


iOS App 瘦身 - 以 Swift App Yep 为例



Damonwong (/u/3a9e1894acc2)

✓ 已关注

2016.10.14 22:29*

字数 4427

阅读 3266

评论 14

喜欢 69

(/u/3a9e1894acc2)

文章最后有我的 12 条小总结。

写在前面

- 最近公司需求不多，正好研究一下 App 瘦身的办法，写了点小总结。
- 如果你不知道下面几个问题，不妨可以看看文章。
 - 使用 .xcassets 有什么好处？
 - @1x 、@2x 和 @3x 会一起内置到安装包中吗？
 - PDF 和 @1x 、@2x 和 @3x 有什么区别？
 - 如果我有一个 10 x 10 的控件和一个 50 x 50 的控件，美工需要制作几张 PDF？
 - Iconfont 是什么？PDF 和 Iconfont 有什么区别？
 - 启动图的正确打开方式？
 - 使用 Swift 或者 混编会增大多少的包体积？
 - Install Smallest or Coding Fastest ？

分析

- 在瘦身之前，首先需要分析一下，我们可以从哪几个方面入手。(以 Yep 为例)

Yep (<https://github.com/CatchChat/Yep>)是一款很优秀的 Swift 开源软件。

目录划分

- App 的瘦身主要是针对于安装包，而在 iOS 中安装包就是一个以 .ipa 结尾的压缩包。我们可以通过 iTunes 下载获取这个 .ipa 来分析。

About.storyboard

AboutCell.nib

AddFriendMoreCell.nib

AddFriends.storyboard

AddFriendsSearchCell.nib

AddSkillsReusableView.nib

AlbumListCell.nib

AppIcon29x29.png

AppIcon29x29@2x.png

AppIcon29x29@2x~ipad.png

AppIcon29x29@3x.png

AppIcon29x29~ipad.png

AppIcon40x40@2x.png

AppIcon40x40@3x.png

AppIcon40x40~ipad.png

AppIcon50x50@2x~ipad.png

AppIcon50x50~ipad.png

AppIcon57x57.png

AppIcon60x60@2x.png

AppIcon60x60@3x.png

AppIcon72x72~ipad.png

AppIcon76x76@2x~ipad.png

AppIcon76x76~ipad.png

SearchChatContactUserCell.nib

SearchedDiscoveredUserCell.nib

SearchedFeedCell.nib

SearchedMessageCell.nib

SearchedUserCell.nib

SearchedUsers.storyboard

SearchFeeds.storyboard

SearchMoreResultsCell.nib

SearchSectionTitleCell.nib

Settings.storyboard

SettingsUserCell.nib

SkillAddCell.nib

SkillAnnotationHeader.nib

SkillCategoryCell.nib

SkillCell.nib

SkillHome.storyboard

SkillRankCell.nib

SkillSelectionCell.nib

SocialWorks.storyboard

TitleCell.nib

UserStateCell.nib

VerifyChangedMobile.storyboard

Yep

zh-Hans.lproj

Assets.car

Base.lproj

BlackList.storyboard

bub3.caf

CameraCell.nib

ChangeMobile.storyboard

ChatLeftSocialWorkCell.nib

ChatSectionDateCell.nib

Contacts.storyboard

ContactsCell.nib

Conversation.storyboard

Conversations.storyboard

CreatorsOfBlockedFeeds.storyboard

DeletedFeedConversationCell.nib

Discover.storyboard

DiscoverCardUserCell.nib

DiscoverNormalUserCell.nib

DiscoverSkillCell.nib

DoNotDisturbPeriod.storyboard

DoNotDisturbPeriodCell.nib

DoNotDisturbSwitchCell.nib

DribbbleShotCell.nib

EditNicknameAndBadge.storyboard

EditProfile.storyboard

EditProfileColoredTitleCell.nib

NSBundleImageBundle

PhotoCell.nib

PickLocation.storyboard

PickLocationCell.nib

PickPhotos.storyboard

PkgInfo

Plugins

PodstHelpYep.storyboard

Profile.storyboard

ProfileFeedCell.nib

ProfileFooterCell.nib

ProfileHeaderCell.nib

ProfileSectionHeaderReusableView.nib

ProfileSeparationLineCell.nib

ProfileSocialAccountBlogCell.nib

ProfileSocialAccountGitHubCell.nib

ProfileSocialAccountImagesCell.nib

QuickPickPhotosCell.nib

SG_Info

SearchContacts.storyboard

SearchConversations.storyboard

Yep IPA

+

🔖

🔗

- 稍微整理一下，大致可以分为以下几类。

<http://www.jianshu.com/p/df52554be812>

1/10

- 资源层面:

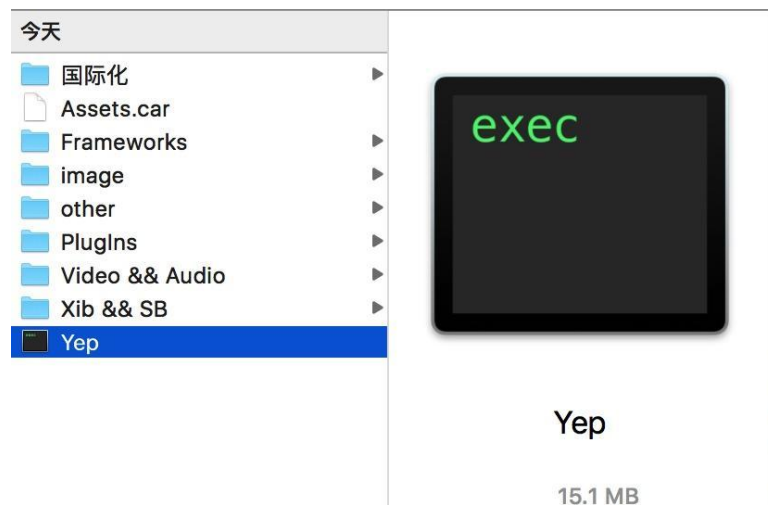
- **Assets.car**: 项目中所有 .xcassets 的压缩包
- **image**: 图片资源文件
- **Video && Audio**: 音频 或者 视频。

- 代码层面:

- **国际化**: 国际化适配的 String ==> 89K
- **Xib && Storyboard**: Xib 和 Storyboard 编译后的文件。
- **Yep**: 项目可执行文件。
- **Frameworks**: Embedded Frameworks, 项目中使用的动态库

- 其他:

- **other**: 配置文件
- **Plugins**: YepShare, 一个共享的插件。



整理后的目录

- 虽然 Yep 不能代表所有的 App, 但是在对于 Yep 的 ipa 分析之后, 大致可以总结出, 对一个 App 的安装包瘦身, 可以从**资源层面**和**代码层面**两个层面入手。

资源层面

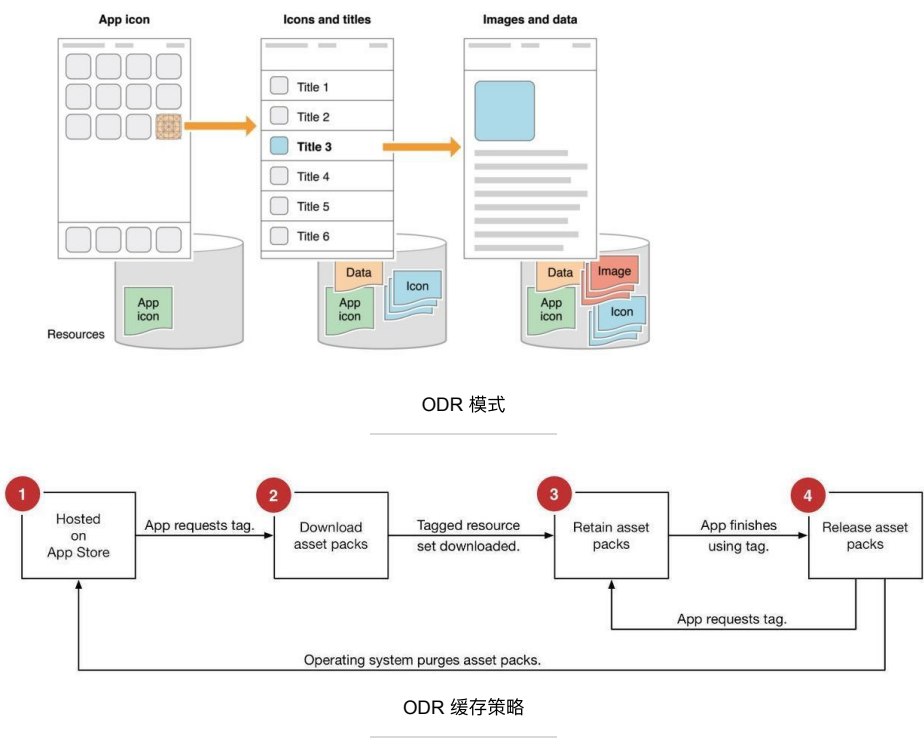
- 在讲资源层面之前, 希望我们能达到一个**共识**, 那就是所谓的**资源文件**指的是 **图片、视频、音频**。
- **Remote**: 将资源文件放在服务器上, 当用户下载完 App 后根据需要再下载。
- **Local**: 将资源文件集成到安装包中的。

Remote

- 对于 **Remote** 的方式, 如果做好策略(比如缓存), 那么理论上, 我们可以把 **非必须的资源文件** 都放到服务器上, 这样对资源压缩率达到了 **100%**。也就是说安装包 **没有任何非必须资源文件**。
 - **必须资源文件**: 例如应用图标、启动图的这种配置图片。



- 苹果的 On-Demand Resources
(https://developer.apple.com/library/content/documentation/FileManagement/Conceptual/On_Demand_Resources_Guide/Chapters/Introduction.html#//apple_ref/doc/uid/TP40015083-CH11-SW1)(翻译 (<http://benbeng.leanote.com/post/On-Demand-Resources-Guide>)) 也是通过这种按需加载资源的思路给我们提供了一种阶段性加载资源的途径，具体的不展开描述，你可以点前面的链接进行查看。但是虽然以关卡、tag这种方式来按需加载资源，但是苹果的服务器对于中国用户来说实在是慢的不行，所以暂时不建议采取这种方式。你们可以在自己服务器上实现这种策略方式来加载图片。



Local

- 当然全部将非必须资源文件放到服务器上明显是不现实的，对于一些必用资源文件，还是需要资源文件 集成到安装包中的。
 - 必用资源文件：安装了 App 肯定会用到。

Local 集成方式

- Create group 和 Create folder references
 - 这两种其实就是直接把资源文件 拖 进去，在 Xcode 打包之后，所有图片都在可执行文件的相同目录下面。这也是很多老的 App 或者目前部分 App 的使用方式。
- .xcassets
 - 这是苹果在 Xcode 5 出来之后，推荐我们使用的图片管理方式，提供了图片渲染、拉伸模式模式、机型适配等功能。在 Xcode 打包之后所有的.xcassets 文件都会放入一个Assets.car文件中。

Local 开发使用方式列举分析。

- 一般 App 的图片内置方式
 - 采用拖的方式,图片包含@1x、@2x 和 @3x。
 - 采用拖的方式,图片只包含 @2x 和 @3x。



- 采用拖的方式,图片只包含 @2x 或 @3x。
 - 采用.xcassets的方式,图片包含@1x、@2x 和 @3x。
 - 采用.xcassets的方式,图片只包含@2x 和 @3x。
 - 采用.xcassets的方式,图片只 包含@2x 或 @3x。
 - 采用.xcassets的方式,图片使用 PDF。
(可能还有其他方式, 希望你能告诉我。)
- 拖的方式
 - 首先@1x、@2x 和 @3x主要是为了适配不同 ppi 的机型而做了一种策略。@1x 主要是为了适配 iPhone 4 之前的 非 Retina 屏幕, @2x 主要是为了适配 非 plus 的 iPhone 设备, @3x 是为了适配一个点的 3 * 3 个像素的手机。
 - 因为 iPhone 4 之前的机型基本没有什么 App 会去适配, 所以一般来说都会删除, 所以就有了『采用拖的方式,图片只包含@2x 和 @3x』的方式。但是假如你只提供一张图片, 例如你只提供了一张 @3x 的图片, iOS 系统在 iPhone 7 上无法找到 @2x 的图片, 会去查找 @3x 或者 @1x 等, 再根据实际分辨率进行拉伸, 最后把像素铺到屏幕上。所以在能够接受查找和拉伸造成的性能消耗的前提下, 我们可以只用一张通用的图片, 所以就有了『采用拖的方式,图片包含@2x 或 @3x』的方式。
 - 以一个 14M 的图片资源(包含@1x、@2x、@3x)来说, 如果所有的图片去除掉 @1x 能减少 1M 左右, 去除掉@2x能去掉 4M 左右。因此采用『采用拖的方式,图片包含@2x 或 @3x』的方式虽然损失了一点性能, 单大概图片资源大概减少了 35%左右, 。
- 采用.xcassets的方式
 - 我们都知道了, 采用『采用拖的方式,图片包含@2x 或 @3x』的方式大概图片资源大概减少了35%左右, 但是稍微损失了一点性能。有什么方式可以减少掉这点性能消耗呢?
 - “很幸运”, 苹果在 iOS 9 终于意识到了这个问题, 然后提供了一个叫做 App Slicing(如下图所示)的东西。App Slicing大致就是App Store会根据不同的设备准备不同的安装包(App Variant), 每个安装包(App Variant)都只有相应尺寸的图片,比如 iPhone 6 去下载时, 只会下载到 @2x 的图片的安装包(App Variant)。但能实现这个功能的前提是图片需要放置在.xcassets去管理。
![[APP Slicing]](<http://7xlv6p.com1.z0.glb.clouddn.com/2016-10-14-APP>
(<http://7xlv6p.com1.z0.glb.clouddn.com/2016-10-14-APP>) Slicing.png)
 - 所以, 目前许多 App 采用『.xcassets的方式,图片只 包含@2x 或 @3x』其实是没意义的, 特别是在你不适配 iOS 8 的时候, 你这么做是强行降低了 App 的性能。当然你要觉得为了 8% 的非 iOS 9 用户 减少 App 安装包大小 而去降低另外 92% 的用户的 App 运行性能 没什么问题, 那么你可以采取上面这种方式。
 - 关于 PDF
 - 我最早是在这一篇博客 (<http://martiancraft.com/blog/2014/09/vector-images-xcode6/>)中看到的, 当然 Yep 也是这种方式。大致是删掉 @2x 和 @3x 的图片, 然后替换成 矢量图 PDF, 最后放入.xcassets中去。
 - 而 Xcode 在打包的过程中, 根据你的矢量PDF图的大小, 生成@1x、@2x和 @3x的图。例如你的PDF图是4545px, 那么Xcode会在编译时生成下面3个

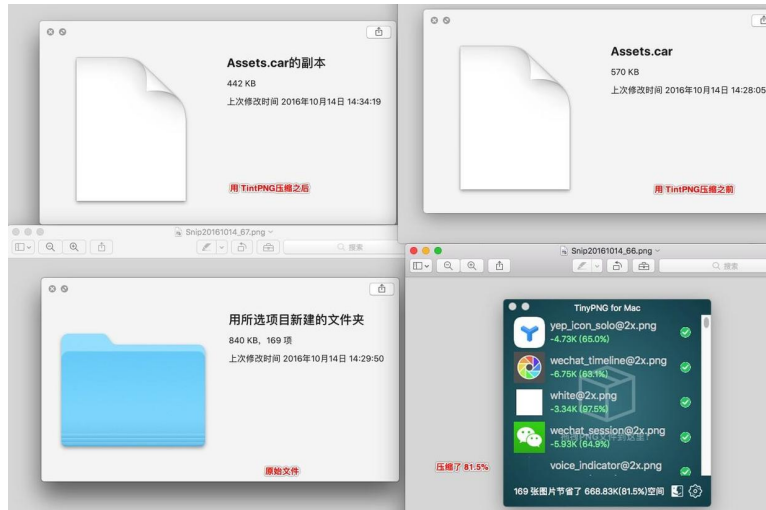


png: 4545px、9090px、135135px, 最后再放入 **Assets.car** 中。所以采用 **@1x**、**@2x** 和 **@3x** 和 **PDF** 两种方式本质上是一样的。

- 在这里有很多人会有一个误解, 例如在 App 中有一个 10 * 10 pt 和一个 50 * 50 pt 的 imageView 都用了一个相同的图标。很多人会以为做一个就够了, 因为 pdf 是矢量图。但是其实是需要一个 10 * 10 px 和 50 * 50 px 的两张 pdf 才可以, 因为只用一张的话, 另外一张用的其实就是 10 * 10px 的 PDF 的产物。

• 关于压缩问题。

- 我是用 tinypng (<https://tinypng.com>) 来压缩的, 应该是以最小的占用量达到了最适合的效果。但是其实 **.xcassets** 也会为你做一部分的压缩。如下图所示:



压缩过程

- **.xcassets** 的压缩应该还对图片进行了处理这也就是为什么 840KB 压缩了 81.5%, **Assets.car** 却没有减少那么多。
- 同时也有人在试验中发现, 用一些压缩工具似乎没有很实际效果, 这也有可能是因为 Xcode 在打包的时候做了一定的处理。

启动图

- 启动图在一个项目资源中占比其实蛮大的, 之前见过一个项目 6 张启动图大概有 5M 左右, 最大的是 2M。
 - iPad 2 and iPad mini (@1x): 768 x 1024
 - iPad and iPad mini (retina @2x): 1536 x 2048
 - iPhone 4s (retina @2x) 640 x 960
 - iPhone 5 (@2x): 640 x 1136
 - iPhone 6 (@2x): 750 x 1334
 - iPhone 6 Plus (@3x): 1242 x 2208
- 但是自从 **LaunchScreen.storyboard** 出来一后完全没必要做这么多张了。只需要将启动图设置为 **LaunchScreen.storyboard** 然后在 **LaunchScreen.storyboard** 上设置一张 imageView。最后再弄一张启动图的 pdf 就可以了。

iconfont

- 首先这个东西估计很多人不知道, 我也是在 @卓同学 () 的提醒下才知道原来 iOS 也是可以用 iconfont 的。最早这个东西是为 Web 设计的, 主要是因为网页的大小直接影



响了加载速度，所以在压缩上连小 icon 都不放过,当然还有一个最主要的目的就是减少请求次数，因为如果是图片的话，一个图片就是一次请求。

- 具体的效果可以看一下，使用IconFont减小iOS应用体积 (<http://johnwong.github.io/mobile/2015/04/03/using-icon-font-in-ios.html>)这篇文章。
- 虽然看上去效果不多，但对于一些比较追求精致的公司可以尝试一下这种方式。

期中，PDF 和 iconfont 两个都是矢量的概念，但是 iconfont 在整个 App 中不管多少种尺寸只需要一个 iconf，但是 PDF 可能需要多个。

HTML 5

- 一些 APP 的一些功能可以用 HTML5 + WebView 的方式来实现。而 HTML 5 这个可以通过下面几种方式一步步优化：
 - 让做前端的给一个最小的包内置到 App，去除无用代码、代码混淆压缩等。
 - 将所有图片 Remote 化。
 - 将所有页面 Remote 化。

其他

- 当然，还要注意资源文件重复的问题。而资源文件重复问题主要有几种：名字相同、名字不同内容相同/相似。
 - 对于名字相同的问题，你可以把原来的拖的方式改为.xcassets，他会自动管理相同名字的图片。然后把多余的去掉
 - 名字不同内容相同/相似:你可以使用Duplicate Photos工具 (<https://www.duplicatephotocleaner.com/>)
- 还有一个问题就是资源文件没有用，却占了空间，可以使用LSUnusedResources (<https://github.com/tinymind/LSUnusedResources/>)将代码中没用到的文件删除。当然可能存在误删，比如用数组加载的图片，这个工具无法识别。

代码层面

Install Smallest VS. Coding Fastest

语言选择

- 虽然说我个人更喜欢用 Swift 来写 App。但从 App 瘦身的角度，不推荐使用 Swift，不论纯 Swift 还是 混编。原因很简单。看一下下面的图：

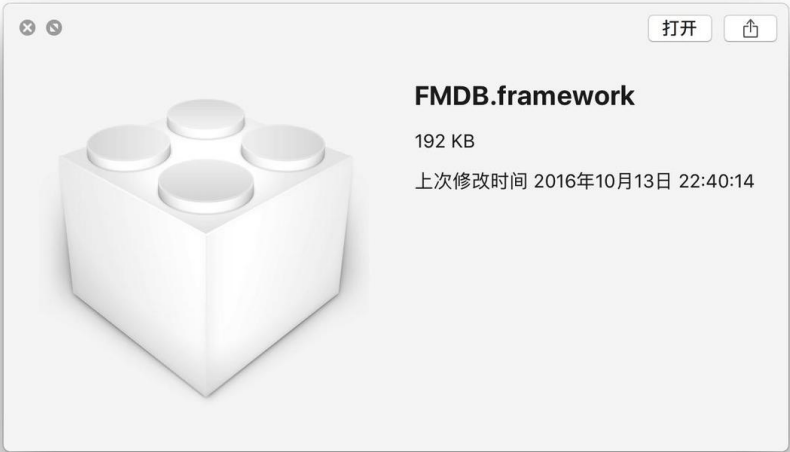


- 这是任何一个包含有 Swift 代码的 App 都有的一个为了支持 Swift 的动态库集合，在 10M 左右。如果你使用 Objective - C 完全不用这个东西。

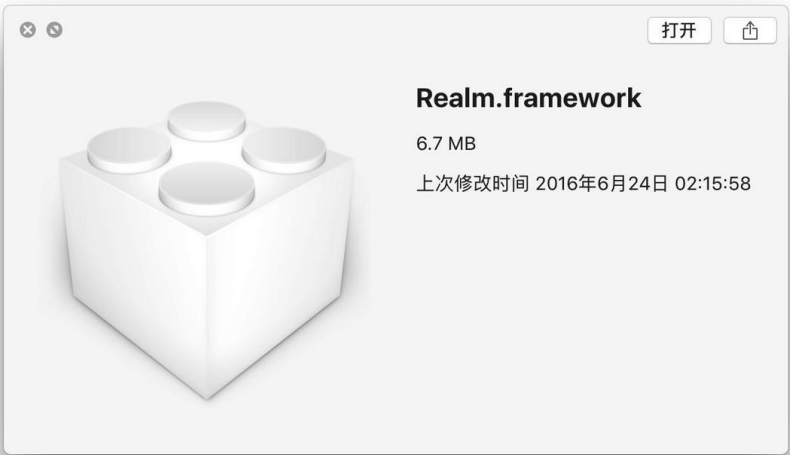
- 当然，我是可以接受安装包大10M 来用 Swift 写的😓。

数据库选择

- 这个问题也是我在分析 Yep 的第三方库的时候发现的问题，因为 Yep 使用的是 Realm，据说是目前是性能最好的移动端数据库。但是在三方库中可以看到，Realm 的支持占了很大的比重，大约在 8M 左右。但是如果使用 FMDB 话只需要192KB，而 CoreData 几乎可以忽略不计。下面是部分截图。



FMDB



Realm

+

🔖

🔗



RealmSwift

MRC VS. ARC

- 最开始是在Bang的这篇文章 (<http://blog.cnbang.net/tech/2544/>)中看到用ARC比用MRC 会导致可执行文件大10%。起初我是不相信的，但是在我用 SDWebImage 的 1.0 版测试之后，ARC 比 MRC 的可执行程序增加了**14% +**。所以MRC 比 ARC 编译成可执行文件之后更小，具体的测试方法可以去他的博客看，这里就不重复了。

总结

- 先分析一下前面几个问题造成的原因：
 - Swift & Realm : 首先 Swift 是因为不稳定，所以支持的动态库没有集成到系统的"dyld的共享缓存"中去。而 Realm 因为不是苹果自己开发的，所以支持的动态库也没有集成到系统的"dyld的共享缓存"中去。所以都内置在了 App 中，而且这两个功能需要写很多代码来实现，因此容量又很大，导致看起来这两个东西占了很大的"无用"的容量。(ps.关于iOS 中库的问题，你可以去我的笔记 (<https://github.com/Damonvong/iOSDevNotes>)中查看~)
 - ARC:因为 ARC 叫做自动引用计数，他的实现方式其实就是 Xcode 在编译的时候自动给你加内存管的代码，但是机器毕竟没人聪明，Xcode 会在很多情况下增加很多没用的代码，这也是为什么 ARC 的底层实现比 MRC 更快，但是实际运行性能上在有些时候却不及 MRC 的原因，而正因为增加了很多没用的代码，ARC 最终编译包会比 MRC 大。
- 总结前面的几个问题，归根结底于一个问题，那就是**Install Smallest VS. Coding Fastest**。很多时候为了追求更快的编码速度，总会有所损失，但是在我看来这些事值得的，不然为什么我们不用 C 来代替 objective-c 或者用汇编来代替 C 呢? 🤔

Bitcode

- bitcode 是被编译程序的一种中间形式的代码。包含 bitcode 配置的程序将会在 App Store 上被编译和链接。bitcode 允许苹果在后期重新优化我们程序的二进制文件，而不需要我们重新提交一个新的版本到 App Store 上。
- 当我们提交程序到 App Store上时，Xcode 会将程序编译为一个中间表现形式(bitcode)。然后 App store 会再将这个 bitcode 编译为可执行的64位或32位程序。
- 所以，通过这个方式，我们可以做到架构级别的**App Slicing**。



Tips

结合上面的内容，再加上Bang大神写的博客
(<http://blog.cnbang.net/tech/2544/>)，我总结了几条 Tips。排名越往前的我觉得越需要去优化。

Tip 1: 去除重复、无用资源文件，解决名字重复问题。

Tip 2: 图片使用.xcassets管理且无须考虑@1x@2x@3x 问题。万不得已再用拖的办法，同时结合一定策略方案进行包瘦身。

Tip 3: 图片使用PDF 优先级高于 PNG,因为 Xcode 会帮你完成剩下的任务。

Tip 4: 使用tinypng (<https://tinypng.com>)压缩PNG图片。视频可以通过 Final cut 等软件进行分辨率压缩。音频则降低码率即可。

Tip 5: icon 使用 iconfont

Tip 6: 非必须资源文件可以放到自己服务器上，但必用资源文件需要内置到安装包中。

Tip 7: HTML 5 需要将图片 Remote 化 或者将整个HTML 5 的页面 Remote 化。

Tip 8: Build Settings->Optimization Leve release版应该选择Fastest, Smallest

Tip 9: 开启 BitCode

以下是几乎不可能去做的优化 Tips

Tip 10: 尽可能的去除无用的代码、控制类名、方法名长度、冗余字符串

Tip 11: 如果你想的话，不使用 Swift、不使用 Realm更甚至于尽量不使用 OC 😊

Tip 12: MRC 比 ARC 编译成可执行文件之后更小。

更多:工作之余，写了点笔记，如果需要可以在我的
GitHub
(<https://github.com/Damonvvong/iOSDevNotes>) 看。

参考文章

- App Thinning
(https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/AppThinning/AppThinning.html#//apple_ref/doc/uid/TP40012582-CH35)
- Confirmed: Objective-C ARC is slow. Don't use it! (sarcasm off) (<http://www.learn-cocos2d.com/2013/03/confirmed-arc-slow/>)
- 4 XCODE ASSET CATALOG SECRETS YOU NEED TO KNOW
(<http://krakendev.io/blog/4-xcode-asset-catalog-secrets-you-need-to-know>)



- 使用IconFont减小iOS应用体积 (<http://johnwong.github.io/mobile/2015/04/03/using-icon-font-in-ios.html>)
- iOS可执行文件瘦身方法 (<http://blog.cnbang.net/tech/2544/>)
- 水平有限，若有错误，希望多多指正！ coderonevv@gmail.com (<mailto:coderonevv@gmail.com>)

iOS-Note (/nb/888253)

举报文章 © 著作权归作者所有




Damonwong (/u/3a9e1894acc2)

写了 21289 字，被 703 人关注，获得了 522 个喜欢 (/u/3a9e1894acc2)

✓ 已关注

一个爱玩游戏的程序员。






喜欢 | 69



更多分享

(<http://cwb.assets.jianshu.io/notes/images/6333907>)

被以下专题收入，发现更多相似内容

- + 收入我的专题
-  有意思 (/c/7dbfdc05bdcc?utm_source=desktop&utm_medium=notes-included-collection)
-  iOS点点滴滴 (/c/50ff01179581?utm_source=desktop&utm_medium=notes-included-collection)
-  iOS Swi... (/c/b6cc50b537d5?utm_source=desktop&utm_medium=notes-included-collection)
-  iOS 开发 (/c/2ffaa203eb6a?utm_source=desktop&utm_medium=notes-included-collection)
-  iOS学习笔记 (/c/bb62bc7671d2?utm_source=desktop&utm_medium=notes-included-collection)
-  移动前沿 (/c/5aac963ca52d?utm_source=desktop&utm_medium=notes-included-collection)
-  iOS Dev... (/c/3233d1a249ca?utm_source=desktop&utm_medium=notes-included-collection)

展开更多 ▾

+

🔖

🔗