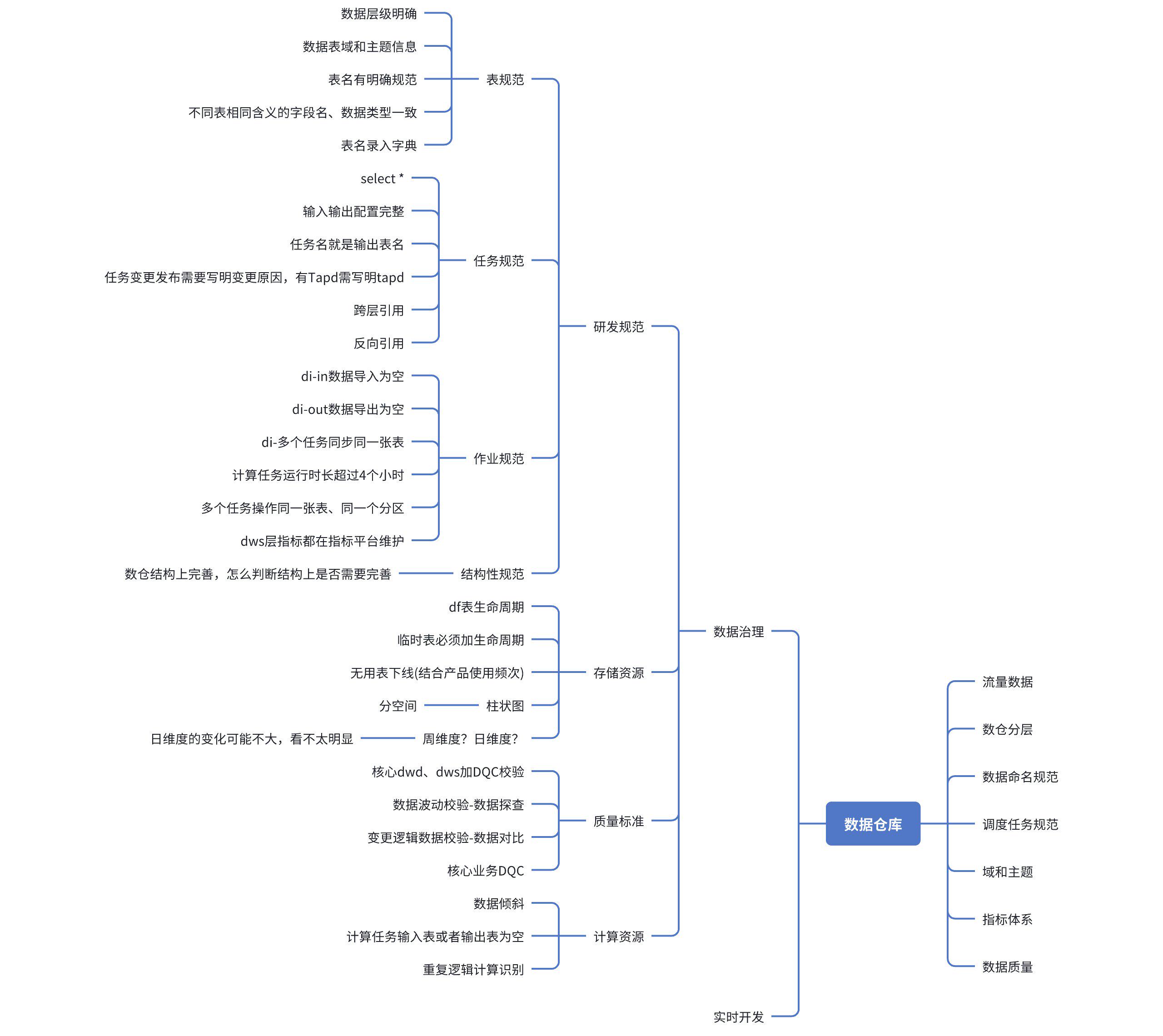
**数据仓库工作记录**



**一、流量数据怎么做**

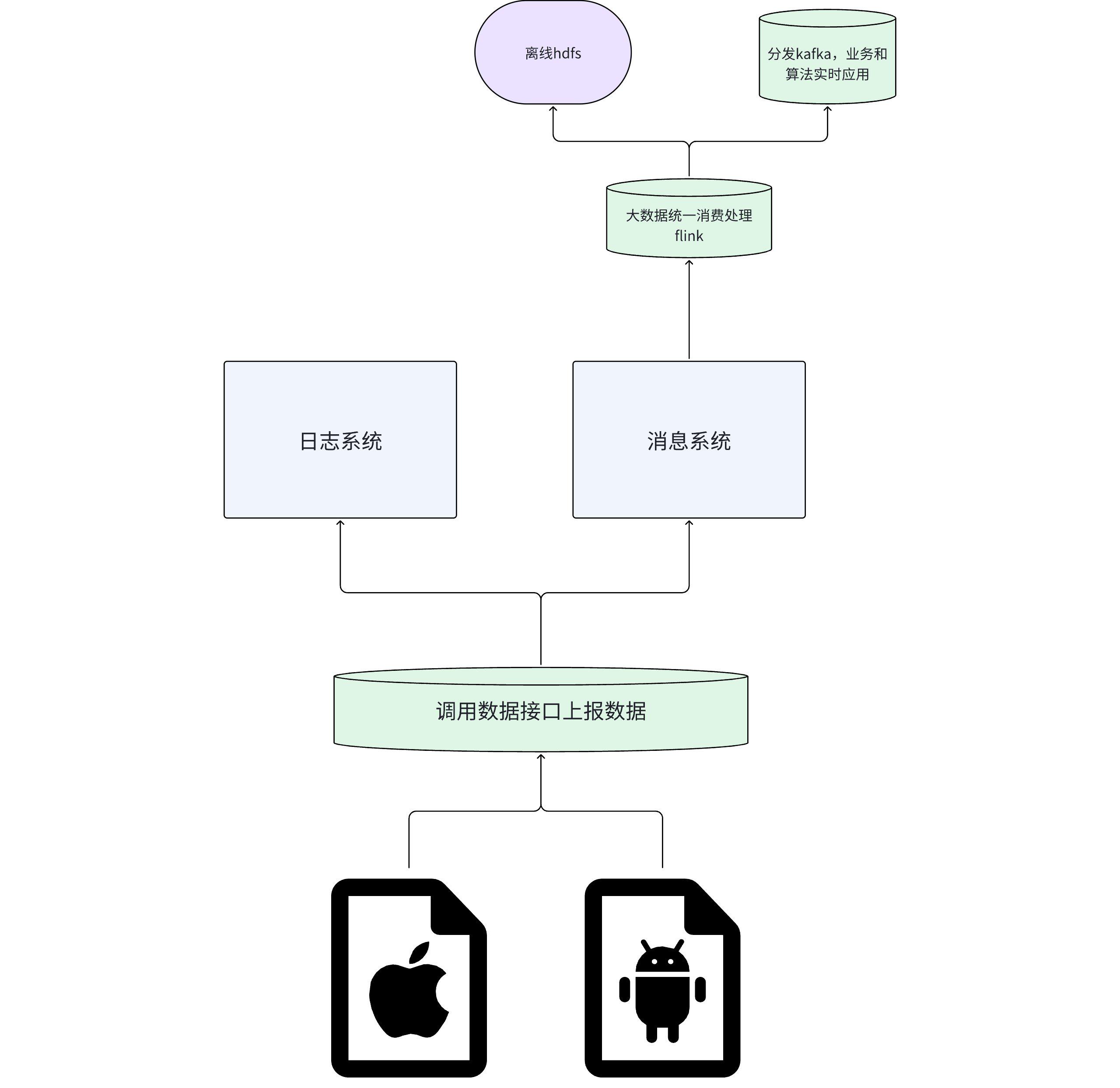
**1、客户端埋点数据**

三个重要前提：

1. 客户端能力决定了流量数据质量的上限
2. 埋点方案决定了流量数据使用的易用性
3. 流量数据是数仓工作中相对复杂、相对技术性的

**1.1 流程示例**

简单来说，就是客户端采集数据上报，大数据部门处理之后做分析和应用；实际的工作中，日志数据有可能是通过接口上报到网关(中间件团队)的，中间件再分发一次，一份可能抽样到日志系统，一份发到消息中间件kafka，供多部门使用



实际不同公司之间会有差异，有的小一点的公司可能没有中间件这一套转发，客户端直接到日志系统也有。

还有客户端数据内容也会分多种，会有客户端的性能埋点，业务埋点等

**1.2 全埋点方案？人工埋点？可视化埋点？**

现在一搜埋点方案一般都写这三种，我没做过前端，不做深入分析；实际工作中见到的全是人工埋点，下面都基于人工埋点介绍

* 优点：每个埋点都是需要开发，因此每个埋点都可以带很多业务信息，数据可靠性高，接收到的埋点数据价值巨大
* 缺点：工作量巨大，埋点代码版本管理困难；实际很可能会iOS、Android端不一样；新老版本不一致

**1.3 埋点平台**

埋点平台有两大功能：

一个是埋点定义，工作流程上一般是产品发起埋点需求，**确认埋点位置和内容**，提交设计稿和埋点文档到平台上，需求到客户端开发，上线之前测试还有可能通过平台来验证埋点准确性。

另一个是数据查看，产品定义完之后基本都需要回收数据，可以通过自己埋点的位置直接看到数据波动情况，同时，对于业务同学看埋点数据更加便利，通过定义信息能看到uv、pv等数据。

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

**1.4 埋点需要埋什么**

用户张三(**who**)在什么时间(**when**)什么位置(**where**)干了什么(**what**)

**where**，怎么定义位置是很重要的，一旦定义在很长周期都需要按照这个方式。同时，一个好的定义方式方便业务和技术沟通

目前见的最多的还是阿里的**spm**系统https://open.taobao.com/doc.htm?docId=959&docType=1；通过4段定义了一个外部位置

参照spm可以自己做拓展，几段都可以，符合自己的应用场景最好；比如第一段可以作为端的标记，有的公司可能会有很多app，第一个段可以定义端，第二段定义页面信息，第三段定义区域信息，第四段定义位置信息，第五段定义位置信息。

**What** 普遍是只记录当前行为，同时会附加**很多业务参数**，比如用户操作的时候知道是哪个位置，还需要知道操作的是哪个信息，以滴滴为例，我点击了订单列表的 再来一单

那要记录下我当前这一单的订单号、金额等一些业务信息，这些普遍是后端通过接口给到前端的，所以很多时候埋点需求牵涉人多，逻辑复杂。



**1.5 埋点类型**

埋点类型就是对埋点分类，最常见的就是分为**点击和曝光**，也会有把页面曝光和停留时长等行为单独拆开的。同一个位置的点击和曝光，如果客户端是分开上报的那么位置信息客户端怎么保障名称一致是个需要注意的问题；最好一开始定义的时候就提好各种类型的诉求，客户端位置定义是同一套，上报的时候区分

**1.6 埋点数据可以干啥呢**

1、简单uv、pv

2、行为链路分析；比如找注册链路上流失最多的一个步骤；行为链路最需要优化的一个行为

3、行为时长分析

4、留存分析等(更多的也可以参照神策的10大分析)

**1.7 埋点数据有哪些经验可以分享**

1、ods层的埋点数据落库最好就是一个json对象，就一个字段，长周期即使数据变更了也能用一个表

2、位置信息可以跟离线的二级分区对应起来，用业务分区对于下游的查询有较大优化，但是数量一定要有限制，过少就没有价值，过多后续维护和元数据管理也是问题

3、埋点下线最好提前有规范，埋点只增不减，存储和计算都是成本

4、离线和实时的架构要提前想好，什么时候做逻辑拆分，什么时候做维度打宽，实时和离线怎么保障最大限度的结构一致性

5、日增量数据超过20亿就可以考虑分表了，场景分表和二级分区不冲突

6、客户端最好保证一套上报机制，比如点击和曝光，只是行为不同，业务参数上报保持一致

7、工作流程上要确定好，要先定义、测试再上报；不定义就上报的要不要给单独踢出来

8、dwd层字段解析常用和通用信息，比如用户相关信息(设备号、user\_id，机型、型号，品牌)，操作时间等，其他的统一放一个字段

**2、pc端埋点数据**

pc端上报的埋点规范上可以跟客户端一致，位置定义，业务参数等可以一样

如果是公司后台页面这些页面，这些数据分析和展示最好和app段分开

**3、后端单独上报的日志数据**

后端一般上报的是业务相关日志，形式多样，内容丰富，这个不经过大数据埋点定义也可以。但是对应的大部门最好有统一的规范，可以通过sdk实现，比如算法工程采用同一套sdk，可以有效保障数据不会过分混乱。

**4、具体工作案例分享**

之前工作中做过一个比较大的流量相关的数据项目，效果也比较好，主要做了几个方面的工作，一个是主要的解析逻辑从离线搬到实时上去，另一个是做了日志分流工作，同时把离线和实时表对应起来。

**4.1 背景**

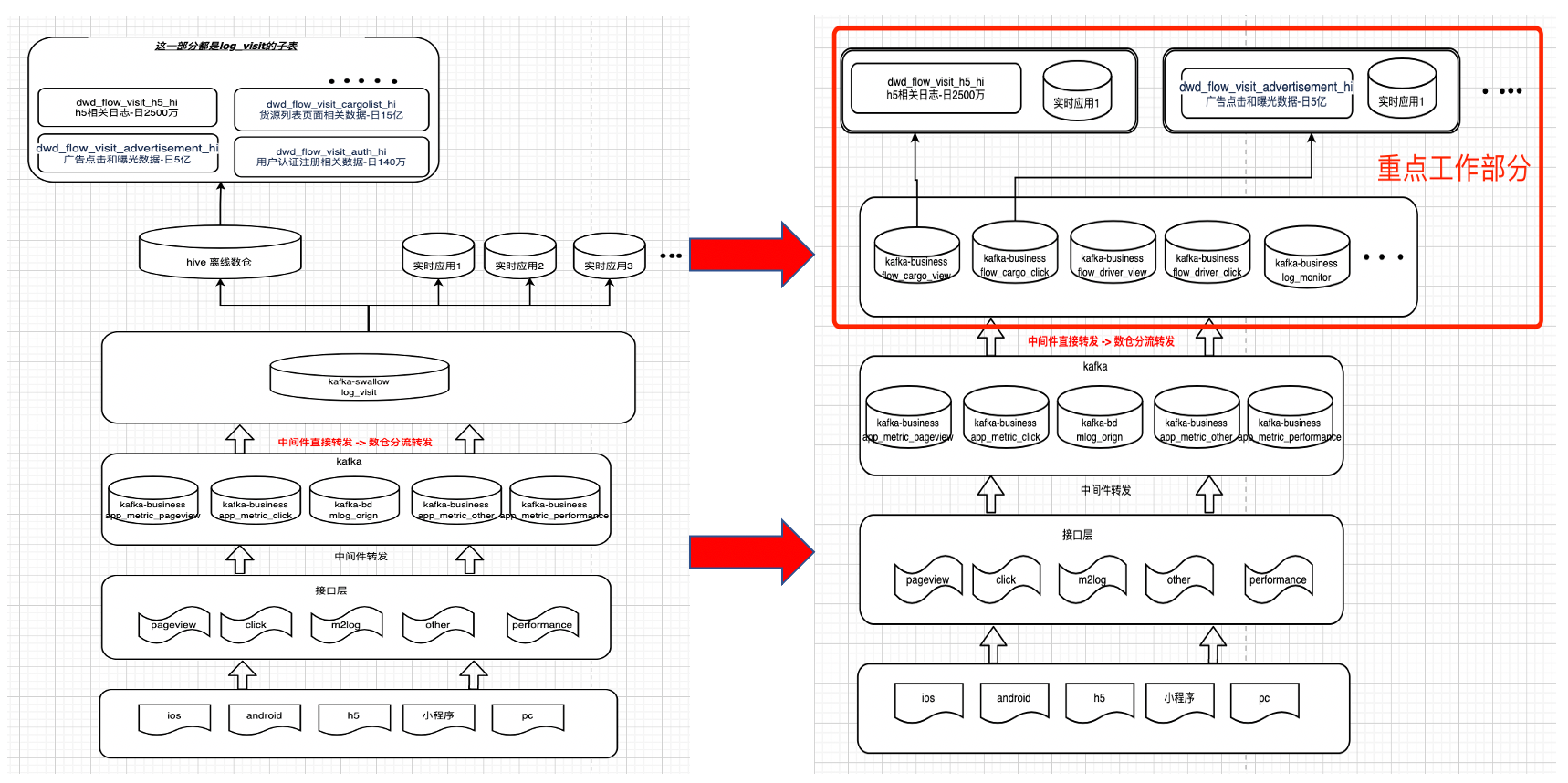
我刚接手的现状是这样的，流量是有完整的主题域的，数据分层结构也清晰，客户端埋点数据是中间件先处理一次发到kafka，数据部门这边通过一个flink任务消费kafka到hdfs文件目录上，也就是对应的ods表。数仓在ods表上做了2个分支到dwd，一个是货源曝光(主要有lateral view)，一个是其他。其他这个上又做了多个子表，主要是为了减轻下游使用时查表压力。任务结构采用小时增量调度，同样是出于性能考虑，每天流量端整体增量数据量大概1000亿，客户端日志大概处理100亿，离线用hive处理，spark也可以用，但是任务容易失败。流量数据计算有单独队列，占公司离线资源的12%。

出于几个原因，这个结构需要调整，一个是离线调度资源太紧张了，离线计算性能也差，公司还搞降本增效，小时调度基本都跑不完；另一个原因是，ods消费的kafka是由中间件处理过的，他们不想维护，且他们那性能到了瓶颈，高峰期经常有延迟；还有一个情况是客户端日志是很多部门大家都用的，有时候只想用中间很小的一个埋点数据，但是这个topic是统一的，需要很大的资源，且这个topic流出流量巨大，有10个消费方，高峰大概30万的qps，运维迫切想让我们拆分下这个topic，且下游经常会觉得离线表使用成本高，随便一扫一天就是四五十亿的数据，从中取出来的数据大概只有百分之一这样，小时分区利用率不高，我们也想改善下情况。

**4.2 具体处理**

1. 把中间件处理逻辑搬运到大数据部门，一些处理逻辑能够前置
2. 配合运维部门把日志的topic给拆分，我们确定下拆分逻辑
3. 离线的结构做对应调整，无用的分表给下线掉，新建分表提效(主要是通过二级分区实现，维表配置，实时读取)
4. 离线的拆分逻辑全部搬运到实时上去，主要是字段逻辑处理，大部分方法需要自己实现

前后结构对比：



**4.3 实际结果**



上面罗列的几项整体都有实现，提前构想的都也做到了。但是实际性能不如开始构想的那么好。

从接口到kafka这一部分，也就是中间件处理的这一部分，一开始认为用flink肯定会比纯java代码好很多，分布式框架肯定优化很多，但是实际并没有，单从cpu核数对比还高很多。还有就是大型任务比较难调整，这一部分共有5个source，加起来100万的qps，开始是完全按照中间件逻辑做的逻辑处理，后面发现任务太难调，拆成了5个任务，好很多。

离线函数实时用的时候需要重新实现，比如coalesce，但是基本都能参考下hive的函数实现，麻烦但不是很难。

**二、数仓分层**

分层是数仓常见策略，目的有很多，分层开发结构会更清晰，一些处理逻辑能够复用，数据血缘关系更容易追溯；与之对应的是域和主题做纵向切分。两者共同作用把数据切分为方格状。

分层总结来说有以下几个好处

1、清晰的数据结构，每一层代表不同的数据逻辑，可以满足更多的需求

2、复杂问题简单化，每一层处理固定的逻辑，有问题从固定层级一级一级排查

3、减少重复开发，在根据业务过程内聚的模型中，大家会相互引用，引用次数就代表了模型的复用性

4、数据链路明确，对于查找问题，追踪数据引用比较方便。

常见是分为四层，ods -> dwd/dim -> dws -> ads这个引用关系

**1、数据贴源层(ods-Operational Data Store)**

跟原始业务库保持一致，把业务系统数据无处理地拉到数仓系统(一般会做分区操作，空值统一转null)。方便从源头查找数据问题，也方便业务规模大规模查数据；一般通过数据同步工具从业务数据库或者消息队列离线拉取或实时消费到大数据这边。需要考虑的是数据归档的问题，一个表出于业务结构的考虑会把数据拆分，可能会基于创建时间，也可能是基于数据中的状态。任务形态会使用中间层，可能会用mid，也可能会用tmp，然后合并为ods层。

**2、事实明细层(dwd-Data Warehouse Detail)**

dwd层在ods之上，对业务过程进行表达，大多数互联网公司使用的是**星型模型，**也就是一个事实表围绕多个维度数据，事实表一般用dwd开头表述，维度表开头用dim。dwd层表一般表达某个业务过程，比如生成订单、页面上点击、后台产生一个请求；生成订单时商品名、商品价格、商品位置等这些都算是维度信息，表达的是交易相关的一些状态信息，以便于更全面分析公司业务。跟我们现实生活还是比较相符的，小王今天去超市买促销商品，购买就是事实，促销就是商品的维度信息，更多的就可以分析到，小王可能是价值敏感性用户，小王可能是上班族等。

dwd层会做一系列加工，首先就是会做维度退化，会把维度数据维护进事实表中，减少下游事实表和维度表的关联；还会做字段命名规范，比如性别字段，叫gender还是sex，字段值有空值也有null值，统一处理；还有就是状态相关的，来源可能是多个，数仓需要统一成一个标准；还有可能有重复数据、测试数据，都要想办法过滤掉；还有就是字段类型，比如时间字段，有可能存string类型，有可能存timestamp，在使用的时候各种兼容。

dwd层是日常工作中占比较高的一层，也是数仓主要产出的部分，虽然简单描述是在做"大宽表"，但是能和业务结合做出通用的表，同时多个细节考虑到维护稳定的质量，让下游使用顺畅，还是需要花心思的，真的需要对数据熟悉，同时能做好各种预发问题兼容。

**3、事实汇总层(dws-Data Warehouse Summary)**

dws层在dwd和dim层之上，对某个业务过程按照一定维度进行汇总。dws层主要有几个作用，一个是对多个业务过程抽象，多个数据都放打平放在一个维度上，比如电商中的商品曝光、点击、成交，dws会把3个过程放在一个dws表；另一个是对常用维度汇总，减少数据存储量，下游也减少数据计算量。

这里也有看到有的博主分享的时候会在dwd和dws之间会多一个dwm(**Data Warehouse Middle**)层，比如上面曝光、点击、成交3个过程表，会首先从3个明细表汇总产出3张表，然后再拼接为一张dws表，这3张过程表可以认为是dwm，但是如果是通用型的数据，dws也无可厚非。这个小规模数仓个人认为都可以忽略这一层。

**4、数据应用层(ads-Application Data Store)**

ads在dws/dwd/dim层之上，也是数仓最上面的一层。一般以应用分类来区分。ads层数据一般用于直接分析或者展示，已经是高度聚合的数据，这个数据可能会用于直接查询，也可能会同步到外部系统供业务方使用，也可能同步至olap系统比如doris这种系统供快速分析。

从数据表的存储形式来分类，常见的有四种类型

1. 增量表，主要是事实表，每个分区记录当前分区下新增的数据，分区可能是各种时间周期
2. 全量表，记录当前所有的数据，一般维度表会这么做(跟小伙伴交流，也有没有这种类型的，全部都是分区表)
3. 快照表，分区内记录当前所有的数据，方便数据追溯，分区可能是各种时间周期，全量表每个分区记录就是快照表
4. 拉链表，记录当前粒度数据从历史到现在的所有状态，类似快递的物流信息，有位置变化的详细数据，好处是在节省存储的情况下记录下历史的所有状态数据，方便查找数据问题。实际工作中只搞过一两次，对于大多数数据，直接就df每日存快照了，拉链表也不好用，节省的存储也有限。

**需要关注的问题：**

**1、数据分层之后就会有跨层引用问题，还需要注意是否有反向引用问题，分层是为了更有层次地建设，反向引用会让结构更加混乱。**

**2、dwd、dws层是否需要冗余维度信息**

根据场景，大多数中小规模公司都会冗余以节省查询性能。冗余的维度信息同样需要考虑缓慢变化维问题。

**3、dim层缓慢变化维问题SCD（Slowly Changing Dimensions）**

可以采用的方式：

* 只保留最早的一份数据
* 只保留最新的一份数据
* 增加维度新行或者新列
* 每日存储快照

首先，维度变化是肯定会面对的问题，**每日存储快照是最常见的方式-df**，每日一个版本可以用于数据追溯和问题排查。且一般维度表较小，存储成本不是大问题，实际工作中没见过专门做拉链表处理这个问题的。对于每个变更都需要关注的，更建议用监听binlog方式存下所有快照。

**4、分层建设需要关注模型使用情况**

实际工作中，也需要关注分层的复用性，建设出来的dwd和dws表是不是足够内聚，大量的查询是否都走到了dws，dws层级数据是否能各个汇总粒度能保持一致。或者说分层是数仓建设的一个壳，实际能做到内聚和耦合且好用才是数仓的灵魂。

**三、数据域和主题**

域和主题在结构上是对业务过程纵向切割，切割之后模块内方便内聚建设，同时也方便数仓人员分工。

数据域是面向业务分析，用来表示数据仓库中的不同业务领域或主题领域。每个域代表了企业中的一个特定方面或功能区域。数据域的划分通常与企业的业务结构和运营模式紧密相关，将业务过程进行抽象，常见的数据域有基础数据域、订单域、生产域等，主题在域下。实际工作中，也见到有不区分业务域，全都只用主题。

主题是一个特定的分析主题，通常与一个或多个业务流程相关联。每个主题集中表示了某个特定领域内的关键业务指标和数据。主题的定义和划分通常基于业务需求和数据分析的目标。实际工作中，大部分主题和业务部分的分工分组还是比较接近的，业务部门实际上已经对业务过程进行了区分，有很强的参考意义。常见的划分有流量、订单、用户、营销、公共等。同时如果有一个新业务或者业务线，一般也会先把这个业务块单独划分为一个主题。公司规模足够大，就变得无限可分，可能也会区分公司级主题和事业部主题等。

看到别的博主给出的分类方式：

根据需求方划分：比如需求方为财务部，就可以设定对应的财务主题域，而财务主题域里面可能就会有员工工资分析，投资回报比分析等主题。

按照功能或应用划分：比如微信中的朋友圈数据域、群聊数据域等，而朋友圈数据域可能就会有用户动态信息主题、广告主题等。

按照部门划分：比如可能会有运营域、技术域等，运营域中可能会有工资支出分析、活动宣传效果分析等主题。

**数据域和主题的划分主要体现在ods和dwd、dws层的分工和命名上，**ads层更多是应用层面，应用层用的数据是各个维度的大汇总，跨域和主题非常常见。dws层也常会有跨域和主题的情况；以淘宝为例，商品sku销量的dws层日汇总表，会用到订单域和商品域2部分的数据，那么这个汇总表应该是属于哪个域和主题？实际工作中更多的是根据维度信息所在域或者主题命名，订单这里先生成一个商品维度汇总表(dwb)，然后商品这边做维度和事实的拼接。从公司业务的角度，分工是为了提高员工效率，从数据角度，能真正了解到公司的整体数据链路，通过数据串联起整个公司的业务，***才是从域和主题的"局部最优解"到了公司层面的"全局最优解"***。

域和主题划分示例：



**点击图片可查看完整电子表格**

**四、数据表和字段命名规范、表存储周期设定**

前面总结了下分层和主题建设，这里是对数据在2个层面进行了区分，再加上数据的更新形式构成了表命名重要的三个部分。

规范命名在复杂的组织中可以形成统一，**做到见名知意，对于提高组织内效率很重要**。一个数仓中几十万张表太正常了，对于不熟悉的人使用有很大的困难，统一命名能尽快找到自己的数据。同时表中字段标准命名对于加快对数据的理解至关重要，比如用户ID，如果都叫user\_id，那么做关联等能很快理解到数据，如果有的表叫user\_id，有的叫uid，有的存的string，有的存的bigint，使用的人大概率是要做数据探查对比的，使用体验贼差。

**1、词根设计(指标体系中也会体现)**

词根属于数仓建设中的规范，属于元数据管理的范畴，现在把这个划到数据治理的一部分。完整的数仓建设是包含数据治理的，只是现在谈到数仓偏向于数据建模， 而谈到数据治理，更多的是关于数据规范、数据管理。

表命名，其实在很大程度上是对元数据描述的一种体现，表命名规范越完善，我 们能从表名获取到的信息就越多。比如：一部分业务是关于货架的，英文名是：rack， rack 就是一个词根，那我们就在所有的表、字段等用到的地方都叫 rack，不要叫成 别的什么。这就是词根的作用，用来统一命名，表达同一个含义。**目的就是公司所有相关信息都用同一个名字，降低使用同学的理解成本**。比如有很多“率”的指标，都可以拆解成 XXX+率，率可以叫 rate，那我 们所有的指标都叫做 XXX+rate。

**2、表命名**

**常见命名策略：**

|  |
| --- |
| ods层：ods\_[业务库名]\_业务表名\_分区形式 eg:ods\_pipline\_page\_online\_df  dwd层：dwd\_主题\_词根\_分区形式 eg:dwd\_flow\_shipper\_click\_di  dim层：dim\_主题\_词根\_[分区形式] eg:dim\_pub\_city  dws层：dws\_数据域\_词根\_[汇总周期/分区形式] eg:dws\_bas\_user\_order\_30d  ads层：ads\_[应用分类]\_词根\_[汇总周期/分区形式] eg:ads\_promotion\_deal\_category\_rate (应用分类：可以理解为专题)  tmp层：tmp\_[数据域主题]\_词根 eg:tmp\_binke\_order\_step1 |

**分区形式/汇总周期：**

**这里只代表分区存储的数据形态，不代表分区内数据的更新形式**

|  |
| --- |
| df:daily full 日全量，天分区内存储当前所有数据  hf:hour full 小时全量，小时分区内存储当前所有数据  di:daily increase 日增量，天分区存储当日增量数据  hi:hour increase 小时增量，小时分区存储分区内增量数据  dc:非分区表，daily cover 日全量覆盖，非分区表，存储当前所有数据  hh:非分区表，存储当日数据  30d: 汇总周期为30天内数据  rt:分区表，分区内数据实时更新 |

**3、存储周期设定**

存储周期设定一般会根据层级和数据重要性打标，设定存储周期，一个是重要的数据保留足够时间，同时不过度冗余浪费存储。

**存储周期有这么个意思，"防小人也防君子"，如果没有规定，大多数同学是没有主观意识去设定的，建表的时候不设定，后面就又要数据治理层面去做了，个人认为规范和治理是互斥的，如果规范做的足够好，治理的空间就足够小。建表最好有平台管控，注释、生命周期等必须写好才给建，在源头就做好管控。**

常见策略：



**点击图片可查看完整电子表格**

**4、字段命名**

字段命名最重要的一点就是不同表相同含义的字段尽量保持一致。

字段命名可以借助于元数据管理中的业务词根，这个是为了相同字段使用同一个名称。但是实际操作起来仅仅依靠这个的话不容易，如果是先有中文含义再起英文字段名，翻译还有可能不一样，比如城市名这个字段，是叫city\_name还是就叫city，实践中元数据管理可以解决一点问题，但是更多的还是要借助工具，在建表阶段通过平台解决掉这个问题，具体细节在常用数据产品中会讲到。

**5、常见问题**

**1、dws层表内不同指标汇总周期不一致汇总周期怎么写？**

比如user\_id有一张30日订单汇总数据表，表中可能会有1d、7d、14d、30d的汇总周期，表名后缀按照最长30d的写

**2、dim层表可以没有后缀吗？**

个人理解是可以的，只要规范能保持一致，比如中国行政区划表，几乎可以理解为不会变化的，没有后缀完全可行

**3、事实表可以是非分区表吗？**

不建议这么搞，虽然能这么做。在某个公司有见过这么设定的，同一份数据根据不同后缀会搜到多份，比如订单表，各种后缀的各有一张表，维护成本也高，结构上也很奇怪。

**4、表名需要和数据更新周期绑定吗？**

这里跟数仓架构有一定关系，比如常见的lamda架构，表名和数据更新就不是绑定的；如果是绑定的，在数仓的发展周期中，肯定会先有天任务，然后到小时，然后到实时，就会存在多个结构，数据字段可能不统一，数据还对不上，维护成本高。不建议这么做。

**5、临时表建议强行设定生命周期**

临时表一个是存储成本问题，另一个是数据集市的搜索干扰，不加控制的话，临时表会泛滥的很厉害。这个最好在数仓一开始建设的时候就有设定，后续再治理会面临多种问题。

**五、指标体系**

指标体系如果去搜，能搜出来一大堆的概念，但是方法论一般到实际落地还有很长的路要走。特别是公司如果一开始没有和指标相关的工作流程和建设思路，后续再开始从头建设，十分困难。

指标平台和公司规模有很大关系，公司规模足够大时，指标平台就是大数据平台的必需品，需要有平台来传达业务和数据的关系，这也是指标平台的意义，同时提供可视化的数据查询方式，对于不会写sql的同学比较友好。但是同时我们需要考虑到不同人对于数据的理解能力，对业务的抽象尽量通俗化，最好能跟公司业务过程或者app上的操作串联起来。

基础知识：

1. 原子指标‌：

* 定义：原子指标是有业务统计含义的数值型数据，通过度量加工得到，一般作为口径定义存在，不具备实际物理值。它们是业务定义中不可拆分的指标，如销售金额、库存数量等。
* 举例：例如，销售金额就是一个原子指标，它直接反映了销售活动的量化结果，无法进一步拆分为更小的组成部分。

1. 派生指标‌：

* 定义：派生指标是基于原子指标的扩展，通过添加修饰词、统计周期和统计粒度来形成。派生指标 = 原子指标 + 统计周期 + 业务限定 + 统计粒度。例如，最近一周内某个渠道的销售金额就是一个派生指标，其中“最近一周”是统计周期，“某个渠道”是业务限定，“销售金额”则是原子指标。
* 举例：假设有一个原子指标“订单总额”，那么“最近一周内北京地区的订单总额”就是一个派生指标，其中“最近一周”和“北京地区”是修饰词，共同作用于“订单总额”这一原子指标上。

1. 复合指标‌：

* 定义：复合指标由一个或多个派生指标通过计算而成，可以是简单的算术运算，也可以是更复杂的逻辑运算。复合指标通常用于进行深入的数据分析和业务决策。
* 举例：假设有两个派生指标，“男生购买数量”和“女生购买数量”，那么“总购买数量”就是一个复合指标，通过简单加法运算将两个派生指标合并得到。

**1、指标平台定位**

主要解决两件事，一是能**查看指标定义**，通过sql或者描述信息能知道这个指标对应的业务；二是**指标和数据查询有机结合**，让数据所见即可得，主要是为了统一数据口径

**2、指标平台解决的问题**

* 公司到底有哪些数据（通过数据地图是一个形式，但是看到表你可能并不知道数据的产生逻辑）
* 需求满足慢（组织规模大了之后的必然结果，同事之间沟通效率低）
* 指标对不齐（数据变更快，变更记录不及时，数据来源多、数据处理人员多、口径多，多份数据对不齐）
* 问题排查慢（系统数据出问题，数据加工链路长，碰到人员流动和交接，问题排查慢，修复耗时长）

**3、具体怎么做**

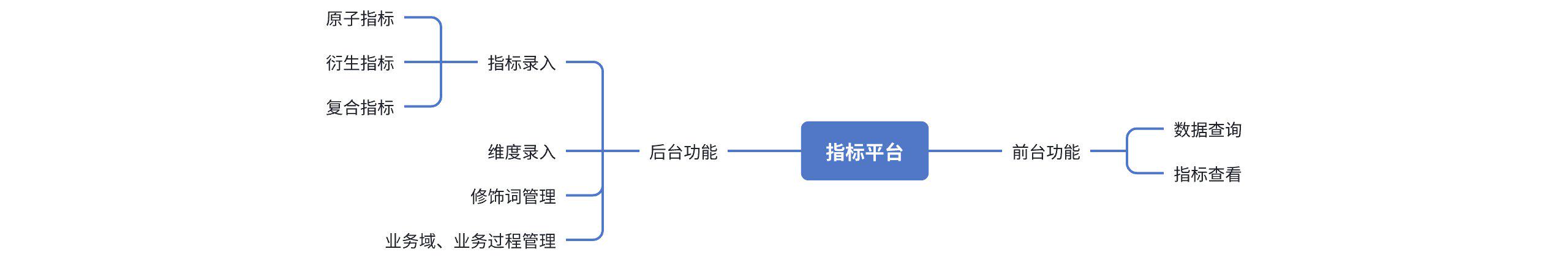
* 具体做之前还是需要思考一下这个产品的形态：

1、内容上这个产品需要有哪些东西？指标录入要填写哪些东西，指标查看展示哪些内容，指标的内容怎么跟查询对应起来

2、实际录入的成本有多高？录入主要是谁来操作，数仓还是数分？录入一个指标大概需要多久，流程过长或者理解困难都会让系统难以维系

3、上线之后怎么评估价值？有多少人查看过定义？使用过定义好的数据做了多少次查询，也可以针对指标平台做二次开发，替换已经存在的日报sql查询。

* 具体的功能模块，标记为黄色的为指标平台必须



**4、根据实际场景分析下我们的功能模块**

**1、简单的日报场景或者报表场景**

这个跟指标平台比较对应，业务一般会提需求，想看某某模块的数据，大多数是这么看的，首先会描述我是什么什么业务的，一般根据组织架构也能知道(这个后续的权限管理也有一定的对应关系)；比如是得物鉴定模块的数据，那么一般会有这么几块的数据

* 模块整体数据，分日期、鉴定的日活、订单数、鉴定结果为真订单数、鉴定为真占比
* 昨日鉴定来源数据，日期，模块入口、入口uv、入口pv、订单数
* 昨日鉴定品牌数据，日期，一级品牌、二级品牌、订单数、鉴定结果为真订单数、鉴定为真占比
* 昨日鉴定类目数据，日期，一级类目、二级类目、订单数、鉴定结果为真订单数、鉴定为真占比

根据此场景，如果没有指标平台，分析或者bi同学做这件事的时候大致是这么个流程，先明确需求，然后找到鉴定的表看下数据，假定是表dwd\_idf\_identify\_order\_di，鉴定的日志表：dwd\_flow\_identify\_click\_di，上面提的数据粒度此表都有维护，那么就是4段sql

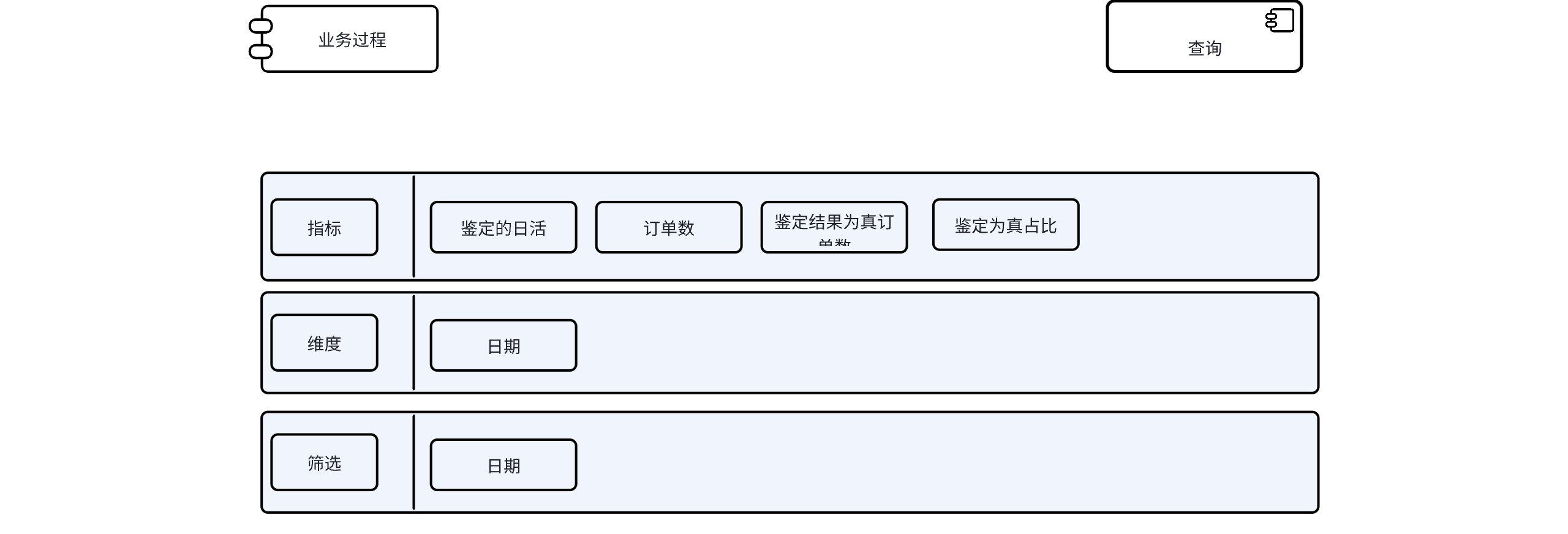
|  |
| --- |
| SQL --模块整体数据 select t1.ds  ,t1.identify\_uv\_1d  ,t2.identify\_order\_cnt\_1d  ,t2.identify\_is\_true\_cnt\_1d  ,t2.identify\_is\_false\_cnt\_1d  ,t2.identify\_is\_true\_cnt\_1d / t2.identify\_order\_cnt\_1d as t2.identify\_is\_true\_rate\_1d from (   select ds  ,count(distinct user\_id) as identify\_uv\_1d  from dwd\_flow\_identify\_click\_di  where ds = {yesterday}  group by ds  ) t1 left  join (   select ds  ,count(1) as identify\_order\_cnt\_1d  ,sum(if(identift\_status = 8,1,0)) as identify\_is\_true\_cnt\_1d  ,sum(if(identift\_status = 9,1,0)) as identify\_is\_false\_cnt\_1d  from dwd\_idf\_identify\_order\_di  where ds = {yesterday}  and order\_status = 1  group by ds  ) t2  on t1.ds = t2.ds  ;      --昨日鉴定来源数据 select t1.ds  ,source\_block\_chi\_name  ,t1.identify\_uv\_1d  ,t2.identify\_order\_cnt\_1d  ,t2.identify\_is\_true\_cnt\_1d  ,t2.identify\_is\_false\_cnt\_1d  ,t2.identify\_is\_true\_cnt\_1d / t2.identify\_order\_cnt\_1d as t2.identify\_is\_true\_rate\_1d from (   select ds  ,source\_block\_chi\_name  ,count(distinct user\_id) as identify\_uv\_1d  from dwd\_flow\_identify\_click\_di  where ds = {yesterday}  group by ds  ,source\_block\_chi\_name  ) t1 left  join (   select ds  ,source\_block\_chi\_name  ,count(1) as identify\_order\_cnt\_1d  ,sum(if(identift\_status = 8,1,0)) as identify\_is\_true\_cnt\_1d  ,sum(if(identift\_status = 9,1,0)) as identify\_is\_false\_cnt\_1d  from dwd\_idf\_identify\_order\_di  where ds = {yesterday}  and order\_status = 1  group by ds  ,source\_block\_chi\_name  ) t2  on t1.ds = t2.ds  and t1.source\_block\_chi\_name = t2.source\_block\_chi\_name  ;     --昨日鉴定品牌数据  select ds  ,first\_brand\_name  ,second\_brand\_name  ,count(1) as identify\_order\_cnt\_1d  ,sum(if(identift\_status = 8,1,0)) as identify\_is\_true\_cnt\_1d  ,sum(if(identift\_status = 9,1,0)) as identify\_is\_false\_cnt\_1d  from dwd\_idf\_identify\_order\_di  where ds = {yesterday}  and order\_status = 1  group by ds  ,first\_brand\_name  ,second\_brand\_name  ;    --昨日鉴定类目数据  select ds  ,first\_category\_name  ,second\_category\_name  ,count(1) as identify\_order\_cnt\_1d  ,sum(if(identift\_status = 8,1,0)) as identify\_is\_true\_cnt\_1d  ,sum(if(identift\_status = 9,1,0)) as identify\_is\_false\_cnt\_1d  from dwd\_idf\_identify\_order\_di  where ds = {yesterday}  and order\_status = 1  group by ds  ,first\_category\_name  ,second\_category\_name  ; |

如果有指标平台，那么这个需求首先需要录入指标，指标平台填写的就是这个



对于订单source\_block\_chi\_name、first\_category\_name、second\_category\_name、first\_brand\_name、,second\_brand\_name信息，在数据粒度这里体现(可以人工填写，识别准确的话可以自动扫表字段)，sql这里只写个口径。

如果上面我们需要的指标都已经录入完成，那么在查询界面大致可以生成这样的查询条件。

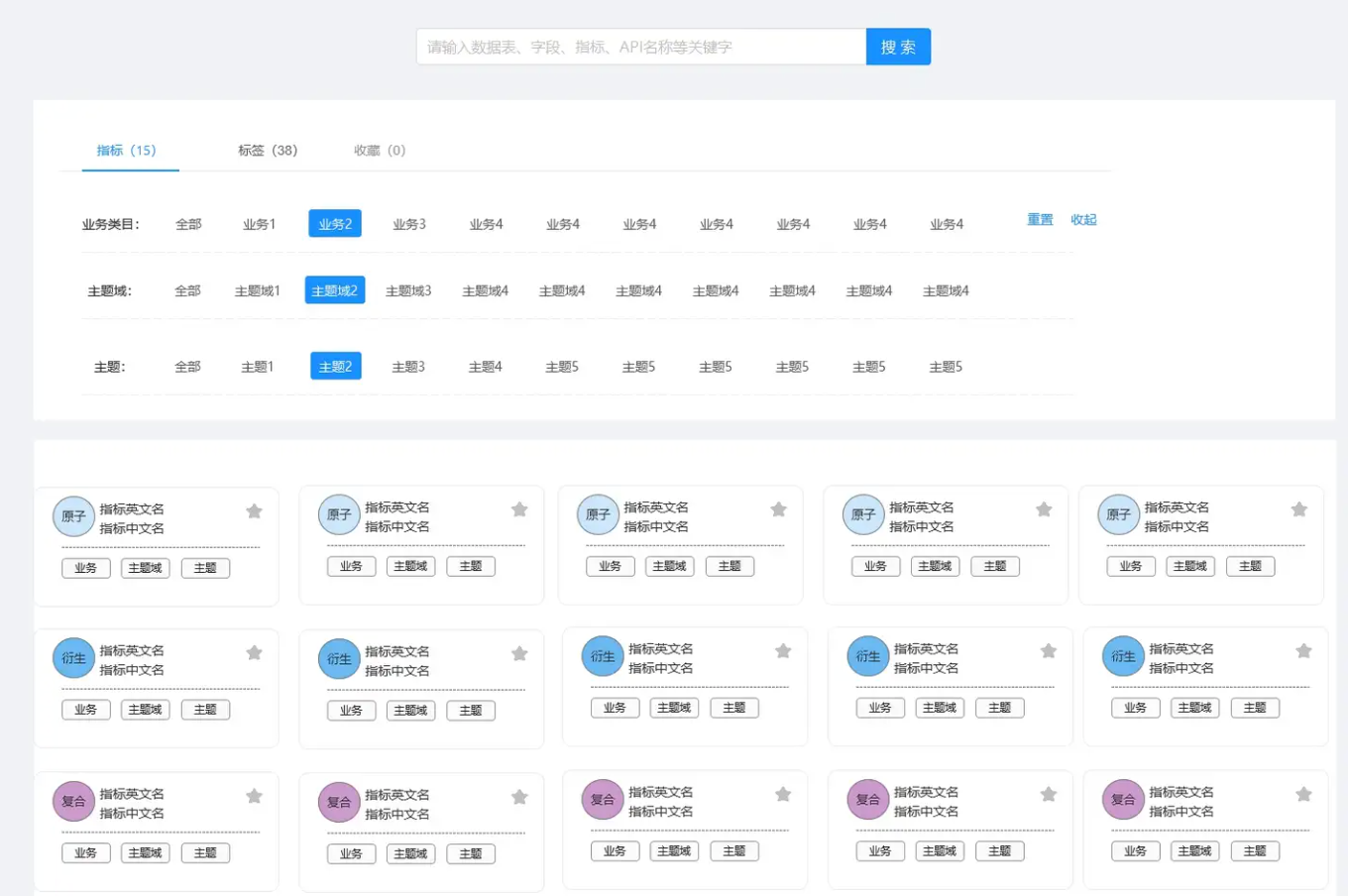


后台根据这个查询条件，生成一个sql查询，查询到数据返回前台，当然能自动化作图展示就更好，然后根据查询自动生成看板，整个需求就算是完结；其他3个查询类似，不再赘述。

**2、查询和查看指标**

查询最重要的就是能筛选到自己想要的数据，看起来像是废话，但是也确实是基于此考虑的。这里个人认为筛选项是符合大家认知的就好，不是一定要区分业务过程，如果运营都是对主题信息比较明确的话，那么按照主题信息划分也可以，如果公司有很多小业务，直接叫业务块划分也行。

eg:



当然现在大模型如火如荼，能通过大模型自动从离线调度的代码里面获取查询条件自动录入也十分可行，或者自动生成查询报告。

**5、可能会碰到的问题**

1、指标平台能解决所有取数问题吗

**不能，我们应该清晰地认识到，指标平台是在数仓建模的基础上做的机构化查询**。能满足大多数常用的数据，但是对于特定的分析场景，还是需要专项数据分析来看数据。个人认为，能满足公司60%的查询分析已经很好了。

2、指标录入时候只录dws层表可行吗

如果dws层建设完善的话，dws可以单独作为一个录入项的，查询实现的时候部分数据就可以走dws查询，性能会好很多。**但是dwd的查询还是要录的，主要是拓展性更好**，dws层数据是必然删减过维度的，一些汇总粒度不一定支持，后续如果想支持指标下钻等，还是需要dwd的数据。

3、指标下钻怎么实现

指标录入的时候有录入过汇总粒度，**在粒度这里还需要父子维度关系维度**，比如一级类目对应二级类目，二级类目对应三级类目。但是如果不是严格包含关系还需要考虑数据对应问题。

4、是不是所有指标都需要写sql

衍生指标都是基于原子指标计算而来，衍生指标 = 维度 + 时间周期 + 修饰词 + 原子指标，如果有修饰词还是需要写的，如果没有可以不用写，或者说这个写的过程可以通过系统识别出来，符合指标录入直接就可以写衍生指标的计算逻辑，比如：衍生指标a / 衍生指标b

5、指标平台怎么推广

配合数分同学，把已有的报表之类的转一部分到指标平台，数分同学自己的报表字段名都向指标平台靠齐，公司各个数据产品指标都保持一致，多跟业务同学沟通。

6、公司规模很小，指标平台太重了，怎么建设

把dws层表作为建设的重点对象，把常用的汇总表扩宽，查询基于这些dws表做，省去录指标这一套。

**六、数据治理**

1、代码解析

calcite：https://strongduanmu.com/blog/apache-calcite-learning-materials.html

**七、UDF**

最近在梳理离线UDF的代码，整合了一下，有的还是比较常用的，简单分享下都建了哪些，从类别分的话大致有3类。

1、外部简单类库引用

a.

2、复杂逻辑处理

|  |
| --- |
| Java |

**七、数仓内部常用工具可以有哪些**

1、代码校验

2、数据探查

3、数据对比

**八、数据主题建设**

1、投放

2、归因

**九、个人思考**

1、数仓工作难点在哪里

首先想象下，一个完美的数仓应该是什么样子，可以从几个方面来思考

1. 已知的一些规范都满足了，分层、分域、分主题、表名、字段名规范都统一了
2. 存储和计算相对最优
3. 任务上下线是有完整流程
4. 新业务融入现有体系成本相对低
5. 任务稳定易维护，数据质量有保障，实时数据产出快，查询性能高
6. 数仓的表确实是好用的(可以从外部的引用次数和主观感受来评价)
7. 排查问题，中间的逻辑非常完善

做到这些，需要有完整的规范，人员对这些规范有充足的认识，在协同工作中产出的任务，思路是一致的。比如任务易维护，这个跟个人素质有很大关系，公司中很难对每个作业做code review；sql虽然逻辑简单，但是实际长了之后没有上下文很难理解。

1、局部最优解和全局最优解的矛盾

2、数仓个人素养体现在哪里

抽象能力

3、

**十、怎么写一个标准的调度任务**

|  |
| --- |
| 1、任务中可以多冗余信息，比如数据从几月几号开始，代码中写死ds > 20240101  2、多段union的代码，可以在多段逻辑中添加数据标记，有数据问题方便找到问题点  3、一个任务最好只有一个输出表，调度依赖配置完善  4、不建议select \*的写法  5、代码规范，层次分明，参照阿里云dataworks的代码格式化工具  6、能用官方函数的都用官方函数  7、代码支持重跑  8、代码性能最优  9、任务名最好和输出表名保持一致 |

代码示例:

|  |
| --- |
| SQL select t8.log\_id  ,t8.user\_id  ,t8.day  ,t8.hour from (  select log\_id  ,user\_id  ,day  ,hour  from dwd.dwd\_flow\_visit\_all\_event\_hi a  where page\_name not in (  'cargolist'  )  and day = ${zdt.add(11,-1).format("yyyyMMdd")}  and hour = ${zdt.add(11,-1).format("HH")}  and app\_type in (1,2,27,28)  ) t8 left join (  select event\_type  ,page\_name  ,element\_id  ,app\_type  from (  select event\_type  ,page\_name  ,element\_id  ,app\_type  from dim.dim\_flow\_page\_element\_history\_used\_dc --历史有用的  union all  select event\_type  ,page\_name  ,element\_id  ,app\_type  from dim.dim\_embedment\_pagename\_element\_dc  where collection\_status in (0,1)  ) t0  group by event\_type  ,page\_name  ,element\_id  ,app\_type  ) t10 on t8.event\_type = t10.event\_type and t8.page\_name = t10.page\_name and t8.element\_id = t10.element\_id and t8.app\_type = t10.app\_type where (  t10.event\_type is not null  and t10.page\_name is not null  and t10.element\_id is not null  and t10.app\_type is not null  ) or (  t8.page\_name = 'home\_tview' --马良页面  or locate('ad\_id',t8.business\_tags) > 0  ) ; |

**十一、实时**

**十二、怎么针对性地做数据建设**