Android常见问题Debug

彭冲

# Crash

1. 当system进程crash的信息：
   1. 开头\*\*\* FATAL EXCEPTION IN SYSTEM PROCESS [线程名]；
   2. 接着输出发生crash时的调用栈信息；
2. 当app进程crash时的信息：
   1. 开头FATAL EXCEPTION: [线程名]；
   2. 紧接着 Process: [进程名], PID: [进程id]；
   3. 最后输出发生crash时的调用栈信息。

分析crash问题时需要导出/data/system/dropbox目录的文件，根据打印出的进程/线程名、进程/线程ID、调用栈去check 代码和logcat 抓取的log，定位具体原因。

dropbox文件名为dropboxTag@xxx.txt

* dropboxTag = processClass(process) + “\_” + eventType;
  + processClass：分为system\_server, system\_app, data\_app;
  + eventType：分为crash,anr,wtf等
* xxx代表的是时间戳;
* 后缀除了.txt，还可以是.txt.gz压缩格式。

例如system\_server\_crash@1234567.txt，代表的是system\_server进程出现crash，记录该文件时间戳为1234567。

# 二、Anr

ANR(Application Not responding)，是指应用程序未响应，Android系统对于一些事件需要在一定的时间范围内完成，如果超过预定时间能未能得到有效响应或者响应时间过长，都会造成ANR。一般地，这时往往会弹出一个提示框，告知用户当前xxx未响应，用户可选择继续等待或者Force Close。

那么哪些场景会造成ANR呢？

* Service Timeout:如前台服务在20s内未执行完成；
* BroadcastQueue Timeout：如前台广播在10s内未执行完成
* ContentProvider Timeout：内容提供者执行超时
* inputDispatching Timeout: 输入事件分发超时5s，包括按键分发事件的超时。

导致Anr的常见原因：

* I/O阻塞
* 网络阻塞；
* 主线程做了耗时操作;
* 多线程死锁

发生Anr时，我们在logcat抓取的log中搜索关键字anr in可以看到：

* 各个进程的CPU使用情况；
* CPU负载；
* IOWait；

很多时候，我们分析logcat抓取的log时会看到类似下面log的信息：

chatty : uid=1000 system\_server expire 144 lines这是因为logd的环形buffer太小，瞬间输出的log过多导致的。在测试前，建议进入开发者选项将log buffer改为16M

导出data/anr/目录下的trace文件，结合trace文件中的调用栈我们可以做进一步分析。

分析trace文件有时会发现block的地方不在java代码中，这时需要我们使用addr2line工具解析so文件，命令如下：

addr2line -C -f -e ./out/target/product/msm8916\_32/symbols/system/lib/ libaudiopolicyservice.so 00035021

该解析命令对应的log：pc 00035021 /system/lib/libaudiopolicyservice.so

执行命令时要修改symbols目录所在的路径及so库的名字以及对应的地址。

需要注意的是，经常有同时在分析anr问题时，看到调用栈的栈顶是binder.so的就下结论说是binder出了问题，这是不对的。

如：

native: #02 pc 0001e905 /system/lib/libbinder.so (\_ZN7android14IPCThreadState14talkWithDriverEb+132)

这时，应该结合代码，进一步分析binder的服务端为何没有及时通过onTransact返回结果。

分析Anr 问题时需要注意线程的状态是否是blocked，如果是，是被哪个线程block。

可以搜索关键字：blocked，held，wait for等

避免Anr：

* UI线程尽量只做跟UI相关的工作
* 耗时的工作()比如数据库操作，I/O，网络操作)，采用单独的工作线程（HandlerThread、AsyncTask的doInBackground等）处理
* 用Handler来处理UIthread和工作thread的交互

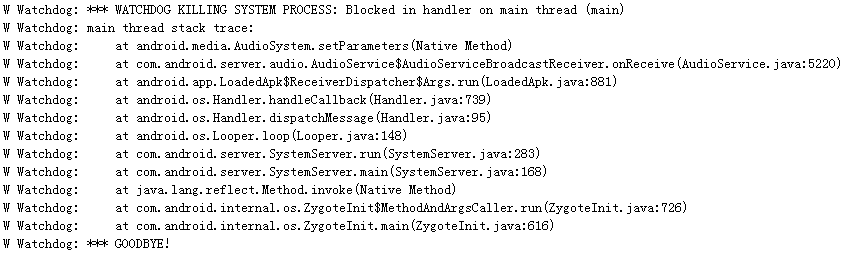
# 三、重启

Android重启大致分为framework重启和Kernel重启。

所谓framework重启，一般是SystemServer发生了重启。

导致SystemServer重启的直接原因一般是watchdog超时，我们可以在logcat抓取的log中搜索watchdog。

如：



分析狗超时要结合trace文件，我们可以从/data/system/dropbox和data/anr/中导出。结合关键字wtf和时间戳我们可以确定哪个文件是我们需要的。

接下来分析trace文件的步骤和分析anr时是相同的。

Watchdog会检测系统服务：

ActivityManagerService

PowerManagerService

WindowManagerService

InputManagerService

NetworkManagementService

MountService

NativeDaemonConnector

BinderThreadMonitor

MediaProjectionManagerService

MediaRouterService

MediaSessionService

导致重启的狗超时基本都发生在这些服务（随着Android版本变化，服务数量在改变），但这并不意味着都是Framework的代码有问题。虽然我们常去修改这些服务超时处的代码，加一些锁的保护，但很多时候，根本原因在于kernel，所以检查到具体哪个服务超时，并确认已经涉及到hal层代码后，可以请驱动的相关同事帮忙一起check，确认下是否和kernel有关。

有一种重启的原因是发生了严重的tombstone。

发生tombstone导致的重启时我们可以导出/data/tombstones目录下的文件去结合调用栈和signal 的值去check原因。

和分析Anr问题相同，我们需要使用addr2line来分析库文件的调用来确定出问题的代码。在tombstone文件中还可以通过解析寄存器信息来获取更多信息，但对于我们Framework debug来说，addr2line已经足够了。

# 四、进入Dump状态

我们在做一些自动化测试时，测试经常会提一些死机的bug，这些死机多数不是真正的死机，而是发生了严重错误，为了保存现场，进入了dump状态。

一般这个时候测试会提供名为Port\_COM5（或者其他数字）的dump信息。

我们需要解析dump去分析发生了什么。

1、配置解析环境：

每个项目的环境配置都会有一些差异，要咨询驱动架构师

2、解压Port\_COM5，在服务器下载对应版本的scm\_debug\_info中的vmlinux放入Port\_COM5目录

1. 服务器根目录下执行解析命令，不同项目的命令会有差异，要咨询驱动架构师，下面是PhoebeM的命令：

ramparser/linux-ramdump-parser-v2/ramparse.py --everything -g ~/Phoebe\_M/LINUX/android/prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-4.9/bin/arm-linux-androideabi-gdb -n ~/Phoebe\_M/LINUX/android/prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-4.9/bin/arm-linux-androideabi-nm --force-hardware=8916 -d --vmlinux ~/phoebem\_dump/Port\_COM5\_646/vmlinux --auto-dump ~/phoebem\_dump/Port\_COM5\_646/ -o ~/phoebem\_dump/Port\_COM5\_646/out

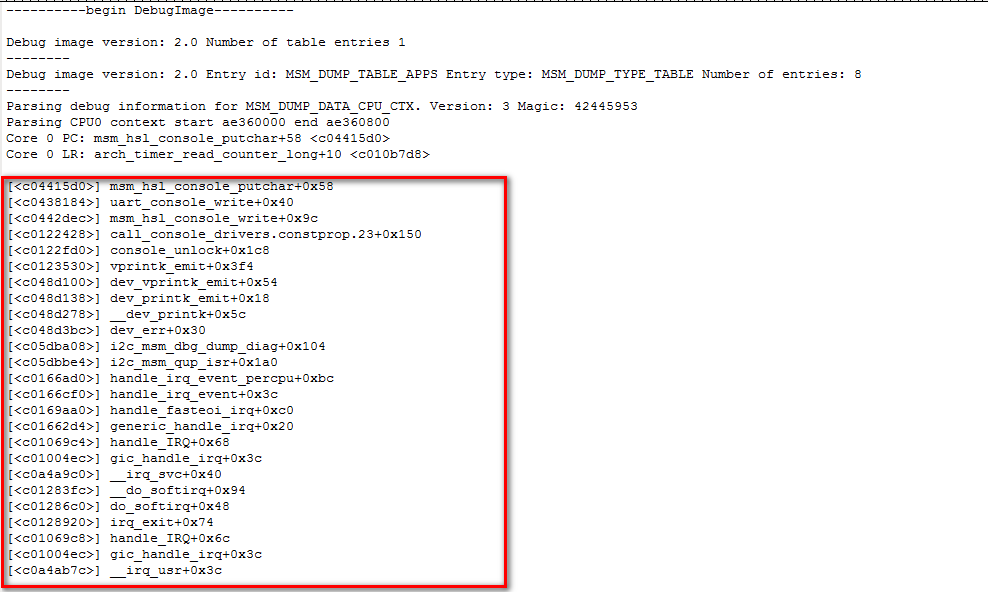
执行命令时要做对应修改，gcc工具要指向自己的项目代码，平台要配置，vmlinux要指向Port\_COM5目录的vmlinux。

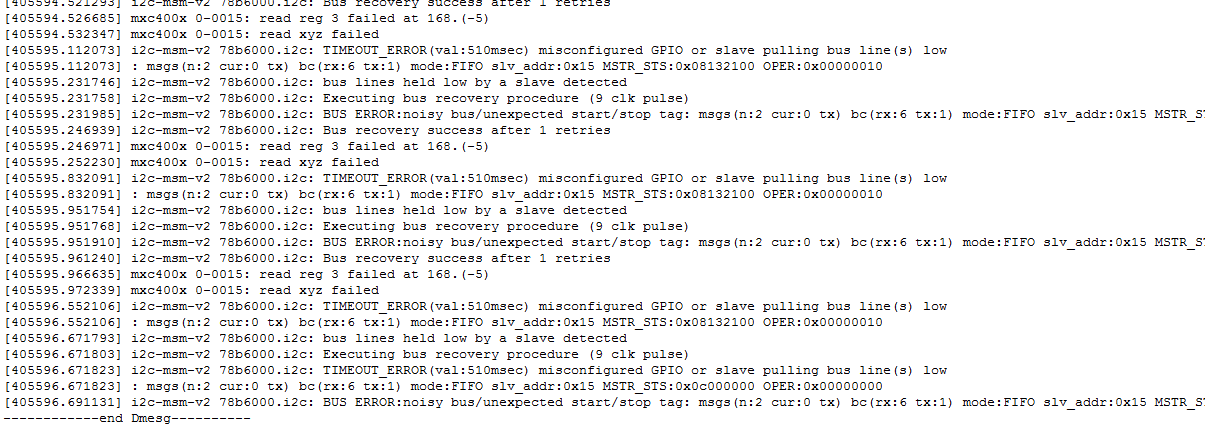
1. 分析解析后得到的log：

解析dump会得到很多文件，对我们定位Framework问题来说，dmesg\_TZ.txt就足够了。

该文件中会有进入dump状态时的堆栈，cpu状态、kernel log等信息。

我们一般关注堆栈和最后的kernel log：





上图是G sensor i2c error导致的机器进入dump状态。

进入dump的原因有很多，有看门狗超时、硬件过热、i2c挂掉等等。但基本都是kernel问题，所以我们只要能简单定位出问题是哪个模块的，转给对应的驱动owner就可以了。

# 五、冻屏、黑屏、卡顿

这是最难分析的问题，因为问题原因复杂多变，又没有靠谱的关键字。

本文仅述我在遇到这类问题时的调试思路和调试工具：

1、工具：bugreport、systrace、MAT：

Bugreport和systrace我在惠冰的《Android 常用调试方法》文档中已经做了简单补充，熟练使用的关键是大量练习，这里就不赘述。

MAT的使用和配置可以参考我之前的MAT文档。

2、debug思路

在遇到冻屏、黑屏、长时间卡顿问题时bugreport可以帮我们还原当时的手机状态，注意一定要处于异常状态时抓取的bugreport才有用。

冻屏和黑屏问题我们要结合logcat log重点关注bugreport中的Activity dump、Window dump和memory dump。

关注是否有死循环和内存泄漏，怀疑有死循环又没有可靠的log可以使用systrace进一步分析，如果怀疑内存泄漏则可以使用MAT做进一步分析。

我目前遇到的冻屏基本就是死循环和内存泄漏两类，相比于冻屏，黑屏问题较好处理一些。

借助Activity dump、Window dump，结合logcat基本就可以定位。

Window dump 和Activity dump 匹配上之后，确认是哪个view导致的黑屏，再结合代码具体分析。

我遇到的系统卡顿问题基本分两类：内存泄漏、cpu在后台做耗时任务。系统卡顿无非就是memory或cpu不够用了。先用bugreport看下是memory还是cpu不足，再结合mat和systrace去分析。分析的过程是艰苦的，而且分析出原因后也很难解决，好在这里问题并不多。在集成Dolby的项目要注意，这类问题会稍多一些，因为我们基本都是host集成，解码和音效处理由cpu完成。

# 六、Audio、Video问题Debug

回想从做audio以来遇到的audio、video问题，无法播放、没声音、不流畅、杂音、不同步、音效差、音量不对、时间戳不对，五花八门，从一开始的无从下手到后来的游刃有余，关键在于对代码的熟悉，对流程的理解。

想要很好的解决Audio模块的问题，要掌握以下技能：

1、熟悉MediaPlayer、AudioTrack、AudioEffect的Framework代码（包括java、jni、native，仅熟悉java是不够的）

2、熟悉MediaPlayerService、Nuplayer的Framework代码。

3、熟悉AudioFlinger、AudioPolicy、AudioMixer的Framework代码。

4、熟悉音视频播放和录制的流程，方便在出问题时知道应该check哪个节点。

5、熟练使用MediaInfo、QXDM、QCAT、Cool Edit等工具。

Media Framework的代码量太大，我之前写过的文档只能覆盖一小部分，而且随着Android版本不断更新有些内容已经过时，网上也没有找到很全面的文档，入门的话可以参考《深入理解Android 5.0系统》和《深入理解Android内核设计思想》中的音频部分。就我个人经验，最好的学习方法是结合具体情景读源码，从应用层读到hal层，看不懂的就打log，查文档，读的多了就掌握了。

在熟悉Media Framework代码之后，结合神器QXDM、QCAT，分析音视频问题不再是一个难题。分析音视频问题时如何使用QXDM和QCAT我在惠冰的《Android 常用调试方法》文档中已经做了补充，这里不再赘述。

分析音视频问题的思路：

1、首先确认是否和网络有关

2、使用MediaInfo解析资源文件，结合平台支持的编解码格式，确认是否和音源文件有关

3、根据问题类型，打开需要的native log开关（audio的log太多，仅打开需要的，不然机器会卡的难以操作），同时使用logcat和QXDM抓取log分析

4、使用QCAT解析QXDM log，根据平台文档，使用Cool Edit看波形，听声音，确认问题出在哪个环节

5、分析出问题环节的log，检查代码，不断的添加log进行debug

总结一下我遇到过的音视频问题的原因：

1、无法播放：

平台不支持Decoder或者Parser，Decoder可以在配置文件中确认，Parser需要结合代码和log进行确认

2、没声音：

A、平台不支持音频的Decoder或者Parser，也可能是Extractor出了问题，

B、Policy错误，打开Output失败。

C、AudioMixer mute了音量

此类问题使用QXDM非常好定位，只要查看每个节点dump出的数据就可以确认是哪个环节出了问题。

3、不流畅：

一般是发生了under run，check usecase，确认是cpu解码还是dsp解码

4、杂音：

杂音可能出现在从资源文件提取开始到硬件输出的任意一个环节，先结合QXDM log定位问题出在哪，再check相关代码。注意，AudioEffect没加好是极有可能导致杂音的。

5、不同步：

音频和视频不同步时多数是视频解码速度过慢导致，一般出现在hevc格式的视频上。人对声音不流畅的敏感强于对视频掉帧的敏感，所以Android的策略是会seek 视频与音频同步。当视频的delay超过某一阈值时这一策略就会生效。在没有超过这一阈值时，我们会感受到视频的delay会越来越严重。

6、音效差：

A、硬件问题，QXDM解析出的最后一个数据流没有问题的话就可以交给驱动和硬件去check

B、附加的AudioEffect有问题，我们的联想项目一般会集成杜比，需要修改Dolby的调音参数。

7、音量不对：

在音量设置流程的每个节点去打印音量参数，确认问题出在哪里。

遇到过以下三个节点有问题的bug：

A、该流类型的音量会被其他流音量设置时影响，完成需求时代码时序有问题

B、音量计算节点，音量曲线有问题（硬件和调音师会要求我们修改），一般交给驱动修改

C、AudioMixer 处理时音量时对不需要音效的流类型附加了AudioEffect

8、时间戳不对：

检查Parser和Extractor，有时是AudioTrack对象复用时没有flush导致。