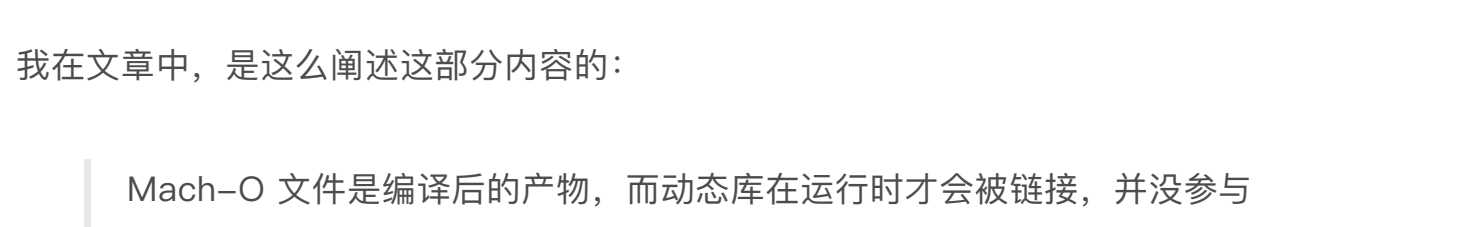


11 | 热点问题答疑 (一) : 基础模块问题答疑

戴铭 2019-04-04



你好，我是戴铭。

专栏上线以来，我通过评论区收到了很多同学提出的问题、建议、心得和经验，当然提的问题居多。虽然我未在评论区对每条留言做出回复，但是我对大家提出的问题却都一一记录了下来，等待这次答疑文章的到来。其实，不光是在留言区，也有一些朋友通过私信跟我反馈了学习专栏遇到的问题。

所以，今天我就借此机会，汇总并整理了几个典型且重要的问题，和你详细说一说，希望可以解答你在学习前面 10 篇文章时的一些困惑。

动态库加载方式的相关问题

@五子棋在看完第 5 篇文章“[链接器：符号是怎么绑定到地址上的？](#)”后，关于动态库是否参与链接的问题，通过私信和我反馈了他的观点。他指出：动态库也是要参与链接的，不然就没法知道函数的标记在哪儿。

为了帮助大家理解这个问题，我把与这个问题相关的内容，再和你展开一下。

我在文章中，是这么阐述这部分内容的：

Mach-O 文件是编译后的产物，而动态库在运行时才会被链接，并没参与 Mach-O 文件的编译和链接，所以 Mach-O 文件中并没有包含动态库里的符号定义。也就是说，这些符号会显示为“未定义”，但它们的名字和对应的库的路径会被记录下来。运行时通过 `dlopen` 和 `dlsym` 导入动态库时，先根据记录的路径找到对应的库，再通过记录的名字符号找到绑定的地址。

细细想来，这种说法并不严谨。关于这个问题，更严谨的说法应该是，加载动态库的方式有两种：

一种是，在程序开始运行时通过 `dylid` 动态加载。通过 `dylid` 加载的动态库需要在编译时进行链接，链接时会做标记，绑定的地址在加载后再决定。

第二种是，显式运行时链接（Explicit Runtime Linking），即在运行时通过动态链接器提供的 API `dlopen` 和 `dlsym` 来加载。这种方式，在编译时是不需要参与链接的。

不过，通过这种运行时加载程动态库的 App，苹果公司是不允许上线 App Store 的，所以只能用于线下调试环节。关于这种方式的适用场景，我也已经在文章第 6 篇文章“[App 如何通过注入动态库的方式实现极速编译测试？](#)”中和你举例说明过，你可以再回顾下相关内容。

在第 5 篇文章中，我将动态库的这两种加载方式混在一起说了，让你感到些许困惑，所以在这里我特地做个补充说明。

接下来，我们就再看看第 6 篇文章后的留言。这篇文章留言区的问题集中在：项目中使用了 CocoaPods 来开发组件，在使用 `InjectionIII` 调试时，遇到了修改源码无法进行注入的问题。

在这里，我首先要感谢 @manajay 同学在 `InjectionIII` 的 issue 里找到了相关的解答，并分享到了留言区。

其实，关于 `InjectionIII` 的这部分内容，我更希望你能够了解 `InjectionIII` 的工作原理，从而加深对运行时动态库加载原理的理解。然后，根据自己的工程情况动手改造或者直接造个新轮子，我相信会极大地提升你的技术水平，至少比直接使用已有轮子的效果要好得多。

所以，还是回到我在[开篇词](#)和你提到的观点：动手就会碰到问题，就会思考，这个主动过程会加深你的记忆，这样后面再碰到问题时，你会更容易将相关知识串联起来，形成创新式的思考。但如果你在碰到困难时，就选择放弃那必定会抱有遗憾。

在第 8 篇文章“[如何利用 Clang 为 App 提质？](#)”中，@鹏哥同学在评论区问了我这样一个问题：

在第 1 篇文章“[建立你自己的 iOS 开发知识体系](#)”中，你提到对某一领域的知识要做到精通的程度，而不能只是了解。那么，你在这个专栏中提到了这么多内容，我应该选择哪些内容去深入研究呢？还是说所有的内容，我都需要去深入研究？

我给出的回答是，根据工作需要来选择。比如说，如果调试速度的问题，确实是你目前工作中面临的最大挑战，那我觉得你就应该在这个点上深挖，并勇敢地克服其中遇到的困难，就像我上次通过“极客时间”的平台直播时，和你分享的我自己学画画的经历一样，挑战素描的过程确实很痛苦，但挺过来了之后我会很受益并享受自己的进步。对我们这种手艺人来说，不断挑战才能不断进步。

最近我在看一个豆瓣评分非常高的日剧《北国之恋》，在第 3 集“决心”里，一位老爷爷在北海道送别朋友时说了一番话，我觉得特别有力量。所以，我把这段话放在这里和你共勉：

不可思议啊，虽然是流行歌曲，不过呢。听到这首歌，这歌流行起来，让人想起那个时代的往事。那年发生了很严重的冻灾，再加上农业机械的引进，农场的经营方式慢慢不一样了。一起来开荒的伙伴们，收拾行李，一个一个地从藤乡离开了。那是 11 月啊，亲密的伙伴们，四家一起放弃了农场，那个时候，我，当然要来送行，稀稀落落地下了雪，那时流行北岛三郎，有四家要离开，来送行的只有我和老婆两个人，大家都一句话也不说。不过，那个时候，我真想把心里想的说出来。你们，这么做事吗？这是输了之后逃跑啊，二十多年来一直在一起努力，你们的心酸、悲哀、悔恨，一切的一切，我自以为都了解。因此，我没有对别人说三道四，没有对别人自以为是指指点点。可是，说这句话的权利我还是有的，你们失败了逃跑了，背叛了我们。逃跑了，这一点，你们给我好好记住。

好了，我们现在继续回到专栏文章上吧。

App 启动时通过 `dylid` 加载动态库，就是运行时动态库加载在 App 启动速度优化上的一个应用场景。在专栏的第 2 篇文章“[App 启动速度怎么做优化与监控？](#)”中，我和你分享了动态库加载后的监控和优化，文后的评论区就有很多同学提到了，想要多了解些动态库加载方面的优化。

关于 App 开始启动到 `main` 函数之间的 `dylid` 内部细节，我推荐你去看苹果公司的 WWDC 2016 Session 406 [Optimizing App Startup Time](#) 视频。这个视频里面，不仅详细剖析了 `dylid`，还提供了构建代码的最佳实践。

除此之外，“[How we cut our iOS app's launch time in half \(with this one cool trick\)](#)”这篇博客，也是个不错的阅读资料。光看名字就很吸引人了，对吧。

关于 App 启动速度的话题，很多同学还提出了其他问题，包括很多关于课后作业的问题。所以接下来，我就针对这个话题再专门做个答疑吧。

App 启动速度的相关问题

专栏的第 2 篇文章“[App 启动速度怎么做优化与监控？](#)”中的大部分问题，我都直接在评论区回复了。下面的答疑内容，我主要是针对课后作业和汇编部分，统一做下回复。

关于课后作业

在这篇文章最后，我留下的课后作业是：

按照今天文中提到的 Time Profiler 工具检查方法耗时原理，你动手实现一个方法耗时检查工具吧。

虽然这个问题的思路，我已经在文章中提到了，但还是有很多同学感觉无从下手。接下来，我们就再一起来看看这个思考题吧。

关于实现思路，文中有怎么一段文字：

定时抓取主线程上的方法调用堆栈，计算一段时间里各个方法的耗时。

现在，我们再一起看一下这个实现思路（我原本未在文中详细展开，是希望多留点思考空间给你）。动手写耗时检查工具时，首先需要开启一个定时器，来定时获取方法调用堆栈。一段时间内方法调用堆栈相同，那么这段时间，就是这个方法调用堆栈的栈顶方法耗时。

这个解题思路里很关键的一步，也是你最容易忽视的一步，就是应该怎么做好获取方法调用堆栈。

`callstackSymbols` 是一种获取方法调用栈的方法，但是只能获取当前线程的调用栈，为了把对主线程的影响降到最小，获取当前线程调用栈的工作就需要在其他线程去做。所以，**这个解题思路就需要换成**：使用系统提供的 `task_threads` 去获取所有线程，使用 `thread_info` 得到各个线程的详细信息，使用 `thread_get_state` 方法去获取线程栈里的所有栈指针。

如果接下来立刻进行符号化去获取方法名，那么就需要去 `__LINKEDIT` segment 里查找栈指针地址所对应符号表的符号，特别当你设置的时间间隔较小的时候，符号化过程会持续消耗较多的 CPU 资源，从而影响主线程。

所以，获取到栈指针后，我们可以不用立刻做符号化，而是先使用一个结构体将栈地址记录下来，最后再统一符号化，将对主线程的影响降到最低，这样获取的数据也会更加准确。

我们可以把记录栈地址的结构体设计为通用回溯结构，代码如下：

```
1 typedef struct SMStackFrame {
2     const struct SMStackFrame *const previous;
3     const uintptr_t return_address;
4 } SMStackFrame;
5
```

在这段代码中，`previous` 记录的是上一个栈指针的地址。考虑 CPU 性能，记录堆栈的数量也不必很多，取最近几条即可。通过栈基地址指针获取当前栈指针地址的关键代码如下：

```
1 // 栈地址初始化
2 SMStackFrame stackFrame = {0};
3 // 栈基地址指针
4 const uintptr_t framePointer = smMachStackBasePointerByCPU(&machineContext);
5 if (framePointer == 0 || smMemCopySafely((void *)framePointer, &stackFrame, size
6     return @"Fail frame pointer";
7 }
8 // 下面的 8 表示堆栈数量
9 for (; i < 8; i++) {
10     // 记录栈地址
11     buffer[i] = stackFrame.return_address;
12     if (buffer[i] == 0 || stackFrame.previous == 0 || smMemCopySafely(stackFrame
13         break;
14     }
15 }
16
```

关于汇编代码的学习

除了课后作业，在这篇文章的评论区中问到的最多的问题就是 `objc_msgSend` 汇编的部分。@西京富贵在评论区留言说到：

看完这篇文章我膨胀了，都敢去翻看 `objc_msgSend` 的源码文件了。嗯，不出意料，一句没看懂。

我想要说的是，汇编并不是必学技能，我们在日常的业务开发工作中也很少会用到。而且，现在编译器对高级语言的优化已经做得非常好了，手写出来的汇编代码性能不一定就会更好。如果你的工作不涉及到逆向和安全领域的话，能够看懂汇编代码就非常不错了。

但是，对于逆向和安全领域来说，掌握汇编技能还是很有必要的。**如果你想学汇编语言的话，同样也需要动手去编写和调试代码，使用 Xcode 工具也没有问题。在开始学习时，你可以按照教程边学边写，其实就和学习其他编程语言的过程一样。**

而具体到 `objc_msgSend` 源码的剖析，你可以参考 Mike Ash 的“[Dissecting objc_msgSend on ARM64](#)”这篇博客，详细讲述了 `objc_msgSend` 的 ARM64 汇编代码。等你看完这篇博客以后，再来看我们这篇文章中的汇编代码就一定会觉得轻松很多。

关于 Clang 的相关问题

专栏已经更新的第 7~ 第 10 这 4 篇文章中，都涉及到了 Clang 的知识以及应用，所以我在这里单独列出了一个问题，和你一起解决关于 Clang 的相关问题。

其实，我在第 7 篇文章“[Clang、Infer 和 OCLint，我们应该使用谁来做静态分析？](#)”中，介绍的 3 款静态分析工具都用到了 Clang，而且 Clang 本身也提供了 LibTooling 这种强大的 C++ 接口来方便定制独立的工具。

当然了，Clang 的知识也是需要投入大量精力才能掌握好。那么，你可能会问，我掌握这些偏底层的知识有什么用呢，好像也解决不了我在现实开发工作中遇到的问题啊？

在我看来，你只有掌握了某个方面的知识，在工作中碰到问题时才能够想到用这个知识去解决问题。如果你都不知道有这么一种方法，又怎么会用它去解决自己的问题呢？

就比如说，你掌握了 Clang 的知识，那在研究[无侵入的埋点方案](#)应该如何实现时，你可能可能会想到用 Clang 的 LibTooling 来开发一个独立的工具，专门以静态方式插入埋点的代码；只有掌握了 Clang 的知识，当你在面对代码量达到百万行的[App 包瘦身需求](#)时，才会想到通过 Clang 静态分析来开发工具，去检查无用的方法和类。

当你掌握了 Clang 的相关知识后，编译前端的技术也就掌握得差不多了；在理解了编译前端的词法分析和语法的分析的套路后，脱离 Clang 的接口完成第 8 篇文章“[如何利用 Clang 为 App 提质？](#)”的课后作业，也就没什么难度了。

在完成这个课后作业之前，你也可以先看看王垠在 2012 年的一篇博客“[怎样写一个解释器](#)”。看完后这篇博客后，你一定会有撸起袖子加油干的冲劲儿。

关于第 8 篇文章的课后作业，如果你还有其他不明白的地方，欢迎继续给我留言。

小结

专栏更新至今，已经发布了 10 篇文章，大家在评论区留下很多高质量的留言，让我非常感动，在这里我也要感谢你的支持与鼓励。

这 10 篇文章学习下来，你可能会觉得这些文章 so easy，也可能会觉得这些文章确实帮你解决了工作中遇到的困惑，还可能会觉得这些文章太难懂了但依旧在努力学习中，我想要和你说的就是：遇到的知识学起来，可能是再坚持一下，并不断重复，只要能比昨天的自己进步一点点，终究可以掌握你想要的知识。

所以，在今天这篇答疑文章，也是我们专栏的第一篇答疑文章中，我不打算大而全地去回复太多的问题，只是甄选了其中其中非常重要、核心的几个问题，和你再一起巩固下我们所学的知识，并和你分享一些我的学习方法。

希望通过今天这篇文章，可以帮你搞明白那些让你困惑的知识点，逐步地建立起自己的知识体系。如果你还有其他的问题，欢迎你给我留言。

最后，虽然这是篇答疑文章，还是要留给你一个小小的思考题。

王垠的博客文章中，除了我在前面提到的“怎样写一个解释器”外，其他文章也都可以帮助你开阔眼界，非常值得一看。在看完他的博客后，你会发现他对编程语言本质的理解非常透彻，而你自己也能从中受益良多。

我在看完他所有的博客文章之后，对很多知识有了更深的理解，但同时知识量也非常大，无法一时都消化掉，感觉需要学习的地方还有很多。所以，我当时的感觉就是酸甜苦辣五味俱全。不知道你看完他的文章后，会有什么感觉呢？我们就把这个话题作为今天文章的思考题，请你在评论区分享一下你的阅后感吧。

感谢你的收听，欢迎你在评论区给我留言分享你的观点，也欢迎把它分享给更多的朋友一起阅读。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

吴开

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

Command + Enter 发表 0/2000字 提交留言

精选留言(19)

绝影

可以分享多一些您关注的博客或博主吗？

作者回复：在后面学习资料推荐篇里我会统一整理好推荐给大哈哈

2019-04-04 6

Mad Mark

关于Mike Ash的那篇objc_msgSend文章，我写过一篇译文 http://madmark.cc/2017/08/01/ARM64_objc-msgSend/

作者回复：赞

2019-04-04 5

时间都去哪了

请问一个新项目在这个时间点用OC还是swift呢?能说说两者应该如何选择呢？

作者回复：新项目用 Swift。不用犹豫

2019-04-07 2

贰叁

垠神是coder的三大浪漫之一——编译原理世界级的大神

2019-04-20 1

陈阿票

将代码编写成动态库，可不可以减少启动时间？

2019-04-10 1

melody

想问下您平时都利用什么途径去获取一些优质苹果的官方文档和博客内容呢？

作者回复：官方文档直接就在官网搜索。发现好的博客文章会订阅博主的博客了，通过 RSS 阅读器看。

2019-04-10 1

Geek_e1d503

老师您好，我想问下并发量大、日流量大的app该注意哪些呢，它的架构和性能上该注意哪些？有没有相关资料可以推荐看下的？

2019-04-04 1

IAMCJ

“一种是，在程序开始运行时通过 `dylid` 动态加载。通过 `dylid` 加载的动态库需要在编译时进行链接，链接时会做标记，绑定的地址在加载后再决定”

所以大部分动态库是会在编译的时候链接是吗？这个时候的“链接”是并没有绑定地址吗？那个“链接”主要做了些什么内容呢？

作者回复：做标记

2019-07-23 1