美团 iOS 客户端动态库实战

叶槚@美团·大众点评
yeshuang@meituan.com

自我介绍

- 美团·大众点评
  - iOS 基础组件维护
  - iOS 美团客户端持续集成与组件管理优化
- 搜狗
  - 输入法内核 + 云输入

大纲

- 起因
- 理论
- 实战
- 踩坑

起因

- __TEXT 60MB

起因

Your app's total uncompressed size must be less than 4GB. Each Mach-O executable file (for example, `app_name.app/app_name`) must not exceed these limits:

- For apps whose `MinimumOSVersion` is less than 7.0: maximum of 80 MB for the total of all `__TEXT` sections in the binary.
- For apps whose `MinimumOSVersion` is 7.x through 8.x: maximum of 60 MB per slice for the `__TEXT` section of each architecture slice in the binary.
- For apps whose `MinimumOSVersion` is 9.0 or greater: maximum of 500 MB for the total of all `__TEXT` sections in the binary.

起因

Your app's total uncompressed size must be less than 4GB. Each Mach-O executable file (for example, `app_name.app/app_name`) must not exceed these limits:

- For apps whose `MinimumOSVersion` is less than 7.0: maximum of 80 MB for the total of all `__TEXT` sections in the binary.
- For apps whose `MinimumOSVersion` is 7.x through 8.x: maximum of 60 MB per slice for the `__TEXT` section of each architecture slice in the binary.
- For apps whose `MinimumOSVersion` is 9.0 or greater: maximum of 500 MB for the total of all `__TEXT` sections in the binary.

起因

- __TEXT 60MB
  - “Each Mach-O executable file”
  - 拆分动态库

理论

- WWDC2016-406
  - https://developer.apple.com/videos/play/wwdc2016/406/
  - Mach-O, __TEXT, dyld

理论

- 构建过程
  - 编译（+ 汇编）
    - .m => .o
  - 链接
    - .o (+ .o ……) => Executable file
    - 决议符号的地址

理论

- 决议符号的地址
  - 若符号来自静态库或 .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

理论

- 静态库
  - 编译，打包
- 动态库
  - 编译，**链接**
  - “没有 main 的”可执行文件

理论

- Load 装载：将库文件载入内存
  - Static Loading：启动时
  - Dynamic Loading：启动后（使用时）
- 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）

理论

- Static Loading + Static Linking
- Static Loading + Dynamic Linking
- Dynamic Loading + Dynamic Linking
- Dynamic Loading + Static Linking

https://www.quora.com/Systems-Programming-What-is-the-exact-difference-between-Dynamic-loading-and-dynamic-linking

理论

- 动态库的两种常见使用方式
  - Dynamic Loading + Dynamic Linking
    - 手动调用 dlopen
    - 动态库可有可无 => 用时加载，插件
    - 官方不推荐 iOS App 使用（@WWDC2016-406）

理论

- 动态库的两种常见使用方式
  - Static Loading + Dynamic Linking
    - 构建时，动态库要参与构建阶段的链接
      - 只打标记不决议符号地址
    - 启动时，由 dyld 自动完成 Load + Link

理论

- 接下来讨论的方式
  - Dynamic Loading + Dynamic Linking
    - 调用方知道符号来自动态库
    - 必须 dlopen + performSelector
  - Static Loading + Dynamic Linking
    - 符号的调用方不知情（源码 / 静态库 / 动态库）
    - 直接调

理论

- 接下来讨论的方式
  - Dynamic Loading + Dynamic Linking
    - 符号的调用方知道是动态库
    - 必须 dlopen + performSelector
  - **Static Loading + Dynamic Linking**
    - **符号的调用方不知情（源码 / 静态库 / 动态库）**
    - **直接调**

理论

- 库之间的依赖问题
  - 静态库 vs 动态库
  - 2 * 2 = 4

深呼吸

理论

- libAuth.a 依赖 libCrypto.a
  - 都进入主程序
  - 有 libCrypto 的头文件就可以做出 libAuth.a

理论

- UIKit.dylib 依赖 Foundation.dylib
  - 彼此独立，且都不会进入主程序
    - “隔离性”
  - UIKit.dylib 的输出需要 Foundation.dylib 参与链接

理论

- libChat.a 依赖 Foundation.dylib
  - libChat 进入主程序，Foundation 不会进入主程序

理论

- ABCNetwork.dylib 依赖 libAFNetworking.a
  - AFNetworking 的内容进入 ABCNetwork.dylib
    - “吸附性”
  - ABCNetwork 不会进入主程序
  - ABCNetwork.dylib 的输出需要 libAFNetworking.a 参与链接

理论

- ABCNetwork.dylib 和 XYZNetwork.dylib 都依赖 libAFNetworking.a
  - 两个动态库里面都会有 AFNetworking 的内容

理论

- 依赖问题总结
  - 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

理论

- Xcode
  - target 对应一个产物
  - project 包含若干个 target
  - workspace 包含若干个 project

理论

- CocoaPods
  - App.workspace
    - App.project
    - Pods.project
      - pod target => .a

进入实战！

实战

- CocoaPods 的 use_frameworks! 用不了
  - 数目问题
    - 26 dylibs => 2 dylibs，240.09ms => 21.75ms
    - @WWDC2016-406
  - 依赖问题
    - 二进制 pod

实战

- 手动拆分
  - 选取部分组件，集体输出为一个动态库
  - 美团 App => 一个主程序 + 一个 Embedded Framework

Podfile

```
platform :ios, '8.0'

source 'https://github.com/CocoaPods/Specs.git'

project 'hello-iOS.xcodeproj'
target 'hello-iOS' do
  pod 'AFNetworking'
  pod 'SDWebImage'
end

target 'HelloDylibCollection' do
  pod 'React'
  pod 'ReactiveCocoa'
end
```

小目标达成

面对依赖关系复杂的私有库

实战

- 依赖问题
  - pod-Chat 进入动态库
  - pod-Mail 进入主程序
  - pod-Chat 和 pod-Mail 都依赖 pod-Util
  - 主程序与动态库都有 pod-Util 的内容

进入分支剧情

分支剧情

- pod install 加速
- 平摊所有依赖，每个组件精确版本号
- 自定义 podspec
  - :podspec => ‘xxx.podspec'
  - Hack 依赖字段
- 每个 pod 都是独立（孤立）的个体

分支剧情

- 去依赖 pod install
  - Trade-off
  - 获得：速度提升
  - 获得：公司项目组件版本的精准控制
  - 失去：自动语义化版本号决议功能

回到主线剧情

实战

- 动态库征召总动员
  - 必须自底向上，从叶子节点选起
  - 当一个 pod 的所有依赖进入了动态库，该 pod 才能进入动态库

实战

- 为什么必须从叶子节点开始
  - pod-Chat 依赖 pod-Util
  - pod-Chat 选进动态库，pod-Util 没有
  - 动态库链接阶段，找不到 pod-Util 的符号，报错

从叶子节点开始，一个一个一层一层征召

实战

- 征召名单变更对业务开发透明
- 避免频繁修改 Podfile

Podfile

```
platform :ios, '8.0'

source 'https://github.com/CocoaPods/Specs.git'

project 'hello-iOS.xcodeproj'
target 'hello-iOS' do
  pod 'AFNetworking'
  pod 'SDWebImage'
end

target 'HelloDylibCollection' do
  pod 'React'
  pod 'ReactiveCocoa'
end
```

实战

- 对业务开发透明，避免频繁修改 Podfile
  - 将动态库征召逻辑写入独立的 .rb 文件
  - 仍然在主程序 target 域内声明所有 pod
  - 延迟声明进入动态库的 pod

Podfile

```
platform :ios, '8.0'

source 'xxx.git'

require 'my_pod.rb'

project 'hello-iOS.xcodeproj'
target 'hello-iOS' do
  my_pod 'AFNetworking', '3.1.0'
  my_pod 'SDWebImage', '3.8.2'
  my_pod 'React', '0.43.3'
  my_pod 'ReactiveCocoa', '2.5.0'
end

target 'METDyldCollection' do
  dyld_pods
end
```

Podfile

```ruby
platform :ios, '8.0'

source 'xxx.git'

require 'my_pod.rb'

project 'hello-iOS.xcodepr
target 'hello-iOS' do
  my_pod 'AFNetworking', '
  my_pod 'SDWebImage', '3.
  my_pod 'React', '0.43.3'
  my_pod 'ReactiveCocoa',
end

target 'METDyldCollection'
  dyld_pods
end
```

my_pod.rb

```ruby
DylibWhiteList = ['React', 'ReactiveCocoa']
DyldCollection = {}

def dyld_pods
  DyldCollection.each do |pod_name, info|
    pod pod_name, info
  end
end

def emit_one_pod(pod_name, info)
  pod_name_base = pod_name.split('/').first
  if DylibWhiteList.include?(pod_name_base)
    DyldCollection[pod_name] = info
  else
    pod pod_name, info
  end
end

def my_pod(pod_name, version)
  pod_name_base = pod_name.split('/').first
  podspec_url = 'http://xxx/xxx.podspec'
  info = {:podspec => podspec_url}
  emit_one_pod(pod_name, info)
end
```

坑

- 私有符号
  - 动态库中的私有符号，链接阶段是不可见的
  - https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/CppRuntimeEnv/Articles/SymbolVisibility.html

```
__attribute__((visibility("hidden")))
@interface MyTestPrivateClass : NSObject
+(void)foo;
@end

@interface MyTestPublicClass : NSObject
+(void)foo;
@end
```

```
$ nm -m test.dylib | grep _MyTest
0000000000020cdd0 (__DATA,__objc_data) non-external (was a private external) _OBJC_CLASS_$_MyTestPrivateClass
0000000000020ce20 (__DATA,__objc_data) external _OBJC_CLASS_$_MyTestPublicClass
0000000000020cda8 (__DATA,__objc_data) non-external (was a private external) _OBJC_METACLASS_$_MyTestPrivateClass
0000000000020cdf8 (__DATA,__objc_data) external _OBJC_METACLASS_$_MyTestPublicClass
```

坑

- [NSBundle mainBundle]
  - podspec 中描述的资源 => pod 所属的 target
  - [NSBundle mainBundle] => 主程序 target

坑

- 引入动态库之后，使用 NSBundle 的正确姿势
  - [NSBundle bundleForClass: [xxx class]]
    - 目标资源所属 pod 中的类名
    - 引用自己的资源，用 self（坑中坑预警）

坑

- 引入动态库之后，使用 NSBundle 的正确姿势
  - [UIImage imageNamed:] 只在主程序 target 中找
  - [UIImage imageNamed:inBundle:compatibleWithTraitCollection:]
    - inBundle 参数值用 bundleForClass 找
    - iOS 8+，做好 if 防御

坑

- 给 UIImage 来个完美的封装?

```objc
@implementation UIImage (DylibExt)

+ (UIImage *)mt_imageNamed:(NSString *)name
{
    if ([[UIImage class] respondsToSelector:@selector(imageNamed:inBundle:
        compatibleWithTraitCollection:)]) {
        NSBundle *bundle = [NSBundle bundleForClass:[self class]];
        return [UIImage imageNamed:name inBundle:bundle
            compatibleWithTraitCollection:nil];
    }
    return [UIImage imageNamed:name];
}
```

坑

- 坑中坑：category + self

```
11 @implementation UIImage (DylibExt)
12
13 + (UIImage *)mt_imageNamed:(NSString *)name
14 {
15     if ([[UIImage class] respondsToSelector:@selector(imageNamed:inBundle:
           compatibleWithTraitCollection:)]) {
16         NSBundle *bundle = [NSBundle bundleForClass:[self class]];
17         return [UIImage imageNamed:name inBundle:bundle
               compatibleWithTraitCollection:nil];
18     }
19     return [UIImage imageNamed:name];
20 }
```

坑

- 找不到纯 C 符号
  - 动态库内无人引用的函数，参与链接但会被抛弃
  - 解决方法：
    - 使用 -force_load 链接参数强制导入某个静态库
      - 简单直接，但可能有空间浪费
    - 动态库 target 里写 Dummy 代码，主动引用这些符号
      - 精细粒度，但略繁琐

__TEXT 最后的续命锦囊

- 来自 Facebook 的链接参数
  - https://everettjf.github.io/2016/08/19/facebook-explore-section-rodata
  - Other Linker Flags

__TEXT 最后的续命锦囊

```
-Wl,-rename_section,__TEXT,__cstring,__RODATA,__cstring
-Wl,-rename_section,__TEXT,__gcc_except_tab,__RODATA,__gcc_except_tab
-Wl,-rename_section,__TEXT,__const,__RODATA,__const
-Wl,-rename_section,__TEXT,__objc_methname,__RODATA,__objc_methname
-Wl,-rename_section,__TEXT,__objc_classname,__RODATA,__objc_classname
-Wl,-rename_section,__TEXT,__objc_methtype,__RODATA,__objc_methtype
```

回顾

- __TEXT 小于 60 MB 是对单文件单架构的要求
- 分离动态库
- 动态库越少越好
- 动态库的“吸附性”与“隔离性”
- 依赖传递，去依赖化与自底向下

一家之言，仅供参考，请多指教

感谢聆听

Q&A