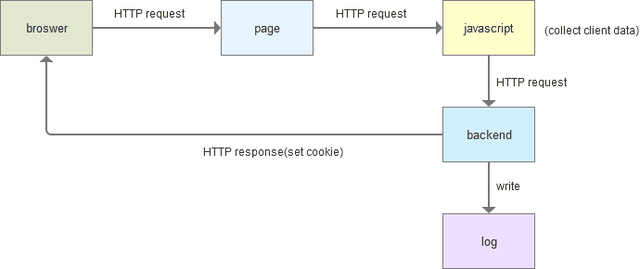
## 埋点服务技术架构

想要确保埋点数据的准确性，需要了解埋点的框架设计，所谓框架设计就是怎样采集数据，上传机制是什么，又如何存储的过程。下面，我们依次分析一下所涉及的技术和内容。



**一、数据采集流程**

当我们在Web或APP中的某个按钮植入埋点代码后，用户点击该按钮触发了点击事件，就会上传定义好的参数信息。我们来看下面这样一个常见的收据采集流程：



前端埋点可以理解为三个阶段：**埋点阶段、数据收集阶段、后端处理阶段**。简单解释如下：

埋点阶段：我们在网页代码中，往往插入小段js代码，即我们的“埋点”；

数据收集阶段：引入js文件，收集数据，然后请求发送给后端；

后端处理阶段：通过解析http request中的参数，拿到数据，然后返回响应内容给前端。

对应上面的流程图，举个例子。例如，用户在网站点击一个跳转链接，这个行为就会触发浏览器对Web服务器的HTTP请求。当网页打开时，页面中的埋点代码会被执行，这个代码一般会动态创建一个script标签，并根据协议将src指向一个单独的js文件，这个文件会被浏览器请求到并执行，这个js往往是真正的数据收集脚本。

数据收集完成后，js会请求一个后端的数据收集脚本，然后将收集到的数据通过HTTP参数的方式传递给后端脚本，后端脚本解析参数，并按照固定记录到日志，同时返回HTTP响应，这个响应包含请求的数据和cookie。

**HTTP解释**

这里顺便说下HTTP。当用户请求一个web页面时，浏览器向服务器发出对该页面所包含对象的HTTP请求报文，服务器接受请求并用包含这些对象的HTTP响应报文进行响应。HTTP使用TCP作为它的支撑运输协议。

即浏览器与Web服务器的HTTP端口建立一个TCP套接字连接，通过TCP套接字，客户端向服务器发送一个请求报文，这个报文包含请求行、首部行、空行和实体主体四部分； 同样，Web服务器接受请求并返回HTTP相应。一个响应包含状态行、首部行、空行、实体主体四部分。HTTP具体规则，可能要单独写一篇关于应用层协议原理的文章才能说明，这里就不多加扩展了。

**二、埋点上传机制**

由于在线日志是直接产生在服务器端， 日志采集工具可以直接从含有日志的服务器上采集日志数据到相应的文件系统， 所以不存在日志上传的问题。但是对于离线日志来说， 数据是产生在客户端的， 所以上传机制必须考虑。

**通常，采用的离线日志上传机制如下：**

**1、实时上报：**服务端提供日志记录接口，当触发事件时，直接调用日志记录接口将日志记录在服务端。如果是频率低，数据量小，实时性要求高的数据可以不设限制。

**优点：**

能实时记录信息到服务器，

**缺点：**

如果埋点较多，产生的数据量太大的情况下，会占用很大的带宽，给用户带来损失。

前端的某些行为，如在某个activity停留时间等也无法通过这种在线的方式捕获。

客户端没有暂存机制，当网络无法使用时，日志记录接口无法正常调用，导致数据丢失。

**2、延时上报：**服务端提供日志上传接口，客户端将日志暂存本地，当达到一定大小或一定时间时，将日志通过上传接口压缩后上传。这个时间和数据量一般根据公司业务情况自定义。

**优点：**

占用带宽小

对用户行为捕获较为灵活

**缺点：**

时效性差，需要等到一定时间或数量才能发送

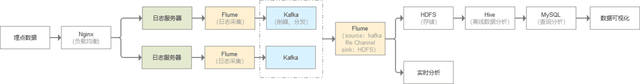
在实际操作中，为了获得更多数据，我们一般会多种方式结合使用。

**三、数据存储过程**

通常系统数据分为2大类，一类是业务交互数据，一类是日志数据。业务交互数据包含业务流程中产生的注册、登录、订单、商品、支付、用户等数据，通常存储在DB中，包含Mysql，Oracle等。一类是日志数据，用户在使用产品过程中，与客户端交互产生的数据，比如页面点击、收藏、停留、评论、分享等。

这里我们主要讲下埋点数据，一般埋点数据主要单独存储在日志服务器中，跟业务服务器分开，主要是为了解耦。有的小型公司为了节省服务器资源（钱）也会将数据都放在一起。

数据达到日志服务器后写入磁盘，然后Flume采集日志到Kafka，使用Kafka进行分发（削峰，且可以根据不同的业务线采集）。随后Kafka会将离线数据通过Flume消费到HDFS，或进入实时存储用于实时分析。HDFS将数据导入到Hive数仓（ODS,DWD,DWS），最终Hive分析完的结果用Sqoop导入MySQL用于查询，最终实现对数据的可视化。



当然，上述选用的技术也是需要根据实际情况决定的。有相同作用的也不止上述这些，比如数据存储还有HBase，Redis等，数据计算还有Spark，Storm等，我们选择哪些什么框架也是一门学问。

总之，数据就是2条流向：

1、实时存储，实时分析生成数据报表；

2、离线存储，进入数据仓库。

**无论是实时的数据采集，还是先本地存储稍后再发送的数据采集，最终都是要进入数据仓库的**。然后这些数据经过层层逻辑加工到达数据应用层。我们熟知的大数据分析平台和行为分析平台等都是应用层的典型案例。

**数据仓库解释，**具体可参考我之前写过的《什么是数据仓库》

数据仓库分为三层：原始数据层（ODS）、数据仓库（DW）、数据应用层（APP）。进入数据仓库的埋点数据会以表的形式存储于ODS（原始数据层）中，然后经过轻度汇总后会转存到数据仓库的基础层，按照一定维度和业务逻辑，对数据进行聚合后转而进入主题层、数据集市。

● **原始数据层（ODS）**：数据无任何更改，不对外开放；为临时存储层，是接口数据的临时存储区域，为后一步数据处理做准备。

● **数据仓库（DW）**：也叫细节层。是对源数据进行了ETL之后的数据。一致、准确、干净的数据。

● **数据应用层（DA或APP）**：前端应用直接读取的数据。根据报表、专题分析需求而计算生成的数据。



**四、常见埋点问题及原因**

进行数据统计时，我们往往会对数据的准确性产生怀疑。当偏差低于5%时，我们很难感知。但当出现较大的偏差时，就很容易发现数据不准确了。以下列举几个常见问题及可能的原因：

**1、平常数据稳定，突然过高**

**1）可能被人恶意刷流量。**很多人觉得流量高了不好吗，但是如果流量的来源有50%以上都是未知数据来源，一次性访问十几个页面这种。搜索引擎会对网站做出惩罚，被降权。

**2）你的统计代码被其他网站使用了。**这种情况比较少，但是之前我遇到过，A站的统计代码，不小心被BCD站也使用了。或者被盗取了，导致呈现的结果突然成倍增加。

**3）非恶意浏览。**流量或下载量突然增加很多，却什么都查不出，显示一切正常，那么只能说恭喜你。

**2、平常数据稳定，报表数据过低**

**1）日志迁移。**由于日志迁移过程太长，导致一些日志没有参与入库。需要找运维同学，重新部署昨天的统计任务，恢复数据。

**2）新增埋点的报表数据都是0的情况**。需要先查日志确认客户端功能是否正常，如果正常，则需要确定稳定是否出在计算和展示层面。一般情况，这种问题还是比较容易恢复的。如果数据都没有上报，那只能等下个版本恢复了。

**3、数据异常**

**1）网络异常。**比如某个高峰期，同时有大量用户提供行为数据，这样容易造成网络拥堵，就行春节期间的12306一样，这样容易导致某些请求丢失，造成数据不准确。对于缓存本地的数据来说，如果数据达到上限，也可能会造成部分数据丢失。

**2）遗漏重要的用户路径。**比如我们肉眼可见平时的UV不低于3000人，却只有几个人，这种情况可能是遗漏了重要的用户路径，需要重新设计埋点。

**3）统计口径不同。**比如启动APP就属于活跃？还是首页加载完成才算活跃，还是说要完成某个关键指标才算活跃。

**4）无效请求。**比如竞争对手的恶意攻击，Spider 等进行的抓取操作，都会导致数据的异常。

**四、如何提升埋点质量**

通过上面常见的问题，那么提高埋点质量，我们需要做什么呢，我从以下几方面来分析：

**1、关键行为的采集，使用后端埋点**

什么是关键行为。比如订单支付、注册成功与否。我们在第2章里已经说过后端埋点的优势，将关键行为通过后端数据采集，既可以提高数据的准确性。

**2、建立完善的埋点规范**

有的企业，代码埋点出错率高达50%，比如简单点击事件的埋点，该埋在代码A上的点，却放在了代码B上。这就说明了如果没有统一的标准，就会导致埋点出错。

**1）埋点命名规范**

比如我们规定埋点名称只能由字母、下划线、数字组成，并保证其的唯一性。例如，设定ck是点击、sw是展示；埋点事件命名规范是：{团队|业务|角色}{组件|页面}{具体元素}\_{动作}

**示例：**

trademark\_NavBar\_bar\_ck 。trademark代表业务线，NavBar代表页面，bar代表功能，ck是点击

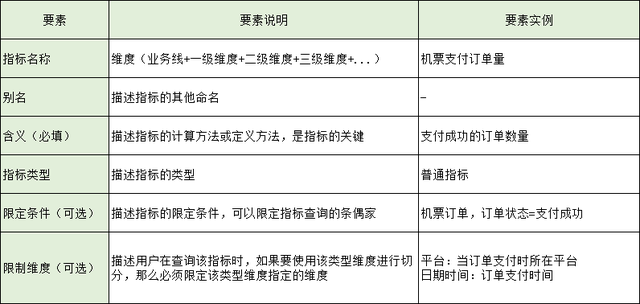
patent\_comt\_share\_ck 。patent代表业务线，comt评价组件，share分享按钮，ck点击

口径不同，那么我们得到的结果就肯定有偏差。这时候，指标字典就显得很重要了

**2）建立指标字典**

指标字典是业务数据标准化的基础，目的是对指标进行统一管理，方便共享，对达成对业务指标的共识。指标字典一般维护在Excel或Wiki中，或者放在数据管理系统中，配合血缘关系，就更方便追踪数据的流转了。

指标一般分为基础指标、普通指标和计算指标三类。指标字典通常包含指标维度和指标量度两部分。



来源：数据产品经理修炼手册

**3）流程规范约束**

埋点需求和业务功能需求一样重要，也需要经历【需求—评审—开发—测试—验收—上线】的完整生命周期，如果能按照这个流程规范一环一环做好，那么我们已经能够避免90%的风险。

需求阶段：确定埋点信息，输出埋点文档，详见上一章《输出埋点需求文档》

评审阶段：产品讲解埋点需求，测试和开发等相关人员提出疑问和建议

开发阶段：产品和开发保持密切沟通，并且完全理解需求背景和需求字段，在埋点规范下进行埋点。开发探测每个埋点对应到的代码文件和代码行，开发根据人均事件量级进行监控报警功能。

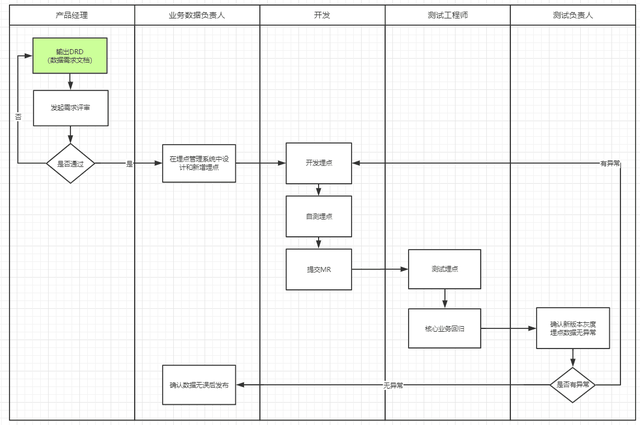
测试阶段：测试需要在完全理解需求背景和业务语义的前提下，检查埋点测试中上传的事件ID、参数等是否符合规范、触发时间是否正确。

验收阶段（非必要）：某些情况下会有需求相关方进行验收，确保相关人员在理解业务语义的前提下，可以通过与自比较和他比较的方式来进行验证。

自比较验证：同一APP某一个业务线的购买冒泡数量一定要小于所有业务线的冒泡数合计；

他比较验证：前端某业务点数量和对应服务端数据应该基本相当。

上线阶段：保证上线环境与测试环境一致，新上线的埋点数据需要进行追踪。



**3、安全性考虑**

采用https协议以确保传输对的完整性和安全性，http在传输过程中可能被劫持（如：JS代码注入），会造成页面出现未知的广告或者被重定向等问题。

**4、构建埋点管理系统**

当公司产品越做越大，埋点越来越多。使用文档或表格记录埋点的编码命名、业务含义或者其他重要信息的时候，就显得很力不从心了。随之诞生了——**埋点信息管理系统。**这也是我们提高埋点效率，保障数据准确性的有力措施。系统主要包含：

1）埋点信息管理：添加埋点、埋点上报、埋点下线；

2）埋点属性信息：通用属性管理、自定义属性管理；

3）辅助功能：埋点数据监控、埋点信息预览，数据可视化等。