

CSDN

博客 (http://blog.csdn.net?ref=toolbar) 学院 (http://edu.csdn.net?ref=toolbar)

下载 (http://download.csdn.net?ref=toolbar) 更多 ▾

🔍

📝 写博客

🗨️ 发布Chat (http://gitbook.cn/new/gitchat/activity?utm_source=csdnblog1)

登录 (https://passport.csdn.net/account/login?ref=toolbar) 注册 (http://passport.csdn.net/account/mobileregister?ref=toolbar&action=mobileRegister)

iOS crash log 解析 symbol address = stack address – slide 运行时获取slide的api 利用dwarfdump从dsym文件中得到symbol

原创

2016年01月04日 16:05:14

标签: [slide \(http://so.csdn.net/so/search/s.do?q=slide&t=blog\)](http://so.csdn.net/so/search/s.do?q=slide&t=blog) / [ios \(http://so.csdn.net/so/search/s.do?q=ios&t=blog\)](http://so.csdn.net/so/search/s.do?q=ios&t=blog) / [exception \(http://so.csdn.net/so/search/s.do?q=exception&t=blog\)](http://so.csdn.net/so/search/s.do?q=exception&t=blog) / [crash-log \(http://so.csdn.net/so/search/s.do?q=crash-log&t=blog\)](http://so.csdn.net/so/search/s.do?q=crash-log&t=blog)

📖 996

概述:

为什么 crash log 内 Exception Backtrace 部分的地址（stack address）不能从 dsym 文件中查出对应的代码？

因为 ASLR（Address space layout randomization），因为 ASLR 引入了一个 slide（偏移）。

symbol address = stack address – slide;
slide 可以在运行时 由 API 获取到

```
1 dyld_get_image_vmaddr_slide()
```

也可以根据运行时的 binary image 和 ELF 文件的 load command 计算的到。

slide = (运行时)load address – (链接时)load address;

注意，如果你没有在运行时用 api 获取slide，那么 binary image 就必须要收集，否则你无法从dsym 文件中解析出符号。

crash log 文件

这是一个 iOS crash log 文件，为了简洁删除了部分不需要的内容

乔志广 (http://blog.csdn....)

+ 关注

(http://blog.csdn.net/xiaofei125145)

码云

原创

粉丝

喜欢

未开通 (https://github.com/xiaofei125145)

97

17

0

utm_source=github

- 他的最新文章
- 更多文章 (http://blog.csdn.net/xiaofei125145)
- ruby to_json "\xE6" from ASCII-8BIT to UTF-8 (Encoding::UndefinedConversionError) (/xiaofei125145/article/details/77540395)

Ruby学习笔记 irb Tab 代码补全 Mac OS (/xiaofei125145/article/details/66472306)

iOS开发 jenkins (DRYPPlugin) + PMD/CPD 检测重复代码 (/xiaofei125145/article/details/52789736)

C 语言中的constructor与destructor (/xiaofei125145/article/details/52597256)

望京二手房出售

在线课程

深度学习部署系统构建 (http://edu.csdn.net/course/detail/578?utm_source=blog9)

神经网络技术分享 (http://edu.csdn.net/course/detail/590?utm_source=blog9)

热门文章

```

1 Incident Identifier: 975CF16A-5259-4DD1-BFDA-D1B155EF5BF0
2 CrashReporter Key: 562a7cefe034ac086cae453c61278cdd9a4b3288
3 Hardware Model: iPad4,1
4 Process: MedicalRecordsFolder [382]
5 Path: /var/mobile/Applications/05C398CE-21E9-43C2-967F-26DD0A32793
6 2/MedicalRecordsFolder.app/MedicalRecordsFolder
7 Identifier: com.xingshulin.MedicalRecordIOS
8 Version: 1 (4.14.0)
9 Code Type: ARM-64 (Native)
10 Parent Process: launchd [1]
11
12
13 Date/Time: 2015-12-03 19:14:59.921 +0800
14 OS Version: iOS 7.1.2 (11D257)
15 Report Version: 104
16
17
18
19
20 Exception Type: EXC_CRASH (SIGABRT)
21 Exception Codes: 0x0000000000000000, 0x0000000000000000
22 Triggered by Thread: 0
23
24
25 Last Exception Backtrace:
26 0 CoreFoundation 0x189127100 __exceptionPreprocess + 132
27 1 libobjc.A.dylib 0x1959e01fc objc_exception_throw + 60
28 2 CoreFoundation 0x189127040 +[NSException raise:format:] + 128
29 3 MedicalRecordsFolder 0x100a8666c 0x10003c000 + 10790508
30 4 libsystem_platform.dylib 0x19614bb0c _sigtramp + 56
31 5 MedicalRecordsFolder 0x1006ef164 0x10003c000 + 7024996
32 6 MedicalRecordsFolder 0x1006e8580 0x10003c000 + 6997376
33 7 MedicalRecordsFolder 0x1006e8014 0x10003c000 + 6995988
34 8 MedicalRecordsFolder 0x1006e7c94 0x10003c000 + 6995092
35 9 MedicalRecordsFolder 0x1006f2460 0x10003c000 + 7038048
36 10 libdispatch.dylib 0x195fb8014 _dispatch_call_block_and_release + 2
37 4
38 11 libdispatch.dylib 0x195fb7fd4 _dispatch_client_callout + 16
39 12 libdispatch.dylib 0x195fba4a8 _dispatch_queue_drain + 640
40 13 libdispatch.dylib 0x195fba4c0 _dispatch_queue_invoke + 68
41 14 libdispatch.dylib 0x195fbf0f4 _dispatch_root_queue_drain + 104
42 15 libdispatch.dylib 0x195fbf4fc _dispatch_worker_thread2 + 76
43 16 libsystem_pthread.dylib 0x19614d6bc _pthread_wqthread + 356
44 17 libsystem_pthread.dylib 0x19614d54c start_wqthread + 4
45
46
47 Thread 0 Crashed:
48 0 libsystem_kernel.dylib 0x00000001960ce58c __pthread_kill + 8
49 1 libsystem_c.dylib 0x0000000196062804 abort + 108
50 2 libc++abi.dylib 0x0000000195288990 abort_message + 84
51 3 libc++abi.dylib 0x00000001952a5c28 default_terminate_handler()
52 + 296
53 4 libobjc.A.dylib 0x00000001959e04d0 _objc_terminate() + 124
54 5 libc++abi.dylib 0x00000001952a3164 std::__terminate(void (*)())
55 + 12
56 6 libc++abi.dylib 0x00000001952a2d38 __cxa_rethrow + 140
57 7 libobjc.A.dylib 0x00000001959e03a4 objc_exception_rethrow + 40
58
59 8 CoreFoundation 0x0000000189025e48 CFRunLoopRunSpecific + 572
60 9 GraphicsServices 0x000000018ecb5c08 GSEventRunModal + 164
61 10 UIKit 0x000000018c156fc0 UIApplicationMain + 1152
62 11 MedicalRecordsFolder 0x000000010018fc70 0x10003c000 + 1391728
63 12 libdyld.dylib 0x0000000195fd3a9c start + 0
64
65
66 Thread 1:
67 0 libsystem_kernel.dylib 0x00000001960b5aa8 kevent64 + 8
68 1 libdispatch.dylib 0x0000000195fb9998 _dispatch_mgr_thread + 48
69 ....
70
71
72 Thread 0 crashed with ARM Thread State (64-bit):

```

学习笔记: shell 中 [-eq] [-ne] [-gt] [-lt] [ge] [le] (/xiaofei125145/article/details/40187031)

24289

学习笔记: shell 中的 set -e, set +e 用法 (/xiaofei125145/article/details/39345331)

9221

mac os x 系统安装 genymotion android 模拟器 (/xiaofei125145/article/details/39005047)

8917

Mac ssh 远程登录 无需密码验证 设置 (/xiaofei125145/article/details/30243535)

7194

ios开发使用red125145article/静态库解决了9691111 protobuf 导致的冲突问题

6137

73

x0: 0x0000000000000000

x1: 0x0000000000000000

x2: 0x0000000000000000

x

74

3: 0xffffffffffffffff

75

x4: 0x0000000000000306

x5: 0x000000016fdc3530

x6: 0x000000000000006e

x

76

7: 0x0000000000000640

77

x8: 0x0000000080000000

x9: 0x0000000040000000

x10: 0x0000000098efe6f7

x1

78

1: 0x0000000198efde94

79

x12: 0x000000000000006f

x13: 0x0000000000000000

x14: 0x0000000000000000

x1

80

5: 0x000000019607bdc

81

x16: 0x0000000000000148

x17: 0x004b96d3524ed02c

x18: 0x0000000000000000

x1

82

9: 0x0000000000000006

83

x20: 0x0000000198f112a0

x21: 0x434c4e47432b2b00

x22: 0x434c4e47432b2b00

x2

84

3: 0x0000000000000001

85

x24: 0x00000001701578c0

x25: 0x0000000000000001

x26: 0x0000000170002ea0

x2

7: 0x00000001963e1410

x28: 0x0000000000000000

fp: 0x000000016fdc34b0

lr: 0x000000019615116c

sp: 0x000000016fdc3490

pc: 0x00000001960ce58c

cpsr: 0x00000000

Binary Images:

0x10003c000 - 0x100f7bfff MedicalRecordsFolder arm64 <b5ae3570a013386688c7007ee2e73978> /var/mobile/Applications/05C398CE-21E9-43C2-967F-26DD0A327932/MedicalRecordsFolder.app/MedicalRecordsFolder

0x12007c000 - 0x1200a3fff dyld arm64 <628da833271c3f9bb8d44c34060f55e0> /usr/lib/dyld

.....

现在来指出其中比较重要的部分

uuid信息

1	Incident Identifier: 975CF16A-5259-4DD1-BFDA-D1B155EF5BF0
---	---

这行指出文件的 uuid ，根据次 uuid 可确定 dsym 文件是否匹配
确定方法如下,后面会打印出该 dsym 文件内所有 架构的 uuid ，看一下 是否包含 就知道了

1	dwarfdump --uuid MedicalRecordsFolder.app.dSYM/
---	---

arch

arch信息不解释

1	Code Type: ARM-64 (Native)
---	----------------------------

异常信息

下面是 异常信息，异常线程 为 thread 0

1	Exception Type: EXC_CRASH (SIGABRT)
2	Exception Codes: 0x0000000000000000, 0x0000000000000000
3	Triggered by Thread: 0

异常线程函数调用栈

- 接下来看 抛出异常的线程的函数调用栈信息
- 左侧
- 第一列，调用顺序
- 第二列，对应函数所属的 binary image
- 第三列，stack address
- 第四列，地址的符号 + 偏移的表示法，实质内容跟第三列一样（此列不理解也无影响）

1	Last Exception Backtrace:		
2	0	CoreFoundation	0x189127100 __exceptionPreprocess + 132
3	1	libobjc.A.dylib	0x1959e01fc objc_exception_throw + 60
4	2	CoreFoundation	0x189127040 +[NSException raise:format:] + 128
5			
6	3	MedicalRecordsFolder	0x100a8666c 0x10003c000 + 10790508
7	4	libsystem_platform.dylib	0x19614bb0c _sigtramp + 56
8	5	MedicalRecordsFolder	0x1006ef164 0x10003c000 + 7024996
9	6	MedicalRecordsFolder	0x1006e8580 0x10003c000 + 6997376
10	7	MedicalRecordsFolder	0x1006e8014 0x10003c000 + 6995988
11	8	MedicalRecordsFolder	0x1006e7c94 0x10003c000 + 6995092
12	9	MedicalRecordsFolder	0x1006f2460 0x10003c000 + 7038048
13	10	libdispatch.dylib	0x195fb8014 _dispatch_call_block_and_release + 24
14	11	libdispatch.dylib	0x195fb7fd4 _dispatch_client_callout + 16
15	12	libdispatch.dylib	0x195fbe4a8 _dispatch_queue_drain + 640
16	13	libdispatch.dylib	0x195fba4c0 _dispatch_queue_invoke + 68
17	14	libdispatch.dylib	0x195fbf0f4 _dispatch_root_queue_drain + 104
18	15	libdispatch.dylib	0x195fbf4fc _dispatch_worker_thread2 + 76
19	16	libsystem_pthread.dylib	0x19614d6bc _pthread_wqthread + 356
20	17	libsystem_pthread.dylib	0x19614d54c start_wqthread + 4

我们从 binary image 这列里面可以看出 好多都是 动态库调用，动态库也就是说 这是 sdk 里面的东西，即使出了 bug 你也修复不了，所以我们需要关心的就只有

第 3、5、6、7、8、9、这些行，只有这行行对应的代码 才是你自己的 创作（我工程名就是 MedicalRecordsFolder）

是 函数调用顺序是 从下往上，也就是说 第 3 行对应的函数才是出问题 正在执行的代码片段。

所以 从 dsym 中找到 第三行对应的 符号信息 才可能定位到 问题代码。

第三行第三列

0x100a8666c，这是 stack address，注意是 stack address，如果系统没有 ASLR 的话，用这个 stack address 就能在 dsym 中找到对应符号信息，但是事实 iOS 是有 ASLR 的。

ASLR

ASLR 技术 Address space layout randomization, ASLR 通过将系统可执行程序随机装载到内存里，从而防止缓冲区溢出攻击

由于 ASLR 的缘故，导致 程序 crash 后生成的 crash log 中的 stack address 与 对应的 symbol address 不一致，有一个偏移量 slide，slide 是程序装定时随机生成的随机数。

stack address

：程序运行时线程栈中 所有 函数调用的地址

symble address

：dsym 文件中函数符号对应的地址，用此地址 在 dsym 文件中可以 查出对应的 符号信息。

无 ASLR 机制时 stack address 等于 symble address。

slide

在 ASLR 机制下每次启动 APP 装之前，会在连接时指定的 进程空间上 加上一个随意的 偏移量，这个偏移量就是 slide。

很简单 $\text{symbol address} = \text{stack address} - \text{slide}$;

但是这个 slide 每次 启动 程序都不同，如何 知道 当时启动时 slide 的值呢？带着疑问继续吧

Load Command

一个 iOS 程序编译链接完之后，生成一个 ELF 二进制文件（也就是程序运行时的映射文件），该文件的详细格式不再赘述，这里只强调一个 segment _TEXT

下面是 使用 otool 工具查看到的 MedicalRecordsFolder（我的程序）的 加载命令。



```
1  $otool -l MedicalRecordsFolder.app/MedicalRecordsFolder
2  MedicalRecordsFolder.app/MedicalRecordsFolder:
3  Load command 0
4      cmd LC_SEGMENT_64
5      cmdsize 72
6      segname __PAGEZERO
7      vmaddr 0x0000000000000000
8      vmsize 0x0000000010000000
9      fileoff 0
10     filesize 0
11     maxprot 0x00000000
12     initprot 0x00000000
13     nsects 0
14     flags 0x0
15 Load command 1
16     cmd LC_SEGMENT_64
17     cmdsize 792
18     segname __TEXT
19     vmaddr 0x0000000010000000
20     vmsize 0x000000000000c000
21     fileoff 0
22     filesize 49152
23     maxprot 0x00000005
24     initprot 0x00000005
25     nsects 9
26     flags 0x0
27
28
29 .....
30
31
32 Load command 2
33     cmd LC_SEGMENT_64
34     cmdsize 1352
35     segname __DATA
36     vmaddr 0x0000000010000c00
37     vmsize 0x0000000000004000
38     fileoff 49152
39     filesize 16384
40     maxprot 0x00000003
41     initprot 0x00000003
42     nsects 16
43     flags 0x0
44
45
46 .....
47
48
49 Load command 3
50     cmd LC_SEGMENT_64
51     cmdsize 72
52     segname __LINKEDIT
53     vmaddr 0x0000000010001000
54     vmsize 0x000000000000c000
55     fileoff 65536
56     filesize 35056
57     maxprot 0x00000001
58     initprot 0x00000001
59     nsects 0
60     flags 0x0
```

_TEXT 段的加载命令如下，可知到映射文件中segment _TEXT 对应的 虚拟地址空间从 0x0000000010000000 开始。

1	segname __TEXT
2	vmaddr 0x0000000100000000
3	vmsize 0x000000000000c000
4	fileoff 0
5	filesize 49152

segname __TEXT 就是代码段，也就是说所有的二进制指令

没有 ASLR 机制时：

加载时 装载器会将此 ELF 文件的前 49152（offset 0，filesize 49152）个字节（因为 offset 0，filesize 49152）映射到 进程空间以 0x0000000100000000 开始的一块虚拟内存空间里。

有 ASLR 机制时：

加载时 装载器会将此 ELF 文件的前 49152（offset 0，filesize 49152）个字节（因为 offset 0，filesize 49152）映射到 进程空间以 0x0000000100000000（+slide）开始的一块虚拟内存空间里。

所以： 如果没有 ASLR 机制，那么运行时的内存布局 就和 Load command 中指定的布局一致，也就意味着 stack address和 symbol address 一致

有 ASLR 的情况也不复杂，只是 加了一个 随意的偏移量 slide

binary image

程序运行时 的 映射 信息，

1	Binary Images:
2	0x10003c000 - 0x100f7bfff MedicalRecordsFolder arm64 <b5ae3570a013386688c7007ee2e73978> /var/.../MedicalRecordsFolder
3	0x12007c000 - 0x1200a3fff dyld arm64 <628da833271c3f9bb8d44c34060f55e0> /usr/lib/dyld

左侧

第一列，虚拟地址空间区块

第二列，映射文件 名

第三列，uuid吧，还不知道,以后再补上

第四列，映射文件路径

计算 slide 和 symbol address

在 binary image 第一行可以看出 进程空间的 0x10003c000 - 0x100f7bfff 这个区域 在运行时被映射为 MedicalRecordsFolder 内的内容，也就是我们的 ELF 文件。

注意这个 区域起始地址 为 0x10003c000

而我们在 Load Command 中看到的却是 0x0000000100000000

1	segname __TEXT
2	vmaddr 0x0000000100000000
3	vmsize 0x000000000000c000

显而易见：

slide = 0x10003c000 - 0x100000000
= 0x3c000;

symbol address = stack address - slide;

stack address 在crash log 中已经找到了。

用的到的symbol地址去 dsym 文件中 查询，命令如下

```
1 $dwarfdump --arch arm64 --lookup 0x00123 MedicalRecordsFolder.app.dSYM/
```

就可以定位下来具体的 代码 函数名，所处的文件，行 等信息了

读取 slide 的 API

这个 slide 的计算还是挺 恶心的，要 查看 binary image 的到 load address，还要查看 对用 ELF 中 _TEXT 的 Load Command 虚拟空间范围。

如果 自己写一个 模块 来 收集 NSException 的话，大可不必这么繁琐，因为 程序 运行时 有 api 是可以 直接获取这个 binary image 对应的 slide 值的。

如下:

```
1 #import <mach-o/dyld.h>
2 void calculate(void) {
3     for (uint32_t i = 0; i < _dyld_image_count(); i++) {
4         if (_dyld_get_image_header(i)->filetype == MH_EXECUTE) {
5             long slide = _dyld_get_image_vmaddr_slide(i);
6             break;
7         }
8     }
9 }
```

这样 就可以 将 stack address 直接 减去 slide 之后 再 上传到自己的 服务端，岂不是 很完美。

版权声明：本文为博主原创文章，未经博主允许不得转载。



相关文章推荐

iOS中线程Call Stack的捕获和解析（二） (/jasonblog/article/details/49909209)

上接iOS中线程Call Stack的捕获和解析（一）。1. 部分参考资料做这一块时也是查阅了很多链接和书籍，包括但不限于：《OS X ABI Mach-O File Format Referenc...

 jasonblog (<http://blog.csdn.net/jasonblog>) 2015-11-18 16:36  5363

iOS crash log 解析 symbol address = stack address – slide 运行时获取slide的api 利用dwarfdump从dsym文件中得到symbol (/xiaofei125145/article/details/50408051)

概述：为什么 crash log 内 Exception Backtrace 部分的地址（stack address）不能从 dsym 文件中查出对应的代码？因为 ASLR（Address spa...

 xiaofei125145 (<http://blog.csdn.net/xiaofei125145>) 2015-12-26 12:38  1676



月薪3万的前端程序员都避开了哪些坑？

程序员薪水有高有低，同样工作5年的程序员，有的人月薪30K、50K，有的人可能只有5K、8K。是什么因素导致了这种差异？

(http://edu.csdn.net/topic/web1?utm_source=blog10)

SSISThe Address in the "To" line is malformed. It is either missing the "@" symbol or is not valid. (/kevinsqlserver/article/details/8060172)

今天做了一个SSIS的小程序，其中有一个功能是发邮件，但是发现发MAIL的组件提示警告，信息如下： The Address in the "To" line is malformed. It...

 SmithLiu328 (<http://blog.csdn.net/SmithLiu328>) 2012-10-11 14:01  1413





iOS友盟崩溃地址解析 通过dSYM文件分析定位线上 APP crash问题 (/csdn100861/article/details/51518526)

有很多问题是在开发测试过程中无法遇到和重现的，这就需要统计线上的崩溃信息进行定位。什么是 dSYMXcode编译项目后，我们会看到一个同名的 dSYM 文件，dSYM 是保存 16 进制函数地址映射信...

 csdn100861 (<http://blog.csdn.net/csdn100861>) 2016-05-27 17:35  553



UI Automation for IOS及Crash .dSYM 文件的解析。 (/xiaobai20131118/article/details/45821157)

最近用过 Xcode 中的 Instruments的Automation工具做了一些 IOS 的端的测试，主要用到开源的github 中开源的monkey脚本。关于IOS for Automatio...

 u012874998 (<http://blog.csdn.net/u012874998>) 2015-05-18 14:39  547



iOS – 命令行工具解析Crash文件,dSYM文件进行符号化 (/yuqingzhude/article/details/54314983)

序 在日常开发中，app难免会发生崩溃。简单的崩溃还好说，复杂的崩溃就需要我们通过解析Crash文件来分析了，解析Crash文件在iOS开发中是

 yuqingzhude (<http://blog.csdn.net/yuqingzhude>) 2017-01-10 16:17  629



利用libbfd获取elf可执行文件的section(节)及symbol(符号)信息 (/crazycoder8848/article/details/51456297)

一. 安装bfd库 libbfd(Binary File Descriptor library是binutils中附带的一个C库。从 <http://ftp.gnu.org/gnu/binutil...>

 crazycoder8848 (<http://blog.csdn.net/crazycoder8848>) 2016-05-19 19:46  1653



iOS -- 友盟工具进行Crash分析/dsym文件 (/sir_coding/article/details/53426588)

项目中集成了友盟统计，自然Crash日志已经在友盟的统计之中，点击错误分析可以看到相关的错误列表，以及简单的crash日志。如果想看详细的crash详情则需要使用友盟的错误分析工具：umcrasht...

 Sir_Coding (http://blog.csdn.net/Sir_Coding) 2016-12-01 20:44  2370

iOS通过dSYM文件分析crash (/u011270282/article/details/44805739)

http://blog.csdn.net/hjy_x/article/details/20929095 重点是dwarfdump --uuid命令 我们在ios开发中会...

 u011270282 (<http://blog.csdn.net/u011270282>) 2015-04-01 13:34  418

iOS通过dSYM文件分析crash (/klabcxy36897/article/details/50847569)

http://blog.csdn.net/hjy_x/article/details/20929095 重点是dwarfdump --uuid命令 我们在ios开发中会碰到的很多cra...

 klabcxy36897 (<http://blog.csdn.net/klabcxy36897>) 2016-03-10 16:03  284

iOS通过dSYM文件分析crash (/zrhloveswallow/article/details/45920345)

我们在ios开发中会碰到的很多crash问题，如果Debug调试模式的话，我们可以往往很容易的根据log的输出定位到导致crash的原因，但对于已经上线的应用，或者是release环境包导致的cras...



zrhloveswallow (<http://blog.csdn.net/zrhloveswallow>) 2015-05-22 18:53 360

内存错误: Address is on thread 1's stack 与创建变量时出现段错误 (/linwh8/article/details/51239673)

内存错误: Address is on thread 1's stack 与创建变量时出现段错误标签: 内存错误 段错误by 小威威今天对代码进行内存检测的时候,发现出现了以下问题: Inva...



linwh8 (<http://blog.csdn.net/linwh8>) 2016-04-25 08:26 767



利用.dSYM和.app文件准确定位Crash位置 (/yinxin2745154/article/details/43604929)

原文: <http://blog.csdn.net/jinzhul17/article/details/20615991> 首先,确保在release (Ad Hoc或者App Store) 一个版本时,保...



yinxin2745154 (<http://blog.csdn.net/yinxin2745154>) 2015-02-07 17:34 298

利用第三方工具dSYM快速定位crash文件中崩溃的函数位置 (/hanxiaodongruanjian/article/details/50965457)

原文转自:点击打开链接 app上线之后程序崩溃信息我们只能通过crash文件获取崩溃信息,但是crash文件中的都是一些16进制的数表示函数地址的. 如下图,根本不能直接看到具体程序崩溃在什么...



hanxiaodongruanjian (<http://blog.csdn.net/hanxiaodongruanjian>) 2016-03-23 19:12 260

利用.dSYM和.app文件准确定位Crash位置 (/zrhloveswallow/article/details/45920375)

首先,确保在release (Ad Hoc或者App Store) 一个版本时,保存了对应的xxx.app和xxx.dSYM文件. 其次,验证xxx.crash、xxx.app和xxx.dS...



zrhloveswallow (<http://blog.csdn.net/zrhloveswallow>) 2015-05-22 18:55 295

利用.dSYM和.app文件准确定位Crash位置 (/klabcxy36897/article/details/51612271)

<http://blog.csdn.net/jinzhul17/article/details/20615991> 利用.dSYM和.app文件准确定位Crash位置...



klabcxy36897 (<http://blog.csdn.net/klabcxy36897>) 2016-06-08 12:25 336

利用.dSYM和.app文件准确定位Crash位置 (/chengkaizone/article/details/44957685)

首先,确保在release (Ad Hoc或者App Store) 一个版本时,保存了对应的xxx.app和xxx.dSYM文件.其次,验证xxx.crash、xxx.app和xxx.dSYM三者的uui...



chengkaizone (<http://blog.csdn.net/chengkaizone>) 2015-04-09 11:41 306

工具解析Crash文件&dSYM文件进行符号化 (/koocui/article/details/75062654)

命令行工具解析Crash文件&dSYM文件进行符号化 话说: 在日常开发中, app难免会发生崩溃.简单的崩溃还好说,复杂的崩溃就需要我们通过解析Crash文件来分析了, 解析Crash文件...



koocui (<http://blog.csdn.net/koocui>) 2017-07-13 15:10 120

IOS学习之路十 (仿人人滑动菜单Slide-out Sidebar Menu) (/yaoqiuxiangcom/article/details/24655693)


最近滑动菜单比较流行,像facebook和人人等都在使用滑动菜单,今天做了一个小demo大体效果如下: 这次用了一个开源的项目ECSlidingViewCon...



u011140147 (<http://blog.csdn.net/u011140147>) 2014-04-28 17:09 1169

iOS 9适配系列教程：URL scheme，iPad适配Slide Over 和 Split View (/bruce__liu/article/details/48541181)

URL scheme 在iOS9中，如果使用URL scheme必须在"Info.plist"中将你要在外部调用的URL scheme列为白名单，否则不能使用。key叫做LSApplication...

 Bruce__Liu (http://blog.csdn.net/Bruce__Liu) 2015-09-18 08:28 898



1

