

使用 lldb

1. 先写一个简单的程序，用 lldb 来调试它

主要调试的是 C/C++ 的程序。要调试 C/C++ 的程序，首先在编译时，我们必须要把调试信息加到可执行文件中。使用编译器（cc/gcc/g++）的 **-g** 参数可以做到这一点。如：

```
$gcc -g -Wall hello.c -o hello
```

如果没有 **-g**，你将看不见程序的函数名、变量名，所代替的全是运行时的内存地址。

Lldb 进行调试的是可执行文件，而不是如“.c”的源代码，因此，需要先通过 Gcc 编译生成可执行文件才能用 Lldb 进行调试。

2. Lldb 的帮助

终端中输入

lldb

输入

help

help 命令只是列出 lldb 的命令种类，如果要看种类中的命令，可以使用 **help <class>** 命令，**help breakpoints**

查看设置断点的所有命令。也可以直接 **help <command>** 来查看命令的帮助。

3. 调试可执行文件

lldb test

在 lldb 的启动画面中指出了 lldb 的版本号、使用的库文件等信息，接下来就进入了由“(lldb)”开头的命令行界面了。

4. lldb 常用命令

在 lldb 的命令中都可使用缩略形式的命令，如“l”代便“list”，“b”代表“breakpoint”，“p”代表“print”等，也可使用“help”命令查看帮助信息。

（1）查看文件

list：简记为 **l**，其作用就是列出程序的源代码，默认每次显示 10 行。

- **list 行号**：将显示当前文件从“行号”的开始 10 行代码，如：**list 12**

- **list** 函数名: 将显示“函数名”为中心的前后十行代码, 如: **list main**
- **list** : 不带参数, 将接着上一次 **list** 命令的, 输出下边的内容。

(2) 设置断点

设置断点是调试程序中是一个非常重要的手段, 它可以使程序到一定位置暂停它的运行。因此, 程序员在该位置处可以方便地查看变量的值、堆栈情况等, 从而找出代码的症结所在。

设置断点的方式:

- **b n** : 在第 **n** 行处设置断点
- **b func**: 在函数 **func()** 的入口处设置断点
- **breakpoint list -f**: 查看所有断点信息
- **breakpoint delete 断点号 n**: 删除第 **n** 个断点
- **breakpoint disable 断点号 n**: 暂停第 **n** 个断点
- **breakpoint enable 断点号 n**: 开启第 **n** 个断点

(3) 运行程序

run: 简记为 **r**, 其作用是运行程序, 当遇到断点后, 程序会在断点处停止运行, 等待用户输入下一步的命令。

(lldb) **r**: 运行程序

(4) 打印变量的值

- **print 表达式**: 简记为 **p**, 其中“表达式”可以是任何当前正在被测试程序的有效表达式, 比如当前正在调试 C 语言的程序, 那么“表达式”可以是任何 C 语言的有效表达式, 包括数字, 变量甚至是函数调用。
 - **print a**: 将显示整数 **a** 的值
 - **print ++a**: 将把 **a** 中的值加 1, 并显示出来
 - **print name**: 将显示字符串 **name** 的值
 - **print Lldb_test(22)**: 将以整数 22 作为参数调用 **Lldb_test()** 函数
 - **print Lldb_test(a)**: 将以变量 **a** 作为参数调用 **Lldb_test()** 函数

小技巧	lldb 在显示变量值时都会在对值之前加上“\$N”标记, 它是当前变量值的引用标记, 所以以后若想再次引用此变量就可以直接写作“\$N”, 而无需写冗长的变量名。
-----	--

(5) 单步运行

单步运行可以使用命令“n”（next）或“s”（step），它们之间的区别在于：若有函数调用的时候，“s”会进入该函数（一般只进入用户自定义函数），执行函数体里的内容，而“n”不会进入该函数。因此，“s”就类似于 VC 等工具中的“step in”，“n”类似与 VC 等工具中的“step over”。

使用“n”后，程序显示函数 sum 的运行结果并向下执行，而使用“s”后则进入到 sum 函数之中单步运行。

（6）恢复程序运行

在查看完所需变量及堆栈情况后，就可以使用命令“c”（continue）恢复程序的正常运行了。这时，它会把剩余还未执行的程序执行完，并显示剩余程序中的执行结果。以下是之前使用“n”命令恢复后的执行结果：

```
(lldb) c
```

```
Continuing.
```

```
The sum of 1-50 is :1275
```

```
Program exited with code 031.
```

可以看出，程序在运行完后退出，之后程序处于“停止状态”。

小知识

在 lldb 中，程序的运行状态有“运行”、“暂停”和“停止”三种，其中“暂停”状态为程序遇到了断点或观察点之类的，程序暂时停止运行，而此时函数的地址、函数参数、函数内的局部变量都会被压入“栈”（Stack）中。故在这种状态下可以查看函数的变量值等各种属性。但在函数处于“停止”状态之后，“栈”就会自动撤销，它也就无法查看各种信息了。

（7）查看表达式的值

- display 表达式：在单步运行时将非常有用，使用 display 命令设置一个表达式后，它将在每次单步进行指令后，紧接着输出被设置的表达式及值。
- 如：在循环处设置一个断点，单步运行 n
- display m
- undisplay 序号 去除显示

（9）查看栈信息

当程序被停住了，你需要做的第一件事就是查看程序是在哪里停住的。当你的程序调用了一个函数，函数的地址，函数参数，函数内的局部变量都会被压入“栈”（Stack）中。你可以

用 ILDB 命令来查看当前的栈帧中的信息。

下面是一些查看函数调用栈信息的 lldb 命令：

backtrace

bt

打印当前的函数调用栈的所有信息。如：

```
(Lldb) bt
#0 func (n=250) at tst.c:6
#1 0x08048524 in main (argc=1, argv=0xbffff674) at tst.c:30
#2 0x400409ed in __libc_start_main () from /lib/libc.so.6
```

从上可以看出函数的调用栈信息：__libc_start_main --> main() --> func()

frame info：查看当前栈帧的信息。

frame select：栈帧序号 选择栈帧

frame variable：查看当前栈帧的局部变量值。

(10) 输出格式

一般来说，lldb 会根据变量的类型输出变量的值。但你也可以自定义 lldb 的输出的格式。

- x 按十六进制格式显示变量。
- d 按十进制格式显示变量。
- u 按十六进制格式显示无符号整型。
- o 按八进制格式显示变量。
- t 按二进制格式显示变量。
- a 按十六进制格式显示变量。
- c 按字符格式显示变量。
- f 按浮点数格式显示变量。

```
(Lldb) p i
$21 = 101
(Lldb) p/a i
$22 = 0x65
(Lldb) p/c i
$23 = 101 'e'
```

(11) 查看内存

你可以使用 **examine** 命令（简写是 **x**）来查看内存地址中的值。**x** 命令的语法如下所示：

```
x/<n/f/u> <addr>
```

n、**f**、**u** 是可选的参数。

- **n** 是一个正整数，表示显示内存的长度，也就是说从当前地址向后显示几个地址的内容。
- **f** 表示显示的格式，参见上面。如果地址所指的是字符串，那么格式可以是 **s**，如果地址是指令地址，那么格式可以是 **i**。
- **u** 表示从当前地址往后请求的字节数，如果不指定的话，LLDB 默认是 4 个 bytes。**u** 参数可以用下面的字符来代替，**b** 表示单字节，**h** 表示双字节，**w** 表示四字节，**g** 表示八字节。当我们指定了字节长度后，LLDB 会从指定内存地址开始，读写指定字节，并将其当作一个值取出来。

<addr> 表示一个内存地址。

n/f/u 三个参数可以一起使用。例如：

命令：**x/3ah 0x54320** 表示，从内存地址 **0x54320** 读取内容，**h** 表示以双字节为一个单位，**3** 表示三个单位，**a** 表示按十六进制显示。

(10) 其它命令

- **finish**：运行程序，直到当前函数完成返回，并打印函数返回时的堆栈地址和返回值及参数值等信息。

当然，lldb 的功能远不止这些，包括多进程/多线程/信号/远程调试等功能在这里均没有提及，有需要的读者可以参考其它信息。