用户，经过一段时间后，又继续使用应用的被认作是留存用户

留存率：留存用户占新增用户的比例即是留存率

比如：2月10日新增用户100，这100人中在2月11日启动过应用的有30人，2月12日启动过应用的有25人，2月13日启动过应用的有32人

则2月10日新增用户次日的留存率是30/100=30%，两日留存率是25/100=25%，三日留存率是32/100=32%

DWS层

- 建表【用户每日留存信息表】

```
create external table dws_user_retention_day
(
    `mid_id` string COMMENT '设备唯一标识',
    `user_id` string COMMENT '用户标识',
    `version_code` string COMMENT '程序版本号',
    `version_name` string COMMENT '程序版本名',
    `lang` string COMMENT '系统语言',
    `source` string COMMENT '渠道号',
    `os` string COMMENT '安卓系统版本',
    `area` string COMMENT '区域',
    `model` string COMMENT '手机型号',
    `brand` string COMMENT '手机品牌',
    `sdk_version` string COMMENT 'sdkVersion',
    `gmail` string COMMENT 'gmail',
    `height_width` string COMMENT '屏幕宽高',
    `app_time` string COMMENT '客户端日志产生时的时间',
    `network` string COMMENT '网络模式',
    `lng` string COMMENT '经度',
    `lat` string COMMENT '纬度',
    `create_date`    string  comment '设备新增时间',
    `retention_day`  int comment '截止当前日期留存天数'
)  COMMENT '每日用户留存情况'
PARTITIONED BY (`dt` string)
```

- 导入数据(每天计算前1天的新用户访问留存明细)

```
insert overwrite table dws_user_retention_day
partition(dt="2020-10-31")
select
    nm.mid_id,
    nm.user_id ,
    nm.version_code ,
    nm.version_name ,
    nm.lang ,
    nm.source,
    nm.os,
    nm.area,
    nm.model,
    nm.brand,
    nm.sdk_version,
    nm.gmail,
    nm.height_width,
    nm.app_time,
    nm.network,
    nm.lng,
    nm.lat,
    nm.create_date,
    1 retention_day
from dws_uv_detail_day ud join dws_new_mid_day nm on ud.mid_id =nm.mid_id
where ud.dt='2020-10-31' and nm.create_date=date_add('2020-10-31',-1);
```

- 导入数据（每天计算前1,2,3的新用户访问留存明细）

```
insert overwrite table dws_user_retention_day
partition(dt="2019-02-11")
select
    nm.mid_id,
    nm.user_id,
    nm.version_code,
    nm.version_name,
    nm.lang,
    nm.source,
    nm.os,
    nm.area,
    nm.model,
    nm.brand,
    nm.sdk_version,
    nm.gmail,
    nm.height_width,
    nm.app_time,
    nm.network,
    nm.lng,
    nm.lat,
    nm.create_date,
    1 retention_day
from dws_uv_detail_day ud join dws_new_mid_day nm  on ud.mid_id =nm.mid_id
where ud.dt='2019-02-11' and nm.create_date=date_add('2019-02-11',-1)
union all
select
    nm.mid_id,
    nm.user_id ,
    nm.version_code ,
    nm.version_name ,
    nm.lang ,
    nm.source,
    nm.os,
    nm.area,
    nm.model,
    nm.brand,
    nm.sdk_version,
    nm.gmail,
    nm.height_width,
    nm.app_time,
    nm.network,
    nm.lng,
    nm.lat,
    nm.create_date,
    2 retention_day
from  dws_uv_detail_day ud join dws_new_mid_day nm   on ud.mid_id =nm.mid_id
where ud.dt='2019-02-11' and nm.create_date=date_add('2019-02-11',-2)
union all
select
    nm.mid_id,
    nm.user_id,
    nm.version_code,
    nm.version_name,
    nm.lang,
    nm.source,
    nm.os,
    nm.area,
    nm.model,
    nm.brand,
    nm.sdk_version,
    nm.gmail,
    nm.height_width,
    nm.app_time,
    nm.network,
    nm.lng,
    nm.lat,
    nm.create_date,
    3 retention_day
from  dws_uv_detail_day ud join dws_new_mid_day nm   on ud.mid_id =nm.mid_id
where ud.dt='2019-02-11' and nm.create_date=date_add('2019-02-11',-3);
```

ADS层

- 建表【用户留存统计表】

```
create external table ads_user_retention_day_count
(
`create_date`       string  comment '设备新增日期',
`retention_day`     int comment '截止当前日期留存天数',
`retention_count`    bigint comment  '留存数量'
)  COMMENT '每日用户留存情况'
row format delimited fields terminated by '\t'
```

- 导入数据

```
insert into table ads_user_retention_day_count
select
    create_date,
    retention_day,
    count(*) retention_count
from dws_user_retention_day
where dt='2020-10-31'
group by create_date,retention_day;
```

- 建表【用户留存率结果表】

```
create external table ads_user_retention_day_rate
(
`stat_date`          string comment '统计日期',
`create_date`        string  comment '设备新增日期',
`retention_day`      int comment '截止当前日期留存天数',
`retention_count`    bigint comment  '留存数量',
`new_mid_count`      bigint comment '当日设备新增数量',
`retention_ratio`    decimal(10,2) comment '留存率'
)  COMMENT '每日用户留存情况'
row format delimited fields terminated by '\t'
```

- 导入数据

```
insert into ads_user_retention_day_rate
select
    '2020-10-31',
    t1.create_date,
    t1.retention_day,
    t1.retention_count,
    t2.new_mid_count,
    t1.retention_count / t2.new_mid_count * 100
from ads_user_retention_day_count t1
join ads_new_mid_count t2
on t1.create_date = t2.create_date

where date_add(t1.create_date,t1.retention_day) = '2020-10-31'
```

六、需求四：新收藏用户数

新收藏用户：指的是在某天首次添加收藏的用户

DWS层建立用户日志行为宽表

> 宽表：从字面意义上讲就是字段比较多的数据库表。通常是指业务主题相关的指标、维度、属性关联在一起的一张数据库表。由于把不同的内容都放在同一张表存储，宽表已经不符合三范式的模型设计规范，随之带来的主要坏处就是数据的大量冗余，与之相对应的好处就是查询性能的提高与便捷。 窄表：严格按照数据库设计三范式。尽量减少数据冗余，但是缺点是修改一个数据可能需要修改多张表
>
> 用户日志行为宽表存储：每个用户对每个商品的点击次数, 点赞次数, 收藏次数

- 建表【用户日志行为宽表】

```sql
CREATE EXTERNAL TABLE dws_user_action_wide_log(
    `mid_id` string COMMENT '设备id',
    `goodsid` string COMMENT '商品id',
    `display_count` string COMMENT '点击次数',
    `praise_count` string COMMENT '点赞次数',
    `favorite_count` string COMMENT '收藏次数')
PARTITIONED BY (`dt` string)
```

- 导入数据

```sql
insert overwrite table dws_user_action_wide_log partition(dt='2020-10-30')
select
    mid_id,
    goodsid,
    sum(display_count)  display_count,
    sum(praise_count)   praise_count,
    sum(favorite_count) favorite_count
from
( select
        mid_id,
        goodsid,
        count(*) display_count,
        0 praise_count,
        0 favorite_count
    from
        dwd_display_log

    where
```

```
            dt='2020-10-30' and action=2
        group by
            mid_id,goodsid

        union all

        select
            mid_id,
            target_id goodsid,
            0,
            count(*) praise_count,
            0
        from
            dwd_praise_log
        where
            dt='2020-10-30'
        group by
            mid_id,target_id

        union all

        select
            mid_id,
            course_id goodsid,
            0,
            0,
            count(*) favorite_count
        from
            dwd_favorites_log
        where
            dt='2020-10-30'
        group by
            mid_id,course_id
    )user_action
    group by
    mid_id,goodsid;
```

DWS层

使用日志数据用户行为宽表作为DWS层表

ADS层

- 建表【每日新收藏用户表】

```
create external table ads_new_favorites_mid_day(
    `dt` string COMMENT '日期',
    `favorites_users` bigint COMMENT '新收藏用户数'
)
row format delimited fields terminated by '\t'
```

- 导入数据

```
insert into table ads_new_favorites_mid_day
select
    '2020-10-30' dt,
    count(*) favorites_users
from
(
    select
        mid_id
    from
        dws_user_action_wide_log
    where
        favorite_count>0 and dt = '2020-10-30'
    group by
        mid_id

)user_favorite;
```

七、需求五：各个商品点击次数top3的用户

需求：基于用户行为宽表统计每个商品被点击次数中排名前3的用户

DWS层

使用日志数据用户行为宽表作为DWS层表

ADS层

- 建表【商品用户点击次数top3表】

```
create external table ads_goods_count(
    `dt` string COMMENT '统计日期',
    `goodsid` string COMMENT '商品',
    `user_id` string COMMENT '用户',
    `goodsid_user_count` bigint COMMENT '商品用户点击次数'
)
row format delimited fields terminated by '\t'
```

- 导入数据

```sql
insert into table ads_goods_count
select
    '2020-10-30',
    goodsid,
    mid_id,
    sum_display_count
from(
    select
      goodsid,
      mid_id,
      sum_display_count,
      row_number() over(partition by goodsid order by sum_display_count desc) rk
    from(
      select
        goodsid,
        mid_id,
        sum(display_count) sum_display_count
      from dws_user_action_wide_log
      where display_count>0
      group by goodsid, mid_id
    ) t1
) t2
where rk <= 3
```

八、需求六：统计每日各类别下点击次数top10的商品

> 需求：统计每日各类别下点击次数top10的商品

DWS层

使用点击日志表作为DWS层数据源

ADS层

- 建表【每日各类别商品点击前10表】

```sql
create external table ads_goods_display_top10 (
    `dt` string COMMENT '日期',
    `category` string COMMENT '品类',
    `goodsid` string COMMENT '商品id',
    `goods_count` string COMMENT '商品点击次数'
)
row format delimited fields terminated by '\t'
```

- 导入数据

```sql
insert into table ads_goods_display_top10
select
  '2010-10-30',
  category,
  goodsid,
  count
from(
  select
    category,
    goodsid,
    count,
    rank() over(partition by category order by count desc) rk
  from(
    select
      category,
      goodsid,
      count(*) count
    from dwd_display_log
    where dt='2010-10-30' and action=2
    group by category, goodsid
  )t1
)t2
where rk<=10;
```