解析线上 app 崩溃报告

1.简述：想来每个 iOS 攻城狮，都免不了要接触.crash 文件，那么什么是.crash 文件?

iOS app 的所有崩溃记录都会记录在设备上，所以对于和我一样没有集成让用户发送崩溃报告功能的 iOS 开发者来说，要获得 crash 文件就必须先连上崩溃过的机器,然后从崩溃过的机器上导出.crash 文件。

2.如何解析.crash 文件

我们先看一眼导出来的.crash 文件，重点看下崩溃部分的记录，如下图：

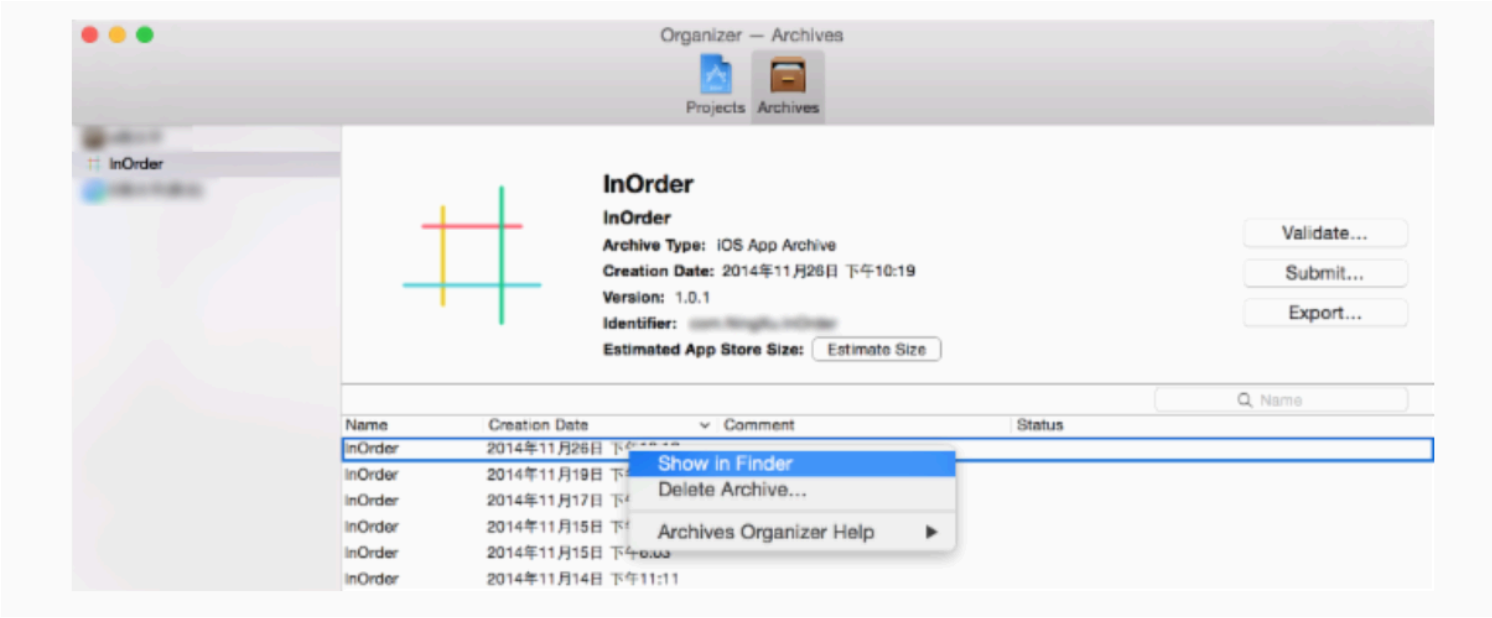
```
Exception Type:  EXC_CRASH (SIGABRT)
Exception Codes: 0x0000000000000000, 0x0000000000000000
Triggered by Thread:  0

Last Exception Backtrace:
0   CoreFoundation                  0x2f452f7e __exceptionPreprocess + 126
1   libobjc.A.dylib                 0x39c03cca objc_exception_throw + 34
2   CoreFoundation                  0x2f3897c6 -[__NSArrayM objectAtIndex:] + 226
3   InOrder                         0x000fe6a2 0xe4000 + 108194
4   InOrder                         0x000fde5e 0xe4000 + 106078
5   InOrder                         0x000fc79c 0xe4000 + 100252
6   InOrder                         0x0011656e 0xe4000 + 206190
7   UIKit                           0x31d6bc30 -[UITextField canBecomeFirstResponder] + 184
8   UIKit                           0x31cee36c -[UIResponder(Internal) _canBecomeFirstResponder] + 16
9   UIKit                           0x31cee088 -[UIResponder becomeFirstResponder] + 204
10  UIKit                           0x31cee3e2 -[UIView(Hierarchy) becomeFirstResponder] + 102
11  UIKit                           0x31d6ac12 -[UITextField becomeFirstResponder] + 42
12  UIKit                           0x31e0f21e -[UITextInteractionAssistant(UITextInteractionAssistant_Internal)
setFirstResponderIfNecessary] + 170
```

显然从这图里，我们没法定位出具体的错误，和崩溃的具体行数。
我们得到是个二进制的报告，这时候我们需要对它进行反编译。
对.crash 文件进行反编译我们需要用到三个文件，缺一不可

2.1 找到发布 app 时的.ipa 文件(就是你打包 app 上传到商店里到那个文件)

2.2 找到 Archive 时生成的.dSYM 文件

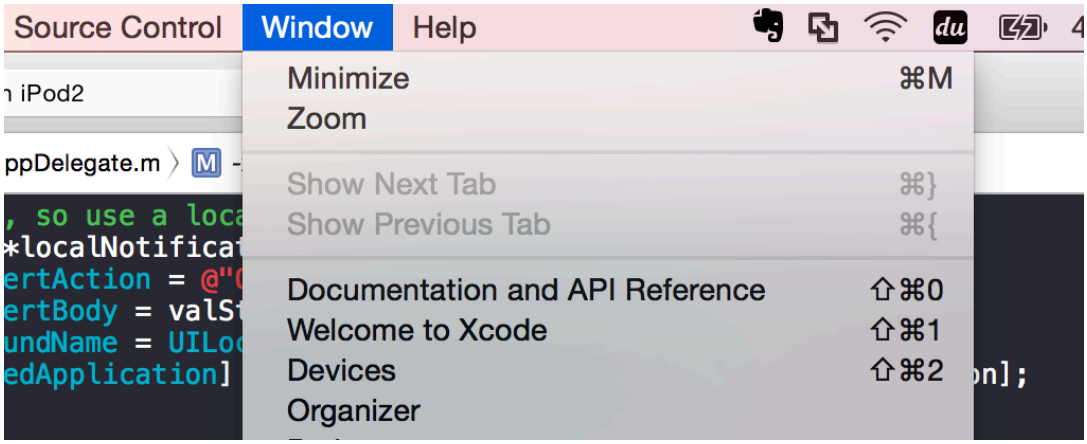


Show in Finder 后对文件夹中的这个.xcarchive 文件右键，显示包内容，就可以看到一个名为 dSYMs 的文件夹，把里面的.dSYM 文件拷出来

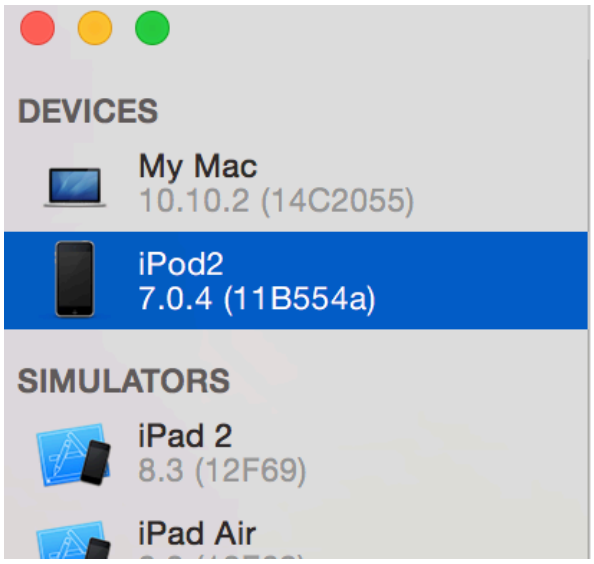
2.3 将.ipa、.crash、.dSYMs 文件放在同一个文件夹(例如：创建一个 bugcrash 文件夹)

2.4 打开你的 xcode，然后打开你发布时的代码(记住必须是发布时的代码，如果不是的话，会出现指向的崩溃行数有误，或者无法解析出来的问题)

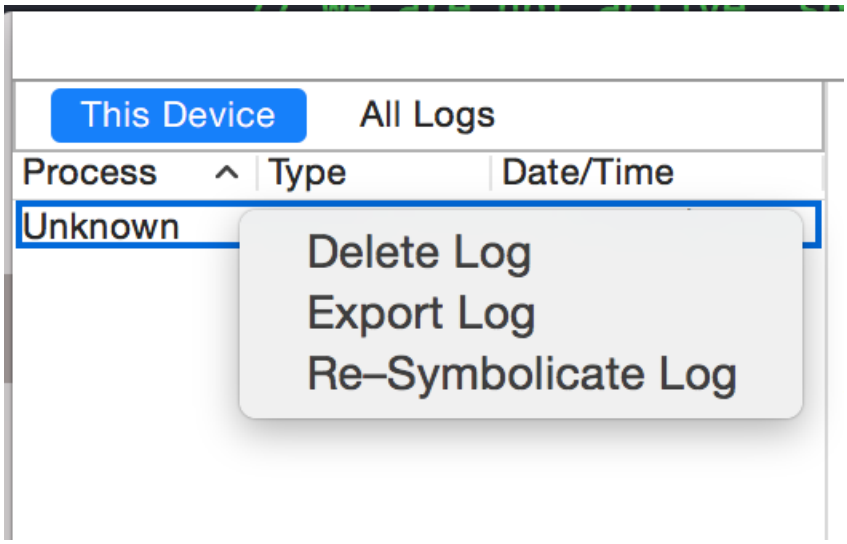
3.利用 xcode 自带 re-symbolicate 解析崩溃报告

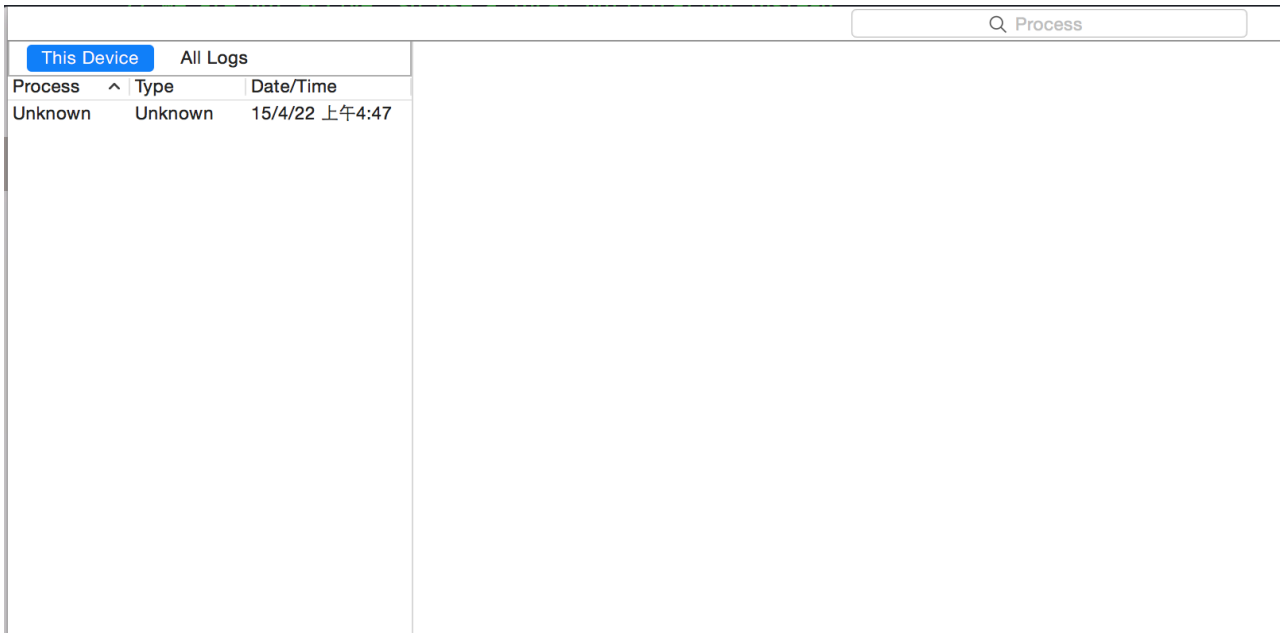


点 Devices 进去



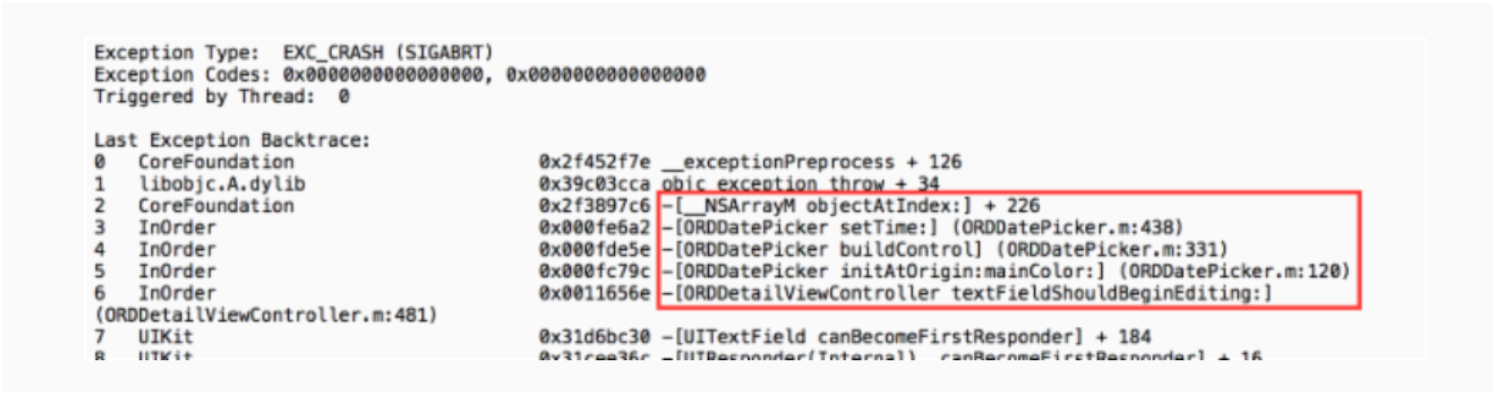
然后点击 View Device Logs 进去





现在打开 bugcrash 文件夹，然后把.crash 文件拖入右框中然后选中右键 re-symbolicate log 进行解析

大约 1-10 秒中之后，我们就能发现之前的二进制数变成了我们想要看到的具体行数和具体的.crash 原因.



```
Exception Type:  EXC_CRASH (SIGABRT)
Exception Codes: 0x0000000000000000, 0x0000000000000000
Triggered by Thread:  0

Last Exception Backtrace:
0   CoreFoundation                  0x2f452f7e __exceptionPreprocess + 126
1   libobjc.A.dylib                 0x39c03cca objc_exception_throw + 34
2   CoreFoundation                  0x2f3897c6 -[__NSArrayM objectAtIndex:] + 226
3   InOrder                         0x000fe6a2 -[ORDDatePicker setTime:] (ORDDatePicker.m:438)
4   InOrder                         0x000fde5e -[ORDDatePicker buildControl] (ORDDatePicker.m:331)
5   InOrder                         0x000fc79c -[ORDDatePicker initAtOrigin:mainColor:] (ORDDatePicker.m:120)
6   InOrder                         0x0011656e -[ORDDetailViewController textFieldShouldBeginEditing:]
(ORDDetailViewController.m:481)
7   UIKit                           0x31d6bc30 -[UITextField canBecomeFirstResponder] + 184
8   UIKit                           0x31cee36c -[UIResponder(Internal) _canBecomeFirstResponder] + 16
```

4.如果 xocde 自带的 Re-symbolicate 仍无法解析，那么我们需要利用 symbolicatecrash 借助命令行手动解析

symbolicatecrash 是一个隐藏工具，它在我的 Mac 中的具体路径如下(Xcode6.1.app 请换成你的 Xcode 名称)

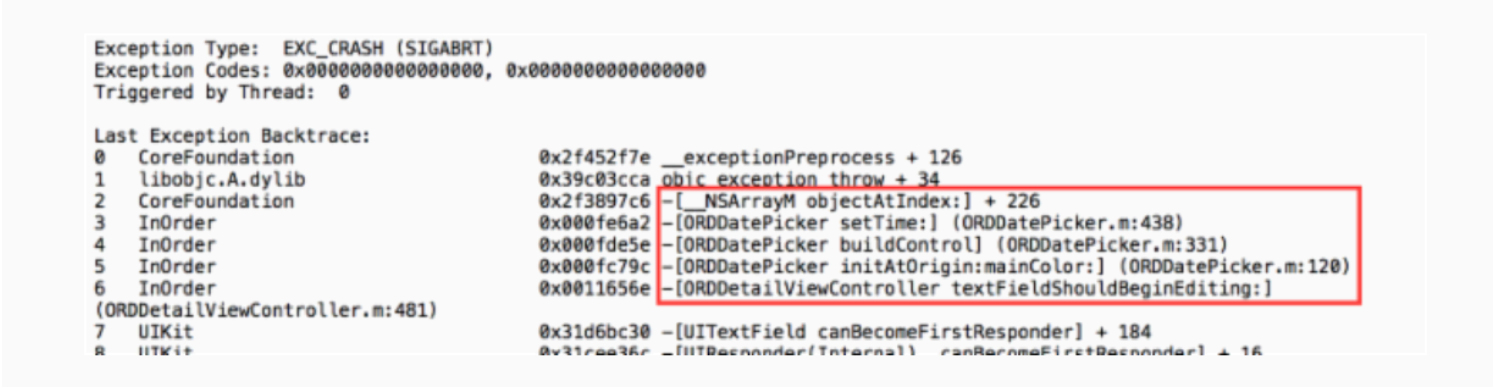
/Applications/Xcode6.1.app/Contents/SharedFrameworks/DTDeviceKitBase.framework/Versions/A/Resources/symbolicatecrash

把这个路径拷贝一下，然后粘到 Finder 的“前往文件夹”下，前往，就可以看到 symbolicatecrash 工具了，现在把它也拷到桌面的 crash 文件夹里。至此，crash 文件夹里现在有 4 个文件了，分别是.app, .crash, .dSYM, symbolicatecrash。接下来就是用终端敲命令，生成更易分析的 crash。

./symbolicatecrash /Users/xxxx/Desktop/bugcrash/InOrder.crash /Users/xxxx/Desktop/bugcrash/InOrder.app.dSYM > Control_symbol.crash

上述命令中，"xxxx"和"InOrder"请自行替换成对应的名称。运行，这时候终端可能会报错 Error: "DEVELOPER_DIR" is not defined at /usr/local/bin/symbolicatecrash line 53. 这时候在终端中再输入如下（Xcode6.1.app 依然是要替换成实际名称）

然后再跑一下刚刚的那个命令，这时候看一下桌面的 crash 文件夹下就会多出一个名为“Control_symbol.crash”的文件，我们打开看一下。和刚刚一样。



```
Exception Type:  EXC_CRASH (SIGABRT)
Exception Codes: 0x0000000000000000, 0x0000000000000000
Triggered by Thread:  0

Last Exception Backtrace:
0   CoreFoundation                  0x2f452f7e __exceptionPreprocess + 126
1   libobjc.A.dylib                 0x39c03cca objc_exception_throw + 34
2   CoreFoundation                  0x2f3897c6 -[__NSArrayM objectAtIndex:] + 226
3   InOrder                         0x000fe6a2 -[ORDDatePicker setTime:] (ORDDatePicker.m:438)
4   InOrder                         0x000fde5e -[ORDDatePicker buildControl] (ORDDatePicker.m:331)
5   InOrder                         0x000fc79c -[ORDDatePicker initAtOrigin:mainColor:] (ORDDatePicker.m:120)
6   InOrder                         0x0011656e -[ORDDetailViewController textFieldShouldBeginEditing:]
(ORDDetailViewController.m:481)
7   UIKit                           0x31d6bc30 -[UITextField canBecomeFirstResponder] + 184
8   UIKit                           0x31cee36c -[UIResponder(Internal) _canBecomeFirstResponder] + 16
```

5.介绍一下如何使用友盟分析工具解析友盟平台生成的 csv 崩溃日志文件

(1)下载 umcrashtool 文件和友盟错误日志.csv 文件

(2)确保对应版本的 dSYM 文件位于以下路径

/Users/ios31/Library/Developer/Xcode/GuDaShi.app.dSYM

(3)将(1)中文件放到同一个文件夹

(4)打开命令行，进入(3)中目录，执行以下命令

./umcrashtool /Users/ios31/Desktop/error/err4.csv

(5)使用系统 number 软件打开(4)中生成的 err4-symbol.csv 文件，可以看到具体崩溃文件和行数

6.注意事项：

(1)解析成功与否关键在于 crash 文件、app 文件和 dsym 文件的 uuid 是一致的，一旦有个不一致，是解析不出来的。

(2)获取文件 uuid 方法：

dwarfdump —uuid YourApp.app/YourApp

dwarfdump —uuid YourApp.app.dSYM

crash 文件，在

```
Binary Images:
0x100034000 - 0x100653fff GuDaShi arm64  <99d2042e0e2c3f0f87df3ac208a3071a>
```

(3) 每次都去敲命令来解析 crash log 本身就是一件很蛋疼的事情，但这还不是麻烦的,最麻烦的是用 symbolicatecrash 还经常遇到问题:怎么 crash log 又解析失败了？怎么批量解析 crash log？那么，有什么好的办法没？答案是肯定的

举个例子，如果我们的应用是在自己的机器上编译生成的，把应用装在真机上如果有崩溃产生，把真机通过数据线连接到Mac电脑上，打开Xcode菜单上的Window——Organizer，找到设备的device logs项中的crash log，稍等片刻，你就会发现这里的crash log已经被自动解析过了（大部分情况会自动解析，如果不行请右键点击选择Re-Sysbomlicate）。

但是如果应用不是在自己的编译上生成的，你会发现 organizer 不会自动解析 crash log(除了系统函数)。怎么在这种情况下也让 organizer 也能自动解析 crash log 呢？其实之前的organizer之所以能自动解析你设备上的crash log，是因为它能根据spotlight的索引来找到对应的.app和dSYM文件,对于这一点,我的猜测是在自己的Mac电脑上编译生成应用时，系统自动对其进行了索引，手动索引的命令是mdimport。比如，把iOS应用的.app和.dSYM文件放到一个文件夹中，执行命令mdimport foldername就可以。命令执行完成后再用刚才的organizer去查看crash log，你会发现也能自动

解析了。这意味着把应用所有版本的.app和.dSYM文件放入一个专门的文件夹中，只要mdimpor这个文件夹，以后的organizer就能自动解析出你所有的crash log。
好处不仅仅是这一点。organizer还有一个import功能，借助这个功能，我们可以把其它Mac电脑上的crash log导入到自己电脑中的organizer，然后就可以自动解析。更好的是，用这个功能可以批量导入收集到的crash log，然后我们就可以批量解析所有的crash log。