NE 问题分析浅析

李庆美

整体介绍

Android 应用程序崩溃类型

JE： Java Exception

ANR： Application Not Responding

NE: Native Exception

KE: Kernel Exception

NE 崩溃常见原因

1. 内存访问问题

   空指针　野指针　数组越界　栈溢出

2. 资源不中足

   内存不足　句柄不足　JNI references　TLS

3. 程序状态错乱

   程序已经检测到程序状态，出现不可恢复状态时，一般主动Crash掉，比如 abort, assert

4. 系统问题

   实现代码逻辑没有问题，但崩溃率比较高，去云诊断根据 hash id，查找其他App是否有同类问题。

5. 诡异问题

   此类问题基本是单机问题，每崩溃的栈都不一样，需要拿到真机 Log 定位，大概率是硬件故障。

案例分析

指针类问题

```
#include <iostream>

using namespace std;

int main(void) {

// null pointer
int * ptr = NULL;
*ptr = 100; // BOOM

// dangling  pointer
string* pStr = new string("hello word");
cout << "string: " << *pStr << endl;
delete pStr;

cout << "string:" << *pStr << endl; // BOOM

}
```

解决办法

使用智能指针，将会大大降低自己维护内存的风险。

注意：智能指针传递过程中的所有者变化

https://zh.cppreference.com/w/cpp/memory

{ 问题分析办法 }

分析思路

1. 收集更多崩溃现场状态。

2. 工具解析堆栈，对应的源码，根据源码分析。

3. 程序状态打点，偿试恢复用户现场。

4. 调整逻辑，复现问题栈，将随机问题转化成必现。

日志分析

```
Time: 2018-12-01 07:31:33
Softversion: PD1709_A_1.18.8
Appversion: versionName = 5.6.3.2 versionCode = 11442
Process: com.vivo.browser
Flags: 0x38983ec5
Package: com.vivo.browser v11442 (5.6.3.2)
Foreground: Yes
Build: vivo/PD1709/PD1709:7.1.1/NMF26X/compil05151836:user/release-keys

*** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
Build fingerprint: 'vivo/PD1709/PD1709:7.1.1/NMF26X/compil05151836:user/release-keys'
Revision: '0'
ABI: 'arm'
pid: 20533, tid: 20886, name: Compositor  >>> com.vivo.browser <<<
signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 0x569
    r0 00000569  r1 00000001  r2 00000001  r3 00000080
    r4 0000001c  r5 edea6aa0  r6 b1bfc1a0  r7 0000001c
    r8 00000000  r9 a376c650  sl 00000001  fp a376c000
    ip bb534761  sp b4fb3d68  lr bbaf19b5  pc bbab6920  cpsr 80030030

backtrace:
    #00 pc 00811920  /data/app/com.vivo.browser-2/lib/arm/libwebviewchromium_vivo.so
    #01 pc 00876abf  /data/app/com.vivo.browser-2/lib/arm/libwebviewchromium_vivo.so
    #02 pc 0087a7cb  /data/app/com.vivo.browser-2/lib/arm/libwebviewchromium_vivo.so
    #03 pc 0084e099  /data/app/com.vivo.browser-2/lib/arm/libwebviewchromium_vivo.so
    #04 pc 0084d8df  /data/app/com.vivo.browser-2/lib/arm/libwebviewchromium_vivo.so
    #05 pc 0084d553  /data/app/com.vivo.browser-2/lib/arm/libwebviewchromium_vivo.so
    #06 pc 0089c4f9  /data/app/com.vivo.browser-2/lib/arm/libwebviewchromium_vivo.so
    #07 pc 0089cbe3  /data/app/com.vivo.browser-2/lib/arm/libwebviewchromium_vivo.so
```

第一步，确认是否挂在我们应用进程，版本是否正确。

第二步，确定PID，方便过滤LOG，Crash 线程信息

第三步，确定Crash 类型。注意0或值较小均为空指针。

第四步，取得相对地址，用于Addr2line来解栈。

函数调用顺利

{ 工具使用 }

使用 AddressSanitizer 编译库文件

AddressSanitizer 可以检测出内存溢出，野指针，越界访问等情况，并 Dump 出相应的调用栈和分析。

clang 3.1 , gcc 4.8 以上版本支持此编译选项。

建议开发版中打开此功能。

参考Url: https://en.wikipedia.org/wiki/AddressSanitizer

```
# liqingmei @ Ubuntu in ~work62/android_packages_apps_Browser_chromium62 on git:vivo_webview_trunk x [16:28:59] C:1
$ g++ test.cpp -o test -fsanitize=address

# liqingmei @ Ubuntu in ~work62/android_packages_apps_Browser_chromium62 on git:vivo_webview_trunk x [16:49:49]
$ ./test
ASAN:SIGSEGV
=================================================================
==18886==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000000 (pc 0x000000400fe3 bp 0x7ffdc0c96d10 sp 0x7ffdc0c96ce0 T0)
    #0 0x400fe2 in main (/home/liqingmei/workspace/source/chromium62/android_packages_apps_Browser_chromium62/test+0x400fe2)
    #1 0x7ffa3c9c682f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)
    #2 0x400eb8 in _start (/home/liqingmei/workspace/source/chromium62/android_packages_apps_Browser_chromium62/test+0x400eb8)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV ??:0 main
==18886==ABORTING
```

堆栈解析工具

使用NDK的 addr2line 工具将调用栈解析成与源码对应行号。

arm-linux-androideabi-addr2line –e /data/sym/libwebviewchromium_vivo.so -i -f –C 0x1234 0xABCE 0x1234

0x0080c784: cc::ScrollbarAnimationController::Animate(base::TimeTicks) at scrollbar_animation_controller.cc:143
0x00870f43: cc::LayerTreeHostImpl::AnimateScrollbars(base::TimeTicks) at layer_tree_host_impl.cc:4084
0x00874c4f: cc::LayerTreeHostImpl::Animate() at layer_tree_host_impl.cc:502
0x00848535: cc::Scheduler::BeginImplFrame(viz::BeginFrameArgs const&, base::TimeTicks) at scheduler.cc:484
0x00847d7b: cc::Scheduler::BeginImplFrameSynchronous(viz::BeginFrameArgs const&) at scheduler.cc:439
0x008479ef: cc::Scheduler::OnBeginFrameDerivedImpl(viz::BeginFrameArgs const&) at scheduler.cc:285
0x008964f9: viz::BeginFrameObserverBase::OnBeginFrame(viz::BeginFrameArgs const&) at begin_frame_source.cc:44
0x00896be3: viz::ExternalBeginFrameSource::OnBeginFrame(viz::BeginFrameArgs const&) at begin_frame_source.cc:329
0x00280139: void base::DispatchToMethod<content::CompositorExternalBeginFrameSource*, void (content::CompositorExternalBeginFrameSource::*)(viz::BeginFrameArgs const&),
std::__ndk1::tuple<viz::BeginFrameArgs> >(content::CompositorExternalBeginFrameSource* const&, void (content::CompositorExternalBeginFrameSource::*)(viz::BeginFrameArgs const&),
std::__ndk1::tuple<viz::BeginFrameArgs>&&) at tuple.h:63
0x01da34e1: bool IPC::MessageT<ViewMsg_BeginFrame_Meta, std::__ndk1::tuple<viz::BeginFrameArgs>, void>::Dispatch<content::CompositorExternalBeginFrameSource,
content::CompositorExternalBeginFrameSource, void, void (content::CompositorExternalBeginFrameSource::*)(viz::BeginFrameArgs const&)>(IPC::Message const*,
content::CompositorExternalBeginFrameSource*, content::CompositorExternalBeginFrameSource*, void*, void (content::CompositorExternalBeginFrameSource::*)(viz::BeginFrameArgs const&)) at
ipc_message_templates.h:146
0x01da33a5: content::CompositorExternalBeginFrameSource::OnMessageReceived(IPC::Message const&) at compositor_external_begin_frame_source.cc:79
0x01da3767: base::RepeatingCallback<void (IPC::Message const&)>::Run(IPC::Message const&) const & at callback.h:92
0x0029116d: base::OnceCallback<void ()>::Run() && at callback.h:64
0x0136774f: blink::scheduler::TaskQueueManager::ProcessTaskFromWorkQueue(blink::scheduler::internal::WorkQueue*, bool, blink::scheduler::LazyNow, base::TimeTicks*) at
task_queue_manager.cc:515

利用 BreakPad 收集崩溃时程序状态

BreakPad 是 Google 开源一套应用程序崩溃时，对程序状态进行收集的系统。

在项目中引用BreakPad库，进行相应的初始化，然后安装一个 ExceptionHandler，在Handler中处理状态收集。

```
exception_handler_ = new
google_breakpad::ExceptionHandler(
google_breakpad::MinidumpDescriptor("/tmp"), nullptr,
ExtraMessageCollection::DumpMessageToFile, nullptr,
true, -1);
```