LLDB 调试命令使用指南

作者：Hanseltu
原文链接：https://ld246.com/article/1556200452086
来源网站：链滴

原文链接 LLDB调试命令使用指南

前言

LLDB Debugger (LLDB) 是一个开源、底层调试器(low level debugger)，具有REPL (Read-Eval-Print Loop，交互式解释器)、C++和Python插件，位于Xcode窗口底部控制台中，也有其他IDE加入了LLB调试器，如CLion，当然其也可以在terminal中使用。搜罗了一圈，发现大多数教程还是可视化的使，本文就命令行使用LLDB调试程序做一个总结。

调试前

需要安装lldb调试器，如果你安装了llvm编译环境，lldb是其自带的工具，就不用安装了。

开始或结束调试

特别注意，需调试的程序一定要加上 -g 参数进行编译，否则调试时无法看到源码。

lldb有两种进入方式，本例使用方法一（假设可执行程序名为 toy，参数为 example1）

方式一，直接运行 lldb toy example 进入调试环境

方式二，先运行 lldb，再通过 file toy进入调试环境

除此之外，还可调试正在运行时的程序，此时需要找到此程序的PID或者进程名，再使用

(lldb) process attach --pid 9939

或者

(lldb) process attach --name Safari

进入调试环境。

退出lldb环境，输入 quit或者 exit即可。

查看代码

用 list查看

和gdb一样，使用 list查看代码，这里有两个小技巧

- 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。
- 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行始往下看10行。

看其他文件的代码

如果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，看的就是你前面设置的文件名的第三行。

看某个函数的代码

直接输入某个函数的名字即可。

演示如下：

设置断点

根据文件名和行号下断点

```
(lldb) breakpoint set --file toy.cpp --line 10
```

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
```

根据某个函数调用语句下断点

```
# 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:]"
```

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 test.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 test.c 10
```

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
```

```
# 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

# 启用断点
(lldb) breakpoint enable 2

# 删除断点
(lldb) breakpoint delete 1
```

演示如下：

调试环境操作

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！前面留一个断点是断在main函数。

```
# run命令就是启动程序
(lldb) run
```

下一步、步入、步出、继续执行

```
# 下一步 (next 或 n)
(lldb) next

# 步入(step 或 s)
(lldb) step

# 步出(finish)
(lldb) finish

# 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
```

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
(lldb) po result_array

 # 查看所有帧(bt)
(lldb) bt

# 跳帧（frame select）
(lldb) frame select 1

# 查看当前帧中所有变量的值（frame variable）
```

```
(lldb) frame variable
```

修改变量或调用函数

使用 expression 命令可以在调试过程中修改变量的值，或者执行函数。

修改变量值

```
(lldb) expression a
(int) $0 = 9
(lldb) frame variable a
(int) a = 9
(lldb) expression a=10
(int) $1 = 10
(lldb) frame variable a
(int) a = 10
(lldb)
```

调用函数

```
(lldb) expression printf("execute function %i",a)
(int) $2 = 19
execute function 10
```

对于执行结果都会自动保存，以备他用。

线程控制

在加载进程开始调试后，当执行到设置断点时，可以使用 thread 命令控制代码的执行。

线程继续执行

```
(lldb) thread continue
Resuming thread 0x2c03 in process 46915
Resuming process 46915
(lldb)
```

线程进入、单步执行或跳出

```
(lldb) thread step-in
(lldb) thread step-over
(lldb) thread step-out
```

单步指令的执行

```
(lldb) thread step-inst
(lldb) thread step-over-ins
```

执行指定代码行数直到退出当前帧

```
(lldb) thread until 100
```

查看线程列表，第一个线程为当前执行线程

```
(lldb) thread list
Process 46915 state is Stopped
* thread #1: tid = 0x2c03, 0x00007fff85cac76a, where = libSystem.B.dylib`__getdirentries64 +
0, stop reason = signal = SIGSTOP, queue = com.apple.main-thread
  thread #2: tid = 0x2e03, 0x0000
```

查看当前线程栈

```
(lldb) thread backtrace
thread #1: tid = 0x2c03, stop reason = breakpoint 1.1, queue = com.apple.main-thread
 frame #0: 0x0000000100010d5b, where = Sketch`-[SKTGraphicView alignLeftEdges:] + 33 at
Projects/Sketch/SKTGraphicView.m:1405
 frame #1: 0x00007fff8602d152, where = AppKit`-[NSApplication sendAction:to:from:] + 95
 frame #2: 0
```

查看所有线程的调用栈

```
(lldb) thread backtrace all
```

设置当前线程

```
(lldb) thread select 2
```

部分参考：

使用LLDB调试程序

LLDB 常用命令