**《存客宝系统》**

解

决

方

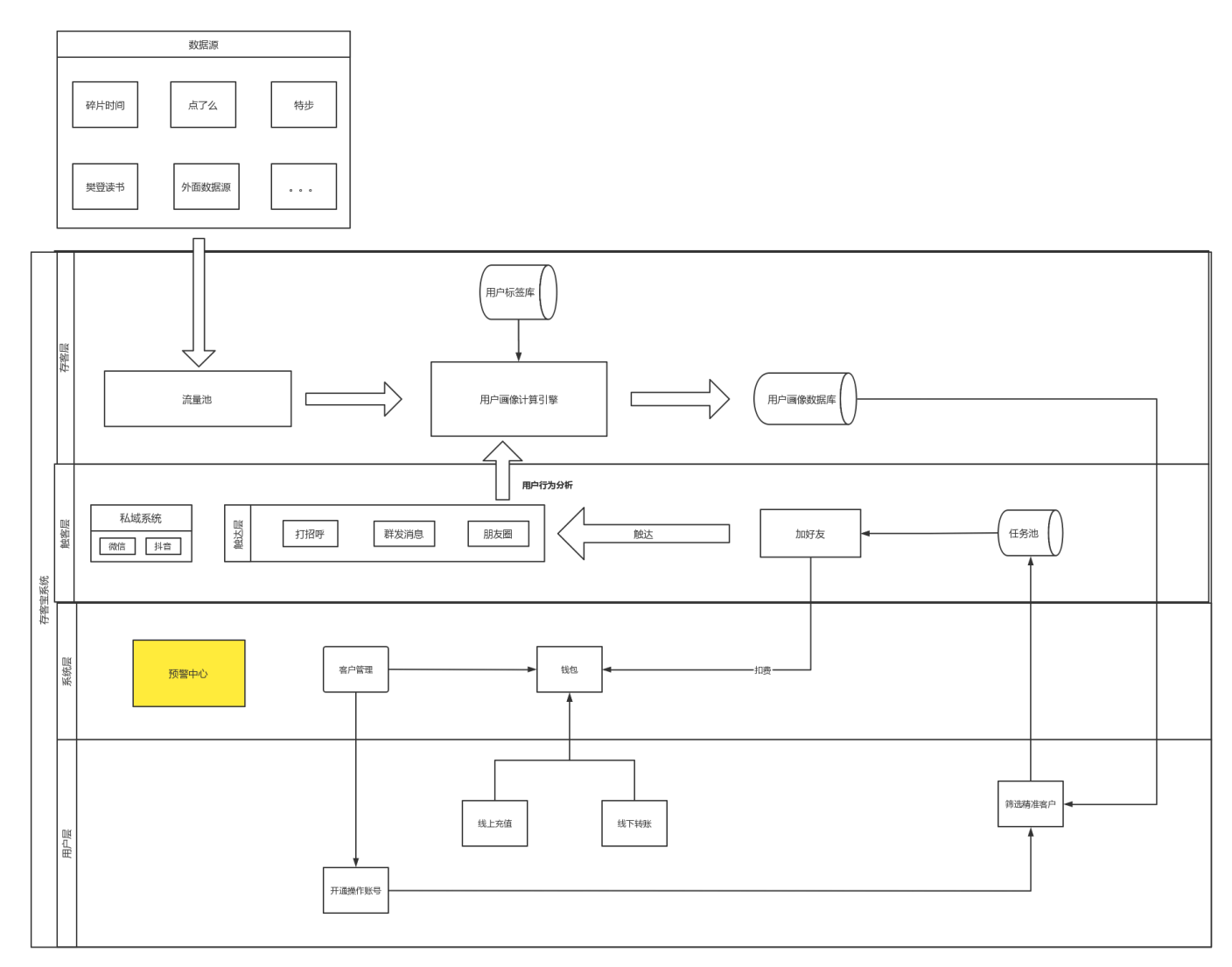
案

**卡若私域**

**2023年5月**

**版本控制信息**

# 系统业务功能流程图



# 各层次详细说明

## 项目层

严格来说，项目层不属于存客宝。它们只是以存客宝为工具，为客户实现特定的场景，以达到留存客户的目的。它们是存客宝的具体应用，它们主要目的就是获取客户的联系方式（手机号，微信号，抖音号等），然后项目层把获取到的联系方式提交给资源层。

## 任务层

在任务层里，可以为每个微信号设置添加好友的任务，每个任务都可以配置详细的添加策略，控制添加的频次。

## 流量层

所有获取的客户信息都进入流量池，再由流量池导入到任务池进行做任务。它是整个系统最有价值的部分。

## 交互层

通过私域系统提供的接口，与客户进行各种互动（打招呼，群聊，点赞，邀请入群等），实现各种场景的落地项目。

## 营销层

同2.1一样，营销层本质上也不属于存客宝系统，都是外部系统使用存客宝提供的交互层接口，进行营销的手段。

## 用户画像系统

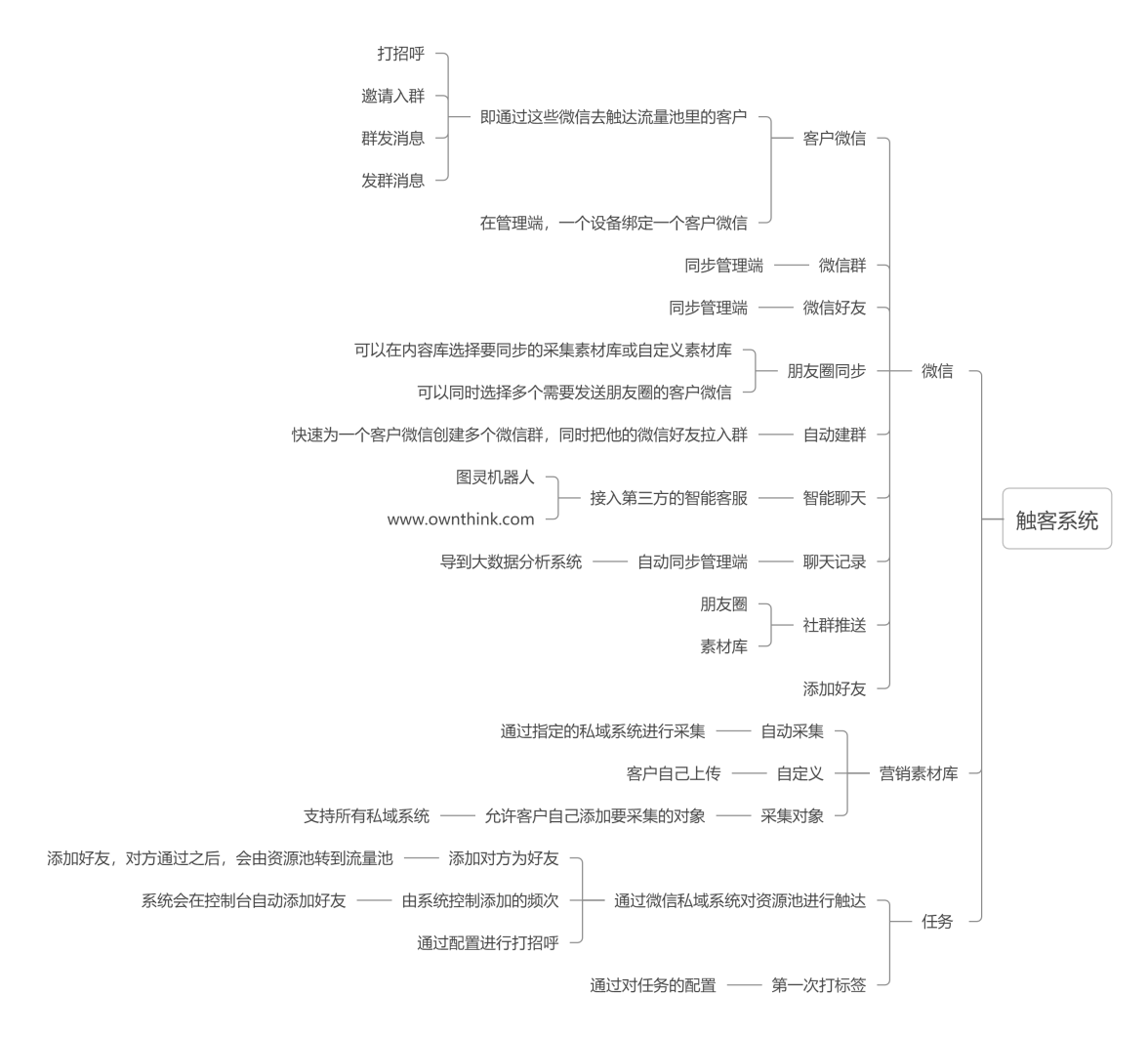
现在公司的客户数据来源，除了项目还有外部购买的客户数据，有几亿条。所以用户画像必须要用标准的大数据技术架构来实现。这个系统最终实现的是可以挑选打了标签的精准客户。

## 预警中心

系统运行的日志都集中到此进行处理，如果出现了异常，需要把异常信息发送给开发人员进行处理。

# 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

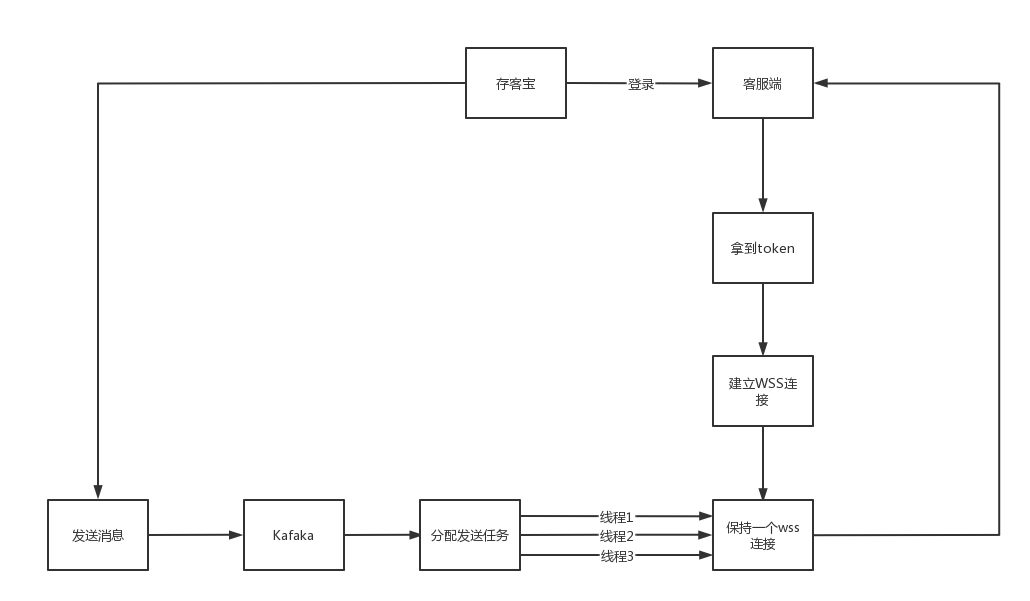
微信的私域系统提供了两个后台：

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

#### 社区推送

把素材库的内容，根据一定的规则推送到指定的微信群

### 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。

### 任务

任务的功能主要是通过私域系统加好友。这个功能是面向客户使用的。**这里暂时只考虑微信的私域系统，如果要做到支持全域，会变得很复杂。**

资源池有以下几点功能：

1. 设定添加好友的招呼语
2. 控制添加好友频次：时间间隔、每日添加人数、允许暂停
3. 通过申请后，通过智能机器人（图灵机器人或阿里云），进行聊天或者通过配置的话术按顺序发送给好友
4. 支持中途添加任务
5. 支持打标签
6. 支持海报带参数导入资源池
7. 数据统计分析
8. 变更所属客户

### 营销素材库

#### 设计目的

一方面可以通过采集目标微信的朋友圈，另一方面可以自己上传素材。目的是节约生产内容的成本以及发布到多个营销微信账号的朋友圈。

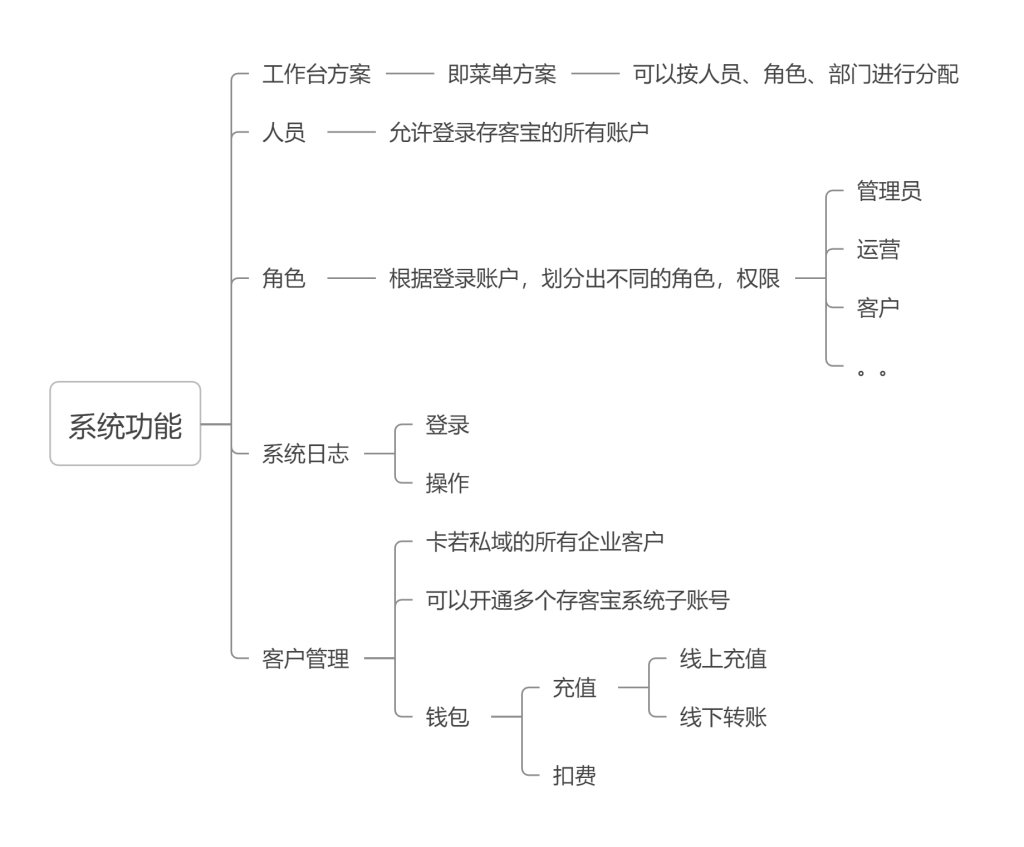
#### 存在问题

碍于现有的技术限制，采集目标微信朋友圈时，采用的是模拟手动点击的方式。采集时，客户的手机会被占用，中途不能操作手机。目前的解决方案是半夜去采集，但是客户抱怨同步太慢了。可以在存客宝的小程序端，让客户可以点击“立即采集”解决这个问题。

#### 功能

1. 采集对象，允许客户自己添加要采集的对象，要能支持任意的私域系统（微信，抖音等）
2. 针对采集对象的采集频率可以后台控制
3. 采集过来的内容，进入营销素材库
4. 同时也支持手动上传素材
5. 素材库可被私域系统使用，比如同步到朋友圈等

## 系统功能

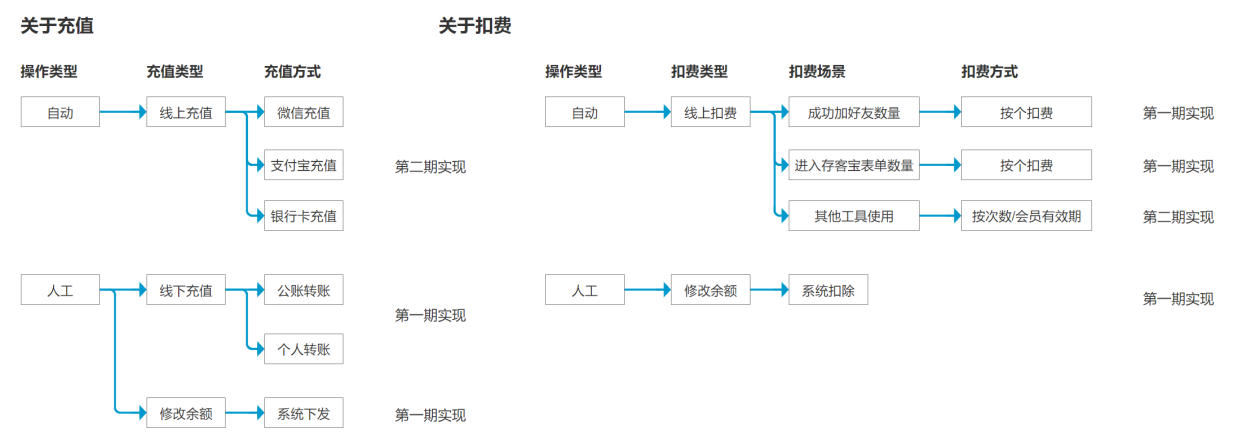


### 客户管理

#### 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。

#### 钱包



支持客户线上、线下充值，钱包的主要功能是使用系统的一些付费功能，比如：加好友、提交表单。需要注意的是，不同公司、不同任务加好友的单价可能是不一样的，需要能单独配置。

#### 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。

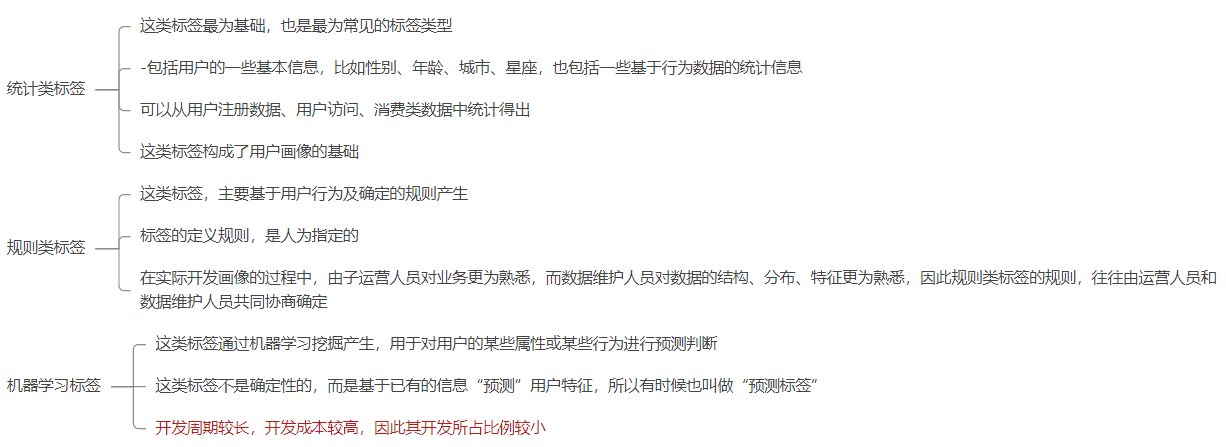
## 存客系统

### 标签管理

详见存客宝系统的思维导图，有详细规划。

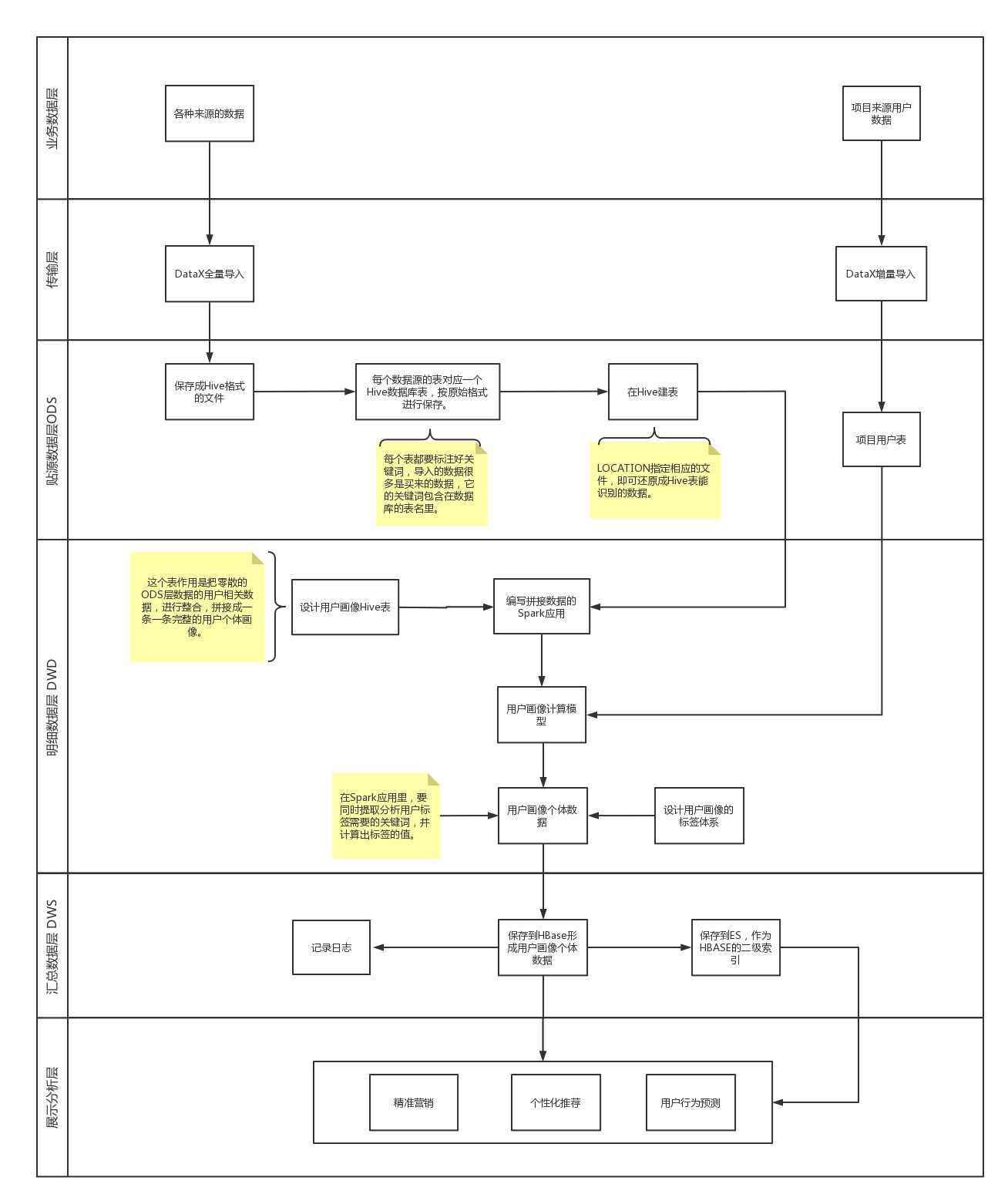
### 用户画像

标签分成三类：统计型、规则型、挖掘型。

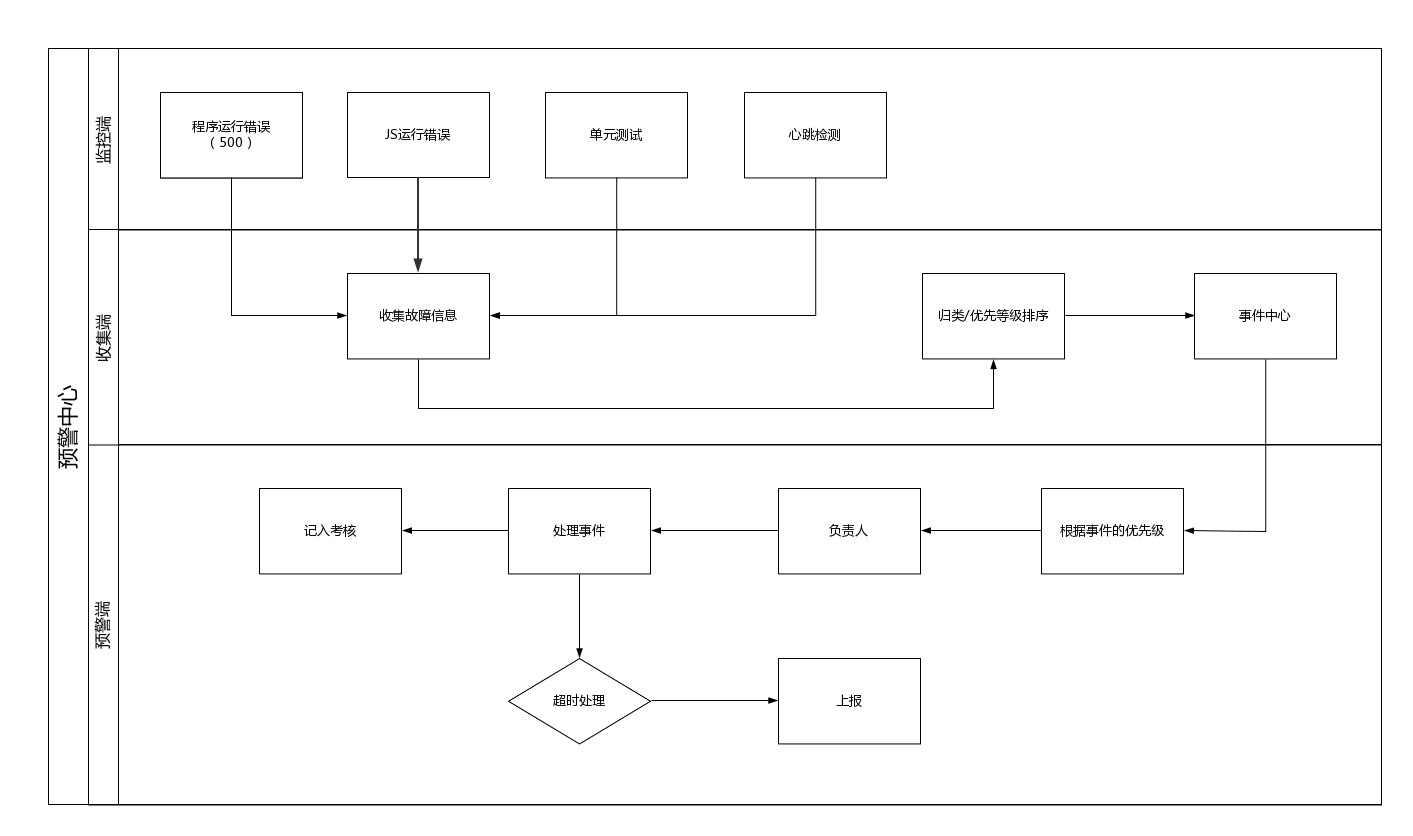


公司的数据有两类：一类是购买的用户数据，另一类是通过项目引流进入存客宝的用户数据。前者属于静态数据，一次导入到用户画像系统，但是每个数据源的处理程序不一样。后者则是动态数据，每天都要进行导入。数据格式相对固定，只需一种计算程序即可。

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

|  |  |  |
| --- | --- | --- |
| **Account 客户** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 客户名称 |
| UpId | 参照型 | 上级客户 |
| dNewTime | 日期时间型 | 创建时间 |
| dEditTime | 日期时间型 | 编辑时间 |
| NewUserId | 参照型 | 新建人员，参照人员 |
| OwnerId | 参照型 | 拥有者，参照人员 |
| EditUserId | 参照型 | 编辑人员，参照人员 |
| sManagerName | 文本型 | 负责人姓名 |
| sManagerMobile | 文本型 | 负责人手机号 |
| ManagerUserId | 参照 | 负责人账号，由这个字段来判断登录人员是否管理员 |
| fBalance | 浮点型 | 钱包余额，这个字段要配合**AccountDealFlow**表来使用，如果余额不正确，要通过这个表来统计余额。 |
| ~~bActive~~ | ~~布尔型~~ | ~~是否启用。如果禁用，该客户相关的微信、后台账号都将被禁用。~~ |

|  |  |  |
| --- | --- | --- |
| **AccountDealFlow 客户钱包交易流水** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 交易摘要 |
| AccountId | 参照型 | 归属客户 |
| dNewTime | 日期时间型 | 交易时间 |
| fBalance | 浮点型 | 本次余额 |
| sObjectName | 文本型 | 交易的对象 |
| ObjectId | 整型 | 交易的ID |

|  |  |  |
| --- | --- | --- |
| **DouYin 抖音号** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 昵称 |
| AccountId | 多选列表 | 归属客户 |
| sAvatar | 文本型 | 微信头像 |
| sRaw | JSON | 保存它的原始信息 |

|  |  |  |
| --- | --- | --- |
| **WeChat 微信号** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 微信昵称 |
| AccountId | 多选列表 | 归属客户，一个微信号可以归属多个客户 |
| sWechatId | 文本型 | 微信ID |
| sAlias | 文本型 | 微信号 |
| bOnline | 布尔型 | 是否在线 |
| sAvatar | 文本型 | 微信头像 |
| sMobile | 文本型 | 微信手机号 |
| dLastFriendRequestFrequentlyTime | 日期时间型 | 上次好友请求频繁时间 |
| dLimitTime | 日期时间型 | 上次限制添加好友时间 |

|  |  |  |
| --- | --- | --- |
| **WeChatFriend 客户微信好友** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 | 直接同步私域系统的Id |
| sName | 文本型 | 微信昵称 |
| WeChatId | 参照 | 归属微信 |
| sWechatId | 文本型 | 好友微信的原始ID |
| sAlias | 文本型 | 好友微信号，如果设定了微信号，那么sWechatId就失效了。 |
| sConRemark | 文本型 | 备注 |
| sPyInitial | 文本型 | 昵称拼音首字母 |
| sQuanPin | 文本型 | 全拼 |
| sAvatar | 附件型 | 头像 |
| GenderId | 列表型 | 性别：0未知 1男 2女 |
| sCountry | 文本型 | 国家 |
| sProvince | 文本型 | 省 |
| sCity | 文本型 | 市 |
| lAddFrom | 整型 | 来源 |
| sSignature | 文本型 | 签名 |
| sLabels | Json | 标签，Json数组 |
| dCreateTime | 日期时间型 |  |
| dDeleteTime | 日期时间型 |  |
| bDeleted | 布尔型 | 是否已删除 |
| bPassed | 布尔型 | 是否已通过 |

|  |  |  |
| --- | --- | --- |
| **WeChatChatroom 客户微信群** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 群名字 |
| WeChatId | 参照 | 在存客宝里，归属客户微信 |
| sWechatAccountId | 文本型 | 在私域系统，所属微信id |
| sAvatar | 附件型 | 头像 |
| sChatroomId | 文本型 | 群ID |
| dCreateTime | 日期时间型 |  |
| dDeleteTime | 日期时间型 |  |
| lUserNum | 整型 | 群成员的数量 |

|  |  |  |
| --- | --- | --- |
| **WeChatChatroomUser 客户微信群成员** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 群成员昵称 |
| WeChatChatroomId | 参照 | 归属群 |
| sAlias | 文本型 | 好友微信号 |
| sWechatId | 文本型 | 好友微信的原始ID |
| sAvatar | 附件型 | 头像 |
| sConRemark | 文本型 | 备注 |
| bFriend | 布尔型 | 是否好友 |

|  |  |  |
| --- | --- | --- |
| **FriendRequestTask 加友任务** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 任务名称 |
| AccountId | 参照 | 归属客户 |
| dNewTime | 日期时间型 | 创建时间 |
| dEditTime | 日期时间型 | 编辑时间 |
| NewUserId | 参照型 | 参照人员 |
| EditUserId | 参照型 | 参照人员 |
| TypeId | 列表型 | 任务类型：  douyin:抖音  poster:海报  mp：公众号  xiaohongshu：小红书  qcode：支付码 |
| dLastExecTime | 日期时间型 | 上次执行时间，注意：记录最后一次发送好友请求的时间，不是计划任务调用的时间。 |
| dNextExecTime | 日期时间型 | 下次执行时间 |
| WeChatId | 多选列表型 | 通过哪几个客服号来发起添加好友的申请，多个用逗号隔开 |
| MemoTypeId | 列表型 | 好友备注  1=缺省配置  2=自定义 |
| MemoParamId | 多选列表型 | 自定义好友备注的配置 |
| PosterId | 参照 | 海报 |
| IncomeTypeId | 列表型 | 收益结算的类型  0=关闭  1=表单录入时  2=好友通过时 |
| fIncome | 金额 | 每一条表单的收益 |
| sHello | 文本型 | 招呼语 |
| sTheme | 文本型 | 备注的主题词 |
| sTip | 文本型 | 获客成功提示 |
| lRequestInterval | 整型 | 添加好友的间隔时间（分钟） |
| lMaxPerDayRequestAmt | 整型 | 每日最大添加好友数 |
| lTargetAmt | 整型 | 目标数 |
| lNoExecAmt | 整型 | 未添加 |
| lSuccessAmt | 整型 | 通过数 |
| jTask | JSON | 私信设置 |
| bEnable | 布尔型 | 是否启用 |

|  |  |  |
| --- | --- | --- |
| **FriendRequestTaskDetail 加友计划明细** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 无业务意义 |
| NewUserId | 参照 | 创建人。这个字段是记录收益的关键字段。 |
| dNewTime | 日期时间型 | 创建时间 |
| FriendRequestTaskId | 参照 | 归属任务 |
| sPhoneOrWechatId | 文本型 | WechatId优先 |
| StatusId | 列表型 | 状态，  0未请求  1已发送请求  2已通过好友  3请求失败  4请求成功  5好友未通过验证  6=二次申请 |
| dSendTime | 日期时间型 | 好友申请的时间 |
| WeChatId | 参照型 | 归属客户微信，在执行之前，是没有分配客服的 |
| WeChatFriendId | 参照型 | 归属微信好友，如果这个字段有值。说明已通过好友申请，或者原本就是客服的好友。 |
| ThirdTaskId | 文本型 | 记录私域系统添加好友功能返回的id，用于日后追溯。 |
| dPassedTime | 日期时间型 | 好友通过的时间 |
| dFailedTime | 日期时间型 | 请求失败的时间，取不到真实的时间，只能取请求私域系统的时间了。 |
| sFailMessage | 文本型 | 请求失败时，系统返回的信息 |
| dSuccessTime | 日期时间型 | 请求私域系统添加好友成功的时间 |

|  |  |  |
| --- | --- | --- |
| **FriendRequestConfig 加友计划私信配置** | | |
| **字段名** | **类型** | **备注** |
| sId | 主键 |  |
| sName | 文本型 | 任务名称 |
| FriendRequestTaskId | 参照 | 归属任务 |
| WeChatId | 参照 | 客服号 |
| TypeId | 列表型 | 1=通过好友申请  2=主动加我为好友  3=主动发起私信 |
| jCond | JSON | 触发条件 |
| jEffectDate | JSON | 任务生效日期 |
| jEffectTime | JSON | 任务生效时间 |
| jMessage | JSON | 私信内容 |
| lInterval | 整型 | 发送间隔 |
| bEnable | 布尔型 |  |

|  |  |  |
| --- | --- | --- |
| **FriendMessageTask 好友私信发送任务** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 |  |
| FriendRequestTaskId | 参照 | 归属任务 |
| FriendRequestTaskDetailId | 参照 |  |
| dNewTime | 日期时间型 | 创建时间 |
| dSendTimePlan | 日期时间型 | 计划发送的时间，这个是由私信任务计算出来的 |
| dSendTime | 日期时间型 | 实际发送时间 |
| TypeId | 列表型 | 1=通过好友申请  2=主动加我为好友  3=主动发起私信 |
| WeChatId | 参照 | 客服号 |
| WeChatFriendId | 参照型 | 好友 |
| jMessage | JSON | 发送消息的具体配置 |
| StatusId | 列表型 | 0=待发送  1=已发送 |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **FriendMessageTpl 好友私信模板** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 |  |
| NewUserId | 参照型 | 新建人 |
| dNewTime | 时间型 |  |
| AccountId | 参照型 | 归属客户 |
| jMessage | JSON | 私信内容 |

|  |  |  |
| --- | --- | --- |
| **WeChatFriendLabel 好友标签** | | |
| **字段名** | **类型** | **备注** |
| FriendRequestTaskDetailId | 参照 |  |
| FriendRequestTaskId | 参照 | 归属任务 |
| WeChatFriendId | 参照 | 归属微信好友 |
| LabelId | 参照 | 归属标签 |
| sValue | 文本型 | 标签的值 |

|  |  |  |
| --- | --- | --- |
| **WeChatFriendLabelLib 好友标签库** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 标签名称 |
| FriendRequestTaskId | 参照 | 归属任务，当bSys=1时为null |
| UpId | 参照 | 上级标签，最多两层 |
| ValueTypeId | 列表型 | 标签值的类型：  List:枚举型，Text:文本型，Num:数值型，Bool:布尔型，Date：日期型 |
| jListValue | JSON | 标签的值是枚举型的，则保存在这个字段。  如果是布尔型的，这里存它的触发关键字（多个用换行符隔开） |
| bSys | 布尔型 | 是否系统标签，如果是不允许被用户修改。系统会通过一定的规则给好友打上标签 |
| bLeaf | 布尔型 | 是否叶子型节点，如果是意味着不能再细分。只有叶子节点才是最终带有值的节点。父节点只是分类作用。 |

|  |  |  |
| --- | --- | --- |
| **Label 标签库** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 标签名称 |
| dNewTime | 日期时间型 | 创建时间 |
| dEditTime | 日期时间型 | 编辑时间 |
| UpId | 参照 | 上级标签 |
| PathId | 文本型 | 完整的路径 |
| TypeId | 列表型 | 标签类型  1=统计型 2=规则型 3=挖掘型 |
| ValueTypeId | 列表型 | 标签值的类型：  List:枚举型，Text:文本型，Num:数值型，Bool:布尔型，Date：日期型 |
| sAttrName | 文本型 | 标签的字段名，这个当做在用户标签数据库里的字段命。一旦确定，就不能修改，否则会匹配不到数据。**必须唯一** |
| jAttrValue | JSON | 如果bLeaf=1，且标签的值是枚举型的，则保存在这个字段。 |
| bLeaf | 布尔型 | 是否叶子型节点，如果是意味着不能再细分。只有叶子节点才是最终带有值的节点。父节点只是分类作用。 |

|  |  |  |
| --- | --- | --- |
| **SysUser 人员** | | |
| **字段名** | **类型** | **备注** |
| AccountId | 参照 | 归属客户 |
| bMain | 布尔值 | 是否主账号 |

|  |  |  |
| --- | --- | --- |
| **Material 素材** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 标题 |
| dNewTime | 日期时间型 | 发布时间 |
| dCollectTime | 日期时间型 | 采集时间 |
| dEditTime | 日期时间型 | 编辑时间 |
| NewUserId | 参照 | 新建人，如果是手工发布，NewUserId则代表发布人。采集过来的设置为NULL |
| MaterialLibId | 参照 | 素材库，采集过来的时候，是null。经过分类之后的素材有值。原始的采集数据仍然是null。这样可以溯源。 |
| TypeId | 列表型 | 类型  1：纯文字  2：图文  3：链接  4：视频  5：小程序  ~~28：视频号~~  ~~30：小程序~~  ~~42：QQ音乐~~ |
| SourceId | 列表型 | 素材来源  moment=朋友圈  chatroom=微信聊天群  manual=手工发布 |
| MiniProgrameLibId | 参照 | 已入小程序库 |
| MaterialId | 参照 | 当素材进行归类时，对它源素材的引用。目的是可以追溯。 |
| sTitle | 文本型 | 当TypeId=3，必填 |
| sLink | 文本型 | 当TypeId=3，必填 |
| sHead | 文本型 | 头部内容 |
| sFoot | 文本型 | 尾部内容 |
| sContent | 文本型 | 正文内容 |
| sContentAI | 文本型 | AI生成的内容 |
| sSourceContent | 文本型 | 原始采集的正文内容，未经处理过的 |
| sThumb | 图片型 | 主图，如果没有主图，取sPic的第一张 |
| sPic | 图片型 | 图片集合 |
| VideoId | 附件型 | 视频 |
| QueueId | 列表型 | 队列  0=未编队  1=优先发送，按最新发送  2=按最早发送 |
| SourceDataId | 文本型 | 源头素材的数据ID  如果是采集来的素材，需要保存它的id，这样可以为日后判重做好准备。不同采集来源的素材，SourceDataId可以有不同的命名规则。  手工发布：/manual/{NewUserId}/{本数据的lId}  朋友圈：/moment/{objectid}/{snsid}  微信聊天群：/chatroom/{chatroomid}/{objectid}/{群消息id}  这里的{objectid}代表采集对象的ID |
| CollectObjectId | 文本型 | 采集对象的ID，有经过格式化处理：  手工发布：/manual/{NewUserId}  朋友圈：/moment/{objectid}  微信聊天群：/chatroom/{chatroomid}/{objectid}  这里的{objectid}代表采集对象的ID |
| bClassify | 布尔型 | 是否已分类，即把采集过来的原始素材，根据MaterialLib的采集配置，进行克隆的过程。默认是0 |
| bManual | 布尔型 | 素材是否手工录入 |
| bAIGenerated | 布尔型 | 是否已通过AI生成内容 |
| bReady | 布尔型 | 是否已准备就绪可以推送朋友圈或群。  影响的因素有两个：1、归属的内容库已启用AI，该内容还没经过AI处理；2、视频或图片还没上传到CDN。  两个条件必须同时具备才会取1 |

|  |  |  |
| --- | --- | --- |
| **MaterialLib 素材库** | | |
| 素材库的概念实际就是给素材进行分类，对素材的来源进行配置。 | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 素材库名称 |
| NewUserId | 参照 | 新建人 |
| sTimeLimit | 文本型 | 采集的时间段限制 |
| sConfigJson | JSON | 素材来源配置：  示例参考  {  "moment":{//采集来源：朋友圈  "objectid":[1,2,3,4]//采集对象，即微信好友的ID集合，取WeChatFriend.lId  ~~"isfull":true //是否全量采集~~  },  "chatroom":{//采集来源：微信聊天群  "objects":[  {  "chatroomid":"xxxxx",//取WeChatRoom.lId  "objectid":[1,2,3,4]//要采集的微信聊天群成员ID集合，取WeChatRoomMember.lId  }  ]  }  }  素材的来源是可以无限扩展的。每扩展一类，就要配备相应的采集器 |
| lAmt | 整型 | 素材数量 |
| sAIRequire | 文本型 | AI的要求，只有当bAIEnable=1时有用 |
| bAIEnable | 布尔型 | 是否启用AI，如果是，则会调用chatgpt，重新生成内容 |
| bEnable | 布尔型 | 是否启用 |
| MaterialLib表一旦有增删改，都要触发一次素材归类的刷新。刷新的算法如下：   1. 删除该MaterialLibId的Material相关的数据，**注意不要删除手动发表的素材ID** 2. 根据MaterialLib的sConfigJson配置，查询出所有相关素材的ID，再插入到Material 3. 完成归类的素材，要更新Material.bClassify=1   MaterialLib表一旦有增删改，也要推送一次采集对象到触客端。 | | |

|  |  |  |
| --- | --- | --- |
| **MiniProgrameLib 小程序库** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 小程序名称 |
| NewUserId | 参照 |  |
| dNewTime | 时间型 |  |
| sThumb | 图片型 | 缩略图 |
| sContent | 文本型 | 小程序的参数配置。 |
| MaterialId | 参照 | 源自哪个素材 |

|  |  |  |
| --- | --- | --- |
| **Poster 海报** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 海报名称 |
| NewUserId | 参照 |  |
| dNewTime | 时间型 |  |
| sPic | 图片型 | 图片 |

|  |  |  |
| --- | --- | --- |
| **MaterialPushMomentTask 朋友圈同步任务** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 任务名称 |
| NewUserId | 参照 | 新建的人员 |
| dNewTime | 时间型 | 新建时间 |
| lPushAmtPerDay | 整型 | 每天同步的朋友圈数量，根据sPeriod+lPushAmtPerDay，计算出推送的间隔 |
| sPeriod | 文本型 | 同步的时间段 |
| lInterval | 整型 | 根据sPeriod+lPushAmtPerDay计算间隔时间，每次保存都要重新计算该值 |
| MaterialLibIds | 多选列表 | 需要推送的素材库，可多选 |
| PushWechatIds | 多选列表 | 需要推送的客服号，可多选 |
| sPushSchedule | 文本型 | 排程，每天凌晨会进行重新排程。编辑任务，也只能第二天才能生效。 |
| TypeId | 列表型 | 1=业务号，循环发送  2=人设号，跟圈 |
| bLoop | 布尔型 | 是否循环发送 |
| lPushQueueAmt | 整型 | 已推送队列数目 |
| lRound | 整型 | 当前推送进行到第几轮。目的是给循环式任务用。从1开始。 |
| dLastPushTime | 时间型 | 上一次发布时间 |
| bDel | 布尔型 | 是否已删除 |
| bEnable | 布尔型 | 是否启用 |

|  |  |  |
| --- | --- | --- |
| **MaterialPushChatroomTask 社群推送任务** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 任务名称 |
| NewUserId | 参照 | 新建的人员 |
| dNewTime | 时间型 | 新建时间 |
| lPushAmtPerDay | 整型 | 每天同步的朋友圈数量，根据sPeriod+lPushAmtPerDay，计算出推送的间隔 |
| sPeriod | 文本型 | 同步的时间段 |
| lInterval | 整型 | 根据sPeriod+lPushAmtPerDay计算间隔时间，每次保存都要重新计算该值 |
| sPushSchedule | 文本型 | 排程，每天凌晨会进行重新排程。编辑任务，也只能第二天才能生效。 |
| MaterialLibIds | 多选列表 | 需要推送的素材库，可多选 |
| PushWechatroomIds | 多选列表 | 需要推送到的微信群，可多选 |
| TypeId | 列表型 | 发送顺序  1=按最早  2=按最新（默认） |
| bLoop | 布尔型 | 是否循环发送 |
| bImmediately | 布尔型 | 是否立即发送 |
| lPushQueueAmt | 整型 | 已推送队列数目 |
| dLastPushTime | 时间型 | 上一次发布时间 |
| lRound | 整型 | 当前推送进行到第几轮。目的是给循环式任务用。从1开始。 |
| bDel | 布尔型 | 是否已删除 |
| bEnable | 布尔型 | 是否启用 |

|  |  |  |  |
| --- | --- | --- | --- |
| **MaterialPushLog 推送日志** | | | |
| **字段名** | **类型** | **备注** | |
| lId | 主键 |  | |
| sName | 文本型 | 日志名称 | |
| dNewTime | 时间型 | 推送的时间 | |
| dTimingTime | 时间型 | 计划定时发送朋友圈的时间，这个再提交的时候计算。这个时间是要给私域系统的。 | |
| dNewMaterialTime | 时间 | 素材库的发布时间，取Material.dNewTime。这个值是计算下一个要推送的素材的依据，非常关键 | |
| TypeId | 列表型 | moment=朋友圈  chatroom=微信群 | TypeId+TaskId才能定位任务 |
| TaskId | ID型 | 推送任务的ID |
| MaterialId | 参照 | 参照素材 | |
| WeChatIds | 多选列表 | 推送给客服的ID | |
| ChatRoomIds | 多选列表 | 推送的聊天群 | |
| RemoteId | 参照 | 私域系统发朋友圈的任务ID，拿到它为的是取得朋友圈发送的状态 | |
| sContent | 文本型 | 推送的内容 | |
| sMd5 | 文本型 | 通过内容+推送客服ID列表+定时时间算出的md5值。目的是确定RemoteId。 | |
| sMd5Json | JSON | 计算出md5值的数据，为了日后可以溯源，把原始数据保留下来。 | |
| sPushScheduleTime | 文本型 | 推送的时间点，这个由排程决定。 | |
| lRound | 整型 | 如果任务是循环式，这个字段就记录这是第几轮的推送。  如果任务非循环，那就取默认值1 | |
| StatusId | 列表型 | 0=待提交  1=已提交  2=待执行  3=执行中  4=执行结束 | |
| 这个推送日志兼容所有类型的同步任务。具体的推送算法如下：   1. 根据sPeriod+lPushAmtPerDay设置，计算出lInterval，即推送的间隔，单位是分钟； 2. 有了lInterval，即把发布的时间段分成了lPushAmtPerDay个等分。在每个等分内，随机取一个时间点，作为推送的时间点； 3. 每个同步任务，都有一个排程表（保存在某个字段），按照前面说的算法事先计算好当天的发布时间点，每天都不一样（每天凌晨由计划任务对所有已启用的任务进行排程，还有就是新建任务时立即排程） | | | |

|  |  |  |
| --- | --- | --- |
| **DownloadVideoSchedule下载视频计划** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 | 取Material.lId |
| sName | 文本型 |  |
| sType | 列表型 | friend=好友消息  chatroom=群消息 |
| WechatId | 参照 | 客服号 |
| ChatroomId | 参照 | 群ID |
| MessageId | Id型 | 消息ID |
| sTencentUrl | 文本型 | 视频下载地址 |
| dSendTime | 时间型 | 发送请求时间，会有一个计划任务执行超时重置下载的任务。 |
| bSend | 布尔型 | 是否已发送下载请求 |
| bDownload | 布尔型 | 是否已下载，如果是，整个流程结束 |

|  |  |  |
| --- | --- | --- |
| **EventTrigger 事件驱动器** | | |
| **字段名** | **类型** | **备注** |
| sId | 主键 | 驱动器的英文名 |
| sName | 文本型 | 驱动器的中文名 |
| jEventHandler | JSON | 订阅该事件的处理器，允许多个 |
|  |  |  |
| 这个对象就是配置事件如何响应，jEventHandler保存了监听该事件的处理器 | | |

|  |  |  |
| --- | --- | --- |
| **Event 事件** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 事件说明，由事件源提供 |
| dNewTime | 时间型 | 事件发生事件 |
| EventTriggerId | 参照型 | 归属事件驱动器 |
| jParam | JSON | 传参，由事件处理器接收 |
| sSourcePath | 文本型 | 记录事件源的路径，可以追溯 |
|  |  |  |
|  |  |  |
| 如果客户端提交的事件，没有匹配到EventTrigger，那就是纯粹记录事件发生，没有响应。 | | |

|  |  |  |
| --- | --- | --- |
| **EventHandler 事件处理器** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| EventId | 参照型 |  |
| EventTriggerId | 参照型 | 归属事件驱动器 |
| EventHandlerId | 参照型 | 响应的事件处理器 |
| dDoneTime | 时间型 | 事件响应完成时间 |
| StatusId | 列表型 | 0：待响应  1：处理中  2：处理完成  -1：处理失败 |
| sFailReason | 文本型 | 失败原因 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **UserIncome 用户收益** | | |
| **字段名** | **类型** | **备注** |
| lId | 主键 |  |
| sName | 文本型 | 收益摘要 |
| dNewTime | 时间型 | 入账时间 |
| NewUserId | 参照 | 收益用户 |
| fIncome | 金额 | 收益金额，注意：这个字段可正可负。正表示收益，负表示提现（如果有提现模块）。 |
| sObjectName | 文本型 | 记录发生收益的对象 |
| ObjectId | ID型 | 记录发生收益的对象的ID |
|  |  |  |