lldb 调试器的简单调试

当我们打了一个断点后，程序运行到断点，停止时，在控制台会显示出一个(lldb) 我们平时都不怎么去关注他，但他其实是Xcode 工程中默认的调试器。

今天就先简单使用以下lldb

主要2个指令 print 简写为 p

另一个是 **expression 简写为 e**

用来打印基本类型

平时我们调试的时候，想知道运行到断点后，一个变量的值一般喜欢用。NSLog()

print 变量名 这种方式，可是直接在控制台打印变量的值

简写为 p 变量名

也可以使用 po 来打印一个变量

大家可能还注意到了，print结果中有个 $0。po 的结果中没有。

实际上$0 是lldb 提供给你的变量。试试 print $0 + 7，你会看到 106。任何以美元符开头的东西都是存在于 LLDB 的命名空间的，它们是为了帮助你进行调试而存在的。

print 和po ，区别不仅仅是有没有lldb提供的变量

还有在对象上的输出区别。

之前我们学习过，我们想NSLog 一个对象时，打印的是<Preson: 0x100103530> 类名+对象地址。这明显不是我们想要的输出，我们想看到的输出是，对象中的值，所以我们会重写类的description方法来返回我们想要的输出。

print 和po 的区别 也在这方面

当对象重写过description 方法后，输出的就会是description方法里返回的值

但是我们一般写类的时候除非特殊需要一般不去写description方法，

大家到现在写的类，都没写过description方法吧

那我们想知道 对象中变量的属性值时应该怎么变呢。

这时候可以使用 p 变量名.属性名来查看

对于类中的基本类型适用，类中的对象也适用

所以还是print 方法比较强大

但是，光打印指也没什么意思。当想改变一个值怎么办？

这时候可以使用

**expression 简写为 e**

**第一个例子：**

**Macintosh HD:Users:dwt:Desktop:Snip20150322_3.png**

从左到右，四个按钮分别是：1.continue，2.step over，3.step into，4.step out。

continue 按钮，会取消程序的暂停，允许程序正常执行 (要么一直执行下去，要么到达下一个断点)。在 LLDB 中，你可以使用continue 命令来达到同样的效果，也可以缩写为 c。

step over 按钮，会以黑盒的方式执行一行代码。如果所在这行代码是一个函数调用，那么就不会跳进这个函数，而是会执行这个函数，然后继续。LLDB 则可以使用next，或者 n 命令。

注：黑盒方式，对应的是白盒方式，软件测试中说的比较多。黑盒就是不关心代码实现， 白盒是看代码实现。在这里就是，不进入调用的函数看调用的函数代码。

3.如果你确实想跳进一个函数调用来调试或者检查程序的执行情况，那就用第三个按钮，step in，或者在LLDB中使用step，或者 s 命令。注意，当前行不是函数调用时，next 和 step 效果是一样的。

大多数人知道 c，n 和 s，但是其实还有第四个按钮，step out。如果你曾经不小心跳进一个函数，但实际上你想跳过它，常见的反应是重复的运行 n 直到函数返回。其实这种情况，step out 按钮是你的救世主。它会继续执行到下一个返回语句 (直到一个堆栈帧结束) 然后再次停止。(指令为**finish**)

**Macintosh HD:Users:dwt:Desktop:Snip20150322_3.png**

对应 简写为 c n s finish

**frame info** 会告诉你当前的行数和源码文件

第二个例子修改时时修改下 headerView中的数据

断点只有打在 return headerView; 上才有用

断点在 headerView.friendGroup = self.friendGroups[section];上是没有效果的

当不知道获得的对象中有那些属性时可以使用

**image lookup --type RMFriend 看类的头文件**

image :概念

lookup :查看

--type:类型

先通过

使用 **e headerView.friendGroup.name = @"123"**

提醒下

断点只有打在 return headerView; 上才有用

断点在 headerView.friendGroup = self.friendGroups[section];上是没有效果的

一下错误

如果想直接修改 self.friendGroups[section]数组中的值

**po self.friendGroups[section]**

**查看类型**

**得到**

<HMFriendGroup: 0x7face3c284b0>

**p self.friendGroups[section] 查看时**

(id) $1 = 0x00007face3c284b0

**image lookup --type** HMFriendGroup**看类的头文件**

**知道**HMFriendGroup里的属性后。就可以查看 [**group.friends[section] 属性名]**

**po [group.friends[section] name] 不出错**

**p [self.friendGroups[section] name]**

**打印时出错**

OC中从数组取出对象后，对象都是 id 类型

因为是id 类型，所以 **p group.friends[0]会出错**

**需要高度lldb group.friends[section] 的类型**

使用**p [(HMFriendGroup \*)self.friendGroups[section] name]**

查看完成后

**e [(HMFriendGroup \*)self.friendGroups[section] name]=@"123"**

**修改会报错。**

**OC中 不能直接改变数组中对象的值 ，和frame有点像**

**补充一个：**

**frame variable \*变量名 看到继承，属性等信息**

例如 ：问题二 的footView

接下来， 之前说过

**p self.friendGroups[section] 查看时**

(id) $1 = 0x00007face3c284b0

的$1 是lldb提供给我们的变量

lldb中可以像在 OC 语言中用 int a = 0 来声明一个变量一样，你也可以在 LLDB 中做同样的事情。不过为了区分变量，必须以美元符开头。

例如：

e NSArray \*$array = @[ @"Saturday", @"Sunday", @"Monday" ]

注意：P是不行的 一定是e

p [$array count]

可以和在程序里的变量一样输出，改变。

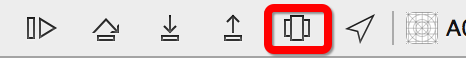
可以在程序输出之前改变其中的值

虽然看着很厉害的样子。不过真的用这种方式调试，还不如暂停程序改下代码。因为没有提示，没有报错。

鼠标悬停在变量上时也可以看到值

不过还是有点用的。比如玩一下

**更新UI**



运行后暂停程序

有了上面的输出，我们可以获取这个 view：

3D图对应的是

po [[[UIApplication sharedApplication] keyWindow] recursiveDescription]

XCode 帮们转换成图形界面了。XCODE 6开始有的。XCode5 的时候还没有

注意：一定要用 id 来设置

(lldb) e id $myView = (id)0x7f82b1d01fd0

这里相当于 新建了一个 类型为id 变量名 $myView 的变量

值为我们指定的控件

然后在调试器中改变它的背景色

(lldb) e (void)[$myView setBackgroundColor:[UIColor redColor]]

但是只有程序继续运行之后才会看到界面的变化。因为改变的内容必须被发送到渲染服务中，然后显示才会被更新。

如果不想运行程序，就能看见效果。

(lldb) e (void)[CATransaction flush]

即使你仍然在调试器中，UI 也会在模拟器或者真机上实时更新。

感觉好像还是用处不大的样子。

那在看个有用的

**查找按钮的 target**

**从UI上抓取一个控件，**想知道，按钮按下的时候谁会接收到按钮发出的 action （ 不需要看代码，一行代码都不用看）

e id $myBtn = (id)0x7f82b1d01fd0

po [$myBtn  allTargets] 显示监听btn action的类

po [$myBtn actionsForTarget:(id)0x7fb58bd2e240 forControlEvent:0] actionsForTarget：获得的类地址

这个还是有点用的哦~~

得到了action 的方法名字之后，那就要去看这个方法了。

比如要去修改这个方法里的代码，时。

那还是需要去看代码来定位是吗？

可我说过的哦。一行代码都不看。

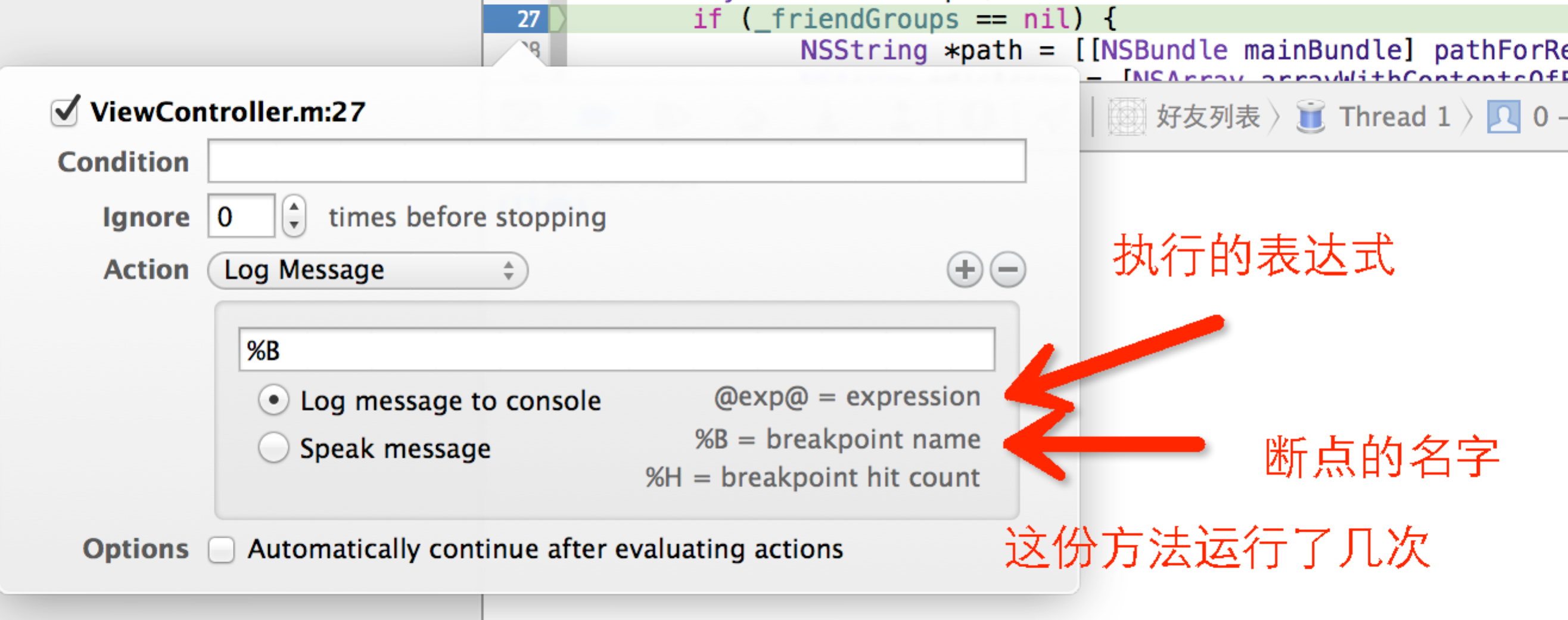
那不看代码怎么定位呢~

就需要用到符号断点(Symbolic breakpoint)

-[类名 方法名] 运行程序

就可以定位到action 这个方法了

之后讲解下断点



变量名 @group.name@ %B 断点名字 %H 断点执行的次数

讲解

