茶茶的小屋 (https://www.valiantcat.cn/)
不管生活有多不容易，都要守住自己的那一份优雅。

组件化-动态库实战续

南栀倾寒 (https://www.valiantcat.cn/index.php/author/1/) • 2017-05-15 • iOS (https://www.valiantcat.cn/index.php/category/iOS/)

回忆之前

> *上篇文章中我们已经完美的解决了 使用 `swift 第三方库` ，使用 混编的组件，使用 `use_framework!`，但是会带来别的问题。*
> *果然是生命不息，折腾不止啊。*

不建议在组件化的项目中使用 Swift 来写业务。

Q: C + +/C 静态库依赖问题
A: 回想下我们在做 C 或者 C + + 开发的时候。如果一个静态库依赖另外一个静态库(A 依赖 B)。那么被依赖库 B 升级的时候 A 用重新编译吗？ 不一定，如果是一些方法的新增，维护，不一定会让 A 重复编译；但是如果修改了 B 里面的数据结构，A 里面又用到了这些数据结构，那么很大可能性我们就要重新编译 A 了。
Q: Objective-C 静态库依赖问题
A: 回想下我们在 iOS 中出现上述的依赖问题，貌似也没有见到要重新编译 A 的情景。主要是 Objc2.0 引入了 non-fragile 特性，同时 OC 是严重依赖于 Runtime 的，只要接口兼容，就算你修改了 B 中的数据结构，一般也是不需要重新编译 A 的。如果你不明白 non-fragile 请看文后的参考链接
Q: Swift 中库依赖问题
A: 由于 Swift 不和 OC 一样，所有的 OC 方法都是通过 Runtime 动态调度的。Swift 对于方法是存在静态调度和动态调度 2 种的。所以 Swift 的库依赖极易引起二进制兼容性问题(https://zh.wikipedia.org/wiki/%E5%BA%94%E7%94%A8%E4%BA%8C%E8%BF%9B%E5%88%B6%E6%8E%A5%E5%8F%A3)。更多关于 Swift 库二进制接口(ABI)兼容性问题，请参考文后链接。

Q: 为什么不建议在组件化的项目中使用 Swift 或者和 OC 混编来写业务？
A: 首先在组件化初期的时候，我们能做到的一般是基础库抽离，业务组件分离这些。但是一般来说我们这时候的壳工程，接入这些分离的组件的时候都是使用源码接入，这时候问题暂时显现不出来。
第二步。当我们的组件化的脚步越走越远的时候，我们出于多方面的考虑可能有以下需求。

1. 开发时重复编译是痛点。我们可能更希望提供的是二进制版本，节省下大量的编译耗时；
2. 我们可能要做权限管理。有时候一个公司业务和人员规模都非常庞大。我们基础库设计到跨业务，跨 APP 使用。我们希望不同团队有不同基础组件的读写权限。那么我们更可能偏向提供二进制库加文档的形式。

综上: 由于使用 Swift 开发 ABI 不兼容问题更易出现。在组件化的项目中，不建议使用 Swift 或者混编。

动态库过多问题

上面说到的问题(麻烦)其实是带给开发者的麻烦，但是动态库多了会给用户带来麻烦(APP 启动耗时)。用了混编的项目我们在 `Podfile` 里面势必要写 `use_framework!`，上篇文章中我们也说到用了这个指令。 `CocoaPods` 会帮我们把所有的库全部编译为动态库。这些动态库是在 APP 启动时做去加载的。我们在组件化的时候，自己的业务组件马上接近上百个。可以预想到以后随着组件化的越来越深入，这些库会越来越多。这个时间可能会达到 1s 的量级。对于用户 这是不可接受的。关于动态库过多导致的启动慢的问题请参考文后的参考链接。

结合公司目前的情况的解决方案

我们公司目前的情况: Swift 第三方库个别，混编组件个别。既然都是个别的，我们总不能因为这些个别的特殊 case 让 APP 原本的 1 个二进制文件变成 1 个二进制文件 + 上百个动态库 framework。这肯定是不合理的。

解决办法

1. 不使用 Swift，包括第三方库和混编组件
2. 部分组件(含有 Swift)动态库化，其他部分仍旧整合进 app 的二进制中

   首先来看办法 1 直观感觉是不合适。首先很多公司的项目在做组件化的时候项目已经达到一定程度(没有一定规模也没必要做组件化)，这就意味着大部分 APP 是有历史包袱的。首先重写这些已有的组件或者功能肯定是有风险的，在公司业务多。用户量大的情况下，影响面会更大，虽然这样是一劳永逸的，但是同时风险是更大的。我们在做组件化的工作中，改善大家开发的痛点，提高开发效率才是主要目标。至于重构甚至重写则是业务方的重心。

   第二种办法就是做到部分组件动态库化。

   我们来回忆下静态库的特点。静态库和主工程链接的时候会把库里面的代码复制到可执行文件中。对于这部分符号在 APP 启动时会省去 load，rebase ，binding 的时间。那么在 iOS 平台中嵌入式动态库的特点是不把库里面的代码复制到可执行文件中，而是单独复制到 APP 里面的 frameworks 路径下。所以通常来说动态库节省内存。在 iOS 平台上做不到。静态库的缺点是会让 APP 安装包增大。那么我们自己做的嵌入式动态库也会有这个问题。并且还会导致 APP 启动变慢。那岂不是优点变成了缺点~~.

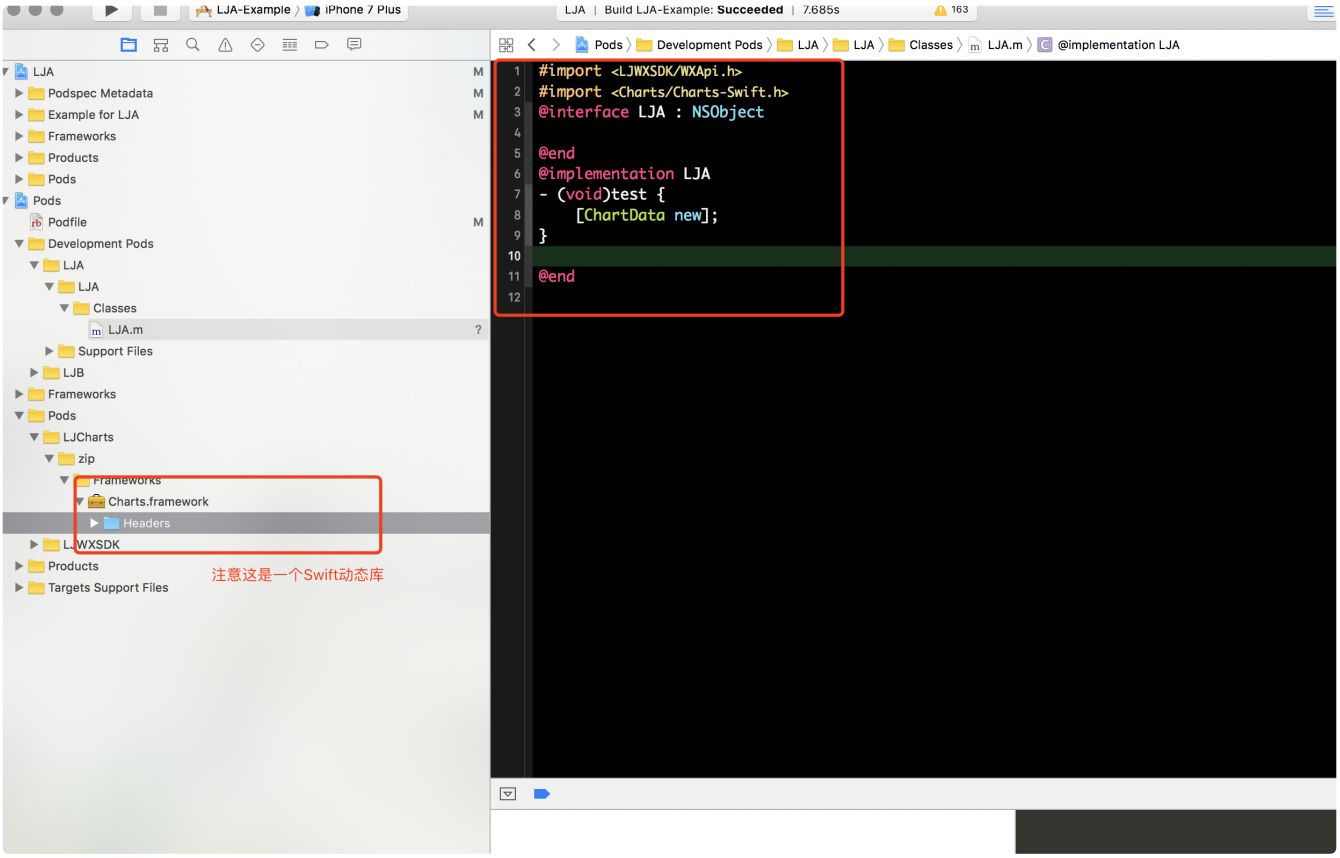
   以上讨论只在正常项目且上架到 APP Store 渠道，越狱开发和企业版证书发布不做讨论

组件化部分动态库实战

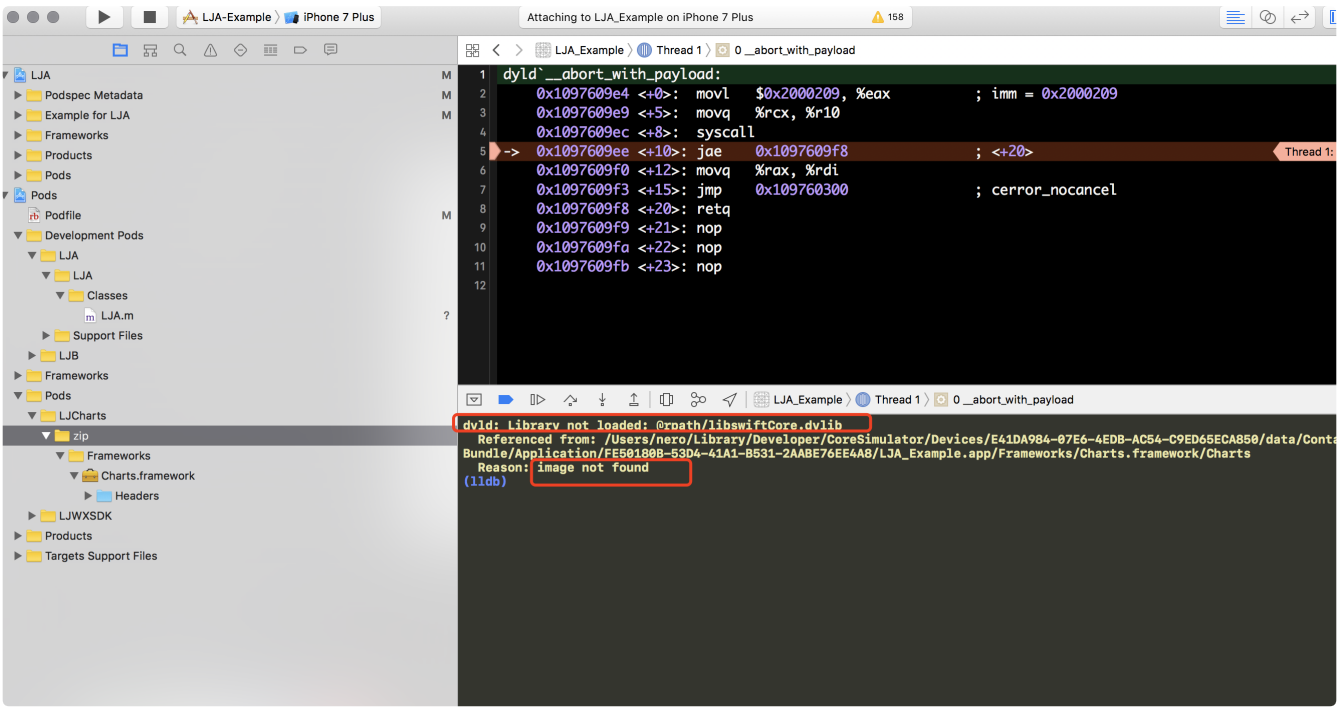
上篇文章中我们知道 只要你的组件库中使用到了 Swift。以 源码 的方式提供给壳工程使用的时候一定要加上 use_framework! ，那么就变成前文说到了上百个动态库了。 那么我们如果不以源码的形式引入呢。对于这些含有 Swift 的下层组件是无依赖的。我们直接将其编译为动态库提供二进制。那么我们在主工程使用的时候就不需要加入 use_framework! .

```
#use_frameworks!

source 'https://github.com/ValiantCat/LJWXSDK'
source 'https://github.com/CocoaPods/Specs.git'  #官方仓库的地址



target 'LJA_Example' do
  pod 'LJA', : path => '../'
  pod 'LJB', : path => '../'

    pod 'LJCharts'
end
```

那么我们拉下来的项目结构是这样的

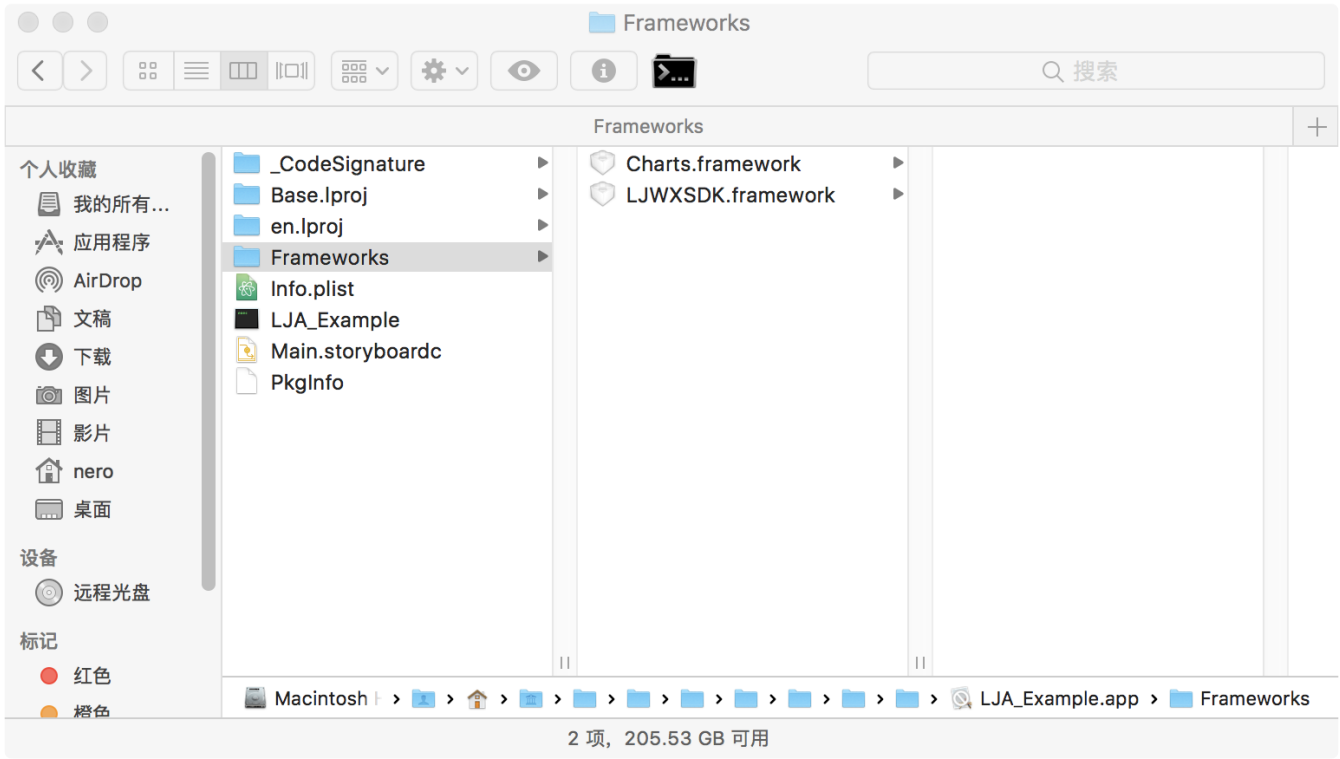


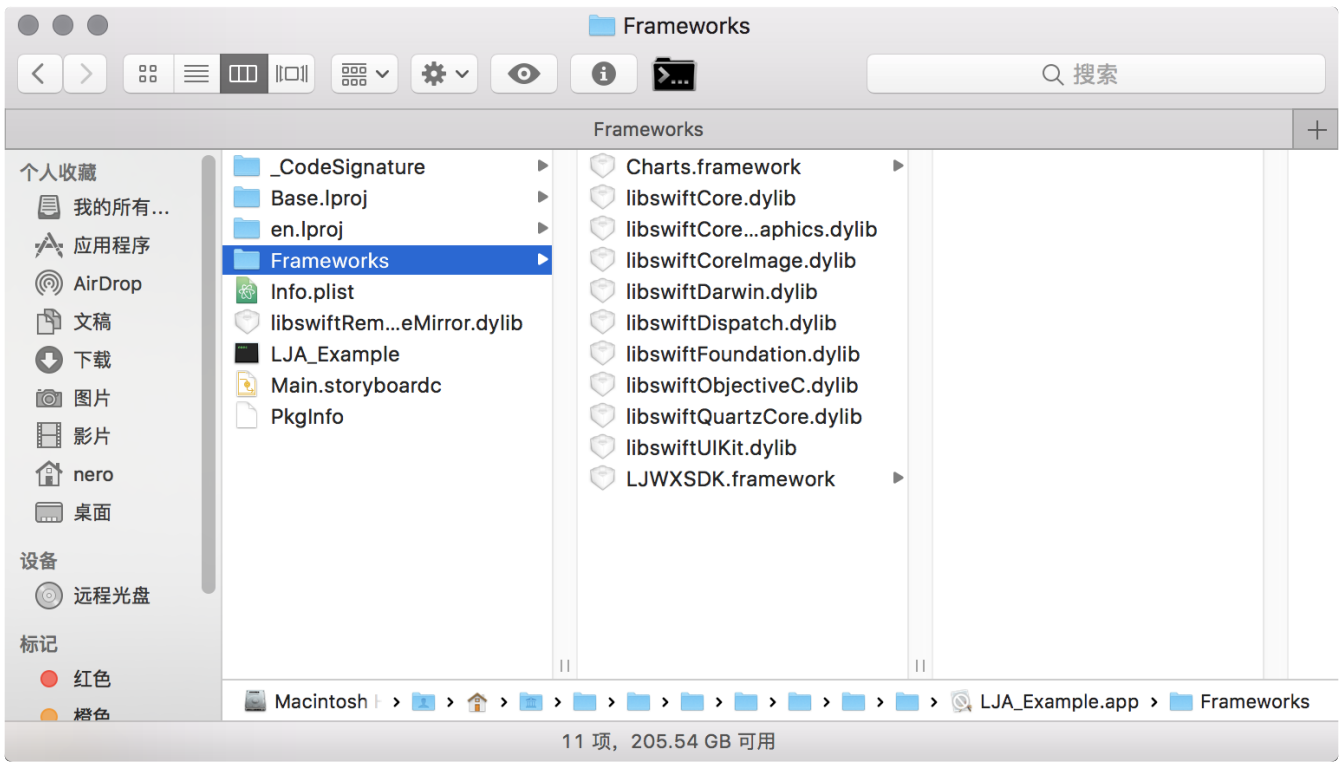
但是我们运行发现 `libswiftCore.dylib` 无法加载。



出现这个原因呢是因为 Xcode 不知道你使用了 Swift 代码，所以并没有把 Swift 的运行时环境(也就是 swift 运行的动态库)复制进 APP 目录。那么解决办法其实很简单。我们在壳工程新建一个 swift 空文件即可。

以下是主工程是否有 swift 文件 APP 目录下动态库的对比





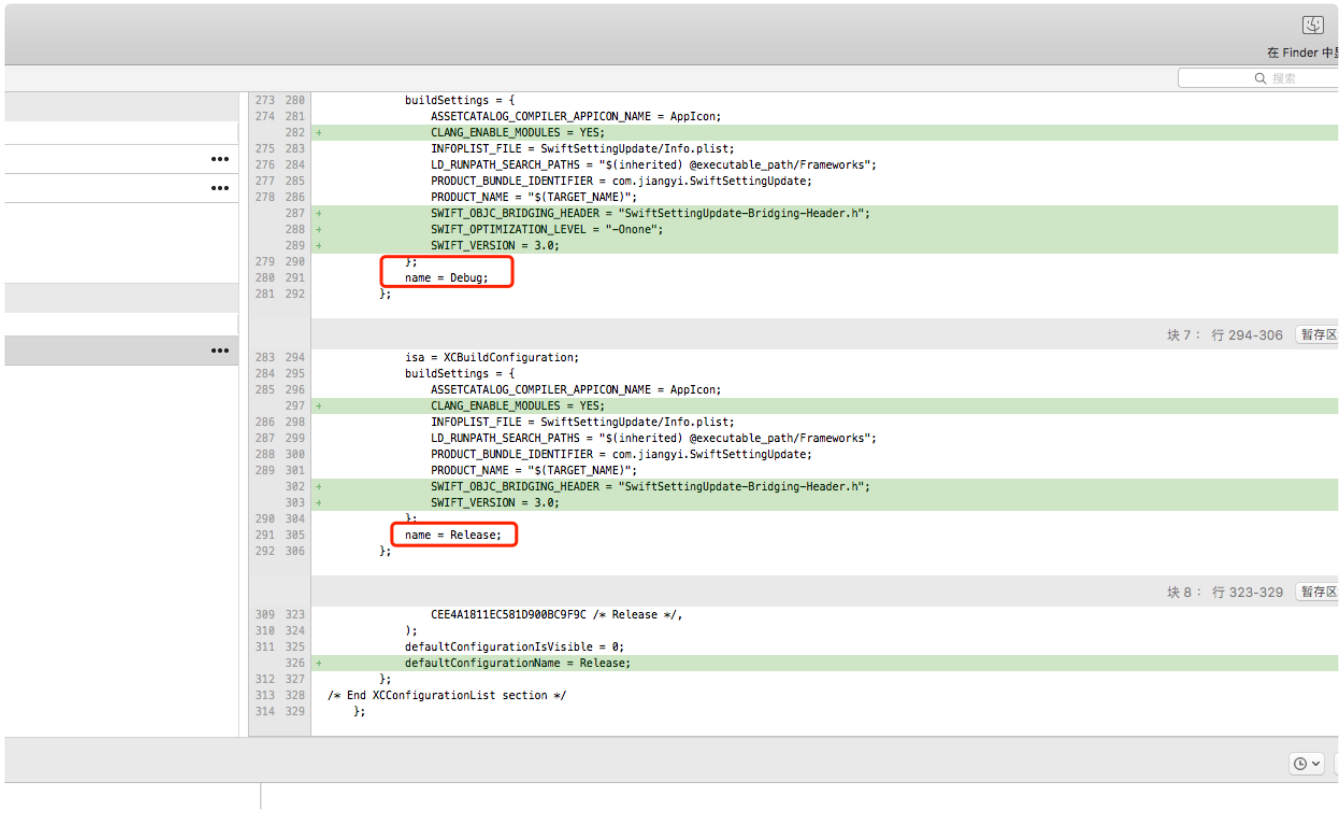
到这里其实这篇文章就好了。剩下的其他组件就继续使用静态库即可。我们可以愉快的玩耍了。

未解决的问题

前面说的问题文中已经解决。但是我觉得有一点不爽。那就是我还需要手动得在壳工程添加空的 Swift 文件。那么我们能不能这一步也自动化呢。

首先我建了一个空工程，往里面添加了一个空的 Swift 文件。然后 diff 了一下两次的 project 文件。

以下是 diff 结果



我们知道 podspec 和 Podfile 其实都是 ruby 代码。Xcode 如何知道我们是否有 swift 其实也是通过工程的配置来知晓的。那么我们其实可以在 Podfile 去写 ruby 代码修改工程文件。这样的话使用方就不需要疑惑为什么要加个空的 swift 文件了。

以下是代码

```
# Pod 设置 ================================

def update_config (config)

  config.build_settings['CLANG_ENABLE_MODULES'] = 'YES'
  config.build_settings['SWIFT_VERSION'] = '3.0'

  # config.target_attributes["LastSwiftMigration"] = "0830"

   if config.name == "Debug" then
      config.build_settings['SWIFT_OPTIMIZATION_LEVEL'] = '-Onone'
   end

#    elsif config.name == "Release" then
#    config.build_settings['CLANG_ENABLE_MODULES'] = 'YES'
#    config.build_settings['SWIFT_VERSION'] = '3.0'
#    end
end

post_install do |installer|


    projects = [
    "SwiftConfig"
    ]

    projects.each do |proj|
        path = "%s.xcodeproj" % [proj]

        single_project = Xcodeproj:: Project.open(path)

        single_project.targets.each do |target|

            target.build_configurations.each do |config|
                print path, ' ', target.name, ' ', config.name
                puts ""

                update_config config
            end
            target.attributes.methods.each do |xx|
              puts xx
            end
        end

        single_project.save
    end
end
```

不过加入这段代码后发现。我虽然确实成功的修改了工程文件。但是发现 Xcode 依旧没有把 Swift 运行时的库给我复制进 APP 里面。
所以这还算是一个不完美的地方。后续有结果的话更新此文。

参考

1. 文中的示例代码 (http://onk2m6gtu.bkt.clouddn.com/%E7%BB%84%E4%BB%B6%E5%8C%96%E5%AE%9E%E6%88%982%E9%83%A8%E5%88%86%E5%8A%A8%E6%8
2. Objective-C 类成员变量深度剖析 (https://quotation.github.io/objc/2015/05/21/objc-runtime-ivar-access.html)
3. Swift 库二进制接口(ABI)兼容性研究 (http://www.jianshu.com/p/5860f5542f21#)
4. Swift 性能相关 (http://www.jianshu.com/p/0d3db4422954)
5. 今日头条 iOS 客户端启动速度优化实践 (https://mp.weixin.qq.com/s?__biz=MzA3ODg4MDk0Ng==&mid=2651113098&idx=1&sn=9edba2142c80f1082b95675c1a836a33&chksm=844c6f57b33be641ee86cda33e066d068
6. 将 Swift 代码编译为静态库 (https://github.com/keith/swift-staticlibs)

标签: 无标签

评论卡

称呼 Email 网址

请填写您的内容。