**《存触宝系统》**

解

决

方

案

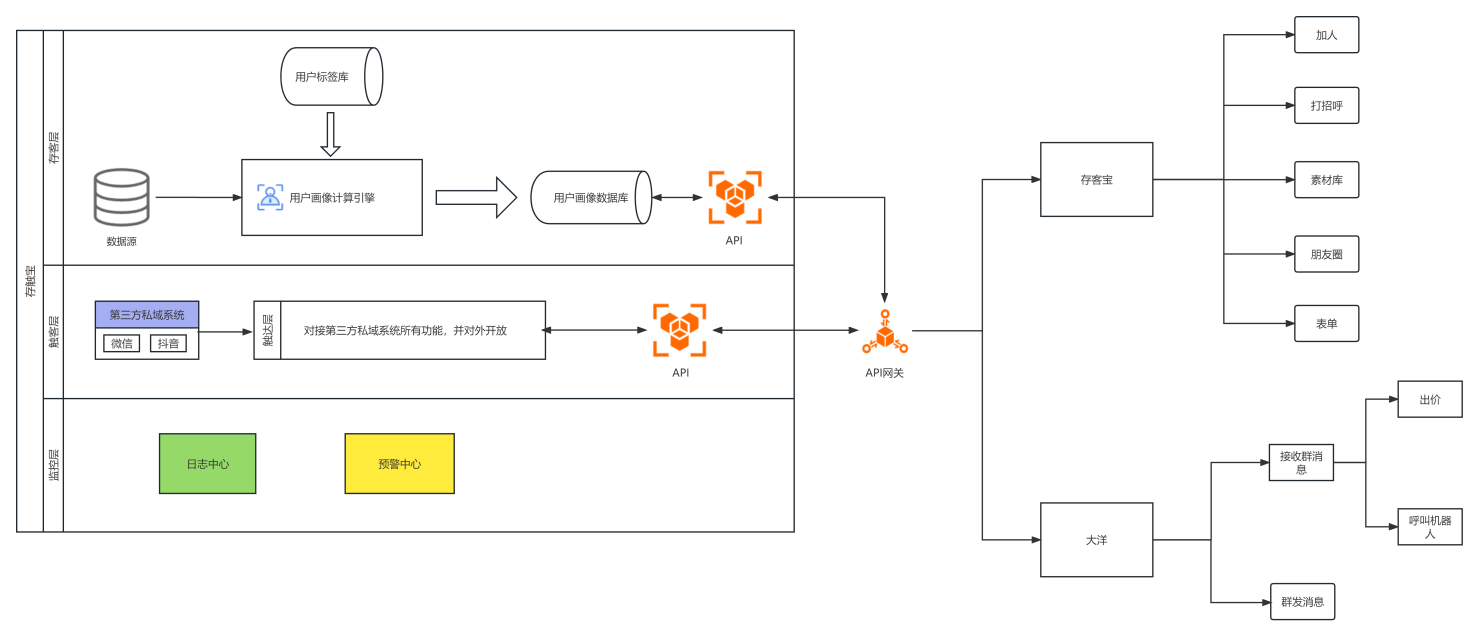
**卡若私域**

**2023年5月**

**版本控制信息**

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

# 系统功能流程图



# 各层详细说明

## 存客层

该层的核心目的就是“存客”，把客户的资料经过用户画像的计算引擎，保存到用户画像的数据库。这个用户画像的数据库，是经过大数据的架构设计，支持大规模数据实时查询。它允许运营人员筛选出具有复杂条件的用户人群，实现了精准的客户营销。

用户画像引擎支持多种数据源输入，也可以自定义输入，总之可以满足各种各样的数据源。搭配丰富的标签库，可以为客户打上各式各样的自定义标签。

总而言之，存客层的目的就是“存”，且是打了标签的客户。这个是整个体系最有价值的部分。

## 触客层

在触客层里，搭配各种私域系统，对客户进行形式多样的“触达”。不同的私域系统有不一样的触达。例如微信：加好友、发消息、发朋友圈等；抖音：点赞、发消息、关注等。

触客层，为下游的项目提供了基础支撑，它是公司业务开展的核心。

## 监控层

监控层包含了日志中心、预警中心。日志中心收集所有系统产生的日志，为故障的现场还原提供重要的依据。预警中心则为维护人员提供了预警提醒，预警分故障等级，不同的等级会有不同的提醒。预警的消息会发送到维护人员的微信群里。

## 网关层

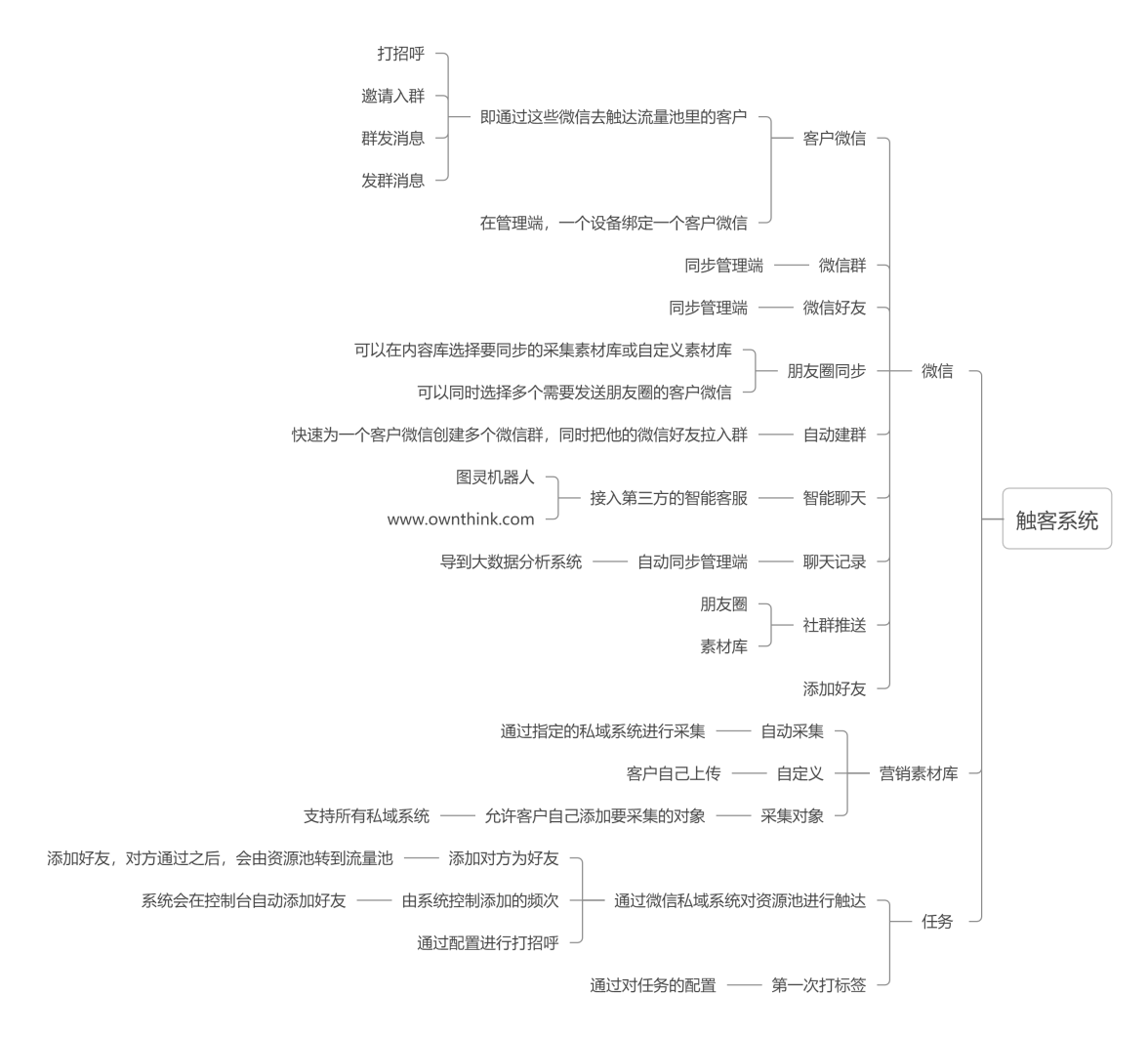
存客层、触客层的功能都是通过API的方式对外开放，为了保障API对外隔离与安全，加一个网关层对所有的访问进行拦截与过滤。

## 项目层

项目层的项目，每个都是独立的产品，他们会用到存触宝的部分功能。

# 业务规则详细说明

## 触客系统



### 微信

目前全国做微信私域系统的有四家，技术解决方案相同。大概的原理就是：首先，把手机的操作系统刷成了他们特制的操作系统，这样可以监听微信的所有通信；其次，在手机上有一个“后门”，外部的应用程序可以直接与这个“后门”进行通信，发出指令，以达到控制微信的目的。

存客宝与某家公司合作，获取了所有的指令，然后结合公司自己特有的商业解决方案，最终形成了存客宝系统。因此，微信的私域系统是公司的核心中的核心，它的稳定性决定了公司的整体业务。

#### 微信私域系统对接的技术解决方案

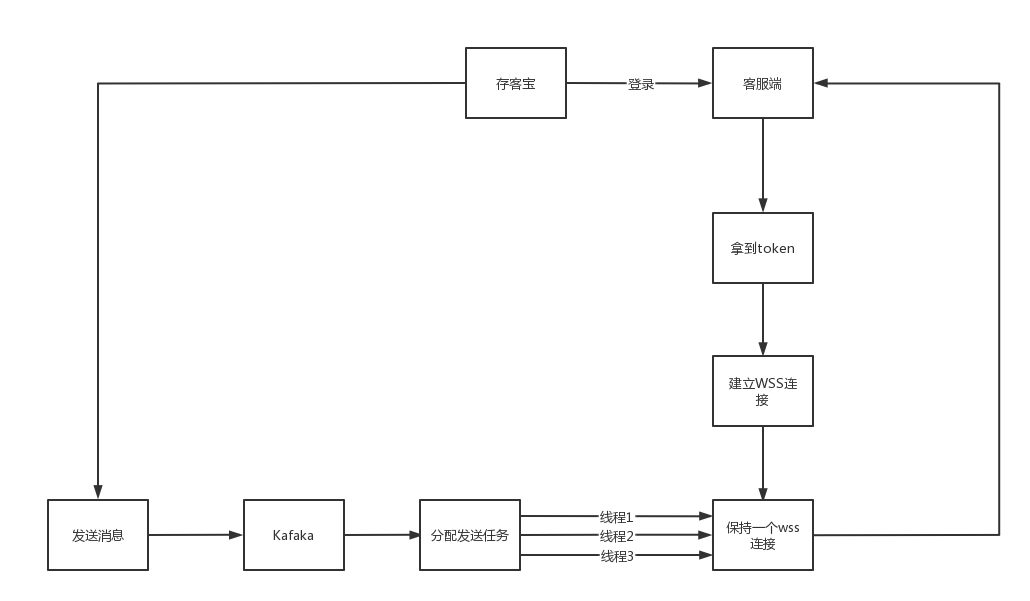
微信的私域系统提供了两个后台：

管理端：<https://kf.quwanzhi.com/admin/#/login>，这个后台提供了丰富的微信端的管理功能，存客宝可以通过管理端的api实现绑定设备、创建微信账号、添加好友等功能。

客服端：https://kf.quwanzhi.com/user2/main/，在这个端可以实现1v1聊天，群聊，添加好友等功能。

存客宝本质就是用技术的手段模拟手工操作那两个端提供的接口，以实现自己的业务需求。因此，只要这两个端有的功能，理论上都可以在存客宝上实现。

以下是存客宝发送消息、群聊、更新在线状态、添加好友等功能的流程图：



在这里引入了消息中间件rocketmq，目的是给websocket的客户端投递任务,还有一个重要作用就是防止系统宕机导致消息发送丢失，因此rocketmq最好用商业的云端部署版，稳定性高，如果考虑成本的话，就自建rocketmq集群。

任务接收器接到来自rocketmq的任务之后，利用java多线程的优势，并发投递消息给websocket服务端(因为websocket是异步投递消息，因此不一定用并发方式投递，看情况)。

为什么选择rocketmq？重要的一点就是，它是严格顺序的队列。

#### 客户微信

即存客宝客户购买设备绑定的微信，通过调用管理端的api可以获取所有绑定的微信。私域系统通过客户微信这个对象进行一系列的触达：聊天、群消息、建群、发朋友圈、添加好友等等。

#### 微信好友

通过管理端的api，将所有客户微信里的好友都同步到存客宝，即微信好友。目前系统可以针对微信好友，进行快速建群和采集朋友圈

#### 微信群

原理同微信好友，存客宝可以针对微信群进行消息发送，例如营销工具里的拍卖。

#### 自动建群

系统可以针对某个客户微信，快速把他微信里的好友均分建群，同时把客服拉倒每个群里。这个功能是用到了管理端的【建群助手】。

#### 朋友圈同步

存客宝可以在采集对象列表里，选择想要同步的微信，系统就会自动同步选中的采集微信的朋友圈内容，同步到设定好的客户微信里去。

#### 下载聊天记录

聊天记录导入大数据分析，得出标签

#### 社群推送

把素材库的内容，根据一定的规则推送到指定的微信群

#### 微信号风控策略

##### 加好友

新号每天10个，老号每天20个。新号和老号的界定，暂用人为。

如果某个号加好友被提示频繁，则禁加好友24小时。24小时之后，每天加人次数减1，直到72小时后恢复到原来的次数。如果在24-72小时之内，再次频繁了，则再禁24小时，加人次数再降1。如此循环往复。

### 抖音

目前存客宝有关抖音，需要做的功能是点赞回复的功能，目的是拿到手机号，然后加对方的微信。这个解决的方案，采用第三方的超级管家。触客宝通过调用它的API，完成与抖音客户的交互，最终拿到客资保存到存客宝的过程。

#### 前置工作

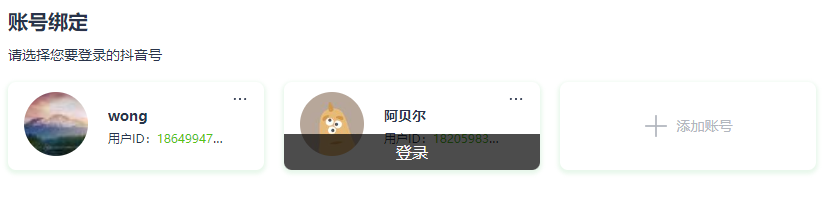
##### 抖音账号要求

必须是带蓝V的认证企业号，才能对接超级管家。详细如图：



##### 添加账号

认证完毕之后，就可以与超级管家进行对接，第一步就是点击添加账号

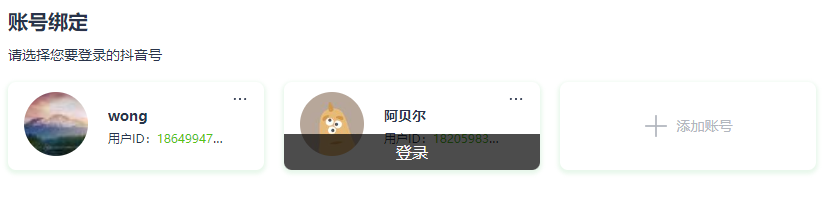


然后点击链接，跳转到抖音的界面进行授权



授权完毕之后，就会在账号绑定里看到客户的抖音号了

##### 登录超级管理



添加账号完毕之后，可以点击登录进入超级管家：



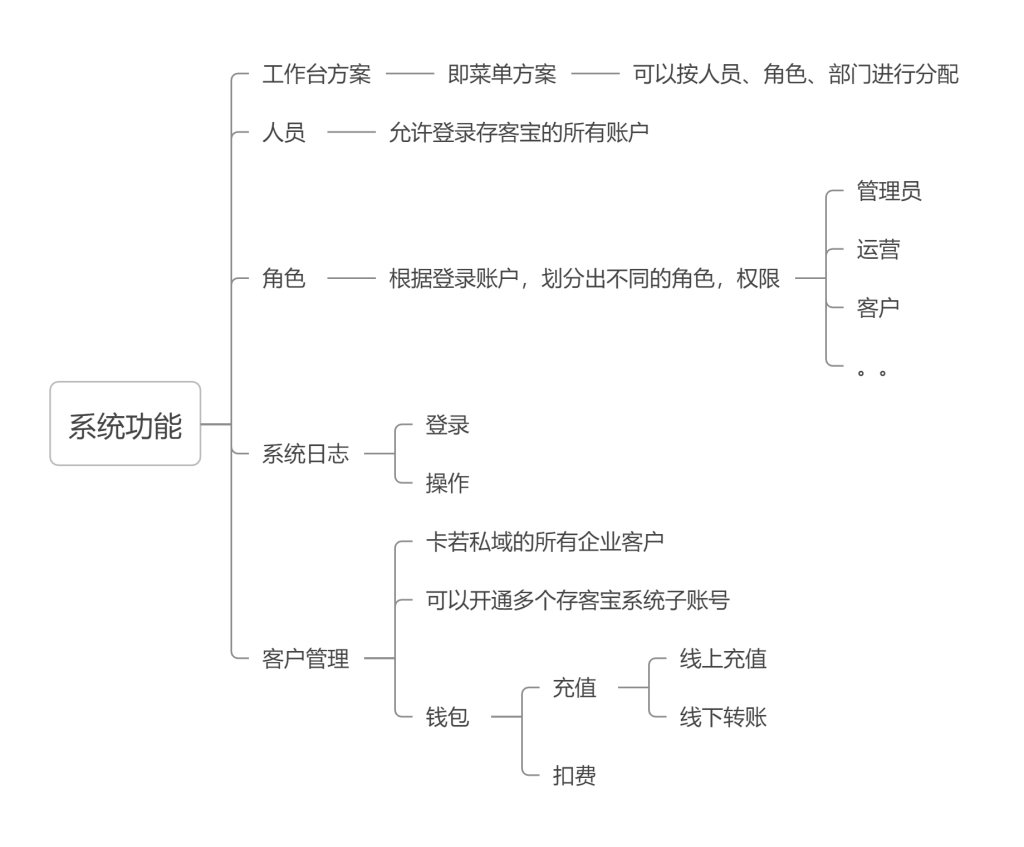
##### 抖音授权

让客户抖音扫码，每三十天会失效，就需要提醒客户再次扫码授权。

#### 接口

|  |  |  |
| --- | --- | --- |
| **功能** | **接口** | **备注** |
| 登录 | https://cjgj.lingyundata.com/api/merchant/wxOpenId/login\_wx?merchantId=1449935735115403265 | POST |
| 客户池 | https://cjgj.lingyundata.com/api/scene/intent\_user/list?pageNumber=1&pageSize=5&keyType=1&nickname=&preference=&hasFans=&activeTime=&activeType=&pageOrder=&pageSort=&tagIds=&lastActiveTimeBegin=&lastActiveTimeEnd= |  |
| 留资客户 | https://cjgj.lingyundata.com/api/scene/leads\_user/list?pageNumber=1&pageSize=5&keyType=1&nickname=&pageOrder=&pageSort= | 必须要授权后，才能看到联系方式。多个联系方式是用半角逗号隔开。客户留下联系方式后，会自动出现在这个列表。 |
| 查看点赞的任务 | https://cjgj.lingyundata.com/api/scene/market\_strategy/detail?id=1453172127370973186 | 这里的id要去登录列表里拿 |
| 新建任务 | https://cjgj.lingyundata.com/api/scene/market\_strategy/save |  |
| 私信列表 | https://cjgj.lingyundata.com/api/scene/intent\_user/find\_im\_user?pageNumber=1&pageSize=10&read=&reply=&tagIds=&nickname=&lastAskTimeBegin=2023-09-20+16:29:15&lastAskTimeEnd=2023-09-21+16:29:15 |  |
| 聊天记录 | https://cjgj.lingyundata.com/api/enterprise/im\_msg/list?fromUserId=225db8cf-2832-423d-97df-650f906cff07&toUserId=6b2b9404-4635-4954-a241-b018a0faaf80 |  |

## 系统功能



### 客户管理

#### 设计目的

管理所有使用存客宝系统的客户，即公司实体。每个客户可以开通若干个存客宝账号。其次，可以让资源池，私域系统的账号有归属。

#### 新建客户

由我们的运营人员在后台新建客户，用客户的联系人和手机号，同时创建成为客户的管理员，这个管理员可以创建后台的操作子账号。要注意管理员和子账号所对应的角色，开发的时候先创建好角色。

#### 开通子账户

允许客户的管理账户在后台创建子账号，子账号可以操作后台的一些特定功能。

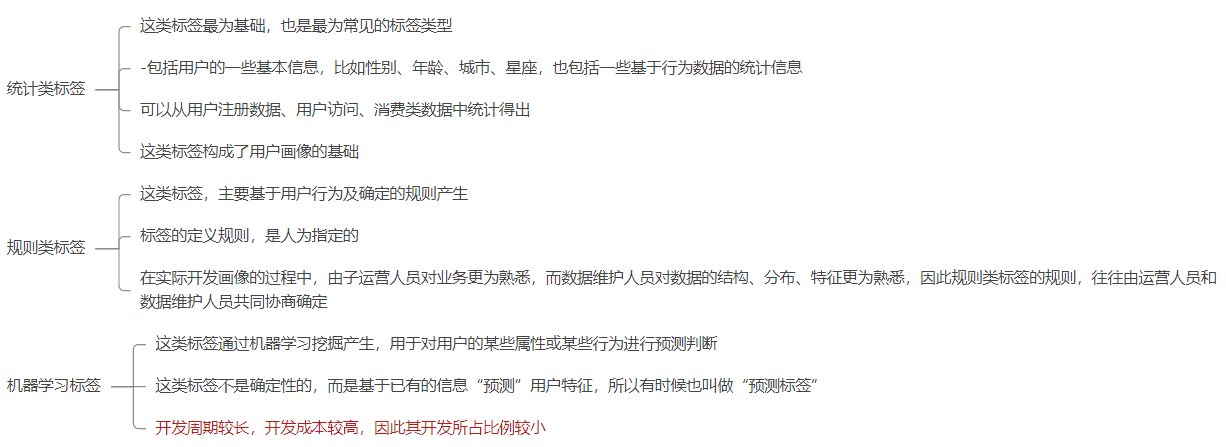
## 存客系统

### 标签管理

详见存客宝系统的思维导图，有详细规划。

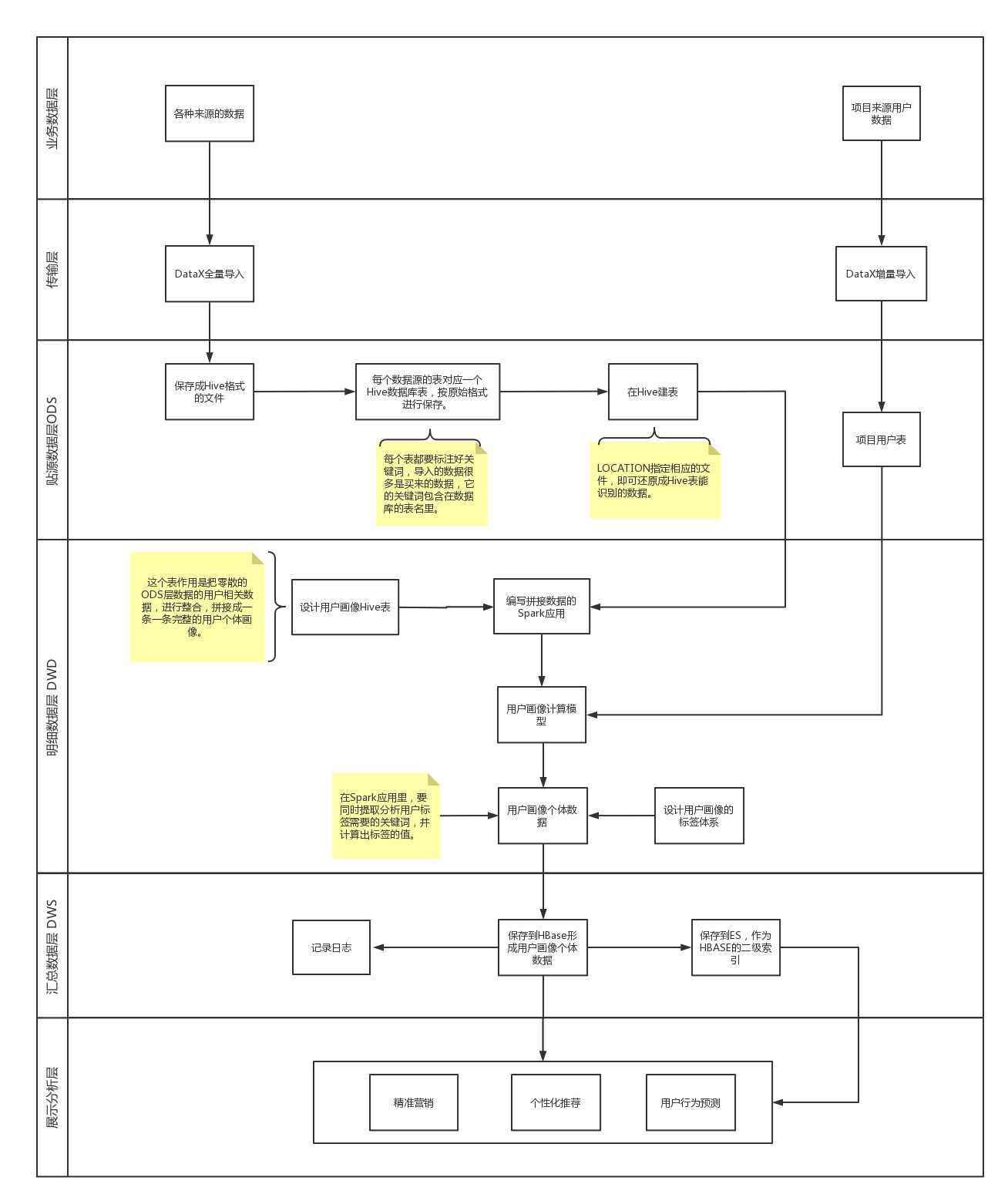
### 用户画像

标签分成三类：统计型、规则型、挖掘型。

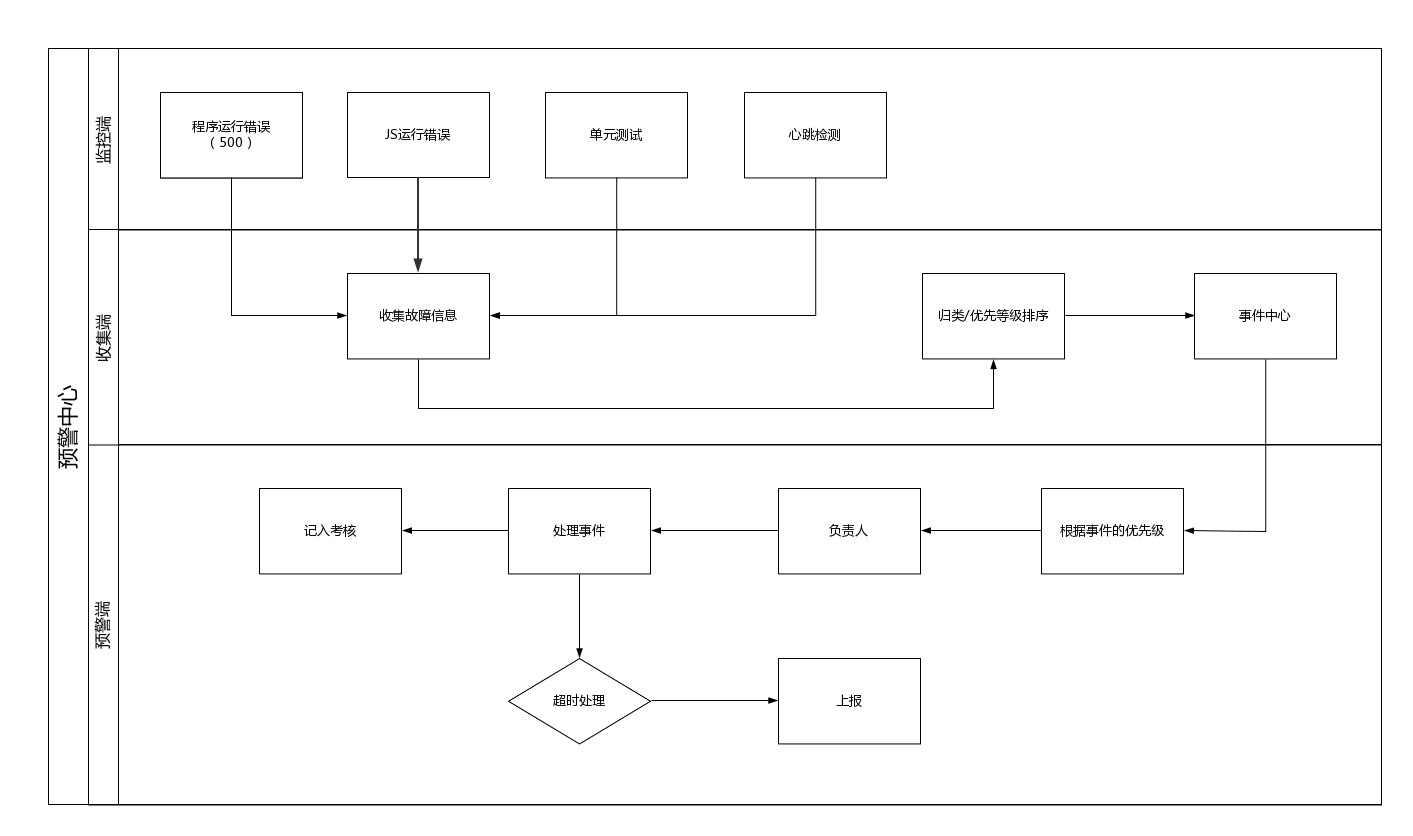


公司的数据有两类：一类是购买的用户数据，另一类是通过项目引流进入存客宝的用户数据。前者属于静态数据，一次导入到用户画像系统，但是每个数据源的处理程序不一样。后者则是动态数据，每天都要进行导入。数据格式相对固定，只需一种计算程序即可。

具体的流程图如下：



## 预警系统



### 各端说明

系统总体分成四端：监控端、收集端、预警端

|  |  |
| --- | --- |
| **端** | **说明** |
| 监控端 | 在业务系统中嵌入监控程序，监控系统的运行状态，收集程序运行的错误信息、JS运行错误信息、运行日志、调试日志、单元测试信息、服务器负载信息、心跳检测。将收集的信息传给收集端进行后期的处理。 |
| 收集端 | 将系统监控端发送过来的故障信息，按照故障等级存入数据库。 |
| 预警端 | 根据收集到的故障信息，按故障等级进行排序划分，发送给不同的负责人和监管人进行处理。负责人处理的进度结果直接记入绩效考核，如果超时处理，要预警上级进行跟踪处理。 |

### 故障的收集端设计

这部分是整个预警系统的核心和难点。需要做到HA且解耦，因此要引入消息中间件。考虑到系统已经有Redis,可以考虑它作为消息的中间件。

收集的流程可以是：监控端 -> Redis -> 收集端

Redis部署在云端，收集端部署在公司的集群服务器

### 故障级别规划

|  |  |  |
| --- | --- | --- |
| **故障类型** | **故障级别** | **监控方式** |
| 瘫痪 | 1 | 定时做心跳检测，可以在收集端做这个动作。 |
| 超高负载 | 1 | 在监测端，运行一个特殊的进程来定时监测服务器和数据库的负载，一旦超过阈值，上报故障信息。 |
| 程序500错误 | 1-3 | 根据报错的位置，具体定故障级别。如果是全局500错误，就定为1，其余的2-3视情况而定。 |
| 404错误 | 3 | 同上 |
| JS报错 | 1-3 | 在客户端捕捉到错误信息，确定优先级后上报故障信息。 |
| 小程序报错 | 1-3 |

### 信息收集格式

|  |  |  |
| --- | --- | --- |
| **字段** | **说明** | **备注** |
| appname | 应用名称 | 具体信息来源的应用名称，例如存客宝、大洋、碎片时间 |
| end | 端名 | 具体是在哪个端发生的,由信息来源自定义  参考：h5、mini、service、pc、backend |
| host | 主机名 |  |
| position | 故障定位 | 这个由信息来源自定义，便于快速定位到信息发生的位置 |
| msg | 故障摘要 | 故障信息摘要 |
| detail | 故障详情 |  |
| level | 故障级别 | 0-5，注意0表示不是故障 |
| type | 类别 | info=运行信息  debug=调试信息  error=故障信息，当level>0时，才是故障 |
| happentime | 发生时间 | 故障发生的时间，由信息源决定 |
| env | 环境信息 | 信息发生时，一些环境信息（POST、GET、用户信息、环境变量等），便于还原现场 |

以上格式的信息，保存成json格式传给redis。

参考代码：

jedis.rpush(QUEUE\_KEY, message);

### 信息收集并保存

1、信息收集处理

Spring Boot 提供了 RedisMessageListenerContainer 类，它充当消息监听器的容器。一旦有新的消息发布到所订阅的频道上，RedisMessageListenerContainer 就会自动触发监听器中的 onMessage() 方法，并将消息传递给该方法。方法里处理完消息之后，使用redisTemplate.delete(channel)删除消息。

2、信息保存

信息最终是保存在公司内部的ES，用ES才能处理大规模数据和即时查询。以下是保存在ES的格式：

|  |  |  |
| --- | --- | --- |
| **字段** | **说明** | **备注** |
| appname | 应用名称 | 具体信息来源的应用名称，例如存客宝、大洋、碎片时间 |
| end | 端名 | 具体是在哪个端发生的,由信息来源自定义  参考：h5、mini、service、pc、backend |
| host | 主机名 |  |
| position | 故障定位 | 这个由信息来源自定义，便于快速定位到信息发生的位置 |
| msg | 信息 | 故障信息 |
| detail | 故障详情 |  |
| level | 故障级别 | 0-5，注意0表示不是故障 |
| type | 类别 | info=运行信息  debug=调试信息  error=故障信息，当level>0时，才是故障 |
| happentime | 发生时间 | 故障发生的时间，由信息源决定 |
| savetime | 保存时间 | 保存到es的时间 |
| env | 环境信息 | 信息发生时，一些环境信息（POST、GET、用户信息、环境变量等），便于还原现场 |
| raw | 原始信息 | 故障信息原原本本保存，方便统一搜索关键字 |
| md5 | Md5 | 通过raw计算出md5值，为的是辨别出重复的信息，以免重复处理。 |

### 通知负责人

|  |  |  |
| --- | --- | --- |
| **触发条件** | **发送内容** | **发送时间** |
| 接收到1-2级故障 | 系统新增一条X级故障：xxxxxxxxxxx。 | 即时 |
| 系统仍有待处理和处理中的故障 | 预警中心：系统仍有x条未完成的故障。其中x级故障x条，x级故障x条。 | 每两个小时，22点-8点不发送 |
| 总结前一天的故障信息 | 昨日共有x条新增故障，处理了x条故障。 | 每天九点 |
|  |  |  |

以上信息发送到指定的微信群。

# 数据库设计

|  |  |  |
| --- | --- | --- |
| **WeChatMoment 微信朋友圈** | | |
| **字段名** | **类型** | **备注** |
| wechatAccountId | int | 客服id |
| wechatFriendId | int | 朋友id |
| snsId | string | 朋友圈id |
| type | int | 类型  1：图文  2：文本  3：链接  15：视频  28：视频号  30：小程序  42：QQ音乐 |
| commentList | string | 评论 |
| likeList | string | 点赞 |
| createTime | Long | 发布时间 |
| content | string | 内容 |
| coverImage | string | 当type=3时，封面图片 |
| title | string | 当type=3和30时，标题 |
| lat | string |  |
| lng | string |  |
| location | string |  |
| urls | string | 当type=3时，urls[0]代表链接  当type=28时，urls[0]代表封面图片 |
| userName | string | 发表朋友圈的好友用户名 |
| status |  | 0=未采集资源  ~~1=已发送采集请求~~  2=已接收到返回，请求结束  只有当type=1、15时，status才会为0，其他type，status都是2 |

|  |  |  |
| --- | --- | --- |
| **WeChatMomentCollectCron 微信朋友圈采集计划任务** | | |
| **字段名** | **类型** | **备注** |
| sId | 主键 | 由WeChatId+’-’+WeChatFriendId组成 |
| WeChatId | 参照 |  |
| WeChatFriendId | 参照 |  |
| sPrevSnsId | 文本型 | 默认是0，表示第一页朋友圈，然后取最后一条朋友圈的snsid，为下一页朋友圈做准备 |
| sCollectInterval | 文本型 | 采集间隔区间，单位分钟。例如：30-60，表示30-60分钟采集一次。可以在这个区间内随机取一个值作为下一次采集的时间。默认值：30-60 |
| sCollectPeriod | 文本型 | 如果空，表示全时间段都可以采集  如果有设置，例如:22-8，表示22点到第二天8点才能采集 |
| dNextCollectTime | 日期时间型 | 下一次采集时间，这个字段是由lCollectInterval和sCollectPeriod共同决定 |
| dLastCollectTime | 日期时间型 | 上次采集时间 |
| lTryTime | 整型 | 重试次数，如果超过3次，就不允许再次采集朋友圈。就要预警开发人员。 |
| bFullCollected | 布尔型 | 是否已全量采集。bFull=1时，首次需要进行一遍全量采集，以后都是增量采集。全量采集后，此字段设置为1。初始为0 |
| bFull | 布尔型 | 是否全量采集 |
| bEnable | 布尔型 | 是否启用 |

|  |  |  |
| --- | --- | --- |
| **WeChatRoomCollectCron 微信聊天群采集计划任务** | | |
| **字段名** | **类型** | **备注** |
| sId | 主键 | 由WeChatRoomId+’-’+sWeChatId组成 |
| WeChatRoomId | 参照 | 群ID |
| sWeChatId | 文本型 | 群成员的微信ID |
| bEnable | 布尔型 | 是否启用 |

|  |  |  |
| --- | --- | --- |
| **ErrorTask 故障派工单** | | |
| **字段名** | **类型** | **备注** |
| lId | 自增键 |  |
| sName | 文本 | 故障信息 |
| sErrorCode | 文本 | 故障代码，保存MD5代码，为的是同一个待处理的事件不要重复提交 |
| dNewTime | 日期时间型 | 提交的时间 |
| LevelId | 列表 | 故障等级 |
| ChargerUserId | 参照 | 负责人 |
| ChargerUpUserId | 参照 | 负责人上级 |
| dResponseTime | 日期时间型 | 负责人响应时间，负责人接到故障通知，点接收的时间 |
| dDoneTime | 日期时间型 | 负责人处理完成的时间 |
| dLastNotifyTime | 日期时间型 | 上一次通知的时间  1级故障：每5分钟  2级故障：每15分钟  3级故障：每个小时  其他故障：每6个小时 |
| fTimeUse | 浮点 | 完成用时，精确到1位小数，单位是H。 |
| StatusId | 列表 | pending：待处理  handling：处理中  done：处理完成 |
| sRawMsg | 长文本 | 故障的原始信息 |

|  |  |  |
| --- | --- | --- |
| **DouYinClient 抖音客户** | | |
| **字段名** | **类型** | **备注** |
| id | 自增键 |  |
| nickname | 文本 | 昵称 |
| contact | 文本 | 联系方式，多个用半角逗号隔开 |
| avatar | 文本 | 头像 |
| createTime | 时间 | 创建时间 |
| raw | 文本 | 保存原本json信息，以便日后分析 |

|  |  |  |
| --- | --- | --- |
| **WeChatUsageLog** | | |
| **字段名** | **类型** | **备注** |
| WeChatId | 整型 |  |
| TypeId | 列表型 | 1=加好友  2=发送群消息  3=发送私信 |
| TargetId | 文本型 | 目标对象的Id |
| dNewTime | 日期型 | 创建时间 |

|  |  |  |
| --- | --- | --- |
| **WeChat 微信号** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 | 微信ID |
| lPerDayReqAmt | 整型 | 当前每日加好友次数，24小时之内允许加好友的次数，由此值来决定。 |
| lPerDayReqAmtDefault | 整型 | 缺省每日加好友次数，如果频繁后24-72小时再次频繁，则此值要降1. |
| dLastReqFrequentlyTime | 日期时间型 | 最近好友请求频繁时间 |
| dLastReqTime | 日期时间型 | 最近加好友的时间 |
| dReqLimitTime | 日期时间型 | 上次限制添加好友时间 |