之前两篇文章之后，打算再分享一点儿经验：之前文章见这里：
1， 全看懂了-加两年经验-语音朗读-语音识别-语音控制软件源码
2，学生作品-配置 NDK 集成开发环境全过程第一版

这次打算通过一个例子，深入解析一下 log 的分析方法以及 Handler 对象,Android 多线程及 MediaPlayer 状态分析 。
先在此占位，边写边发 。
特别注意，本文的内容全部是原创，经验所得 。特别是 Log 分析方法，网上搜了一下，没有 Log 的分析方法文章，特此贡献一下，希望对程序员有帮助。

 dumpstate_app_anr.rar (190.3 KB, 下载次数: 29)

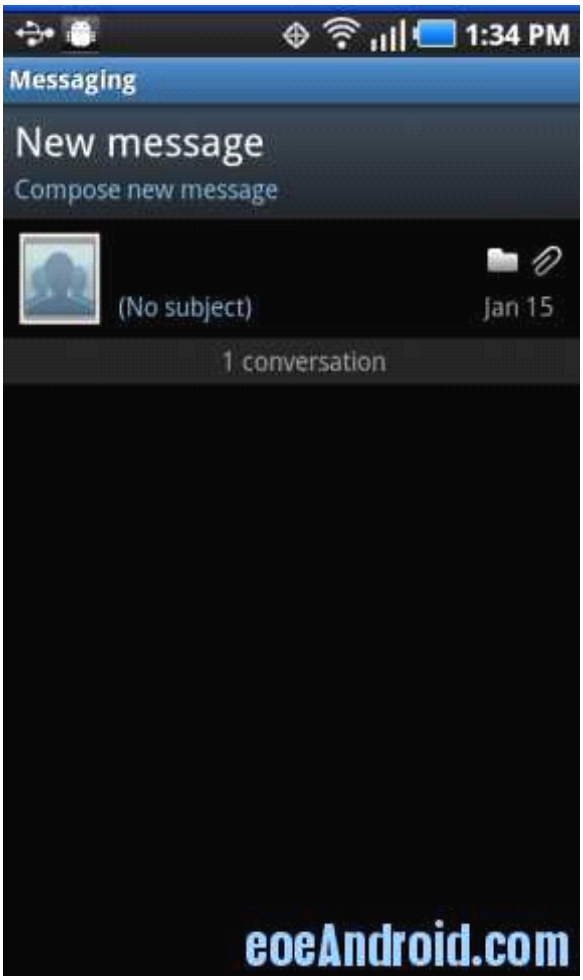
一，Bug 出现了， 需要“干掉”它 (update on Jan. 17)

今天在玩手机的时候发现自己的三星手机出现了 bug ， 是在 Message 模块的， 具体操作过程如下：
Idle -> Message -> New Message -> Attach -> Slideshow-> Attach Video -> preview Video-> Stress Press Test
Cause - > ANR (ForceClose)

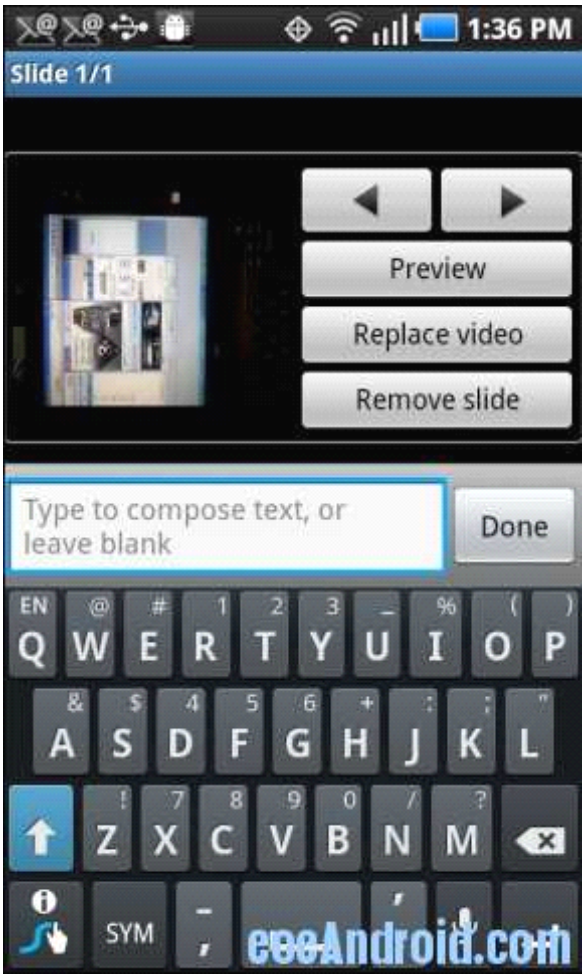
简单来说，就是建立一个幻灯作为短消息主题，然后附加内容为一段视频，预览这段视频，然后人工压力测试，就是狂点播放器的控制栏 。
出现了 ANR 无响应问题， 最后 ForceClose 关闭 。

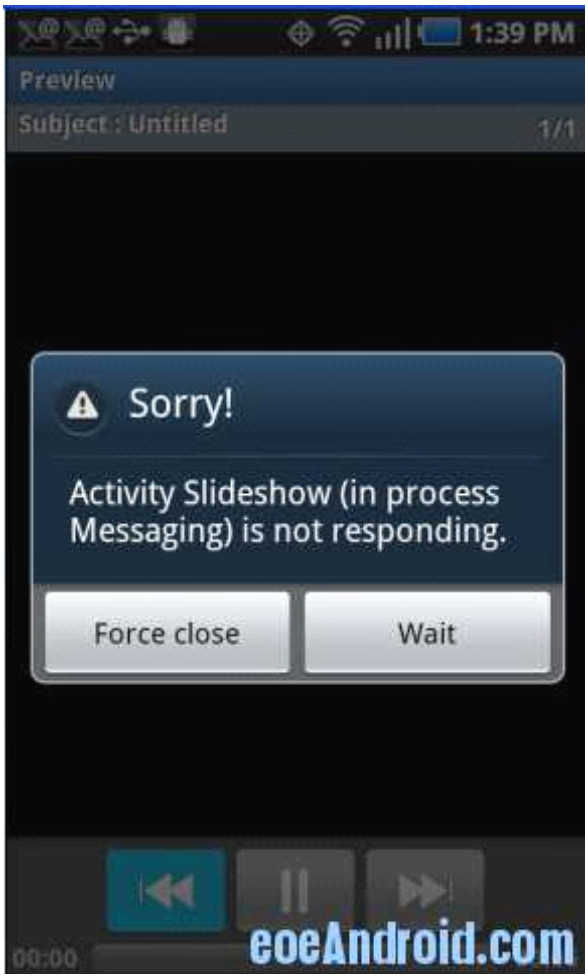
图片描述如下：


1:33 PM
ThinkFree Office
V CAST Music
V CAST Tones
V CAST Videos
Video Player
Voice Commands
Voice Recorder
Voice Search
Voicemail
VZ Navigator
Write & Go
Dev Tools
YouTube
ExtensionTest
Spare Parts
Messaging
Phone
eocAndroid.com


1:34 PM
Messaging
New message
Compose new message
(No subject)
Jan 15
1 conversation
eoeAndroid.com


1:34 PM
Attach
Pictures
Capture picture
Videos
Capture video
Audio
Record audio
Slideshow
eoeAndroid.com

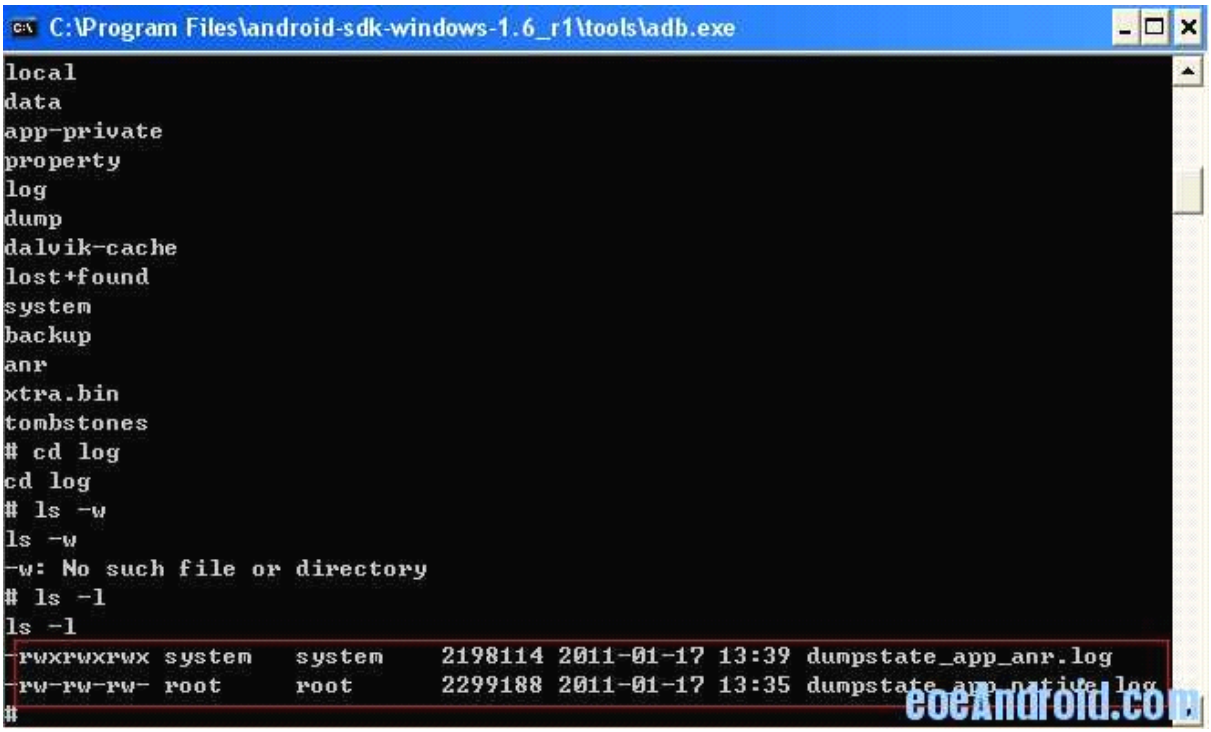

1:36 PM
Slide 1/1
Preview
Replace video
Remove slide
Type to compose text, or leave blank
Done
SYM
eeeAndroid.com



ANR 出现了 。 开始做修改准备工作 ， 得到 log 文件 。

有人问 log 文件在哪儿 ？

一般在/data/log 下面 。你可以通过执行命令 adb shell 进去看看 ， 如下是我的截图 。

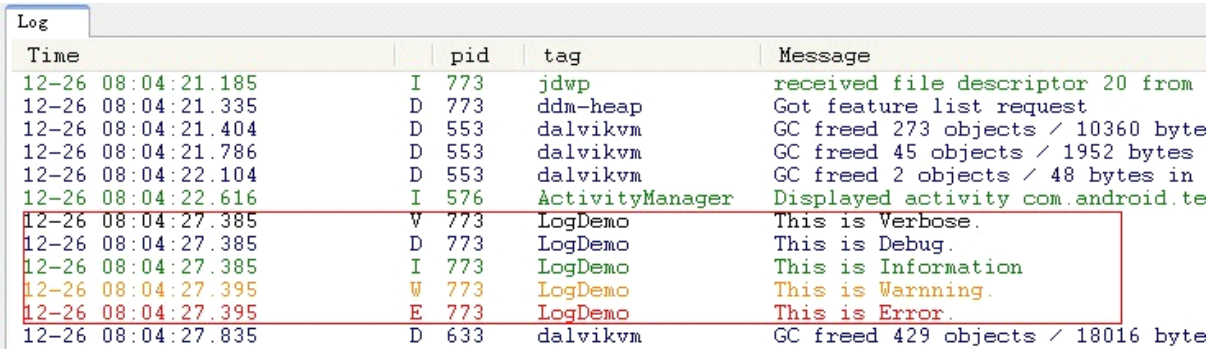


好，得到 log 文件了， 我们就准备开始工作了 。 我将 log 文件上传到附件供大家参考 。

二，如何分析和研究 Log 文件 ，如何看日志信息 。

Log 在 android 中的地位非常重要，要是作为一个 android 程序员不能过分析 log 这关，算是 android 没有入门吧 。 下面我们就来说说如何处理 log 文件 。

什么时候会有 Log 文件的产生 ？



Log 的产生大家都知道， 大家也都知道通过 DDMS 来看 log， 但什么时候会产生 log 文件呢 ？ 一般在如下几种情况会产生 log 文件 。

1，程序异常退出， uncaused exception

2，程序强制关闭， Force Closed (简称 FC)

3，程序无响应， Application No Response （简称 ANR)， 顺便，一般主线程超过 5 秒么有处理就会 ANR

4，手动生成 。

拿到一个日志文件，要分成多段来看 。 log 文件很长，其中包含十几个小单元信息，但不要被吓到 ，事实上他主要由三大块儿组成 。

1，系统基本信息 ，包括 内存，CPU ，进程队列 ，虚拟内存 ， 垃圾回收等信息 。------
MEMORY INFO (/proc/meminfo) ------
------ CPU INFO (top -n 1 -d 1 -m 30 -t) ------
------ PROCRANK (procrank) ------
------ VIRTUAL MEMORY STATS (/proc/vmstat) ------
------ VMALLOC INFO (/proc/vmallocinfo) ------

格式如下：

```
------ MEMORY INFO (/proc/meminfo) ------
MemTotal:         347076 kB
MemFree:           56408 kB
Buffers:            7192 kB
Cached:           104064 kB
SwapCached:            0 kB
Active:           192592 kB
Inactive:          40548 kB
Active(anon):     129040 kB
Inactive(anon):     1104 kB
Active(file):      63552 kB
Inactive(file):    39444 kB
Unevictable:        7112 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                44 kB
Writeback:             0 kB
AnonPages:        129028 kB
Mapped:            73728 kB
Shmem:              1148 kB
Slab:             13072 kB
SReclaimable:      4564 kB
SUnreclaim:        8508 kB
KernelStack:       3472 kB
PageTables:        12172 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:      173536 kB
Committed_AS:    7394524 kB
VmallocTotal:     319488 kB
```

VmallocUsed: 90752 kB
VmallocChunk: 181252 kB

2，事件信息， 也是我们主要分析的信息 。
------ VMALLOC INFO (/proc/vmallocinfo) ------
------ EVENT INFO (/proc/vmallocinfo) ------

格式如下：
------ SYSTEM LOG (logcat -b system -v time -d *:v) ------
01-15 16:41:43.671 W/PackageManager(2466): Unknown permission com.wsomacp.permission.PROVIDER in package com.android.mms
01-15 16:41:43.671 I/ActivityManager(2466): Force stopping package com.android.mms uid=10092
01-15 16:41:43.675 I/UsageStats(2466): Something wrong here, didn't expect com.sec.android.app.twlauncher to be paused
01-15 16:41:44.108 I/ActivityManager(2466): Start proc com.sec.android.widgetapp.infoalarm for service com.sec.android.widgetapp.infoalarm/.engine.DataService: pid=20634 uid=10005 gids={3003, 1015, 3002}
01-15 16:41:44.175 W/ActivityManager(2466): Activity pause timeout for HistoryRecord{48589868 com.sec.android.app.twlauncher/.Launcher}
01-15 16:41:50.864 I/KeyInputQueue(2466): Input event
01-15 16:41:50.866 D/KeyInputQueue(2466): screenCaptureKeyFlag setting 0
01-15 16:41:50.882 I/PowerManagerService(2466): Ulight 0->7|0
01-15 16:41:50.882 I/PowerManagerService(2466): Setting target 2: cur=0.0 target=70 delta=4.6666665 nominalCurrentValue=0
01-15 16:41:50.882 I/PowerManagerService(2466): Scheduling light animator!
01-15 16:41:51.706 D/PowerManagerService(2466): enableLightSensor true
01-15 16:41:51.929 I/KeyInputQueue(2466): Input event
01-15 16:41:51.933 W/WindowManager(2466): No focus window, dropping: KeyEvent{action=0 code=26 repeat=0 meta=0 scancode=26 mFlags=9}

3，虚拟机信息， 包括进程的，线程的跟踪信息，这是用来跟踪进程和线程具体点的好地方 。
------ VM TRACES JUST NOW (/data/anr/traces.txt.bugreport: 2011-01-15 16:49:02) ------
------ VM TRACES AT LAST ANR (/data/anr/traces.txt: 2011-01-15 16:49:02) ------

格式如下 ：
----- pid 21161 at 2011-01-15 16:49:01 -----
Cmd line: com.android.mms

DALVIK THREADS:

```
"main" prio=5 tid=1 NATIVE
  | group="main" sCount=1 dsCount=0 s=N obj=0x4001d8d0 self=0xccc8
  | sysTid=21161 nice=0 sched=0/0 cgrp=default handle=-1345017808
  | schedstat=( 4151552996 5342265329 10995 )
  at android.media.MediaPlayer._reset(Native Method)
  at android.media.MediaPlayer.reset(MediaPlayer.java:1218)
  at android.widget.VideoView.release(VideoView.java:499)
  at android.widget.VideoView.access$2100(VideoView.java:50)
  at android.widget.VideoView$6.surfaceDestroyed(VideoView.java:489)
  at android.view.SurfaceView.reportSurfaceDestroyed(SurfaceView.java:572)
  at android.view.SurfaceView.updateWindow(SurfaceView.java:476)
  at android.view.SurfaceView.onWindowVisibilityChanged(SurfaceView.java:206)
  at android.view.View.dispatchDetachedFromWindow(View.java:6082)
  at android.view.ViewGroup.dispatchDetachedFromWindow(ViewGroup.java:1156)
  at android.view.ViewGroup.removeAllViewsInLayout(ViewGroup.java:2296)
  at android.view.ViewGroup.removeAllViews(ViewGroup.java:2254)
  at com.android.mms.ui.SlideView.reset(SlideView.java:687)
  at com.android.mms.ui.SlideshowPresenter.presentSlide(SlideshowPresenter.java:189)
  at com.android.mms.ui.SlideshowPresenter$3.run(SlideshowPresenter.java:531)
  at android.os.Handler.handleCallback(Handler.java:587)
  at android.os.Handler.dispatchMessage(Handler.java:92)
  at android.os.Looper.loop(Looper.java:123)
  at android.app.ActivityThread.main(ActivityThread.java:4627)
  at java.lang.reflect.Method.invokeNative(Native Method)
  at java.lang.reflect.Method.invoke(Method.java:521)
  at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:858)
  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:616)
  at dalvik.system.NativeStart.main(Native Method)
```

闲话少说， 我总结了观察 log 文件的基本步骤 。 1，如果是 ANR 问题 ， 则搜索“ANR”关键词 。 快速定位到关键事件信息 。

2，如果是 ForceClosed 和其它异常退出信息，则搜索"Fatal" 关键词， 快速定位到关键事件信息 。

3，定位到关键事件信息后 ， 如果信息不够明确的，再去搜索应用程序包的虚拟机信息 ，查看具体的进程和线程跟踪的日志，来定位到代码 。

用这种方法，出现问题，根本不需要断点调试 ， 直接定位到问题，屡试不爽 。

下面，我们就开始来分析这个例子的 log 。

打开 log 文件 ， 由于是 ANR 错误，因此搜索"ANR " ， 为何要加空格呢，你加上和去掉比较一下就知道了 。 可以屏蔽掉不少保存到 anr.log 文件的无效信息 。

定位到关键的事件信息如下：

01-15 16:49:02.433 E/ActivityManager(2466): ANR in com.android.mms (com.android.mms/.ui.SlideshowActivity)
01-15 16:49:02.433 E/ActivityManager(2466): Reason: keyDispatchingTimedOut
01-15 16:49:02.433 E/ActivityManager(2466): Load: 0.6 / 0.61 / 0.42
01-15 16:49:02.433 E/ActivityManager(2466): CPU usage from 1337225ms to 57ms ago:
01-15 16:49:02.433 E/ActivityManager(2466): sensorserver_ya: 8% = 0% user + 8% kernel / faults: 40 minor
.....

01-15 16:49:02.433 E/ActivityManager(2466): -com.android.mms: 0% = 0% user + 0% kernel
01-15 16:49:02.433 E/ActivityManager(2466): -flush-179:8: 0% = 0% user + 0% kernel
01-15 16:49:02.433 E/ActivityManager(2466): TOTAL: 25% = 10% user + 14% kernel + 0% iowait + 0% irq + 0% softirq
01-15 16:49:02.436 I/   (2466): dumpmesg > "/data/log/dumpstate_app_anr.log"

我们用自然语言来描述一下日志，这也算是一种能力吧 。
01-15 16:49:02.433 E/ActivityManager(2466): ANR in com.android.mms (com.android.mms/.ui.SlideshowActivity)
翻译：在 16:49 分 2 秒 433 毫秒的时候 ActivityManager （进程号为 2466) 发生了如下错误：com.android.mms 包下面的.ui.SlideshowActivity 无响应 。

01-15 16:49:02.433 E/ActivityManager(2466): Reason: keyDispatchingTimedOut
翻译：原因， keyDispatchingTimeOut - 按键分配超时

01-15 16:49:02.433 E/ActivityManager(2466): Load: 0.6 / 0.61 / 0.42
翻译：5 分钟，10 分钟，15 分钟内的平均负载分别为：0.6 , 0.61 , 0.42

在这里我们大概知道问题是什么了，结合我们之前的操作流程，我们知道问题是在点击按钮某时候可能处理不过来按钮事件，导致超时无响应 。那么现在似乎已经可以进行工作了 。 我们知道 Activity 中是通过重载 dispatchTouchEvent(MotionEvent ev)来处理点击屏幕事件 。 然后我们可以顺藤摸瓜，一点点分析去查找原因 。 但这样够了么 ？
其实不够， 至少我们不能准确的知道到底问题在哪儿， 只是猜测 ，比如这个应用程序中，我就在顺藤摸瓜的时候发现了多个 IO 操作的地方都在主线程中，可能引起问题，但不好判断到底是哪个 ，所以我们目前掌握的信息还不够 。

于是我们再分析虚拟机信息， 搜索“Dalvik Thread”关键词，快速定位到本应用程序的虚拟机信息日志，如下：
----- pid 2922 at 2011-01-13 13:51:07 -----
Cmd line: com.android.mms

DALVIK THREADS:
"main" prio=5 tid=1 NATIVE

```
  | group="main" sCount=1 dsCount=0 s=N obj=0x4001d8d0 self=0xccc8
  | sysTid=2922 nice=0 sched=0/0 cgrp=default handle=-1345017808
  | schedstat=( 3497492306 15312897923 10358 )
  at android.media.MediaPlayer._release(Native Method)
  at android.media.MediaPlayer.release(MediaPlayer.java:1206)
  at android.widget.VideoView.stopPlayback(VideoView.java:196)
  at com.android.mms.ui.SlideView.stopVideo(SlideView.java:640)
  at com.android.mms.ui.SlideshowPresenter.presentVideo(SlideshowPresenter.java:443)
  at
com.android.mms.ui.SlideshowPresenter.presentRegionMedia(SlideshowPresenter.java:219)

  at com.android.mms.ui.SlideshowPresenter$4.run(SlideshowPresenter.java:516)
  at android.os.Handler.handleCallback(Handler.java:587)
  at android.os.Handler.dispatchMessage(Handler.java:92)
  at android.os.Looper.loop(Looper.java:123)
  at android.app.ActivityThread.main(ActivityThread.java:4627)
  at java.lang.reflect.Method.invokeNative(Native Method)
  at java.lang.reflect.Method.invoke(Method.java:521)
  at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:858)
  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:616)
  at dalvik.system.NativeStart.main(Native Method)

"Binder Thread #3" prio=5 tid=11 NATIVE
  | group="main" sCount=1 dsCount=0 s=N obj=0x4837f808 self=0x242280
  | sysTid=3239 nice=0 sched=0/0 cgrp=default handle=2341032
  | schedstat=( 32410506 932842514 164 )
  at dalvik.system.NativeStart.run(Native Method)

"AsyncQueryWorker" prio=5 tid=9 WAIT
  | group="main" sCount=1 dsCount=0 s=N obj=0x482f4b80 self=0x253e10
  | sysTid=3236 nice=0 sched=0/0 cgrp=default handle=2432120
  | schedstat=( 3225061 26561350 27 )
  at java.lang.Object.wait(Native Method)
  - waiting on <0x482f4da8> (a android.os.MessageQueue)
  at java.lang.Object.wait(Object.java:288)
  at android.os.MessageQueue.next(MessageQueue.java:146)
  at android.os.Looper.loop(Looper.java:110)
  at android.os.HandlerThread.run(HandlerThread.java:60)

"Thread-9" prio=5 tid=8 WAIT
  | group="main" sCount=1 dsCount=0 s=N obj=0x4836e2b0 self=0x25af70
  | sysTid=2929 nice=0 sched=0/0 cgrp=default handle=2370896
  | schedstat=( 130248 4389035 2 )
  at java.lang.Object.wait(Native Method)
  - waiting on <0x4836e240> (a java.util.ArrayList)
  at java.lang.Object.wait(Object.java:288)
  at com.android.mms.data.Contact$ContactsCache$TaskStack$1.run(Contact.java:488)
```

```
  at java.lang.Thread.run(Thread.java:1096)

"Binder Thread #2" prio=5 tid=7 NATIVE
  | group="main" sCount=1 dsCount=0 s=N obj=0x482f8ca0 self=0x130fd0
  | sysTid=2928 nice=0 sched=0/0 cgrp=default handle=1215968
  | schedstat=( 40610049 1837703846 195 )
  at dalvik.system.NativeStart.run(Native Method)

"Binder Thread #1" prio=5 tid=6 NATIVE
  | group="main" sCount=1 dsCount=0 s=N obj=0x482f4a78 self=0x128a50
  | sysTid=2927 nice=0 sched=0/0 cgrp=default handle=1201352
  | schedstat=( 40928066 928867585 190 )
  at dalvik.system.NativeStart.run(Native Method)

"Compiler" daemon prio=5 tid=5 VMWAIT
  | group="system" sCount=1 dsCount=0 s=N obj=0x482f1348 self=0x118960
  | sysTid=2926 nice=0 sched=0/0 cgrp=default handle=1149216
  | schedstat=( 753021350 3774113668 6686 )
  at dalvik.system.NativeStart.run(Native Method)

"JDWP" daemon prio=5 tid=4 VMWAIT
  | group="system" sCount=1 dsCount=0 s=N obj=0x482f12a0 self=0x132940
  | sysTid=2925 nice=0 sched=0/0 cgrp=default handle=1255680
  | schedstat=( 2827103 29553323 19 )
  at dalvik.system.NativeStart.run(Native Method)

"Signal Catcher" daemon prio=5 tid=3 RUNNABLE
  | group="system" sCount=0 dsCount=0 s=N obj=0x482f11e8 self=0x135988
  | sysTid=2924 nice=0 sched=0/0 cgrp=default handle=1173688
  | schedstat=( 11793815 12456169 7 )
  at dalvik.system.NativeStart.run(Native Method)

"HeapWorker" daemon prio=5 tid=2 VMWAIT
  | group="system" sCount=1 dsCount=0 s=N obj=0x45496028 self=0x135848
  | sysTid=2923 nice=0 sched=0/0 cgrp=default handle=1222608
  | schedstat=( 79049792 1520840200 95 )
  at dalvik.system.NativeStart.run(Native Method)

----- end 2922 -----
```

每一段都是一个线程， 当然我们还是看线程号为 1 的主线程了。通过分析发现关键问题是这样：

```
  at com.android.mms.ui.SlideshowPresenter$3.run(SlideshowPresenter.java:531)
```

定位到代码：

```
mHandler.post(new Runnable() {
                public void run() {
                    try {
```

```
            presentRegionMedia(view, (RegionMediaModel) model, dataChanged);
        } catch (OMADRMException e) {
            Log.e(TAG, e.getMessage(), e);
            Toast.makeText(mContext,
                    mContext.getString(R.string.insufficient_drm_rights),
                    Toast.LENGTH_SHORT).show();
        } catch (IOException e){
            Log.e(TAG, e.getMessage(), e);
            Toast.makeText(mContext,
                    mContext.getString(R.string.insufficient_drm_rights),
                    Toast.LENGTH_SHORT).show();


        }
    }
```

很清楚了， Handler.post 方法之后执行时间太长的问题 。 继续看 presentRegionMedia(view, (RegionMediaModel) model, dataChanged);方法， 发现最终是调用的 framework 中 MediaPlayer.stop 方法 。
至此，我们的日志分析算是告一段落 。 可以开始思考解决办法了 。

三，如何通过 Handler 或者多线程来解决某操作执行时间过程的问题 。 (update on Jan.19)

结合上面的分析，我们知道问题似乎是线程队列中某个操作 presentRegionMedia(view, (RegionMediaModel) model, dataChanged);执行时间太长所导致的界面无响应 。 因此比较典型的做法当然是控制线程队列 。 在这里我们不得不提一下 Handler .

Handler 在 Android 中是什么样的作用和地位呢？

1， 线程之间消息传递， 通过 sendMessage 方法 。 我们通常用来后台子线程向主线程传递消息，主线程接到通知之后做更新界面等操作 。

2， 通过管理消息队列(MessageQueue)来安排计划任务 。 这个常常会被人忽略，很多书上也没有提到这个作用 。

Handler 这个单词中文意思是管理者，处理者的意思 。 通过这个意思顾名思义，我们知道这个对象就是个操作对象。那么要操作谁呢 ？

当然是消息队列（MessageQueue） 。Android 消息队列类似于 Win32 队列设计 。 都是采用线性结构，先进先出 。 其实在智能手机平台很久以前就用这种消息结构了 。 比如 Palm， 只不过 Palm 是整个进程共享一个消息队列，而 Android 是线程为单位的队列罢了 。

那么是否每个线程或者子线程都有消息队列呢 ？

很遗憾，不是的，也没有必要 。 在 Android 中，只有使用了 Looper 的线程才有消息队列 。 当然如果你要简单建立一个有消息队列的线程也很方便，直接使用 HandlerThread 即可，这个类继承于 Thread 类 。怎么用我就不多说了吧 。你懂的 ！

Handler 有两种方式来操作消息队列 。

一种是通过 sendMessage(Message)方法 ， 发送消息体

另一种是通过 post(Runnable) 方法 ， 发送 Runnable 对象 。

注意：这点请注意 ， 虽然发送方法含参不同 ， 但他们使用的是同一个消息队列 。 我记得 Mars 的视频教程上说有两个队列，一个是消息队列，一个是线程队列 。 这种说法是错误的 。事实上只有一个消息队列，没有所谓的线程队列 。 当然了 ， post(Runnable)也没有启动新的线程，仍然是在当前线程 。

注意：还有一种说法 ， 说 Handler 对象在主线程，这种说法也是错误的 ， 准确的说是在产生他的线程中 。 虽然常常我们是在主线程产生他的 。

那么我们要在 Android 建立多线程程序该如何做呢？很简单，就是 Java 的多线程方式。要么实现 Runnable 接口，要么继承 Thread 类 。

关于线程同步，线程锁定，线程异步 ，线程池 这些概念也是一样的 。 我就不累述了。

好了，经过一点儿简单的介绍，我们有了一些 Handler 的基础，现在开始回到我们的问题开始来分析 ：

```
mHandler.post(new Runnable() {
            public void run() {
               try {
                  presentRegionMedia(view, (RegionMediaModel) model, dataChanged);
               } catch (OMADRMException e) {
                  Log.e(TAG, e.getMessage(), e);
                  Toast.makeText(mContext,
                        mContext.getString(R.string.insufficient_drm_rights),
                        Toast.LENGTH_SHORT).show();
               } catch (IOException e){
                  Log.e(TAG, e.getMessage(), e);
                  Toast.makeText(mContext,
                        mContext.getString(R.string.insufficient_drm_rights),
                        Toast.LENGTH_SHORT).show();
```

```
        }
    }
```

从上面这段代码中，我们可以看出，在做播放器控制按钮（比如播放，暂停，停止）等操作的时候， 是通过 Handler.post(Runnable)来放到消息队列中， 排序来处理 。 那么之所以这里出现了无响应，很有可能是因为某一项控制操作太耗时或者耗资源 。 这时候又接收到新的要处理的消息，就会处理不过来了 。 因此我试图让队列中同时只有一个控制播放器按钮的任务在 。 我对代码做了如下改动：

```
Runnable r = new Runnable(){
        public void run() {
            try {
                presentRegionMedia(view, (RegionMediaModel) model, dataChanged);
            } catch (OMADRMException e) {
                Log.e(TAG, e.getMessage(), e);
                Toast.makeText(mContext,
                        mContext.getString(R.string.insufficient_drm_rights),
                        Toast.LENGTH_SHORT).show();
            } catch (IOException e){
                Log.e(TAG, e.getMessage(), e);
                Toast.makeText(mContext,
                        mContext.getString(R.string.insufficient_drm_rights),
                        Toast.LENGTH_SHORT).show();


            }
        }
```

```
mHandler.removeCallbacks(r) ;

mHandler.post（r） ;
```

代码慢慢看，思路很简单 ：其实就是在 post Runnable 之前先清除队列中已存的相同 Runnable 实例 。 这样可以保证同时队列中只有一个操作在处理 。

很遗憾，不生效 。： （ ，改动之后，问题依然存在 ，欲哭无泪 。

再来，我将整个模式改为 message 再试试 ， 核心代码如下 ：

```
if(mHandler.hasMessages(MEDIA_PLAY_WHAT_MESSAGEFLAG))
```

```
                {
                    return  ;
                }
                Message msg =
mHandler.obtainMessage()  ;
                msg.what =
this.MEDIA_PLAY_WHAT_MESSAGEFLAG  ;
                msg.obj = mMeidaPlayMessageObj  ;
                mHandler.sendMessageDelayed(msg,
1000)  ;
```

代码慢慢看，思路也很简单，通过发消息的方式， 先检测如果有相关消息队列，就直接跳出函数，不做任何处理， 否则延迟一秒后再向队列发送一条消息 。

为何我用了 1 秒这个这么长的时间呢，因为这么长时间如果都处理不了，那就不是压力测试的问题了，而是方法本身的问题 了，这也是通过排除法来试图排除是因为点击屏幕过快产生的问题 。

编译，再试， 很不辛，又不生效，不幸被我猜中了 。 仰望苍天 ！

现在问题很明显了 ：不是压力测试时候点击过快导致的 ANR，而是某些方法本身有问题。

通过之前我们的日志

```
----- pid 2922 at 2011-01-13 13:51:07 -----
Cmd line: com.android.mms

DALVIK THREADS:
"main" prio=5 tid=1 NATIVE
  | group="main" sCount=1 dsCount=0 s=N obj=0x4001d8d0 self=0xccc8
  | sysTid=2922 nice=0 sched=0/0 cgrp=default handle=-1345017808
  | schedstat=( 3497492306 15312897923 10358 )
  at android.media.MediaPlayer._release(Native Method)
  at android.media.MediaPlayer.release(MediaPlayer.java:1206)
  at android.widget.VideoView.stopPlayback(VideoView.java:196)
  at com.android.mms.ui.SlideView.stopVideo(SlideView.java:640)
```

很容易就知道了问题出在每次执行完了 MediaPlayer.stop()方法调用之后会调用 release()来释放播放器资源 。 而这个方法中又死在了_release()方法上 。 这是一个 Native 方法 。

因此，真相大白 ，问题是在 Framework 层的 MediaPlayer 调用的 Native 方法_release()上 。

（打的手酸了，休息一下，大家对下面的内容还感兴趣么？）

四，我们遇到了新问题

五，研究 Framework 代码， 看看 MediaPlayer 到底是个什么 ， 以及是否线程安全的 。

六，终于解决了！收工 。