**原文链接：**<http://blog.csdn.net/oldmtn/article/details/8889654>

# 方法一：使用ndk-stack输出调用堆栈

我这里的ndk-stack位置为：/home/hwh/Android\_Project/Environment/android-ndk-r8e

'ndk-stack' 是一个工具，可以将.so文件的地址映射到相应的编译此.so文件的.h/.cpp的具体地址.

举例：

我这里有2个文件SampleJNIBinder.h和SampleJNIBinder.cpp，编译后生成libSampleJNIBinder.so这个动态库。

调用此.so文件的时候，出现了如下的错误。

**[python]** [view plain](http://blog.csdn.net/oldmtn/article/details/8889654" \o "view plain) [copy](http://blog.csdn.net/oldmtn/article/details/8889654" \o "copy" \t "http://blog.csdn.net/oldmtn/article/details/_blank) [print](http://blog.csdn.net/oldmtn/article/details/8889654" \o "print" \t "http://blog.csdn.net/oldmtn/article/details/_blank)[?](http://blog.csdn.net/oldmtn/article/details/8889654" \o "?)

1. F/libc    ( 1783): Fatal signal 11 (SIGSEGV) at 0x00000000 (code=1)
2. I/DEBUG   (  115): \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\*
3. I/DEBUG   (  115): Build fingerprint: 'intel/mfld\_dv10/mfld\_dv10:4.0.4/IMM76D/release.20130301:eng/dev-keys'
4. I/DEBUG   (  115): pid: 1783, tid: 1783  >>> com.example.apis <<<
5. I/DEBUG   (  115): signal 11 (SIGSEGV), code 1 (SEGV\_MAPERR), fault addr 00000000
6. I/DEBUG   (  115):  eax bfb0eb4e  ebx 5e35753c  ecx 00000000  edx 5f200019
7. I/DEBUG   (  115):  esi bfb0eb48  edi 08b20db0
8. I/DEBUG   (  115):  xcs 00000073  xds 0000007b  xes 0000007b  xfs 00000000 xss 0000007b
9. I/DEBUG   (  115):  eip 5e35464a  ebp bfb0eb98  esp bfb0eb30  flags 00010206
10. <span style="color:#FF0000;"><strong>I/DEBUG   (  115):     #00  eip: 0000264a  /data/data/com.example.apis/lib/libSampleJNIBinder.so</strong></span>
11. I/DEBUG   (  115):     #01  eip: 0007a959  /system/lib/libdvm.so (\_Z16dvmCallJNIMethodPKjP6JValuePK6MethodP6Thread+0x1e9)

红色部分的代码是错误的位置，对于eip这个寄存器里面的值：0000264a，我们找不到任何线索。

但是可以通过'adb logcat > foo.txt' 将错误信息导出到当前目录的foo.txt文件中。

然后通过：'./ndk-stack -sym /home/<user-name>/workspace\_android/HHDemos/obj/local/x86 -dump foo.txt'这个命令可以解析

这样返回的结果为：

**[java]** [view plain](http://blog.csdn.net/oldmtn/article/details/8889654" \o "view plain) [copy](http://blog.csdn.net/oldmtn/article/details/8889654" \o "copy" \t "http://blog.csdn.net/oldmtn/article/details/_blank) [print](http://blog.csdn.net/oldmtn/article/details/8889654" \o "print" \t "http://blog.csdn.net/oldmtn/article/details/_blank)[?](http://blog.csdn.net/oldmtn/article/details/8889654" \o "?)

1. huanghao@Mtn:~/Projects/AndroidNDK$ ./ndk-stack -sym /home/huanghao/workspace\_android/HHDemos/obj/local/x86 -dump foo.txt
2. \*\*\*\*\*\*\*\*\*\* Crash dump: \*\*\*\*\*\*\*\*\*\*
3. Build fingerprint: 'intel/mfld\_dv10/mfld\_dv10:4.0.4/IMM76D/release.20130301:eng/dev-keys'
4. pid: 2319, tid: 2319  >>> com.example.apis <<<
5. signal 11 (SIGSEGV), code 1 (SEGV\_MAPERR), fault addr 00000000
6. <span style="font-size:14px;color:#FF6600;"><strong>Stack frame #00  eip: 0000264a  /data/data/com.example.apis/lib/libSampleJNIBinder.so:
7. Routine Java\_com\_example\_apis\_JNI\_SampleJNIBinder\_JTCGetStringFromJNI
8. in /home/huanghao/workspace\_android/HHDemos/jni/SampleJNIBinder/SampleJNIBinder.cpp:35</strong></span>
9. Stack frame #01  eip: 0007a959  /system/lib/libdvm.so (\_Z16dvmCallJNIMethodPKjP6JValuePK6MethodP6Thread+0x1e9)
10. Stack frame #02  eip: 00063ba1  /system/lib/libdvm.so (\_Z21dvmCheckCallJNIMethodPKjP6JValuePK6MethodP6Thread+0x41)
11. Stack frame #03  eip: 00085d2d  /system/lib/libdvm.so (\_Z22dvmResolveNativeMethodPKjP6JValuePK6MethodP6Thread+0x13d)
12. Stack frame #04  eip: 00125d34  /system/lib/libdvm.so
13. Stack frame #05  eip: 0003f466  /system/lib/libdvm.so (\_Z11dvmMterpStdP6Thread+0x46)
14. Stack frame #06  eip: 0003bb06  /system/lib/libdvm.so (\_Z12dvmInterpretP6ThreadPK6MethodP6JValue+0xd6)
15. Stack frame #07  eip: 000a81fa  /system/lib/libdvm.so (\_Z15dvmInvokeMethodP6ObjectPK6MethodP11ArrayObjectS5\_P11ClassObjectb+0x51a)

这里我们很容易看出SampleJNIBinder.cpp:35有错误，那么我们找到那部分的代码：

如下

**[cpp]** [view plain](http://blog.csdn.net/oldmtn/article/details/8889654" \o "view plain) [copy](http://blog.csdn.net/oldmtn/article/details/8889654" \o "copy" \t "http://blog.csdn.net/oldmtn/article/details/_blank) [print](http://blog.csdn.net/oldmtn/article/details/8889654" \o "print" \t "http://blog.csdn.net/oldmtn/article/details/_blank)[?](http://blog.csdn.net/oldmtn/article/details/8889654" \o "?)

1. JNIEXPORT jstring JNICALL
2. Java\_com\_example\_apis\_JNI\_SampleJNIBinder\_JTCGetStringFromJNI(JNIEnv \*env, jobject thiz) {
3. **int** \* p = NULL;
4. \*p = 100; //这里是35行
6. **int** n1 = **true**;
7. **int** n2 = **false**;
8. LogPrintf("jni-hh", "n1: %d, n2: %d", n1, n2);

这里是一个给空指针赋值的错误，很简单吧。

好了，需要总结一下。

使用'ndk-stack'查看backstack信息的步骤如下：

1. 进入Android-NDK的路径

我这里为：/home/<user-name>/Projects/AndroidNDK

$ cd /home/<user-name>/Projects/AndroidNDK

2.导出logcat信息到foo.txt中

$ adb logcat > foo.txt

3.通过GPF的地址和.so文件 获取实际在.cpp文件错误的地址

例子：<user-name>@Mtn:~/Projects/AndroidNDK$ ./ndk-stack -sym <.so文件目录> -dump foo.txt

我的用法：<user-name>@Mtn:~/Projects/AndroidNDK$ ./ndk-stack -sym /home/<user-name>/workspace\_android/HHDemos/obj/local/**x86** -dump foo.txt

当然，我们也可以将上面2句合并成一句：

如下：

$ adb logcat | $NDK\_HOME/ndk-stack -sym /home/hwh/Android\_Project/Code/HHDemos/obj/local/x86

注意：

1. 此时我们的操作路径是AndroidSDK目录, foo.txt也在该目录。

2.上面的x86是你平板的CPU决定了，如果你的平板是ARM的CPU那么应该改为：armeabi了。

但是我公司同事的机子adb logcat好像自动就将错误的地址转换为.cpp文件中的地址，可能需要具体配置。

暂时不清楚，我弄好了的话就在这里添加。。。。。；

法二：使用i686-linux-android-addr2line输出调试信息

下面是该文件的路径

/home/hwh/Android\_Project/Environment/android-ndk-r8e/toolchains/x86-4.6/prebuilt/linux-x86/bin

其中android-ndk-r8e是我在ndk官网下载的android-ndk-r8e-linux-x86.tar.bz2解压缩后的文件夹。

用法：

$ ./i686-linux-android-addr2line -f -e /home/hwh/Android\_Project/Code/HHDemos/obj/local/x86/libSampleJNIBinder.so 0000415C

总结一下就是：

$ ./i686-linux-android-addr2line -f -e <.so文件的pathName> <出错的地址>

输出结果为：

**[cpp]** [view plain](http://blog.csdn.net/oldmtn/article/details/8889654" \o "view plain) [copy](http://blog.csdn.net/oldmtn/article/details/8889654" \o "copy" \t "http://blog.csdn.net/oldmtn/article/details/_blank) [print](http://blog.csdn.net/oldmtn/article/details/8889654" \o "print" \t "http://blog.csdn.net/oldmtn/article/details/_blank)[?](http://blog.csdn.net/oldmtn/article/details/8889654" \o "?)

1. hwh@Mountain:~/Android\_Project/Environment/android-ndk-r8e/toolchains/x86-4.6/prebuilt/linux-x86/bin$ ./i686-linux-android-addr2line -f -e
2. /home/hwh/Android\_Project/Code/HHDemos/obj/local/x86/libSampleJNIBinder.so 0000415C
3. Java\_com\_example\_apis\_JNI\_SampleJNIBinder\_JTCGetStringFromJNI
4. /home/hwh/Android\_Project/Code/HHDemos/jni/SampleJNIBinder/SampleJNIBinder.cpp:35

**[cpp]** [view plain](http://blog.csdn.net/oldmtn/article/details/8889654" \o "view plain) [copy](http://blog.csdn.net/oldmtn/article/details/8889654" \o "copy" \t "http://blog.csdn.net/oldmtn/article/details/_blank) [print](http://blog.csdn.net/oldmtn/article/details/8889654" \o "print" \t "http://blog.csdn.net/oldmtn/article/details/_blank)[?](http://blog.csdn.net/oldmtn/article/details/8889654" \o "?)

1. i686-linux-android-addr2line
2. 返回的信息为：

1. 出错的函数

2. 出错地点在该.cpp文件的行数

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Android ndk-stack tool  
----------------------  
  
Introduction:  
-------------  
  
This document describes the 'ndk-stack' tool that is distributed withthe Android NDK, since release r6.  
  
Overview:  
---------  
  
'ndk-stack' is a simple tool that allows you to filter stack traces as theyappear in the output of 'adb logcat' and replace any address inside a shared  
library with the corresponding : values.

In a nutshell, it will translate something like:  
  
    I/DEBUG   (   31): \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\*  
    I/DEBUG   (   31): Build fingerprint: 'generic/google\_sdk/generic/:2.2/FRF91/43546:eng/test-keys'  
    I/DEBUG   (   31): pid: 351, tid: 351  %gt;%gt;%gt; /data/local/ndk-tests/crasher <<<  
    I/DEBUG   (   31): signal 11 (SIGSEGV), fault addr 0d9f00d8  
    I/DEBUG   (   31):  r0 0000af88  r1 0000a008  r2 baadf00d  r3 0d9f00d8  
    I/DEBUG   (   31):  r4 00000004  r5 0000a008  r6 0000af88  r7 00013c44  
    I/DEBUG   (   31):  r8 00000000  r9 00000000  10 00000000  fp 00000000  
    I/DEBUG   (   31):  ip 0000959c  sp be956cc8  lr 00008403  pc 0000841e  cpsr 60000030  
    I/DEBUG   (   31):          #00  pc 0000841e  /data/local/ndk-tests/crasher  
    I/DEBUG   (   31):          #01  pc 000083fe  /data/local/ndk-tests/crasher  
    I/DEBUG   (   31):          #02  pc 000083f6  /data/local/ndk-tests/crasher  
    I/DEBUG   (   31):          #03  pc 000191ac  /system/lib/libc.so  
    I/DEBUG   (   31):          #04  pc 000083ea  /data/local/ndk-tests/crasher  
    I/DEBUG   (   31):          #05  pc 00008458  /data/local/ndk-tests/crasher  
    I/DEBUG   (   31):          #06  pc 0000d362  /system/lib/libc.so  
    I/DEBUG   (   31):  
  
Into the more readable output:  
  
    \*\*\*\*\*\*\*\*\*\* Crash dump: \*\*\*\*\*\*\*\*\*\*  
    Build fingerprint: 'generic/google\_sdk/generic/:2.2/FRF91/43546:eng/test-keys'  
    pid: 351, tid: 351  >>> /data/local/ndk-tests/crasher <<<  
    signal 11 (SIGSEGV), fault addr 0d9f00d8  
    Stack frame #00  pc 0000841e  /data/local/ndk-tests/crasher : Routine zoo in /tmp/foo/crasher/jni/zoo.c:13  
    Stack frame #01  pc 000083fe  /data/local/ndk-tests/crasher : Routine bar in /tmp/foo/crasher/jni/bar.c:5  
    Stack frame #02  pc 000083f6  /data/local/ndk-tests/crasher : Routine my\_comparison in /tmp/foo/crasher/jni/foo.c:9  
    Stack frame #03  pc 000191ac  /system/lib/libc.so  
    Stack frame #04  pc 000083ea  /data/local/ndk-tests/crasher : Routine foo in /tmp/foo/crasher/jni/foo.c:14  
    Stack frame #05  pc 00008458  /data/local/ndk-tests/crasher : Routine main in /tmp/foo/crasher/jni/main.c:19  
    Stack frame #06  pc 0000d362  /system/lib/libc.so  
  
Usage:  
------  
  
To do this, you will first need a directory containing symbolic versions of yourapplication's shared libraries. If you use the NDK build system (i.e. ndk-build),  
then these are always located under $PROJECT\_PATH/obj/local/<ab>, where<ab> stands for your device's ABI (i.e. 'armeabi' by default).  
  
You can feed the logcat text either as direct input to the program, e.g.:  
  
   adb logcat | $NDK/ndk-stack -sym $PROJECT\_PATH/obj/local/armeabi  
  
Or you can use the -dump option to specify the logcat as an input file, e.g.:  
  
   adb logcat > /tmp/foo.txt  
   $NDK/ndk-stack -sym $PROJECT\_PATH/obj/local/armeabi -dump foo.txt  
  
  
TODO:  
-----  
  
A future version of 'ndk-stack' will try to launch 'adb logcat' and select thelibrary path automatically. For now, you'll have to do these steps manually.  
  
As of now, ndk-stack doesn't handle libraries that don't have debug informationin them. It may be useful to try to detect the nearest function entry point to  
a given PC address (e.g. as in the libc.so example above).

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

转载文章2：<http://terryblog.blog.51cto.com/1764499/796583>

Android开发中，我们也会经常遇到段错误，也就是SIGSEGV（11），这个时候libc的backtrace会打印出对应的堆栈信 息，而你看到的仅仅是一对数字，好像无从查起。

如下面这一从串断错误：

ActivityManager( 1105): Displayed activity com.android.browser/.BrowserActivity: 2460 ms (total 2460 ms)  
I/DEBUG   (13002): \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\*  
I/DEBUG   (13002): Build fingerprint: 'unknown'  
I/DEBUG   (13002): pid: 20363, tid: 20375  >>> com.android.browser <<<  
I/DEBUG   (13002): signal 11 (SIGSEGV), fault addr ffc00000  
I/DEBUG   (13002):  r0 059fc2a0  r1 4a3bcef8  r2 e59fc2a0  r3 4a3bcc58  
I/DEBUG   (13002):  r4 4a3bc101  r5 4ebe0a3c  r6 4a3bc120  r7 012fff10  
I/DEBUG   (13002):  r8 500de101  r9 500ee12d  10 a87dfb20  fp 4ebe58e0  
I/DEBUG   (13002):  ip ffc00000  sp 4ebe0a30  lr 4a3bcc58  pc a862f3a0  cpsr 00000030  
I/DEBUG   (13002):  d0  0000001100000011  d1  0000001100000011  
I/DEBUG   (13002):  d2  0000001100000011  d3  0000001100000011  
I/DEBUG   (13002):  d4  0000001100000011  d5  0000001100000011  
I/DEBUG   (13002):  d6  0000001100000011  d7  4060000000000080  
I/DEBUG   (13002):  d8  41d3d1762e40d70a  d9  41d3d1762e440a3d  
I/DEBUG   (13002):  d10 0000000000000000  d11 0000000000000000  
I/DEBUG   (13002):  d12 0000000000000000  d13 0000000000000000  
I/DEBUG   (13002):  d14 0000000000000000  d15 0000000000000000  
I/DEBUG   (13002):  d16 3ff0000000000000  d17 3ff0000000000000  
I/DEBUG   (13002):  d18 40cd268000000000  d19 3f3b9cc1b0bac000  
I/DEBUG   (13002):  d20 3ff0000000000000  d21 8000000000000000  
I/DEBUG   (13002):  d22 0000000000000000  d23 0000000000000000  
I/DEBUG   (13002):  d24 3ff0000000000000  d25 0000000000000000  
I/DEBUG   (13002):  d26 0000000000000000  d27 0000000000000000  
I/DEBUG   (13002):  d28 0000000000000000  d29 3ff0000000000000  
I/DEBUG   (13002):  d30 0000000000000000  d31 3ff0000000000000  
I/DEBUG   (13002):  scr 60000013  
I/DEBUG   (13002):   
I/DEBUG   (13002):          #00  pc 0032f3a0  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #01  pc 003243b0  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #02  pc 003167b2  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #03  pc 0038f2de  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #04  pc 0038f416  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #05  pc 0030d392  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #06  pc 003796e2  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #07  pc 0038e36a  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #08  pc 003189f0  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #09  pc 00377f82  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #10  pc 0037ae0c  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #11  pc 0038e254  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #12  pc 003189f0  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #13  pc 0031cf2c  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #14  pc 0038e52a  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #15  pc 0038c2d0  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #16  pc 0031cf76  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #17  pc 0038e546  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #18  pc 003189f0  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #19  pc 0031ca40  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #20  pc 0038e3be  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #21  pc 0038c2d0  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #22  pc 0031cf76  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #23  pc 0038e546  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #24  pc 0038c2d0  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #25  pc 00379054  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #26  pc 0031d254  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #27  pc 0030d5d6  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #28  pc 0030d7d2  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #29  pc 0031e354  /system/lib/libwebcore.so  
I/DEBUG   (13002):          #30  pc 0034ab3c  /system/lib/libwebcore.so  
I/DEBUG   (13002):   
I/DEBUG   (13002): code around pc:  
I/DEBUG   (13002): a862f380 469e4694 cc04f853 0e04f1a3 510cea4f   
I/DEBUG   (13002): a862f390 f41c0d09 bf080f00 44714249 c008f8d1   
I/DEBUG   (13002): a862f3a0 e000f8dc 0c1ff10e bf0842b8 2d04f853   
I/DEBUG   (13002): a862f3b0 0d010510 0f00f412 4249bf08 f8c2185a   
I/DEBUG   (13002): a862f3c0 e006c008 d1042b0c 99019b05 18426818   
I/DEBUG   (13002):   
I/DEBUG   (13002): code around lr:  
I/DEBUG   (13002): 4a3bcc38 e58d0000 e49d0004 e598200b e582002f   
I/DEBUG   (13002): 4a3bcc48 e52d0004 e3100001 0a000018 e3a03030   
I/DEBUG   (13002): 4a3bcc58 e59fc2a0 e002100c e59fc29c e151000c   
I/DEBUG   (13002): 4a3bcc68 0a000012 e59fc294 e002100c e0813003   
I/DEBUG   (13002): 4a3bcc78 e1a03123 e1c2200c e3530b02 ba000004   
I/DEBUG   (13002):   
I/DEBUG   (13002): stack:  
I/DEBUG   (13002):     4ebe09f0  50bfd848    
I/DEBUG   (13002):     4ebe09f4  50bfd858    
I/DEBUG   (13002):     4ebe09f8  50bfd834    
I/DEBUG   (13002):     4ebe09fc  afd19a05  /system/lib/libc.so  
I/DEBUG   (13002):     4ebe0a00  50bd3264    
I/DEBUG   (13002):     4ebe0a04  a86510ef  /system/lib/libwebcore.so  
I/DEBUG   (13002):     4ebe0a08  00000004    
I/DEBUG   (13002):     4ebe0a0c  50bfd854    
I/DEBUG   (13002):     4ebe0a10  002ece20  [heap]  
I/DEBUG   (13002):     4ebe0a14  4a3ba000    
I/DEBUG   (13002):     4ebe0a18  4ebe0a3c    
I/DEBUG   (13002):     4ebe0a1c  4ebe0a3c    
I/DEBUG   (13002):     4ebe0a20  4a3bc101    
I/DEBUG   (13002):     4ebe0a24  4ebe0a3c    
I/DEBUG   (13002):     4ebe0a28  df002777    
I/DEBUG   (13002):     4ebe0a2c  e3a070ad    
I/DEBUG   (13002): #00 4ebe0a30  002ece20  [heap]  
I/DEBUG   (13002):     4ebe0a34  49f627d0    
I/DEBUG   (13002):     4ebe0a38  a87d63c0  /system/lib/libwebcore.so  
I/DEBUG   (13002):     4ebe0a3c  4a3bd0e7    
I/DEBUG   (13002):     4ebe0a40  4a3bd0b8    
I/DEBUG   (13002):     4ebe0a44  4a3bcc58    
I/DEBUG   (13002):     4ebe0a48  00000003    
I/DEBUG   (13002):     4ebe0a4c  00000000    
I/DEBUG   (13002):     4ebe0a50  00001100    
I/DEBUG   (13002):     4ebe0a54  0000001f    
I/DEBUG   (13002):     4ebe0a58  00001074    
I/DEBUG   (13002):     4ebe0a5c  4ebe0b04    
I/DEBUG   (13002):     4ebe0a60  a87d63c0  /system/lib/libwebcore.so  
I/DEBUG   (13002):     4ebe0a64  4ebe0acc    
I/DEBUG   (13002):     4ebe0a68  4a3bc101    
I/DEBUG   (13002):     4ebe0a6c  a86243b5  /system/lib/libwebcore.so  
I/DEBUG   (13002): #01 4ebe0a70  4ebe0b38    
I/DEBUG   (13002):     4ebe0a74  00000064    
I/DEBUG   (13002):     4ebe0a78  003f0914  [heap]  
I/DEBUG   (13002):     4ebe0a7c  fffffc00    
I/DEBUG   (13002):     4ebe0a80  50bfd834    
I/DEBUG   (13002):     4ebe0a84  a87d63c0  /system/lib/libwebcore.so  
I/DEBUG   (13002):     4ebe0a88  4ebe0b38    
I/DEBUG   (13002):     4ebe0a8c  4ebe0b04    
I/DEBUG   (13002):     4ebe0a90  4ebe0acc

[复制代码](http://terryblog.blog.51cto.com/)

I/DEBUG   (13002):     4ebe0a94  a86167b7  /system/lib/libwebcore.so

我们的板子上的lib 经常被strip过了，没有了符号信息。不过我们可以通过编译时候生成的库来获取对应的符号信息。编译器也为我们提供了相应的工具:addr2line 全名为：arm-eabi-addr2line ，可在对应板子源码目录找到。

通过上面的断错误分析，位于system/lib/libwebcore.so 这个库出现了断错误，可以将其pull下来正逐行分析。命令为：arm-eabi-addr2line -f -e ~/桌面/libwebcore.so 0038f2de

这种分析法同样适用于使用jni开发的库。