目录

[快应用push开发总结 3](#_Toc14601)

[一、 设备信息保存 3](#_Toc22110)

[1、 设备id的生成 3](#_Toc5133)

[1-1生成规则 3](#_Toc20968)

[1-2生成弊端 3](#_Toc19407)

[1-3更为简单的方式保证唯一性 4](#_Toc15142)

[2、 不合法regId的过滤 4](#_Toc12209)

[3、 上报更多设备信息 5](#_Toc15696)

[4、 程序实现逻辑 6](#_Toc31201)

[4-1 程序流程图 6](#_Toc24142)

[4-2 实现逻辑核心思想描述 6](#_Toc17239)

[5、线上出错问题以及解决方式 7](#_Toc31551)

[5-1设备信息索引重复 7](#_Toc1197)

[5-2设备信息保存并发死锁 7](#_Toc25188)

[二、 消息推送 8](#_Toc29417)

[1、 厂商消息推送对比 8](#_Toc15695)

[1-1厂商消息推送token的获取 8](#_Toc16241)

[1-2厂商消息推送的频率、每天推送总量 11](#_Toc21881)

[1-3消息推送时间的限制 15](#_Toc15554)

[1-4消息推送regid批次大小 16](#_Toc20152)

[2、java消息id的生成 16](#_Toc13718)

[生成原因 16](#_Toc3622)

[生成规则 17](#_Toc29354)

[3、 异步多线程分厂商推送 17](#_Toc31093)

[4、厂商推送程序实现（单推、群推、全推） 17](#_Toc2880)

[2-1 程序流程图 17](#_Toc22989)

[2-2 程序UML类基础结构缩略图 18](#_Toc16136)

[2-3 实现逻辑描述 18](#_Toc18082)

[5、线上出错问题以及解决方式 19](#_Toc19491)

[5-1数据库保存异常regid，导致推送异常报错 19](#_Toc24609)

[5-2 vivo推送次数超出总用户限制 19](#_Toc2643)

[5-3 huawei批次regids大小不能超过100 20](#_Toc7721)

[5-4 vivo服务端返回报文变更，空指针报错 21](#_Toc7320)

[5-5 vivo权限认证失败 21](#_Toc24553)

[5-6小米三次握手连接出错 22](#_Toc21706)

[5-6 oppo单推服务限流 22](#_Toc2461)

[5-7 oppo非法appkey 23](#_Toc240)

[三、 消息推送结果统计 23](#_Toc10497)

[1、 回执与自主调用统计接口 23](#_Toc22629)

[1-1厂商的统计方式分析 23](#_Toc7782)

[1-2厂商的统计方式对比 24](#_Toc12040)

[1-3 push选择的统计方式 25](#_Toc30625)

[2、 推送结果统计的实现 26](#_Toc1803)

[2-1消息统计时序图 26](#_Toc21279)

[2-2 实现逻辑描述 26](#_Toc7709)

[2-3从redis更新到数据库逻辑描述 28](#_Toc5345)

[3、 统计的数据结构使用 28](#_Toc14597)

[3-1送达数统计 28](#_Toc17468)

[3-2点击数统计 28](#_Toc18658)

# 快应用push开发总结

## 设备信息保存

### 设备id的生成

#### 1-1生成规则

线上快应用设备id的生成规则：（总共36位）

1、时间（13）+Android（16）+厂商（1）+随机（6）

2、时间（13）+imei（16）+厂商（1）+随机（6）

3、时间（13）+md5型号后随机取（16）+厂商（1）+随机（6）

为了保证一个手机设备，快应用存在唯一的设备id采用以上的生成方式，依次当前规则不能生成则由下一个规则去生成设备Id。

#### 1-2生成弊端

那么，以上生成方式存在的问题在于：快应用数据被清空（相当于卸载快应用），那么重新安装的时候设备id又重新生成。导致，设备id对手机设备不是唯一不变的标识。

然而，快应用的数据清空操作复杂，一般清除桌面图标以及缓存清除不能是快应用真正卸载。因此，设备Id变更并不会在重新添加快应用图标到桌面时频繁发生。

#### 1-3更为简单的方式保证唯一性

1、Android（16）

2、imei（16）

3、时间（13）+md5型号后随机取（16）+厂商（1）+随机（6）

由于，基本**品牌手机**能够获取到andriodId，不能获取到，则在用户允许访问设备权限的情况下获取到imei，以上都无法获取才添加变化数值去生成作为deviceId。

这样保证，用户就算熟练快应用清空数据（卸载）操作，那么，下一次安装快应用，基本上的手机设备id都不会发生改变。

### 不合法regId的过滤

regId即为注册Id，这个值为前端调用厂商接口获取该种快应用下的唯一标识，那么消息推送也是根据regId去调用对应厂商接口推送消息。

不明原因，会出现test\_id，3，regid-abdcef这种不合法的regId上报。那么，可以在设备id上报保存之前，对这些不合法的regId进行过滤，不存于数据库。

### 上报更多设备信息

快应用除了以上两种设备信息，还包括：自定义的厂商品牌编号provider，快应用版本号version。

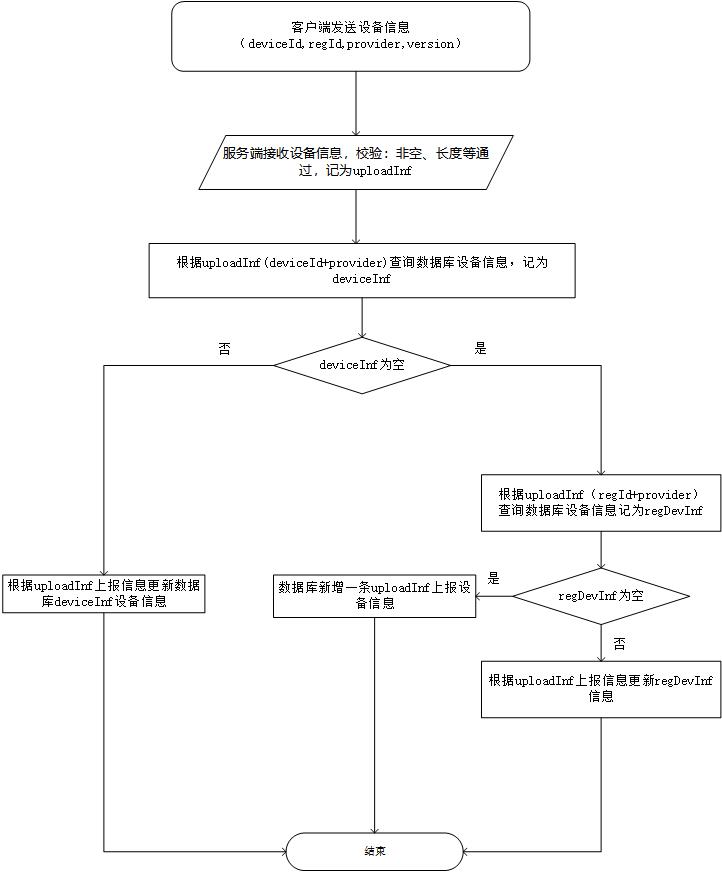
那么，更好的是上报将设备品牌、型号、imei 获取到的情况下，也一同上报数据库保存。

其中，上报设备品牌可以将品牌编号的生成由后端维护。避免了，新品牌的发生导致快应用的重新发版厂商审核，更利于有效的维护。

再者，上报更多的设备信息，发现打印设备信息错误的日志打印，那么根据imei可以更好的与厂商技术支持，协调沟通排查问题。

### 程序实现逻辑

#### 4-1 程序流程图



#### 4-2 实现逻辑核心思想描述

1、保证数据库存储设备Id的唯一性

2、保证数据库某个厂商下regId的唯一性

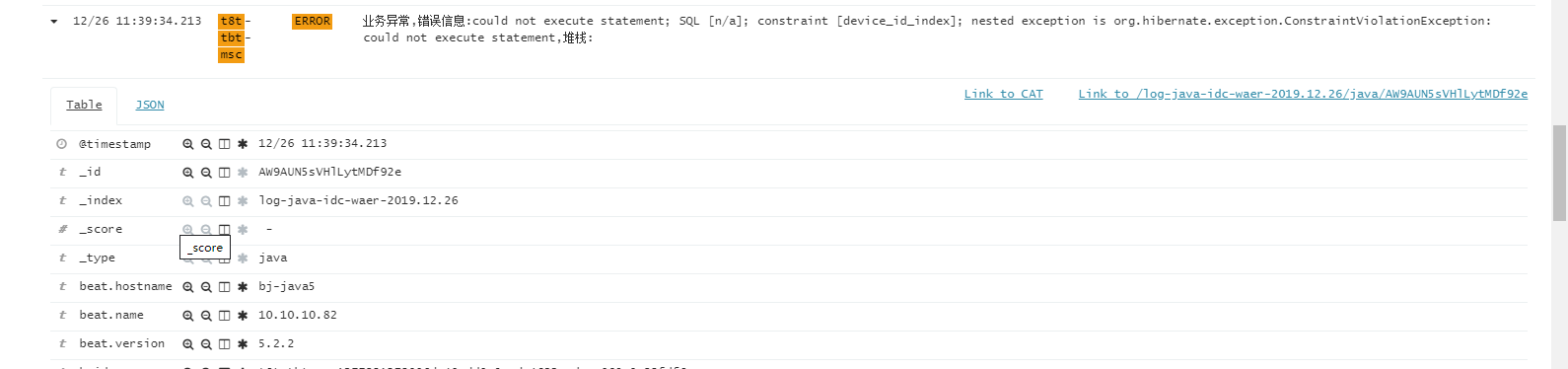
3、保证deviceId与regId一对一的双向绑定：由deviceId更新regId，由regId与厂商编号provider更新deviceId。

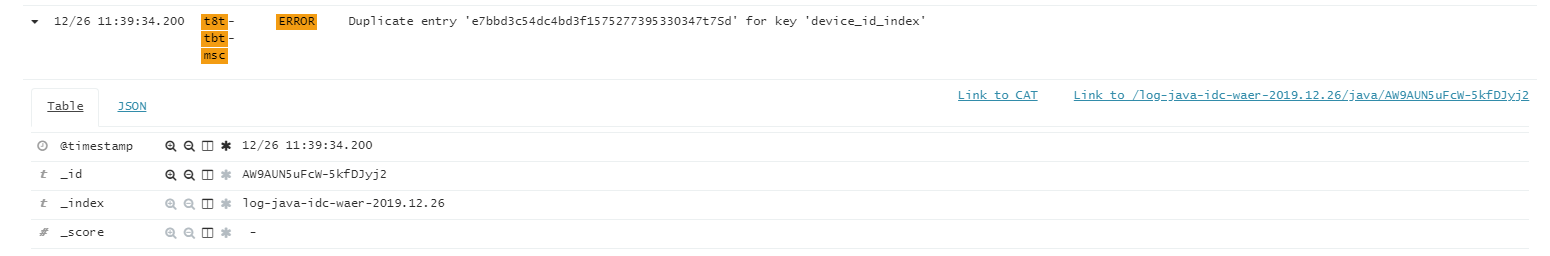
### 5、线上出错问题以及解决方式

#### 5-1设备信息索引重复

问题描述：

前端设备信息在网络不良情况下重复上报，且同时存入数据库导致数据库报错唯一索引deviceid重复。





解决方式：

对ConstraintViolationException进行异常捕捉，打印warn级别日志信息。

#### 5-2设备信息保存并发死锁

问题描述：

只出现过一次该异常，具体原因有待究查：

org.springframework.dao.CannotAcquireLockException: could not execute statement; SQL [n/a]; nested exception is org.hibernate.exception.LockAcquisitionException: could not execute statement

bsid：t8t-tbt-msc157768782484546e62bb8-dbac-4be9-b64a-77f36aea22ba



## 消息推送

### 厂商消息推送对比

#### 1-1厂商消息推送token的获取

##### 厂商对比

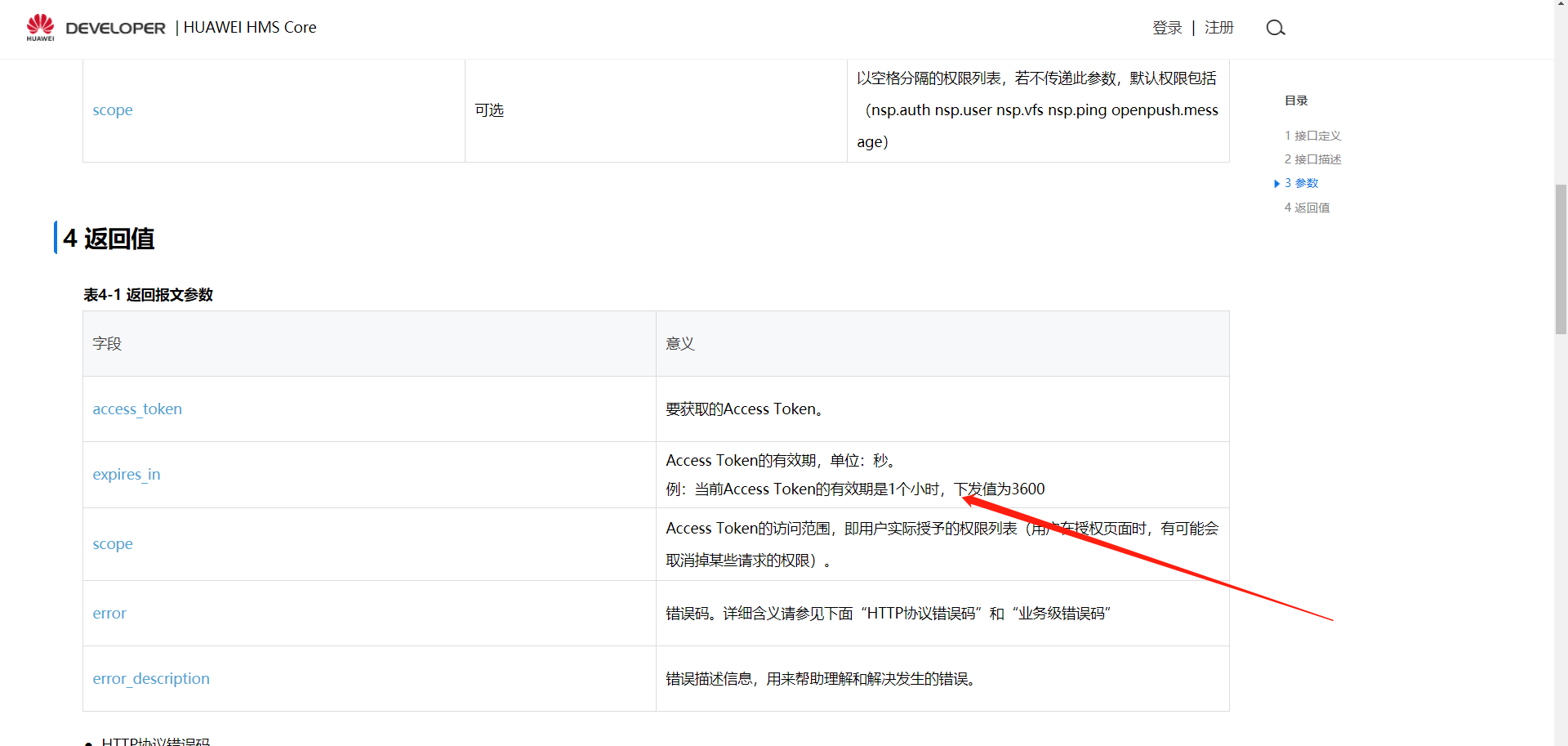
|  |  |
| --- | --- |
| Oppo | Oppo不限制，但是每次单推都获取依次token，在每天5w单推批量下可能会出现几个网络链接错误。新token官方介绍存活24小时，旧token存活10分钟。(官方文档介绍) |
| Vivo | Vivo每天只能获取1w次token，单推每次推送获取token，上万数量下token会超出总限制数报错。(官方文档介绍) |
| 小米 | 不需要获得token进行推送，详情可查看小米sdk快应用push源码  (无官方文档介绍) |
| 华为 | Token的有效期为1小时，无次数限制。官方技术支持建议做缓存。  (无官方文档介绍) |

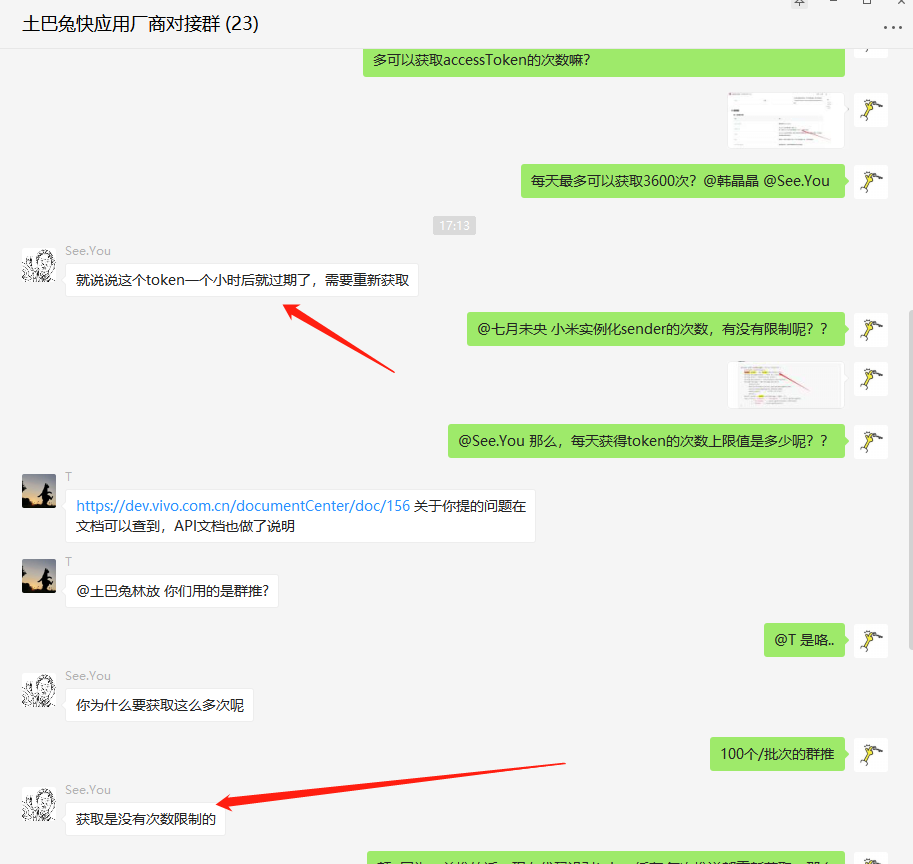
vivo





oppo

华为



##### 处理的方式

1. 对华为、vivo的token进行redis缓存，大量级减少单推获取token对厂商服务的请求
2. 线上目前对oppo的token没做缓存，原因在于oppo的sdk存在只有有参构造且无无参构造，构造器内每次实例化都实现对token的获取。那么处理方式，是在oppo网络链接错误的时候，那么便休眠一段时间进行sender实例化获取token的重试。
3. oppo线上如果确实需要token缓存才解决问题，可将oppo的sdk中像Sender，HttpClientTool等default类源码copy出来，对Sender的构造方法进行修改，进行token的缓存。

#### 1-2厂商消息推送的频率、每天推送总量

##### 厂商对比

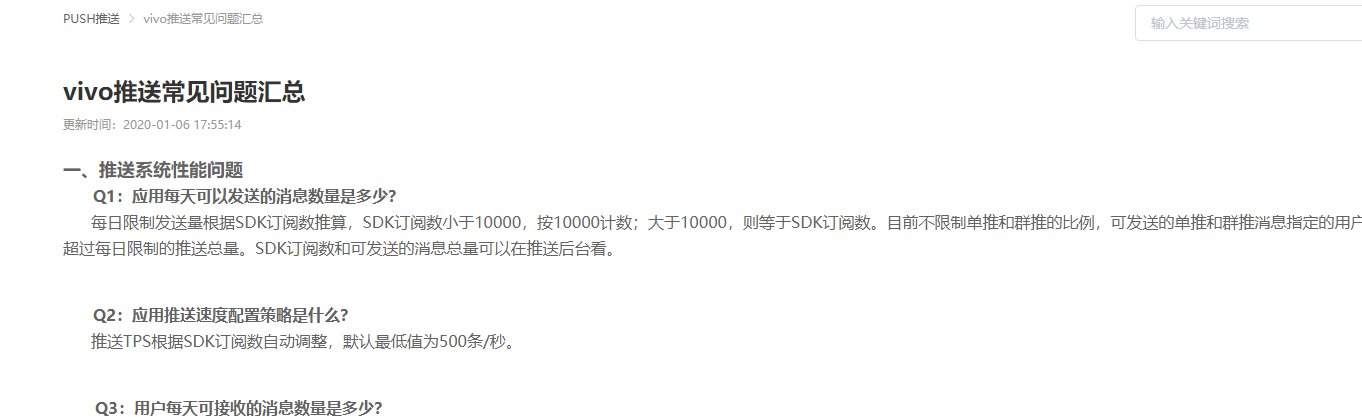
|  |  |
| --- | --- |
| Oppo | 无频率限制，有推送消息数量限制为用户数量的2倍  （详情查看官方文档） |
| Vivo | 可发送的单推和群推消息指定的用户量不得超过每日限制的推送总量  推送TPS根据SDK订阅数自动调整，默认最低值为500条/秒。  （详情查看官方文档） |
| 小米 | 目前无频率、数量限制。往后会做一些限制。  （详情查看官方文档） |
| 华为 | 无频率、总量限制（官方文档无介绍） |

1、Oppo链接地址：<https://open.oppomobile.com/wiki/doc#id=10200> 搜ctrl+f搜 技术问题

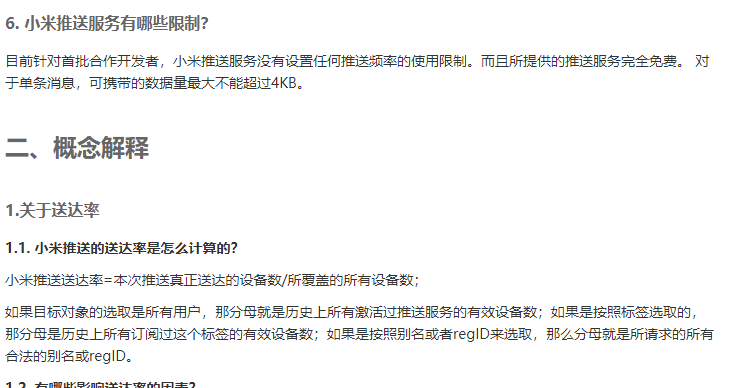




1. vivo链接地址：<https://dev.vivo.com.cn/documentCenter/doc/156>



1. 小米链接地址：<https://dev.mi.com/console/doc/detail?pId=1292#_0_5>



1. 华为链接地址：

<https://developer.huawei.com/consumer/cn/service/hms/catalog/fastapp.html?page=fastapp_fastapp_faq#%E6%8A%A5%E8%A1%A8%E5%B8%B8%E8%A7%81%E9%97%AE%E9%A2%98>

##### 处理的方式

1. oppo、vivo做推送总量超限后，将超限信息存储于redis。每次推送从redis查询是否存在超限信息，不超限则继续推送。
2. 对于厂商推送报网络错误等，小米有设置可重发次数，为了避免小米重发后手机收到几条相同的信息，必须对重发消息做id的标识，使客户端对消息去重。



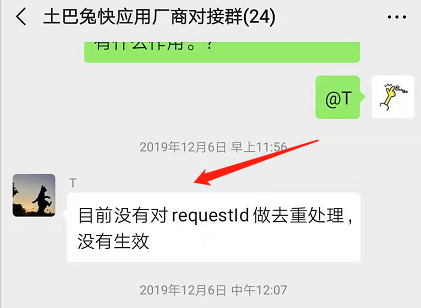
其它厂商：oppo、vivo也**有设置**消息id标识进行消息重发的去重，而华为没有。那么设可以置标志id为uuid，Oppo、vivo厂商消息推送sdk的无重发策略，需要开发者捕捉网络异常实现重发，这边不做重发。

Oppo为可选字段设置：



**Vivo为必填字段设置：（然而，厂商push服务没有对这个字段进行去重处理，仅仅是有设置）**

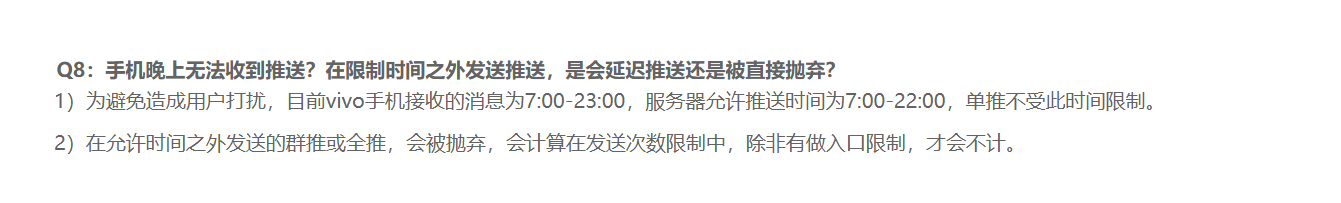




#### 1-3消息推送时间的限制

##### 厂商对比

|  |  |
| --- | --- |
| Oppo | 无介绍 |
| Vivo | 有允许发送时间段，会返回推送异常错误码 |
| 小米 | 无介绍 |
| 华为 | 无介绍 |

vivo

##### 处理的方式

1. 设置消息可推送的时间范围，消息推送前进行可推送时间的校验
2. 将今天到达推送时间结束点未推送的消息顺延平移到到下个时间起始点推送。

举个例子：今晚10点推送，那么顺延到明早8点推送。今晚11点推送，那么顺延到明早9点推送。

#### 1-4消息推送regid批次大小

##### 厂商对比

|  |  |
| --- | --- |
| Oppo | 1000个 |
| Vivo | 1000个 |
| 小米 | 1000个 |
| 华为 | 100个 |

##### 处理的方式

1. 特别注意华为快应用push的regid批次数量上限为100。而app push上限数量为1000。
2. 设置厂商统一批次推送regid数量大小：100

### 2、java消息id的生成

#### 生成原因

将消息的统计与php管理后台消息id传值进行依赖的解耦，即使不依赖外部消息id依然能对一类消息进行统计

#### 生成规则

时间戳+消息url+随机数的md5加密成32位

### 异步多线程分厂商推送

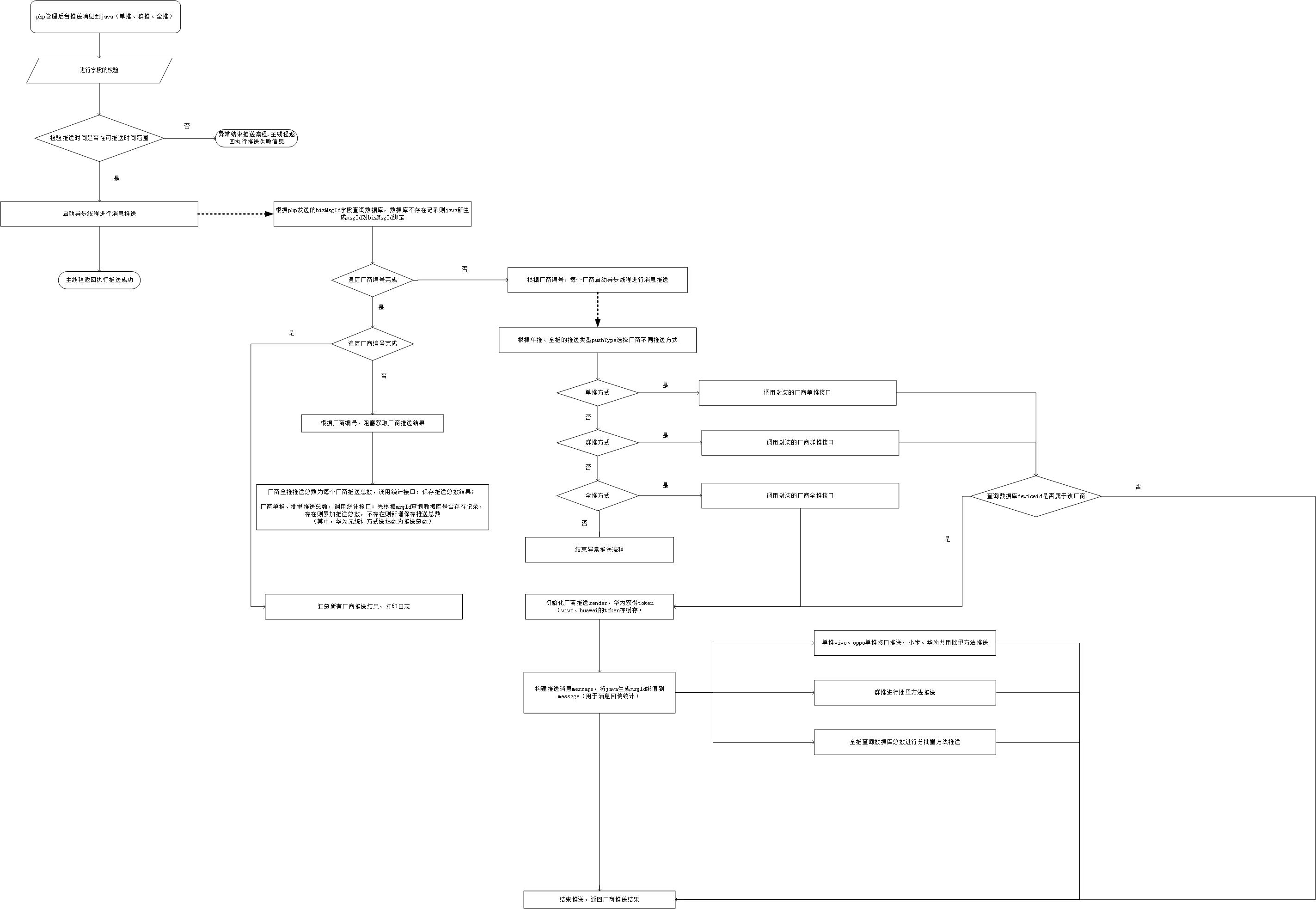
原因：

1. 更便于统计：把所有厂商混在一起推送，去统计各厂商推送数量实现逻辑更为复杂。
2. 为厂商设置统一的分批推送量：每次批次推送各厂商的量都相同，不会有的推送很少有的很多。
3. 耦合性低：代码逻辑清晰，在调试代码中更好的去找到出错的位置进行调试。在新增厂商推送，只需要配置新增厂商编号与实现厂商推送公共接口，不需要对原有的推送判断逻辑做明显修改，扩展性更好。

4、多线程提高发送效率。

### 4、厂商推送程序实现（单推、群推、全推）

#### 2-1 程序流程图



#### 2-2 程序UML类基础结构缩略图

这里厂商举例两种：vivo、oppo



#### 2-3 实现逻辑描述

1、java生成维护消息id对同一种消息进行标识绑定。

2、各厂商推送提供者实现共同的推送接口。

3、异步分厂商进行消息推送。

4、各厂商推送regid的获取：

* 1. 全量推送分页获取批次各厂商regids推送
  2. 群推、单推根据传值regids，各厂商分别去获取属于自己

部分的regids进行推送。

1. 阻塞式获取各厂商推送总数存库：
   1. 全量推送直接新增记录存库
   2. 单推、群推根据库内是否存在msgId的数据，是则更新推

送数量，否则新增记录存库

6、汇总计算各厂商推送数量总和。

### 5、线上出错问题以及解决方式

#### 5-1数据库保存异常regid，导致推送异常报错

问题描述：

设备信息上报保存，会上报”test\_id”、”regid-abcdef”等不合法regid存库。

解决方式:

1. 在设备信息保存前过滤非法regid，避免存库。
2. 在厂商sdk本地校验regid出错时，catch异常。使用厂商过滤规则过滤非法regid后重新推送。

#### 5-2 vivo推送次数超出总用户限制

问题描述：

vivo日推送限制为用户SDK订阅数，可在vivo的push管理后台查看数值。

当天进行两次分批次的全量推送，第二次推送超出限制数。返回异常码：10070



解决方式:

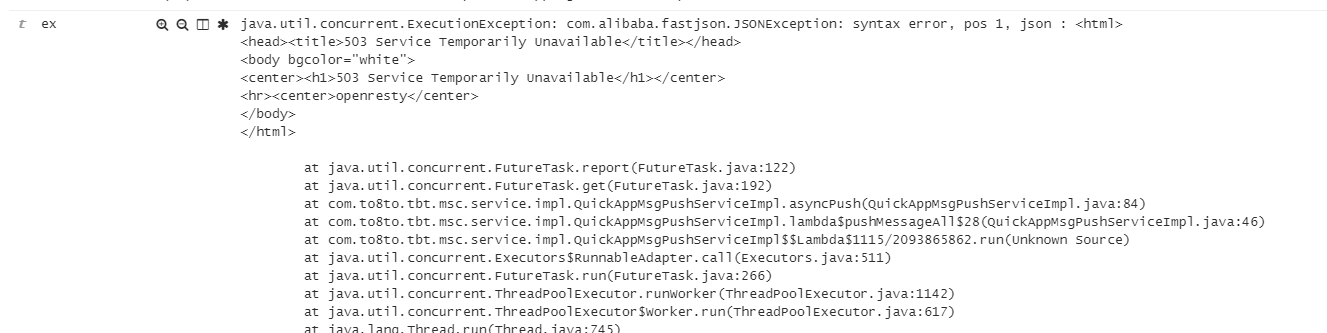
1. 产品人员每天全量推送条数设置不超过1条。
2. 对所有厂商的异常码进行检查。抽取各厂商正常的返回码，对于异常的返回码程序不执行后续的解析统计操作，直接返回全部推送失败的统计结果。
3. oppo、vivo做推送总量超限的异常码时，将超限信息存储于redis。每次推送会从redis查询是否存在超限信息，不超限则继续推送。

#### 5-3 huawei批次regids大小不能超过100

华为超限返回结果，异常码80300010：



线上批量超限的高频率的非法推送，华为直接限制推送，返回503服务不可用的text数据，导致json解析异常：



解决方式:

1、捕捉JSONException进行解析异常日志打印。

2、对所有厂商的异常码进行检查。抽取各厂商正常的返回码，对于异常的返回码程序不执行后续的解析统计操作，直接返回全部推送失败的统计结果。

3、对厂商的返回结果以及取值属性做判空处理。

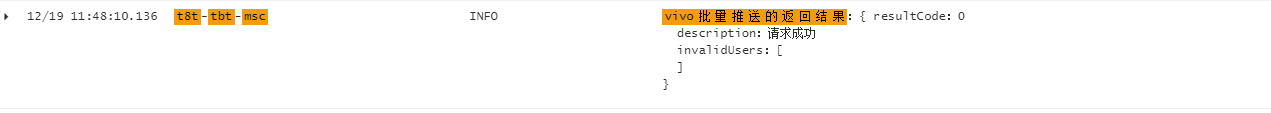
4、将厂商推送批次大小设置为100。

#### 5-4 vivo服务端返回报文变更，空指针报错

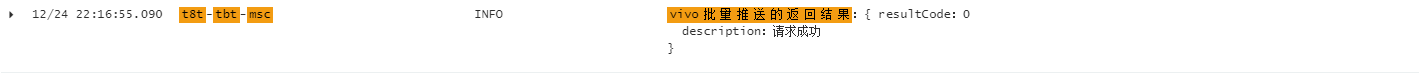
问题描述：

vivo的invalidUsers字段，vivo服务校验全部为合法用户：从原来的返回invalidUsers空列表json字段，变更为不返回值。

从如下报文格式：



变更为：



解决方式:

1、对所有厂商的返回结果以及取值属性做判空处理。

#### 5-5 vivo权限认证失败

问题描述：

vivo的token获取次数超出1w次，返回异常码：



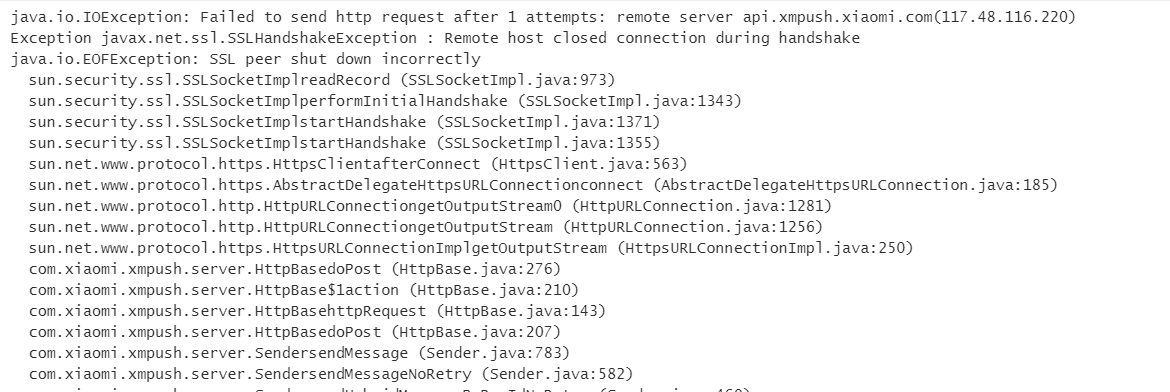
解决方式:

1、vivo、华为的token设置失效时间存缓存，采用分布式锁实现并发场景下token失效后重新获取，只调用vivo、华为推送服务去获取一次token。

#### 5-6小米三次握手连接出错

问题描述：

小米push发送群推信息，一天6w次左右推送下，会发生10次左右。三次握手失败的网络连接失败IOException。



解决方式:

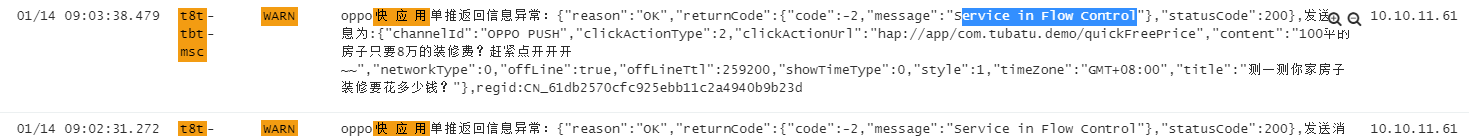
1. 设置小米sdk的网络连接失败重发次数为3，为避免重发导致消息重复。那么，需要每次调用厂商接口群推时都生成并设置一个8个字符的jobkey参数。

生成规则为：当前时间戳秒的16进制字符串。

#### 5-6 oppo单推服务限流

问题描述：

一天5w次单推下，会返回4次左右，状态码为-2的限流结果。



解决方式：

对单推限流不做重推处理，也暂时不做限制发送频率。打印warn日志，继续观察在更大推送次数下的推送情况。

#### 5-7 oppo非法appkey

问题描述：

怪异的异常，报错快应用appkey错误，问题只出现过一次，具体原因有待究查。

get token error: {"reason":"OK","returnCode":{"code":14,"message":"Invalid App Key"},"statusCode":200} at com.oppo.push.server.Auth.getAuthResultWithRetry(Auth.java:55) at com.oppo.push.server.Auth.getAuthResult(Auth.java:33) at com.oppo.push.server.Sender.setToken(Sender.java:56) at com.oppo.push.server.Sender.<init>(Sender.java:49) at com.to8to.tbt.msc.service.push.impl.OppoQuickAppPushProvider.initMsgProvider(OppoQuickAppPushProvider.java:86) at com.to8to.tbt.msc.service.push.impl.OppoQuickAppPushProvider.singlePush(OppoQuickAppPushProvider.java:70) at com.to8to.tbt.msc.service.impl.QuickAppMsgPushServiceImpl.lambda$asyncSinglePush$8(QuickAppMsgPushServiceImpl.java:113) at com.to8to.tbt.msc.service.impl.QuickAppMsgPushServiceImpl$$Lambda$1073/1325208402.call(Unknown Source) at java.util.concurrent.FutureTask

解决方式：

获取token报错非法appKey。对异常进行捕捉，打印warn日志，抛出rpc业务异常。**重现与oppo技术支持沟通**

## 消息推送结果统计

### 回执与自主调用统计接口

#### 1-1厂商的统计方式分析

oppo、vivo、小米均有callback回执接口以及可自定义回执参数进行回传，这样方便统计，以下是每个厂商推送获取统计数据方式以及特点：

1. vivo**单推不支持**自主去获得消息的统计信息，只支持**批量和全推**去主动调厂商接口获得统计信息，**单推支持回执，批量不支持回执**。那么vivo的单推到达数统计采用回执方式，而批量全推可采取调用vivo统计接口自主统计的方式。
2. oppo支持当天所有信息送达量的查询，但不支持不同的标识给予区分不同类型消息的送达量统计。**oppo支持单推、群推的回执**。
3. 小米需要维护厂商生成在返回结果中消息id与自己所生成消息id的关系，通过厂商生成在返回结果中消息id去查询统计信息。小米可以使用**厂商回执统计单推、批量全推送达量与点击量**。
4. 华为不支持消息回执以及自主去统计查询。那么华为推送总量替代华为送达量。

#### 1-2厂商的统计方式对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **厂商回执** | | | | |
| 厂商 | 单推 | 群推 | 点击量统计 | 送达量统计 |
| oppo | 支持 | 支持 | 不支持 | 支持 |
| vivo | 支持 | 不支持 | 不支持 | 支持 |
| 小米 | 支持 | 支持 | 支持 | 支持 |
| 华为 | 不支持 | 不支持 | 不支持 | 支持 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **自主调用厂商统计接口** | | | | |
| 厂商 | 单推 | 群推 | 点击量统计 | 送达量统计 |
| oppo | 不支持 | 不支持 | 不支持 | 支持 |
| vivo | 支持 | 支持 | 支持 | 支持 |
| 小米 | 支持 | 支持 | 支持 | 支持 |
| 华为 | 不支持 | 不支持 | 不支持 | 不支持 |

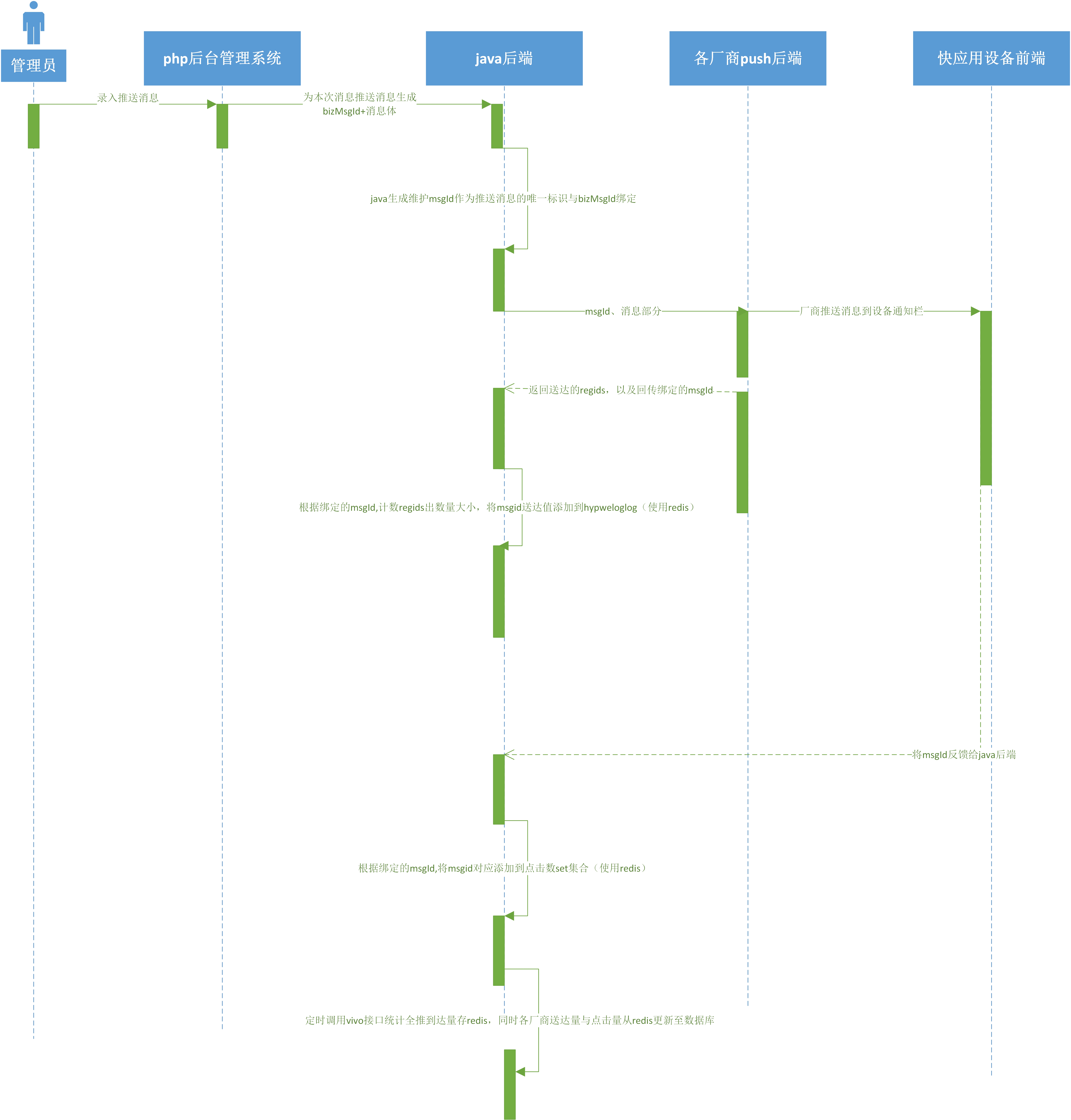
#### 1-3 push选择的统计方式

|  |  |  |
| --- | --- | --- |
| 厂商 | 单推 | 群推 |
| oppo | 回执统计（送达量） | 回执统计（送达量） |
| vivo | 回执统计（送达量） | 自主调厂商接口统计  （厂商点击量、送达量） |
| 小米 | 回执统计  （厂商点击量、送达量） | 回执统计  （厂商点击量、送达量） |
| 华为 | 推送总量替代送达量 | |

|  |  |
| --- | --- |
| 厂商 | 实际点击量 |
| oppo、vivo  小米、华为 | 用户点击链接，前端调用java接口回传msgId参数进行统计 |

### 推送结果统计的实现

#### 2-1消息统计时序图



#### 2-2 实现逻辑描述

1. php为全推（定时消息）或者单推（延迟消息）生成bizMsgId，

传递给java后端。

2、java生成维护msgId:

2-1根据bizMsgId查询数据库是否存在记录，存在则取出关联的msgId。不存在记录，**则根据（时间戳毫秒+厂商编号+url+随机数）的md5加密值。**

2-2 Java后端给各个厂商推送设置厂商回执url，以及将msgId绑定在厂商回执透传参数字段进行消息推送。

3、Java推送完成后便在数据库表push\_msg\_result新增msgid、推送总量的

每个厂商为一条记录。

**其中华为送达数设置为华为推送总量。**

**在数据库表push\_msg\_reslut\_key新增msgId\_provider的4条记录代表**

**4个厂商redis存储key的前缀，其中，vivo在表中provider\_msg\_mark存储taskId。**

4、各厂商回执到对应接口进行送达量、厂商点击量统计。（写入redis，**适应单推一条msgId消息可能长时间推送,过期时间每次更新为消息离线时间3天**）

5、前端回执到对应接口进行对应厂商点击量统计。（写入redis，**过期时间每次更新为消息离线时间3天**，定时每一小时进行数据库表送达量的更新）

6、定时任务调用vivo厂商统计接口，根据vivo全推消息构建的taskId去查询批量全推的统计值更新到redis，同时将redis各厂商点击数、到达数、实际点击数值更新到数据库。

7、根据msgId将各厂商，点击量、送达量求和返回给php管理后台。

#### 2-3从redis更新到数据库逻辑描述

1、push\_msg\_reslut\_key表查找所有redis key的消息前缀记录。

2、根据redis key前缀拼接厂商点击量、送达量、实际点击量后缀标识，查询redis是否存在记录 , 若实际点击量与送达量记录都不存在,则将该前缀从push\_msg\_reslut\_key表中删除

3、将redis中的值根据前缀(msgId与厂商编号标识provider),更新到厂商统计表中

### 统计的数据结构使用

#### 3-1送达数统计

对于送达数这种大数据量的统计实现。采用hyperloglog数据结构进行基数计数统计，特点为: 高级不精确去重的数据结构，(一般是超过一百个就开始不准确了)占用空间小(一个键最多12k,可以计算2^64个元素)。统计错误率约为0.81%。

#### 3-2点击数统计

对于点击数这种数据量可观的统计实现。采用set集合，对点击内容链接的设备id设置到set集合保存实现去重。同时设置set的大小上限，对接口攻击刷数据的场景达到点击数统计上限后预警通知。