使用LLDB调试程序 - Casa Taloyum

使用LLDB调试程序

Date Wed 19 November 2014 Tags lldb / debug

简述

LLDB是XCode下默认的调试工具，苹果向来都会把界面做得很好，XCode中的lldb也不例外：无缝集成，方便简单。 嗯，casa是命令行控，也不喜命令行LLDB来调试C程序。 LLDB和GDB有很多相似之处，如果你GDB玩得比较熟，那么相信你LLDB一会儿就能上手了。阅读这篇文章不需要有G我们开始了。

如果你是因为不知道怎么退出lldb才搜到这篇文章的, 直接告诉你退出命令就是`quit`, 你可以关网页去愉快地玩耍啦。

准备工作

1. 安装lldb

Ubuntu用户:

```
    sudo apt-get install lldb-3.5

    # 然后去`~/.bashrc`里面添加一句话:
    alias lldb="lldb-3.5"
    #你安装好lldb之后，要跑lldb-3.5才能调用到lldb这个程序，所以上面这句话做了一个别名，以后直接lldb就可以了。
```

Mac用户:

```
去装个xcode，然后再装个toolchain
```

Windows用户:

```
呵呵
```

2. 写一段C程序，记得编译成可执行文件

程序自己随便写一个就好了，来个if-else判断, 然后有一个随便你做什么的子函数，然后输出个helloworld就好。`编译时记得带-g参数`,这样编译器就信息。

lldb调试之旅

进入调试状态

1. 调试可执行文件

```
    lldb DebugDemo.run
    # DebugDemo.run 是编译出来的可执行文件
```

2. 调试运行时带参数的可执行文件

如果运行这个程序时是要带参数的，那么就这样：

```
    lldb -- DebugDemo.run 1 2 3
    # 等价于你在终端运行 DebugDemo.run 1 2 3
```

3. 调试某个正在运行中的进程

```
    # 先启动lldb

    ➜  ~  lldb
    (lldb)

    #----------------------------------------------------------------------

    # 然后你找一个进程的pid出来，我找的是QQ音乐
    # 你可以ps aux | grep casa （casa是我的用户名） 在这个列表里面挑一个程序的pid
    # 输入process attach --pid 你找到的pid，来把调试器挂到这个进程上去调试

    (lldb) process attach --pid 9939        # 简写命令: attach -p 9939
                                                pro att -p 9939
    Process 9939 stopped
    Executable module set to "/Applications/QQMusic.app/Contents/MacOS/QQMusic".
    Architecture set to: x86_64h-apple-macosx.
    (lldb)

    #----------------------------------------------------------------------

    # 你也可以告诉lldb你要挂在那个进程名下

    ➜  ~  lldb
    (lldb) process attach --name Safari     # 简写命令: attach -n Safari
                                                pro att -n Safari
    Process 8362 stopped
    Executable module set to "/Applications/Safari.app/Contents/MacOS/Safari".
    Architecture set to: x86_64h-apple-macosx.
    (lldb)

    #----------------------------------------------------------------------

    # 此时你就可以使用调试器命令了
    # 这个命令后面我会解释的
    (lldb) bt
    * thread #1: tid = 0x17d6e4, 0x00007fff9105152e libsystem_kernel.dylib`mach_msg_trap + 10, queue = 'com.apple.
gnal SIGSTOP
      * frame #0: 0x00007fff9105152e libsystem_kernel.dylib`mach_msg_trap + 10
        frame #1: 0x00007fff9105069f libsystem_kernel.dylib`mach_msg + 55
        frame #2: 0x00007fff8d9ffb14 CoreFoundation`__CFRunLoopServiceMachPort + 212
        frame #3: 0x00007fff8d9fefdb CoreFoundation`__CFRunLoopRun + 1371
        frame #4: 0x00007fff8d9fe838 CoreFoundation`CFRunLoopRunSpecific + 296
```

```
        frame #5: 0x00007fff8d45443f HIToolbox`RunCurrentEventLoopInMode + 235
        frame #6: 0x00007fff8d4541ba HIToolbox`ReceiveNextEventCommon + 431
        frame #7: 0x00007fff8d453ffb HIToolbox`_BlockUntilNextEventMatchingListInModeWithFilter + 71
        frame #8: 0x00007fff97b0d6d1 AppKit`_DPSNextEvent + 964
        frame #9: 0x00007fff97b0ce80 AppKit`-[NSApplication nextEventMatchingMask:untilDate:inMode:dequeue:] + 194
        frame #10: 0x00007fff97b00e23 AppKit`-[NSApplication run] + 594
        frame #11: 0x00007fff97aec2d4 AppKit`NSApplicationMain + 1832
        frame #12: 0x000000010ff2b1d2 QQMusic`main + 34
        frame #13: 0x000000010ff2b1a4 QQMusic`start + 52

    #-----------------------------------------------------------------------
```

看代码

进入到调试状态之后, lldb和gdb一样，也给了你看代码的命令: `list` 或 `l`, 但只有在编译时候带 `-g` 才能看哦

1. 使用list看代码

```
➜  DebugDemo git:(master) lldb DebugDemo.run
(lldb) target create "DebugDemo.run"
Current executable set to 'DebugDemo.run' (x86_64).
(lldb) l
   7    int main () {
   8        size_t result_array[2] = {0, 0};
   9
   10       FILE *file_handler = fopen("TestData", "r");
   11       json_error_t error;
   12
   13       json_t *root = json_loadf(file_handler, JSON_DECODE_ANY, &error);
   14       if (!root) {
   15           printf("there is an error on line:%d, test:%s\n", error.line, error.text);
   16       } else {
(lldb)
```

tips:

1. 不输入命令的时候直接按回车，就会执行上一次执行的命令。
2. 一直 `list` 到底了之后再 `list` 就没有了，这时候怎么办? `list 1` 就回到第一行了。 `l 13` 就是从第13行开始往下看10行。

2. 看其他文件的代码

如果你的这个程序编译的时候是由很多文件组成的，那么就可以使用 `list 文件名` 看其他文件的代码, 以后再执行 `list 3` 的时候，看的就是你前面

```
(lldb) list ArrayUtils.c
   1    #include "ArrayUtils.h"
   2
   3    int
   4    array_count(void *array) {
   5        return 10;
   6    }

(lldb) l 3
   3    int
   4    array_count(void *array) {
   5        return 10;
   6    }
```

3. 看某个函数的代码

```
# 直接输入函数名字即可
(lldb) list main
File: /Users/casa/playground/algorithm/leetcode/DebugDemo/src/DebugDemo.c
```

```
2    #include "ArrayUtils.h"
3    #include <jansson.h>
4
5    void yell(void);
6
7    int main () {
8        size_t result_array[2] = {0, 0};
9
10       FILE *file_handler = fopen("TestData", "r");
11       json_error_t error;
```

下断点

我们把调试器挂上程序了，也看到代码了，接下来就是找一个地方下断点，然后让程序跑起来，看看这里面到底发生了些什么～o

1. 根据文件名和行号下断点

```
(lldb) breakpoint set --file DebugDemo.c --line 10
Breakpoint 1: where = DebugDemo.run`main + 92 at DebugDemo.c:10, address = 0x0000000100000a7c
```

2. 根据函数名下断点

```
# C函数
(lldb) breakpoint set --name main

# C++类方法
(lldb) breakpoint set --method foo

# Objective-C选择器
kpoint set --selector alignLeftEdges:
```

3. 根据某个函数调用语句下断点(Objective-C比较有用)

```
# lldb有一个最小子串匹配算法，会知道应该在哪个函数那里下断点
breakpoint set -n "-[SKTGraphicView alignLeftEdges:]"
```

4. 一个小技巧

你可以通过设置命令的别名来简化上面的命令

```
# 比如下面的这条命令
(lldb) breakpoint set --file DebugDemo.c --line 10

# 你就可以写这样的别名
(lldb) command alias bfl breakpoint set -f %1 -l %2

# 使用的时候就像这样就好了
(lldb) bfl DebugDemo.c 10
```

5. 查看断点列表、启用/禁用断点、删除断点

```
#----------------------------------------------------------------------

# 查看断点列表

(lldb) breakpoint list

Current breakpoints:
```

```
    1: file = 'DebugDemo.c', line = 10, locations = 1
      1.1: where = DebugDemo.run`main + 92 at DebugDemo.c:10, address = DebugDemo.run[0x0000000100000a7c], unresol

    2: name = 'main', locations = 1
      2.1: where = DebugDemo.run`main + 68 at DebugDemo.c:8, address = DebugDemo.run[0x0000000100000a64], unresolv

    #----------------------------------------------------------------------

    # 禁用断点
    # 根据上面查看断点列表的时候的序号来操作断点
    (lldb) breakpoint disable 2
    1 breakpoints disabled.
    (lldb) breakpoint list
    Current breakpoints:
    1: file = 'DebugDemo.c', line = 10, locations = 1
      1.1: where = DebugDemo.run`main + 92 at DebugDemo.c:10, address = DebugDemo.run[0x0000000100000a7c], unresol

    2: name = 'main', locations = 1 Options: disabled
      2.1: where = DebugDemo.run`main + 68 at DebugDemo.c:8, address = DebugDemo.run[0x0000000100000a64], unresolv

    #----------------------------------------------------------------------

    # 启用断点
    (lldb) breakpoint enable 2
    1 breakpoints enabled.
    (lldb) breakpoint list
    Current breakpoints:
    1: file = 'DebugDemo.c', line = 10, locations = 1
      1.1: where = DebugDemo.run`main + 92 at DebugDemo.c:10, address = DebugDemo.run[0x0000000100000a7c], unresol

    2: name = 'main', locations = 1
      2.1: where = DebugDemo.run`main + 68 at DebugDemo.c:8, address = DebugDemo.run[0x0000000100000a64], unresolv

    #----------------------------------------------------------------------

    # 删除断点
    (lldb) breakpoint delete 1
    1 breakpoints deleted; 0 breakpoint locations disabled.
    (lldb) breakpoint list
    Current breakpoints:
    2: name = 'main', locations = 1
      2.1: where = DebugDemo.run`main + 68 at DebugDemo.c:8, address = DebugDemo.run[0x0000000100000a64], unresolv

    #----------------------------------------------------------------------
```

运行环境操作

1. 启动

OK, 我们前面已经下好断点了，现在就要启动这个程序了！前面留了一个断点是断在main函数的哈。

```
    # run命令就是启动程序
    (lldb) run
    Process 11500 launched: '/Users/casa/Playground/algorithm/leetcode/DebugDemo/DebugDemo.run' (x86_64)
    Process 11500 stopped   # 这里执行到断点了
    * thread #1: tid = 0x1af357, 0x0000000100000a64 DebugDemo.run`main + 68 at DebugDemo.c:8, queue = 'com.apple.m
akpoint 2.1
        frame #0: 0x0000000100000a64 DebugDemo.run`main + 68 at DebugDemo.c:8
       5    void yell(void);
       6
       7    int main () {
    -> 8        size_t result_array[2] = {0, 0};  # 这一行前面的箭头表示调试器在这里停住了
       9
       10       FILE *file_handler = fopen("TestData", "r");
       11       json_error_t error;
```

2. 下一步、步入、步出、继续执行

```
    # 下一步 (next 或 n)
    (lldb) next
    Process 11500 stopped
    * thread #1: tid = 0x1af357, 0x0000000100000a7c DebugDemo.run`main + 92 at DebugDemo.c:10, queue = 'com.apple.
ep over
        frame #0: 0x0000000100000a7c DebugDemo.run`main + 92 at DebugDemo.c:10
       7    int main () {
       8        size_t result_array[2] = {0, 0};
       9
    -> 10       FILE *file_handler = fopen("TestData", "r");
      11        json_error_t error;
      12
      13        json_t *root = json_loadf(file_handler, JSON_DECODE_ANY, &error);

    #-----------------------------------------------------------------------

    # 步入(step 或 s)
    Process 11668 stopped
    * thread #1: tid = 0x1b4e9d, 0x0000000100000c06 DebugDemo.run`main + 486 at DebugDemo.c:29, queue = 'com.apple
reakpoint 3.1
        frame #0: 0x0000000100000c06 DebugDemo.run`main + 486 at DebugDemo.c:29
       26               printf("input array is %"JSON_INTEGER_FORMAT"\n", json_integer_value(item));
       27           }
       28
    -> 29           yell();
       30
       31           json_array_foreach(input_array, index, item) {
       32

    (lldb) step
    Process 11668 stopped
    * thread #1: tid = 0x1b4e9d, 0x0000000100000e1f DebugDemo.run`yell + 15 at DebugDemo.c:57, queue = 'com.apple.
ep in
        frame #0: 0x0000000100000e1f DebugDemo.run`yell + 15 at DebugDemo.c:57
       54
       55   void yell()
       56   {
    -> 57       printf("here i am yelling\n");
       58   }

    #-----------------------------------------------------------------------

    # 步出(finish)
    (lldb) finish
    here i am yelling
    Process 11668 stopped
    * thread #1: tid = 0x1b4e9d, 0x0000000100000c0b DebugDemo.run`main + 491 at DebugDemo.c:31, queue = 'com.apple
tep out
        frame #0: 0x0000000100000c0b DebugDemo.run`main + 491 at DebugDemo.c:31
       28
       29           yell();
       30
    -> 31           json_array_foreach(input_array, index, item) {
       32
       33               result_array[0] = index;
       34               json_int_t value = json_integer_value(item);

    #-----------------------------------------------------------------------

    # 继续执行到下一个断点停，后面没有断点的话就跑完了（continue 或 c）
    (lldb) continue
    Process 11668 resuming
    result is 2, 3
    Process 11668 exited with status = 0 (0x00000000)

    #-----------------------------------------------------------------------
```

3. 查看变量、跳帧查看变量

```
    # 使用po或p，po一般用来输出指针指向的那个对象，p一般用来输出基础变量。普通数组两者都可用
    Process 11717 stopped
    * thread #1: tid = 0x1b7afc, 0x0000000100000a7c DebugDemo.run`main + 92 at DebugDemo.c:10, queue = 'com.apple.
ep over
        frame #0: 0x0000000100000a7c DebugDemo.run`main + 92 at DebugDemo.c:10
       7    int main () {
       8        size_t result_array[2] = {0, 0};
       9
    -> 10       FILE *file_handler = fopen("TestData", "r");
       11       json_error_t error;
       12
       13       json_t *root = json_loadf(file_handler, JSON_DECODE_ANY, &error);

    (lldb) po result_array
     ([0] = 0, [1] = 0)

    (lldb) p result_array
    (size_t [2]) $2 = ([0] = 0, [1] = 0)

    #-----------------------------------------------------------------------------

    # 查看所有帧(bt)
    (lldb) bt
    * thread #1: tid = 0x1bb7dc, 0x0000000100000e1f DebugDemo.run`yell + 15 at DebugDemo.c:57, queue = 'com.apple.
ep in
      * frame #0: 0x0000000100000e1f DebugDemo.run`yell + 15 at DebugDemo.c:57
        frame #1: 0x0000000100000c0b DebugDemo.run`main + 491 at DebugDemo.c:29
        frame #2: 0x00007fff99d175c9 libdyld.dylib`start + 1

    #-----------------------------------------------------------------------------

    # 跳帧（frame select）
    (lldb) frame select 1
    frame #1: 0x0000000100000c0b DebugDemo.run`main + 491 at DebugDemo.c:29
       26               printf("input array is %"JSON_INTEGER_FORMAT"\n", json_integer_value(item));
       27           }
       28
    -> 29           yell();
       30
       31           json_array_foreach(input_array, index, item) {
       32

    #-----------------------------------------------------------------------------

    # 查看当前帧中所有变量的值（frame variable）
    (lldb) frame variable

    (size_t [2]) result_array = ([0] = 0, [1] = 0)
    (FILE *) file_handler = 0x00007fff7cfdd070
    (json_error_t) error = (line = 0, column = 0, position = 0, source = "", text = "?)
    (json_t *) root = 0x0000000000000000

    #-----------------------------------------------------------------------------
```

结束

这只是这篇文章结束了，还有watchpoints这一门没有写。不过这篇文章里面的东西知道了以后，调试个程序问题就不大。 想要进阶的同学可以去看