# [android native 内存泄露检查（libc.debug.malloc ）](http://blog.csdn.net/haima1998/article/details/51508947)

\*c c++代码，由于其特殊性质，没有虚拟机概念，内存则直接是由用户管理，比如申请，释放，都是需要用户主动去触发，如果用户出现使用了申请，但是用完之后，没有调用释放，则会引起内存泄露。这种叫真正意义的内存泄露，只有重启机子，才能恢复。

相对而已java端的内存泄露，指的是一个应用长期运行，导致相互引用，无法释放，GC没法回收，引起的有效内存越来越小，我们将此现象叫做，内存泄露，通过关闭此应用，重新打开即可恢复内存。因此看来，java内存泄露和c c ++ 的 还是有本质区别的。

java本身的虚拟机里面会关注对象的申请，释放，这些不需要用户直接注，java虚拟机通过管理机制，将调用c c++里面真正的malloc free 方法，封装起来，将java对象的生命周期和malloc free 进行关联，则可以保证在对象不使用的时候，内存紧张时，释放掉不再被引用的对象，GC回收就是在做这件事请。回到我们这节的主要内容，如何定位我们的c c++的内存泄露。

Android对内存的使用包括内存泄漏和内存越界，内存泄漏会导致系统内存减少，最终分配不到内存，这样大的程序就不能运行，甚至系统没有内存而崩溃。Android中kernel和应用程序都可能会有内存泄漏和越界。对于Java代码，在越界的时候虚拟机会加以检查并抛出异常。而对于C/C++代码，越界的时候就悄无声息地让程序出错或crash

## 调试手段

在adb shell之下:确保/system/lib/ libc\_malloc\_debug\_leak.so存在

root@zs600b:/ # stop

root@zs600b:/ # setprop libc.debug.malloc 1

root@zs600b:/ # start

执行成功之后，浏览器和cs-app都无法打开了。。。

指针溢出，野指针，堆内存指针释放多遍等问题如何调试？

Android这边打开bionic/libc/bionic/malloc\_debug\_common.c里的配置

如adb shell setprop persist.libc.debug.malloc 10 再重启

之后可以复现问题，抓bugreport来分析问题，当然要匹配symbols(out/target/product/project/symbols)

此时overhead会比较重，可能会有一些不预期的anr(出现ANR时请点击等待)，但不影响测试。

start & stop are tools under /system/bin (system.img)

## 常见问题

/system/bin/sh: Missing module libc\_malloc\_debug\_leak.so required for malloc debug level 1: dlopen failed: library "libc\_malloc\_debug\_leak.so" not found

userdebug才是自带so，否则自己手动push进去吧。。，。

可以看看eng或这user-debug版中的/system/lib/下是否有这两个文件，其中libc\_malloc\_debug\_qemu.so是模拟器用的。

## 复现场景

堆引起的异常(调用malloc申请的内存后使用不当引起的异常)：

1. 申请后多次释放 (double free)

2. 释放后又去使用 (used after free)

3. 使用越界 (比如申请了50节内存，结果在使用时多用了8字节的内存，这样就把后面的内存的内容踩坏，引起堆结构异常)

4. 释放时传给free()的地址不是malloc()申请的地址，比如：p = malloc(10); free(p + 5);

5. 内存泄露：申请内存后，忘记释放或某些代码路径没有释放

## 源码分析

此malloc的调试原理是：当系统发现我们有libc.debug.malloc的一些列配置成立时，此时系统会将malloc free 等方法，重新指向到 lib\_malloc\_debug.so里面的对应实现方法，lib\_malloc\_debug.so里面的方法，像比较而言，多了一些记录信息，将每次的申请时的地址，堆栈，so等信息记录下来，然后我们需要的时候，则通过工具ddms dump出来，进行分析每个申请的内存，是否正常的释放了，是否出现了内存泄露。

|  |
| --- |
| // The value of libc.debug.malloc.  #if !defined(LIBC\_STATIC)  static int g\_malloc\_debug\_level = 0;  #endif  // Initializes memory allocation framework once per process.  static void malloc\_init\_impl() {  const char\* so\_name = NULL;  // If debug level has not been set by memcheck option in the emulator,  // lets grab it from libc.debug.malloc system property.  if (g\_malloc\_debug\_level == 0 && \_\_system\_property\_get("libc.debug.malloc", env)) {  g\_malloc\_debug\_level = atoi(env);  }  // Choose the appropriate .so for the requested debug level.  switch (g\_malloc\_debug\_level) {  case 1:  case 5:  case 10:  so\_name = "libc\_malloc\_debug\_leak.so";  break;  }  // Load .so that implements the required malloc debugging functionality.  void\* malloc\_impl\_handle = dlopen(so\_name, RTLD\_LAZY);  // No need to init the dispatch table because we can only get  // here if debug level is 1, 5, 10, or 20.  static MallocDebug malloc\_dispatch\_table \_\_attribute\_\_((aligned(32)));  switch (g\_malloc\_debug\_level) {  case 1:  InitMalloc(malloc\_impl\_handle, &malloc\_dispatch\_table, "leak");  break;  }  } |

## REF

[android native 代码内存泄露 定位方案](http://www.jianshu.com/p/2b43abdd6647)

[**[MTK软件原创] 如何调试malloc(堆越界)问题**](http://bbs.16rd.com/thread-54815-1-1.html)

[more detail about adb shell start, stop and reboot](http://blog.csdn.net/johnnylq/article/details/6401531)

[android native 内存泄露检查（libc.debug.malloc ）](http://blog.csdn.net/haima1998/article/details/51508947)