Android 平台避免 ANR 编程注意事项 V1.0

版本信息：

版本	修改人	日期	修改内容	备注
V1.0	陶飞	20170317	安卓平台避免 ANR 编程注意事项	

1. 目的：

为了规范软件工程师在 android 代码编写过程中产生应用或系统 ANR 无响应行为，降低手机崩溃次数和改善手机性能，使得 vivo 产品性能得到极大提升。

2. 适用范围

该规范适用于 android 平台驱动代码、框架代码以及应用代码。

3. 测试复现

因为 ANR 问题错综复杂，涉及模块多，所以没有一个固定可循的复现步骤。最常见的是通过 monkey 等压力测试来暴露问题，当然不可避免的会存在测试本身引起的性能问题。对于有固定线索，如 ：

- ANR 与网络/io 有关：可切换不同的网络、拷贝文件等方式复现；
- ANR 与调用接口有关：比如调用 telephony、audio 等接口，可操作 SIM 卡、拨打电话、播放音视频、拍照等方式复现；
- ANR 与频率有关的：比如可频繁发通知与短信、安装大量 apk 扫描、频繁亮灭屏等方式复现。

4. 避免 ANR 的典型注意事项

4.1. 应用主线程耗时

•严重性：致命　　发生可能性：频繁

应用 app 主线程禁止做以下耗时工作：**网络连接超时操作、IO 操作、数据库耗时操作、Binder 频繁操作、反射操作、大量级的循环、位图变换、计算操作、频繁循环查询动作、广播 OnReceive()方法耗时、获取数据埋点、Handler 消息处理耗时、调用 waitDone()、Sleep()等操作**；这些耗时操作主线程都应避免，使用子线程执行，最好使用线程池进行管理。其中大部分模块最常见的耗时错误有

•主线程数据库查询耗时操作：

```
Cursor listsNumberCursor = resolver.query(uri, projectIn, where.toString(), null,null);
```

数据库查询、插入、删除等操作不要在主线程或者 ui 线程执行，这种类型操作对系统影响也较大；数据库好的设计应该是：如果有频繁操作数据库的需求，首先查询数据库，将获取的数据缓存到内存，然后监听数据库，并实时更新内存数据。其他逻辑层直

接查询内存数据。

•主线程获取数据埋点判断属数据库查询：

if (mCollector.getControlInfo(EVENT_ID_GLOBAL_SEARCH))

•主线程计算耗时：

如 title.setText(titleContent)问题，使用超长字符串最后会走到进行计算工作，此外这个 setText 也是在 UI 线程做 View.draw 绘制的时候会增加性能消耗。

•主线程 ContentObserver 处理耗时：

ContentObserver 构造函数是一个 handler 初始化的接口，默认情况下 handler 将是运行在主线程中（构造时当前的线程），OnChange 的响应（如果有耗时）也是在该 handler 中执行的，所以 onChange 中不能出现耗时的操作。可以使用一个 handlerThread 构造一个 handler 初始化 ContentObserver，这样 onChange 就运行在子线程中了。

•主线程 Handler 消息处理耗时：

注意识别 handler 运行态(主线程、子线程，默认为主线程，根据 Looper 决定)，对于运行于主线程的 handler 消息处理，不能存在耗时操作。考虑一下 handlerthread。

4.2. 广播注册过多或耗时

•严重性：致命　　发生可能性：频繁

应用 app 应避免注册过多动态有序或静态广播：**SCREEN_ON、SCREEN_OFF、TIME_TICK、sim 卡切换、网络切换**等广播与系统交互频繁，容易阻塞且难查；

失效原因也有可能并非广播本身处理耗时原因，打个比方：SCREEN_ON 注册或监听频繁的话发生该类型 ANR 的概率就会比其他模块高，原生代码弃用的广播也请及时去除减少崩溃概率；此外，主线程广播 OnReceive()不能处理耗时操作，耗时操作放入子线程处理或另起 service。

•最常见 ANR 异常的广播有：

Broadcast of Intent { act=android.intent.action.SCREEN_ON flg=0x50000010 }
Broadcast of Intent { act=android.intent.action.SCREEN_OFF flg=0x50000010 }
Broadcast of Intent { act=android.intent.action.TIME_TICK flg=0x50000014 (has extras) }

4.3. 主线程频繁跨进程调用

•严重性：致命　　发生可能性：频繁

应用 app 禁止主线程频繁跨进程通信：**跨进程本身 binder 比较耗时操作，返回结**

果速度取决于系统状态，而且还容易卡在 binder 对端不返回。跨进程尽量少调用，比如有的方法是不需要通过 binder 查询数据。

•如果有频繁跨进程(binder)查询数据(如白名单)的需求，首先只跨进程调用一次并本地缓存，然后监听缓存列表并实时更新，其它情况直接查询缓存列表，减少每次跨进程查询频率(防止 binder 阻塞或 binder 被占满)。

•adb shell cat /d/binder/stats | sed -n '/2029/,+35p'命令查看，如果在某个功能运行过程中，BC_TRANSACTION 的值增大的速度越快，说明 binder 调用越频繁，就要引起重视了，会增加了引起 ANR 的概率。

案例：

1) wifi getScanResult:本身较频繁耗时，还通过跨进程获取后台运行的应用程序；
2) 长截屏：频繁调用系统数据库 putInt 写值，开了一个线程去写数据库，又为了防止频繁写，又做了一个抓取后台进程的操作，把一个公共控件和一个系统服务给整到一起不合理；
3) 锁屏：在 touch 事件分发的时候去频繁跨进程调用 PowerManager.userActivity 更新亮屏时间；
4) 桌面：launcher 查询应用是否支持分身，安装的应用过多的话存在频繁跨进程耗时 CloneUtilities.initWhiteList();//获取应用分身白名单。

4.4. 警惕耗时接口

•严重性：致命　　发生可能性：频繁

AMS 耗时接口： getRunningAppProcesses()、forceStopPackage()、getRunningServices()、getRunningTasks()、getRecentTasks()、getCurrentUser()、getProcessMemoryInfo() 、screenTurnedOff()等，应用 app 在调用这些接口的时候需要注意使用场景方法**不要频繁调用**，会长时间持有 AMS 的锁影响性能，容易卡顿；其中 getRunningTasks(1)：参数尽量小，每多 1 就会多循环 1 遍列表，增加持有锁的时间。

IO 耗时接口：getPhoneStorageState()、getExternalStorageState()、isPhoneStorageMounted()等 io 接口，应用 app 应**禁止在主线程调用**这些耗时接口，这类接口存在 io 耗时且容易卡在 PKMS \ MountService binder 返回，当拷贝大量文件的时候会被频繁调用，插入损坏 SD 卡情况下也会容易卡住。

查询耗时接口：getControlInfo()属于数据库查询、applicationInfo.loadLabel()获取过多 app 的 lable 时解析耗时，应用 app 应**禁止在主线程调用**此接口。

案例：

1) getRunningAppProcesses 接口：框架频繁调用 notifyStatusBarColorInfo()接口通知 systemui 变色，systemui 再频繁调用 getRunningAppProcesses()接口获取当前界面的包名判断是否是待机存在性能耗时；
2) screenTurnedOff 接口：screenTurnedOff() 这个接口只能在 DisplayPowerController thread 中调用，android6.0 之后，其持有了 AMS 锁，有发生死锁的风险。为了解决 PD1610 等机型在待机时 Screen 状态没有通知到 Battery 的问题，调用了一个系统原生的接口 DisplayPowerController. setScreenState(int)，最终会调用到 PhoneWindowManager.screenTurnedOff()，产生死锁。
3) SensorEventListener 接口：sensor 数据变化较频繁，多线程持锁易引起死锁问题。例如监听到 sensor 数据变化后，会判断夜明珠的显示状态，这时使用了 NightPearlManager 中的 isNightPearlShowing 方法，并申请了对象锁，与主线程产生了互锁。

4.5. 预防占用 CPU 高

•严重性：致命　　发生可能性：频繁

进程方应避免多线程并发操作(加锁)，避免频繁扫描\认证失败后持续扫描\认证情况，避免主线程与子线程死锁，避免子线程使用完不释放等情况；native 层采用的线程数不建议超过 CPU 核数/2。

一个进程占用 CPU 高会非常影响系统性能，造成其它系统以及三方应用连续 ANR 崩溃，也是最常见引起 ANR 的场景。正常情况为某个进程启用多个线程同时处理或单个线程持有主线程锁耗时过久占用 CPU；大量线程同时开启，极大占用 cpu，并导致更新 ui 卡顿抢占 ui 线程资源； 可根据“开发者选项” > ”显示 CPU 使用情况”进行调试。

占用 CPU 常见的系统进程： mediaServer、systemserver、dex2oat、gxFpDaemon、mmcqd、kworker、RenderThread、VivoOMXMsg

占用 CPU 常见的应用进程：com.qileyx.ddz.vivo、com.yinhan.hunter.bbg 、

com.yhgame.xh.vivo、com.tencent.mobileqq、com.zhongduomei.rrmj.society、air.tv.douyu.android

占用 CPU 常见的测试进程：adb、mobile_log_d(抓 log)、snake(root)

案例：

1) Dex2oat 占用 CPU: 微信"tinker"热修复 dex2oat cpu 占用率高, 微信为了加快补丁的修复，会开启 3~4 个进程(每个进程 4 个线程)同时编译；
2) VivoOMXMsg 占用 CPU：vivoextractor 在析构的时候 mMessageThread 没有释放，开启多个 VivoOMXMsg 线程；
3) Settings 进程占用 CPU：主线程和子线程未加锁同时 AsyncTask.execute 并发锁死，代码执行过久 CPU 高，造成卡顿引起黑白屏；
4) gxFpDaemon 占用 CPU：指纹单线程认证时间过长导致一直占用 kernel，引起 gxFpDaemon CPU 高。

4.6. 公共进程预防 binder 被占满

•严重性：致命　　发生可能性：频繁

system_server、mediaserver、phone 应重点关注死锁或者应用频繁请求定位或数据库查询等情况，避免 16 个 binder 被占满。

这三个进程是各个应用最频繁跨进程调用的公共核心进程，而且 system_server 里面跑很多框架服务、 mediaserver 里面跑 camera、auido、video、bluetooth 等各种功能、phone 也是与 sms、sim 卡、mms 等各种交互频繁，一旦这三个进程出异常 16 个 binder 被占满情况，各应用 binder 调用其接口均会遭殃卡住引发大范围引起 ANR，从而影响手机性能。

案例：

1) Systemserver 进程 binder 被占满：LocationManagerService 定位原生问题，app 写的不规范不停的向系统请求网络定位，当前这个进程 died 后，LMS 会不停的收到 binderDied 回调，此时会持有锁，然后在这个里面做 remove 动作;当问题 app 请求的定位多的时候，系统服务进程的 binder 都被 LMS 的线程占用了,从而导致 system_server 进程对其它客户的的 binder 调用就会卡了 anr，甚至时间太久系统会出现上层 watchdog 超时；

2) Systemserver 进程 binder 被占满：vivo_firewall_thread 防拉起:关联启动框架里面频繁调用 onChange 查询数据库耗时，回占用 system_server 的 binder 线程导致其它进程无法得到 binder 资源 ANR，且短时间大量的消息，然后"vivo_firewall_thread"没有执行超时，也会容易 wdt 了。
3) Phone 进程 binder 被占满：SIM 模块，360 安全卫士、微信电话本、触宝电话等在 sim 卡联系人缓存初始化时，会进行多次查询，lock 等待，占用 binder，使得其他应用来查询时，binder 数量达到上限（16 个），phone ANR，查询的应用（联系人、system UI、i 管家）也发生 ANR。
4) Phone 进程 binder 被占满：SMS 模块，SmsProvider 中的事务处理不规范(没有在 finally 中 db.endTransaction())，一旦处理逻辑出现异常就有可能没有跑 db.endTransaction()，以致数据库卡死，造成 phone 进程所有 binder 阻塞。
5) MediaServer 进程 binder 被占满：CameraService connect 函数连接异常导致 AutoConditionLock 死锁，导致 mediaserver 所有 binder 等待 AutoConditionLock 的锁从而被占满，其它进程此时调用 media 接口如(camera.open()、media.prepare()、AudioSystem.setParameters()、AudioManager.getStreamVolume()、media.SoundPool.play()等)均会得不到 mediaserver binder 资源而无响应。

4.7. 注意 SettingsUtils 类的读写

•严重性：一般　　发生可能性：频繁

应用在主线程以及 UI 线程应尽量减少调用 Settings.System.putInt、getInt、putString、getString 等读写操作。这是一个读写数据耗时操作，系统性能较差、loading 较重、或刷新时间较久时均会引起 ANR。

原生逻辑，但是调用频繁的话读写数据库或多或少都有性能问题，目前状态栏、桌面、vivo 输入法、i 管家、设置、信息、电话/信息存储、指纹解锁等模块均有轻量级此问题。

案例：

添加长截屏时有在 AbsListView.java 中添加 setListiewBottom 操作，该操作会让很多 APP 主线程绘制更新时跑到 putInt 操作耗时。

4.8. 应用减少调用框架接口

•严重性：一般　　发生可能性：频繁

应用在主线程时应尽量减少跨进程调用，一是上层很难识别是应用问题还是框架问题，二是容易 binder 阻塞，三是对于应用上层来说追查 binder 对端情况比较困难以及推进缓慢。尤其像框架接口 systemserver、mediaserver(audio\media\camera\bt)、wifi、telephony 等又是非常容易出错的地方，根源是框架去解框架异常。

应用主线程调用上报的是崩溃 ANR，在大数据中会贡献异常次数。子线程调用的话则是功能异常，比如 telephony 会是通话异常，可能会直接报到框架 telephony 处。在子线程处理时需要注意 handlerthread 泄露问题，退出时必须注销，防止泄露。另外，在主线程 try…catch 了，post 到子线程时也要 try…catch,否则异常时无法捕获到。

4.9. 注意应用进程 provider timeout 会阻塞 system_server

•严重性：致命　　发生可能性：一般

应用进程因某些异常发生 Content Provider 一直 timeout，通常这类异常：存在大量同时请求访问数据库，开机启动和应用频繁杀死拉起会导致 systemserver 的 binder 线程处于 ContentProvider 状态等待而被占满，导致其他进程无法和 systemserver 进行 binder 通信，然后出现 anr。

Provider　timeout 模块需检查：

*需要找出 Provider 为何会迟迟未 publish（完成启动）

*需要找出同时多次请求访问该数据库的地方

*进程频繁启动被杀后又被频繁拉起是否正常

案例：

1) com.vivo.weather.provider/.WeatherProvider：优化天气存储逻辑，去掉开机启动和应用杀死拉起
2) com.bbk.iqoo.logsystem/.databases.CommonProvider：因为 log 采集数据需要常驻的进程，所以需要提高进程的优先级，使得进程不容易被杀死。

4.10. 异步、锁的情况

•严重性：一般　　发生可能性：频繁

异步代码逻辑过多会导致软件质量不可控，比如异步更新 UI\耗时类操作异步通知\异步线程优化性能等操作，异步操作数量过多缺乏切实有效的统一规范，容易制造以下几类异常:

1.异步回调执行的时候，环境导致空指针异常

2.线程间缺乏必要的同步机制，压力环境下会导致竞争冲突类问题，尤其是异步优化性能引起的问题

案例：

1) 相机：存在 400 处 void run()类型的异步回调类操作；

2) 输入法：子线程异步并发操作内核抢占 CPU 达到 300%，导致内核操作无返回。

由于锁的机制，加锁的代码块如果需要执行必须首先要申请到锁，所以就会出现等待，如果该等待发生在主线程就可能出现 anr。锁要谨慎使用，万不得已才使用，因为其不仅可能导致 anr 影响性能，而且可能使得多线程退化为单线程；对于加锁的代码块不能存在耗时操作，不然可能因为锁的联系导致主线程阻塞。

（1）高并发时，同步调用应该去考量锁的性能损耗。能用无锁数据结构，就不要用锁；能锁区块，就不要锁整个方法体；能用对象锁，就不要用类锁。

（2）在实现远程接口时，应尽量避免多把锁同时存在；如果有这样的情况，一定要谨慎处理，避免死锁。

（3）尽量不要在 Main 线程中参与抢锁，获取到锁后，建议完成必要的操作后，尽快将锁释放，把锁让出来给其它线程。

（4）如果用 wait()方法等待数据，应保证在一定时间能会返回数据。如果做不到这一点，建议在 wait()方法中加超时机制。

（5）所有 wait/notify/notifyall 均需要在 synchronized 或者 object 对象 lock 里面。

（6）备份对象，快速释放锁。

（7）避免同步锁嵌套，(全局列表)可以在同一线程串行执行。

（8）对持有锁的情况，避免进行延时处理。

（9）持锁后，避免频繁的或不必要的跨进程调用。

（10）被保护的数据对象不同，尽量用两把不同的锁。

（11）多线程并行处理定时任务时，Timer 运行多个 TimeTask 时，只要其中之一没有捕获抛出的异常，其它任务便会自动终止运行。

4.11. 使用 GLThread 资源防止锁死

•严重性：一般　　发生可能性：频繁

有一类是绘制资源 GL 线程锁死或耗时，导致 UI 线程锁死引发 ANR。一般在使用 OpenGL 资源的 apk 才会出现。GLThread 是有锁的，子线程持有主线程的锁在做耗时的操作造成 GLThread 一直 wait，比如执行不到 unfreeze 函数、死循环、上传文理、加载多个 so 库、destory 失败等情况。

子线程减少持有 GLThread 对象的时间，减少锁竞争，耗时需优化、如分段工作、移到其它线程等。

案例：

1) 相册：通过一个延迟的主线程消息来调用 unfreeze 的。相册在 onResume 中调用 freeze 后，GLThread 被置于 wait 状态，然后发送消息走完 onResume，界面开始刷新，主线程进入 wait 状态，而消息循环运行在主线程 mHandler 也就处于 wait 了，消息处理被迫停止，从而 unfreeze 方法一直得不到调用，最终导致 ANR。
2) 开心消消乐：在 GLThread 线程加载多个 so 库耗时，这些 so 库还很大，初始化过程中加载时间久会导致主线程 GLThread 锁 Object.wait。
3) 锁屏：Bitmap 加载及其纹理加载全在 OpenGL 线程中会耗时。
4) 状态栏：openGl 线程 destory 时异常卡住会导致主线程浮光掠影 onPause 时得不到 GL 线程锁 ANR。

4.12. 防呆资源损坏场景

•严重性：一般　　发生可能性：一般

一些场景下，资源(媒体、图片)损坏情况存在解析失败或者一直扫描耗时，资源被破坏无法解析引起 ANR。此种情况需要考虑防呆措施，如：增加超时退出机

制或增加类似异常 mp3 文件 meta 数据检查时间跳出循环。

案例：

1) i 音乐：/storage/emulated/目录下歌词和音频文件损坏，挂件获取到的路径，获取不到文件会一直去搜索一直扫描更新；

2) 锁屏：壁纸被破坏，libpng.so 解析一直卡住；

3) i 音乐： MP3 文件损坏，media 检测 sample rate 的时候死循环，导致 i 音乐调用 media 接口无响应。

4.13. timeout 防超时防止功能异常

•严重性：一般　　发生可能性：一般

一些情况下存在认证、网络连接、查询等操作，若一直执行不返回有可能触发 ANR，应用为了解决 ANR 崩溃设置 timeout 超时防止一直无响应；此处，需要考虑 timeout 时间限制是否影响一些功能异常。

案例：X7 扫音频二维码播放不了

多媒体之前为解决在线播放视频 ANR 问题，MediaHTTPConnection.java 创建网络连接的时候，设置了 4.5 秒超时。但部分服务器器响应慢，存在在线读取资源（距尾部 128 个字节处数据），超过 4.5s 被 timeout， 从而引起 X7 扫音频二维码播放不了问题。

4.14. static 语句块不要耗时

•严重性：一般　　发生可能性：一般

static 语句是在<clinit>方法中调用的，对一个 ClassLoader 来说，一个类的<clinit>方法只会执行一次，而且是加锁执行。所以 static 语句中后续第二个线程就算没阻塞，也不会输出了。因此，static 语句块中不要做耗时操作。

4.15. Log 输出要规范

•严重性：一般　　发生可能性：一般

根据 Android 平台 Log 输出规范 V1.0 要求，Verbose 和 Debug 等级的新增 Log，不允许在 user 版本中出现，使用宏统一控制。Log 输出的频率需要控制, 特别是 UI 线程，过多的 Log 将可能影响性能，重点关注循环中 log 数量。禁止使用 new Exception("print

trace").printStackTrace()或者 Log.getStackTraceString(Exception)方式打印普通调试信息，因为这种方式打印 Log 非常消耗系统资源。此种方式打印 Log 一般只出现 try..catch 某个异常使用。

案例：

上划：UI 线程绘制更新时 user 版本输出 log.d 加重性能影响，trace 最卡在 android.util.Log.d 关键字。

4.16. 系统连续出现多个应用 ANR

•严重性：致命　　发生可能性：一般

往往应用层缺乏这种辨别意识，需注意这种情况。系统连续出现多个应用 ANR，一般非上层应用问题，系统问题居多，比如：

•MTK aee 机制和 ams 一直在 dumptrace，造成比较大的 IO 压力；

•某个 common 接口异常，多个应用调用该接口出现 ANR；

•系统内存紧张、压力测试、dump、Low Ram 配置、Kswapd 非常活跃、gc 等复杂情况。

4.17. 测试调试环境合理

•严重性：一般　　发生可能性：频繁

内部测试以及调试首要一点关闭用户体验计划。压力测试应考虑合理，不能多个工具并行压力测试，如之前高通并行压力测试 Monkey king+ Memory leak + CPU/GPU/DDR stress 导致很严重的性能影响。

从大数据云诊断 log 中发现很多性能问题均是调试引起，如测试内存泄露 dumpHprof、抓取内存信息使用 adb shell dumpsys meminfo、压力测试 monkey 性能 loading 重、机器调试 demo 等情况。

关键词：

测试内存泄露 VMDebug.dumpHprofData

抓取内存 Debug.getMemoryInfo

monkey 压力测试 com.android.commands.monkey

调试 demo 类似 com.example.test 且 IMEI 号为内部号码：865407010000009

4.18. 云诊断分析方法

云诊断 ANR log 缺陷：

1) Trace 和 log 信息不足，难以分析
2) 聚类规则不准，需要人工分析，工作量大
3) 涉及模块多，难以协作和推动

针对云诊断 ANR log 分析没有好的办法，只能人工固定时间(比如每两周)进行一次大数据排除分析；对于缺少 system_server、mediaserver trace 的同一崩溃异常建议多查看一些 log 从中找出关联信息；对于跨进程的异常及时找到对端 owner 进行协作处理。

4.19. 其它需注意场景

•严重性：一般　　发生可能性：一般

1) gc 引起的性能卡顿导致 ANR

应用层常见 OOM 场景：对大图片未做大小限制处理；或者其它层出现内存泄露， memory 不足引起了 gc，gc 频繁容易引发 ANR。

2) 用户非法 root 行为导致 ANR

用户使用"刷机精灵"等刷机工具进行非法操作 root 导致一般表现为 root 进程的 snake 线程多且占比 CPU 高引发系统多个应用卡顿问题。特例用户非法操作，不作处理。

3) 主线程底层绘制 RenderThread 卡住

应用的主线程在绘制画面时，卡在了 ThreadedRenderer.draw()函数中导致发生 ANR。这类跟 GPU 性能有很大的关系。