使用LLDB调试程序（二）

Date Sat 06 December 2014 Tags lldb / debug

简述

这篇文章主要讲了如何使用lldb来处理以下三种场景:

- 死循环
- 异常断点
- 多线程

之所以挑选这3个场景，主要还是因为他们比较常见，有些不常见的其实用基础命令也能hold住。

死循环

死循环的场景发生得不太多，即便有，大部分也都能立刻发现并且改掉。但有时候发生的死循环比较难解，主要是因为程序定在一个地方不动，不

```
Enter a number (0 to quit): 2 is not prime
Enter a number (0 to quit): 2 is not prime
Enter a number (0 to quit): 2 is not prime
Enter a number (0 to quit): 2 is not prime
EnteProcess 2650 stopped        # 按了ctrl+c之后就停下来了。
* thread #1: tid = 0x6957c, 0x00007fff8fe81976 libsystem_kernel.dylib`__write_nocancel + 10, queue = 'com.apple.ma
    frame #0: 0x00007fff8fe81976 libsystem_kernel.dylib`__write_nocancel + 10
libsystem_kernel.dylib`__write_nocancel + 10:
-> 0x7fff8fe81976:  jae    0x7fff8fe81980            ; __write_nocancel + 20
   0x7fff8fe81978:  movq   %rax, %rdi
   0x7fff8fe8197b:  jmp    0x7fff8fe7cca3            ; cerror_nocancel
   0x7fff8fe81980:  retq
```

你会落到程序暂停的地方，大部分情况是一个莫名奇妙的地方，这时候就要用`finish`命令啦。`finish`命令可能要按很多次，于是你就会跳到你

```
(lldb) finish
Process 2650 stopped
* thread #1: tid = 0x6957c, 0x0000000100000ef0 a.out`main(argc=1, argv=0x00007fff5fbffa40) + 272 at a.c:23, queue
    frame #0: 0x0000000100000ef0 a.out`main(argc=1, argv=0x00007fff5fbffa40) + 272 at a.c:23
   20           if (count == 2)
   21              printf("%d is prime\n",n);
   22           else
-> 23              printf("%d is not prime\n",n);
   24        }
   25    }
```

到了这里你就可以结合上下文，找到死循环出没的点了

异常断点

开发的时候经常会遇到一个程序抛了异常然后没有人catch就挂了，我们使用XCode调试的时候都会采用 `异常断点` 的方法来确定在哪儿抛出的异常。

```
    (lldb) breakpoint set -E objc       # 这里也可以是breakpoint set -E c++或者breakpoint set -E c
    Breakpoint 2: where = libobjc.A.dylib`objc_exception_throw, address = 0x000000010b444b8a

    (lldb) continue
    Process 3772 resuming
    2014-12-06 21:52:23.066 TownShipHelper[3772:232356] -[UITableView crash]: unrecognized selector sent to instar
```

这个时候我们发现程序停住了，我们看一下调用栈：

```
(lldb) bt
* thread #1: tid = 0x38ba4, 0x000000010b444b8a libobjc.A.dylib`objc_exception_throw, queue = 'com.apple.main-threa
  * frame #0: 0x000000010b444b8a libobjc.A.dylib`objc_exception_throw
    frame #1: 0x000000010bb5b50d CoreFoundation`-[NSObject(NSObject) doesNotRecognizeSelector:] + 205
    frame #2: 0x000000010bab37fc CoreFoundation`___forwarding___ + 988
    frame #3: 0x000000010bab3398 CoreFoundation`__forwarding_prep_0___ + 120
    frame #4: 0x000000010af011e3 TownShipHelper`-[TSMenuViewController tableView](self=0x00007fe9896092b0, _cmd=0x
    frame #5: 0x000000010af0138a TownShipHelper`-[TSMenuViewController viewDidLoad](self=0x00007fe9896092b0, _cmd=
```

看一下0-3号栈帧都是系统runtime的，4号栈帧我们切过去看一下：

```
(lldb) frame select 4
frame #4: 0x000000010af011e3 TownShipHelper`-[TSMenuViewController tableView](self=0x00007fe9896092b0, _cmd=0x0000
   31           _tableView.delegate = self;
   32           _tableView.dataSource = self;
   33           [_tableView registerClass:[UITableViewCell class] forCellReuseIdentifier:@"TableViewCell"];
-> 34           [_tableView performSelector:@selector(crash) withObject:nil];
   35       }
   36       return _tableView;
   37   }
```

果然，在34行这里发现_tableView这个实例调用了一个不存在的selector，导致runtime抛出异常。 关于异常断点还有一个细节要讲，我们可以使用如：

```
    breakpoint set -E objc -w true -h true      # 表示异常在throw的时候和在catch的时候都停下。
```

多线程

调试多线程程序绝对是个揪心的问题，GDB在这方面做的也不是很好，LLDB在多线程调试方面做得是非常棒的。主要是通过 `thread` 指令做一些多

一般情况下我们会在一个地方下断点：

```
(lldb) l TSDispatchViewControllerFactory.m:43
   43       dispatch_once(&onceToken, ^{
   44           factory = [[TSDispatchViewControllerFactory alloc] init];
   45       });
   46       dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
   47           NSLog(@"this is another thread");   # 断点下在这儿，这儿是一个异步调用
   48       });
   49       return factory;
   50   }
```

跑起来之后断点就会停住：

```
frame #0: 0x0000000108084ee7 TownShipHelper`__49+[TSDispatchViewControllerFactory sharedInstance]_block_invoke_2(.
   44           factory = [[TSDispatchViewControllerFactory alloc] init];
   45       });
   46       dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
-> 47           NSLog(@"this is another thread");
   48       });
   49       return factory;
   50   }
```

此时我们看一下当前都有哪些线程：

```
(lldb) thread backtrace all
  thread #1: tid = 0x4d8a7, 0x00000001085d15d4 libobjc.A.dylib`lookUpImpOrForward + 82, queue = 'com.apple.main-th
    frame #0: 0x00000001085d15d4 libobjc.A.dylib`lookUpImpOrForward + 82
    frame #1: 0x00000001085de0d3 libobjc.A.dylib`objc_msgSend + 211
    frame #2: 0x0000000108084ce6 TownShipHelper`-[TSDispatchViewControllerFactory viewContorllerList](self=0x00007

  thread #2: tid = 0x4d8bf, 0x000000010b56c22e libsystem_kernel.dylib`kevent64 + 10, queue = 'com.apple.libdispatc
    frame #0: 0x000000010b56c22e libsystem_kernel.dylib`kevent64 + 10
    frame #1: 0x000000010b20a252 libdispatch.dylib`_dispatch_mgr_invoke + 247
    frame #2: 0x000000010b209ff7 libdispatch.dylib`_dispatch_mgr_thread + 54

* thread #3: tid = 0x4d8c0, 0x0000000108084ee7 TownShipHelper`__49+[TSDispatchViewControllerFactory sharedInstance
  * frame #0: 0x0000000108084ee7 TownShipHelper`__49+[TSDispatchViewControllerFactory sharedInstance]_block_invoke
    frame #1: 0x000000010b1fccc6 libdispatch.dylib`_dispatch_call_block_and_release + 12
    frame #2: 0x000000010b21a7f4 libdispatch.dylib`_dispatch_client_callout + 8

  thread #4: tid = 0x4d8c1, 0x000000010b56b946 libsystem_kernel.dylib`__workq_kernreturn + 10
    frame #0: 0x000000010b56b946 libsystem_kernel.dylib`__workq_kernreturn + 10
    frame #1: 0x000000010b59c757 libsystem_pthread.dylib`_pthread_wqthread + 869
    frame #2: 0x000000010b59a4a1 libsystem_pthread.dylib`start_wqthread + 13

  thread #5: tid = 0x4d8c2, 0x000000010b56b946 libsystem_kernel.dylib`__workq_kernreturn + 10
    frame #0: 0x000000010b56b946 libsystem_kernel.dylib`__workq_kernreturn + 10
    frame #1: 0x000000010b59c757 libsystem_pthread.dylib`_pthread_wqthread + 869
    frame #2: 0x000000010b59a4a1 libsystem_pthread.dylib`start_wqthread + 13

  thread #6: tid = 0x4d8c3, 0x000000010b56b946 libsystem_kernel.dylib`__workq_kernreturn + 10
    frame #0: 0x000000010b56b946 libsystem_kernel.dylib`__workq_kernreturn + 10
    frame #1: 0x000000010b59c757 libsystem_pthread.dylib`_pthread_wqthread + 869
    frame #2: 0x000000010b59a4a1 libsystem_pthread.dylib`start_wqthread + 13
```

前面打`*`的线程就是我们当前所处的线程，我们当前处在3号线程0号栈帧上。那么我们现在切换一下线程，使用`thread select`：

```
(lldb) thread select 1
(lldb) bt
```

```
* thread #1: tid = 0x4d8a7, 0x00000001085d15d4 libobjc.A.dylib`lookUpImpOrForward + 82, queue = 'com.apple.main-th
  * frame #0: 0x00000001085d15d4 libobjc.A.dylib`lookUpImpOrForward + 82
    frame #1: 0x00000001085de0d3 libobjc.A.dylib`objc_msgSend + 211
    frame #2: 0x0000000108084ce6 TownShipHelper`-[TSDispatchViewControllerFactory viewContorllerList](self=0x00007
```

我们可以看到，现在我们已经切换到1号线程了，就是这么简单。后面跳帧，看变量，都跟这篇文章里介绍的一样了。更复杂的指令可以在lldb命令

结尾

本来这篇文章应当是紧跟着这一篇的，但是因为一个异常断点的问题一直没解决给耽搁了，其实也是我的疏忽,在看 `help` 文档的时候没有注意看开
文档则非常详细，建议大家在后续的学习中，尽量多使用 `help` 指令。

关于lldb的使用方面我就先写到这里了，我们可以在评论区里讨论关于lldb更深层次的问题。

Comments

0 条评论 Casa Taloyum

按评分高低排序 ▾

开始讨论...

在 **CASA TALOYUM** 上还有……

使用DOT语言和Graphviz绘图(翻译)

6 条评论 • 2个月前

allen 项 — 直接通过文本定义转换成图形，比起自己从意识转换文本转换图形由计算机代为执行了一步关键的输出操作，学习了

pthread的各种同步机制

2 条评论 • 11天前

casatwy — 嗯，一般只要涉及到多线程的同步机制，范围就不会超出这些，关键是用对用好就行。

订阅 在您的网站上使用Disqus 隐私