**Android crash 培訓**

1. java 部分的崩潰：

exception代碼的位置：libcore/luni

Java exception的框架圖



**Error問題：**

系統虛擬機的問題，是程序无法处理的错误。

示例：

Hiau的CITS#51 DUT can not Power off. It will boot-up to home always

Log 分析：

system\_server发生了crash导致shutdown的线程关闭失败，同時系統重新启动system\_server，导致关机失败进入home界面。

分析system\_server发生了crash 原因：

發現在system\_server发生crash 之前發生了大量的OutOfMemoryError，

java.lang.OutOfMemoryError: Failed to allocate a 14745612 byte allocation with 4194208 free bytes and 13MB until OOM

通過分析發現根本原因：

Shutdown Animation 一張圖要快 14 MB (1440x2560)，需要 21 張 (14\*21 = 294 MB)，所以Heap size 256 Mb 會不夠用.

解決方案：減小關機動畫圖片的大小、擴大虛擬機堆的大小。

Android 虛擬機堆大小查看和修改。

/system/ build.prop

dalvik.vm.heapsize= 512 m

**異常問題：**

是程序本身可以处理的异常。

**自定義異常：**

Extend Exception。

NameNotFoundException （PackageManager）

RemoteException

示例：

A55ML\_DTUL\_L50\_SENSE70 CITS#692：Device got Exception issue (com.htc.music) during running SST

分析：

Log：

03-27 20:27:41.426 13114 14642 E AndroidRuntime: FATAL EXCEPTION: RSMessageThread  
03-27 20:27:41.426 13114 14642 E AndroidRuntime: Process: com.htc.music, PID: 13114  
03-27 20:27:41.426 13114 14642 E AndroidRuntime: android.renderscript.RSDriverException: Error processing message from RenderScript.  
03-27 20:27:41.426 13114 14642 E AndroidRuntime: at android.renderscript.RenderScript$MessageThread.run(RenderScript.java:1177)

Code：

if (mRS.nContextGetUserMessage(mRS.mContext, rbuf) !=  
RS\_MESSAGE\_TO\_CLIENT\_NEW\_BUFFER) {  
throw new RSDriverException("Error processing message from RenderScript.");  
continue;  
}

解決方案：

將Exception用Log.e替換。

**RuntimeException：**

不捕捉、也不声明抛出运行时异常。

ArithmeticException：算术错误，典型的就是0作为除数的时候。

ClassNotFoundException： load class不存在。

NullPointerException： 對象沒有分配堆內存，調用方法。

IllegalArgumentException：非法参数。

ArrayIndexOutOfBoundsException: 数组越界

示例：

A53ML\_DTUL\_L50\_SENSE70 CITS#137: com.android.systemui Java Crash

Log:

03-09 13:24:49.713 31771 31771 E AndroidRuntime: Caused by: java.lang.ClassNotFoundException: Didn't find class "com.android.systemui.SystemUIApplication" on path: DexPathList[[zip file "/system/priv-app/SystemUI/SystemUI.apk"],nativeLibraryDirectories=[/vendor/lib64, /system/lib64]]

03-09 13:24:49.713 31771 31771 E AndroidRuntime: at dalvik.system.BaseDexClassLoader.findClass(BaseDexClassLoader.java:56)

03-09 13:24:49.713 31771 31771 E AndroidRuntime: at java.lang.ClassLoader.loadClass(ClassLoader.java:511)

03-09 13:24:49.713 31771 31771 E AndroidRuntime: at java.lang.ClassLoader.loadClass(ClassLoader.java:469)

03-09 13:24:49.713 31771 31771 E AndroidRuntime: at android.app.Instrumentation.newApplication(Instrumentation.java:986)

03-09 13:24:49.713 31771 31771 E AndroidRuntime: at android.app.LoadedApk.makeApplication(LoadedApk.java:566)

分析：

java.lang.ClassNotFoundException 一般是load的class不存在導致的，通過代碼發現load的class是存在的。

03-09 13:24:49.713 31771 31771 E AndroidRuntime: Suppressed: java.io.IOException: Failed to open oat file from dex location '/system/priv-app/SystemUI/SystemUI.apk'  
03-09 13:24:49.713 31771 31771 E AndroidRuntime: at dalvik.system.DexFile.openDexFileNative(Native Method)  
03-09 13:24:49.713 31771 31771 E AndroidRuntime: at dalvik.system.DexFile.openDexFile(DexFile.java:295)  
03-09 13:24:49.713 31771 31771 E AndroidRuntime: at dalvik.system.DexFile.(DexFile.java:80)  
03-09 13:24:49.713 31771 31771 E AndroidRuntime: at dalvik.system.DexFile.(DexFile.java:59)

通過log發現open oat file 失敗了。該問題轉給了ssd。最後他們得出的結論是屬於dex file corruption issue。

**異常拋出：**

1. throws

格式：

语法：[(修饰符)](返回值类型)(方法名)([参数列表])[throws(异常类)]{......}

public void doA(int a) throws Exception1,Exception3{......}

表明該方法中以及其調用的函數中可能會拋出這些異常。但是該方法中不一定處理這些異常。假如拋出異常沒有處理，那麼會將該異常拋到上一級調用的函數中，直到異常被處理。

作用：

Client調用接口的實現代碼可能是jar庫，導致client端不能看到庫中會拋出哪些異常，這個時候就需要異常說明。

2.throw 拋出異常。

throw总是出现在函数体中，用来抛出一个Throwable类型的异常。

**異常的補足：**

補足的格式：

Try {

可能拋出a、b異常。

} catch(Exception a)

{

} catch(Exception b)

{

} finally {

}

使用的注意點：

1. 子類異常的catch放在前面，父類可以補足子類異常
2. 不管在try還是在catch中return finally block都會執行。

**異常的使用的一些小技巧：**

Log.i("callstack ", " callstack = " ,new Throwable("function"));

Logcat grep出callstack堆棧

看代碼時查看調用的堆棧