iOS-----用LLDB调试，让移动开发更简单（二）

标签：

image lookup –address

当我们有一个地址，想查找这个地址具体对应的文件位置，可以使用image lookup --address，简写为image lookup -a e.g: 当我们发生一个crash

```
2015-12-17 14:51:06.301 TLLDB[25086:246169] *** Terminating app due to uncaught exception ‘NSRangeException‘, reason: ‘*** -[__NSArray0 objectAtIndex:]: index 1 beyond bounds for empty NSArray‘

*** First throw call stack:

(

    0   CoreFoundation                      0x000000010accde65 __exceptionPreprocess + 165

    1   libobjc.A.dylib                     0x000000010a746deb objc_exception_throw + 48

    2   CoreFoundation                      0x000000010ac7c395 -[__NSArray0 objectAtIndex:] + 101

    3   TLLDB                               0x000000010a1c3e36 -[ViewController viewDidLoad] + 86

    4   UIKit                               0x000000010b210f98 -[UIViewController loadViewIfRequired] + 1198

    5   UIKit                               0x000000010b2112e7 -[UIViewController view] + 27
```

我们可以看到是由于-[__NSArray0 objectAtIndex:]:超出边界而导致的crash，但是objectAtIndex:的代码到底在哪儿呢?

```
(lldb) image lookup -a 0x000000010a1c3e36

      Address: TLLDB[0x0000000100000e36] (TLLDB.__TEXT.__text + 246)

      Summary: TLLDB`-[ViewController viewDidLoad] + 86 at ViewController.m:32
```

根据0x000000010a1c3e36 -[ViewController viewDidLoad]里面的地址，使用image lookup --address查找，我们可以看到代码位置在ViewController.m里面的32行

image lookup –name

当我们想查找一个方法或者符号的信息，比如所在文件位置等。我们可以使用image lookup --name，简写为image lookup -n。

e.g: 刚刚遇到的真问题，某个第三方SDK用了一个我们项目里原有的第三方库，库里面对NSDictionary添加了category。也就是有2个class对NSDictionary添加了名字相同的category，项目中调用自己的category的地方实际走到了第三方SDK里面去了。最大的问题是，这2个同名category方法行为并不一致，导致出现bug

现在问题来了，怎么寻找到底是哪个第三方SDK？方法完全包在.a里面。

其实只需使用image lookup -n即可：

```
(lldb) image lookup -n dictionaryWithXMLString:
2 matches found in /Users/jiangliancheng/Library/Developer/Xcode/DerivedData/VideoIphone-aivsnqmlwjhxapdlvmdmrubbdxpq/Build/Products/Debug-iphoneos/BaiduIphoneVideo.app/BaiduIphoneVideo:
    Address: BaiduIphoneVideo[0x00533a7c] (BaiduIphoneVideo.__TEXT.__text + 5414908)
    Summary: BaiduIphoneVideo`+[NSDictionary(SAPIXmlDictionary) dictionaryWithXMLString:] at XmlDictionary.m
     Module: file = "/Users/jiangliancheng/Library/Developer/Xcode/DerivedData/VideoIphone-aivsnqmlwjhxapdlvmdmrubbdxpq/Build/Products/Debug-iphoneos/BaiduIphoneVideo.app/BaiduIphoneVideo", arch = "armv7"
   CompileUnit: id = {0x00000000}, file = "/Users/jiangliancheng/Development/Work/iOS_ShareLib/SharedLib/Srvcs/BDPassport4iOS/BDPassport4iOS/SAPI/Extensive/ThirdParty/XMLDictionary/XmlDictionary.m", language = "Objective-C"
      Function: id = {0x23500000756}, name = "+[NSDictionary(SAPIXmlDictionary) dictionaryWithXMLString:]", range = [0x005a6a7c-0x005a6b02)
      FuncType: id = {0x23500000756}, decl = XmlDictionary.m:189, clang_type = "NSDictionary * (NSString *)"
        Blocks: id = {0x23500000756}, range = [0x005a6a7c-0x005a6b02)
     LineEntry: [0x005a6a7c-0x005a6a98): /Users/jiangliancheng/Development/Work/iOS_ShareLib/SharedLib/Srvcs/BDPassport4iOS/BDPa
```

ssport4iOS/SAPI/Extensive/ThirdParty/XMLDictionary/XmlDictionary.m

Symbol: id = {0x0000f2d5}, range = [0x005a6a7c-0x005a6b04), name="+
[NSDictionary(SAPIXmlDictionary) dictionaryWithXMLString:]"

Variable: id = {0x23500000771}, name = "self", type = "Class", location = [sp+32], decl =

Variable: id = {0x2350000077e}, name = "_cmd", type = "SEL", location = [sp+28], decl =

Variable: id = {0x2350000078b}, name = "string", type = "NSString *", location = [sp+24], decl
= XmlDictionary.m:189

Variable: id = {0x23500000799}, name = "data", type = "NSData *", location = [sp+20], decl =
XmlDictionary.m:192

Address: BaiduIphoneVideo[0x012ee160] (BaiduIphoneVideo.__TEXT.__text + 19810016)

Summary: BaiduIphoneVideo`+[NSDictionary(XMLDictionary) dictionaryWithXMLString:] at
XMLDictionary.m

Module: file = "/Users/jiangliancheng/Library/Developer/Xcode/DerivedData/VideoIphone-
aivsnqmlwjhxapdlvmdmrubbdxpq/Build/Products/Debug-
iphoneos/BaiduIphoneVideo.app/BaiduIphoneVideo", arch = "armv7"

CompileUnit: id = {0x00000000}, file =
"/Users/wingle/Workspace/qqlive4iphone/iphone_4.0_fabu_20150601/Common_Proj/mobileTAD/V
IDEO/Library/Third Party/XMLDictionary/XMLDictionary.m", language = "Objective-C"

Function: id = {0x79900000b02}, name = "+[NSDictionary(XMLDictionary)
dictionaryWithXMLString:]", range = [0x01361160-0x0136119a)

FuncType: id = {0x79900000b02}, decl = XMLDictionary.m:325, clang_type = "NSDictionary *
(NSString *)"

Blocks: id = {0x79900000b02}, range = [0x01361160-0x0136119a)

LineEntry: [0x01361160-0x01361164):
/Users/wingle/Workspace/qqlive4iphone/iphone_4.0_fabu_20150601/Common_Proj/mobileTAD/VI
DEO/Library/Third Party/XMLDictionary/XMLDictionary.m

Symbol: id = {0x0003a1e9}, range = [0x01361160-0x0136119c), name="+
[NSDictionary(XMLDictionary) dictionaryWithXMLString:]"

Variable: id = {0x79900000b1e}, name = "self", type = "Class", location = r0, decl =

Variable: id = {0x79900000b2c}, name = "_cmd", type = "SEL", location = r1, decl =

Variable: id = {0x79900000b3a}, name = "string", type = "NSString *", location = r2, decl =
XMLDictionary.m:325

Variable: id = {0x79900000b4a}, name = "data", type = "NSData *", location = r2, decl =
XMLDictionary.m:327

东西有点多，我们只需关注里面的file这一行：

CompileUnit: id = {0x00000000}, file =

"/Users/jianglianchengDevelopment/Work/iOS_ShareLib/SharedLib/Srvcs/BDPassport4iOS/BDPa

ssport4iOS/SAPI/Extensive/ThirdParty/XMLDictionary/XmlDictionary.m", language = "Objective-C"

CompileUnit: id = {0x00000000}, file =

"/Users/wingle/Workspace/qqlive4iphone/iphone_4.0_fabu_20150601/Common_Proj/mobileTAD/V

IDEO/Library/Third Party/XMLDictionary/XMLDictionary.m", language = "Objective-C"

可以清晰的看到，LLDB给我们找出来了这个方法的位置。 当然这个命令也可以找到方法的其他相关信息，比

如参数等.

image lookup –type

当我们想查看一个类型的时候，可以使用image lookup --type，简写为image lookup -t:

e.g: 我们来看看Model的类型：

(lldb) image lookup -t Model

Best match found in /Users/jianglianchengLibrary/Developer/Xcode/DerivedData/TLLDB-

beqoowskwzbttrejseahdoaivpgq/Build/Products/Debug-iphonesimulator/TLLDB.app/TLLDB:

id = {0x30000002f}, name = "Model", byte-size = 32, decl = Modek.h:11, clang_type = "@interface

Model : NSObject{

NSString * _bb;

NSString * _cc;

NSString * _name;

}

@property (getter = name,setter = setName:,readwrite,nonatomic) NSString * name;

@end

"

可以看到，LLDB把Model这个class的所有属性和成员变量都打印了出来，当我们想了解某个类的时候，直接使

用image lookup -t即可

target stop-hook

我们知道，用LLDB debug，大多数时候需要让程序stop，不管用breakpoint还是用watchpoint。

target stop-hook命令就是让你可以在每次stop的时候去执行一些命令

target stop-hook只对breakpoint和watchpoint的程序stop生效，直接点击Xcode上的pause或者debug view hierarchy不会生效

target stop-hook add & display

假如我们想在每次程序stop的时候，都用命令打印当前frame的所有变量。我们可以添加一个stop-hook：

(lldb) target stop-hook add -o "frame variable"

Stop hook #4 added.

target stop-hook add表示添加stop-hook，-o的全称是--one-liner，表示添加一条命令。

我们看一下，当执行到一个断点的时候会发生什么?

- Hook 1 (frame variable)

(ViewController *) self = 0x00007fd55b12e380

(SEL) _cmd = "viewDidLoad"

(NSMutableURLRequest *) request = 0x00007fd55b1010c0

在程序stop的时候，他会自动执行frame variable，打印出了所有的变量。

大多情况下，我们在stop的时候可能想要做的是打印一个东西。正常情况我们需要用target stop-hook add -o "p xxx"，LLDB提供了一个更简便的命令display。

e.g: 下面2行代码效果相同

(lldb) target stop-hook add -o "p self.view"

(lldb) display self.view

也可以用display来执行某一个命令。p,e,expression是等效的。

target stop-hook list

当添加完stop-hook之后，我们想看当前所有的stop-hook怎么办呢？使用stop-hook list

(lldb) target stop-hook list

Hook: 4

State: enabled

Commands:

frame variable

Hook: 5

State: enabled

Commands:

expression self.view

Hook: 6

State: enabled

Commands:

expr -- self.view

我们可以看到，我们添加了4个stop-hook，每个stop-hook都有一个id，他们分别是4，5，6

target stop-hook delete & undisplay

有添加的命令，当然也就有删除的命令。使用target stop-hook delete可以删除stop-hook，如果你觉得这个命令有点长，懒得敲。你也可以用undisplay

(lldb) target stop-hook delete 4

(lldb) undisplay 5

我们用target stop-hook delete和undisplay分别删除了id为4和5的stop-hook

target stop-hook disable/enable

当我们暂时想让某个stop-hook失效的时候，可以使用target stop-hook disable

```
(lldb) target stop-hook disable 8
```

如果我们想让所有的stop-hook失效，只需不传入stop-hookid即可：

```
(lldb) target stop-hook disable
```

有disable就有enable，我们又想让stop-hook生效了。可以使用target stop-hook enable

```
(lldb) target stop-hook enable 8
```

同理，不传入参数表示让所有stop-hook生效

```
(lldb) target stop-hook enable
```

Extension

前几天@兔be南玻1在微博上给出一个小技巧。LLDB中@import UIKit即可打印frame等变量（默认情况下打不出来）微博链接。

```
(lldb) p self.view.frame
error: property ‘frame‘ not found on object of type ‘UIView *‘
error: 1 errors parsing expression
(lldb) e @import UIKit
(lldb) p self.view.frame
(CGRect) $0 = (origin = (x = 0, y = 0), size = (width = 375, height = 667))
```

由于每次run Xcode，LLDB的东西都会被清空。所以每次run你都需要在LLDB中输入e @import UIKit才能使用这个方便的功能，有点麻烦呀！

之后有人提出了比较方便的一个办法。给UIApplicationMain设置一个断点，在断点中添加执行e @import

UIKit。

这种方法非常方便，不用自己输入了，但是断点我们可能会误删，而且断点是对应工程的。换一个工程又得重新打一个这样的断点。还是有点麻烦。有没有更简便的方法呢?

我们首先想到的是LLDB在每次启动的时候都会load '~/.lldbinit'文件。在这里面执行e @import UIKit不就行了么？不会被误删，对每个工程都有效!

然而想法是美好的，现实却是残酷的！因为UIKit这个库是在target中。而load '~/.lldbinit'的时候target还没创建。所以无法import UIKit。stackoverflow详细解释

这时候我们又想到，可不可以在'~/.lldbinit'中给UIApplicationMain设置一个断点，在断点中添加执行e @import UIKit呢?

答案是不行。原因跟前面一样，load '~/.lldbinit'执行时间太早。断点是依赖target的，target还未创建，断点加不上去。好事多磨，道路坎坷呀~~~

后来我们又想到用stop-hook行不行呢？stop-hook不依赖target。一般我们p frame的时候，都需要先stop，理论上是可行的

事实证明stop-hook的方法完全ok。只需要在'~/.lldbinit'中添加这2条命令即可：

```
display @import UIKit

target stop-hook add -o "target stop-hook disable"
```

命令1：使用display表示在stop的时候执行@import UIKit

命令2：由于我们只需要执行一次@import UIKit，所以执行完成之后，执行target stop-hook disable，使原有的所有stop-hook失效

这个命令有个缺陷，直接点击Xcode上的pause和debug view hierarchy，stop-hook不会生效。正在探索有没有更好的办法完成@import UIKit，如果你想到了，可以联系我~

target symbols add(add-dsym)

说这个命令之前，先简单解释一下dSYM文件。程序运行的时候，都会编译成二进制文件。因为计算机只识别二进制文件，那为什么我们还能在代码上打断点?

这主要是因为在编译的时候Xcode会生成dSYM文件，dSYM文件记录了哪行代码对应着哪些二进制，这样我们对代码打断点就会对应到二进制上。dSYM详细资料

当Xcode找不着dSYM文件的时候，我们就无法对代码打断点，进行调试。target symbols add命令的作用就是让我们可以手动的将dSYM文件添加上去。LLBD对这个命令起了一个别名: add-dsym

e.g: 当我们对接framework的时候，如果只有framework代码，没有工程代码，能不能debug呢？其实我们只需要拿到工程的ipa和dSYM文件，就可以debug了，通过Attach to Process，使用命令add-dsym将dSYM文件加入target，即可只debug framework，不需要工程的代码

```
add-dsym ~/.../XXX.dSYM
```

详细细节可以查看iOS中framework的联调

help & apropos

LLDB的命令其实还有很多，很多命令我也没玩过。就算玩过的命令，我们也非常容易忘记，下次可能就不记得是怎么用的了。还好LLDB给我们提供了2个查找命令的命令:help & apropos

help

直接在LLDB中输入help。可以查看所有的LLDB命令

```
(lldb) help

Debugger commands:


  apropos           -- Find a list of debugger commands related to a particular
                       word/subject.
  breakpoint        -- A set of commands for operating on breakpoints. Also see
                       _regexp-break.
```

```
help            -- Show a list of all debugger commands, or give details
                   about specific commands.
script          -- Pass an expression to the script interpreter for
                   evaluation and return the results. Drop into the
                   interactive interpreter if no expression is given.
settings         -- A set of commands for manipulating internal settable
                   debugger variables.
source           -- A set of commands for accessing source file information
target          -- A set of commands for operating on debugger targets.
thread           -- A set of commands for operating on one or more threads
                   within a running process.
type            -- A set of commands for operating on the type system
version          -- Show version of LLDB debugger.
watchpoint       -- A set of commands for operating on watchpoints.
....(东西太多，只截取了一部分)
```

如果我们想看具体某一个命令的详细用法，可以使用help <command-name> e.g: 我们查看watchpoint命令

```
(lldb) help watchpoint
The following subcommands are supported:

    command -- A set of commands for adding, removing and examining bits of
               code to be executed when the watchpoint is hit (watchpoint
               'commmands').
    delete  -- Delete the specified watchpoint(s).  If no watchpoints are
               specified, delete them all.
    disable -- Disable the specified watchpoint(s) without removing it/them.
               If no watchpoints are specified, disable them all.
    enable  -- Enable the specified disabled watchpoint(s). If no watchpoints
               are specified, enable all of them.
    ignore  -- Set ignore count on the specified watchpoint(s).  If no
               watchpoints are specified, set them all.
```

```
list     -- List all watchpoints at configurable levels of detail.

modify  -- Modify the options on a watchpoint or set of watchpoints in
           the executable.  If no watchpoint is specified, act on the
           last created watchpoint.  Passing an empty argument clears the
           modification.

set      -- A set of commands for setting a watchpoint.
```

有的时候，我们可能并不能完全记得某个命令，如果只记得命令中的某个关键字。这时候我们可以使用apropos搜索相关命令信息。

e.g: 我们想使用stop-hook的命令，但是已经不记得stop-hook命令是啥样了

```
(lldb) apropos stop-hook
The following built-in commands may relate to 'stop-hook':
  _regexp-display           -- Add an expression evaluation stop-hook.
  _regexp-undisplay         -- Remove an expression evaluation stop-hook.
  target stop-hook          -- A set of commands for operating on debugger
                               target stop-hooks.
target stop-hook add      -- Add a hook to be executed when the target stops.
target stop-hook delete   -- Delete a stop-hook.
target stop-hook disable  -- Disable a stop-hook.
target stop-hook enable   -- Enable a stop-hook.
target stop-hook list     -- List all stop-hooks.
```

可以看到使用apropos stop-hook搜索一下，即可将所有stop-hook相关命令搜索出来

常用的Debug快捷键

debug的时候，使用快捷键是一个很好的习惯，我简单列举了几个debug的快捷键

End

东西有点多，感谢大家耐心看完这篇文章。LLDB命令非常多，有很多LLDB命令我也没玩过。这些命令我们不一定要完全记住，只要有个印象LLDB可以实现哪些功能就可以了。具体用的时候再用help或者apropos查找。

标签：

原文地址：http://www.cnblogs.com/CoderAlex/p/5295736.html