## IDS数据收集分析平台分析与规划

### 目标：

最大程度的收集终端用户的操作，高效的收集硬件设备的回传数据。高效、稳定、可扩展性强。对数据展示和报表引擎做强有力的支撑。

逻辑极简，结构强规则！

规划两套日志系统：

1. （主要）从logServer服务器进行数据接入：相对实时，数据收发频率较高，传送的数据有固定的数据格式，结构不变。
2. （次要）采用action的方式进行数据接入：无实时性要求，数据格式自定义，进行独立解析。

### 数据来源

按照数据来源区分APP/WEB/Beacon.

1. app数据来源于APP捕获用户的操作，包括点击、使用查看等所有的操作信息，这部分数据可以相对比较全面的描述一个用户的连续性操作。手机可用它通过硬件信息来识别唯一的用户设备。
2. WEB数据来源于浏览器打开页面后所能捕获的操作，包括点击、停留、查看等动作。WEB数据通过一次有效的cookie记录来标识唯一用户的使用。
3. Beacon数据，智能IDS上报的rssi数据，目前这部分数据以MAC地址来唯一标识用户。

### 数据类型划分：

将收集的数据描述成属性+对象的数据逻辑结构，具体属性和对象细分罗列如下。具体数据结构将定义成为：

**public** **class** LogOpPkg {

**public** **long** snH;

**public** **long** snL;

List<OpItem> opList;

}

**public** **class** OpItem {

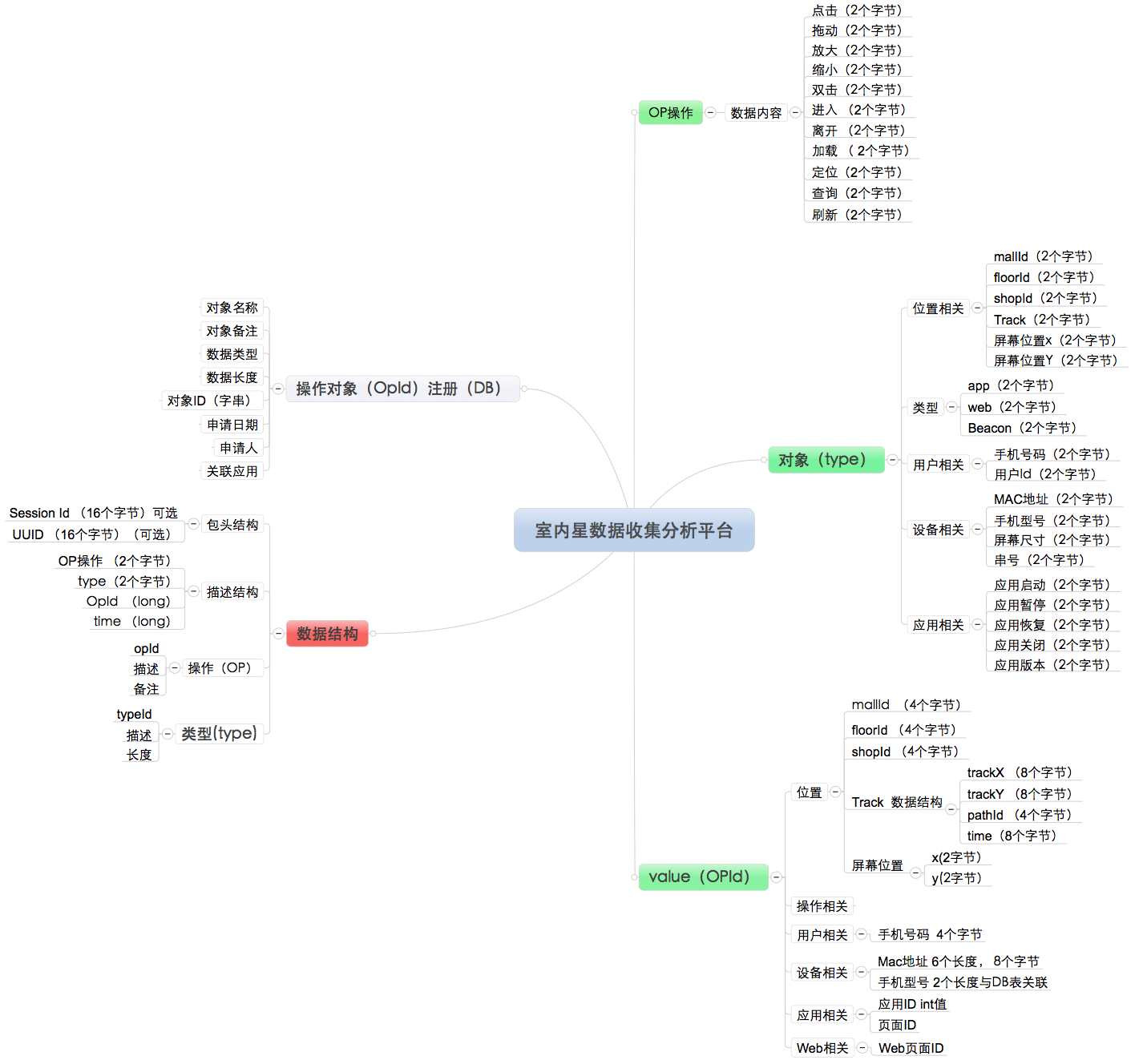
**public** **short** op;

public short type;

**public** **long** opId;

public long time;

}



操作（OP） + 对象（type）+ 值（OPID）+ 时间（Time）

（动作） （实体） （唯一ID） （操作时间）

简化起见，将所有的OpId统一成long类型进行存储，带来的问题是会有空间的浪费，但是处理非常简洁，可以采用数据压缩的方案进行缓解。

操作对象注册的设计：

前提：每个页面的唯一ID值由专人负责进行分发

操作对象保存在关系DB中。

其中mallId、floorId等数据库已有数据从原有库表中获取。

DB设计：

表一： **d\_logopid**

{名字，备注，类型，长度，id，日期，申请人，关联应用，应用版本}

{name，remark，type，length，id，time，proposerId，appId, appver}

名字：注册对象中文名称

备注：针对web页面存储web的url， 针对app存储其页面描述。

类型：描述ID对象的数据类型。

长度：描述ID的字节长度。

ID： OPID的ID值，在DB中保存long类型

日期：对象的注册时间。

申请人：该对象的维护负责人，关联用户表。

关联应用：该对象归属于那个应用，关联应用描述表。

表二：d\_appversion

{版本Id，应用Id，应用版本，应用版本描述}

{id,appid,ver,des }

**表三：d\_appinfo**

{应用Id，应用名称，应用发布日期，app系统}

{id,name,publishtime,updatetime,appsystem}

应用ID：应用描述，可能是web，可能是app，都在本表内进行标记,

应用名称：中文字串。

发布日期：应用对外发布的时间。

应用版本：当前版本的

数据流向说明：

App/web 流向 logServer，流向MQ队列， 流向Hbase



### 接口说明：

针对现有平台情况，ios，Android接口，web js 接口，Java接口。具体应该至少包含如下接口：

1. InitLog（uuid,IP，port）;  
   初始化
2. SendOP（op，type，opId，time）;
3. Int Flush();

App、PC等应用数据底层采用TCP接口同logServer进行交互。

针对Web数据收集，实现独立action进行交互。

### 数据收集分析平台指标监控：

消息处理量：通过每秒接收消息值进行反馈。

活跃用户数：通过识别活跃设备进行反馈。

中间数据结构设计：

原始数据结构是原子操作的描述单元，原始数据结构在系统中会完整保留下来。

中间数据结构完全根据业务需求来制定。

以广告点击收费为例：

* 收集广告点击操作的信息，仅仅关心点击操作的信息【数据结构为，广告Id，点击数量，时间】

另假想数了刘翔据字段全关联我们会定义

【广告Id，用户id，时间，设备信息，地点，浏览器，广告位置信息，应用信息】

而上述数据是针对广告，要是我们要统计应用的话表可能会设计成

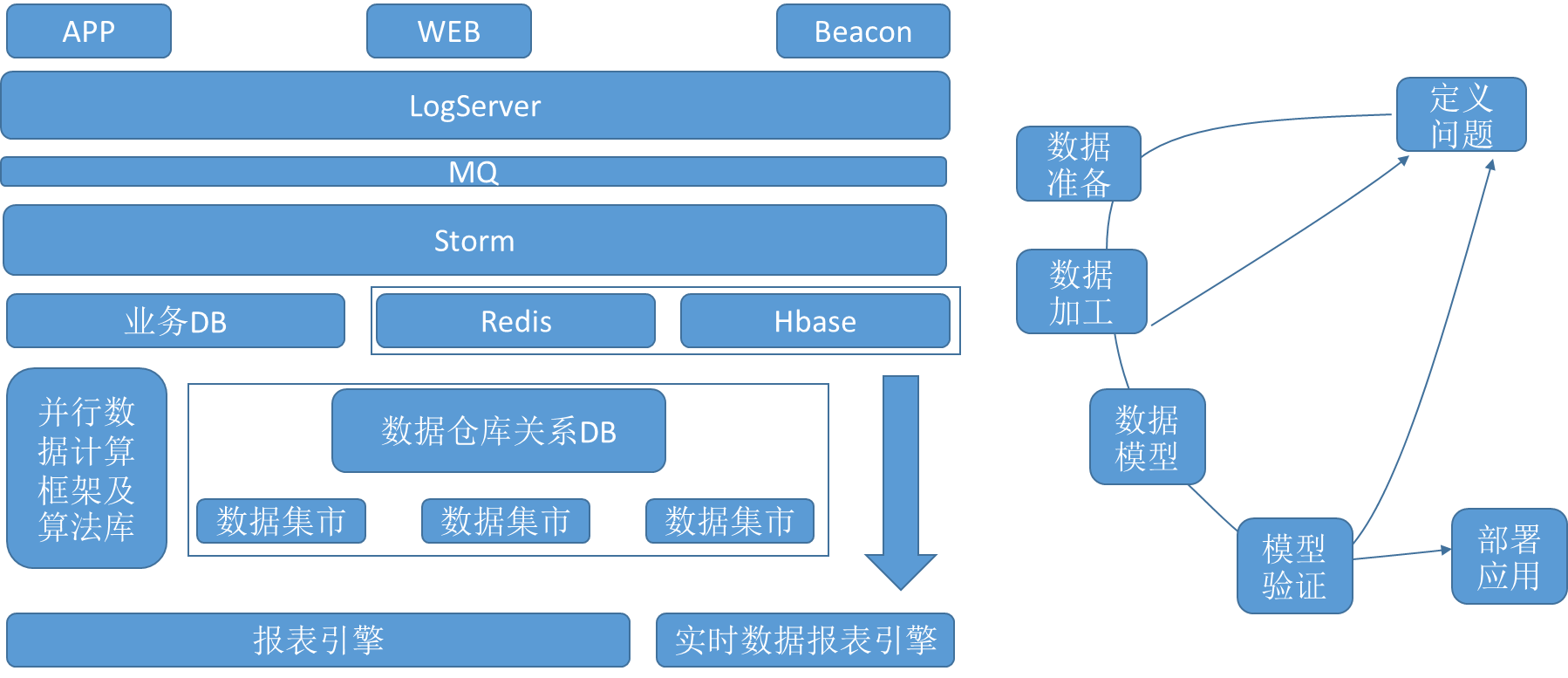
【应用ID，应用操作，时间，设备信息，地点，软件环境】

所以这部分内容，首先在客户端上目标是收集所有可以收集的数据。在服务端，对数据收据重要性进行区分，哪些实时处理，哪些延后处理，并且定义数据的时间有效性。

上述表内容存放在Hbase中，在前端报表有明确需求后，将Hbase数据在进行第二次ETL操作加载到对应DB或者数据集市中。

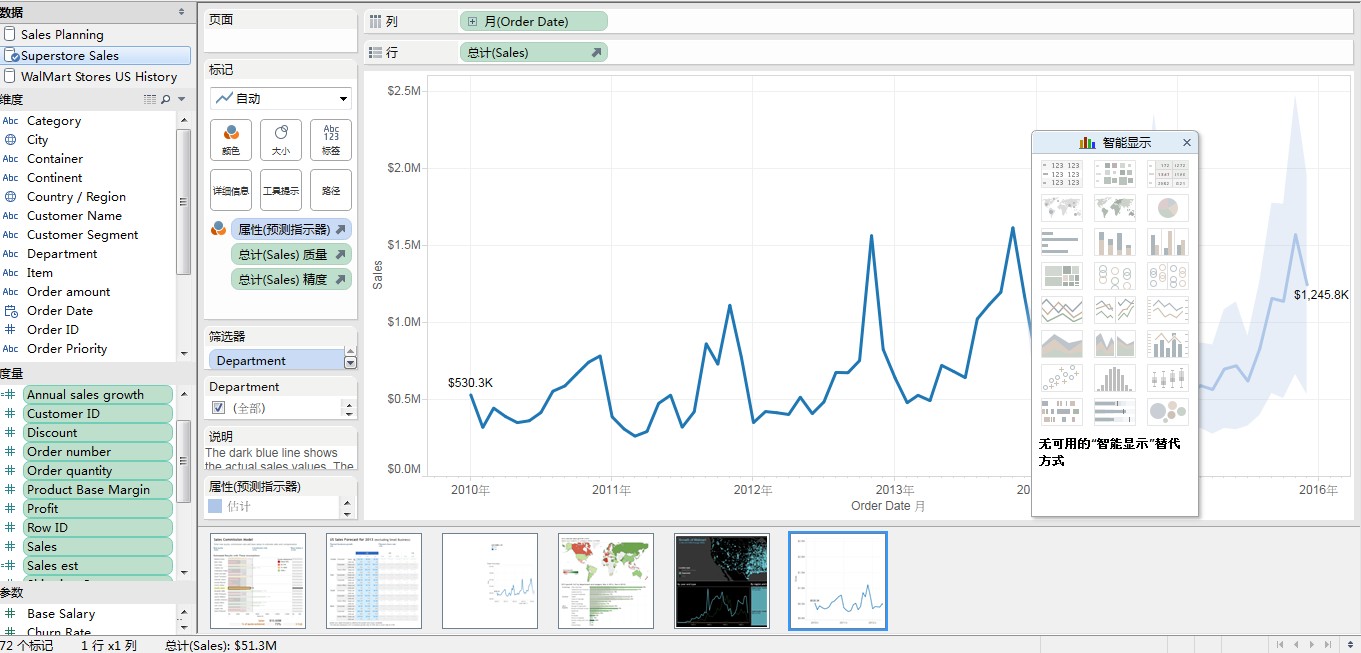
\*\*\*\*以上过程的关键技术点是1.收集尽可能多的数据2.服务端对数据进行数据修补和完善。3.针对特定需求对数据进行必要的转换。

### 系统架构：



针对需求和业务特点设定架构如上图：

* 数据收集主要集中在logServer上，不同的数据将在logServer上进行解析，并且按照不同的队列进行投递。（此处不排除在有条件的客户端上直接向MQ投递数据，但目前不建议，会增加客户端的复杂度，同时缺少logServer对数据进行清洗的过程，前后台数据的定义和调整也变得复杂）
* MQ作为消息中间件，保证消息的有序和数据的缓冲，将高实时的数据进行优先处理，重要数据进行可靠性传输，
* Storm对数据进行清洗和转换，主要工作包括1. 持久化所有原始数据 2. 分析数据上下文关系 3. 转换目标数据结构 如track计算。
* 实时数据持续持久化在redis中， 大量记录保存在Hbase中，用作后续分析，DB保存频率不高的实时报表更新数据。
* 针对Hbase中和redis中及Db中的数据记性批量的高延迟计算，转换数据到数据仓库中的一张表或者几张表中。（此处会是定时运行，因为计算的查询转换的延迟会比较高）
* 传统报表引擎：通过连接数据仓库DB进行数据的展示和设计。下图是典型的报表BI工具，维度和度值对应数据库中对应的列项，通过拖拽展示各种图形。根据业务需求调试后台DB的数据，前端快速的将结果进行展示。



* 实时报表引擎：通过对Storm实时计算统计的数据进行展示，这部分数据后台会不停地更新，前台刷新频率也会相对较高，保证在系统后台数据压力较大的情况下可以及时的反馈实时的统计结果。 这部分内容基本是必须开发完成，没有现成的工具直接使用。

后台需要保证的是数据可以源源不断的被收集上来，在需要某些数据的时候可以通过数据的转换找到想要的数据。

实时数据部分根据前端显示需要做数据定制，目前看这部分就是有新的定制需求，后台的数据准备基本是要重新做一遍。但是谈不上多少的复用。

### 计划：

首先，数据收的下来，存的住，离线分析的了。（初步框架雏形已经基本完成）

其次，捕获大量我们能想到的数据。前提是要有传感器或者用户可以给我们源源不断的上报数据。通过web，app等多种手段进行数据收集。

再次，对数据进行清洗和转换，进行面向业务的常规性业务报表实现，并且逐渐提升其中关键报表数据的实时性。

最后，对历史数据进行深度的挖掘，分析和预测。（面向用户行为，轨迹）

最终目标，通过数据对相应的人员进行辅助决策或者精准的人物画像分析。