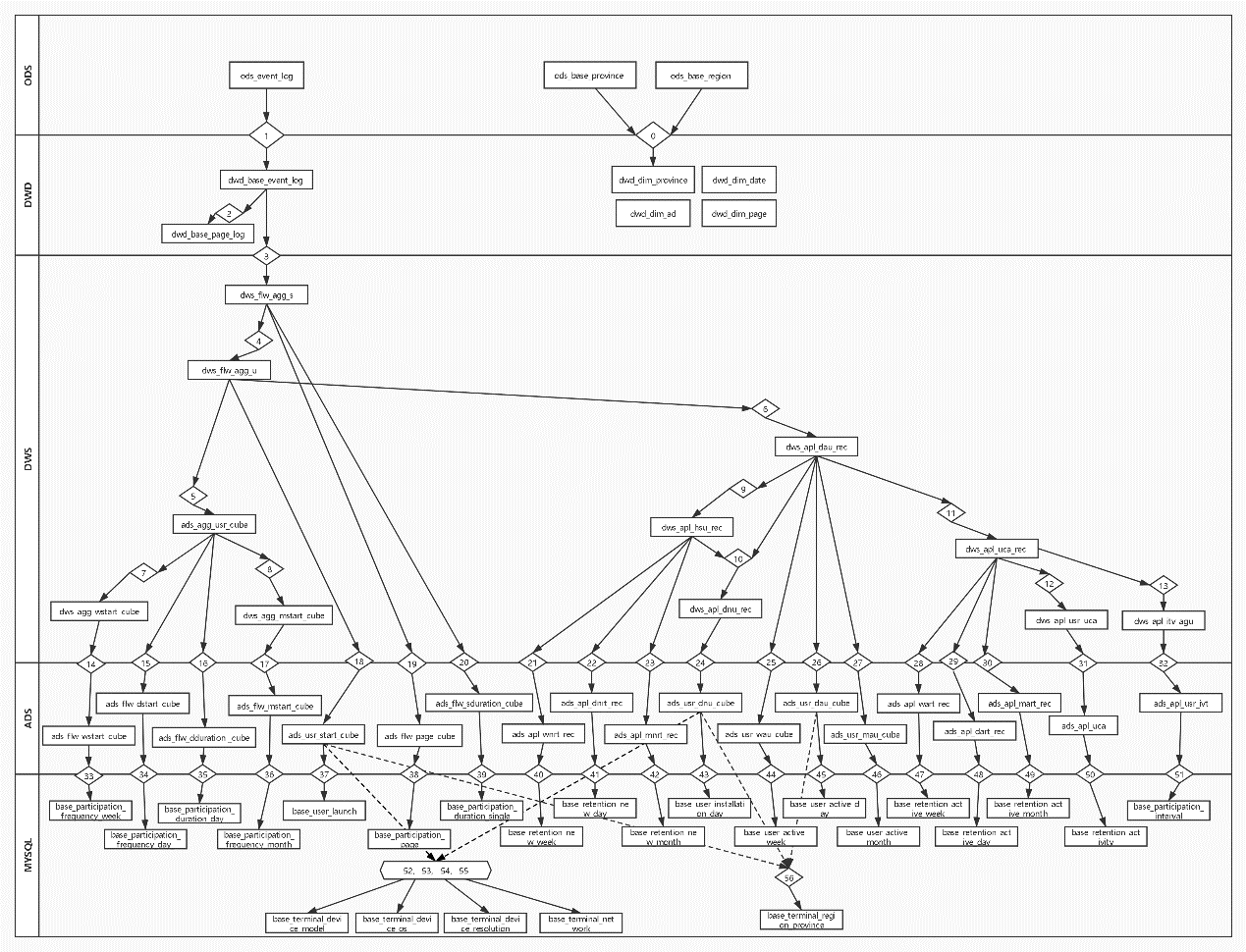
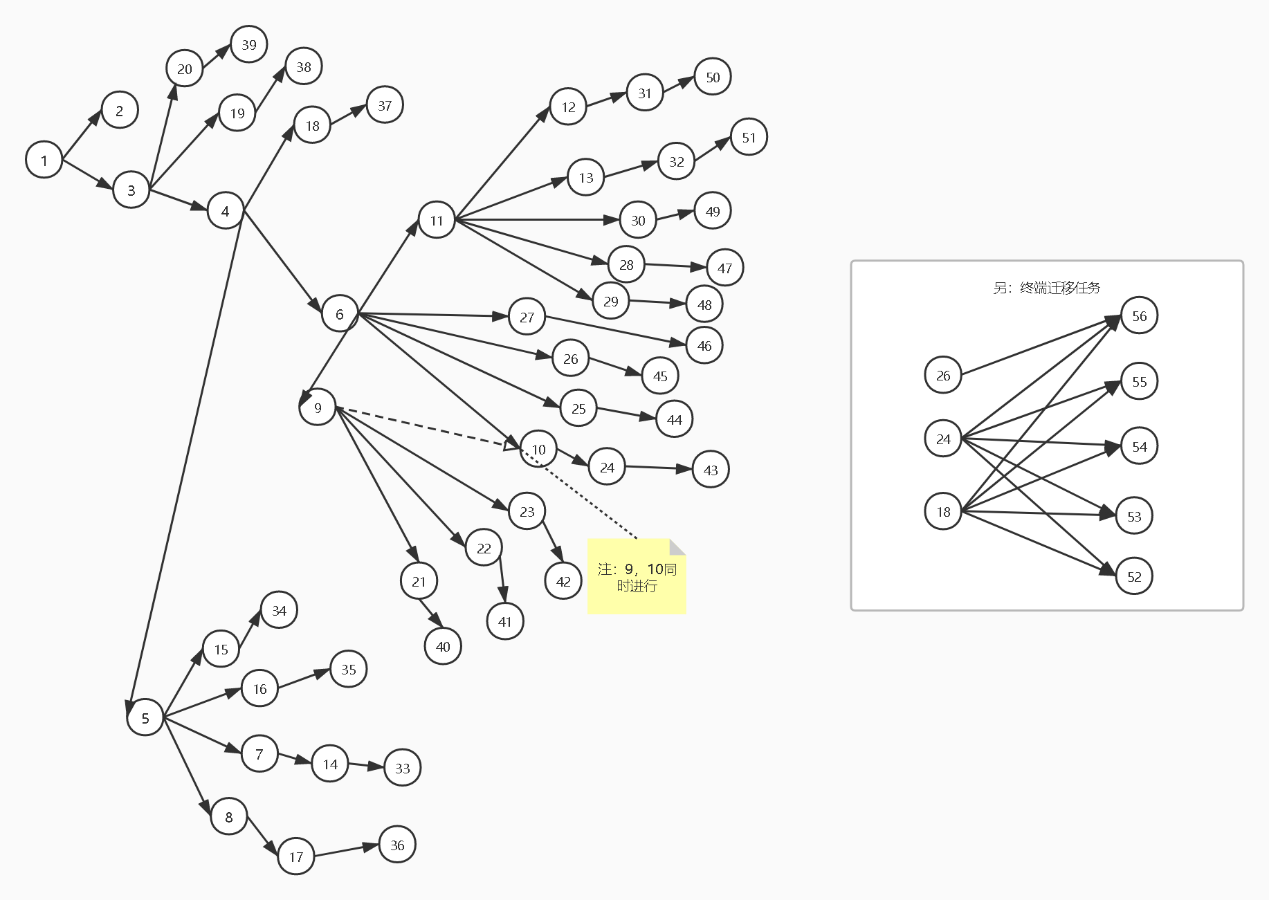
# 数仓定时任务设计

表与任务的总体关系



任务之间依赖关系



任务列表

1. 将ods层的省份表和地域表通过join生成dwd层身份维度表。
2. 将ods层原始日志文件每行的json字符串进行解析，生成dwd层基本日志事件表。
3. 筛选基本日志事件表中的页面点击事件，将事件内容进行进一步解析形成页面日志表。
4. 将dwd的基本日志事件表以sessionid进行聚合并做离散化处理，得到dws层的流量会话聚合表。
5. 将流量会话聚合表以用户id和关注各种维度进行聚合，得到流量用户聚合表。
6. 将流量用户聚合表根据用户id和流量关注维度进一步聚合和离散化处理，得到用户聚合多维表作为中间结果。
7. 从流量用户聚合表中抽取关心的字段形成日活记录表。
8. 将用户聚合多维表一周启动数数据进行进一步聚合，得到周启动多维表做中间结果。
9. 将用户聚合多维表一月启动数数据进行进一步聚合，得到月启动多为表做中间结果。
10. 利用日活记录表更新历史访问记录表。
11. 任务9同时利用昨日的历史访问记录表和今日的日活记录表产生日新记录表。
12. 利用日活记录表更新用户活跃区间表。
13. 利用用户活跃区间表，筛选当天活跃的用户根据用户最近的活跃区间得到其活跃的天数，作为活跃度的中间结果。
14. 利用用户活跃区间表获得每个用户在一个月内各种时间间隔发生的次数作为中间结果的访问间隔分布用户聚合表。
15. 将周启动多维表按照启动数范围聚合得到周启动数流量多维报表。
16. 将用户聚合多维表按照启动数范围聚合得到日启动数流量多维报表。
17. 将用户聚合多维表按照使用时长范围聚合得到日使用时长流量多维报表。
18. 将月启动多维表按照启动数范围聚合得到月启动数流量多维报表。
19. 将流量用户聚合表根据选择需要的列进行聚合得到日启动数多维报表。
20. 将流量会话聚合表根据访问页面数范围进行聚合得到访问页面数多维报表。
21. 将流量会话聚合表根据使用时长范围进行聚合得到单次使用时长多维报表。
22. 利用更新后的历史访问记录表，选择最后登录日期为今天的记录，最早与最后登录日期差值即为新增用户的留存天数，根据最早登录日期聚合进行留存人数计算。
23. 同23，区别为选择周范围。
24. 同23，区别为选择周范围。
25. 将日新记录表按照所需列进行聚合得到日新数多维报表。
26. 将日活记录表按照所需列进行聚合得到周活数多维报表。
27. 同27，时间区间为天。
28. 同27，时间区间为月。
29. 从用户活跃区间记录表内选择本周活跃过的用户，将区间展开进行聚合，得到周活跃用户留存表。
30. 从用户活跃区间记录表内选择当日活跃过的用户，将区间展开进行聚合，得到日活跃用户留存表。
31. 从用户活跃区间记录表内选择本月活跃过的用户，将区间展开进行聚合，得到月活跃用户留存表。
32. 任务11同时利用昨天的用户活跃区间表和今天的日活记录表做全连接得到活跃天数多维报表。
33. 将访问间隔分布用户聚合表按照间隔天数范围做聚合得到用户访问间隔多维报表。

33-56. 迁移任务，先读取对应维度表进行id转化，经过补零后插入到对应的mysql数据库内的表。

技术选型

1. 存储利用Hive、Mysql，hdfs。
2. 项目构建使用maven。
3. 使用scala2.12语言编写
4. 分析、迁移利用spark core/sql/graph。
5. 任务调度利用Azkaban。

项目结构

|--resource 资源文件夹，包括各种配置文件。

|--scala 源代码文件

| |--common 公用部分包

| | |--conf 配置管理包

| | |--constan 常量管理包

| | |--utils 工具类包

| |--analysis 分析部分顶级包

| | |--base 基础表分析任务实现包

| | |--flow 流量表分析任务实现包

| | |--retention 留存表分析任务实现表

| | |--dim 维度表初始化包

| | |--apl 其他分析任务实现包

| | | |--app 执行将多个任务按依赖关系串联形成的粗粒度的任务的对象包

| | | |--bean 存放封装业务数据的案例类

| | | |--function 其中每一个对象对应每一个定时任务

| | | |--udf 存放spark-sql中使用的自定义函数（UDAF、UDTF）

| |--migrantion 迁移部分顶级包

| | |--baseAnalysis 包含基本统计分析结果展示表的迁移任务

| | |--participation 包含参与度分析结果展示表的迁移任务

| | |--retention 包含留存分析结果展示表的迁移任务

| | |--terminal 包含终端属性分析结果展示表的迁移任务

| | | |--app 同analysis包内app

| | | |--function 同analysis包内function

任务细节设计

1. ods\_event\_log 🡪 dwd\_base\_event\_log
   1. 相关类：
      1. analysis.base.bean.EventLogBean

封装了json解析之后的字段，与dwd\_base\_event\_log 表中的列对应。

* + 1. analysis.base.function.IdMapFunction

负责利用今日的标识码的上一天的id-map数据产生今天的id-map

* + 1. :analysis.base.function.JSONParseFunction

将ods层的json日志进行解析

* 1. 整体逻辑：
     1. 将当日每条记录标识字段为顶点，两两组合为边，使用spakr-graph计算出连通子图，为每个子图内的标识字段分配统一的guid。
     2. 将昨天的id‘-map转化为图，与当日的图进行合并并更新guid，如果标识字段有guid，就利用历史的guid替换今日的，最后将合并后的图转化为guid文件。
     3. 读取ods\_event\_log表内当日分区内的数据，将line列进行json解析，并根据id-map和每行的标识字段分配guid，将解析结果插入到dwd\_base\_event\_log当日定分区。

1. dwd\_base\_event\_log 🡪dwd\_base\_page\_log
   1. 相关类：analysis;base.function.FilterFunction
   2. 整体逻辑：根据dwd\_base\_event\_log内的eventid字段过滤出页面浏览事件，根据事件规格进行event字段的展开，将其插入dwd\_base\_page\_log中。
2. dwd\_base\_event\_log 🡪 dws\_flw\_agg\_s
   1. 相关类：analysis;base.function.SessionAggFunction
   2. 整体逻辑：根据dwd\_base\_event\_log内的sessionid和维度列进行分组聚合，根据组内时间最大值和最小值确定开始结束时间和访问时长，组内eventid为访问页面的数目为访问页面数，并将其离散化处理转化为范围。
3. dws\_flw\_agg\_s 🡪 dws\_flw\_agg\_u
   1. 相关类：analysis.base.function.UserAggFunction
   2. 整体逻辑：根据dwd\_flw\_agg\_s表内的guid和相关维度进行分组聚合，同时求和组内的单次使用时长为日使用时长，求和单次页面访问数为日总访问数、组内session数为日启动数。
4. dws\_flw\_agg\_u 🡪 dws\_agg\_usr\_cube
   1. 相关类：analysis.flow.function.AggUsrCubeFunction
   2. 整体逻辑：
      1. 先将来自表dws\_flw\_agg\_u的dt = '$currentDate'(时间等于当前日期)的view\_num（启动次数）,view\_time（访问时长）,pv\_num（页面访问数）通过guid,version,channel聚合进行sum求和，之后通过guid,version,channel 进行cube，之后将guid, version, channel，sum(view\_num) view\_num, sum(duration) view\_time, sum(pv\_num) pv\_num形成一个临时视图，
      2. 之后读取临时视图，建几个udf函数将sum(view\_num) view\_num, sum(duration) view\_time, sum(pv\_num) pv\_num转换成范围，之后将guid, version, channel连同三个范围start\_num\_range,duration\_range, pv\_num\_range一起插入到目标表dws\_agg\_usr\_cube
5. dws\_flw\_agg\_u 🡪 dws\_apl\_dau\_rec
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.apl.function.DauRecFunction
   2. 整体逻辑：从当天的dws\_flw\_agg\_u表中选出相应的字段插入到dws\_apl\_dau\_rec表中即可。
6. dws\_agg\_usr\_cube 🡪 dws\_agg\_wstart\_cube
   1. 相关类：analysis.flow.function.AggWStartCubeFunction
   2. 整体逻辑：
      1. 先将来自表dws\_agg\_usr\_cube的trunc(dt, 'week') = '$currentWeek'(trunc用来求出dt时间的周一的日期，看等不等于我们所求的星期的周一的日期)的view\_num（启动次数）通过guid,version,channel聚合进行sum求和，之后将guid, version, channel，sum(view\_num) view\_num形成一个临时视图，
      2. 之后读取临时视图，建一个udf函数将sum(view\_num) view\_num转换成范围，之后将guid, version, channel连同一个范围start\_num\_range一起插入到目标表dws\_agg\_wstart\_cube的分区PARTITION(dt = '$currentWeek')
7. dws\_agg\_usr\_cube 🡪 dws\_agg\_mstart\_cube
   1. 相关类：analysis.flow.function.AggMStartCubeFunction
   2. 整体逻辑
      1. 整体逻辑：先将来自表dws\_agg\_usr\_cube的trunc(dt, 'MM') = '$currentMonth'的view\_num（启动次数）通过guid,version,channel聚合进行sum求和，之后将guid, version, channel，sum(view\_num) view\_num形成一个临时视图，
      2. 之后读取临时视图，建一个udf函数将sum(view\_num) view\_num转换成范围，之后将guid, version, channel连同一个范围start\_num\_range一起插入到目标表dws\_agg\_mstart\_cube的分区PARTITION(dt = '$currentMonth')
8. dws\_apl\_dau\_rec 🡪 dws\_apl\_hsu\_rec
   1. 相关类：analysis.apl.function.HsuRecFunction
   2. 整体逻辑：
      1. 把dws\_apl\_dau\_rec表中今天的数据选出来，然后按照 guid, version, channel 聚合，并且根据 version，channel 做出一个 cube
      2. 选出 dws\_apl\_hsu\_rec 表中前一天的数据，与上面的 cube 做全连接，更新用户的 firstLoginDate（如果是首次登录）和 lastLoginDate（如果当日活跃），将更行后的表写入到新的分区中。
9. dws\_apl\_hsu\_rec  dws\_apl\_dnu\_rec
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.apl.function.DnuRecFunction
   2. 整体逻辑：
      1. 把dws\_apl\_dau\_rec表中今天的数据选出来，然后按照 guid, version, channel 聚合，并且根据 version，channel 做出一个 cube
      2. 选出 dws\_apl\_hsu\_rec 表中前一天的数据，与上面的 cube 做全连接，更新用户的 firstLoginDate（如果是首次登录）和 lastLoginDate（如果当日活跃），将更行后的表写入到新的分区中。
10. dws\_apl\_dau\_rec 🡪 dws\_apl\_uca\_rec
    1. 相关类：analysis.apl.function.UpdateUCARecFunction
    2. 整体逻辑：
       1. 约定活跃区间记录表中结束日期为‘9999-12-31’为一直活跃至今。
       2. 选取当日的dws\_apl\_dau\_rec日活记录表的guid和相关维度的grouping sets与昨日的活跃区间表中昨日活跃的记录进行全连接。
       3. 类型1：全连接结果中日活记录guid为空，活跃区间记录表不为空，即当日未活跃，将其的结束日期设为昨日。
       4. 类型2：全连接结果中日活记录guid不为空，活跃区间记录表为空，即当日为连续活跃的第一天，在记录表中新增一条开始日期为当天，结束日期为‘9999-12-31’的记录。
       5. 类型3：全连接结果中两表内容均不为空，为持续活跃，保留其记录。
       6. 类型4：上日的连续活跃区间记录表中结束日期不为最大值的，代表不可能更改的记录。
       7. 将上述四种类型的数据做Union，得到今天的连续活跃区间记录表。
11. dws\_apl\_uca\_rec 🡪 dws\_apl\_usr\_uca
    1. 相关类： analysis.apl.function.UserUCAFunction
    2. 整体逻辑：从当日更新过的用户连续活跃区间表内选择当天活跃的用户记录，根据区间起始日期和当天日期确定每个用户连续活跃的天数，并增加对天数经过离散化处理的列将具体天数转化为范围类别。
12. dws\_apl\_uca\_rec 🡪 dws\_apl\_itv\_agu
    1. 相关类：
       1. analysis.apl.function.IntervalUserAggFunction

负责实现整体的业务逻辑

* + 1. analysis.apl.udf.StringConcatUDAF

自定义的用于有序拼接字符串的UDAF

* + 1. analysis.apl.udf.IntervalUDTF

自定义的用于间隔天数提取的UDTF

* 1. 整体逻辑：
     1. 筛选结束日期距今30天内的记录。
     2. 将同一标识（guid，channel，version）的记录进行分组，组内每条开始日期与结束日期进行拼接（startDate+‘~’+endDate）然后利用自定义的UDAF对拼接结果进行有序聚合（start1~end1，start2~end2…）。
     3. 将聚合后的每条结果利用自定义的UDTF进行转换，输入聚合后的日期字符串，然后先按照’,’进行split，得到的数组内每个元素为start~end，然后end-start的天数即为使用间隔为0天的人数。
     4. 然后按照’~’进行split，对于数组中每个元素按照‘,’进行split，筛选其中元素长度大于1的，然后得到每个元素为（end1，start2）将其转化为（start2-end1-1,1）的格式，其中start2-end1-1为非0的登录间隔，最后在进行reduceBykey，得到的就是统一用户非零间隔的天数的求和。然后件间隔天数离散化转为新增范围字段。

1. dws\_agg\_wstart\_cube 🡪 ads\_flw\_wstart\_cube
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.flow.function.FlwWStartCubeFunction
   2. 整体逻辑：先将来自表dws\_agg\_wstart\_cube的trunc(dt, 'week') = '$currentWeek'(trunc用来求出dt时间的周一的日期，看等不等于我们所求的星期的周一的日期)的COUNT(DISTINCT guid)通过guid,version,channel聚合进行count计数，之后将guid, version, channel, start\_num\_range, COUNT(DISTINCT guid)一起插入到表ads\_flw\_wstart\_cube的分区PARTITION(dt = '$currentWeek')
2. dws\_agg\_usr\_cube 🡪 ads\_flw\_dstart\_cube
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.flow.function.FlwDStartCubeFunction
   2. 整体逻辑：先将来自表dws\_agg\_usr\_cube的dt = '$currentDate'的COUNT(DISTINCT guid)通过version, channel, start\_num\_range聚合进行count计数，之后将version, channel, start\_num\_range, COUNT(DISTINCT guid)一起插入到表ads\_flw\_dstart\_cube的分区dt = '$currentDate'
3. dws\_agg\_usr\_cube 🡪 ads\_flw\_dduration\_cube
   1. 相关类：

cn.edu.neu.titan.titanSpark.analysis.flow.function.FlwDDurationCubeFunction

* 1. 整体逻辑：先将来自表dws\_agg\_usr\_cube的dt = '$currentDate'的COUNT(DISTINCT guid)通过version, channel, duration\_range聚合进行count计数，之后将version, channel, duration\_range, COUNT(DISTINCT guid)一起插入到表ads\_flw\_dduration\_cube的分区dt = '$currentDate'

1. dws\_agg\_mstart\_cube 🡪 ads\_flw\_mstart\_cube
   1. 相关类：analysis.flow.function.AggMStartCubeFunction
   2. 整体逻辑：先将来自表dws\_agg\_mstart\_cube的dt = '$currentMonth'的COUNT(DISTINCT guid)通过guid,version,channel聚合进行count计数，之后将guid, version, channel, start\_num\_range, COUNT(DISTINCT guid)一起插入到表ads\_flw\_mstart\_cube的分区PARTITION(dt = '$currentMonth')
2. dws\_flw\_agg\_u 🡪 ads\_usr\_start\_cube
   1. 相关类：analysis.flow.function.StartCountFunction
   2. 整体逻辑：将dws\_flw\_agg\_u表中当日数据根据指定维度的grouping sets进行分组。每组内对view\_num进行求和得到本分组的启动数。
3. dws\_flw\_agg\_s 🡪 ads\_flw\_page\_cube
   1. 相关类：analysis.flow.function.PageCountFunction
   2. 整体逻辑：将dws\_flw\_agg\_s表内当日数据根据指定维度的grouping sets((pv\_num\_range),(pv\_num\_range,version),(pv\_num\_range,channel),(pv\_num\_range,version,channel))进行分组，对组内的记录进行count(1)得到的即为单次访问页面数在该范围内的启动数。
4. dws\_flw\_agg\_s 🡪 ads\_flw\_sduration\_cube
   1. 相关类：analysis.flow.function.SDurationFunction
   2. 整体逻辑：将dws\_flw\_agg\_s表内当日数据根据指定维度的grouping sets((duration\_range),(duration\_range,version),(duration\_range,channel),(duration\_range,version,channel))进行分组，对组内的记录进行count(1)得到的即为单次访问时长在该范围内的启动数。
5. dws\_apl\_hsu\_rec 🡪 ads\_apl\_wnrt\_rec
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.retention.function.DnrtRecFunction
   2. 整体逻辑：
      1. 从历史表中选出最后活跃日期为当天的用户的 guid, version,channel，以及通过lastLoginDate - firstLoginDate来计算出来的新用户留存天数 nrt\_days
      2. 再根据 version, channel, nrt\_days 做聚合，计算 count(distinct guid) 来求出每个版本、渠道、nrt\_days 下的留存用户数，将其插入到 ads\_apl\_dnrt\_rec 表中。
6. dws\_apl\_hsu\_rec 🡪 ads\_apl\_dnrt\_rec
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.retention.function.WnrtRecFunction
   2. 整体逻辑：
      1. 从历史表中选出lastLoginDate为本周的 guid,version,channel，以及通过自定义的 udf 函数来计算 weeksBetweek(lastLoginDate, firstLoginDate) 即 nrt\_weeks
      2. 再根据 version,channel, nrt\_weeks做聚合，计算 count(distinct guid)来求出每个渠道、版本、nrt\_weeks 下的留存用户数
7. dws\_apl\_hsu\_rec 🡪 ads\_apl\_mnrt\_rec
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.retention.function.MnrtRecFunction
   2. 整体逻辑：
      1. 从历史表中选出lastLoginDate为本周的 guid,version,channel，以及通过自定义的 udf 函数来计算 monthsBetweek(lastLoginDate, firstLoginDate) 即 nrt\_weeks
      2. 再根据 version,channel, nrt\_weeks做聚合，计算 count(distinct guid)来求出每个渠道、版本、nrt\_months 下的留存用户数
8. dws\_apl\_dnu\_rec 🡪 ads\_usr\_dnu\_cube
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.apl.function.DnuCubeFunction
   2. 整体逻辑：从ads\_apl\_dnu\_rec中选出当天的数据，分别根据将provinceId, os, resolution, model, carrier, network 与 version, channel 做聚合，形成一个 cube表
9. dws\_apl\_dau\_rec 🡪 ads\_usr\_wau\_cube
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.retention.function.DartRecFunction
   2. 整体逻辑：因为该表要从一行数据变成多行数据，所以计算逻辑比较复杂

i. 选出当天活跃用户的 guid,version,channel

ii. 将选出的当天数据与 dws\_apl\_uca\_rec 做内连接，选出该用户在以往所有的活跃区间(currentDate - startDate <= 31)，得出的结果形式为 version,channel,startDate,endDate

iii. 将每一个条数据都展开，例如 2020-06-01 ~ 2020-06~25，那么就会展开成 25 条数据每条数据都有其对应的留存天数，例如 1号留存：30 天，2号留存 29天...25号留存：5天，将其保存成 dt, version, channel, art\_days 的格式

iv. 根据 dt, version, channel, art\_days 做聚合，并且 count(1) 就能算出每天，对应渠道，版本，留存天数下的人数

1. dws\_apl\_dau\_rec 🡪 ads\_usr\_dau\_cube
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.apl.function.ActiveUserCountFunction
   2. 整体逻辑：从dws\_apl\_dau\_rec中选取当天数据进行按维度划分grouping sets进行分组聚合，count（distinct guid）即为活跃人数。
2. dws\_apl\_dau\_rec 🡪 ads\_usr\_mau\_cube
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.apl.function.ActiveUserCountFunction
   2. 整体逻辑：同26选取的为本周内的数据。
3. dws\_apl\_uca\_rec 🡪 ads\_apl\_wart\_rec
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.apl.function.ActiveUserCountFunction
   2. 整体逻辑：同26选取的为本月内的数据。
4. dws\_apl\_uca\_rec 🡪 ads\_apl\_dart\_rec
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.retention.function.MartRecFunction
   2. 整体逻辑：该任务为计算留存用户的活跃月数，与计算留存天数有很大不同，如：一个用户在一个月内有两个活跃区间，如果用计算留存天数的方法，就会产生重复计算，而且还有可能会有区间交叉的情况。

i. 选出本月活跃过的用户 guid, version, channel,并且做去重处理（第一次去重）

ii. 选出本周与 dws\_apl\_uca\_rec 做内连接，选出该用户在以往活跃区间的起始周以及结束周，结果形式为 guid, version, channel, startDate, endDate

iii. 把每一条数据展开，形成 dt, guid, version, channel 的形式，此时这种数据中会出现数据交叉，所以再一次进行 dinstinct 处理（第二次去重）

举例：对于同一guid,版本，渠道，在连续活跃区间中有：05-28~06-03, 06-05~06-15, 06-20~06-27, 06-29~07-01，那么转成 month 之后就是：05-01~06-01，06-01~06-01，06-01，06-01，06-01~07-01，此时在df中的结果就是: (guid, version, channel, 05-01)，(guid, version, channel, 06-01),(guid, version, channel, 06-01),(guid, version, channel, 06-01),(guid, version, channel, 07-01)，所以需要去重，生成 (guid, version, channel, 05-01)，(guid, version, channel, 06-01),(guid, version, channel, 07-01)

iv. 根据 dt 利用 monthbetween 函数计算留存月数 art\_month，再根据 dt, version, channel, art\_month，做聚合，并且 count(1) 来计算各月各版本、渠道留存月数对应的人数。

1. dws\_apl\_uca\_rec 🡪 ads\_apl\_mart\_rec
   1. 相关类：cn.edu.neu.titan.titanSpark.analysis.retention.function.WartRecFunction
   2. 整体逻辑：与活跃用户月留存的计算方法类似，把月换成了周
2. dws\_apl\_usr\_uca 🡪 dws\_apl\_usr\_uca
   1. 相关类：analysis.apl.function.UCACountFunction
   2. 整体逻辑：从dws\_apl\_usr\_uca表里面选择当日且version，channel为空的数据，根据连续活跃期间范围进行聚合，count（1）即为该范围内人数。
3. dws\_apl\_itv\_agu 🡪 dws\_apl\_itv\_agu
   1. 相关类：analysis.apl.function.IntervalCountFunction
   2. 整体逻辑：从dws\_apl\_itv\_agu选取当日数据，根据维度(version，channel)和间隔范围进行聚合，count(1)即为人数。

33-55. 迁移任务，ads层到mysql数据库

1. 相关类：migration包内所有类。
2. 整体逻辑：将mysql维度所需维度做笛卡儿积，与ads中表做左外连接进行补零操作，然后将日期设为当日日期。