

组件化-动态库实战



南栀倾寒 (/u/cc1e4faec5f7)  

2017.04.25 18:30* 字数 5042 阅读 1623 评论 34 喜欢 63

/u/cc1e4faec5f7)

原文地址 (<https://link.jianshu.com?t=https://www.valiantcat.cn/index.php/2017/04/24/45.html>),此简书只做备份，强烈推荐原文，毕竟格式比简书好看，还清晰

起因

去年，链家网iOS端，之前由于所有的业务端代码都是混乱管理，造成开发有很多痛点 无法单测 团队成员提交代码冲突机率大 CI配合效果差 功能性代码多端无法复用 单仓库代码量大 编译时间长 等等痛点，领导和组内多次沟通开始着手组件化开发，希望能改进这些开发中的痛点，成立组件化团队。

组件化的方案大同小异，基础性代码封装私有库，业务组件交互交由中间件负责，项目依赖工具用 iOS项目事实上的标准 CocoaPods

前期的基础性组件拆分都较为顺利，从依赖树的叶子节点开发是最合适的方案。随着组件抽离的越来越多，私有库的依赖体系也越来越复杂，慢慢过渡到了业务组件。业务组件用了Swift的第三方组件，用了Swift库的同学都知道必须加上 `use_frameworks!`，这个标记是说Pod管理的依赖全部编译为 动态库，然后呢我们的很多组件又依赖了诸如百度地图，微信分享等 静态库，于是我在执行 `pod install` 报了一个没有碰见过的错误

```
[!] The 'Pods-LJA_Example' target has transitive dependencies that include static b
```

```
+ Example git:(master) x pod install
Analyzing dependencies
Fetching podspec for 'LJA' from '../'
Fetching podspec for 'LJB' from '../'
[!] Unable to find a specification for 'libWeChatSDK' depended upon by 'LJA'

[!] Automatically assigning platform ios with version 8.3 on target LJA_Example because no platform was specified. Please specify a platform for this target in your Podfile. See 'https://guides.cocoapods.org/syntax/podfile.html#platform'.
+ Example git:(master) x pod install
Analyzing dependencies
Fetching podspec for 'LJA' from '../'
Fetching podspec for 'LJB' from '../'

-----
Notice:LJA SDK is zip now
-----

Notice:LJA SDK is zip now
-----

Downloading dependencies
Installing LJA 0.1.0 (was 0.1.0)
Using LJB 0.1.0
Using LJA SDK 0.1.0
Installing libWeChatSDK 1.7.51
[!] The 'Pods-LJA_Example' target has transitive dependencies that include static binaries: (/Users/hero/Desktop/Static_Dynamic/Component/Example/Pods/libWeChatSDK/libWeChatSDK.a)

[!] Automatically assigning platform ios with version 8.3 on target LJA_Example because no platform was specified. Please specify a platform for this target in your Podfile. See 'https://guides.cocoapods.org/syntax/podfile.html#platform'.
+ Example git:(master) x
```

installError



这就尴尬了，于是一阵疯狂的搜索 google stackoverflow 等，然而并没有什么卵用，而且上面催得急，根本没时间处理这些 小问题 业务重构是最主要的，以至于我们的业务组件没有做到独立仓库拆分。
直到最近终于找到了解决办法:(主要是自己的功力不够深厚

理论功底

动态库和静态库

介绍

首先静态库和动态库都是以二进制提供代码复用的代码库

- 静态库 常见的是 .a
- 动态库常见的是 .dll(windows) .dylib(mac) so(linux)
- framework(in Apple): Framework是Cocoa/Cocoa Touch程序中使用的一种资源打包方式，可以将代码文件、头文件、资源文件、说明文档等集中在一起，方便开发者使用。
也就是说我们的framework其实是资源打包的方式，和静态库动态库的本质是没有关系的

静态库和动态库的区别

静态库: 链接时会被完整的复制到可执行文件中，所以如果两个程序都用了某个静态库，那么每个二进制可执行文件里面其实都含有这份静态库的代码

动态库: 链接时不复制，在程序启动后用dyld加载，然后再决议符号，所以理论上动态库只用存在一份，好多个程序都可以动态链接到这个动态库上面，达到了节省内存(不是磁盘是内存中只有一份动态库)，还有另外一个好处，由于动态库并不绑定到可执行程序上，所以我们想升级这个动态库就很容易，windows 和linux上面一般插件和模块机制都是这样实现的。

But我们的苹果爸爸在iOS平台上规定不允许存在动态库，并且所有的IPA都需要经过苹果爸爸的私钥加密后能用，基本你用了动态库也会因为签名不对无法加载，(越狱和非APP store除外)。于是就把开发者自己开发动态库掐死在幻想中。

直到有一天，苹果爸爸的iOS升级到了8，iOS出现了 APP Extension，swift 编程语言也诞生了，由于iOS 主APP需要和Extension共享代码，Swift语言的机制也只能有动态库，于是苹果爸爸尴尬了，不过这难不倒我们的苹果爸爸，毕竟我是爸爸，规则是我来定，我想怎样就怎样，于是提出了一个概念 Embedded Framework，这种动态库允许 APP 和 APP Extension 共享代码，但是这份动态库的生命被限定在一个APP进程内。简单点可以理解为 被阉割的动态库。

举个例子，iOS项目中使用Embeded Framework

如果你把某个自己开发的动态库(系统的不算，毕竟苹果是爸爸)放在了 Linked Frameworks and Libraries 里面，程序一启动就会报 Reason: Image Not Found，你只能把它放在 Embedded Binaries 里面才能正常使用，
看图:



静态库和动态库如何构建和加载

简单点，说话的方式简单点~~

上面的介绍貌似有点抽象啊 套用在美团技术分享大会上的话就是：

静态库：一堆目标文件(.o/.obj)的打包体(并非二进制文件)

动态库：一个没有main函数的可执行文件

这里我们来复习下C语言的基本功，编译和链接

编译：将我们的源代码文件编译为目标文件

链接：将我们的各种目标文件加上一些第三方库，和系统库链接为可执行文件。

由于某个目标文件的符号(可以理解为变量，函数等)可能来自其他目标文件，其实链接这一步最主要的操作就是 决议符号的地址。

- 若符号来自静态库(本质就是.o的集合包)或 .o，将其纳入链接产物，并确定符号地址
- 若符号来自动态库，打个标记，等启动的时候再说---交给dyld去加载和链接符号

于是链接加装载就有了不同的情况

1. Load 装载：将库文件载入内存

- Static Loading：启动时
- Dynamic Loading：启动后（使用时）

2. Link 链接：决议符号地址

- Static Linking：构建（链接）时
- Dynamic Linking：运行时（启动时或使用时）

然后组合起来就是 $2 \times 2 = 4$ 了

1. Static Loading + Static Linking
2. Static Loading + Dynamic Linking
3. Dynamic Loading + Dynamic Linking
4. ~~Dynamic Loading + Static Linking~~

第一种是纯静态库相关了

第二种就是静态加载(启动时)，动态链接，链接时，动态库参与链接，但是这时候只是给符号打了标记告诉我这个符号来自与动态库，程序启动时，iOS或者Mac OS操作系统的dyld自动 load+link。

既然全部都是自动的。那么符号的调用方完全不知道你到底是源码还是静态库，动态库。

第三种收到调用dlopen + performSelector 通常iOS的APP不适用这里不讨论

第四种，没见过，个人也不是特别懂

有需求请参看文后的 程序员的自我修养 一书

静态库和动态库依赖关系

既然有2种库，那么依赖关系又是 2×2 喽

1. libA.a dependency libB.a



2. UIKit.dylib dependency Foundation.dylib
3. libA.a dependency Foundation.dylib
4. MyXX.dylib dependency libA.a

第一种 静态库互相依赖，这种情况非常常见，制作静态库的时候只需要有被依赖的静态库头文件在就能编译出来。但是这就意味者你要收到告诉使用者你的依赖关系

幸运的是 CocoaPod 就是这样做的

第二种动态库依赖动态库，两个动态库是相互隔离的具有 隔离性，但是制作的静态库的时候需要被依赖动态库参与链接，但是具体的符号决议交给 dyld 来做。

第三种，静态库依赖动态库，也很常见，静态库制作的时候也需要动态库参与链接，但是符号的决议交给dyld来做。

第四种，动态库依赖静态库，这种情况就有点特殊了。首先我们设想动态库编译的时候需要静态库参与编译，但是静态库交由dyld来做符号决议，but 这和我们前面说的就矛盾了啊。静态库本质是一堆.o的打包体，首先并不是二进制可执行文件，再者你无法保证主程序把静态库参与链接共同生成二进制可执行文件。这就尴尬了。

怎么办？

目前的编译器的解决办法是，首先我无法保证主程序是否包含静态库，再者静态库也无法被 dyld 加载，那么我直接把你静态库的.o偷过来，共同组成一个新的二进制。也被称做 吸附性

那么我有两份动态库都依赖同样的静态库，这就尴尬了，每个动态库为了保证自己的正确性会把静态库吸附进来。然后两个库包含了同样的静态库，于是问题就出现了。看到这里想必前面出现的错误你已经能猜出来了把_

后面再详细解释

先来个总结

可执行文件（主程序或者动态库）在构建的链接阶段

- 遇到静态库，吸附进来
- 遇到动态库，打标记，彼此保持独立

Xcode 项目结构

target-对于一个产物(app,.a ,.framework)

project-一个项目包含多个target

workspace:一个包含多个target

schema: 指定了一个产物是按照何种的依赖关系，编译-链接到最终的一个产物

iOS依赖管理事实上的标准

这么多年，Apple的博客和文档也就告诉了我们什么是静态库 什么是动态库，如何制作等。但是并没有给我们提供一系列的依赖管理工具。所以CocoaPods成了事实上的标准。

通常CocoaPods管理的工程结构如下：

```
• CocoaPods
  + App.workspace
    + App.project
      • Pods.project
        • pod target => .a
```

那么当我们按下CMD+B的时候，整个项目按照先编译被依赖Pod，然后依赖其他Pod的Pod也被构建出来，最终所有的组件被编译为一个 lib-Pods-XXXAPP.a 被添加进项目进去。资源通过CocoaPods提供的脚本也一并被复制进去。想了解CocoaPods做了什么的读者可以参看后面的链接



解决问题

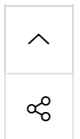
这么多理论功底的建立，相信我们已经能分析出来之前 `pod install` 的原因了。就是用了 `use_frameworks` 那么我们的所有Pod都会以动态库(Embedded Framework)的形式去构建，于是那些非开源的库(如 百度地图，微信分享)如果被多个Pod依赖(组件化开发中太常见了)于是被吸附到动态库里面，所以CocoaPod直接就不让我们install成功。因为你现在的依赖管理就是错误的。

在听取美团叶犊老师分享的时候 他们的出发点是因为要绕过苹果爸爸在iOS9以下对 `__text`段60M的限制使用了动态库方案，我们是因为某些swift库必须要用到(历史遗留原因)动态库。美团的做法是摘除依赖关系，自定义CocoaPods(开源的本来就是用着不爽我就改)。但是我是个小菜鸡啊。我也不会ruby(以后会学的)，但是叶犊老师给我提了别的idea。前面我们知道 动态库和静态库是 隔离性，动态库依赖静态库具有 吸附性，那么我们可以自定义一个动态库把百度地图这种静态库吸附进来。对外整体呈现的是动态库特性。其他的组件依赖我们自定义的动态库，由于 隔离性 的存在，不会出现问题。

制作动态库

1 创建动态库项目这里以wx举例

2 按照微信的官方文档。添加依赖库(我是因为pod install巨慢 所以我直接拽进来了)



3 将wx的PublicHeader暴露出来，注意由于我并没有使用到wx相关API所以链接器帮我们链接动态库 的时候可能并不会把wx静态库吸附进来。我们手动在build Setting的other link flags加上 `-all_load` 标记

4.在Schema里面跳转编译配置为Release，并且选择所有的CPU架构



5 然后选择模拟器或者Generic iOS Device运行编译就会生成对应版本的Framework了。

6.但是为了保证开发者使用的是真机模拟器都能正常使用，我们需要合并不同架构
这里在 Build Phases 里添加以下脚本，真机和模拟器都Build一遍之后就会在工程目录下生成Products文件夹，

```
if [ "${ACTION}" = "build" ]
then
INSTALL_DIR=${SRCROOT}/Products/${PROJECT_NAME}.framework
DEVICE_DIR=${BUILD_ROOT}/${CONFIGURATION}-iphoneos/${PROJECT_NAME}.framework
SIMULATOR_DIR=${BUILD_ROOT}/${CONFIGURATION}-iphonesimulator/${PROJECT_NAME}.framework
if [ -d "${INSTALL_DIR}" ]
then
rm -rf "${INSTALL_DIR}"
fi
mkdir -p "${INSTALL_DIR}"
cp -R "${DEVICE_DIR}/" "${INSTALL_DIR}/"
#ditto "${DEVICE_DIR}/Headers" "${INSTALL_DIR}/Headers"
lipo -create "${DEVICE_DIR}/${PROJECT_NAME}" "${SIMULATOR_DIR}/${PROJECT_NAME}" -output "${INSTALL_DIR}/${PROJECT_NAME}"
open "${INSTALL_DIR}"
fi
```



于是我们有了我们自己的私有动态库LJWXSDK，那么我们来验证我们之前的问题
首先指定一个LJWXSDK.podspec这里我直接传到了我的Github
(<https://link.jianshu.com?t=https://github.com/ValiantCat/LJWXSDK>)上面

```
#
# Be sure to run `pod lib lint LJPod.podspec` to ensure this is a
# valid spec before submitting.
#
# Any lines starting with a # are optional, but their use is encouraged
# To learn more about a Podspec see http://guides.cocoapods.org/syntax/podspec.html
#

Pod::Spec.new do |s|
  s.name             = 'LJWXSDK'
  s.version          = '0.1.0'
  s.summary          = 'A short description of LJWXSDK.'

  s.description      = <<--DESC
  TODO: Add long description of the pod here.
  DESC

  s.homepage         = 'https://github.com/ValiantCat/LJWXSDK'

  s.license          = { :type => 'MIT', :file => 'LICENSE' }
  s.author           = { 'ValiantCat' => '519224747@qq.com' }
  s.source = { :http => 'http://onk2m6gtu.bkt.clouddn.com/LJWXSDK.framework.zip' }

  s.ios.deployment_target = '8.0'
  s.default_subspec = 'zip'

  s.subspec 'zip' do |zip|

    puts '-----'
    puts 'Notice:LJWXSDK is zip now'
    puts '-----'

    zip.ios.vendored_frameworks = '*.framework'
  end
end

end
```

注意上面我是把二进制压缩丢进了七牛的oss文件存储。毕竟免费还快。

然后通过pod lib create创建了一个pod用来验证之前我们的传递性依赖问题，
文件夹结构如下

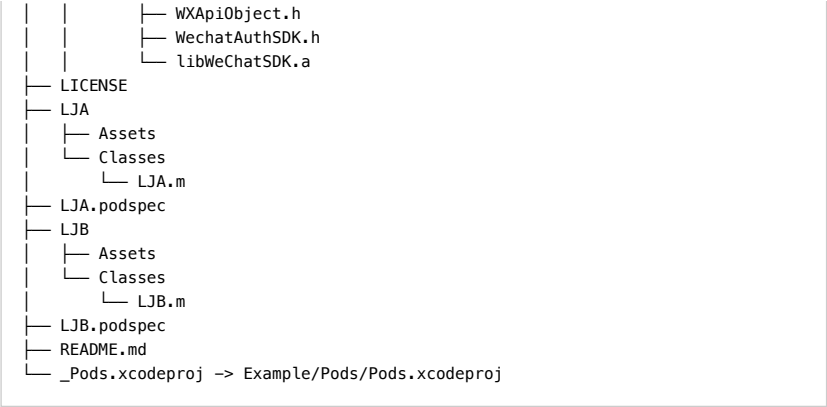



```

.
├── Example
│   ├── LJA
│   │   ├── Base.lproj
│   │   │   ├── LaunchScreen.storyboard
│   │   │   └── Main.storyboard
│   │   ├── Images.xcassets
│   │   │   ├── AppIcon.appiconset
│   │   │   └── Contents.json
│   │   ├── LJA-Info.plist
│   │   ├── LJA-Prefix.pch
│   │   ├── LJAppDelegate.h
│   │   ├── LJAppDelegate.m
│   │   ├── LJViewController.h
│   │   ├── LJViewController.m
│   │   ├── en.lproj
│   │   │   └── InfoPlist.strings
│   │   └── main.m
│   ├── LJA.xcodeproj
│   ├── LJA.xcworkspace
│   ├── Podfile
│   ├── Podfile.lock
│   └── Pods
│       ├── Headers
│       ├── LJWXSDK
│       │   ├── LJWXSDK.framework
│       │   │   ├── Headers
│       │   │   │   ├── LJWXSDK.h
│       │   │   │   ├── WXApi.h
│       │   │   │   ├── WXApiObject.h
│       │   │   │   └── WechatAuthSDK.h
│       │   │   ├── Info.plist
│       │   │   ├── LJWXSDK
│       │   │   ├── Modules
│       │   │   │   └── module.modulemap
│       │   │   ├── _CodeSignature
│       │   │   │   └── CodeResources
│       │   │   └── read_me.txt
│       │   ├── Local\ Podspecs
│       │   │   ├── LJA.podspec.json
│       │   │   ├── LJB.podspec.json
│       │   │   └── LJWXSDK.podspec.json
│       │   ├── Manifest.lock
│       │   ├── Pods.xcodeproj
│       │   │   ├── project.pbxproj
│       │   │   └── project.xcworkspace
│       │   └── Target\ Support\ Files
│       │       ├── LJA
│       │       │   ├── Info.plist
│       │       │   ├── LJA-dummy.m
│       │       │   ├── LJA-prefix.pch
│       │       │   ├── LJA-umbrella.h
│       │       │   ├── LJA.modulemap
│       │       │   └── LJA.xcconfig
│       │       ├── LJB
│       │       │   ├── Info.plist
│       │       │   ├── LJB-dummy.m
│       │       │   ├── LJB-prefix.pch
│       │       │   ├── LJB-umbrella.h
│       │       │   ├── LJB.modulemap
│       │       │   └── LJB.xcconfig
│       │       ├── Pods-LJA_Example
│       │       │   ├── Info.plist
│       │       │   ├── Pods-LJA_Example-acknowledgements.markdown
│       │       │   ├── Pods-LJA_Example-acknowledgements.plist
│       │       │   ├── Pods-LJA_Example-dummy.m
│       │       │   ├── Pods-LJA_Example-frameworks.sh
│       │       │   ├── Pods-LJA_Example-resources.sh
│       │       │   ├── Pods-LJA_Example-umbrella.h
│       │       │   ├── Pods-LJA_Example.debug.xcconfig
│       │       │   ├── Pods-LJA_Example.modulemap
│       │       │   └── Pods-LJA_Example.release.xcconfig
│       │       ├── Pods-LJA_Tests
│       │       │   ├── Info.plist
│       │       │   ├── Pods-LJA_Tests-acknowledgements.markdown
│       │       │   ├── Pods-LJA_Tests-acknowledgements.plist
│       │       │   ├── Pods-LJA_Tests-dummy.m
│       │       │   ├── Pods-LJA_Tests-frameworks.sh
│       │       │   ├── Pods-LJA_Tests-resources.sh
│       │       │   ├── Pods-LJA_Tests-umbrella.h
│       │       │   ├── Pods-LJA_Tests.debug.xcconfig
│       │       │   ├── Pods-LJA_Tests.modulemap
│       │       │   └── Pods-LJA_Tests.release.xcconfig
│       └── libWeChatSDK
│           ├── README.md
│           └── WXApi.h

```





测试工程我也丢在7牛上面。下载 (<https://link.jianshu.com?t=http://onk2m6gtu.bkt.clouddn.com/Component.zip>)测试即可编译运行。完美。我们又可以愉快的和swift第三方库配合使用。很多人可能会问 诸如百度地图 微信这种sdk为什么官方不支持动态库版(所说的都是 embeded Framework)，猜测是为了兼容更低iOS7版本吧。很多人会觉得麻烦的要死。首先每个公司多多少少都有历史包袱，麻烦也要做，再者这是一次对基本功的补充，即便你们没有用到，但是为了学习，这篇教程所做的也值得你尝试一次。

剖析下动态库Framework吧

上述解决了我们一开始遇到的问题。but既然动态库和静态库压根就不一回事，所以里面还是有很多细节值得我们去了解的。

回过头来看Embeded Framework

首先我们之前记得如果一个动态库加在 `LinkedFrameworksand Libraies` 程序启动就会报 `ImageNotFound`，如果放在 `EmbededBinaries` 里面就可以。这是为什么呢。我们拿 `MacoView`来看下两种情况下可执行文件的细节



其中@rpth这个路径表示的位置可以查看Xcode 中的链接路径问题

(<https://www.jianshu.com/p/cd614e080078>)

这样我们就知道了其实加在 `EmbeddedBinaries` 里面的东西其实会被复制一份到`xx.app`里面，所以这个名字起得还是不错的直译就是 嵌入的框架

Why Swift does not Support Staic Libraies

造成这个的主要原因是Swift的运行时库(不等同于OC的runtime概念)，由于Swift的ABI不稳定，静态库会导致最终的目标程序中包含重复的运行库，相关可以看下最后的参考文章SwiftInFlux#static-libraries (<https://link.jianshu.com?t=https://github.com/ksm/SwiftInFlux#static-libraries>)。等到我们的SwiftABI稳定之后，我们的静态库支持可能就又会出现了。当然也可能不出，Swift伴随诞生的SPM(Swift, Package Manager)，可能有更好的 官方的 包依赖管理工具。让我们期待吧。

CocoaPods 使用Use_framework!

既然加了Swift的第三方库之后就需要在 `Podfile` 里面加上 `use_framework!` 那么CocoaPods就会帮我们生成动态库，但是奇怪的是，我们并没有在主工程的 `embedded binaries` 看到这个动态库，这又是什么鬼。其实是CocoaPods使用脚本帮我们加进去了。脚本位置在主工程的 `build Phase` 下的 `Emded Pods frameworks`

```
"${SRCROOT}/Pods/Target Support Files/Pods-LJA_Example/Pods-LJA_Example-frameworks.s
```

动态库Framework的文件结构

```
.
├── Headers
│   ├── LJWXSDK.h
│   ├── WXApi.h
│   ├── WXApiObject.h
│   └── WechatAuthSDK.h
├── Info.plist
├── LJWXSDK
├── Modules
│   └── module.modulemap
├── _CodeSignature
│   └── CodeResources
```

1. Headers 一般是头文件。非private里面的头文件都会在里面
2. info.plist 配置信息，不深究
3. Modules 这个文件夹里有个module.modulemap文件，后面在讲解
4. 二进制文件，这就是上面提到的 不带 main的二进制文件了，.o的打包体
5. _codeSignature 签名文件 (苹果爸爸的约束)
6. more 资源文件。这里暂时没用到，所以没有，但是这个也是个大坑

更愉快的导入文件



@class, @protocol 不说了就是声明一个类，并不导入。

#import <>, #import"" 是加强版的 #include<>, #include"" 防止重复导入的。

#import<> : 通过 build setting里面中的 header Search Path里面去找

#import"" : 第一步先搜索user Header search Path 再搜索 header search Path 。所以对我们的framework来说， CocoaPod 帮我们加到了 Header search Path 目前2种导入方式都是可以支持的。

上面的导入方式都带了 某个framework的路径 <XX/xx.h> "xx/xx.h"，我们在开发自己主工程的时候会发现我们导入主工程其他类是不需要导入前缀的。这又是怎么回事。

看下面的配置

目前的配置是non-recursive。如果把non去掉意思就是我可以递归的去查找某些framework下面的头文件了。但是Xcode的效率肯定就会有影响。还是不建议修改的好。

大家都知道iOS7之后多了@import，这又是什么鬼。

简单理解这个方式叫做Module导入，好处就是使用了@import之后不需要在project setting手动添加 framework，系统会自动加载，而且效率更高。

最主要的是swift也只能这样用。

导入的时候系统会查找如果有模块同名的文件就会导入这个文件。如果没有CocoaPods帮我们生成一个 module-umbrella.h 文件，然后就是导入的这个文件。

回过头来看我们的framework的结构 里面有个 Modules 文件夹，里面有个文件

module.modulemap

```
framework module LJWXSDK {
    umbrella header "LJWXSDK.h"

    export *
    module * { export * }
}
```

我们可以看到其实被暴露的header就是这个文件，之前我在按照 #import "/" 的时候有个警告

而且按照@import导入的东西发现没有导入可用的头文件就是因为并没有在 umbrella header的头文件中加入其他头文件。

加入之后我们就可以完美的使用 @import ， 并且 #import"/" 也不会报warning

更多关于 umbrella Header 参看文后参考

资源问题

首先我们来看常见的资源文件:主要分为图片和其他类资源那么加载图片和加载其他资源都是怎么做的？

1: [UIImage imageNamed:]

2: [NSBundle bundleForClass:[XXX class]]



其实方式1去本质就是去 mainBundle 去拿资源，方式2从 xxx 所在的框架里面去拿。
前面也说道framework只是资源的打包方式，本质上是有两种的。
我们这个framework如果本质是静态库，那么无需改变使用方式，资源最终都会打包到 Main Bundle 里面
如果我们这个framework本质是动态库，那么我们的资源就发生了变化，资源就会被存放在 framework里面。所以我们需要使 [NSBundle bundleForClass:[XXX class]]。需要注意的是很多人为了简单，下意识的使用 self class 传递，但是有可能这个 self实例 不在资源所属的framework。所以会出现资源加载 失败。一定要谨慎使用。

参考

程序员的自我修养，链接，装载 和库 (<https://link.jianshu.com?t=https://item.jd.com/10362683979.html>)
iOS里的动态库和静态库 (<https://link.jianshu.com?t=https://www.zybuluo.com/qidiandasheng/note/603907>)
Systems Programming: What is the exact difference between Dynamic loading and dynamic linking? (<https://link.jianshu.com?t=https://www.quora.com/Systems-Programming-What-is-the-exact-difference-between-Dynamic-loading-and-dynamic-linking>)
CocoaPods 都做了什么？ (<https://link.jianshu.com?t=http://draveness.me/cocoapods.html>)
Dynamic Linking of Imported Functions in Mach-O (<https://link.jianshu.com?t=https://www.codeproject.com/articles/187181/dynamic-linking-of-imported-functions-in-mach-o>)
OS里的导入头文件 (<https://link.jianshu.com?t=https://www.zybuluo.com/qidiandasheng/note/602118>)
iOS - Umbrella Header在framework中的应用 (<https://link.jianshu.com?t=http://blog.startry.com/2015/08/25/Renaming-umbrella-header-for-iOS-framework/>)
SwiftInFlux#static-libraries (<https://link.jianshu.com?t=https://github.com/ksm/SwiftInFlux#static-libraries>)
iOS里的导入头文件 (<https://link.jianshu.com?t=https://www.zybuluo.com/qidiandasheng/note/602118>)
iOS - Umbrella Header在framework中的应用 (<https://link.jianshu.com?t=http://blog.startry.com/2015/08/25/Renaming-umbrella-header-for-iOS-framework/>)
@import vs #import - iOS 7 (<https://link.jianshu.com?t=https://stackoverflow.com/questions/18947516/import-vs-import-ios-7>)

SwiftBlog (/nb/1198331)

举报文章 © 著作权归作者所有



南栀倾寒 (/u/cc1e4faec5f7) ♂

写了 45445 字，被 2681 人关注，获得了 1024 个喜欢
(/u/cc1e4faec5f7)

+ 关注

iOS工程师 有兴趣加入群173499350 此简书只做同步使用，原文在<https://www.valiantcat.cn/>

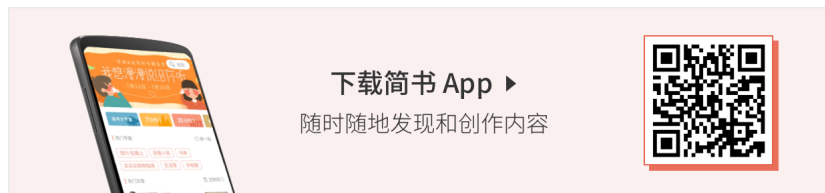
喜欢 (/sign_in?utm_source=desktop&utm_medium=not-signed-in-like-button) | 63



更多分享

(<http://cwb.assets.jianshu.io/notes/images/1114944>)





(/apps/download?utm_source=nbc)

被以下专题收入，发现更多相似内容



寒哥管理的技术专题 (/c/5be41e88940c?

utm_source=desktop&utm_medium=notes-included-collection)



SwiftBlog (/c/b228190a31cc?utm_source=desktop&utm_medium=notes-

included-collection)



iOS Swi... (/c/b6cc50b537d5?utm_source=desktop&utm_medium=notes-

included-collection)



iOS 大神之路 (/c/9bafec2216bd?

utm_source=desktop&utm_medium=notes-included-collection)



iOS 进阶之路 (/c/e25b25208315?

utm_source=desktop&utm_medium=notes-included-collection)



待读清单 (/c/b0b744c450e5?utm_source=desktop&utm_medium=notes-

included-collection)

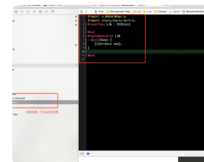


移动前沿 (/c/5aac963ca52d?utm_source=desktop&utm_medium=notes-

included-collection)

展开更多

(/p/d1b994a0c9c0?



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

组件化-动态库实战续 (/p/d1b994a0c9c0?utm_campaign=maleskine&ut...

原文地址,此简书只做备份,强烈推荐原文,毕竟格式比简书好看,还清晰 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...



南栀倾寒 (/u/cc1e4faec5f7?

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

iOS 第三方库、插件、知名博客总结 (/p/fa0b6f594c36?utm_campaign=m...

用到的组件1、通过CocoaPods安装项目名称项目信息 AFNetworking网络请求组件 FMDB本地数据库组件 SDWebImage多个缩略图缓存组件 UICKeyChainStore存放用户账号密码组件 Reachability监测网络状态...



大灰狼的小绵羊哥哥 (/u/524ab6e6e423?

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

静态库，动态库与 Framework (/p/ee2affaa3bac?utm_campaign=malesk...

静态库与动态库的区别 首先来看什么是库,库(Library)说白了就是一段编译好的二进制代码,加上头文件就可以供别人使用。什么时候我们会用到库呢?一种情况是某些代码需要给别人使用,但是我们不希望别人...




吃瓜群众呀 (/u/78e37148ba47?

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

iOS开发 非常全的三方库、插件、大牛博客等等 (/p/63f47b492b5a?utm_c...



UI 下拉刷新 EGOTableViewPullRefresh- 最早的下拉刷新控件。 SVPullToRefresh- 下拉刷新控件。
MJRefresh- 仅需一行代码就可以为UITableView或者CollectionView加上下拉刷新或者上拉刷新功能。可以...


 yunxiu (/u/0e04231643ce?
utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/f9d656ed3dad?




utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)
三方库转载 (/p/f9d656ed3dad?utm_campaign=maleskine&utm_content=...

下拉刷新 EGOTableViewPullRefresh - 最早的下拉刷新控件。 SVPullToRefresh - 下拉刷新控件。 MJRefresh
- 仅需一行代码就可以为UITableView或者CollectionView加上下拉刷新或者上拉刷新功能。可以自定义上下...

 Carden (/u/1f4b38fd4718?
utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


舒缓之想 (/p/73fa26718deb?utm_campaign=maleskine&utm_content=n...

事件：今天早上和老公一起带女儿去吃早餐然后带女儿去看她想看的电影。车上聊到我们吵架我出去住的这
些日子，我说你怎么可以把我赶出去，怎么可以说那些伤人的话，让我觉得结婚几年只是在这个房子里借...

 怪怪妮儿 (/u/d26fccde0a7f?
utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

烦 (/p/3e1fad39c5aa?utm_campaign=maleskine&utm_content=note&u...

最近心里感觉好累，好累。压力太大，。感觉力不从心，好想离开，却又不得不留在这里。面对这些冷淡的
人们。没有一点感恩的心，我真的好不喜欢，不喜欢这里的一切，不喜欢这里的人，没有上进心，不尊重...


 芒果baby (/u/995a3ad2555a?
utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/120ee4a6be9e?



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)
Mac 虚拟机 CentOS7 走起 (一) (/p/120ee4a6be9e?utm_campaign=ma...


1，安装前准备： 下载 Oracle VM VirtualBox 虚拟机下载 CentOS7 系统 2，开始安装： 2.1，打开
VirtualBox 新建虚拟电脑。 2.2，填写名称-类型-版本，CentOS 属于 Red Hat 的发行版。2.3，选择系统运...

 JohnnyB0Y (/u/8939e3430d49?
utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/73bc617f5b4e?




utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)
珍爱生命 远离背硬包美女 (/p/73bc617f5b4e?utm_campaign=maleskine&...

 一书onebook (/u/40bf9b5e3bef?
utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

JSON 基础知识总结 (/p/c30adfdb04ae?utm_campaign=maleskine&utm...

摘要：JSON，说白了就是JavaScript用来处理数据的一种格式，这种格式非常简单易用。JSON大部分都是
用来处理JavaScript和web服务器端之间的数据交换，把后台web服务器的数据传递到前台，然后使用...



 乱蓬头199302 (/u/e98fe2f529b0?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

