了解和分析iOS Crash Report

Quinn "The Eskimo!"
Developer Technical Support
iOS Frameworks Engineer

最终发布帖子文档：Understanding and Analyzing Application Crash Reports

iOS应用崩溃是不可避免的，但是通过分析崩溃报告，开发者可以了解崩溃的原因，并进行修复。本文将介绍如何阅读崩溃报告，以帮助开发者分析崩溃原因，并修复崩溃问题。

本文主要内容：

- 崩溃报告的组成部分
- 如何符号化崩溃报告
- 如何分析崩溃报告，以帮助开发者修复崩溃问题

介绍

Crash Report，又称崩溃报告，是记录程序崩溃时状态信息的文本文件。当程序发生崩溃时，系统会生成一份崩溃报告，以帮助开发者分析崩溃原因，并进行修复。

Crash Report，又称崩溃报告，是记录程序崩溃时状态信息的文本文件。当程序发生崩溃时，系统会生成一份崩溃报告，以帮助开发者分析崩溃原因，并进行修复。

Crash Report 包括以下内容：

- 崩溃时的线程信息
- 崩溃时的寄存器信息
- 崩溃时的内存信息
- 崩溃时的堆栈信息

崩溃报告的内容，取决于崩溃的原因，以及程序的配置。

崩溃报告的内容，取决于崩溃的原因，以及程序的配置。

获取Crash Report的几种方式

获取崩溃报告的几种方式：

- 通过Xcode连接设备，从设备上获取崩溃报告
- 通过iTunes Connect，从App Store用户的设备上获取崩溃报告
- 通过第三方工具，从用户的设备上获取崩溃报告

通过Xcode连接设备，从设备上获取崩溃报告

通过iTunes Connect，从App Store用户的设备上获取崩溃报告

通过第三方工具，从用户的设备上获取崩溃报告

符号化一份Crash report

符号化崩溃报告，是将崩溃报告的内存地址，转换为对应的函数名、文件名、行号的过程。

符号化崩溃报告，是将崩溃报告的内存地址，转换为对应的函数名、文件名、行号的过程。

崩溃报告，是记录程序崩溃时状态信息的文本文件。当程序发生崩溃时，系统会生成一份崩溃报告，以帮助开发者分析崩溃原因，并进行修复。

崩溃报告，是记录程序崩溃时状态信息的文本文件。当程序发生崩溃时，系统会生成一份崩溃报告，以帮助开发者分析崩溃原因，并进行修复。

symbolicatecrash 脚本，位于Xcode的Developer目录下，可以将崩溃报告中的内存地址，转换为对应的函数名、文件名、行号。symbolicatecrash 脚本，位于Xcode的Developer目录下，可以将崩溃报告中的内存地址，转换为对应的函数名、文件名、行号。

Illegal Instruction [EXC_BAD_INSTRUCTION / SIGILL]

当尝试执行一个无法或者未定义的指令时会触发该异常。有可能是因为程序一个配置错误的函数指针的结果下跳转到bad一个无效地址，或ios的情况下，这也可能是由于一个EXC_BAD_INSTRUCTION异常，但是这个通常用来表调试用途。在mac的环境上，Swift会在运行时插入断言指令来加快优化时的崩溃。详情参考Trap Trap。

Quit [SIGQUIT]

这个异常是由子其它进程请有角优先级后可以管理本进程（因此被其他优先级进程kill掉）所导致。SIGQUIT不代表进程发生Crash了，但是它通常反映了某种不合理的行为。例如，如果占用了太长时间，键盘扩展程序会强制要求主app杀不掉，因此，这种情况的原因下不太可能会在Crash report中出现。在实践中，大概率是因为一些其它行为在后台时占用了太长时间是在后台时间限制（看门狗的时间限制，见上文中的表）或功能不了，但是你的调查和extension的流程时候被强制要求执行了，你应该这样声明。什么时候会有一个Termination的原因分析时间，把时间预算的问题和background任务管理问题一起考虑（看近到extension加载程序）。

Killed[SIGKILL]

进程收到系统指令被干掉，通常行查看Termination Reason来区别这种程序干掉的原因。Termination Reason字段中会给出一个命名空间和代码。例如：

- 代码 `0x10badf1` 表示watchOS因后台任务占用了过多内存此时系统导致watch app被干掉。想要解决这个问题，优化后台任务，确保内存使用不过高。或者减少后台的任务运行数量。
- 代码 `0x8badf00d` 表示后台的watchOS应用没有完成指定的后台任务在系统导致watch应用干掉。想要解决这个问题，需要查看Sysq在后台运行时间之内app的处理任务。
- 代码 `0x8badf00d` 表示watch app发现在指定时间内完成后台任务，且系统一直等待可以至于app无法获取足够的CPU时间来完成后台任务。虽然一个app可以通过或者设置后台任务的后台计划来来完成这个问题，但是 `0x8badf00d` 这个错误把大多指向了过高的后台负载。查阅watch各种后台应用。

Guarded Resource Violation [EXC_RESOURCE]

这种是当进程下一个被保护的资源，系统商业把特定的文件强迫终结以为被 `wedged`。因此在同样这些文件被进程的情况下被干扰的原因指出 `EXC_GUARD` 原因（Guard）及特定的保护资源文件描述符，它可以指明被锁定的进程。一个Guard violation 表示后台的watchOS应用以有完成指定的后台任务的系统时间限制可以快速重置，例如，如果一个app长时间了非常频繁的Core Data 存储的SQLite文件的文件锁定，你会发现Core Data的一些从网络sqlite_quard exception日志子文之后出现的日志。通常这种情况，你应该查看CPU Crash report日志在Exception Subtype和Exception Message字段查看是否有 `EXC_GUARD` 原因发现的详细信息。在macOS这种情况下，Crash report中，这类信息会被包括在第一个Exception Code中以信息表述行呈现。它可以被这么解读：

- [63:61] - Guard Type 被保护的资源类型。0x2表示资源文件描述符。
- [60:32] - Flavor 应用特定之下出现的问题。如果第一个(1<<0)的被设置，则进程管试在一个被保护的文件描述符上进行Close()
- [31:0] - Target 被保护的文件描述符。

如果第二个(1<<1)的被设置，则进程管试在被保护的文件描述符上用 `F_SETFD` 或 `F_SETFD_CLOEXEC` 调用 `dup()`，`dup2()`，或 `fcntl()` 命令。

如果第三个(1<<2)的被设置，则进程管试通过socket发送一个被保护的文件描述符。

如果第四个(1<<3)的被设置，则进程管试写一个被保护的文件描述符。

- [31:0] - File Descriptor：进程管试修改被保护的文件描述符。

Resource Limit [EXC_RESOURCE]

这种的资源过限定被强，这各种通讯协议的，表示进程占有了太多的资源。常编的资源包括了 `Exception Subtype` 的字段值。如果 `Exception Note` 字段包含了 `NON-FATAL CONDITION`（非严重错误），则即便是生成了Crash report，进程也不会被kill掉。

- 如果 `EXCEPTION SUBTYPE` 显示是 `MEMORY` 则表示了进程占用已经超过系统限制。如果之后出现提示系统此刚正在进程被kill，可能和这有关。
- 如果 `EXCEPTION SUBTYPE` 显示是 `WAKES` 则表示进程每秒钟唤醒太多次了，这告每秒CPU唤醒时间要减少且是或者最频繁。通常，这种事发生在进程后台（通过 `performExpiringActivityWith` 或者 `dispatch_async`），而且会以这种方式发生的原因。因为发生这种异常的通信或被加长的后台时间，所以很多时后台进程会出现或者重复同样情况直到一些后台任务了它们重必须大小的。

Other Exception Types

如果你的Crash report可能会出现无名的Crash type，这时候会出现个字段上的总结信息与进制数据（例如0000000）。如果你收到这样的Crash report，直接在Exception Code查看更多信息。

- 如果Exception Code是 `0xdeaddaad` 则说明此应用程序被强制杀掉，并且crash report，可以通过同样（手机）做出强制结果重新来记录被杀掉，通常情况下，这种Crash是用户主动杀掉的，并非表示错误。
- 如果Exception Code是 `0xbad22222` 表示一个watch应用因为频繁智能唤醒而系统终止掉。
- 如果Exception Code是 `0xc00ffee0` （或是其他0xc00ffee0）则说明一个应用因为频繁了看门狗的系统时间系统终止掉，每可能是在每周了太长时间后台，终止，或者是调后系统事件，一种常见原因是在后台时间限制内未结束任务管理，不过下面的Termination Reason字段会提示这种情况。
- 如果Exception Code是 `0xc00ffee1` 则说明此应用不能启动。
- 如果Exception Code是 `0xc00ffee2` 则说明此应用不能启动，因为在watchOS上的一个应用不能在后台启动。
- 如果Exception Code是 `0xc00ffee3` 则说明此应用不能启动，因为在watchOS上的一个应用不能在后台启动。

Application Specific Information:
com.crashlytics.ios/crashlytics.framework: failed to stream write_to_fd() (errno=28, No space left on device)

Exception Type: EXC_CRASH (SIGABRT)
Exception Codes: 0x0000000000000000, 0x0000000000000000
Exception Note: EXC_CORPSE_NOTIFY

Termination Reason: Namespace SPRINGBOARD, Code 0x8badf00d
Terminating process: exit(0)
Application Specific Backtrace 1:
0x1a2e7a0c0 0x1a2e7a0c0 0x1a2e7a0a0 0x1a2e7a0a0 0x1a2e7a0a0 0x1a2e7a0a0 0x1a2e7a0a0 0x1a2e7a0a0
Application Specific Backtrace 2:
0x1a2e7a0c0 0x1a2e7a0c0 0x1a2e7a0a0 0x1a2e7a0a0 0x1a2e7a0a0 0x1a2e7a0a0 0x1a2e7a0a0 0x1a2e7a0a0
Thread 0 name: Dispatch queue: com.apple.main-thread
Thread 0 Crashed:
0 ThetSnapshot 0x0000000100000000 -[ThreadsTimeoutViewController myTimeoutCode:]
1 UIKit 0x0000000196c4e978 -[UIViewAnimationInstance sendDeceleratedCallsToDelegate:]
2 UIKit 0x0000000196c4e738 -[UIViewAnimationInstance animationDidStop:finished:]
3 CoreAnimation 0x0000000196e7a4d4 -[CAAnimationDelegateProxy animationDidStop:finished:]
4 libdispatch.dylib 0x0000000196e5e9f8 _dispatch_client_callout + 20
5 libdispatch.dylib 0x0000000196e5e1dc _dispatch_main_queue_callback_4CF + 1004
6 CoreFoundation 0x0000000196e4f0c8 __CFRunLoopRun + 1968
7 CoreFoundation 0x0000000196e4e5b8 CFRunLoopRunSpecific + 444
8 GraphicsServices 0x0000000196f3119b GSEventRunModal + 108
9 UIKit 0x0000000196c5a3d8 -[UIApplication _run] + 684
10 UIKit 0x0000000196c1c988 UIApplicationMain + 208
11 ThetSnapshot 0x0000000100000a64 main (main.m:25)
12 libdyld.dylib 0x0000000196e5e0d8 start + 0
Thread 1:
0 libsystem_kernel.dylib 0x0000000196e6a2a8 __workq_kernreturn + 8
1 libsystem_pthread.dylib 0x0000000196f7f1b8 _pthread_wqthread + 968
2 libsystem_pthread.dylib 0x0000000196f7e9a4 start_wqthread + 4
...

第一行用法了当前的线程，以及当前的执行时间。其余各行均为此刻的一个堆栈中堆栈对应信息，从左到右分别是：

- 堆栈行号，堆栈的起始顺序会和调用顺序一致，并且给出堆栈等级进入时间执行的函数，只有1个函数用并指向函数，以此类推。
- 在堆栈对应中显示的执行函数的名称
- 并且的代表机器指令在堆栈上的实际内存地址，其它并且表示如果并且执行完成之后下一个执行的并且地址
- 在一个符号化的Crash report中，代表在堆栈对应中的函数名称

异常

Objective-C中的所有通常用来表明在运行时发生的代码错误。例如数组访问越界，或者发送(nonnullable)的参数，没有实现selector中会源发现的方法，或者给键值变化发送识别的对象发送信息。

Note：自定义的异常的对象发送其自定义 `NSUnrecognizedSelectorException` 异常类型Crash。而你的异常的结果，这会在崩溃报告的第之后会出现了之前的发送信息代表的时，如果你的异常是由一个 `NSUnrecognizedQueueException` 发生Crash（在堆栈信息中查看 `NSObject(NSDeprecated)doesNotRecognizeSelector:]`），考虑通过 Zombies instrument 来profiling的过程，来排除掉才是别的内存管理问题。

如果异常没有被捕获，它会导致一个 `uncaught exception` 万法用终止。默认的 `uncaught exception` 的日志会显示到设备的控制台，之后会终止进程。异常堆栈信息会生成的crash report的上一个异常堆栈（Last Exception Backtrace）下，然后再执行时，异常消息会被Crash report记录。通常会提取了一个带有上一个异常堆栈（Last Exception Backtrace）的crash report，但是当在异常期间设备并获取其控制台日志信息，来更好的了解发生crash的原因。

List10：发生了上一个异常堆栈（Last Exception Backtrace）的未符号化crash report摘录

```
Last Exception Backtrace:
0x18bcec418 0x18b851c5c 0x18bec5a98 0x187f0a1a8 0x187f0a1a8 0x187f0a1a8 0x187f0a1a8 0x187f0a1a8
```

一个只包含16进制信息的Last Exception Backtrace信息的crash日志会非常难符号化。这即取决于你的崩溃信息，能像用来符号化。

列表11：一个包含Last Exception Backtrace信息的符号化的crash report，这个异常出现在加载app的storyboard时，需要调试的时候NSLoadNibFailed这个关键字。

```
Last Exception Backtrace:
0   CoreFoundation          0x19ec4e1c8   _exceptionPreprocess + 124
1   libobjc.A.dylib         0x19b851c5c   objc_exception_throw + 56
2   CoreFoundation          0x19ec5e5d8   -[NSCoder(NSExtra) raise] + 12
3   FaceSetlan              0x187b5a1a8   -[NSObject(NSObjectScalarLoading) setValue:forKeyp] + 27
4   UIKit                   0x187e3b2fc4  -[UIStoryboardUnit setValue:forKeyp:] + 184
5   UIKit                   0x187e3b1f28  -[UIStoryboardUnit initWithCoder:] + 123
6   UIKit                   0x187e3b1d8c  -[UIStoryboardObjectUnit initWithCoder:] + 48
7   UIKit                   0x187e3a8f4   -[UIBlob instantiateWithOwner:options:] + 1756
8   UIKit                   0x187e3f4128  -[UIStoryboard instantiateInitialViewControllerWithOptiof
9   UIKit                   0x187e3f4128  -[UIStoryboardSceneManager instantiateFromDefaulto
10  UIKit                   0x187e3f4c7c  -[UIStoryboardSceneManagerTemplate _perform:] + 56
11  UIKit                   0x187e3f7f38  -[UIStoryboardSceneManager perform:] + 168
12  UIKit                   0x187e4e4504  -[UITabletOrder _selectRowAtIndexPath:animated:scroll:
13  UIKit                   0x187e4e4ec   _overTarget:withOrder:withSelectedIndexPath:withSelectedI
14  UIKit                   0x187e4f1d8c  _runAfterCACommitDeferredBlocks + 292
15  UIKit                   0x187e4e1d8c  _afterCACommitHandlerCallbacks + 1024
16  UIKit                   0x187e4e1d8c  _afterCACommitHandler + 168
17  CoreFoundation          0x19ec5d178c  __CFRUNLOOP_IS_CALLING_OUT_TO_OBSERVER_CALLBACK_
18  CoreFoundation          0x19ec5d0ec   __CFRunLoopDoObservers + 319
19  CoreFoundation          0x19ec5d0fbc  __CFRunLoopRun + 1824
20  CoreFoundation          0x19ec5c8d8   CFRunLoopRunSpecific + 444
21  GraphicsServices        0x19b6f3f28   GSEventRunModal + 108
22  UIKit                   0x187e4c1988  -[UIApplication _run] + 684
23  UIKit                   0x187e4c1988  UIApplicationMain + 208
24  ThetSnapshot            0x100ec4c0c   main (main.m:25)
```

如果你发现未可以被编译的异常并没有被捕获，请确定定义宏在Guidelines应用程序中的设置为了 `no_compact_unwind` 标签。

64位OS使用了 `zero-cost` 的异常实现机制，在 `zero-cost` 系统里，每个函数都有一个额外的数据，它会描述如果一个异常发生在这个特定的函数内部发生，该如何展开的信息。如果一个异常发生在这个特定的函数里没有可展开的数据，那么异常处理函数会然无法跟踪并记录，也许在堆栈帧上面的地方有异常处理函数，但是如果那里没有一个合适的异常处理，这会从发生异常的地方直接崩溃。所以在不编译的函数里加异常（也就是说无法通过部分函数被显示的函数的异常）。

Thread State(x86/arm64)

这章列出了crash报告的线程状态。这里列出了注解过的值，在你进一个crash report的时候，了解线程状态并会这些，但是这里的信息对除了解Crash的用不，这点在你的报告。

列表12：ARM64设备的crash report的一段Thread State摘录

```
Thread 0 crashed with ARM Thread State (64-bit):
    x0: 0x0000000000000000   x1: 0x0000000197779c8   x2: 0x0000000000000000   x3: 0x0000000197779c8
    x4: 0x0000000000000000   x5: 0x0000000000000001   x6: 0x0000000000000000   x7: 0x0000000000000000
    x8: 0x0000000000000000   x9: 0x0000000000000000  x10: 0x0000000197187f0   x11: 0x0000000000000007
   x12: 0x0000000000000000  x13: 0x0000000197fb409   x14: 0x0000000000000000  x15: 0x0000000000000000
   x16: 0x0000000197ef168   x17: 0x0000000101a0d8c   x18: 0x0000000000000000  x19: 0x0000000000000000
   x20: 0x0000000197ef050   x21: 0x0000000000000001   x22: 0x0000000197df780   x23: 0x0000000198fc0d9
   x24: 0x0000000000000000   x25: 0x0000000000000000   x26: 0x0000000000000000   x27: 0x0000000000000000
   x28: 0x0000000000000000    fp: 0x0000000197ef0e0    lr: 0x0000000198fc178
    sp: 0x0000000197ef060    pc: 0x0000000198fc1b4  cpsr: 0x60000000
```

Binary Images

这一章列出了这个崩溃程序此时加载在进程中的二进制文件（binary images）。

列表13：一段crash report的完整二进制文件摘录

```
Binary Images:
0x100000000 - 0x100007fff ThetSnapshot arm64 <2dcf8bacd8f33e2af4d5f8c2a5e5d5c2> /var/containers/Bundle/A
...
```

每一行都包含了一个二进制文件的以下细节信息：

- 在进程内的二进制文件的地址空间
- 一段二进制的名称和版本（仅针对macOS），一个MacOS的crash report，如果二进制的OS的一部分，也包括哪些之一。
- （仅针对macOS）二进制的唯一标识符（version和Bundle版本，通过新版本号分别）。
- （仅针对iOS）二进制文件的路径，一个二进制文件的位置，一个二进制文件的位置，其中只有一个可以被加载到进程中。
- 一个可以被一张二进制文件的（仅针对macOS），这个值会随每一次构建而发生变化，并且它会用来定位需要符号化的dSYM文件。
- 最后，二进制文件的arch。

读懂机内存 report(Low Memory Reports)

当系统检测到内存不足时，iOS系统要通知各应用程序的内存系统告诉各应用来释放内存。每个运行着的应用都会接收到系统发来的内存警告（Low-memory notifications）。要求释放内存空间，从而迫使减少整体内存消耗的目的。如果内存压力依然存在，系统可能会终止最后的最近或最低优先级的应用，以释（释放）内存用来腾出空间。你的应用可以被提前杀：不然，你的应用会被iOS杀掉。因为可怜的应用会因此内存不足，这时候会生成一个内存报告（Low-Memory Report）并存储在磁盘目录中。然后在report目录的文件夹下以Crash report格式写入。它会写出用的堆栈信息，一个内存报告report的Header会包含Crash report的header和各部分，紧接着是header的信息子字段的系统信息的内容信息，这是子百大小（Page Size）字段，每一个进程的内存占用大小是根据内存的页的数量来 report的，一个内存报告report数量要的部分是进程表格，这个表格列出了所有的运行进程，包括系统在生成机内存report时的运行进程，如果一个进程被广广，会在此表列一列进入表格的进，一个进程可能表现在的原因有：

- **[per-process-limit]**表明占用超过了它的最大内存量，每一个进程在常规的工作上的限制是不过高的为每个应用分配了的，超过这个限制会导致进程被系统干掉。
- 注意：扩展程序（Extension app, 例如输入法扩展或小内存键盘），一些技术，例如地图相关的和Siri(siri_d 占用中是等的是最的内存，因此它是会用在扩展程序里。
- **[vm-page-shortage]**和**[vm-thrashing]**[vm]由于系统内存压力最不得，
- **[vnode-limit]** 打开文件文件了。

注意：系统会尽量避免在vnodes已经达到的时候干掉某程序，因此你的应用如果在后台，即使并没有占用什么vnodes，因有可能被杀掉。

- **[highwater]**一个系统守护进程超过了它的内存限制，Locating Memory Issues in Your App 的这
- **[jetsoned]**进程因为其它不可接受的原因被杀掉。

如果你没有在你的应用或者扩展程序里看到那些，那crash的原因就不是你的内存压力，仔细查看一下crash report（在之前章节里有写）。

当你发现一个机内存Crash，与其担心哪一部分的代码出现问题，还不如好好研究一下自己的内存使用习惯和打对您的内存管理（Low-memory warning）的处理情况。Locating Memory Issues in Your App 的这 了如何使用Leaks Instrument工具来找到泄漏，和如何使用Allocations Instrument和Mark Heap 来跟踪内存消费。Memory Usage Performance Guidelines 讨论了如何处理接受到的内存告警的问题，以 及如何高效地使用内存，当然，也推荐在查看了Apple的内存管理的 Advanced Memory Analysis with Instruments 和一章节。

重要：Leaks和Allocation工具不能检测所有的内存使用情况。你需要和VM Tracker工具一起运行你应在Allocation工具重现重要的内存运行，默认VM Tracker是不可用的。如果想通过VM Tracker 和profile的目的，请在Instruments工具，选择“Automatic Snapshotting”和设定要多长动态产 Snapshot Now”按钮。

相关文档

如果您要如何使用Zombies模拟工具来恢复内存释放的Crash，可以查看Eradicating Zombies with the Zombies Trace Template，这个模板是专门用于调试的信息，请参考 App Crashes on Launch，如果您了解关于Crash log的解读，请参考 Understanding Crash Reports on iPhone OS WWDC 2010 Session 。

分享： iOS 标签： iOS

登录/注册后可立即获得以下权益：
免费试学体验
精选推荐内容
查阅浏览记录
收藏喜欢内容
订阅优质专栏
关注优秀作者
立即登录
登录/注册代表同意《隐私政策》