Native Crash手把手教学

MIUI系统组

Native Crash

- 概念:运行Native程序(动态库)时产生crash，统称为Native Crash
- 产生：Android在linker加载native程序的时候会注册相关的信号处理函数，当发生Crash时会通知debuggerd去获取crash相关的信息.

Android Native Crash Signals

Signal	comment	Android
SIGABRT	abort函数产生，进程非正常退出	通常有abort message
SIGSEGV	非法内存访问	内存未映射或者属性不匹配
SIGBUS	通常内存访问引起	map的文件在访问时被截短导致
SIGFPE	浮点异常，如被0除	
SIGILL	非法指令异常	
SIGSTKFLT	数学协处理器的栈异常	
SIGTRAP	一般是调试异常	

Native Crash

- Crash info
  - logcat关键字: "Fatal signal", "F DEBUG"

```
03-26 12:00:27.103   721   721 F libc    : Fatal signal 6 (SIGABRT), code -6 in tid 721 (provider@2.4-se)
03-26 12:00:27.218  7142  7142 F DEBUG   : *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
03-26 12:00:27.218  7142  7142 F DEBUG   : Build fingerprint:
'Xiaomi/polaris/polaris:8.0.0/OPR1.170623.032/8.3.26:user/release-keys'
03-26 12:00:27.218  7142  7142 F DEBUG   : Revision: '0'
03-26 12:00:27.218  7142  7142 F DEBUG   : ABI: 'arm'
03-26 12:00:27.218  7142  7142 F DEBUG   : pid: 721, tid: 721, name: provider@2.4-se  >>>
/vendor/bin/hw/android.hardware.camera.provider@2.4-service <<<
03-26 12:00:27.218  7142  7142 F DEBUG   : signal 6 (SIGABRT), code -6 (SI_TKILL), fault addr --------
03-26 12:00:27.222  7142  7142 F DEBUG   : Abort message: 'invalid pthread_t 0xe067f970 passed to libc'
03-26 12:00:27.222  7142  7142 F DEBUG   :     r0 00000000  r1 000002d1  r2 00000006  r3 00000008
03-26 12:00:27.222  7142  7142 F DEBUG   :     r4 000002d1  r5 000002d1  r6 fff3c2f0  r7 0000010c
03-26 12:00:27.222  7142  7142 F DEBUG   :     r8 e0703088  r9 00000011  sl 0000000d  fp 00000015
03-26 12:00:27.222  7142  7142 F DEBUG   :     ip 00000000  sp fff3c2e0  lr eeef99b7  pc eef2a11c  cpsr 200f0010
03-26 12:00:27.280  7142  7142 F DEBUG   :
03-26 12:00:27.280  7142  7142 F DEBUG   : backtrace:
03-26 12:00:27.280  7142  7142 F DEBUG   :     #00 pc 0004b11c  /system/lib/libc.so (tgkill+12)
03-26 12:00:27.280  7142  7142 F DEBUG   :     #01 pc 0001a9b3  /system/lib/libc.so (abort+54)
03-26 12:00:27.280  7142  7142 F DEBUG   :     #02 pc 0001efad  /system/lib/libc.so (__libc_fatal+24)
03-26 12:00:27.280  7142  7142 F DEBUG   :     #03 pc 00048703  /system/lib/libc.so
(_Z23__pthread_internal_findl+82)
03-26 12:00:27.280  7142  7142 F DEBUG   :     #04 pc 00048781  /system/lib/libc.so (pthread_join+24)
03-26 12:00:27.280  7142  7142 F DEBUG   :     #05 pc 0006556b  /vendor/lib/libssc.so (_ZN6workerD2Ev+46)
```

Native Crash

- Tombstone info
  - stack, register info, memory map, other threads info

```
stack:
        fff3c2c8  ffffffdf
        fff3c2cc  00000000
        fff3c2d0  000002d1
        fff3c2d4  6fe42b19
        fff3c2d8  000002d1
        fff3c2dc  eeef99ad  /system/lib/libc.so (abort+48)
    #00  fff3c2e0  00000000
```

栈内存信息

```
memory near r6:
    fff3c2d0 000002d1 6fe42b19 000002d1 eeef99ad  .....+.o........
    fff3c2e0 00000000 fff3c320 fff3c318 6fe42b19  .... ........+.o
    fff3c2f0 ffffffdf e067f970 e070307c fff3c508  ....p.g.|0p.....
    fff3c300 fff3c3e0 eeefdfb1 fff3c314 fff3c3e0  ................
```

寄存器内存信息

```
memory map:
    9eb80000-9ebfffff rw-         0     80000  [anon:libc_malloc]
    9ef48000-9ef48fff ---         0      1000  [anon:thread stack guard page]
    9ef49000-9f044fff rw-         0     fc000
    9f045000-9f04dfff r-x         0      9000  /vendor/lib/hw/android.hardware.graphics.mapper@2.0-impl.so (BuildId:
1a6204decfb3b7315e7bbe8b4806ef21)
```

memory map信息

```
  pid: 721, tid: 922, name: android.hardwar  >>> /vendor/bin/hw/android.hardware.camera.provider@2.4-service <<<
signal 5 (SIGTRAP), code -32763 (?), fault addr 0x39a
    r0 e81910f0  r1 00000089  r2 05894002  r3 00000000
    r4 00000000  r5 ffffffff  r6 ed92ae67  r7 000000f0
    r8 00000589  r9 00000000  sl 00000002  fp 00004000
    ip ed1ff7a0  sp ed1ff790  lr eef27d1d  pc eeef8118  cpsr 000f0010
```

other thread信息

Native Crash

- Backtrace分析
  - c++filt工具

```
#03 pc 00048703  /system/lib/libc.so (_Z23__pthread_internal_findl+82)
```

```
mi:$ c++filt _Z23__pthread_internal_findl
__pthread_internal_find(long)
```

Native Crash

- Backtrace分析
  - addr2line工具,依赖对应版本的symbols
    - http://eng.pt.miui.com/?r=eng&dir=/symbols
    - Build fingerprint: 'Xiaomi/polaris/polaris:8.0.0/OPR1.170623.032/8.3.26:user/release-keys'
    - polaris_symbols_8.3.26_8.0_6646958d40_ge491b0b.tgz

```
#03 pc 00048703  /system/lib/libc.so (__Z23__pthread_internal_findl+82)
```

```
mi:$ addr2line -Cfe symbols/system/lib/libc.so 00048703
__pthread_internal_find(long)
bionic/libc/bionic/pthread_internal.cpp:121
```

pthread_internal.cpp对应的代码

```
120   } else {
121     __libc_fatal("invalid pthread_t %p passed to libc", thread);
122   }
```

Native Crash

- Backtrace分析
  - stack工具,$(SOURCE_CODE)/development/scripts目录

```
mi@mi-OptiPlex-7040:~/mount/source/mido-7.0/development/scripts$ git diff .
diff --git a/scripts/stack b/scripts/stack
index 8e65dba61..cbd637d25 100755
--- a/scripts/stack
+++ b/scripts/stack
@@ -46,6 +46,7 @@ def main():
   try:
     options, arguments = getopt.getopt(sys.argv[1:], "",
                                        ["arch=",
+                                        "symbols=",
                                         "help"])
   except getopt.GetoptError, unused_error:
     PrintUsage()
@@ -55,6 +56,8 @@ def main():
       PrintUsage()
     elif option == "--arch":
       symbol.ARCH = value
+    elif option == "--symbols":
+      symbol.SYMBOLS_DIR = value

   if len(arguments) > 1:
     PrintUsage()
diff --git a/scripts/symbol.py b/scripts/symbol.py
index 872580860..1942f4ca6 100755
--- a/scripts/symbol.py
+++ b/scripts/symbol.py
@@ -42,7 +42,8 @@ def FindSymbolsDir():
   finally:
     os.chdir(saveddir)

-SYMBOLS_DIR = FindSymbolsDir()
+#SYMBOLS_DIR = FindSymbolsDir()
+SYMBOLS_DIR = None

 ARCH = None
```

Native Crash

- Backtrace分析
  - 用stack脚本工具解析backtrace

```
mi:$ cat tomb.txt
backtrace:
    #00 pc 0004b11c  /system/lib/libc.so (tgkill+12)
    #01 pc 0001a9b3  /system/lib/libc.so (abort+54)
    #02 pc 0001efad  /system/lib/libc.so (__libc_fatal+24)
    #03 pc 00048703  /system/lib/libc.so (_Z23__pthread_internal_findl+82)
    #04 pc 00048781  /system/lib/libc.so (pthread_join+24)
    #05 pc 0006556b  /vendor/lib/libssc.so (_ZN6workerD2Ev+46)
    #06 pc 000655ed  /vendor/lib/libssc.so (_ZN18ssc_qmi_connectionD1Ev+20)
```

```
mi:$ stack --symbols=symbols tomb.txt
Searching for native crashes in tomb.txt
Reading symbols from symbols
Using arm toolchain from: /home/mi/mount/source/mido-7.0/prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-4.9/bin/

Stack Trace:
  RELADDR   FUNCTION                                  FILE:LINE
  0004b11c  tgkill+12                                 /proc/self/cwd/bionic/libc/arch-arm/syscalls/tgkill.S:10
  0001a9b3  abort+54                                  bionic/libc/bionic/abort.cpp:49
  0001efad  __libc_fatal+24                           bionic/libc/bionic/libc_logging.cpp:621
  00048703  __pthread_internal_find(long)+82          bionic/libc/bionic/pthread_internal.cpp:121
  00048781  pthread_join+24                           bionic/libc/bionic/pthread_join.cpp:39
  0006556b  worker::~worker()+46                      vendor/qcom/proprietary/sensors-see/ssc/worker.h:55
  000655ed  ssc_qmi_connection::~ssc_qmi_connection()+20 vendor/qcom/proprietary/sensors-see/ssc/ssc_connection.cpp:145
```

Native Crash

- Backtrace汇编分析
  - objdump工具

```
#03 pc 00048703  /system/lib/libc.so ( Z23__pthread_internal_findl+82)
```

```
mi:$ arm-linux-androideabi-objdump -d symbols/system/lib/libc.so > libc.d
```

打开libc.d文件：

```
000486b0 <_Z23__pthread_internal_findl>:
   486b0:       b510            push    {r4, lr}
   486b2:       4604            mov     r4, r0
   486b4:       ee1d 0f70       mrc     15, 0, r0, cr13, cr0, {3}
   ......
   486fe:       a004            add     r0, pc, #16     ; (adr r0, 48710 <_Z23__pthread_internal_findl+0x60>)
   48700:       4621            mov     r1, r4
   48702:       f7d6 fc47       bl      1ef94 <__libc_fatal>
   48706:       bf00            nop
```

```
pthread_internal.cpp:
120     } else {
121       __libc_fatal("invalid pthread_t %p passed to libc", thread);
122     }
```

48703 != 48702 ARM状态跳转到Thumb状态的程序中执行

Native Crash

- Backtrace汇编分析
  - oatdump工具

#00 pc 0000000000083fa4 /system/framework/arm64/boot-framework.oat (offset 0x2460000)

map offset File offset

0000000000083fa4 + 0x2460000 = 0x24e3fa4

0x24e3fa4 就是oat文件中机器指令相对于文件头的偏移

那么它在oatdump中的相对地址应该是0x24e3fa4 - 0x1000 = 0x24e2fa4

  - 获取dump文件

```
mi:$ simg2img system.img raw.img
mi:$ mkdir test
mi:$ sudo mount -t ext4 raw.img test -o loop
mi:$ oatdump --oat-file=test/ramework/arm64/boot-framework.oat --output=boot-framework.oat.d
```

Native Crash

- 查看0x24e2fa4位置的代码，对应dex_pc=0x75

```
CODE:
        StackMap [native_pc=0x24e2f90] (dex_pc=0x75, native_pc_offset=0x1fc, dex_register_map_offset=0xc,
inline_info_offset=0xffffffff, register_mask=0xe200000, stack_mask=0b000000000000)
      ......
      0x024e2f90: b9016340  str w0, [x26, #352]
      0x024e2f94: aa1a03e1  mov x1, x26
      0x024e2f98: aa0003fb  mov x27, x0
      0x024e2f9c: 52800022  mov w2, #0x1
      0x024e2fa0: b9400020  ldr w0, [x1]
      0x024e2fa4: f94a2c00  ldr x0, [x0, #5208]
      0x024e2fa8: f940181e  ldr lr, [x0, #48]
      0x024e2fac: d63f03c0  blr lr
```

Native Crash

- dex_pc=0x75对应的java方法为android.view.ViewConfiguration.getScaledTouchSlop

```
756: Landroid/view/View; (offset=0x0123c658) (type_idx=1567) (StatusInitialized)
(OatClassSomeCompiled)

   ...

   8: void android.view.View.<init>(android.content.Context) (dex_method_idx=12388)

DEX CODE:

      ....

      0x0071: 7110 7d33 0800            | invoke-static {v8},
android.view.ViewConfiguration
android.view.ViewConfiguration.get(android.content.Context) // method@13181

      0x0074: 0c01                      | move-result-object v1

      0x0075: 6e10 9b33 0100            | invoke-virtual {v1}, int
android.view.ViewConfiguration.getScaledTouchSlop() // method@13211

      0x0078: 0a01                      | move-result v1

      0x0079: 5971 f319                 | iput v1, v7, I
android.view.View.mTouchSlop // field@6643

      0x007b: 6e20 2333 3700            | invoke-virtual {v7, v3}, void
android.view.View.setOverScrollMode(int) // method@13091
```

Native Crash

- Stack分析
  - ATPCS规定：arm使用r0-r3传递参数，arm64使用x0-x7传递参数。当参数超过可用的寄存器时，使用栈来传递参数。参数从右向依次压栈。
  - 看一下这个测试代码对应的汇编传递参数过程

```
int func(int a, int b, int c, int d, int e, int f)
{
    return a + b + c + d + e;
}

int main(void)
{
        func(1, 2, 3, 4, 5, 6);
        return 0;
}
```

  - 编译并反汇编

```
mi$ arm-linux-androideabi-gcc -c test.c -o test.o
mi$ arm-linux-androideabi-objdump -d test.o > test.d
```

Native Crash

- *Stack分析*

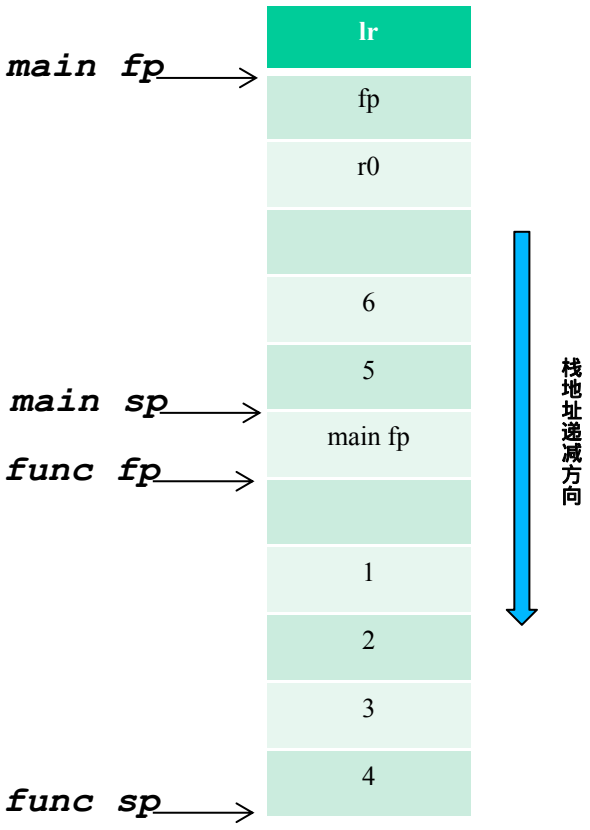
```
00000000 <func>:
   0:   e52db004        push    {fp}            ; (str fp, [sp, #-4]!)
   4:   e28db000        add     fp, sp, #0
   8:   e24dd014        sub     sp, sp, #20
   c:   e50b0008        str     r0, [fp, #-8]
  10:   e50b100c        str     r1, [fp, #-12]
  14:   e50b2010        str     r2, [fp, #-16]
  18:   e50b3014        str     r3, [fp, #-20]  ; 0xffffffec
  1c:   e51b2008        ldr     r2, [fp, #-8]
  20:   e51b300c        ldr     r3, [fp, #-12]
  24:   e0822003        add     r2, r2, r3
  28:   e51b3010        ldr     r3, [fp, #-16]
  2c:   e0822003        add     r2, r2, r3
  30:   e51b3014        ldr     r3, [fp, #-20]  ; 0xffffffec
  34:   e0822003        add     r2, r2, r3
  38:   e59b3004        ldr     r3, [fp, #4]
  3c:   e0822003        add     r2, r2, r3
  40:   e59b3008        ldr     r3, [fp, #8]
  44:   e0823003        add     r3, r2, r3
  48:   e1a00003        mov     r0, r3
  4c:   e24bd000        sub     sp, fp, #0
  50:   e49db004        pop     {fp}            ; (ldr fp, [sp], #4)
  54:   e12fff1e        bx      lr

00000058 <main>:
  58:   e92d4800        push    {fp, lr}
  5c:   e28db004        add     fp, sp, #4
  60:   e24dd010        sub     sp, sp, #16
  64:   e3a03005        mov     r3, #5
  68:   e58d3000        str     r3, [sp]
  6c:   e3a03006        mov     r3, #6
  70:   e58d3004        str     r3, [sp, #4]
  74:   e3a00001        mov     r0, #1
  78:   e3a01002        mov     r1, #2
  7c:   e3a02003        mov     r2, #3
  80:   e3a03004        mov     r3, #4
  84:   ebfffffe        bl      0 <func>
  88:   e50b0008        str     r0, [fp, #-8]
  8c:   e3a03000        mov     r3, #0
  90:   e1a00003        mov     r0, r3
  94:   e24bd004        sub     sp, fp, #4
  98:   e8bd8800        pop     {fp, pc}
```

四个参数压栈

取第五，六个参数运算

第五，六个参数压栈

传递前四个参数



Native Crash

- Tombstone Stack内存分析
  - #04 pc 00048781 /system/lib/libc.so (pthread_join+24)
  - stack 04的内容

```
#04  fff3c328  00000000
     fff3c32c  00000000
     fff3c330  00000000
     fff3c334  eeb04499  /system/lib/vndk-sp/libc++.so (_ZNSt3__15mutex6unlockEv+6)
     fff3c338  e0703048  [anon:libc_malloc]
     fff3c33c  e070307c  [anon:libc_malloc]
     fff3c340  fff3c508  [stack]
     fff3c344  fff3c3e0  [stack]
     fff3c348  ed8e385f  /vendor/lib/hw/camera.qcom.so
     fff3c34c  00000011
     fff3c350  0000000d
     fff3c354  ed57156f  /vendor/lib/libssc.so (_ZN6workerD2Ev+50)
#05  fff3c358  e0703000  [anon:libc_malloc]
```

  - 栈长度为12*4=48Bytes

```
00048768 <pthread_join>:
   48768:       e92d 47f0       stmdb   sp!, {r4, r5, r6, r7, r8, r9, sl, lr}
   4876c:       b084            sub     sp, #16
```

  - stmdb压栈,8个寄存器。接着有预留了16个bytes的空间。

Native Crash

- Tombstone Stack分析

```
487d8:      2500            movs    r5, #0
487da:      e002            b.n     487e2 <pthread_join+0x7a>
487dc:      6820            ldr     r0, [r4, #0]
487de:      6027            str     r7, [r4, #0]
487e0:      e00e            b.n     48800 <pthread_join+0x98>
487e2:      f7cc edc8       blx     15374 <__errno@plt>
487e6:      4604            mov     r4, r0
487e8:      20f0            movs    r0, #240        ; 0xf0
487ea:      4631            mov     r1, r6
487ec:      2200            movs    r2, #0
487ee:      4653            mov     r3, sl
487f0:      6827            ldr     r7, [r4, #0]
487f2:      e9cd 5500       strd    r5, r5, [sp]    栈操作
487f6:      9502            str     r5, [sp, #8]
487f8:      f7cc ee04       blx     15404 <syscall@plt>
```

Native Crash

- Tombstone Stack分析

```
mi:$ addr2line -Cfe symbols/system/lib/libc.so 487f8
__futex(void volatile*, int, int, timespec const*, int)
bionic/libc/private/bionic_futex.h:48
```

```
43 static inline __always_inline int __futex(volatile void* ftx, int op, int value,
44                                          const struct timespec* timeout,
45                                          int bitset) {
46   // Our generated syscall assembler sets errno, but our callers (pthread functions) don't want to.
47   int saved_errno = errno;
48   int result = syscall(__NR_futex, ftx, op, value, timeout, NULL, bitset);
```

```
268 #define __NR_futex             (__NR_SYSCALL_BASE+240)
```

Native Crash

- *Tombstone Stack分析*
  - *调用关系如下，pthread_join调用过程：*

```
59  // Wait for the thread to actually exit, if it hasn't already.
60  while (*tid_ptr != 0) {
61    __futex_wait(tid_ptr, tid, NULL);
62  }
64  static inline int __futex_wait(volatile void* ftx, int value, const struct timespec* timeout) {
65    return __futex(ftx, FUTEX_WAIT, value, timeout, 0);
66  }
```

```
43  static inline __always_inline int __futex(volatile void* ftx, int op, int value,
44                                            const struct timespec* timeout,
45                                            int bitset) {
46    // Our generated syscall assembler sets errno, but our callers (pthread functions) don't want to.
47    int saved_errno = errno;
48    int result = syscall(__NR_futex, ftx, op, value, timeout, NULL, bitset);
```

  - *syscall调用有7个参数，最后的三个参数要通过栈传递。其中timeout=bitset=0,栈上由地址到高地址，依次为timout, NULL, bitset。*

```
487d8:      2500            movs    r5, #0
```

```
487f2:      e9cd 5500       strd    r5, r5, [sp]
487f6:      9502            str     r5, [sp, #8]
```

```
#04  fff3c328  00000000
     fff3c32c  00000000
     fff3c330  00000000
```

Native Crash

- Tombstone Stack分析
  - 04栈上所有的数据如下：

```
#04  fff3c328  00000000                                  // timeout
     fff3c32c  00000000                                  // NULL
     fff3c330  00000000                                  // bitset
     fff3c334  eeb04499  /system/lib/vndk-sp/libc++.so (_ZNSt3__15mutex6unlockEv+6)//脏数据
     fff3c338  e0703048  [anon:libc_malloc]                  // r4
     fff3c33c  e070307c  [anon:libc_malloc]                  // r5
     fff3c340  fff3c508  [stack]                             // r6
     fff3c344  fff3c3e0  [stack]                             // r7
     fff3c348  ed8e385f  /vendor/lib/hw/camera.qcom.so       // r8
     fff3c34c  00000011                                      // r9
     fff3c350  0000000d                                      // sl
     fff3c354  ed57156f  /vendor/lib/libssc.so (_ZN6workerD2Ev+50)// lr, 函数返回地址
#05  fff3c358  e0703000  [anon:libc_malloc]
```

  - 栈上的数据必须初始化才能使用

Native Crash

- coredump抓取
  - wiki链接: http://wiki.n.miui.com/pages/viewpage.action?pageId=63620246

Native Crash

- GDB coredump
  - prebuilts/gdb/linux-x86/bin/gdb
  - 对应版本的symbols
  - coredump文件

```
mi:$ prebuilts/gdb/linux-x86/bin/gdb
(gdb) file symbols/system/bin/app_process
(gdb) core core-system_server-2016
(gdb) set sysroot symbols/
```

file	ABI: 'arm'	ABI: 'arm64'
app	app_process	app_process64
/system/bin/xxxx	/system/bin/xxxx	/system/bin/xxxx

- 常用GDB命令
  - bt/backtrace:查看调用栈
  - p/print:查看变量的值
  - ptype:查看对象的类型
  - x:查看内存的数据
  - l/list: 查看代码
  - *disass/disassemble:反汇编代码*
  - *f/frame: 查看frame信息*
  - *info locals:查看局部变量*
  - *更多信息请参考gdb help*

Native Crash

- GDB脚本
  - shell脚本
    - 遍历某一个链表，如g_thread_list
    - source加载脚本

```
(gdb) source gdb.sh
(gdb) p_glist g_thread_list
$155 = (pthread_internal_t *) 0x9ddff970
$156 = 6012
$157 = (pthread_internal_t *) 0x9f044970
$158 = 5992
......
$183 = (pthread_internal_t *) 0xdfd7f970
$184 = 0
$185 = (pthread_internal_t *) 0xe81924c0
$186 = 2268
```

```
#!/bin/sh
define p_glist
  set $list = (pthread_internal_t *)$arg0
  while $list != 0
    p $list
    p $list->tid
    set $list = $list->next
  end
end
```

  - python脚本
  - source gdb.py

```
(gdb) source gdb.py
(gdb) cgdb_help
Hello world
```

```
#gdb.py
import os
import sys
sys.path.append(os.path.dirname(os.path.realpath(__file__)))
import shadow
import gdb_engine as dbg

class cgdb_help(gdb.Command):
    def __init__(self):
        gdb.Command.__init__(self, 'cgdb_help', gdb.COMMAND_OBSCURE)

    def invoke(self, arg, from_tty):
        shadow.help()
cgdb_help()
#shadow.py
def help():
    print('Hello world')
```

Native Crash

- 案例分析
  - http://wiki.n.miui.com/pages/viewpage.action?pageId=75699707