Push推送系统设计文档

|  |  |
| --- | --- |
| 模块名称 |  |
| 所属系统 |  |
| 模块负责人 | 郝文智 |
| 项目负责人 |  |
| 文档提交日期 | 2019/11/18 |

海豚家

修改记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | 修改后  版本号 | 修改内容简介 | 修改日期 | 修改人 |
|  | V1.0.1 | 增加中断续发功能 | 2019/12/3 | 郝文智 |
|  | V1.1.0 | 增加按指定文件(用户UID、手机号)发送功能 | 2020/4/14 | 郝文智 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目 录

[Push推送系统设计文档 1](#_Toc29287982)

[1 背景和目的 1](#_Toc29287983)

[2 设计目标 1](#_Toc29287984)

[2.1 功能目标 1](#_Toc29287985)

[3 设计思路及折衷 1](#_Toc29287986)

[3.1 框架选择 2](#_Toc29287987)

[3.2 已有数据梳理 2](#_Toc29287988)

[3.3 存储引擎选择 2](#_Toc29287989)

[3.4 详细的表结构设计 2](#_Toc29287990)

[3.4.1 Push任务表 3](#_Toc29287991)

[4 模块设计 3](#_Toc29287992)

[4.1 任务保存逻辑 3](#_Toc29287993)

[4.2 Push任务消费消息逻辑 3](#_Toc29287994)

[4.3 Pusher脚本逻辑 4](#_Toc29287995)

[4.3.1 消息推送 4](#_Toc29287996)

[4.3.2 中断续发 4](#_Toc29287997)

[5 接口设计 7](#_Toc29287998)

[6 设计评审意见 7](#_Toc29287999)

[7 附件及参考资料 7](#_Toc29288000)

# 背景和目的

老板驱动

# 设计目标

高性能、高稳定

## 功能目标

要实现的功能点

1、后台设置用户的过滤条件

2、测试用户是否能正常收到短信 or Push消息

3、建立延迟任务,触发Pusher脚本,遍历所有用户,按照后台预置的五个条件筛选 (注册时间、订单支付时间、最后支付时间、累计订单数量、品类偏好),满足条件的用户将会收到短信 or Push消息

# 设计思路及折衷

1、Push任务通过Laravel自带database类型队列实现延迟发送效果,这里为啥不用其它类型的队列,如SQS、Redis等,主要考虑到SQS需要维护成本,Redis性能上较database要高,但是考虑到项目对延迟性能要求并不高,而且最重要的是database可以很直观的在rdb上看到执行情况,由于是延迟事件,因此在事件未发生之前很有可能用户进行变更,因此这里增加了任务version的字段,并且在进行任务消费的时刻,会首先进行任务状态、任务版本、任务触发时间点是否跟计划时间点差距过大这三个因素的校验逻辑,这样完全规避了用户误操作、多次修改而带来脚本重复触发的风险

2、Push推送脚本用到了“分治”的思想,采用GO开发,主要考虑到并行计算的特性,以及高性能、低资源占用(内存),利用多协程分不同的ID区间 (select \* from tableName where id > offset limit 50)并行读取用户大表,理论上性能应该是成倍的,但实际大多是打折的,但适当的设置协程数理论上是可以让单机的性能最大化的,Push和短信主要用到公司内网服务(API),两种类型均是通过积攒用户到一定数量(Push是450,SMS是950)然后再进行一次发送操作(HTTP请求),这样可以让网络交互次数大量减少,进一步提升性能,分批跑完后将结果(发送成功、失败数)汇总并保存

3、脚本后续考虑到异常中断的情况,即“中断续发”问题,主要考虑到三种情况: 进程被终止(信号)、DB超时(连接超时、请求数据超时)、HTTP超市(连接超时、请求数据超时),解决方案就是在前面说的三种情况发生时触发保存当前任务位置信息、然后利用这个位置信息进行手动重跑达到“中断续发”的目的,但是进程如果被强制的Kill(SIGKILL),这种情况目前能想到的解决方案就是利用单独的一个协程轮询地将任务位置信息进行保存,这种方式只能一定程度上降低损失,并不能从根本上解决

## 框架选择

服务端采用Laravel 5.5框架

## 已有数据梳理

|  |  |
| --- | --- |
| 数据 | 表 |
| 用户属性表 | user\_attribute\_list  user\_id 用户ID,int类型,必填字段,唯一索引  user\_phone 手机号,int类型,默认为0  create\_date 用户注册时间,int类型,格式为20191119  pay\_date 最后一次支付时间距离今天天数,int类型  1\_day、3\_day … 360\_day 近1、3、7…360天有无订单,int类型,1为有,0为没有  sum\_day 所有18、19年有无订单,int类型,1为有,0为没有  order\_num 累计订单总数,int类型  like\_category 类目: -1表示没有; 大于0的四位数为正常类目 |

## 存储引擎选择

MySQL InnoDB

## 现有表结构修改

# 按文件内容进行消息推送

alter table dfs\_haitun . yunying\_push\_tasks

add `data\_source` varchar(8) NULL default 'db' COMMENT '数据来源: db 数据库; file 文件;' after `type`,

add `filename` varchar(256) NULL default '' COMMENT '文件名;数据来源为file时会用到;' after `data\_source`;

## 详细的表结构设计

### Push任务表

CREATE TABLE `yunying\_push\_tasks` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT COMMENT '任务ID',

`task\_name` varchar(64) NOT NULL COMMENT '任务名称',

`begin\_ts` datetime NOT NULL COMMENT '开始推送时间',

`end\_ts` datetime DEFAULT NULL COMMENT '推送结束时间',

`type` varchar(4) NOT NULL COMMENT '推送类型: push push类型; sms 短信类型;',

`sms\_content` varchar(255) DEFAULT '' COMMENT '短信内容',

`test\_mobile` char(11) DEFAULT '' COMMENT '测试手机号',

`push\_title` varchar(64) DEFAULT '' COMMENT 'push标题',

`push\_content` varchar(128) DEFAULT '' COMMENT 'push内容',

`push\_link` varchar(255) DEFAULT '' COMMENT 'push链接',

`test\_user\_id` bigint(20) unsigned DEFAULT '0' COMMENT '测试用户ID',

`registration\_time\_logic` varchar(16) NOT NULL COMMENT '注册时间逻辑动作: invalid 无效; exclude 排除; or 且包含; and 并包含;',

`registration\_from\_date` date DEFAULT NULL COMMENT '注册开始时间',

`registration\_to\_date` date DEFAULT NULL COMMENT '注册截止时间',

`pay\_date\_logic` varchar(16) NOT NULL COMMENT '订单支付时间逻辑: 同 registration\_time\_logic',

`pay\_date\_type` varchar(16) NOT NULL COMMENT '支付时间类型: 1\_day 近1天; 3\_day 近3天; .... sum\_day 所有(18、19年);',

`pay\_date\_status` tinyint(3) unsigned NOT NULL COMMENT '支付时间状态: 1 有订单 0 无订单',

`last\_pay\_date\_logic` varchar(16) NOT NULL COMMENT '品类偏好逻辑: 同 registration\_time\_logic',

`last\_pay\_days\_symbol` varchar(5) NOT NULL COMMENT '最后支付时间比较运算符: =、>、>=、<、<=',

`last\_pay\_days` int(10) unsigned DEFAULT '0' COMMENT '最后支付近N天',

`sum\_orders\_logic` varchar(16) NOT NULL COMMENT '品类偏好逻辑: 同 registration\_time\_logic',

`sum\_orders\_symbol` varchar(5) NOT NULL COMMENT '累计订单数量操作符号: =、>、>=、<、<=',

`sum\_orders\_num` int(10) unsigned DEFAULT '0' COMMENT '累计订单数量',

`user\_type\_logic` varchar(16) NOT NULL COMMENT '用户类型逻辑: 同 registration\_time\_logic',

`user\_types` varchar(32) DEFAULT '' COMMENT '用户类型,多个类型用逗号分隔,如: 1,2,3',

`reg\_source` varchar(32) DEFAULT '' COMMENT '用户注册来源: 1 砍价; 2 0元礼; 3 1分购; 4 拼团; 5 玩赚金豆; 6 免费拿; 7 新人0元购; 8 邀请有礼; 9 自然增长; 10 信息流; 11 其他;',

`reg\_source\_logic` varchar(16) NOT NULL COMMENT '用户注册来源逻辑: 同 registration\_time\_logic',

`avg\_amount` int(10) unsigned DEFAULT '0' COMMENT '平均客单价,单位: 分',

`avg\_amount\_symbol` varchar(5) NOT NULL COMMENT '平均客单价操作符号: =、>、>=、<、<=',

`avg\_amount\_logic` varchar(16) NOT NULL COMMENT '平均客单价逻辑: 同 registration\_time\_logic',

`sum\_amount` int(10) unsigned DEFAULT '0' COMMENT '累计金额,单位: 分',

`sum\_amount\_symbol` varchar(5) NOT NULL COMMENT '累计金额操作符号: =、>、>=、<、<=',

`sum\_amount\_logic` varchar(16) NOT NULL COMMENT '累计金额逻辑: 同 registration\_time\_logic',

`last\_payment\_amount` int(10) unsigned DEFAULT '0' COMMENT '最后一次订单金额,单位: 分',

`last\_payment\_amount\_symbol` varchar(5) NOT NULL COMMENT '最后一次订单金额操作符号: =、>、>=、<、<=',

`last\_payment\_amount\_logic` varchar(16) NOT NULL COMMENT '最后一次订单金额逻辑: 同 registration\_time\_logic',

`uid\_suffix` varchar(18) DEFAULT '' COMMENT '用户ID后缀',

`uid\_suffix\_logic` varchar(16) NOT NULL COMMENT '用户ID后缀逻辑: 同 registration\_time\_logic',

`debug\_mode` tinyint(3) unsigned DEFAULT '1' COMMENT '调试模式: 1 开启; 2 关闭;',

`like\_category\_logic` varchar(16) NOT NULL COMMENT '品类偏好逻辑: 同 registration\_time\_logic',

`like\_category\_type` int(11) DEFAULT '0' COMMENT '品类偏好分类: 0为所有分类',

`create\_ts` datetime NOT NULL COMMENT '创建时间',

`success` int(10) unsigned DEFAULT '0' COMMENT '成功数量',

`failed` int(10) unsigned DEFAULT '0' COMMENT '失败数量',

`creator` varchar(64) DEFAULT '' COMMENT '创建人',

`status` tinyint(3) unsigned DEFAULT '1' COMMENT '状态：1 正常 2 删除',

`version` int(10) unsigned NOT NULL DEFAULT '1' COMMENT '版本号',

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='运营push任务表';

# 模块设计

## 任务保存逻辑

1、保存主要包含创建和修改,如果存在任务ID则认为是修改、反之是创建

2、保存需要对各个字段进行类型、长度分别进行校验

3、条件逻辑符号如果都为“or”(“invalid”除外)则不允许保存,因为“or”没有与之对应的有效条件(排除exclude、并且 and)

4、如果“数据来源”为“file”,则需要check下上传的数据文件字段是否为空

## Push任务消费消息逻辑

1、检查任务,有以下之一则程序退出

1、如果任务状态改变了

2、当前的版本号跟队列中版本号不一致

3、任务触发时间跟预定时间差距太大(前后相差1小时以上)

2、触发Pusher

这里目前没有设置重试,因为Pusher脚本跟PHP是完全分离的(业务解耦),会存在脚本触发失败的情况,不过目前失败的情况只有两种: 1、脚本不存在 2、PHP没有Pusher脚本的可执行权限,出现执行失败的情况研发快速解决问题也是比较迅速的

## Pusher脚本逻辑

### 消息推送

1、多协程进行发送

如果“数据来源”是“db”,逻辑如下:

1、分多个协程按照ID offset分批跑(select \* from tableName where id > offset\_id limit 50)

2、按照任务详情中的推送规则逐条进行筛选,如果命中则放到一个数组(非竞态变量)中,当数组中的数量超过可发送的数量时(push为450,sms为950)则触发一次批量发送操作

如果“数据来源”是“file”,逻辑如下:

1. 以指定行数、指定文件前缀切割完整用户数据文件
2. 根据文件前缀扫描chunk文件集合，worker数量为chunk文件数量，传参中的worker数量为同一时刻最大能运行的worker数量

3、逐行扫描,如果符合要求(push是uid,满足int类型,sms是11位手机号)则放到一个数组(非竞态变量)中,当数组中的数量超过可发送的数量时(push为450,sms为950)则触发一次批量发送操作

2、分批运行完成最后汇总发送成功、失败数量,并更新任务完成时间、状态、发送成功/失败数量后退出

3、命令说明:

-debug 1|0 debug模式

-task-id 任务ID

-workers worker数量,目前只对数据来源为“db”的推送任务有效,数据来源为“file”的任务会按照行数进行(目前暂定10w行为一个文件块),因此worker数量是由文件行数决定的

### 中断续发

#### 中断

脚本中维护了一个全局变量restoreInfo,里面保存着任务主要信息及各个worker的结构信息



每次批量发送成功后都会更新该结构,当出现以下几种情况发生后,会触发一个全局变量interrupted = true的操作,随即各个worker在遍历用户的时候如果发现这个变量值改变了,则提前退出,随即main函数会汇总所有的worker,正常操作会对结果数据汇总并更新到对应的任务中,中断操作不会更新到任务,只是会save当前任务信息以及每个worker位置信息到临时文件以便后续RD进行恢复

1、意料之内的并且可以处理的异常

两种情况: 脚本被Kill(SIGINT、SIGTERM)、依赖的第三方超时(DB、Push、SMS连接、请求数据超时)

情况一: 设置一个bufferd channel监听系统接收到信号SIGINT或SIGTERM

情况二: 分别在DB查询时、Push、SMS发送HTTP请求时发生异常(连接超时、请求超时)

2、意料之外的并且不可处理的异常

目前能想到的情况: 系统宕机、Kill(SIGKILL)、第三方依赖服务(DB、Push Api、SMS Api)

解决方案: 单独一个协程定时进行轮询save

#### 续发

1、“-snapshot 快照文件名”进行恢复,如果指定该参数、其它选项“-task-id”、“-workers”均失效

2、文件解析、校验文件内容(用到的文件可以是中断save产生的、也可以是RD手动编辑的)

3、续发(复用worker逻辑) ,这里产品化为一个接口来触发这个脚本,主要逻辑如下:

1、检查快照文件是否存在

1、不存在说明任务根本没有正常跑或者快照文件没有权限生成

2、检查是否存在相同任务进程

1、检查是否存在任务锁(主要任务、恢复任务),失效时间暂设为20分钟(目前任务最多也就10分钟,这里没有考虑机器负载高导致脚本运行缓慢的因素),如果存在则需要RD手动解锁

#### 快照格式

1、以Push任务ID命名(如snapshot\_9527.log)

2、根据数据来源分为两种快照格式:

1、“数据来源”为“db”文件内容格式如下:



字段说明如下:

taskId: 任务ID

minId: 上次拉取的最小用户ID

maxId: 上次拉取的最大用户ID

workerCnt: 上次运行时设置的worker数量

worker0 nextOffset 第一个worker的下一个offset

worker0 maxId 第一个worker的最大Id

success: 上次中断时已经成功推送的用户数

failed: 上次中断时已经推送失败的用户数

2、“数据来源”为“file”文件内容格式如下:



字段说明如下:

taskId: 任务ID

workerCnt: 上次运行时设置的worker数量

worker0 chunk filename 第一个worker所负责的“块儿”的文件名

worker0 next line no第一个worker待发送的下一行行号

success: 上次中断时已经成功推送的用户数

failed: 上次中断时已经推送失败的用户数

# 接口设计

<http://10.81.8.30:3000/project/123/interface/api/cat_691>

# 设计评审意见

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | 问题描述 | 提出人 | 处理方式/说明 | 状态 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 附件及参考资料