

Success!

Menu

- [产品](#)
 - [Realm 移动端平台](#)
 - [Realm 移动端数据库](#)
 - [Realm Studio](#)
- [价格](#)
- [文档](#)
- [Support](#)
 - [Overview](#)
 - [Forums](#)
- [Blog](#)
- [Academy](#)
- - [日本語](#)
 - [中文](#)
 - [한국어](#)
 - [English](#)

main navigation



产品

- [Realm 移动端平台](#)
- [Realm 移动端数据库](#)
- [Realm Studio](#)

价格

文档

- [Realm 移动端平台](#)
- [Realm Java](#)
- [Realm Objective-C](#)
- [Realm React Native](#)
- [Realm Swift](#)
- [Realm .NET](#)

Support

- [Overview](#)
- [Forums](#)

Blog

Academy

Language

- [日本語](#)
- [中文](#)
- [한국어](#)
- [English](#)

Free Trial

[Manage your local and synced realm in one place – Download Realm Studio!](#)

[Menu](#)

realm

- [产品](#)
 - [Realm 移动端平台](#)
 - [Realm 移动端数据库](#)
 - [Realm Studio](#)
 - [价格](#)
 - [文档](#)
 - [Support](#)
 - [Overview](#)
 - [Forums](#)
 - [Blog](#)
 - [Academy](#)
 - - [日本語](#)
 - [中文](#)
 - [한국어](#)
 - [English](#)
- [Free Trial](#)

main navigation



产品

- [Realm 移动端平台](#)
- [Realm 移动端数据库](#)
- [Realm Studio](#)

价格

文档

- [Realm 移动端平台](#)
- [Realm Java](#)
- [Realm Objective-C](#)
- [Realm React Native](#)
- [Realm Swift](#)
- [Realm .NET](#)

Support

- [Overview](#)
- [Forums](#)

Blog

Academy

Language

- [日本語](#)
- [中文](#)

- [한국어](#)
- [English](#)

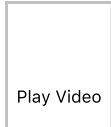
[Free Trial](#)



学习 CocoaPods: Swift、框架以及模块

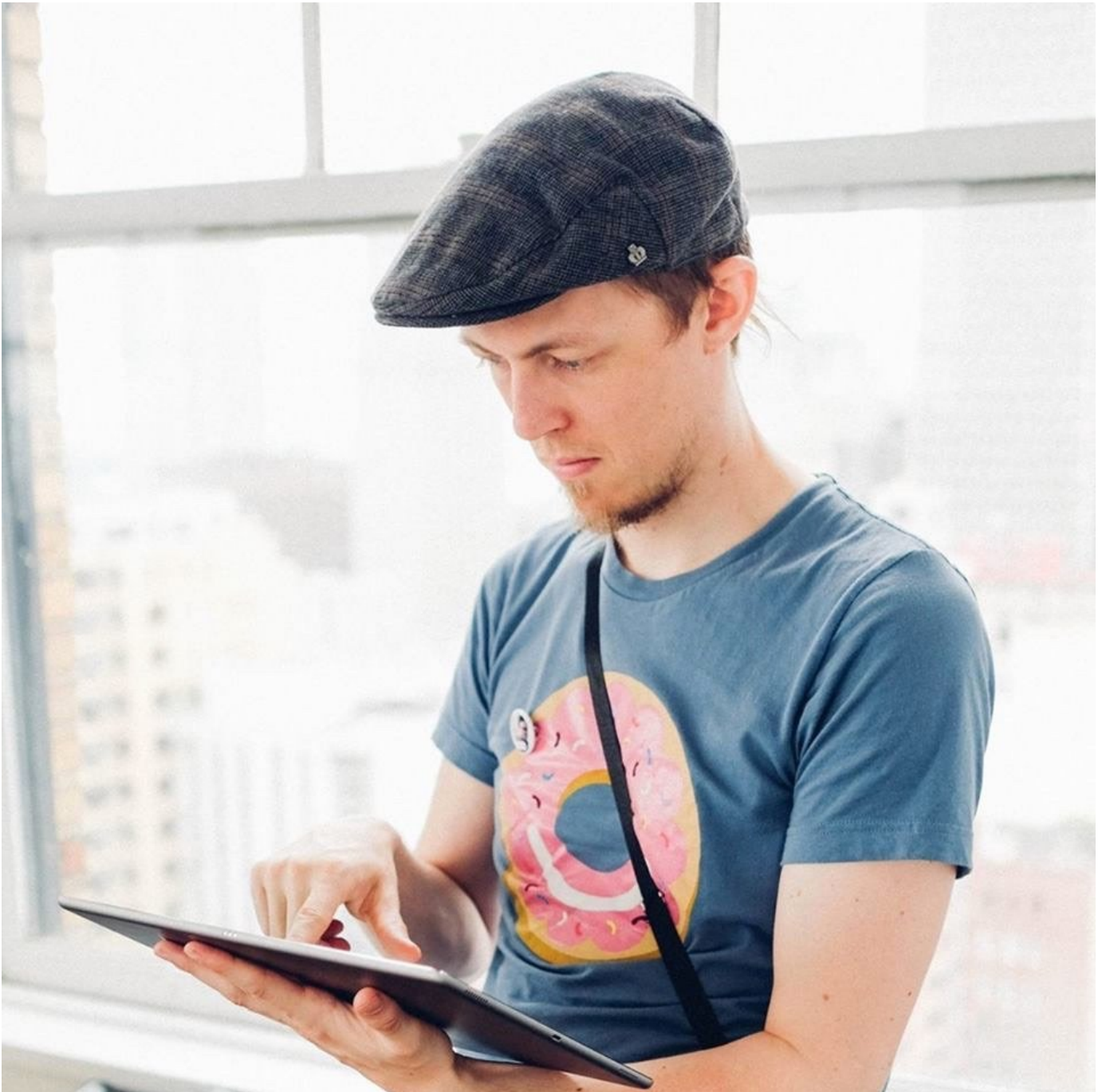
by [Marius Rackwitz](#) [Orta Therox](#)

Dec 4 2015



[< Return to Academy](#)

- [📄 Transcript](#)



[About the speakers](#)[About the content](#)

- This talk was delivered live in October 2015 at [goto: Copenhagen](#). The video was transcribed by Realm and is published here with the permission of the conference organizers.
-
-
-
-

去年年初, CocoaPods 宣布他们正准备发布稳定的 1.0.0 正式版本。然而之后, 苹果推出了 Swift 以及动态框架(framework), 这导致一年后的今天 CocoaPods 仍在努力追赶苹果的步伐。在 [GOTO Conference CPH 2015](#) 的一场讲座中, CocoaPods 动态框架支持的负责人 Marius 和 CocosPods 设计总监 Orta 分享了为 Swift 动态框架建立支持的经历和体验。

Marius Rackwitz 和 Orta Therox 是 CocoaPods 的主要贡献者之一。Marius 完成了 CocoaPods 中所有 *Swiftification* 以及 *Frameworkification* 的工作 (用新的方法替代了旧方法); Orta 则是设计总监, 并且他的贡献远不止于此 (比如说维护本身就是一个庞大的项目的 CocoaDocs, CocoaPods 的官网、插件等等)。他们将会谈论关于 CocoaPods 动态框架的实际过渡流程, 并且提供如何制作动态框架的相关建议。在本次讲座中, 将会包含以下内容:


- **CocoaPods 是什么?**: 它是一个第三方库管理器, 更为重要的是, 它是一个社区。它自建立起已经有五个年头了, 非常成熟和稳定;
- **术语介绍**: 目标(target), 通常存在于您项目中的应用或者应用扩展;
- **编译阶段(build phase)**: CocoaPods 的编译阶段, 不是 Xcode 的。

整合历史 [\(03:36\)](#)

动态框架并不是一直都有提供支持的。在动态框架出现之前, 我们可以使用静态库 (直到 iOS 7 之后便不再可用)。您可以通过 Linked Frameworks and Libraries 来偷使用静态库以及标准框架, 然而这样的话你就不能使用 App Bundle 来分发你得框架了。我们对它们进行了整合工作, 因为它们只能在 OS X 以及 iOS 平台 (仅有的两个平台) 上可用了!

什么是静态库 [\(03:48\)](#)

静态库并不会进行所谓的“链接”, 它们是一系列对象文件的集合 (对象文件通过元信息进行了编译)。它将机器代码从原始位置中提炼出来, 总而言之它们只是编译过的对象文件而已。

当我们使用 Clang 编译我们的源代码的时候, 我们将会得到对象文件, 然后将其放入到 Ranlib 当中 (将它们进行打包)。归档文件中拥有不少内容 (比如说整个文件中某个对象文件位于何种位置), 存有文件信息, 以及对象文件的原始名称。我们为每个架构都进行了归档, 最后使用 Lipo 命令将所有归档文件整合到一起,  得到一个庞大的二进制文件。

在幻灯片上, 我们可以看到一个BananaKit库 (和另一个Monkey库建有依赖关系), 以及一个 SealEye 工具 (这是我们准备构建的程序, 它引用了这两个库)。对于简单的应用来说, 我们只需要把所有东西放在一起, 然后一起编译, 建立关联即可。对于 CocoaPods 来说, 我们会建立排斥项目(exclude project)以及静态库目标。此外 (由于静态库并不包含或者说并不支持资源文件以及其他关联文件), 我们需要诸如关联模型(correlater model)之类的东西。CocoaPods 通过使用一个额外的名为 CocoaPods 的编译文件 (用于所集成的目标) 解决了这个问题, 它是一个资源编译文件。因此, 这和我们使用脚本来编译文件没什么区别: 我们运行 shell 脚本 (存放于另一个 shell 文件当中, 因为重新生成一个单独的 shell 文件很容易, 而让这个脚本存放在 Xcode 中的 Pods 项目中就没那么简单了, 这会导致集成的结果发生较大的变化)。而现在, 我们需要动态框架了 (因为 Swift 只能用它)。

动态框架 (Swift) [\(08:37\)](#)

框架内部 + Cocoa Touch 框架 [\(09:37\)](#)

Receive news and updates from Realm straight to your inbox

对于框架来说, 如果你在 Finder (编译项目中) 中查看其中的内容, 你会发现它不过只是一个文件包而已。比如说 Alamofire 这个非常热门的网络库 (如果我们使用 Swift 来完成网络访问):

- 第一个文件是可执行文件: Alamofire, 一个动态库文件。(实际上, 它们是静态框架; 你可以在这个地方放上其他的静态库, 你会发现它竟然可以正常工作);
- 下面是头文件, 有 Swift 链接头文件, 还有用于对象桥接 Swift 模型的一部分。Umbrella 头文件是非常重要的 (和模型映射文件在一起的), 因为它对定义公共接口来说十分有用。Umbrella 头文件将会被模型映射文件所引用。由这个头文件所导入的所有内容都将是公共模型接口的一部分, 并且和其建立连接的所有对象都可以使用;
- 我们同样还可以看到其余文档信息, 这些信息都会被工具的其余部分所使用。这些都是专门给特定平台所使用的 (我们可以通过文件名看出来);
- Info.plist, 这里面包含了诸多元信息。其中包含了存在 Umbrella 头文件中的相关信息 (比如说版本和版权信息), 默认情况下这些信息都会被用以进行读取。这里有一个很关键的字段——库版本号, 这是一个常量。在框架编译的时候这个版本信息就会从 info.plist 文件中提取出来以供读取。C 文件会读取这个字段, 并且它并不会在错误的时刻从 info.plist 中读取这个字段, 因为它作为二进制文件的

一部分，将会被编译进去。这个信息仅在导入和建立链接的时候有用。如果你将你的应用打包为 App Store 预发布版本，info.plist 会提供关于库文件的相关信息。如果你在应用和应用扩展之间共享代码的话，还需要提供受保护的信息、框架文档等等内容。

动态库 (14:01)

动态库是可以关联映像的文件（在其上运行有链接器）。我们将源代码进行编译，然后在上运行链接器(linker)。这里有 Mac 对象文件，它是一个拥有元数据信息的头文件（比如说依赖库），并且我们会对每个所需的架构重复执行这些个操作流程。对于 Swift 文件来说也是同样的道理，只不过编译过程不一样而已。首先我们把 Monkey 这个 Swift 文件进行编译，然后给我们的公共接口写入数据。接着我们编译好 BananaKit（我们必须将我们所使用的外部符号：MKMonkey，链接给所有引用它的 Monkey Swift 文件、库文件以及注释等等）。我们编译完所有的程序，当然这时候我们仍然不能将其导出，因为程序只是对自身进行了链接，并没有其他的程序链接给它。我们必须给这两个库解决符号引用的问题。

动态框架必须嵌入到包（框架文件夹）当中。构建文件是一个基于名字拷贝的文件。【Marius 运行了一个关于 pod 文件的样式如何的示例】。我们需要拿出一个脚本，允许嵌入到 pods 对于当前编译配置是可用的（Xcode 不允许我们对此进行配置）。头文件需要被分离出来，然后框架需要签名。所有的一切都需要对自己本身进行签名：框架会被单独分配一个签名，其他的可执行文件单独分配一个签名。

框架和静态库的对比 (18:44)

优点：

- 一旦框架构建并编译完毕之后，就更容易进行分发操作以及集成到应用当中。（因为框架将所有内容都整合到了一起，并且允许向包中写入资源文件）。并且它们拥有独立的命名空间。
- 如果你在应用和应用扩展之间通过框架共享代码的话，可以有效减少文件大小。
- 在应用扩展中不同的框架文件夹：你可以框架进行分割，也可以拥有相同框架的两个版本。通过分离包，就可以只允许在框架包当中通过名字访问资源。

缺点：

- 优化变得很困难，因为在你构建好之后就已经建立关联了。
- 由于静态链接的问题，剥离死代码变得很困难。你可以随意导入对象文件，当它们不再需要或者不再被包含的时候只能手动删除。
- 增加加载时间。

对于 CocoaPods 来说这意味着什么？ (23:19)

对于 CocoaPods 来说，我们最终需要：

- 代码扩展（比如说 Clang 模组）。要实现此功能，我们必须允许你在使用 CocoaPods 的时候对我们的代码进行扩展（比如说在使用动态框架的时候）。我们此前这样做的目的是确保人们直到他们正在使用动态框架（当您不再使用的时候我们最后会自行帮您删除掉动态框架）。
- 我们需要集成脚本：在 Podfile 末尾加上这些信息是很有必要的，这样可以使使用过程变得清晰。
- 我们需要借助苹果提供的工具。代码签名非常难以实现（苹果不会在它进行更新的时候发出任何通知）。很明显，我们不能在我们的持续继承流程中加入最终部署到 App Store 的操作。我们需要了解用户的反馈。

Orta：迁移 (25:30)

我曾经将三大应用从 CocoaPods 静态库迁移到了 CocoaPods 动态框架。我使用一种可以将 CocoaPods 迁移到项目中的插件——CocoaPods-deintegrate。这样当你下次运行 pod-install 的时候你就会有一个干净的状态。

总之，这个插件将会删除 pods/ 文件夹，改变某些编译阶段设置，并且将所有 CocoaPods 生成的空 Xcode 组文件夹移除。这样，执行下面三个步骤就会十分简单：在 Podfile 中声明动态框架，这样就会在下次你执行 pod install 的时候将静态库切换为动态框架。接着，执行测试即可（从中你可以找到所有的失败原因）。

常见错误 (27:09)

有这么一些常见的错误：

- Pods 取决于静态库（比如说 Google Analytics、Flurry）。我为了让两个 pods 能够同时工作，我最终需要添加代码让这两个文件链接到一起。另一个问题是所有发布库的人员，你需要开始发布动态框架。
- 库会在 main bundle 中导入资源文件。它们会假设所有资源都已经存放在 main bundle 中了，这意味着你必须处理丢失的资源文件（绝大多数时候）。
- 对于 UIFont Pods，我们需要使用动态链接（我们甚至不能依赖将字体名字写入到info.plist文件当中）。
- Pods 通过使用 #define 来改变行为模式。

使用 CocoaPods 框架的应用 (29:00)

有三个应用转换了过来：**Eidolon**，一个 Swift 应用；**Energy**，花费了两到三天来编辑 pods 文件以及改变内部结构；**Eigen**，最为复杂（超过了60个pods文件）。我们对其进行了转换，将其转为使用动态框架进行工作。在应用中链接有60个 pods 文件，在启动的时候发现它们并没有工作。我的最终解决方案就是将它们转为原来的静态库。作为一个应用，我们不使用 Swift 以及扩展——没有任何理由能说服我们使用它。

帮助以及问答时刻 (32:05)

问：你们对 Carthage 怎么看呢？为什么 CocoaPods 的依赖模型更好呢？

Marius：这个问题很难回答（至少对于后半部分来说，因为要花费很多时间）。我要说的是这两个依赖库管理器实现目标的方式是有很大差异的，CocoaPods 试图完成所有的整合和管理工作，尽可能让用户明白我们的工作。我们的所有工作你都可以看到，因此当你在 Xcode 中使用 CocoaPods 的时候你可以对 CocoaPods 进行一些自定义操作。即时是编译过程都是使用的 Xcode 自身提供的。因此这很容易忽略 Xcode 中这些依赖库的存在，因为你不需要让目前工作的 Xcode 项目持有我们的 pods 文件。如果你不喜欢在 Xcode 工作项目中内含第三方库的话，pod specs 是你的最佳选择。而 Carthage 则给出了另外一种实现方式。对于现有的四个平台来说，我觉得我们更喜欢坚持我们现在所使用的方式。

问：没错，但是有人说通过将第三方项目直接放进项目的方式来管理依赖会增进用户体验，对此你们怎么看？

Orta：是的，Tumblr 应用向我们报告说它们不喜欢这么用，因为它们近百个依赖库都要平均花费5到6秒的时间来编译。当然，目前我们.....

问：所以这个讲座的潜台词是“如果能自己管理框架就自己管理咯”？

Orta：我想有很大一部分应用都会面临这样一个问题，因此我的回答是“或许吧”。对于某些框架或者静态库而言你必须找到一个超级复杂的方法来解决它。

Marius：我觉得这取决于你的应用的实际情况，包括如何使用的依赖库、有多少依赖库、以及是否对你正在使用的外部项目有所了解。第三方依赖库本质上并不是属于“第三方”的，因为它们当中还包含了另外的开源项目。如果你的代码中四处散落着各种小型库的话，那么在加载过程中可能会发生冲突。但是如果一小部分大型依赖的话，那么你就会在加载过程中得到好处（一个大的可执行文件比小的但却拥有大型依赖的可执行文件要花费更多时间加载）。

Orta：对的，就像多面体一样，有很多面，但是体积依然很小，这是很合理的。我们仍旧致力于处理单个或多个的框架，但是目前并没有打算将他们进行整合的计划。

问：很高兴你们尝试在自己的应用中使用为 Swift 版本准备的 CocoaPods。那么你们有没有一个使用所有 CocoaPods 版本的测试应用发布到App Store上了呢？这可以确保苹果仍旧接受你们目前所做的工作？

Marius：没有，我们不会这样做，因为没有有一个比较好的方式来这样做。

问：那这样的话如果出现了问题，有人告诉你们“因为你们的东西苹果拒绝了上架”的话怎么办？

Marius：事实上，在我们的代码团队中有一批经验丰富的开发者正进行着代码核审工作，我们使用自己构建的技术来防止这种事的发生。不过边界情况还是会时有发生，尤其是使用了扩展以及诸如上一个版本提供的共享依赖之类的情况，在 watchKit 扩展或者 watchKit 应用中这些东西很难保证，我们无法通过一个大的集成方案来很好地测试它们。

问：有人有一个想法，如果你们有空的话，为什么不搭建一个服务器，检查能够进行编译并分发给 App Store 的应用所使用最新的 CocoaPods 版本呢？

Marius：如果能够从 iTunes Connect 抛弃版本的话，或许可以试试

Orta：因为你不能保证苹果会.....

Marius：尤其是对于一个工具来说，这样收集用户信息可能会是个大问题，不过如果遵循相关规定的话，这也是可以的。

Orta：我们可以让 Felix 来做这个工作。如果我们礼貌地请求的话，他或许一天就可以搞定了。

问：另一个问题是 pods 文件是否应该经常进行检查，不仅仅是包括支持文件的版本，而且还包括了 pod 文件本身？

Marius：这个问题我们俩都可以回答。我想说这得看情况，但是就我个人而言我更喜欢将 pods 文件放到根目录下，因为这对持续继承很友好。当运行 pod install 的时候，如果有新的额外版本的话，那么无需强制推送到 master 分枝中，只需要用相同的版本提交一个 commit 即可。我们很依赖于远程仓库提供的帮助，当某些东西出问题的时候（比如说强制推送）那么你不应该改变任何核心东西。如果你不这样做的话，如果代码发生了改变，那么可能会发生一些不一样的事情。你无法阻止这种事情发生，因为一旦你在目录中存放了 pods 文件夹，那么它就可能无法再运行 CocoaPods 了，即时 CocoaPods 已经安装了。因此这种做法最好是应该避免的，我知道一些开发者特别喜欢使用 GitHub 所使用的 pods 目录模型。

Orta：在我们所有的应用当中，我们不会这么做；而在我们所有的库当中，我们会这么做。对于库来说，你可以很容易的得到 Carthage 的支持；在进行开发的很长一段时间后，你都可能要回到这个库当中进行一些操作。同样，你不能完全依赖于别人的代码；检查 pods 文件意味着你可以经常让库保证更新，并且保证这个库对你的应用来说是有用的。比如说 Twitter 突然决定替换掉 Twitter pod，并且不再支持所有非官方老版本的 Twitter SDK。如果你检查过你的 pods 文佳后你会发现你仍然可以在你的 fork 中（或者 repo 中）使用这个 SDK。但是通常情况下，我们正在工作的应用都已经开发了很长很长的时间，并且仍然维持着更新。当 pod 消失的话，这很容易在第二天发现问题，我们会为这个问题找到替代方案。在你的代码复查中并没有这些乱糟糟的问题，同样在我们 pods 文件夹中也包含有你应用的键值，在这里面我们

也包含了不少有用的信息。对于人们为什么会问这个问题并没有太大的疑问,在我看来,对于这些问题来说并没有一个很好的答案。我们正致力于解决这个问题,通过一个图表表明这么做的好处和坏处。您对此应该有一个自己的想法。

Orta: 我想设立一个关于 CocoaPods 的新网站(我不想成为写代码的一份子,我只想做设计,如果有人对 Ruby 网站有经验的话,那么是再好不过了)。我想做一个专门谈论 pods 的小网站。你可以在上面看到哪些 pods 是我认为可以共同工作或者会发生冲突的。上面也会有很多函数响应式编程的 pods,或者其他有价值的内容。不管怎样这些内容都是要有价值的,这个网站将会成为人们发表意见的一个全新方式。这正是我所想看到的。

Marius: 我想要看到大家问一些比较容易解决的问题,对于框架编写来说也是一样的。我们仍想要启用“配置依赖 pod 文件”;如果有这个玩意儿的话,我们就能够用拷贝文件、资源文件来进行替代,然后让 Xcode 来完成这个工💎💎💎(这样就可以不用我们所编写的不稳定的脚本,我们尝试完成这个工作,但是没有任何消息透露出来我们如何实现这个功能)。它们必须复制所有有效的文件,然后在我们自己的 shell 脚本当中完成这些任务。在很早以前这已经被证明是很容易出错的,尤其是当 Apple 改变了或者调整了内部的某些东西的时候。我想尽可能实现集成可以更加接近 Apple 所想实现的方式。

Orta: 我对此还要补充一点。当你每次在 CocoaPods 项目中按下“构建”按钮之后,你会发现这个运行的脚本阶段。解决这个问题的话可以为大家解决一个心里的梗。有将近 40,000 个项目在使用 CocoaPods。如果你在使用我们的产品的话,我们一旦解决了这个问题,你就可以节省至少一秒的时间。如果每个人节省的时间加起来,就很可观了。这完全值得有人为此建个纪念网站(当然首先要把我提出的那个网站完成)。谢谢大家!

Marius: 谢谢大家!

About the content

This talk was delivered live in October 2015 at [goto: Copenhagen](#). The video was transcribed by Realm and is published here with the permission of the conference organizers.

Marius Rackwitz

Marius is an iOS Developer based in Berlin. As a Core Member of the CocoaPods team, he helped implement Swift & framework support for CocoaPods. When he's not busy working on development tools for iOS & Mac, he works on Realm, a new database for iOS & Android.

[Twitter](#)

Orta Therox

Orta is the lead iOS developer at Artsy, building beautiful portfolio apps for some of the biggest Art galleries in the world. Encouraged by Artsy's awesome commitment to open source, he regularly devotes time to working on and around the CocoaPods ecosystem, building tools like CocoaDocs, maintaining the Specs repository, and pruning documentation. If the CocoaPods team had fancy titles, he'd probably be called a community manager.

[Twitter](#)

Share

-
-
-
-

新闻精选

新闻精选



by [Jake Wharton](#)

[探索 Java 隐藏的开销](#)

SOLID Principles for Android Developers



Single
Responsibility
Principle

[S 代表着单一职责原则](#)

by [Donn Felker](#)

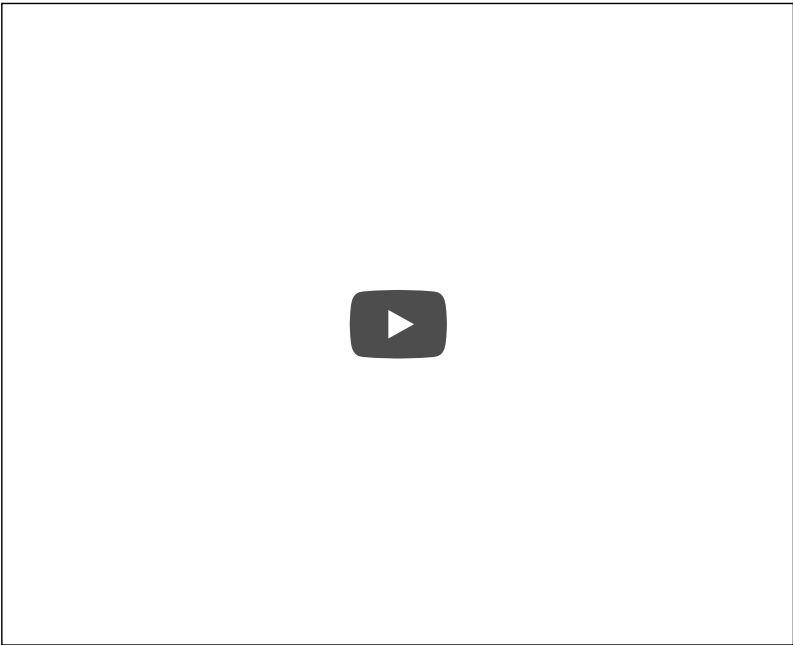


[为什么要用函数式编程？](#)

by [Daniel Steinberg](#)

[更多新闻](#)

[4 design patterns for a RESTless mobile integration »](#)



新闻精选

新闻精选



[探索 Java 隐藏的开销](#)

by [Jake Wharton](#)

SOLID Principles for Android Developers



Single
Responsibility
Principle

[S 代表着单一职责原则](#)

by [Donn Felker](#)



[为什么要用函数式编程?](#)

by [Daniel Steinberg](#)

[更多新闻](#)

\



产品

- [Realm 移动端平台](#)
- [Realm 移动端数据库](#)
- [Realm Studio](#)

[价格](#)



Solutions

- [Realtime Collaboration](#)
- [API Mobilization](#)
- [Offline First](#)
- [扩展](#)



[文档](#)

- [Realm 移动端平台](#)
- [Java](#)
- [Swift](#)
- [Objective-C](#)
- [JavaScript](#)
- [.NET](#)



[Support](#)

- [Overview](#)
- [Forums](#)

[Blog](#)

[Academy](#)



社区

- [Java](#)
- [Objective-C](#)
- [JavaScript](#)
- [Swift](#)
- [Xamarin](#)

[Realm 报告](#)



Realm 公司

- [关于](#)
- [Customers](#)
- [工作机会](#)
- [报道我们](#)
- [法律相关](#)
- [联系我们](#)
- 

Get the latest news in your inbox every week

Subscribe for Realm tutorials, new features, and company announcements

- 
- 
- 
- 
- 
- 

[Realm Blog](#)



[Realm Academy](#)

- 



Get the latest news in your inbox every week

Subscribe for Realm tutorials, new features, and company announcements

- 
- 
- 
- 



[Realm Blog](#)



[Realm Academy](#)



Realm: BUILD BETTER APPS FASTER
© [Realm](#) 2014–2017, all rights reserved.

Thanks for subscribing

You will be receiving an email shortly with details on your subscription

Oops something went wrong

You will not be receiving an email shortly with details on your subscription