Android NDK开发Crash错误定位

原创 2015年01月01日 19:50:44

标签：NDK调试 (http://so.csdn.net/so/search/s.do?q=NDK调试&t=blog) / ndk-stack (http://so.csdn.net/so/search/s.do?q=ndk-stack&t=blog) / objdump (http://so.csdn.net/so/search/s.do?q=objdump&t=blog) / addr2line (http://so.csdn.net/so/search/s.do?q=addr2line&t=blog) / NDK错误定位 (http://so.csdn.net/so/search/s.do?q=NDK错误定位&t=blog)

在Android开发中，程序Crash分三种情况：未捕获的异常、ANR（Application Not Responding）和闪退（NDK引发错误）。其中未捕获的异常根据logcat打印的堆栈信息很容易定位错误。ANR错误也好查，Android规定，应用与用户进行交互时，如果5秒内没有响应用户的操作，则会引发ANR错误，并弹出一个系统提示框，让用户选择继续等待或立即关闭程序。并会在/data/anr目录下生成一个traces.txt文件，记录系统产生anr异常的堆栈和线程信息。如果是闪退，这问题比较难查，通常是项目中用到了NDK引发某类致命的错误导致闪退。因为NDK是使用C/C++来进行开发，熟悉C/C++的程序员都知道，指针和内存管理是最重要也是最容易出问题的地方，稍有不慎就会遇到诸如内存地址访问错误、使用野针对、内存泄露、堆栈溢出、初始化错误、类型转换错误、数字除0等常见的问题，导致最后都是同一个结果：程序崩溃。不会像在Java层产生的异常时弹出“xxx程序无响应，是否立即关闭”之类的提示框。当发生NDK错误后，logcat打印出来的那堆日志根据看不懂，更别想从日志当中定位错误的根源，让我时常有点抓狂，火冒三丈，喝多少加多宝都不管用。当时尝试过在各个jni函数中打印日志来跟踪问题，那效率实在是太低了，而且还定位不到问题。还好老天有眼，让我找到了NDK提供的几款调试工具，能够精确的定位到产生错误的根源。

NDK安装包中提供了三个调试工具：addr2line、objdump和ndk-stack，其中ndk-stack放在$NDK_HOME目录下，与ndk-build同级目录。addr2line和objdump在ndk的交叉编译器工具链目录下，下面是我本机NDK交叉编译器工具链的目录结构：

```
yangxin-MacBook-Pro:toolchains yangxin$ pwd
/Users/yangxin/Documents/devToos/java/android-ndk-r9d/toolchains
yangxin-MacBook-Pro:toolchains yangxin$ ls
arm-linux-androideabi-4.6      llvm-3.3                       mipsel-linux-android-clang3.3  x86-4.8
arm-linux-androideabi-4.8      llvm-3.4                       mipsel-linux-android-clang3.4  x86-clang3.3
arm-linux-androideabi-clang3.3 mipsel-linux-android-4.6       renderscript                   x86-clang3.4
arm-linux-androideabi-clang3.4 mipsel-linux-android-4.8       x86-4.6
yangxin-MacBook-Pro:toolchains yangxin$ _
```


从上图的目录结构中可以看出来，NDK针对不同的CPU架构实现了多套相同的工具。所以在选择addr2line和objdump工具的时候，要根据你目标机器的CPU架构来选择。如果是arm架构，选择arm-linux-androidabi-4.6/4.8（一般选择高版本）。x86架构，选择x86-4.6/4.8。mipsel架构，选择mipsel-linux-android-4.6/4.8。如果不知道目标机器的CPU架

构，把手机连上电脑，用adb shell cat /proc/cpuinfo可以查看手机的CPU信息。下图是我本机的arm架构工具链目录结构：

```
yangxin-MacBook-Pro:bin yangxin$ pwd
/Users/yangxin/Documents/devToos/java/android-ndk-r9d/toolchains/arm-linux-androideabi-4.8/prebuilt/darwin-x86_64/bin
yangxin-MacBook-Pro:bin yangxin$ ls
arm-linux-androideabi-addr2line  arm-linux-androideabi-gcc-ar     arm-linux-androideabi-nm
arm-linux-androideabi-ar         arm-linux-androideabi-gcc-nm     arm-linux-androideabi-objcopy
arm-linux-androideabi-as         arm-linux-androideabi-gcc-ranlib arm-linux-androideabi-objdump
arm-linux-androideabi-c++        arm-linux-androideabi-gcov       arm-linux-androideabi-ranlib
arm-linux-androideabi-c++filt    arm-linux-androideabi-gdb        arm-linux-androideabi-readelf
arm-linux-androideabi-cpp        arm-linux-androideabi-gprof      arm-linux-androideabi-size
arm-linux-androideabi-elfedit    arm-linux-androideabi-ld         arm-linux-androideabi-strings
arm-linux-androideabi-g++        arm-linux-androideabi-ld.bfd     arm-linux-androideabi-strip
arm-linux-androideabi-gcc        arm-linux-androideabi-ld.gold
arm-linux-androideabi-gcc-4.8    arm-linux-androideabi-ld.mcld
yangxin-MacBook-Pro:bin yangxin$
```

下面通过NDK自带的例子hello-jni项目来演示一下如何精确的定位错误

```cpp
#include <string.h>
#include <jni.h>
// hell-jni.c
#ifdef __cplusplus
extern "C" {
#endif
    void willCrash()
    {
        int i = 10;
        int y = i / 0;
    }

    JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM* vm, void* reserved)
    {
        willCrash();
        return JNI_VERSION_1_4;
    }

    jstring
    Java_com_example_hellojni_HelloJni_stringFromJNI( JNIEnv* env,
                                                      jobject thiz )
    {
    // 此处省略实现逻辑。。。
    }

#ifdef __cplusplus
}
#endif
```

第7行定义了一个willCrash函数，函数中有一个除0的非法操作，会造成程序崩溃。第13行JNI_OnLoad函数中调用了willCrash，这个函数会在Java加载完.so文件之后回调，也就是说程序一启动就会崩溃。下面是运行程序后打印的log：

```cpp
01-01 17:59:38.246: D/dalvikvm(20794): Late-enabling CheckJNI
01-01 17:59:38.246: I/ActivityManager(1185):
Start proc com.example.hellojni for activity com.example.hellojni/.HelloJni: pid=20794 uid=1035
01-01 17:59:38.296: I/dalvikvm(20794): Enabling JNI app bug workarounds for target SDK version
01-01 17:59:38.366: D/dalvikvm(20794): Trying to load lib /data/app-lib/com.example.hellojni-
1/libhello-jni.so 0x422a4f58
01-01 17:59:38.366: D/dalvikvm(20794): Added shared lib /data/app-lib/com.example.hellojni-
1/libhello-jni.so 0x422a4f58
01-01 17:59:38.366: A/libc(20794): Fatal signal 8 (SIGFPE) at 0x0000513a (code=-6), thread 207
01-01 17:59:38.476: I/DEBUG(253): pid: 20794, tid: 20794, name: xample.hellojni  >>> com.exampl
01-01 17:59:38.476: I/DEBUG(253): signal 8 (SIGFPE), code -6 (SI_TKILL), fault addr 0000513a
01-01 17:59:38.586: I/DEBUG(253):     r0 00000000  r1 0000513a  r2 00000008  r3 00000000
01-01 17:59:38.586: I/DEBUG(253):     r4 00000008  r5 0000000d  r6 0000513a  r7 0000010c
01-01 17:59:38.586: I/DEBUG(253):     r8 75226d08  r9 00000000  sl 417c5c38  fp bedbf134
01-01 17:59:38.586: I/DEBUG(253):     ip 41705910  sp bedbf0f0  lr 4012e169  pc 4013d10c  cpsr
```


JavaEE (http://blog.csdn.net/x... 11篇

展开

博主专栏

JNI/NDK开发指南 (http://blog.csdn.net/column/details/blogjnindk.html)

14

179130

(http://blog.csdn.net/column/details/blogjnindk.html)

文章存档

2016年11月 (http://blog.csdn.... 4篇

2016年10月 (http://blog.csdn.... 5篇

2016年9月 (http://blog.csdn.n... 5篇

2016年8月 (http://blog.csdn.n... 1篇

2016年7月 (http://blog.csdn.n... 10篇

展开

他的热门文章

深入分析Java ClassLoader原理 (http://blog.csdn.net/xyang81/article/details/7292380)

108594

Supervisor安装与配置（Linux/Unix进程管理工具） (http://blog.csdn.net/xyang81/article/details/51555473)

38200

Keepalived安装与配置 (http://blog.csdn.net/xyang81/article/details/52554398)

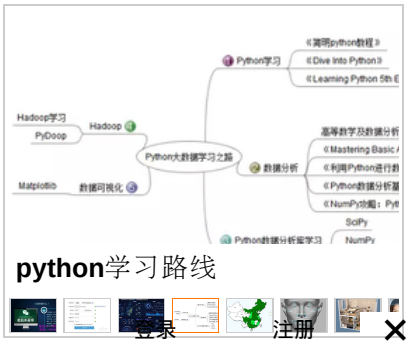
35607

开启Tomcat APR运行模式，优化并发性能 (http://blog.csdn.net/xyang81/article/details/51502766)

28434

Android NDK开发Crash错误定位 (http://blog.csdn.net/xyang81/article/details/42319789)

26357



```
                                        // 省略部份日志 。。。。。。
01-01 17:59:38.596: I/DEBUG(253): backtrace:
01-01 17:59:38.596: I/DEBUG(253):     #00  pc 0002210c  /system/lib/libc.so (tgkill+12)
01-01 17:59:38.596: I/DEBUG(253):     #01  pc 00013165  /system
/lib/libc.so (pthread_kill+48)
01-01 17:59:38.596: I/DEBUG(253):     #02  pc 00013379  /system/lib/libc.so (raise+10)
01-01 17:59:38.596: I/DEBUG(253):     #03  pc 00000e80  /data/app-lib/com.example.hellojni-
1/libhello-jni.so (__aeabi_idiv0+8)
01-01 17:59:38.596: I/DEBUG(253):     #04  pc 00000cf4  /data/app-lib/com.example.hellojni-
1/libhello-jni.so (willCrash+32)
01-01 17:59:38.596: I/DEBUG(253):     #05  pc 00000d1c  /data/app-lib/com.example.hellojni-
1/libhello-jni.so (JNI_OnLoad+20)
01-01 17:59:38.596: I/DEBUG(253):     #06  pc 00052eb1  /system
/lib/libdvm.so (dvmLoadNativeCode(char const*, Object*, char**)+468)
01-01 17:59:38.596: I/DEBUG(253):     #07  pc 0006a62d  /system/lib/libdvm.so
01-01 17:59:38.596: I/DEBUG(253):          // 省略部份日志 。。。。。。
01-01 17:59:38.596: I/DEBUG(253): stack:
01-01 17:59:38.596: I/DEBUG(253):          bedbf0b0  71b17034  /system/lib/libsechook.so
01-01 17:59:38.596: I/DEBUG(253):          bedbf0b4  7521ce28
01-01 17:59:38.596: I/DEBUG(253):          bedbf0b8  71b17030  /system/lib/libsechook.so
01-01 17:59:38.596: I/DEBUG(253):          bedbf0bc  4012c3cf  /system
/lib/libc.so (dlfree+50)
01-01 17:59:38.596: I/DEBUG(253):          bedbf0c0  40165000  /system/lib/libc.so
01-01 17:59:38.596: I/DEBUG(253):          // 省略部份日志 。。。。。。
01-01 17:59:38.736: W/ActivityManager(1185):   Force finishing activity com.example.hellojni
/.HelloJni
```

日志分析:

第3行开始启动应用，第5行尝试加载应用数据目录下的so，第6行在加载so文件的时候产生了一个致命的错误，第7行的Fatal signal 8提示这是一个致命的错误，这个信号是由linux内核发出来的，信号8的意思是浮点数运算异常，应该是在willCrash函数中做除0操作所产生的。下面重点看第15行backtrace的日志，backtrace日志可以看作是JNI调用的堆栈信息，以“#两位数字 pc”开头的都是backtrace日志。注意看第20行和21行，是我们自己编译的so文件和定义的两个函数，在这里引发了异常，导致程序崩溃。

```
[cpp]
1.  01-01 17:59:38.596: I/DEBUG(253):     #04  pc 00000cf4  /data/app-lib/com.example.hellojni-
    1/libhello-jni.so (willCrash+32)
2.  01-01 17:59:38.596: I/DEBUG(253):     #05  pc 00000d1c  /data/app-lib/com.example.hellojni-
    1/libhello-jni.so (JNI_OnLoad+20)
```

开始有些眉目了，但具体崩在这两个函数的哪个位置，我们是不确定的，如果函数代码比较少还好查，如果比较复杂的话，查起来也费劲。这时候就需要靠NDK为我们提供的工具来精确定位了。在这之前，我们先记录下让程序崩溃的汇编指令地址，willCrash：00000cf4，JNI_OnLoad：00000d1c

方式1：使用arm-linux-androideabi-addr2line 定位出错位置

以arm架构的CPU为例，执行如下命令：

```
[cpp]
1.  /Users/yangxin/Documents/devToos/java/android-ndk-r9d/toolchains/arm-linux-androideabi-
    4.8/prebuilt/darwin-x86_64/bin/arm-linux-androideabi-addr2line -e /Users/yangxin/Documents
    /devToos/java/android-ndk-r9d/samples/hello-jni/obj/local/armeabi-v7a/libhello-
    jni.so 00000cf4 00000d1c
```

-e：指定so文件路径

0000cf4 0000d1c：出错的汇编指令地址

结果如下：

```
yangxin-MacBook-Pro:bin yangxin$ /Users/yangxin/Documents/devToos/java/android-ndk-r9d/toolchains/arm-linux-a
ndroideabi-4.8/prebuilt/darwin-x86_64/bin/arm-linux-androideabi-addr2line -e /Users/yangxin/Documents/devToos
/java/android-ndk-r9d/samples/hello-jni/obj/local/armeabi-v7a/libhello-jni.so 00000cf4 00000d1c
/Users/yangxin/Documents/devToos/java/android-ndk-r9d/samples/hello-jni/jni/hello-jni.c:10
/Users/yangxin/Documents/devToos/java/android-ndk-r9d/samples/hello-jni/jni/hello-jni.c:15
```

是不是惊喜的看到我们想要的结果了，分别在hello-jni.c的10和15行的出的错，再回去看看hello-jni.c的源码，15行的

Jni_OnLoad函内调用了willCrash函数，第10行做了除0的操作引发的crash。

方式2：使用arm-linux-androideabi-objdump 定位出错的函数信息

在第一种方式中，通过addr2lin已经获取到了代码出错的位置，但是不知道函数的上下文信息，显得有点不是那么的“完美”，对于追求极致的我来说，这显然是不够的，下面我们来看一下怎么来定位函数的信息。

首先使用如下命令导出so的函数表信息：

```cpp
/Users/yangxin/Documents/devToos/java/android-ndk-r9d/toolchains/arm-linux-androideabi-4.8/prebuilt/darwin-x86_64/bin/arm-linux-androideabi-objdump -S -D /Users/yangxin/Documents/devToos/java/android-ndk-r9d/samples/hello-jni/obj/local/armeabi-v7a/libhello-jni.so > Users/yangxin/Desktop/dump.log
```

在生成的asm文件中，找出我们开始定位到的那两个出错的汇编指令地址（在文件中搜索cf4或willCrash可以找到），如下图所示：

```
void willCrash()
{
cd4: e92d4800  push {fp, lr}
cd8: e28db004  add  fp, sp, #4
cdc: e24dd008  sub  sp, sp, #8
    int i = 10;
ce0: e3a0300a  mov  r3, #10
ce4: e50b300c  str  r3, [fp, #-12]
    int y = i / 0;
ce8: e51b300c  ldr  r3, [fp, #-12]
cec: e1a00003  mov  r0, r3
cf0: e3a01000  mov  r1, #0
cf4: eb000020  bl   d7c <__aeabi_idiv>
cf8: e1a03000  mov  r3, r0
cfc: e50b3008  str  r3, [fp, #-8]
}
d00: e24bd004  sub  sp, fp, #4
d04: e8bd8800  pop  {fp, pc}

JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM* vm, void* reserved)
{
d08: e92d4800  push {fp, lr}
d0c: e28db004  add  fp, sp, #4
d10: e24dd008  sub  sp, sp, #8
d14: e50b0008  str  r0, [fp, #-8]
d18: e50b100c  str  r1, [fp, #-12]
    willCrash();
d1c: ebfffffec  bl   cd4 <willCrash>
    return JNI_VERSION_1_4;
d20: e3a03004  mov  r3, #4
d24: e3403001  movt r3, #1
}
```


通过这种方式，也可以查出这两个出错的指针地址分别位于哪个函数中。

方式3：ndk-stack

如果你觉得上面的方法太麻烦的话，ndk-stack可以帮你减轻操作步骤，直接定位到代码出错的位置。

使用adb获取logcat的日志，并通过管道输出给ndk-stack分析，并指定包含符号表的so文件位置。如果程序包含多种CPU架构，需要根据手机的CPU类型，来选择不同的CPU架构目录。以armv7架构为例，执行如下命令：

```cpp
adb logcat | ndk-stack -sym /Users/yangxin/Documents/devToos/java/android-ndk-r9d/samples/hello-jni/obj/local/armeabi-v7a
```

当程序发生crash时，会输出如下信息：

```cpp
pid: 22654, tid: 22654, name: xample.hellojni  >>> com.example.hellojni <<<
signal 8 (SIGFPE), code -6 (SI_TKILL), fault addr 0000587e
Stack frame #00  pc 0002210c  /system/lib/libc.so (tgkill+12)
Stack frame #01  pc 00013165  /system/lib/libc.so (pthread_kill+48)
Stack frame #02  pc 00013379  /system/lib/libc.so (raise+10)
Stack frame #03  pc 00000e80  /data/app-lib/com.example.hellojni-1/libhello-jni.so (__aeabi_idiv0+8): Routine __aeabi_idiv0 at /s/ndk-toolchain/src/build/../gcc/gcc-4.6/libgcc/../gcc/config/arm/lib1funcs.asm:1270
Stack frame #04  pc 00000cf4  /data/app-lib/com.example.hellojni-1/libhello-jni.so (willCrash+32): Routine willCrash at /Users/yangxin/Documents/devToos/java/android-ndk-r9d/samples/hello-jni/jni/hello-jni.c:10
Stack frame #05  pc 00000d1c  /data/app-lib/com.example.hellojni-1/libhello-jni.so (JNI_OnLoad+20): Routine JNI_OnLoad at /Users/yangxin/Documents/devToos/java/android-ndk-r9d/samples/hello-jni/jni/hello-jni.c:15
Stack frame #06  pc 00052eb1  /system/lib/libdvm.so (dvmLoadNativeCode(char const*, Object*, char**)+468)
Stack frame #07  pc 0006a62d  /system/lib/libdvm.so
```

第7行和第8行分别打印出了在源文件中出错的位置，和addr2line得到的结果一样。

先获取日志再分析:

这种方式和上面的方法差不多，只是获取log的来源不一样。适用于应用或游戏给测试部们测试的时候，测试人员发现crash，用adb logcat保存日志文件，然后发给程序员通过ndk-stack命令分析。操作流程如下：

```cpp
adb logcat > crash.log
ndk-stack -sym /Users/yangxin/Documents/devToos/java/android-ndk-r9d/samples/hello-jni/obj/local/armeabi-v7a -dump crash.log
```

得到的结果和上面的方式是一样的。

本文已收录于以下专栏：JNI/NDK开发指南 (http://blog.csdn.net/column/details/blogjnindk.html)

(/qq_19986309) 回复 11楼

. build ndk-stack python-packages sources
.. ndk-build ndk-which shader-tools sysroot
.DS_Store ndk-depends platforms simpleperf toolchains

CHANGELOG.md ndk-gdb prebuilt source.properties

我的目录下没有 simple 只有一个 simpleperf

14

simpleperf ls -a
annotate.config bin report.py
.. annotate.py binary_cache_builder.config report_sample.py
NOTICE app_profiler.config binary_cache_builder.py simpleperf_report_lib.py
README.md app_profiler.py repo.prop utils.py

而且 cd 进去是一堆这个鬼 求 hello-jni

loveniy (/loveniy) 2017-04-13 15:02 1条回复 回复 10楼

(/loveniy)

刚登上账号来顶的，请问楼主，ndk-stack和arm-linux-androideabi-addr2line工具去哪找呢？是去官网下载ndk，还是去本地工程里找呢？我昨天用官网的一直报错，然后用本地工程里才可以，不知楼主都是去哪找工具

loveniy (/loveniy) 2017-04-13 15:00 回复 9楼

(/loveniy)

刚登上账号来顶的，请问楼主，ndk-stack和arm-linux-androideabi-addr2line工具去哪找呢？是去官网下载ndk，还是去本地工程里找呢？我昨天用官网的一直报错，然后用本地工程里才可以，不知楼主都是去哪找工具

查看 17 条热评

android使用**ndk-stack**调试**JNI**部分的**C/C++**代码

法一：使用ndk-stack输出调用堆栈我这里的ndk-stack位置为：/home/ oldmtn 2013年05月06日 11:50 15419
hwh/Android_Project/Environment/android-ndk-r8e 'ndk...

(http://blog.csdn.net/oldmtn/article/details/8889654)

ndk-stack的使用

cheyiliu 2015年12月11日 22:26 1922

问题及解决方法 jni开发过程中native崩溃log样式如下 F/libc (12115): Fatal signal 11 (SIGSEGV) at 0x37413144 (code=1),..
..

(http://blog.csdn.net/cheyiliu/article/details/50269687)

JNI NDK开发**Crash**错误定位 调试 **- 艺搜 - CSDN**博客

2018-1-10

JNI NDK开发Crash错误定位 调试 2016-08-24 16:37 190人阅读 评论(0) 收藏 举报 分类: 未分类(86) 作者同类文章X 版权声明:时间宝贵,只能复制+粘贴,若...

(http://blog.csdn.net/elesos/article/details/52301838?>)

解决**JNI**调用在**Android5.0+**闪退问题 **- CSDN**博客

2018-1-18

标签: android / jni / crash /5746 编辑 删除 日志信息如下: 08-14 15:48:41.127: A/art(5526): art/runtime/check_jni.cc:70] JNI DETECTED ...

(http://blog.csdn.net/linchaolong/article/details/47663253?>)

Jni C/C++运行时遇到异常怎么办**?**捕获与抛出 **- CSDN**博客

2018-1-17

有个头疼的问题,Jni C/C++遇到问题闪退怎么办?有办法,我们可以在异常发生后通过判断清除异常解决,保持程序及时反应处理。比如:package crash; import java.security...

(http://blog.csdn.net/zhangbuzhangbu/article/details/52939424?>)

JNI 实战全面解析 **- CSDN**博客 2018-1-17

= libvlcjni.c libvlcjni-util.c libvlcjni-track.c libvlcjni-medialist.c aout.c vout.c libvlcjni-equalizer.c native_crash_handler.c LOCAL_SRC_...

(http://blog.csdn.net/banketree/article/details/40535325?>)

14

Android NDK开发**Crash**错误定位 **- CSDN**博客 2018-1-17

第7行定义了一个willCrash函数,函数中有一个除0的非法操作,会造成程序崩溃。第13行JNI_OnLoad函数中调用了willCrash,这个函数会在Java加载完.so文件之后回调,也就...

(http://m.blog.csdn.net/yzl11/article/details/51923190?>)

Android NDK开发**Crash**错误定位 **- CSDN**博客 2018-1-15

是不是惊喜的看到我们想要的结果了,分别在hello-jni.c的10和15行的出的错,再回去看看hello-jni.c的源码,15行的Jni_OnLoad函内调用了willCrash函数,第10行做...

(http://blog.csdn.net/bingqingsuimeng/article/details/51364854?>)

在**JNI** 编程中避免内存泄漏与崩溃 **- CSDN**博客 2018-1-17

JNI,Java Native Interface,是 native code 的编程接口。JNI 使 Java 代码程序...23/catching-posix-signals-on-android/ 这篇文章主要讲述如何捕获jni crash,.....

(http://blog.csdn.net/joewolf/article/details/52202718?>)

Android的**Crash**崩溃解决方案**-Bugly**的使用 **- CSDN**博客 2018-1-18

腾讯的Bugly可以在app出现崩溃的时候上传错误信息,定位错误原因和语句,并且可以查看影响的用户数和程序Crash次数等等信息 下面说一哈Eclipse+Android时Bugly的配置和使用...

(http://blog.csdn.net/tuke_tuke/article/details/51584879?>)

Android NDK开发**Crash**错误定位 **- CSDN**博客 2018-1-15

第7行定义了一个willCrash函数,函数中有一个除0的非法操作,会造成程序崩溃。第13行JNI_OnLoad函数中调用了willCrash,这个函数会在Java加载完.so文件之后回调,也就...

(http://blog.csdn.net/wzwind/article/details/50344131?>)

Android NDK开发**Crash**错误定位 **- CSDN**博客 2018-1-15

01-01 17:59:38.596: I/DEBUG(253): #04 pc 00000cf4 /data/app-lib/com.example.hellojni-1/libhello-jni.so (willCrash+32) 01-01 17:59:38.59...

(http://blog.csdn.net/xyang81/article/details/42319789?>)

ndk-stack的使用 c_boy_lu 2015年11月13日 10:22 751

这是对ndk-stack使用文档的翻译，文档所在的路径是：\android-ndk-r9d\docs\NDK-STACK.html 介绍：这篇文档描述的是ndk-stack工具，从R6版本起...

(http://blog.csdn.net/c_boy_lu/article/details/49814003)

使用 **ndk-stack** 寻找**Android**程序**Crash**的原因

开篇废话很久不研究cocos2d-x了，也不知道如今发展如何了。先前写游戏时会分几块。主要功能代码都是用C++编写，编辑器用vs android平台相关功能使用 Java 编写，编辑器用 Eclip...

change_from_now 2016年12月02日 17:02 2708

(http://blog.csdn.net/change_from_now/article/details/53436581)

分析安卓ANR tombstone使用ndk-stack addr2line

转自https://my.oschina.net/u/2424583/blog/535296 刷机过程中会碰到很多crash问题，此时安卓会在/data/tombstones目录下保存9个s...

zerokkqq 2017年06月15日 07:01 581

(http://blog.csdn.net/zerokkqq/article/details/73274171)

ndk-stack定位不出崩溃代码行的问题

flying8127 2014年11月03日 17:54 2597

NDK开发包中自带的NDK-STACK工具可以

(http://blog.csdn.net/flying8127/article/details/40743875)

ndk-stack 学习使用

wanzhihui0000 2014年04月04日 17:06 15114

最近在mac上编译android 版本，各种崩溃让人蛋疼，网上学习了下ndk-stack使用方法。...

(http://blog.csdn.net/wanzhihui0000/article/details/22946073)

Android studio下使用ndk-stack定位crash

a568478312 2017年10月09日 14:42 154

Android studio下使用ndk-stack定位crash

(http://blog.csdn.net/a568478312/article/details/78182422)

【玩转cocos2d-x之三十八】如何使用NDK-STACK tool来恢复Cocos2d-x安卓错...

很多童鞋在纠结在Cocos2d-x中安卓项目如何调试JNI部分的C++代码，在吃完2个茶叶蛋后我决定放大招。。。本文翻译自：http://www.cocos2d-x.org/forums/6/to...

jackyvincefu 2014年03月31日 23:38 6679

(http://blog.csdn.net/jackyvincefu/article/details/22699129)

signal 6 (SIGABRT), code -6 (SI_TKILL), fault addr

摘抄自：http://zhidao.baidu.com/link?url=gOXPCxE4HSq9GOGH0QGNS5zLXUWkeeHrmWpD_W3DrUlINLgJF7OGV4RCAEQGDGQ...

daweibalang717 2015年04月02日 16:17 11340

(http://blog.csdn.net/daweibalang717/article/details/44833709)

linux signal 处理

fz_ywj 2013年06月18日 22:13 21458

源地址：http://blog.csdn.net/zhuixundelang/article/details/5979465 linuxsignal 处理 说明: 本文主要翻译自UL...

(http://blog.csdn.net/fz_ywj/article/details/9124401)

链接libc.so导致crash

chenrunhua 2012年12月05日 19:02 2540

...再进时就crash了。网上搜大部分说是内存开辟和释

放不对的导致出错。后来用工具arm-eabi-addr2line -f -e xx...

(http://blog.csdn.net/chenrunhua/article/details/8261663)

魅族和三星Galaxy 5.0webView 问题Android Crash Report - Native crash at /sy…

解决办法是当前activity 销毁的时候 webView.destroy(); hine: ConnectedState (whe...

u012629497 2016年04月29日 11:46 5541

14

(http://blog.csdn.net/u012629497/article/details/51280392)

Android Crash Report - Native crash at /system/lib/libc.so caused by webview

hine: ConnectedState (when=-2ms what=131155 arg1=657!CMD_RSSI_POLL 657 0 “HDFin-Tech-1” 38:22:d6:94...

eandroidhu 2016年12月22日 10:59 1778

(http://blog.csdn.net/eandroidhu/article/details/53811540)

Android系统文件夹结构解析(五)--/system/lib

/system/lib lib目录中存放的主要是系统底层库，如平台运行时 zhimibuhui188 2012年11月20日 09:53 1772
库。 /system/lib/libaes.so /system/lib/libagl.so /system/lib/...

(http://blog.csdn.net/zhimibuhui188/article/details/8202314)

求救,heap corruption detected by dlmalloc,求大神

求助大神,本人在开发的时候碰到碰到APP自动退出的问题,stacktrace里面抓不到异常信息,但是提示有一个heap corruptio
n detected by dlmalloc,在网上查阅相关资...

aihuo4431 2017年01月02日 12:59 440

(http://blog.csdn.net/aihuo4431/article/details/53977499)

Android crash 日志 分析

chenjianjk 2014年07月01日 16:41 10083

在Android开发的过程中，有时候需要在

(http://blog.csdn.net/chenjianjk/article/details/36190485)

Android 运行崩溃找不到so包解决方案

shikangkai 2016年06月08日 10:51 9613

错误信息最近在把Bing提供语音识别服务集成到自己的应用中来时遇到了一些问题：java.lang.UnsatisfiedLinkError: dalvi
k.system.PathClassLoader...

(http://blog.csdn.net/shikangkai/article/details/51610997)

android 7.0 因为.so文件而崩溃事件解决

version1_0 2017年05月31日 15:18 4114

android 7.0 因为.so文件而崩溃事件解决

(http://blog.csdn.net/version1_0/article/details/72820575)

Andorid 引用.so库导致的崩溃问题解决办法

wblyuyang 2016年09月25日 17:17 790

使用第三方SDK的时候，经常会有.so库。使用时一个重要问题是：主项目和引用库项目中lib的目录结构必须保持一致，
即库项目中含有arm64-v8a目录，住工程和其他库工程也必须含有arm64-v8a目...

(http://blog.csdn.net/wblyuyang/article/details/52663054)