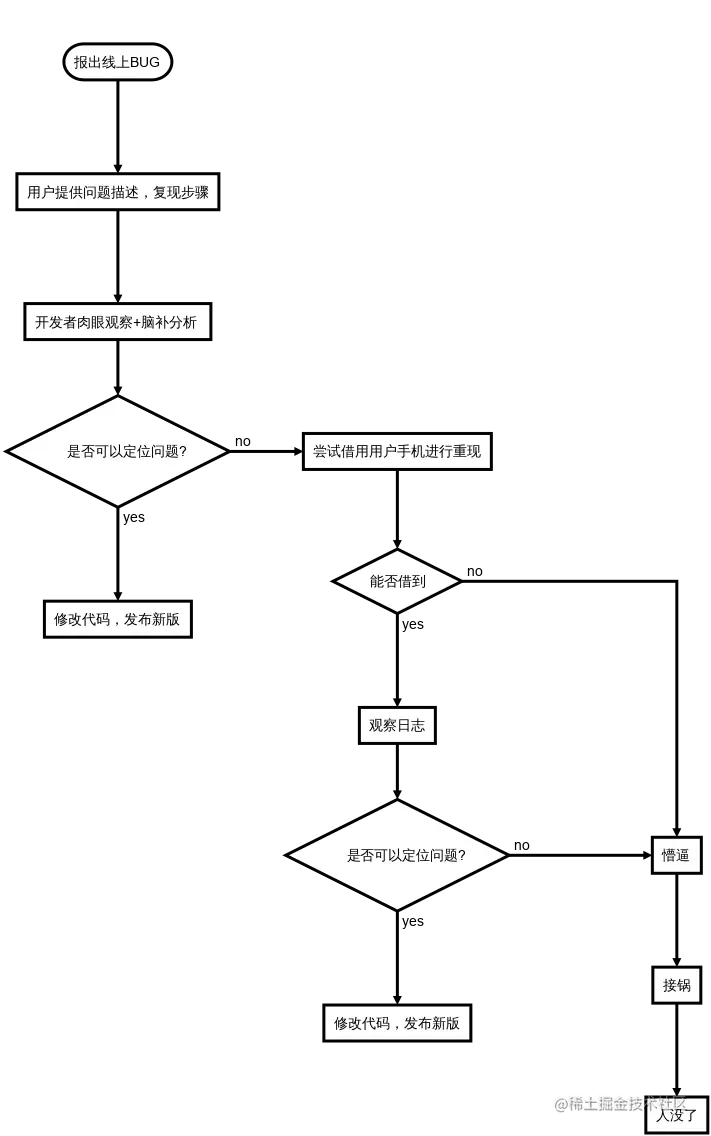
剖析问题根本，线上故障，分开两步走，第一，**线上**，App已经发版到用户手上了，相当于放出去的风筝，那么风筝线还在不在，就决定我们能否准确知道用户现场发生了什么。第二，**故障**，众所周知，故障是分等级的，有可能是体验上的不顺手，这是轻微的，也有可能是功能无法使用，这个可轻可重，具体要看该功能是否是核心模块，问题能否定性为特定机型的兼容问题，最严重的就是 应用崩溃，当然也分为有触发条件的崩溃 和 启动崩溃。总之各种奇奇怪怪的异常，在App发布到线上，如果我们无法获得第一手信息，就无法准确定位问题从而解决。

情境

讲个笑话，下面的图，是我们通常遇到用户报BUG之后的常规处理方式。



上图应该是我们App开发者的定位线上问题的常规方式，这种方式在一些特定场景之下能够解决问题，但是如果遇到用户不方便提供设备，甚至有些用户无法提供任何关于问题的准确信息时，会对我们的app体验优化造成很大的困扰。

比如说，给大领导开发了一个App，然后发现了他的手机出现了崩溃，出现了关键功能兼容问题，想拿领导的手机来现场调试？头都给你拧掉。

又比如，用户远在千里之外，只能远程联系你，告知问题的描述，你是否可以准确定位问题？

如果都做不到，那么我们确实需要一个有效的机制来捞取任意设备的日志了，用日志来作为排查问题的重要依据。

其实目前有一些手段能够帮助我们获知线上问题，比如一些统计平台，如友盟，bugly，谷歌的firebase，它们在一定程度上可以统计出 App的crash日志频率并提供崩溃日志。我们的app也可以通过接入这些平台的埋点，来获知App在某些场景下出现了哪些预料之内的问题。或者有些公司的某些团队会自行开发类似功能的统计平台，并接入到本公司的App中。

但是这种方式都存在两个关键问题，**第一**，不实时，统计平台都是有上报统计延时的，今天报了问题，可能明天才能看到。**第二**，不精确，统计平台上报出的问题，通常是按照发生频次去排序的，无法满足精确定位指定用户的问题。等我们解决问题，可能用户体验已经凉了。

如何在解决问题的用户体验上更进一步，在统计平台的宏观统计之下，我们自己做一套微观定位问题的有效方式。

任务目标

相当于上图中的被动处理方式，其实我们可以主动去抓取问题用户的设备日志。

理想的处理方式应该就是：

用户反馈bug

开发者登录后台

捞取该用户的设备日志

从日志中去排查问题

修复bug，合并代码到主分支

发布新版

问题解决

方案如果落地，我们还得思考两个关键问题：

全量

如何准确获取App上的全量日志，日志如果残缺，排查问题就会缺少依据

协同

如何准确定位需要捞取日志的对用用户设备，只抓取需要的用户日志

实施步骤

全量

日志在开发过程中不可或缺，它为我们排查问题提供依据，梳理业务流程，通常我们看到日志都是在编译器中，也就是 在线调试，通过日志追查问题解决问题。通常，我们开发中写日志包括但不限于以下几种情况：

流程的开始和结束时

触发关键程序分支时

异常捕获时

网络请求时

IO操作时

调用第三方库的输入输出时

线程切换时

应用崩溃时

这些打印在编译器控制台上的日志，我们必须同步写入到 手机磁盘内进行持久化，然后在合适的时机上传到后台。经过一些方案的试验，考虑了 完整性，侵入性 等因素之后，最终选择使用 logcat命令 的 方式去捞日志。

类似的方案：比如，设计一套特有的 日志写入工具类，让业务方去写日志的时候，必须使用特定的方法。这种侵入性太强，而且 仅仅只能捞取 特定范围之内的日志，所以排除。

logcat的命令去获取全量日志，示例代码为：

private class LogRunnable : Runnable {

override fun run() {

val sdf = SimpleDateFormat("yyyy-MM-dd")

var fileName = "flutter-${sdf.format(Date())}.log"

var reader: BufferedReader? = null

try {

val process = ProcessBuilder("logcat", /\*"-s", "flutter",\*/ "-v", "threadtime").start()

reader = BufferedReader(InputStreamReader(process.inputStream))

var line: String?

while (reader.readLine().also { line = it } != null) {

FileIOUtils.writeFileFromString(

Global.savePath + "/" + fileName,

"$line\n",

true

)

judgeUploadCounter()

}

} catch (ignored: IOException) {

} finally {

if (reader != null) {

try {

reader.close()

} catch (ignored: IOException) {

}

}

}

}

}

开启一个日志捞取的线程，线程中创建 ProcessBuilder 开启logcat日志输出，然后把每一行日志都写入到指定的文件。

协同

协同的具体的表现就是确保我在后台看到的日志，一定就是当前用户的设备日志。两者保持尽可能同步。

app通过某种机制触发日志的上传动作，然后我们在后台下载日志进行阅读分析。

关键两个问题：

如何准确判断要抓日志的用户设备

通常来说，区分用户，通常是依赖于app的登录信息，我们假设App都是单点登录的，也就是说，任何用户同一时间都只能在一台设备上留存有效的登录信息，不存在多点同时登录的情况。此种情况下，我们可以直接把用户名作为接口入参，去请求接口，后端维护一个debug用户的列表，允许多个用户开启debug模式。上述接口完成之后，后端只需要提供一个上传接口，以及 指定用户日志文件的下载。

但是分析一下全场景，如果某些app允许多点登录，那么一个仅仅用一个用户名就无法标记一台设备，所以，为了兼容更多场景，匹配接口时，我采用 给接口传json对象的方式，允许传入多个不同的参数，然后后台建立规则，与传入的字段进行匹配，匹配成功则返回true。

协同的过程，必须前端和后台合作解决。后台需要提供一个debug判定接口，入参 json格式，表示 用户设备上的 关键信息映射，返回值，bool即可，表示当前用户是否开启了debug模式，true开始，false关闭。在开启的情况下，捞日志的进程才会执行。

如何精确触发日志的捞取和上传动作

在能够精确判断用户设备是否处于debug模式之下（debug模式才要抓日志）之后，接下来要考虑的就是，当用户反馈问题之后，我们能够知道 这个用户是谁，登录信息是什么，这些我们都可以拿到，但是 如何去开启他手机上的日志捞取进程呢？

是不是可以通过线上联系的方式告知用户如何去操作？

不，排除，用户的沟通成本是很高的，一般情况下，用户和技术员不会有直接的沟通，说到成本高，尝试过现场给 小白用户解决问题的同学应该有感受。

那就在app上开启一个长连接吧，与后台相连，app运行时，由后台主动去开启。好办法。但是 长连接 协议的前后端开发成本也是有点高的。是否有现成的方式？

还很有。推送。常用的极光推送，个推推送等，在App在线状态下的推送成功率几近百分之百，可以认为是可靠的。

我们可以直接 让后台发送一条封装的消息，去告知app当前是开启debug模式捞取日志，还是，现在就把 app捞好的日志传到后台。

这样就免了我们做技术的直接去和用户沟通。用户只需要反馈问题即可，剩下的都交给技术流程。

但是，这里有一些比较特殊的日志，

崩溃日志

在app崩溃之后，我们可以通过CrashHandler，在主线程尚未崩溃之前，开一个service去把崩溃日志写入到本地。

特殊逻辑日志

比如 风控日志，开发过程中预想到了一些风险场景，根据触发的用户量来调整业务逻辑。

像上面这些日志，就无需后台发推送消息去抓了，直接写到磁盘，到 后台发上传命令时，再一并上传。

结果

按照上面的思路，主要的工作量分为前后端两块：

app插件SDK

1.接入推送，制定 消息解读协议，执行 打开debug模式，或者 上传日志的动作

2.支持全量无侵入式日志捞取

3.支持封装关键的用户信息 复制代码

后台

1. 支持用户参数的设定，精确 区分执行动作的 用户
2. 接入推送，协调 消息体协议，封装 两个动作 （打开debug模式，上传日志）

3. 提供日志文件的过滤，下载入口复制代码

1. 完成之后，接入了插件SDK的安卓应用，则只需要在 Application中设置用户的基本信息，用于区分用户设备即可。其他的，全都交给SDK，并且捞取日志的全程，app的正常使用不受到任何影响。

问题：

万一用户无法再复现，那就没法解决了，而且用户也不可能这么配合。

类似的用户，将他标记成重点用户开启日志开关进行观察，或者针对他的机型进行观察，帮助定位问题。