|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| D:\02-资料分类\04-文档模板\扉页图1-new.jpg | | | | |
|  | AndroidR\_ANR原理和分析调试指南 | | |  |
| 文档版本 V1.0 |  | |
| 发布日期 2020-08-27 |  | |
|  | | | | |
|  | 紫光展锐科技有限公司 | |  |  |

|  |
| --- |
| 版权所有 © 紫光展锐科技有限公司。保留一切权利。  本文件所含数据和信息都属于紫光展锐所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负责任何与本文件相关的直接或间接的、任何伤害或损失。  请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。 |



|  |
| --- |
| 紫光展锐科技有限公司 |

前 言

概述

本文档介绍Android中ANR的基础和基于ANR实例分析问题等。

读者对象

本文档适用于需要了解androidr ANR问题的分析和处理的同事，及其他对此感兴趣的同事。

缩略语

|  |  |  |
| --- | --- | --- |
| 缩略语 | 英文全名 | 中文解释 |
| ANR | Application Not Response | 应用程序无相应 |
|  | androidr | Android 11版本 |
|  | androidq | Android 10版本 |

变更信息

| 文档版本 | 发布日期 | 作者 | 修改说明 |
| --- | --- | --- | --- |
| V1.0 | 2020-08-10 | Mingwen.Ge | 初稿 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

关键字

ANR、

目 录

[1 概述 6](#_Toc49524654)

[1.1 ANR发生的条件 6](#_Toc49524655)

[1.2 ANR的类型 7](#_Toc49524656)

[1.3 ANR的触发原因 7](#_Toc49524657)

[1.4 ANR原因归类 8](#_Toc49524658)

[2 ANR问题分析基础知识 8](#_Toc49524659)

[2.1 什么是死锁 9](#_Toc49524660)

[2.2 什么是Binder 10](#_Toc49524661)

[2.3 Lowmemorykill机制 13](#_Toc49524662)

[2.4 线程状态 13](#_Toc49524663)

[2.5 ANR相关的log 15](#_Toc49524664)

[3 ANR相关log的详细说明 16](#_Toc49524665)

[3.1 system.log 16](#_Toc49524666)

[3.2 main.log 21](#_Toc49524667)

[3.3 event.log 21](#_Toc49524668)

[3.4 kernel.log 23](#_Toc49524669)

[3.5 trace.log 25](#_Toc49524670)

[3.6 sysinfo.log 28](#_Toc49524671)

[3.7 sgm.csv 29](#_Toc49524672)

[3.8 ANR打印调试log常用命令 30](#_Toc49524673)

[4 ANR分析流转流程 31](#_Toc49524674)

[4.1 ANR问题分析的步骤 31](#_Toc49524675)

[4.2 ANR问题流转流程图 41](#_Toc49524676)

[4.3 ANR问题分析模板（应用部分） 41](#_Toc49524677)

[4.4 分析ANR问题的建议 43](#_Toc49524678)

[5 ANR问题分析实例 43](#_Toc49524679)

[5.1 activity多实例 43](#_Toc49524680)

[5.2 应用crash导致的ANR问题 44](#_Toc49524681)

[5.3 iowait占用高 45](#_Toc49524682)

[5.4 log过多打印对系统的影响 46](#_Toc49524683)

[5.5 Lowmemorykiller杀进程导致的ANR 47](#_Toc49524684)

[5.6 焦点窗口切换问题 48](#_Toc49524685)

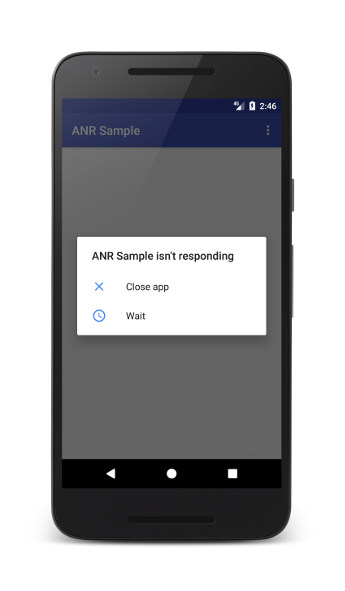
[5.7 GPU问题导致ANR 48](#_Toc49524686)

[5.8 应用导致的ANR问题 51](#_Toc49524687)

[6 小结 51](#_Toc49524688)

# 概述

在ANR(Application Not Responding)，即“应用程序无响应”。在 Android 中，ActivityManagerService （简称 AMS ）和 WindowManagerService （简称 WMS ）会监测应用程序的响应时间，如果应用程序主线程（即 UI 线程）在超时时间内对输入事件没有处理完毕，或者对特定操作没有执行完毕，就会出现 ANR。



## ANR发生的条件

一般地，ANR不是凭空出现的,ANR问题的产生一般那需要同时满足如下三个条件：

**主线程：**只有应用程序进程的主线程响应超时才会产生ANR。因为只有主线程也就是UI线程需要与用户进行交互，子线程的阻塞或者缓慢只要不影响主线程就不会引发ANR。

**超时时间：**不同类型ANR的超时时间不同，只有主线程在这个时间上限内没有响应才发生会ANR。

**输入事件/特定操作：**输入事件是指按键、触屏等设备输入事件，特定操作是指BroadcastReceiver和Service的生命周期中的各个函数，产生ANR的场景不同，报出ANR的原因也会不同。

## ANR的类型

ANR通常分为如下四个大类：

1 InputDispatchingTimedOut

应用程序主线程在5秒内没有完成用户的input事件。Android原生超时检测时间为5秒，平台版本为10秒。

2 Service Timeout

应用程序没有执行完成service的bind/create/start/destroy/unbind操作前台服务20秒超时，后台服务200秒超时，前景服务时间8秒超时。

3 Broadcast Timeout

应用程序在规定时间内没有执行完成onReceive操作。前台广播25秒超时，后台广播90秒超时。

4 Content Provider Timeout

应用程序在20秒内没有执行完成ContentProvider相关操作

## ANR的触发原因

1. InputEventTimeout——输入事件（包括按键和触屏事件）在5秒内无响应，就会弹出ANR提示框，供用户选择继续等待程序响应或者关闭这个应用程序（也就是杀掉这个应用程序的进程）。输入超时类的ANR可以细分为以下两类：

* 处理消息超时，顾名思义这一类是指因为消息处理超时而发生的ANR，在log，会看到“Input dispatching timed out (Waiting because the focused window has not finished processing the input events that were previously delivered to it.)”
* 无法获取焦点：这一类通常因为新窗口创建慢或旧窗口退出慢而造成窗口无法获得焦点从而发生ANR，典型Log“Reason: Waiting because no window has focus but there is a focused application that may eventually add a window when it finishes starting up.”

1. BroadcastTimeout——BroadcastReceiver在规定时间内（前台广播10秒，后台广播60秒）无法处理完成，即报出广播超时的消息，这一类型没有提示框弹出，多发于statusbar，settings应用中
2. ServiceTimeout——Service在20秒内无法处理完成，即报出服务超时，同样没有提示框弹出。这一类ANR有时会在Bluetooth和wifi中出现，但是很少碰到。
3. ContentProvider Not Responding——这一类ANR是在Android4.4上新出现的，是对ContentProvider操作超时造成的，多发生在android.process.media中。

## ANR原因归类

ANR按其发生的场景基于原因大致分为如下类别:

1 死锁

2 焦点问题

3 新窗口创建慢

4 应用异常导致ANR问题

5 内存问题

6 CPU占用高

7 GPU问题

8 多实例问题

9 iowait高

10 log过多打印对系统的影响

# ANR问题分析基础知识

通过第1部分的概述，对ANR应该有了初步的了解，ANR问题除了应用自身的问题，更多的是系统性的问题，知识储备越丰富分析问题才能越深入，此处列出几个知识点供参考。

## 什么是死锁

死锁：指两个或两个以上的进程（或线程）在执行过程中，因争夺资源而造成的一种相互等待的现象，若无外力作用，它们都将无法推进下去。此时成系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。死锁产生的4个必要条件（缺一不可）：

互斥条件：顾名思义，线程对资源的访问是排他性，当该线程释放资源后下一线程才可进程占用。

请求和保持：简单来说就是自己拿的不放手又等待新的资源到手。线程T1至少已经保持了一个资源R1占用，但又提出对另一个资源R2请求，而此时，资源R2被其他线程T2占用，于是该线程T1必须等待，但又对自己保持的资源R1不释放。

不可剥夺：在没有使用完资源时，其他线程不能进行剥夺。

循环等待：一直等待对方线程释放资源。

Trace log中对应关键字一般包含”held by”,”Blocked”，摘录部分对应trace log内容：

|  |
| --- |
| anr\_2019-06-06-14-02-29-419  "main" prio=5 tid=1 Blocked  | group="main" sCount=1 dsCount=0 flags=1 obj=0x76e5e790 self=0x70f4014c00  | sysTid=1070 nice=0 cgrp=default sched=0/0 handle=0x7179dbf548  | state=S schedstat=( 1264076458 1532113644 2741 ) utm=96 stm=30 core=2 HZ=100  | stack=0x7fda2cd000-0x7fda2cf000 stackSize=8MB  | held mutexes=  at com.android.settingslib.bluetooth.CachedBluetoothDeviceManager.getCachedDevicesCopy(CachedBluetoothDeviceManager.java:-1)  - waiting to lock <0x0aa22fa5> (a com.android.settingslib.bluetooth.CachedBluetoothDeviceManager) held by thread 24  at com.android.systemui.statusbar.policy.BluetoothControllerImpl.getDevices(BluetoothControllerImpl.java:201)  主线程在等待线程24释放锁  "SysUiBg" prio=5 tid=24 Blocked  | group="main" sCount=1 dsCount=0 flags=1 obj=0x13846a80 self=0x70ea279400  | sysTid=1246 nice=10 cgrp=default sched=0/0 handle=0x70d90884f0  | state=S schedstat=( 112699141 1272500191 644 ) utm=6 stm=4 core=2 HZ=100  | stack=0x70d8f85000-0x70d8f87000 stackSize=1041KB  | held mutexes=  at com.android.settingslib.bluetooth.CachedBluetoothDevice.dispatchAttributesChanged(CachedBluetoothDevice.java:772)  - waiting to lock <0x0030bf7a> (a java.util.ArrayList) held by thread 1  at com.android.settingslib.bluetooth.CachedBluetoothDevice.onActiveDeviceChanged(CachedBluetoothDevice.java:548)  at  具体可以参见bug 1092456 |

## 什么是Binder

Binder是Android的一个类，它实现了IBinder接口。从IPC角度来说，Binder是Android中一种跨进程通信方式，还可以理解为是一种物理设备，它的设备驱动是 /dev/binder ;从Android Framework角度来说，Binder是ServiceManager连接各种Manager（ActivityManager、WindowManager，等等）和相应ManagerService的桥梁；从Android应用层来说，Binder是客户端和服务端进行通信的媒介，当bindService的时候，服务端会返回一个包含了服务端业务调用的Binder对象，通过这个Binder对象，客户端就可以获得服务器提供的服务或数据，这里的服务包括普通服务和基于 AIDL 的服务。 Android 开发中，Binder主要用在Service中，包括AIDL和Message，Message的底层其实就是AIDL。

Binder异常导致ANR问题：

Binder阻塞

|  |
| --- |
| Cmd line: com.android.wallpaper  "main" prio=5 tid=1 Native  | group="main" sCount=1 dsCount=0 flags=1 obj=0x766eb058 self=0x7b8b29ec00  | sysTid=6085 nice=-10 cgrp=default sched=0/0 handle=0x7b8c7feed0  | state=S schedstat=( 986933481 116782153 1065 ) utm=66 stm=32 core=6 HZ=100  | stack=0x7fc29e1000-0x7fc29e3000 stackSize=8192KB  | held mutexes=  kernel: (couldn't read /proc/self/task/6085/stack)  native: #00 pc 00000000000cc884 /apex/com.android.runtime/lib64/bionic/libc.so (\_\_ioctl+4)  native: #01 pc 000000000008a38c /apex/com.android.runtime/lib64/bionic/libc.so (ioctl+132)  native: #02 pc 000000000005829c /system/lib64/libbinder.so (android::IPCThreadState::talkWithDriver(bool)+256)  native: #03 pc 0000000000059150 /system/lib64/libbinder.so (android::IPCThreadState::waitForResponse(android::Parcel\*, int\*)+60)  native: #04 pc 0000000000058eec /system/lib64/libbinder.so (android::IPCThreadState::transact(int, unsigned int, android::Parcel const&, android::Parcel\*, unsigned int)+180)  native: #05 pc 000000000004df38 /system/lib64/libbinder.so (android::BpBinder::transact(unsigned int, android::Parcel const&, android::Parcel\*, unsigned int)+72)  native: #06 pc 000000000014957c /system/lib64/libandroid\_runtime.so (android\_os\_BinderProxy\_transact(\_JNIEnv\*, \_jobject\*, int, \_jobject\*, \_jobject\*, int)+152)  at android.os.BinderProxy.transactNative(Native method)  at android.os.BinderProxy.transact(BinderProxy.java:514)  at android.content.ContentProviderProxy.openTypedAssetFile(ContentProviderNative.java:721)  at android.content.ContentResolver.openTypedAssetFileDescriptor(ContentResolver.java:1687)  对应bug 1163895 |

Binder满

|  |
| --- |
| "FmRadioServiceThread" prio=5 tid=4 Native  | group="main" sCount=1 dsCount=0 flags=1 obj=0x12dc01c0 self=0x720a2a6400  | sysTid=11721 nice=0 cgrp=default sched=0/0 handle=0x720af99d50  | state=S schedstat=( 4503116 11034768 11 ) utm=0 stm=0 core=7 HZ=100  | stack=0x720ae97000-0x720ae99000 stackSize=1039KB  | held mutexes=  kernel: (couldn't read /proc/self/task/11721/stack)  native: #00 pc 00000000000cc884 /apex/com.android.runtime/lib64/bionic/libc.so (\_\_ioctl+4)  native: #01 pc 000000000008a38c /apex/com.android.runtime/lib64/bionic/libc.so (ioctl+132)  native: #02 pc 000000000005829c /system/lib64/libbinder.so (android::IPCThreadState::talkWithDriver(bool)+256)  native: #03 pc 0000000000059150 /system/lib64/libbinder.so (android::IPCThreadState::waitForResponse(android::Parcel\*, int\*)+60)  native: #04 pc 0000000000058eec /system/lib64/libbinder.so (android::IPCThreadState::transact(int, unsigned int, android::Parcel const&, android::Parcel\*, unsigned int)+180)  native: #05 pc 000000000004df38 /system/lib64/libbinder.so (android::BpBinder::transact(unsigned int, android::Parcel const&, android::Parcel\*, unsigned int)+72)  native: #06 pc 000000000014957c /system/lib64/libandroid\_runtime.so (android\_os\_BinderProxy\_transact(\_JNIEnv\*, \_jobject\*, int, \_jobject\*, \_jobject\*, int)+152)  at android.os.BinderProxy.transactNative(Native method)  at android.os.BinderProxy.transact(BinderProxy.java:514)  at android.hardware.radio.IRadioService$Stub$Proxy.openTuner(IRadioService.java:197)  at android.hardware.radio.RadioManager.openTuner(RadioManager.java:1763)  at com.android.fmradio.FmRadioManager.openTuner(FmRadioManager.java:64)  at com.android.fmradio.FmRadioManager.openDev(FmRadioManager.java:75)  main log  07-18 08:37:04.494 959 978 W IPCThreadState: Waiting for thread to be free. mExecutingThreadsCount=31 mMaxThreads=31  Line 363164: M093FD5 07-18 08:37:04.532 959 978 W IPCThreadState: Waiting for thread to be free. mExecutingThreadsCount=32 mMaxThreads=31  Line 363165: M093FD6 07-18 08:37:04.534 959 978 W IPCThreadState: Waiting for thread to be free. mExecutingThreadsCount=31 mMaxThreads=31  Line 363184: M093FF8 07-18 08:37:06.198 959 978 W IPCThreadState: Waiting for thread to be free. mExecutingThreadsCount=32 mMaxThreads=31  Line 363185: M093FF9 07-18 08:37:06.200 959 978 W IPCThreadState: Waiting for thread to be free. mExecutingThreadsCount=31 mMaxThreads=31  Line 363186: M093FFA 07-18 08:37:06.201 959 978 W IPCThreadState: Waiting for thread to be free. mExecutingThreadsCount=32 mMaxThreads=31  Line 363187: M093FFB 07-18 08:37:06.203 959 978 W IPCThreadState: Waiting for thread to be free. mExecutingThreadsCount=31 mMaxThreads=31  见bug 1128088 |

## Lowmemorykill机制

Linux系统当可用内存较低的时候oom killer机制会根据一定的规则去杀掉一些进程来释放内存，而Android系统的LowMemoryKiller机制就是以此功能为基础做了一些调整。Android系统中的APP在使用完成之后并不会马上被杀掉，而是驻留在内存中，当下一次在此进入此应用的时候可以省去进程创建的过程，加快启动速度。LowMemoryKiller机制会在内存资源紧张的时候，杀掉一些进程来回收内存。

LowMemoryKiller是基于进程的adj来杀掉运行的进程的，不要在ANR前看到有lowmemorykiller的log就说内存紧张了，要看杀到了adj多少了。对应lowmemorykiller的log:

|  |
| --- |
| M3A07DF 07-15 13:19:47.103 3524 3524 I lowmemorykiller: Kill 'android.process.acore' (10385), uid 10008, oom\_adj 0 to free 91944kB |

## 线程状态

从ANR的trace调用栈，我们能看到线程的状态变化，线程的状态主要有如下几种：

| **java thread 状态** | **cpp thread状态** | **说明** |
| --- | --- | --- |
| TERMINATED | ZOMBIE | 线程死亡，终止运行 |
| RUNNABLE | RUNNING/RUNNABLE | 线程可运行或正在运行 |
| TIMED\_WAITING | TIMED\_WAIT | 执行了带有超时参数的wait、sleep或join函数 |
| BLOCKED | MONITOR | 线程阻塞，等待获取对象锁 |
| WAITING | WAIT | 执行了无超时参数的wait函数 |
| NEW | INITIALIZING | 新建，正在初始化，为其分配资源 |
| NEW | STARTING | 新建，正在启动 |
| RUNNABLE | NATIVE | 正在执行JNI本地函数 |
| WAITING | VMWAIT | 正在等待VM资源 |
| RUNNABLE | SUSPENDED | 线程暂停，通常是由于GC或debug被暂停 |
|  | UNKNOWN | 未知状态 |

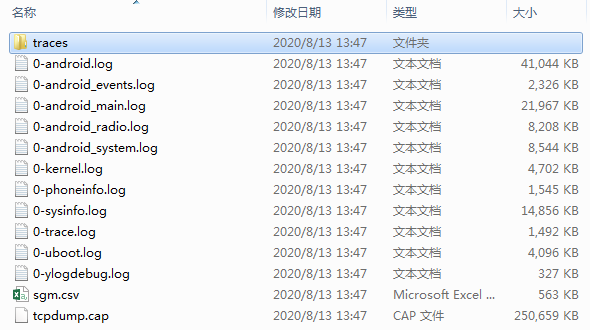
从trace log摘一段调用栈做下说明：

|  |
| --- |
| "main" prio=5 tid=1 Native  | group="main" sCount=1 dsCount=0 flags=1 obj=0x70bf0c28 self=0xa6de6e00  | sysTid=23565 nice=-10 cgrp=default sched=0/0 handle=0xa7315470  | state=S schedstat=( 44673175678 16346565170 49791 ) utm=3162 stm=1304 core=4 HZ=100  | stack=0xbe23a000-0xbe23c000 stackSize=8192KB  | held mutexes=  native: #00 pc 0005ddac /apex/com.android.runtime/lib/bionic/libc.so (syscall+28)  native: #01 pc 0013126d /apex/com.android.art/lib/libart.so (art::ConditionVariable::WaitHoldingLocks(art::Thread\*)+88)  native: #02 pc 004d9443 /apex/com.android.art/lib/libart.so (art::GoToRunnable(art::Thread\*)+342)  native: #03 pc 004d92c5 /apex/com.android.art/lib/libart.so (art::JniMethodEnd(unsigned int, art::Thread\*)+8)  at android.graphics.HardwareRenderer.nLoadSystemProperties(Native method)  at android.graphics.HardwareRenderer.loadSystemProperties(HardwareRenderer.java:767)  at android.view.ThreadedRenderer.loadSystemProperties(ThreadedRenderer.java:547) |

main" prio=5 tid=1 Native这行log分别说明了线程名称:main，优先级:5，线程锁id:1和线程状态:Native。基于线程的状态判断当前的运行情况。

## ANR相关的log

ylog下ap log解析后大致包含如下文件：



在一份标准的ylog中，通常有以下文件可用于分析ANR的原因：

**system.log** 包含ANR发生时间点信息、ANR发生前的CPU信息，还包含大量系统服务输出的信息。

**main.log** 包含ANR发生前应用自身输出的信息，可供分析应用是否有异常；此外还包含dalvik输出的GC信息，可供分析内存回收的速度，判断系统是否处于低内存或内存碎片化状态。

**event.log** 包含AMS与WMS输出的应用程序声明周期信息，可供分析窗口创建速度以及焦点转换情况。

**sgm.csv** 每秒打印一次cpu的负载情况，通过该log可以看当前时间段的cpu和iowait情况。

**kernel.log** 包含kernel打出的信息，LowMemoryKiller杀进程、内存碎片化或内存不足，mmc驱动异常都可以在这里找到。

**snapshot.log或trace.log** 在有些项目中这两个文件可能被合并为一个，包含ANR发生时的应用堆栈信息、PS信息和meminfo信息。根据定制需要，还可以包含ANR发生时的system\_server堆栈、SurfaceFlinger堆栈、文件句柄状况等信息。

**tomestone** 有些应用的ANR是由于之前应用已经崩溃导致的，需要注意以下在ANR发生前如果在tomestone文件夹中此应用已经发生了Native Crash，那么ANR很有可能就是由此导致的。

**Dropbox** 该文件会把snapshot中的信息备份一份，如果因为某些原因导致snapshot文件丢失，可以尝试在dropbox中寻找ANR发生时的堆栈信息。

**sysinfo.log** 该文件中包含meminfo, vmstat, buddyinfo, slabinfo等信息。

# ANR相关log的详细说明

## system.log

system.log 包含ANR发生时间点信息、ANR发生前的CPU信息，还包含大量系统服务输出的信息。分析ANR问题的第一要务是找到ANR发生的第一时间点。在system.log中搜ANR in是我们比较熟悉的，但这不是ANR的第一时间点，不同的ANR对应的ANR时间点不尽相同。基于几类ANR的特点列出对应关键字和参考log。

**1广播超时类ANR**

通过ANR in搜到对应ANR

|  |
| --- |
| 06-28 17:29:56.535 1063 19737 E ActivityManager: ANR in com.android.music  06-28 17:29:56.535 1063 19737 E ActivityManager: PID: 4486  06-28 17:29:56.535 1063 19737 E ActivityManager: Reason: Broadcast of Intent { act=android.intent.action.SCREEN\_OFF flg=0x50200010 }  06-28 17:29:56.535 1063 19737 E ActivityManager: Load: 7.08 / 6.66 / 4.46  06-28 17:29:56.535 1063 19737 E ActivityManager: ----- Output from /proc/pressure/memory -----  06-28 17:29:56.535 1063 19737 E ActivityManager: some avg10=0.00 avg60=0.00 avg300=0.00 total=887414  06-28 17:29:56.535 1063 19737 E ActivityManager: full avg10=0.00 avg60=0.00 avg300=0.00 total=613345  06-28 17:29:56.535 1063 19737 E ActivityManager: ----- End output from /proc/pressure/memory -----  06-28 17:29:56.535 1063 19737 E ActivityManager:  06-28 17:29:56.535 1063 19737 E ActivityManager: CPU usage from 292896ms to 0ms ago (2020-06-28 17:24:59.740 to 2020-06-28 17:29:52.636):  06-28 17:29:56.535 1063 19737 E ActivityManager: 90% 616/media.extractor: 40% user + 49% kernel / faults: 2641137 minor |

通过ActivityManager: Reason:的打印可以知道原因是Broadcast of Intent { act=android.intent.action.SCREEN\_OFF flg=0x50200010 }，即android.intent.action.SCREEN\_OFF广播处理超时，但ANR in的打印一般在ANR的第一时间之后，找广播超时的第一时间，对应log:

|  |
| --- |
| 06-28 17:29:52.634 1063 1098 W BroadcastQueue: Timeout of broadcast BroadcastRecord{a4973ec u-1 android.intent.action.SCREEN\_OFF} - receiver=android.os.BinderProxy@e520f9a, started 25001ms ago  06-28 17:29:52.634 1063 1098 W BroadcastQueue: Receiver during timeout of BroadcastRecord{a4973ec u-1 android.intent.action.SCREEN\_OFF} : BroadcastFilter{45b20a8 10126/u0 ReceiverList{4cdc0cb 4486 com.android.music/10126/u0 remote:e520f9a}}  对应bug 1353482 |

再根据超时时间是前台广播25s还是后台广播90秒，基于ANR时间点向前推25s或者90s看系统和应用的状况。

**2 input或者不能获得焦点窗口类ANR**

不能获得焦点窗口类

|  |
| --- |
| S4196C9 08-11 09:11:09.408 871 7906 E ActivityManager: ANR in com.android.soundrecorder (com.android.soundrecorder/com.sprd.soundrecorder.RecorderActivity)  S4196C9 08-11 09:11:09.408 871 7906 E ActivityManager: PID: 7740  S4196C9 08-11 09:11:09.408 871 7906 E ActivityManager: Reason: Input dispatching timed out (ActivityRecord{1858f33 u0 com.android.soundrecorder/com.sprd.soundrecorder.RecorderActivity t1722} does not have a focused window)  S4196C9 08-11 09:11:09.408 871 7906 E ActivityManager: Parent: com.android.soundrecorder/com.sprd.soundrecorder.RecorderActivity  S4196C9 08-11 09:11:09.408 871 7906 E ActivityManager: Load: 4.19 / 5.38 / 5.81  S4196C9 08-11 09:11:09.408 871 7906 E ActivityManager: ----- Output from /proc/pressure/memory -----  S4196C9 08-11 09:11:09.408 871 7906 E ActivityManager: some avg10=0.00 avg60=0.00 avg300=0.06 total=27281738  S4196C9 08-11 09:11:09.408 871 7906 E ActivityManager: full avg10=0.00 avg60=0.00 avg300=0.00 total=11596615  S4196C9 08-11 09:11:09.408 871 7906 E ActivityManager: ----- End output from /proc/pressure/memory -----  S4196C9 08-11 09:11:09.408 871 7906 E ActivityManager:  S4196C9 08-11 09:11:09.408 871 7906 E ActivityManager: CPU usage from 4934ms to -3611ms ago (2020-08-11 09:10:58.875 to 2020-08-11 09:11:07.419):  S4196C9 08-11 09:11:09.408 871 7906 E ActivityManager: 40% 871/system\_server: 26% user + 13% kernel / faults: 22168 minor 17 major  S4196C9 08-11 09:11:09.408 871 7906 E ActivityManager: 20% 7061/com.android.camera2: 16% user + 4.4% kernel / faults: 11774 minor |

通过ActivityManager: Reason:的打印可以知道原因是Input dispatching timed out不能获得焦点窗口，找到ANR的第一时间点

|  |
| --- |
| S4187E5 08-11 09:11:03.636 871 1130 I WindowManager: Input event dispatching timed out sending to application ActivityRecord{1858f33 u0 com.android.soundrecorder/com.sprd.soundrecorder.RecorderActivity. Reason: ActivityRecord{1858f33 u0 com.android.soundrecorder/com.sprd.soundrecorder.RecorderActivity t1722} does not have a focused window  对应bug 1389688 |

根据该类ANR超时时间向前推10s,看该段时间的系统运行情况，结合具体应用分析问题。

**3 service timeout类超时ANR**

|  |
| --- |
| 005E13 12-26 15:01:00.575 824 838 E ActivityManager: ANR in system  12-26 15:01:00.575 824 838 E ActivityManager: PID: 824  12-26 15:01:00.575 824 838 E ActivityManager: Reason: executing service com.android.server.telecom/.components.TelecomService  12-26 15:01:00.575 824 838 E ActivityManager: Load: 16.04 / 9.62 / 4.1  12-26 15:01:00.575 824 838 E ActivityManager: CPU usage from 285038ms to 0ms ago (2019-12-26 14:56:12.701 to 2019-12-26 15:00:57.739):  12-26 15:01:00.575 824 838 E ActivityManager: 5.6% 353/surfaceflinger: 3% user + 2.6% kernel / faults: 58 minor  12-26 15:01:00.575 824 838 E ActivityManager: 5.2% 199/mmcqd/0: 0% user + 5.2% kernel  12-26 15:01:00.575 824 838 E ActivityManager: 4.3% 418/bootanimation: 1.5% user + 2.7% kernel / faults: 16602 minor |

通过ActivityManager: Reason:的打印可以知道原因是TelecomService服务超时，对应ANR的第一时间点

|  |
| --- |
| 0051EC 12-26 15:00:57.737 824 838 W ActivityManager: Timeout executing service: ServiceRecord{6443b3e u0 com.android.server.telecom/.components.TelecomService}  对应bug 1234454 |

**4 Content Provider超时类ANR**

|  |
| --- |
| S070E3F 10-17 14:18:49.448 4169 4201 E ActivityManager: ANR in android.process.media  S070E3F 10-17 14:18:49.448 4169 4201 E ActivityManager: PID: 9075  S070E3F 10-17 14:18:49.448 4169 4201 E ActivityManager: Reason: ContentProvider not responding  S070E3F 10-17 14:18:49.448 4169 4201 E ActivityManager: Load: 0.0 / 0.0 / 0.0  S070E3F 10-17 14:18:49.448 4169 4201 E ActivityManager: CPU usage from 38966ms to 0ms ago (2019-10-17 14:18:10.040 to 2019-10-17 14:18:49.006):  S070E3F 10-17 14:18:49.448 4169 4201 E ActivityManager: 15% 3592/surfaceflinger: 6.4% user + 8.6% kernel / faults: 351 minor 2 major |

通过ActivityManager: Reason:打印可以知道是ContentProvider无响应，对应ANR的第一时间点

|  |
| --- |
| E070A30 10-17 14:18:49.051 4169 4201 I am\_anr : [0,9075,android.process.media,952647237,ContentProvider not responding]  对应bug 1183666 |

ANR in打印log说明：

1. **ANR进程名称与PID：**进程名和PID是找到发生ANR的应用的主要依据，但是有两种例外情况。如果PID为0，说明应用在发生ANR之前就已经被LowMemoryKiller杀死或者已经崩溃。这种情况下应用程序无法处理广播或按键消息，因此出现ANR。
2. **ANR类别：**可以据此判断ANR超时时间，决定需要回溯多久查找ANR的原因。比如用户输入时间处理超时回溯10秒，广播超时回溯25秒还是90秒等情况。
3. **CPU平均负载：**表示一、五、十分钟内有多少进程在等待CPU调度。单核手机一般不超过10，四核手机一般不超过14。如果CPU负载太高应用程序主线程长时间得不到CPU时间片就会发生ANR甚至Watchdog重启。
4. **CPU统计时间段：**ago和Later分别表示 ANR 发生前后一段时间内的 CPU 使用率,并不是某一时刻的值。需要注意的是，这个统计本身也会收到CPU负载高的影响，可能无法统计到ANR发生之前的CPU状况。
5. **进程CPU占用率：**单核手机上，如果进程CPU占用率较高，且截取到的是ANR发生之前的CPU情况，那么ANR可能与CPU占用率有关。
6. **页错误数量：**分为次要页错误和主要页错误，分别表示内存与缓存的命中情况。页错误过高说明内存频繁换页，会导致分配内存与GC速度显著变慢。
7. **新增进程：**ANR发生前出现大量的新增进程说明可能广播风暴或密集的延时重启。
8. **总CPU占用率：**在单核设备上可以保证准确，在支持热插拔的设备上一般不准确。
9. **线程CPU占用率：**可配合snapshot中的应用调用堆栈分析单个进程CPU占用率高问题。

CPUinfo信息打印，搜dumpstate关键字

|  |
| --- |
| S6A07F1 08-12 21:07:42.418 22777 22777 D dumpstate: Cpu Core Count: 8  S6A07F2 08-12 21:07:42.418 22777 22777 D dumpstate: Timer Circle: 10000ms.  S6A07F3 08-12 21:07:42.418 22777 22777 D dumpstate: From time 21740320ms(2020-08-12 13:07:27.060722145 UTC) to 21750320ms(2020-08-12 13:07:37.060723522 UTC).  S6A07F4 08-12 21:07:42.418 22777 22777 D dumpstate:  S6A07F5 08-12 21:07:42.418 22777 22777 D dumpstate: \* CPU USAGE: | \* OTHER COUNTS:  S6A07F6 08-12 21:07:42.418 22777 22777 D dumpstate: -2- IDLE USER SYSTEM NICE IOWAIT IRQ SOFTIRQ STEAL TOTAL | CTXT\_SWITCH FG\_FAULT FG\_MAJ\_FAULT  S6A07F7 08-12 21:07:42.418 22777 22777 D dumpstate: cpu0(1): 4.80% 17.20% 43.90% 3.20% 17.50% 8.20% 5.30% 0.00% 100.00% | 27293 70653 8946 |

如上对应的前三行Log，第一行是CPU的核数是几核，第二行是计时周期是10秒，第三行是统计运行情况的起始时间，这个时间是UTC时间格式，要在此基础上加8个小时，能对应打印log的实际时间戳。后面就是每个状态下的CPU占用情况，看下idle剩余,kernel和user占用情况，iowait占用情况等。这些信息之后是各个进程的占用情况的打印，可以看下有无明显占用较高的情况。

## main.log

**main.log** 包含ANR发生前应用自身输出的信息，可供分析应用是否有异常；此外还包含dalvik输出的GC信息，可供分析内存回收的速度，判断系统是否处于低内存或内存碎片化状态。

针对ANR问题一般围绕应用主线程及上下文log打印对log进行分析。

main.log中主要是应用输出的信息，相对system.log而言会比较散乱。从中挖掘ANR的关键信息需要一定的耐心。在分析时应当关注以下几个方面。

1、**反复打出相同log：**如果应用CPU占用率很高，且反复打出相同log，很可能是出现死循环。

05-20 14:26:32.875 2896 2896 V BluetoothSettings: onPreferenceChange()shouldScanDisabled = false

05-20 14:26:32.895 2896 2896 V BluetoothSettings: onPreferenceChange()shouldScanDisabled = false

05-20 14:26:32.915 2896 2896 V BluetoothSettings: onPreferenceChange()shouldScanDisabled = false

05-20 14:26:32.925 2896 2896 V BluetoothSettings: onPreferenceChange()shouldScanDisabled = false

05-20 14:26:32.945 2896 2896 V BluetoothSettings: onPreferenceChange()shouldScanDisabled = false

05-20 14:26:32.965 2896 2896 V BluetoothSettings: onPreferenceChange()shouldScanDisabled = false

05-20 14:26:32.975 2896 2896 V BluetoothSettings: onPreferenceChange()shouldScanDisabled = false

05-20 14:26:33.005 2896 2896 V BluetoothSettings: onPreferenceChange()shouldScanDisabled = false

05-20 14:26:33.025 2896 2896 V BluetoothSettings: onPreferenceChange()shouldScanDisabled = false:

**2、应用输出的log是否存在异常：**如下例，Camera尝试分配26214400 Byte显存，但是从16:42:16开始申请直到16:42:27才分配完毕，这段时间主线程一直被阻塞，因此发生ANR。在应用程序容易出现性能问题的关键点适度添加log，对查找ANR问题非常有帮助。

01-01 16:42:16.942 150 1676 I SprdCameraHardware: allocateCaptureMem, mJpegHeapSize = 26214400, mRawHeapSize = 26214400

01-01 16:42:16.942 150 1676 I SprdCameraHardware: allocateCaptureMem:mRawHeap align size = 26214400 . count 1

......

01-01 16:42:27.142 150 1676 I SprdCameraHardware: allocCameraMem: mm\_iova: phys\_addr 0x20000000, data: 0xaf50d000, size: 0x1900000, phys\_size: 0x1900000.

01-01 16:42:27.142 150 1676 I SprdCameraHardware: allocateCaptureMem: initJpeg

**3、是否有多个应用都打出相同的异常信息：**有时一些ANR问题是由共同的底层问题导致的。。

## event.log

Event.log中主要是ActivityManager和WindowManager打出的信息。在分析由性能问题导致的ANR时，应用程序可能并没有死锁或者阻塞，而是在OnCreate中浪费4秒，留给OnResume执行的时间已经不够了。分析这类问题时就不能简单地看应用程序主线程堆栈停在哪里，而是要分析窗口生命周期各个方法的执行时间，找到运行迟缓的部分。Event.log中需要关注的信息主要有：

|  |  |
| --- | --- |
| **Log关键字** | **含义** |
| am\_proc\_start | 开始创建应用进程 |
| am\_proc\_bound | 应用进程创建完毕 |
| am\_restart\_activity | realActivityStart，创建进程完成后首次启动应用 |
| am\_resume\_activity | 窗口Resume开始 |
| am\_on\_resume\_called | 窗口Resume完毕 |
| am\_pause\_activity | 窗口Pause开始 |
| am\_on\_paused\_called | 窗口Pause完毕 |
| am\_failed\_to\_pause | 窗口Pause超时 |
| am\_finish\_activity | 应用Finish开始 |
| am\_proc\_died | 进程死亡（比如被LowMemoryKiller杀死） |

在掌握了以上窗口生命周期log后，就可以从event.log中分析究竟是哪一个过程慢导致ANR发生，见下例。

**1、创建进程慢：**正常情况下启动应用创建进程所需的时间应当是300～500ms，在系统内存碎片化分配不出连续内存段或者CPU变频不正常时进程创建速度就会明显变慢，下面log中am\_proc\_start到am\_proc\_bound一共花费了4.5秒才创建出一个进程。这就可以排除应用问题，转而查找底层原因。

|  |
| --- |
| 01-23 10:08:28.590 615 631 I am\_proc\_start: [0,31528,10040,com.sprd.note,activity, com.sprd.note]  01-23 10:08:33.230 615 1174 I am\_proc\_bound: [0,31528,com.sprd.note] |

**2、窗口创建缓慢：**log中可见，进程启动速度正常，但是用了6秒才完成Resume走到am\_on\_resume\_called。如果单一应用反复出现此现象，就需要在应用的声明周期方法中分段添加log查找执行缓慢的代码；如果多个应用随机出现此现象，就需要对系统整体性能进行分析，查找阻塞点。

|  |
| --- |
| 07-12 12:39:53.830 592 608 I am\_proc\_start: [0,12735,10008,com.android.gallery3d,activity,com.android.gallery3d/.app.GalleryActivity]  07-12 12:39:54.340 592 1322 I am\_proc\_bound: [0,12735,com.android.gallery3d]  07-12 12:39:55.060 592 1322 I am\_restart\_activity: [0,1110839416,99,com.android.gallery3d/.app.GalleryActivity]  07-12 12:40:01.470 12735 12735 I am\_on\_resume\_called: [0,com.android.gallery3d.app.GalleryActivity]  07-12 12:40:02.070 592 608 I am\_anr : [0,12735,com.android.gallery3d,14203973,Input dispatching timed out (Waiting because no window has focus but there is a focused application that may eventually add a window when it finishes starting up.)] |

**3、窗口Pause慢：**log中可见，SlideshowEditActivity在05:14:57就已经收到AMS的am\_pause\_activity命令，但直到05:15:08才完成am\_on\_paused\_called走完Pause流程。对于单一应用而言，前一个窗口Pause不下去新窗口就没法Resume出来，焦点长时间处于null状态就会触发窗口焦点转换ANR。这个例子需要应用检查窗口Pause慢的原因。

|  |
| --- |
| 06-09 05:14:57.913 628 25582 I am\_pause\_activity: [0,1115385600,com.android.mms/.ui.SlideshowEditActivity]  06-09 05:14:58.413 628 644 I am\_restart\_activity: [0,1116031944,213,com.android.mms/.ui.SlideEditorActivity]  06-09 05:15:08.933 2946 2946 I am\_on\_paused\_called: [0,com.android.mms.ui.SlideshowEditActivity]  06-09 05:15:08.933 628 25580 I am\_failed\_to\_pause: [0,1115385600,com.android.mms/.ui.SlideshowEditActivity,(none)]  06-09 05:15:09.193 628 644 I am\_anr : [0,2946,com.android.mms,1082703461,Input dispatching timed out (Waiting because no window has focus but there is a focused application that may eventually add a window when it finishes starting up.)] |

## kernel.log

Kernel.log主要包含HAL层和Kernel层的信息，对于平时习惯于分析system.log和mian.log的程序员而言，其中的信息会比较难于理解。其实在分析ANR过程中，主要关注有无内存占用高，binder fail等情况。

针对ANR问题分析一般关注lmk查杀频繁，或者杀到adj 200到 0，主要查看的Log有out of memory, ememinfo的打印等。

emem相关打印

|  |
| --- |
| ++++++++++++++++++++++E\_SHOW\_MEM\_BEGIN++++++++++++++++++++  7ED34 <6> [45932.234430][08-22 16:06:30.234] emem: The killed process adj = 0  7ED35 <6> [45932.234435][08-22 16:06:30.234] e\_show\_mem: Enhanced Mem-Info:  7ED36 <6> [45932.234440][08-22 16:06:30.234] e\_show\_mem: E\_SHOW\_MEM\_ALL  7ED37 <6> [45932.234444][08-22 16:06:30.234] e\_show\_mem: Enhanced Mem-info :SHOW MEM  7ED38 <4> [45932.234449][08-22 16:06:30.234] Mem-Info:  7ED39 <4> [45932.234464][08-22 16:06:30.234] active\_anon:16852 inactive\_anon:15603 |

page allocation failure: order的log及调用dump

|  |
| --- |
| 222EC <4> [55857.482977][08-11 17:41:54.482] CpuTracker: page allocation failure: order:0, mode:0x1094020(GFP\_ATOMIC|\_\_GFP\_COMP|\_\_GFP\_NOMEMALLOC), nodemask=(null)  222ED <6> [55857.483001][08-11 17:41:54.483] CpuTracker cpuset=foreground mems\_allowed=0  222EE <4> [55857.483017][08-11 17:41:54.483] CPU: 4 PID: 1101 Comm: CpuTracker Tainted: G O 4.14.188+-ab33247 #1  222EF <4> [55857.483020][08-11 17:41:54.483] Hardware name: Spreadtrum SC9863A SoC (DT)  222F0 <4> [55857.483024][08-11 17:41:54.483] Call trace:  222F1 <4> [55857.483038][08-11 17:41:54.483] dump\_backtrace+0x0/0x1a0  222F2 <4> [55857.483043][08-11 17:41:54.483] show\_stack+0x20/0x2c  222F3 <4> [55857.483049][08-11 17:41:54.483] dump\_stack+0xa4/0xf0  222F4 <4> [55857.483056][08-11 17:41:54.483] warn\_alloc+0x118/0x1ac  222F5 <4> [55857.483061][08-11 17:41:54.483] \_\_alloc\_pages\_nodemask+0xf98/0xfb8  222F6 <4> [55857.483067][08-11 17:41:54.483] new\_slab+0x138/0x680  222F7 <4> [55857.483072][08-11 17:41:54.483] \_\_\_slab\_alloc+0x440/0x598  222F8 <4> [55857.483075][08-11 17:41:54.483] kmem\_cache\_alloc+0x280/0x2fc |

当然不是所有这些打印都有问题，要结合具体问题。

## trace.log

系统生成的Trace文件保存在data/anr,可以用命令adb pull data/anr/取出。traces.txt只保留最后一次ANR的信息,Android系统有个DropBox功能功能，它能记录系统出现的andr,crash等错误.因此，在android log下找不到trace log时，要看下dropbox中是否保留有发生过的ANR的信息.(log路径:/data/system/dropbox)。

Trace log中保存ANR前运行的线程的调用栈信息，利用这些信息查看主线程的运行情况，是否有blocked耗时等情况。对应的每个调用栈信息大致说明如下，以

1. "Signal Catcher" daemon prio=5 tid=3 Runnable
2. | group="system" sCount=0 dsCount=0 obj=0x32c050d0 self=0x7f97dd1400
3. | sysTid=9729 nice=0 cgrp=default sched=0/0 handle=0x7fa200e450
4. | state=R schedstat=( 217991249 1074429 82 ) utm=15 stm=6 core=4 HZ=100
5. | stack=0x7fa1f14000-0x7fa1f16000 stackSize=1005KB
6. | held mutexes= "mutator lock"(shared held)

第1行："Signal Catcher"：线程名称，daemon：是否是daemon线程（如果不是，则不打印“daemon”），prio=5：java线程Thread对象中的优先级，tid=3：vm中对应的 threadid，Runnable：线程在虚拟机中的状态；（如果当前线程没有attach，则第一行显示： “name” prio=num (not attached)）；

第2行：group: ThreadGroup，sCount: Suspend count， dsCount: debugger suspend count（小于等于sCount），obj：对应java线程 java.lang.Thread对象，self：native 对应的 thread 指针；

第3行：sysTid：对应linux线程 tid， nice：线程调度执行优先级，cgrp: cgroup,cpu调度group，sched：调度策略和调度优先级，handle：当前线程对应的pthread\_t

nice:

线程调度优先级（getpriority获取），-20 ~ 20 之间，越小，优先级越高， -1代表获取优先级失败；

cgrp:

cat /proc/self/task/%d/cgroup,

5:freezer:/

4:cpuset:/background

3:cpu:/bg\_non\_interactive

2:memory:/

1:cpuacct:/uid\_10024/pid\_6850

cgrp=bg\_non\_interactive

sched调度策略：

#define SCHED\_NORMAL 0

#define SCHED\_OTHER 0

#define SCHED\_FIFO 1

#define SCHED\_RR 2

SCHED\_OTHER

　　它是默认的线程分时调度策略，所有的线程的优先级别都是0，线程的调度是通过分时来完成的。简单地说，如果系统使用这种调度策略，程序将无法设置线程的优先级。请注意，这种调度策略也是抢占式的，当高优先级的线程准备运行的时候，当前线程将被抢占并进入等待队列。这种调度策略仅仅决定线程在可运行线程队列中的具有相同优先级的线程的运行次序。

　　SCHED\_FIFO

　　它是一种实时的先进先出调用策略，且只能在超级用户下运行。这种调用策略仅仅被使用于优先级大于0的线程。它意味着，使用SCHED\_FIFO的可运行线程将一直抢占使用SCHED\_OTHER的运行线程J。此外SCHED\_FIFO是一个非分时的简单调度策略，当一个线程变成可运行状态，它将被追加到对应优先级队列的尾部((POSIX 1003.1)。当所有高优先级的线程终止或者阻塞时，它将被运行。对于相同优先级别的线程，按照简单的先进先运行的规则运行。我们考虑一种很坏的情况，如果有若干相同优先级的线程等待执行，然而最早执行的线程无终止或者阻塞动作，那么其他线程是无法执行的，除非当前线程调用如pthread\_yield之类的函数，所以在使用SCHED\_FIFO的时候要小心处理相同级别线程的动作。

　　SCHED\_RR

　　鉴于SCHED\_FIFO调度策略的一些缺点，SCHED\_RR对SCHED\_FIFO做出了一些增强功能。从实质上看，它还是SCHED\_FIFO调用策略。它使用最大运行时间来限制当前进程的运行，当运行时间大于等于最大运行时间的时候，当前线程将被切换并放置于相同优先级队列的最后。这样做的好处是其他具有相同级别的线程能在“自私“线程下执行。返回值 0表示设置成功 其他表示设置不成功

第4行：state：linux线程的状态，schedstat：线程调度情况，utm=15：线程在用户态运行的时间， stm=6：线程在内核态运行的时间， core=4：线程最后运行在哪个cpu上， HZ=100：系统时钟频率

state=R 任务的状态，R:running, S:sleeping (TASK\_INTERRUPTIBLE), D:disk sleep (TASK\_UNINTERRUPTIBLE), T: stopped, T:tracing stop,Z:zombie, X:dead

schedstat：cat /proc/self/task/%d/schedstat

schedstat=( 217991249 1074429 82 ) 表示：（累计运行的物理时间（ns） 累计在就绪队列里的等待时间 主动切换和被动切换的累计次数）

state，utm, stm等从 /proc/self/task/%d/stat 中获取

\* struct task\_cputime - collected CPU time counts

\* @utime: time spent in user mode, in &cputime\_t units

\* @stime: time spent in kernel mode, in &cputime\_t units

\* @sum\_exec\_runtime: total time spent on the CPU, in nanoseconds

utm,stm 单位是jiffies，时钟中断次数;

频率是周期的倒数，一般是一秒钟中断产生的次数，所以 1/100 = 0.01s = 10ms, 每10ms产生一次中断；

第5行：stack=0x7fa1f14000-0x7fa1f16000 stackSize=1005KB

线程栈的start 和 end，以及 stack size；

第6行：held mutexes= "mutator lock"(shared held)

线程持有的当前虚拟机中的mutex的名称，及持有方式：shared held: 共享锁，exclusive held：独占锁；

每个线程在完成suspend时，都会把 “mutator lock”释放；

实际上，Suspend所有线程时，判断是否suspend完成，就是通过获取"mutator lock"独占锁来判断的，

如果能获取独占锁，说明其他线程都不再 独占/共享 持有 "mutator lock" ，说明所有线程suspend已经完成。

| **关键词** | **解释** |
| --- | --- |
| prio | 优先级 |
| tid | 线程id |
| group | 组名 |
| sCount | 挂起次数 |
| dsCount | 调试下 挂起计数 |
| obj | 同辈线程对象 |
| self | 当前线程对象 |
| sysTid | 线程id |
| nice | 进程优先级 |
| sched | 调度者优先级 |
| cgrp | 调度者组名 |
| handle | 线程缓存句柄 |
| state | native线程状态 |
| schedstat | 调度者状态 |
| utm | 用户态CPU时间 |
| stm | 内核态CPU时间 |
| core | 运行所在核 |

## sysinfo.log

sysinfo.log中主要是系统的相关信息，如meminfo,vmstat, buddyinfo, slabinfo等信息，当然对这些信息的了解越多，分析问题的维度和思路将更广阔，ANR主要看当前系统的内存情况及io影响等。

|  |
| --- |
| cat /proc/meminfo on 08-14 17:25:58  MemTotal: 1873896 kB  MemFree: 36264 kB  MemAvailable: 550116 kB  Buffers: 384 kB  Cached: 623584 kB  SwapCached: 20216 kB  Active: 492452 kB  Inactive: 510152 kB  Active(anon): 217836 kB  Inactive(anon): 221860 kB  Active(file): 274616 kB  Inactive(file): 288292 kB  Unevictable: 54404 kB  Mlocked: 54404 kB  SwapTotal: 1026180 kB  SwapFree: 255420 kB  Dirty: 708 kB  Writeback: 0 kB  AnonPages: 431348 kB  Mapped: 352336 kB  Shmem: 11440 kB  Slab: 157028 kB  SReclaimable: 47932 kB  SUnreclaim: 109096 kB  KernelStack: 34608 kB  PageTables: 51624 kB  NFS\_Unstable: 0 kB  Bounce: 0 kB  WritebackTmp: 0 kB  CommitLimit: 1963128 kB  Committed\_AS: 53378824 kB  VmallocTotal: 263061440 kB  VmallocUsed: 0 kB  VmallocChunk: 0 kB  run finished on 08-14 17:25:58 |

在sysinfo log中搜索meminfo关键字，会搜索到下面这样的信息，指108-14 17:25:58这个时间点cat /proc/memifo 得到的系统内存情况。 对我们而言最重要的是MemFree 和Cached这两个值。当这两个值相加小于20M时，表明系统在当前时间点内存紧张。

由于meminfo反映的是一个时间点的内存信息，有的时候在这个时间点之前，lowMemeoryKiller等机制已经开始杀掉一些进程，释放了一部分内存，使这个时间点的内存情况看起来还不是很紧张，但是发生ANR的时间段有可能因为低内存使系统操作缓慢。这个时候，我们也可以通过kernel.log或者main.log中的low memory killer打印出来的log(androidr中在main.log中查看)，判断系统在ANR的这段时间内内存的情况。

|  |
| --- |
| vmstat on 08-14 17:25:58  procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----  r b swpd free buff cache si so bi bo in cs us sy id wa  6 4 777904 36772 384 627968 428 601 907 161 0 965 10 8 81 0  run finished on 08-14 17:25:58 |

基于meminfo中Dirty:值和vmstat中si,so,bi,bo是否有明显占用较多，再sgm.log等iowait占用判断io占用情况。

## sgm.csv

打印每秒的总的CPU负载，iowait和每个核的CPU负载,iowait和频率等信息，该log每秒打印一次，可以直观看下当前系统的负载情况。

|  |
| --- |
| 8-14 17:24:24.176< 7131.405328>,10,1,0,0,0,0,30,2,768000,0,0,0,22,3,768000,0,0,0,13,2,768000,0,0,0,10,1,768000,0,0,0,4,2,768000,0,0,0,1,1,768000,0,0,0,0,0,768000,0,0,0,0,0,768000,0,0,0,0,0,50652,1,0,1873896,677900,559704,0,0,0,0,0,0  8-14 17:24:25.178< 7132.407397>,3,0,0,0,0,0,10,0,768000,0,0,0,8,0,768000,0,0,0,4,0,768000,0,0,0,3,0,768000,0,0,0,0,0,768000,0,0,0,0,0,768000,0,0,0,0,0,768000,0,0,0,0,0,768000,0,0,0,2055,2166,50654,1,0,1873896,677112,559036,0,0,0,0,0,0  8-14 17:24:26.180< 7133.409292>,8,0,0,0,0,0,24,1,768000,0,0,0,19,1,768000,0,0,0,16,2,768000,0,0,0,7,1,768000,0,0,0,2,0,768000,0,0,0,0,0,768000,0,0,0,0,0,768000,0,0,0,1,0,768000,0,0,0,0,0,50663,2,0,1873896,672852,560028,0,0,0,0,0,0  8-14 17:24:27.182< 7134.411657>,4,0,0,0,0,0,10,0,768000,0,0,0,10,0,768000,0,0,0,4,0,768000,0,0,0,2,0,768000,0,0,0,3,0,768000,0,0,0,0,0,768000,0,0,0,0,0,768000,0,0,0,0,0,768000,0,0,0,2203,2334,50664,1,0,1873896,675452,559448,0,0,0,0,0,0  8-14 17:24:28.185< 7135.414029>,7,0,0,0,0,0,18,0,768000,0,0,0,14,0,768000,0,0,0,9,1,768000,0,0,0,5,0,768000,0,0,0,3,0,768000,0,0,0,0,0,768000,0,0,0,0,0,768000,0,0,0,0,0,768000,0,0,0,0,0,50666,1,0,1873896,674528,559100,0,0,0,0,0,0 |

标蓝颜色的项为总的CPU负载和iowait，标红的项为每个核的CPU负载，iowait和当前运行频率，当然具体看是几核手机。

## ANR打印调试log常用命令

分析ANR问题主要集中在分析main.log,system.log,event.log,trace.log几个文件的log,其余主要看下系统负载情况，内存情况等。

main.log,system.log,event.log,radio.log都是logcat命令抓取的，抓取Log类型是否带时间戳等主要跟的参数不同。仅列出如下几个命令供参考。

**1 打印到终端屏幕上的命令**

adb wait-for-device logcat

adb logcat -v time //默认是-b main -b system

adb logcat -v time -b main //main log

adb logcat -v time -b radio //radio log

adb logcat -v time -b system //system log

adb logcat -v time -b events //events log

打印到文件中的命令主要是多加一个文件名

log打印到指定文件的命令

adb wait-for-device logcat > logcat.txt

adb logcat -v time > logcat.txt //默认是-b main -b system

adb logcat -v time -b main > main.log //main log

adb logcat -v time -b radio > radio.log //radio log

adb logcat -v time -b system > system.log //system log

adb logcat -v time -b events > events.log //events log

**2 导出ANR trace命令**

发生ANR后，生成的trace文件在/data/anr目录下，使用如下命令导出：

adb pull /data/anr/ d:\

3 如果需要查询每一个进程所占用的cpu，dumpsys这个命令可以查，可以打印load值等。

adb shell dumpsys cpuinfo

也可以用如下adb命令打印CPU USAGE:情况信息。

adb shell cat /sys/kernel/debug/sprd\_debug/cpu/cpu\_usage

以上命令有些是在userdebug版本才能使用，如CPU USAGE的命令。

为了方便后续操作一般建议root,先执行时adb root,adb remount命令。

# ANR分析流转流程

## ANR问题分析的步骤

ANR问题产生的原因不尽相同，分析问题时要尽可能的全面的分析，不要仅仅看到一个点就妄下结论，犯以偏概全，归因谬误的错误。分析的过程要把能分析的点尽可能的覆盖到。大致按照如下几个步骤分析：

**1 找ANR时间点**

1. system log通过搜ANR in找到要分析应用的ANR

|  |
| --- |
| S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager: ANR in com.android.gallery3d (com.android.gallery3d/.v2.app.GalleryActivity2)  S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager: PID: 21098  S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager: Reason: Input dispatching timed out (9f0d5bc com.android.gallery3d/com.android.gallery3d.v2.app.GalleryActivity2 (server) is not responding. Waited 10007ms for KeyEvent(deviceId=-1, source=0x00000101, displayId=-1, action=DOWN, flags=0x00000008, keyCode=75, scanCode=0, metaState=0x00000000, repeatCount=0), policyFlags=0x6b000000)  S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager: Parent: com.android.gallery3d/.v2.app.GalleryActivity2  S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager: Load: 12.31 / 11.41 / 11.97  S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager: ----- Output from /proc/pressure/memory -----  S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager: some avg10=0.22 avg60=0.58 avg300=0.49 total=74433880  S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager: full avg10=0.10 avg60=0.27 avg300=0.21 total=36745380  S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager: ----- End output from /proc/pressure/memory -----  S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager:  S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager: CPU usage from 0ms to 7042ms later (2020-08-24 10:48:31.320 to 2020-08-24 10:48:38.362):  S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager: 96% 1041/system\_server: 59% user + 37% kernel / faults: 30723 minor 101 major  S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager: 39% 546/surfaceflinger: 18% user + 20% kernel / faults: 1895 minor 2 major  S1285CB 08-24 10:48:38.401 1041 23039 E ActivityManager: 38% 21098/com.android.gallery3d: 21% user + 16% kernel / faults: 19994 minor 88 major |

通过这些信息能知道ANR的应用，大致负载情况，占用较高的进程有哪些，缺页情况等信息。再向前搜WindowManager: Input event dispatching timed out关键字，搜到后记录下来。

|  |
| --- |
| S1276B7 08-24 10:48:31.132 1041 1333 I WindowManager: Input event dispatching timed out sending to com.android.gallery3d/com.android.gallery3d.v2.app.GalleryActivity2. Reason: 9f0d5bc com.android.gallery3d/com.android.gallery3d.v2.app.GalleryActivity2 (server) is not responding. Waited 10007ms for KeyEvent(deviceId=-1, source=0x00000101, displayId=-1, action=DOWN, flags=0x00000008, keyCode=75, scanCode=0, metaState=0x00000000, repeatCount=0), policyFlags=0x6b000000 |

这个比较接近ANR的第一时间检点，针对input事件超时类ANR可以看下该Log后WindowManager: Dump Win No.的数字最大到多少，如果数字大于50一般有多实例问题。这是针对input事件窗口超时找ANR时间点的例子。其他情况类似可以看下第3部分3.1system.log的说明章节。

1. event.log通过搜am\_anr关键字找到要分析应用的ANR

|  |
| --- |
| E1277A5 08-24 10:48:31.417 1041 23039 I am\_anr : [0,21098,com.android.gallery3d,685292613,Input dispatching timed out (9f0d5bc com.android.gallery3d/com.android.gallery3d.v2.app.GalleryActivity2 (server) is not responding. Waited 10007ms for KeyEvent(deviceId=-1, source=0x00000101, displayId=-1, action=DOWN, flags=0x00000008, keyCode=75, scanCode=0, metaState=0x00000000, repeatCount=0), policyFlags=0x6b000000)] |

这个时间点一般与system.log中WindowManager: Input event dispatching timed out关键字对应的log时间点比较接近，一般都可以作为参考的第一时间点。再基于ANR类型向前查看event.log情况。

一般通过system.log和event.log能找到ANR时间点，基于ANR超时时间向前查看10秒，25秒，90秒等时间段的Log做进一步的分析。

**2 通过trace log查看调用栈是否有死锁，阻塞等情况。**

检查主线程是否存在前述的死锁、等待、阻塞或者死循环；是否将耗时的工作（比如大图片加载，过多控件的加载，网络连接等）放在了主线程；meminfo中free + cached是否小于20M；ps中应用是否创建了数百子线程，主线程是否处于D状态；

dumpstate cpu log中是否高iowait，有没有单一线程占用了过多CPU，是否有过多主要页错误，多核设备是否只有一个CPU在工作。

先区分下是否正常的调用栈，避免看到主线程的此类调用栈就说blocked在该调用栈上。正常的调用栈一般都是下面的打印。

|  |
| --- |
| "main" prio=5 tid=1 Native  | group="main" sCount=1 dsCount=0 flags=1 obj=0x71a5d980 self=0xb400007babcc8010  | sysTid=1432 nice=-10 cgrp=default sched=0/0 handle=0x7cd28134f8  | state=S schedstat=( 61959068036 13930852424 121728 ) utm=4821 stm=1374 core=7 HZ=100  | stack=0x7fc7b12000-0x7fc7b14000 stackSize=8192KB  | held mutexes=  native: #00 pc 000000000009b568 /apex/com.android.runtime/lib64/bionic/libc.so (\_\_epoll\_pwait+8)  native: #01 pc 00000000000199fc /system/lib64/libutils.so (android::Looper::pollInner(int)+184)  native: #02 pc 00000000000198dc /system/lib64/libutils.so (android::Looper::pollOnce(int, int\*, int\*, void\*\*)+112)  native: #03 pc 0000000000110da0 /system/lib64/libandroid\_runtime.so (android::android\_os\_MessageQueue\_nativePollOnce(\_JNIEnv\*, \_jobject\*, long, int)+44)  at android.os.MessageQueue.nativePollOnce(Native method)  at android.os.MessageQueue.next(MessageQueue.java:335)  at android.os.Looper.loop(Looper.java:183)  at android.app.ActivityThread.main(ActivityThread.java:7928)  at java.lang.reflect.Method.invoke(Native method)  at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:603)  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:947)  对应bug 1191064 |

这个调用栈是正常运行的调用栈，这时候就要从应用运行过程，系统负载等情况再进一步做分析。

异常调用栈一般是有blocked的情况，如：

对应bug 1366119的调用栈：

|  |
| --- |
| "main" prio=5 tid=1 Native  | group="main" sCount=1 dsCount=0 flags=1 obj=0x71fd3c28 self=0xa9273e00  | sysTid=1335 nice=0 cgrp=default sched=0/0 handle=0xa96a2470  | state=S schedstat=( 5231159830 4214114808 7294 ) utm=348 stm=174 core=1 HZ=100  | stack=0xbe00c000-0xbe00e000 stackSize=8192KB  | held mutexes=  native: #00 pc 0009ada0 /apex/com.android.runtime/lib/bionic/libc.so (\_\_ioctl+8)  native: #01 pc 00068e13 /apex/com.android.runtime/lib/bionic/libc.so (ioctl+26)  native: #02 pc 000399b3 /system/lib/libbinder.so (android::IPCThreadState::talkWithDriver(bool)+238)  native: #03 pc 0003a5f5 /system/lib/libbinder.so (android::IPCThreadState::waitForResponse(android::Parcel\*, int\*)+32)  native: #04 pc 0003a3c3 /system/lib/libbinder.so (android::IPCThreadState::transact(int, unsigned int, android::Parcel const&, android::Parcel\*, unsigned int)+126)  native: #05 pc 000350c7 /system/lib/libbinder.so (android::BpBinder::transact(unsigned int, android::Parcel const&, android::Parcel\*, unsigned int)+98)  native: #06 pc 000c6f5f /system/lib/libandroid\_runtime.so (android\_os\_BinderProxy\_transact(\_JNIEnv\*, \_jobject\*, int, \_jobject\*, \_jobject\*, int)+82)  at android.os.BinderProxy.transactNative(Native method)  at android.os.BinderProxy.transact(BinderProxy.java:545)  at android.net.wifi.IWifiManager$Stub$Proxy.getCurrentNetwork(IWifiManager.java:3129)  at android.net.wifi.WifiManager.getCurrentNetwork(WifiManager.java:4989)  at com.android.settingslib.wifi.WifiStatusTracker.updateStatusLabel(WifiStatusTracker.java:218)  at com.android.settingslib.wifi.WifiStatusTracker.access$000(WifiStatusTracker.java:43)  at com.android.settingslib.wifi.WifiStatusTracker$3.onCapabilitiesChanged(WifiStatusTracker.java:81)  at android.net.ConnectivityManager$NetworkCallback.onAvailable(ConnectivityManager.java:3327)  at android.net.ConnectivityManager$CallbackHandler.handleMessage(ConnectivityManager.java:3607)  at android.os.Handler.dispatchMessage(Handler.java:106)  at android.os.Looper.loop(Looper.java:223)  at android.app.ActivityThread.main(ActivityThread.java:7923)  at java.lang.reflect.Method.invoke(Native method)  at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:603)  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:947) |

对应bug 1397196的调用栈：

|  |
| --- |
| "main" prio=5 tid=1 Native  | group="main" sCount=1 dsCount=0 flags=1 obj=0x71c1eeb0 self=0xb40000724809d7b0  | sysTid=21098 nice=-10 cgrp=default sched=0/0 handle=0x736eba94f8  | state=S schedstat=( 1790721611 324935683 3596 ) utm=122 stm=56 core=0 HZ=100  | stack=0x7fc5644000-0x7fc5646000 stackSize=8192KB  | held mutexes=  native: #00 pc 000000000004b20c /apex/com.android.runtime/lib64/bionic/libc.so (syscall+28)  native: #01 pc 00000000001af840 /apex/com.android.art/lib64/libart.so (art::ConditionVariable::WaitHoldingLocks(art::Thread\*)+136)  native: #02 pc 000000000065bacc /apex/com.android.art/lib64/libart.so (art::GoToRunnable(art::Thread\*)+464)  native: #03 pc 000000000065b8b8 /apex/com.android.art/lib64/libart.so (art::JniMethodEnd(unsigned int, art::Thread\*)+28)  at android.graphics.HardwareRenderer.nSyncAndDrawFrame(Native method)  at android.graphics.HardwareRenderer.syncAndDrawFrame(HardwareRenderer.java:433)  at android.view.ThreadedRenderer.draw(ThreadedRenderer.java:658)  at android.view.ViewRootImpl.draw(ViewRootImpl.java:4163)  at android.view.ViewRootImpl.performDraw(ViewRootImpl.java:3890)  at android.view.ViewRootImpl.performTraversals(ViewRootImpl.java:3152)  at android.view.ViewRootImpl.doTraversal(ViewRootImpl.java:2005)  at android.view.ViewRootImpl$TraversalRunnable.run(ViewRootImpl.java:8234)  at android.view.Choreographer$CallbackRecord.run(Choreographer.java:972)  at android.view.Choreographer.doCallbacks(Choreographer.java:796)  at android.view.Choreographer.doFrame(Choreographer.java:731)  at android.view.Choreographer$FrameDisplayEventReceiver.run(Choreographer.java:957)  at android.os.Handler.handleCallback(Handler.java:938)  at android.os.Handler.dispatchMessage(Handler.java:99)  at android.os.Looper.loop(Looper.java:223)  at android.app.ActivityThread.main(ActivityThread.java:7928)  at java.lang.reflect.Method.invoke(Native method)  at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:603)  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:947) |

也可以通过搜索关键字held by看是否有持锁情况等。有了这些信息再分析main.log应用的执行流程是否正常等来做版本是否真的有问题。

**3 查看CPU及io负载情况**

system.log中ANR in的打印前后通过dumpstate关键字搜cpuinfo的打印，仅在userdebug版本有该打印，log大致如下：

|  |
| --- |
| S12869B 08-24 10:48:38.480 23309 23309 D dumpstate: Cpu Core Count: 8  S12869C 08-24 10:48:38.480 23309 23309 D dumpstate: Timer Circle: 9999ms.  S12869D 08-24 10:48:38.480 23309 23309 D dumpstate: From time 32050445ms(2020-08-24 02:48:27.201379961 UTC) to 32060445ms(2020-08-24 02:48:37.201192646 UTC).  S12869E 08-24 10:48:38.480 23309 23309 D dumpstate:  S12869F 08-24 10:48:38.480 23309 23309 D dumpstate: \* CPU USAGE: | \* OTHER COUNTS:  S1286A0 08-24 10:48:38.480 23309 23309 D dumpstate: -1- IDLE USER SYSTEM NICE IOWAIT IRQ SOFTIRQ STEAL TOTAL | CTXT\_SWITCH FG\_FAULT FG\_MAJ\_FAULT  S1286A1 08-24 10:48:38.480 23309 23309 D dumpstate: cpu0(1): 53.47% 16.46% 24.40% 1.38% 0.29% 2.97% 1.09% 0.00% 100.00% | 30920 9028 496  S1286A2 08-24 10:48:38.480 23309 23309 D dumpstate: cpu1(1): 55.97% 14.34% 25.89% 2.09% 0.19% 0.69% 0.79% 0.00% 100.00% | 28466 11716 522  S1286A3 08-24 10:48:38.480 23309 23309 D dumpstate: cpu2(1): 61.00% 14.41% 21.91% 1.18% 0.19% 0.59% 0.69% 0.00% 100.00% | 27014 8346 343  S1286A4 08-24 10:48:38.480 23309 23309 D dumpstate: cpu3(1): 62.98% 12.93% 21.22% 1.38% 0.19% 0.49% 0.78% 0.00% 100.00% | 22872 12667 541  S1286A5 08-24 10:48:38.480 23309 23309 D dumpstate: cpu4(1): 73.96% 11.83% 12.22% 0.69% 0.19% 0.39% 0.39% 0.00% 100.00% | 19586 3489 456  S1286A6 08-24 10:48:38.480 23309 23309 D dumpstate: cpu5(1): 78.99% 9.36% 10.05% 0.59% 0.09% 0.49% 0.39% 0.00% 100.00% | 17600 2761 384  S1286A7 08-24 10:48:38.480 23309 23309 D dumpstate: cpu6(1): 57.82% 17.90% 18.08% 3.56% 1.49% 0.56% 0.56% 0.00% 100.00% | 8711 9887 953  S1286A8 08-24 10:48:38.480 23309 23309 D dumpstate: cpu7(1): 25.60% 13.50% 52.30% 6.80% 0.90% 0.50% 0.60% 0.00% 100.00% | 7903 6912 622  S1286A9 08-24 10:48:38.480 23309 23309 D dumpstate: ------------------  S1286AA 08-24 10:48:38.480 23309 23309 D dumpstate: Total: 58.78% 13.86% 23.17% 2.21% 0.45% 0.83% 0.66% 0.00% 100.00% | 163072 64806 4317  S1286AB 08-24 10:48:38.480 23309 23309 D dumpstate: |

如上对应的前三行Log，第一行是CPU的核数是几核，第二行是计时周期是10秒，第三行是统计运行情况的起始时间，这个时间是UTC时间格式，要在此基础上加8个小时，能对应打印log的实际时间戳。后面就是每个状态下的CPU占用情况，看下idle剩余,kernel和user占用情况，iowait占用情况等。这些信息之后是各个进程的占用情况的打印，可以看下有无明显占用较高的情况。

Userdebug版本没有上面的log打印，这是可以看下sgm.csv的log,对应第3部分3.7已做了具体说明。

**4 查看内存情况是否正常**

(1)main.log中搜lowmemorykiller: Kill关键字

|  |
| --- |
| Line 101714: M3E71AA 08-26 17:02:23.593 460 460 D lowmemorykiller: handle\_vmpressure 80 current cache 190248 swapfree 158632  Line 102310: M3E7402 08-26 17:02:24.226 460 460 E lowmemorykiller: Kill 'com.skype.raider:sync' (29162), uid 10153, oom\_adj 800 to free 37896kB  Line 102311: M3E7403 08-26 17:02:24.226 460 460 D lowmemorykiller: handle notify\_lmfs\_process\_killed done for 29162 |

lmk次数是否较多，swapfree是否为0或者剩余较少，lmk oom\_adj是否200及以下。

(2)通过sysinfo.log查看内存情况等，请查看第3部分3.6中的说明。

(3)看下kernel.log有无异常打印

|  |
| --- |
| Line 13713: A109E <4> [14452.127725][08-26 17:00:41.673] swcnblk-3-8: page allocation failure: order:0, mode:0x1080020(GFP\_ATOMIC), nodemask=(null)  Line 13747: A10C0 <4> [14452.128161][08-26 17:00:41.674] swcnblk-3-8: page allocation failure: order:0, mode:0x1080020(GFP\_ATOMIC), nodemask=(null) |

更多分析一般需要memory模块负责人做分析。

**5应用执行情况排查**

查看System.log和Main.log ANR超时时间段内是否有异常现象？应用做了什么操作？例如应用创建过程异常，做了不该做的操作，或者少做了一些操作；打出了应用异常的log或者系统的异常log，尤其是那种忽然反复出现的打印；有没有Native Crash、Fatal Exception、Looper Error、Fatal signal的信息等。

应用运行过程到ANR之间的报错和异常打印需要重点关注下，有无耗时的情况。此处以bug 1395268为例。

|  |
| --- |
| M4B0F1D 08-20 22:51:43.141 738 3640 I MediaPlayerService: media.extractor died. Sending death notification.  M4B0F1E 08-20 22:51:43.142 15788 18042 E MediaPlayerNative: error (100, 1)  M4B0F1F 08-20 22:51:43.142 15788 15788 E MediaPlayer: Error (100,1)  M4B0F20 08-20 22:51:43.142 15788 15788 V MediaPlayer: resetDrmState: mDrmInfo=null mDrmProvisioningThread=null mPrepareDrmInProgress=false mActiveDrmScheme=false  M4B0F21 08-20 22:51:43.142 15788 15788 V MediaPlayer: cleanDrmObj: mDrmObj=null mDrmSessionId=null  M4B0F22 08-20 22:51:43.142 2197 3913 W MediaProvider: java.io.IOException: Failed to create thumbnail  M4B0F22 08-20 22:51:43.142 2197 3913 W MediaProvider: at android.media.ThumbnailUtils.createAudioThumbnail(ThumbnailUtils.java:160)  M4B0F22 08-20 22:51:43.142 2197 3913 W MediaProvider: at com.android.providers.media.MediaProvider$6.getThumbnailBitmap(MediaProvider.java:4832)  M4B0F22 08-20 22:51:43.142 2197 3913 W MediaProvider: at com.android.providers.media.MediaProvider$Thumbnailer.ensureThumbnail(MediaProvider.java:4800)  M4B0F22 08-20 22:51:43.142 2197 3913 W MediaProvider: at com.android.providers.media.MediaProvider.ensureThumbnail(MediaProvider.java:5719)  M4B0F22 08-20 22:51:43.142 2197 3913 W MediaProvider: at com.android.providers.media.MediaProvider.openFileCommon(MediaProvider.java:5645)  M4B0F22 08-20 22:51:43.142 2197 3913 W MediaProvider: at com.android.providers.media.MediaProvider.openTypedAssetFileCommon(MediaProvider.java:5697)  M4B0F22 08-20 22:51:43.142 2197 3913 W MediaProvider: at |

对于这样的情况应用应该可以看出可能的异常点。

当然具体问题具体分析，根据问题情况选择要分析的log，做到有的放矢，不做无用功。但如上几步如果都能分析覆盖到了得出的结论和仅看一个点得出的结论可能是不同的，所以，分析ANR问题时需要通盘考虑，综合log中各方面的信息。切忌只看问题的一个侧面就轻率作出结论。

## ANR问题流转流程图



## ANR问题分析模板（应用部分）

下表提供ANR问题，具体ANR问题Owner的分析模板，请owner能按照下述模板分析填充到bugzilla上，确定需要其他模块进一步分析时，再转给其他owner分析。

|  |  |
| --- | --- |
| **ANR发生时间点和时间段** | 【说明】指出ANR发生的准确时间点，并根据原因确定超时时间段  例：ANR发生在04-29 00:30:40.580  Browser应用正在执行创建的工作，在00:30:32.050创建到在00:30:41.070创建成功获得焦点，耗时9秒，创建超过8秒此在  04-29 00:30:40.580发生ANR。并附上相应Log… |
| **在时间段内应用做了什么？是否有异常？** | 【说明】请查看定位的时间段，查看system.log和main.log确定应用在做什么操作？是否出现fatal exception、native crash、应用自己打出的异常信息，或者应用内该做的没做（比如说创建某个activity，应该做ABCDE, 而从log看它只做了ACE…），一直反复循环打印不正常log …. |
| **CPU信息** | 【说明】  请在system.log打出的ANR信息中check：   * CPU负载情况，单核大于10，四核大于14   例：E ActivityManager: Load: 8.6 / 12.81 / 12.55 CPU队列负载超过7，请关注  请把system.log或sgm.log中的cpu performance的log贴上，并参考system.log一起分析下列问题：   * 发生ANR的应用CPU的占用情况，请说明 * 是否有占CPU很高的进程 * 请列出CPU占前三的进程 * 总的CPU占有率 * IOWait是否高 * 从system.log里看，ANR之后发生ANR应用的CPU的占用率是否高 |
| **Memory信息** | 【说明】查看snapshot文件，memFree+caches<20M时，重点标记；是否有大量的GC操作 |
| **SnapShot(trace log)信息分析** | 【说明】应用主线程是否阻塞，尤其是有没有耗时的操作，请贴出来主线程的trace信息；应用进程状态是否正常；应用进程是否创建过多的线程；如果阻塞在跨进程调用，尝试找服务端服务。 |
| **其他异常** | 【说明】其他可疑信息 |
| **结论** | 转到其他模块的理由或者得出引起问题的原因等，还是需要协助分析。 |

## 分析ANR问题的建议

由于ANR问题通常不是由单一原因产生，建议在分析ANR问题时，如果有怀疑点尽量同时寻找两个及以上的证据支持怀疑点，不找孤据。

为了提高分析ANR问题的能力，建议应用同事在转走bug时，尽量cc给自己追踪Bug状态，便于以后分析同类ANR问题的效率。

一段时间内如果某个应用反复报出同一类ANR，则需要应用Owner配合做更进一步的调查确认。

# ANR问题分析实例

前面已针对ANR分析的log用了较多的篇幅做了说明，下面以几个具体ANR bug实例来指导后续分析ANR问题。由于log占用篇幅较多，此处仅列出部分，详情请参阅bug对应的comment分析内容。

## activity多实例

【类型描述】多实例问题

【特征】这一类问题，在CPU统计数据中，可以看到system\_server进程的占用较高；查看发生ANR进程的主线程trace，可以没有明显异常的情况。

【实例分析】Bug 1384433 - [14.04][BM&CI][monkey][android11.0\_sharkl3\_GMS]monkey 测试中com.android.contacts出现ANR问题。

1. 找ANR时间点

|  |
| --- |
| E0BD62D 08-06 15:56:13.700 897 27976 I am\_anr : [0,2088,com.android.contacts,818462277,Input dispatching timed out (ActivityRecord{330130c u0 com.android.contacts/.quickcontact.QuickContactActivity t4325} does not have a focused window)]  S0C03DA 08-06 15:56:29.280 897 27976 E ActivityManager: PID: 2088  S0C03DA 08-06 15:56:29.280 897 27976 E ActivityManager: Reason: Input dispatching timed out (ActivityRecord{330130c u0 com.android.contacts/.quickcontact.QuickContactActivity t4325} does not have a focused window)  S0C03DA 08-06 15:56:29.280 897 27976 E ActivityManager: Parent: com.android.contacts/.quickcontact.QuickContactActivity  S0C03DA 08-06 15:56:29.280 897 27976 E ActivityManager: Load: 8.53 / 9.45 / 9.8 |

得知ANR类型和初步负载情况。

2．发生ANR的进程是com.android.contacts，从trace看怀疑有阻塞的情况

|  |
| --- |
| 该trace无明显异常未列出 |

1. CPU负载情况

System.log中dumpstate关键字CPU负载信息。

1. ANR前系统运行情况分析

|  |
| --- |
| S0BD5F4 08-06 15:56:13.283 897 1152 I WindowManager: Dump Win No.38 win= Window{6c4b60a u0 VolumeDialogImpl}, flags=0x18c0028, canReceive=false, ViewVisibility=0, isVisibleOrAdding=true  S0BD5F6 08-06 15:56:13.283 897 1152 I WindowManager: Dump Win No.39 win= Window{1b630bc u0 VolumeDialogImpl}, flags=0x18c0028, canReceive=false, ViewVisibility=0, isVisibleOrAdding=true  从Dump Win No.的信息可以看到VolumeDialogImpl有多个实例。 |

能够判断得出应用多实例的结论即可转模块分析。

## 应用crash导致的ANR问题

【类型描述】应用crash导致ANR

【特征】这一类问题，在CPU统计数据中，可以看到system\_server进程的占用较高；查看发生ANR进程的主线程trace，可以没有明显异常的情况。

【实例分析】[Bug 1387490](https://bugzilla.unisoc.com/bugzilla/show_bug.cgi?id=1387490) - [11.13 h][BM&CI][monkey首错][android11.0\_sharkL3-WTWD]monkey测试中com.android.launcher3 出现ANR问题（PID: 1720累计出现16次）

忽略了ANR时间点，CPU负载情况，trace.log几log分析点，直接列出关键点。此问题关注的点是焦点窗口切换。

|  |
| --- |
| S381A55 08-11 13:36:11.796 1053 2314 I WindowManager: Input focus has changed to Window{592637d u0 com.android.camera2/com.android.camera.CameraLauncher}  S381B10 08-11 13:36:12.031 1053 2333 I WindowManager: Input focus has changed to null  S38220D 08-11 13:36:24.890 1053 1300 I WindowManager: Input focus has changed to Window{9cabca6 u0 com.android.launcher3/com.android.searchlauncher.SearchLauncher}  从焦点窗口的切换打印看是从CameraLauncher窗口切换到SearchLauncher窗口 |

再结合ANR前进程的执行情况，看到了camera crash

|  |
| --- |
| M381AE7 08-11 13:36:11.991 8049 8049 V CAM\_CameraActivity Drea: onConfigurationChanged  M381AE8 08-11 13:36:11.993 732 1874 E CameraService: supportsCameraApi: Unknown camera ID 1  M381AE9 08-11 13:36:11.995 8049 8049 D AndroidRuntime: Shutting down VM  C381AEA 08-11 13:36:11.997 8049 8049 E AndroidRuntime: FATAL EXCEPTION: main  C381AEA 08-11 13:36:11.997 8049 8049 E AndroidRuntime: Process: com.android.camera2, PID: 8049  C381AEA 08-11 13:36:11.997 8049 8049 E AndroidRuntime: java.lang.IllegalArgumentException: supportsCameraApi:2342: Unknown camera ID 1  C381AEA 08-11 13:36:11.997 8049 8049 E AndroidRuntime: at android.hardware.camera2.CameraManager.throwAsPublicException(CameraManager.java:1001)  C381AEA 08-11 13:36:11.997 8049 8049 E AndroidRuntime: at android.hardware.camera2.CameraManager.getCameraCharacteristics(CameraManager.java:449) |

得出结论：系统整体负载不大，焦点刚切换到camera，出现了camera crash，后面窗口没能及时切换到launcher导致的ANR，需要camera处理下camera crash问题。

## iowait占用高

【类型描述】iowait占用高导致ANR

【特征】这一类问题，在dumpstate打印的CPU统计数据中，可以看到几个核idle,user,system占用不高，iowait占用都超过50%；查看发生ANR进程的主线程trace，可以没有明显异常的情况。

【实例分析】Bug 1389353 - [PSST][Android11] [Sharkl3][3G][WTWD][推荐]monkey测试,#3707测试5.58小时出现ANR in com.android.contacts

(com.android.contacts/.quickcontact.QuickContactActivity),1/8(台)累计出现2次

该分析过程忽略了ANR时间点，trace.log几log分析点，直接列出关键点。此问题关注的点是iowait占用高。

System.log

|  |
| --- |
| S939399 08-12 21:49:48.617 10191 10191 D dumpstate: -1- IDLE USER SYSTEM NICE IOWAIT IRQ SOFTIRQ STEAL TOTAL | CTXT\_SWITCH FG\_FAULT FG\_MAJ\_FAULT  S93939A 08-12 21:49:48.618 10191 10191 D dumpstate: cpu0(1): 0.00% 5.80% 16.70% 1.60% 72.70% 2.00% 1.30% 0.00% 100.00% | 25836 13229 3502  S93939B 08-12 21:49:48.618 10191 10191 D dumpstate: cpu1(1): 0.00% 5.90% 14.10% 0.40% 78.10% 0.80% 0.80% 0.00% 100.00% | 15680 6722 1710  S93939C 08-12 21:49:48.618 10191 10191 D dumpstate: cpu2(1): 0.00% 4.60% 12.30% 0.50% 81.20% 0.60% 0.60% 0.00% 100.00% | 12890 3948 2258  S93939D 08-12 21:49:48.618 10191 10191 D dumpstate: cpu3(1): 0.00% 4.80% 11.40% 1.20% 81.20% 0.70% 0.70% 0.00% 100.00% | 9564 6681 1206  S93939E 08-12 21:49:48.618 10191 10191 D dumpstate: |

得出结论：系统负载idle剩余较少，user，system占用不高，iowait占用较高。另外就是包含RippleForeground:关键字的log 较多打印对io的影响。

## log过多打印对系统的影响

【类型描述】某类log过多打印

【特征】这一类问题，CPU整体负载看不大，ANR统计信息是logd和adb占用负载高；查看发生ANR进程的主线程trace没有明显异常的情况。

【实例分析】[Bug 1391064](https://bugzilla.unisoc.com/bugzilla/show_bug.cgi?id=1391064) - [Multimedia][Video][Android11][Sharkl5Pro][T610][4G][CTCC][Native] 打开屏幕录制，设置开启录制设备音频和麦克风，开始录制，闹钟响起，偶现“系统界面没有反应”

该分析过程忽略了ANR时间点，trace.log几log分析点，直接列出关键点。此问题关注的点是adb log过多打印影响。

system.log

|  |
| --- |
| S81DF51 08-13 10:21:41.642 1007 16203 E ActivityManager: CPU usage from 211786ms to 0ms ago (2020-08-13 10:18:05.665 to 2020-08-13 10:21:37.451):  S81DF51 08-13 10:21:41.642 1007 16203 E ActivityManager: 49% 299/logd: 8.3% user + 41% kernel / faults: 557 minor  S81DF51 08-13 10:21:41.642 1007 16203 E ActivityManager: 43% 584/adbd: 10% user + 32% kernel / faults: 1433 minor |

main log

|  |
| --- |
| 818CBF 08-13 10:21:35.327 584 925 I adbd : SubmitRead return ture  M818CC0 08-13 10:21:35.328 584 925 I adbd : Update block->pending -> false  M818CC1 08-13 10:21:35.328 584 925 I adbd : Update block->pending -> ture  M818CC2 08-13 10:21:35.328 584 925 I adbd : SubmitRead return ture  M818CC3 08-13 10:21:35.329 584 925 I adbd : Update block->pending -> false  M818CC4 08-13 10:21:35.329 584 925 I adbd : Update |

得出结论：系统整体负载不大,log中大量打印adbd相关的log，对系统负载有一定的影响。

## Lowmemorykiller杀进程导致的ANR

【类型描述】lowmemorykiller杀进程导致ANR

【特征】这一类问题，CPU整体负载看不大，ANR统计信息是system\_server和swapd0占用负载高；查看发生ANR进程的主线程trace没有明显异常的情况。

【实例分析】Bug 1398519 - [PSST][Android11][Sharkl5Pro][T618][3G][GMS]monkey测试,#0498测试7.16小时出现ANR in com.android.launcher3 (com.android.launcher3/com.android.searchlauncher.SearchLauncher),1/10(台)累计出现7次

该分析过程忽略了ANR时间点，trace.log几log分析点，直接列出关键点。此问题关注的点是lowmemorykiller杀到了adj 0，且频繁被杀。

|  |
| --- |
| MC0C081 08-22 16:06:28.978 300 300 I lowmemorykiller: Kill 'com.android.launcher3' (20497), uid 10166, oom\_adj 0 to free 64752kB  MC0C0C6 08-22 16:06:29.097 300 300 I lowmemorykiller: Kill 'com.android.launcher3' (20519), uid 10166, oom\_adj 0 to free 65032kB  MC0C108 08-22 16:06:29.186 300 300 I lowmemorykiller: Kill 'com.android.launcher3' (20542), uid 10166, oom\_adj 0 to free 62128kB  MC0C14C 08-22 16:06:29.307 300 300 I lowmemorykiller: Kill 'com.android.launcher3' (20564), uid 10166, oom\_adj 0 to free 64760kB  MC0C18D 08-22 16:06:29.409 300 300 I lowmemorykiller: Kill 'com.android.launcher3' (20586), uid 10166, oom\_adj 0 to free 61700kB |

仅从频繁杀adj0可以得知内存肯定有些问题，如果需要细看就要结合kernel log再做分析。

## 焦点窗口切换问题

【类型描述】焦点窗口切换类ANR

【特征】这一类问题，CPU整体负载看不大，ANR统计信息没有明显占用高的进程，应用执行过程没有明显耗时等；查看发生ANR进程的主线程trace没有明显异常的情况。

【实例分析】Bug 1363478 - [PSST][Android11][Sharkl3][3G][WTWD][PTR2Block]monkey测试,#3296测试0.59小时出现ANR in com.android.systemui,9/9(台)累计出现73次

该分析过程忽略了ANR时间点，CPU负载内存情况，trace.log几log分析点，直接列出关键点。此问题关注的点是焦点窗口切换。

System log

|  |
| --- |
| S156390 07-08 15:53:01.859 975 3395 I WindowManager: Input focus has changed to Window{170ebb8 u0 NotificationShade}  S1565A6 07-08 15:53:03.710 975 2457 I WindowManager: Input focus has changed to Window{cadc750 u0 com.android.contacts/com.android.contacts.quickcontact.QuickContactActivity}  窗口焦点已从NotificationShade切换到QuickContactActivity但ANR在com.android.systemui |

|  |
| --- |
| M0C2D88 07-07 10:44:49.732 976 1214 W InputDispatcher: Dispatching key to Window{7bad4bd u0 com.android.contacts/com.android.contacts.quickcontact.QuickContactActivity} even though there are other unprocessed events  S0C2DC3 07-07 10:44:49.960 976 1214 I WindowManager: Input event dispatching timed out sending to NotificationShade. Reason: 7ef2b36 NotificationShade (server) is not responding. Waited 10000ms for MotionEvent(deviceId=-1, source=0x00010004, displayId=0, action=MOVE, actionButton=0x00000000, flags=0x00000000, metaState=0x00000000, buttonState=0x00000000, classification=NONE, edgeFlags=0x00000000, xPrecision=1.0, yPrecision=1.0, xCursorPosition=nan, yCursorPosition=nan, pointers=[0: (4.0, 2.0)]), policyFlags=0x6b000000 |

QuickContactActivity有没处理的event，但ANR是发送给NotificationShade的input事件超时。最后得出结论：系统整体负载不大，从log看焦点窗口已经切换，log一直有InputDispatcher: Still no focused window. Will drop the event的log打印，最后提示input事件没有分发到NotificationShade。

## GPU问题导致ANR

【类型描述】GPU问题导致的ANR类

【特征】这一类问题，CPU整体负载看不大，ANR统计信息没有明显占用高的进程，应用执行过程没有明显耗时等；查看发生ANR进程的主线程trace一般执行graphics的相关调用上。

【实例分析】Bug 1363478 - [PSST][Android11][Sharkl3][3G][WTWD][PTR2Block]monkey测试,#3296测试0.59小时出现ANR in com.android.systemui,9/9(台)累计出现73次

该分析过程忽略了ANR时间点，CPU负载内存情况，直接列出关键点。此问题关注的点是graphics相关调用栈，和应用执行过程的log。

trace log

|  |
| --- |
| "main" prio=5 tid=1 Waiting  | group="main" sCount=1 dsCount=0 flags=1 obj=0x719cdd70 self=0xdd983810  | sysTid=1559 nice=10 cgrp=default sched=0/0 handle=0xeace8470  | state=S schedstat=( 45228208787 7423698999 22749 ) utm=4113 stm=409 core=5 HZ=100  | stack=0xff2a2000-0xff2a4000 stackSize=8192KB  | held mutexes=  at sun.misc.Unsafe.park(Native method)  - waiting on an unknown object  at java.util.concurrent.locks.LockSupport.park(LockSupport.java:190)  at java.util.concurrent.locks.AbstractQueuedSynchronizer.parkAndCheckInterrupt(AbstractQueuedSynchronizer.java:868)  at java.util.concurrent.locks.AbstractQueuedSynchronizer.acquireQueued(AbstractQueuedSynchronizer.java:902)  at java.util.concurrent.locks.AbstractQueuedSynchronizer.acquire(AbstractQueuedSynchronizer.java:1227)  at java.util.concurrent.locks.ReentrantLock$NonfairSync.lock(ReentrantLock.java:216)  at java.util.concurrent.locks.ReentrantLock.lock(ReentrantLock.java:294)  at android.view.SurfaceView.updateSurface(SurfaceView.java:1018)  at android.view.SurfaceView.setVisibility(SurfaceView.java:329)  at com.xunmeng.pinduoduo.goods.ProductDetailFragment.onPause(ProductDetailFragment.java:1047)  at android.support.v4.app.Fragment.performPause(Fragment.java:2645) |

该主线程的调用栈执行在waiting状态，子线程10772

|  |
| --- |
| "Thread-20" prio=5 tid=150 Native  | group="main" sCount=1 dsCount=0 flags=1 obj=0x13752c60 self=0xb556ca10  | sysTid=10772 nice=0 cgrp=default sched=0/0 handle=0xa046a1c0  | state=R schedstat=( 12930167993 135957315 610 ) utm=2 stm=1290 core=7 HZ=100  | stack=0xa0367000-0xa0369000 stackSize=1040KB  | held mutexes=  native: #00 pc 000715d8 /apex/com.android.runtime/lib/bionic/libc.so (\_\_ioctl+12)  native: #01 pc 0003f1c7 /apex/com.android.runtime/lib/bionic/libc.so (ioctl+30)  native: #02 pc 00040e11 /vendor/lib/libsrv\_um.so (???)  native: #03 pc 0000755f /vendor/lib/libsrv\_um.so (???)  native: #04 pc 00007397 /vendor/lib/libsrv\_um.so (???)  native: #05 pc 00012e01 /vendor/lib/libsrv\_um.so (???)  native: #06 pc 000135b5 /vendor/lib/libsrv\_um.so (PVRSRVCacheOpBatchExec+336)  native: #07 pc 00013277 /vendor/lib/libsrv\_um.so (PVRSRVCacheOpBatchAdd+418)  native: #08 pc 00003c53 /vendor/lib/hw/gralloc.sp9863a.so (???)  native: #09 pc 00005add /vendor/lib/hw/gralloc.sp9863a.so (???)  native: #10 pc 0000bb0d /vendor/lib/hw/gralloc.sp9863a.so (???)  native: #11 pc 00007f8f /vendor/lib/hw/gralloc.sp9863a.so (???) |

Kernel log中的异常打印

|  |
| --- |
| 15EF5 <3> [16473.751150][07-11 16:06:24.751] PVR\_K:(Error): 1559-10772: PMRAcquireKernelMappingData() failed (PVRSRV\_ERROR\_PMR\_NO\_KERNEL\_MAPPING) in CacheOpPMRExec() [1562]  15EF6 <3> [16473.754384][07-11 16:06:24.754] PVR\_K:(Error): 1559-10772: PMRAcquireKernelMappingData() failed (PVRSRV\_ERROR\_PMR\_NO\_KERNEL\_MAPPING) in CacheOpPMRExec() [1562]  15EF7 <3> [16473.757257][07-11 16:06:24.757] PVR\_K:(Error): 1559-10772: PMRAcquireKernelMappingData() failed (PVRSRV\_ERROR\_PMR\_NO\_KERNEL\_MAPPING) in CacheOpPMRExec() [1562]  15EF8 <3> [16473.760174][07-11 16:06:24.760] PVR\_K:(Error): 1559-10772: PMRAcquireKernelMappingData() failed (PVRSRV\_ERROR\_PMR\_NO\_KERNEL\_MAPPING) in CacheOpPMRExec() [1562]  15EF9 <3> [16473.762913][07-11 16:06:24.762] PVR\_K:(Error): 1559-10772: PMRAcquireKernelMappingData() failed (PVRSRV\_ERROR\_PMR\_NO\_KERNEL\_MAPPING) in CacheOpPMRExec() [1562] |

这个就需要尽可能多的分析log才能分析出来，要对异常打印的Log敏感。PVR\_K:(Error)可能开始不知道是哪个模块的log，但大量打印是否正常需要考虑到，再结合调用栈等情况综合分析，得出结论：系统整体负载不大，两次ANR调用栈相同，主线程waiting装态，从VR\_K:(Error): 1559-10772: PMRAcquireKernelMappingData() failed (PVRSRV\_ERROR\_PMR\_NO\_KERNEL\_MAPPING) in CacheOpPMRExec() [1562]的log打印看，子线程10772在/vendor/lib/libsrv\_um.so的调用。

## 应用导致的ANR问题

【类型描述】应用问题导致的ANR类

【特征】这一类问题，CPU整体负载看不大，ANR统计信息没有明显占用高的进程，应用执行过程没有明显耗时等；查看发生ANR进程的主线程trace一般执行graphics的相关调用上。

【实例分析】Bug 1360255 - [TJ][Telephony][Android11][Sharkl5Pro][T610][3G][CTCC][Native][monkey]monkey测试偶现com.android.email happen ANR

该分析过程忽略了ANR时间点，CPU负载内存情况，直接列出关键点。此问题关注的点是trace调用栈是执行在contentprovider相关的binder调用上，不要见到有binder调用就说binder有问题了。这个问题经过多次流转的主要原因是没有利用好调用栈，获取不到焦点窗口时没有考虑应用耗时的情况。这个通过bug comment来看整个分析过程更好。

# 小结

本文主要围绕着分析和处理ANR问题展开，从ANR的概述谈起，知道ANR是什么，有哪些类型，触发原因，常见ANR原因几个方面说起。ANR的基础知识只能起到个抛砖引玉的作用，只列出了android知识体系的很小一部分。android系统的知识体系很庞大，作为一个android开发者应该具备了一定的基础，当然随着分析问题的深入可能需要用到更多的知识，这些需要自己分析问题时做积累。分析android问题最关键的是分析log，针对ANR常用的几个log文件做了简要说明，主要是指导分析过程看哪些log。第4部分介绍了ANR问题的分析流转流程，ANR问题的分析步骤其实不是固定的流程，怎么能有效的分析问题才是关键，问题流转是基于分析log的证据转出到其他模块，转出过程不要太随意，新手可以从ANR问题分析模板开始指导自己分析问题，随着分析能力的提升可以摒弃模板找到核心问题点。针对ANR问题的分析建议是多看别人的分析过程和结论。最后以最近分析ANR问题实例对常见问题做分析，以指导正常的分析ANR问题。由于知识维度有限，有些知识点写的难免有偏差和不足之处，欢迎各位批评指正，不吝赐教，谢谢