iOS 静态库，动态库与 Framework

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看到源码，就需要以库的形式进行封装，只暴露出头文件。另外一种情况是，对于某些不会进行大的改动的代码，我们想减少编译的时间，就可以把它打包成库，因为库是已经编译好的二进制了，编译的时候只需要 Link 一下，不会浪费编译时间。

上面提到库在使用的时候需要 Link，Link 的方式有两种，静态和动态，于是便产生了静态库和动态库。

静态库

静态库即静态链接库（Windows 下的 .lib，Linux 和 Mac 下的 .a）。之所以叫做静态，是因为静态库在编译的时候会被直接拷贝一份，复制到目标程序里，这段代码在目标程序里就不会再改变了。

静态库的好处很明显，编译完成之后，库文件实际上就没有作用了。目标程序没有外部依赖，直接就可以运行。当然其缺点也很明显，就是会使用目标程序的体积增大。

动态库

动态库即动态链接库（Windows 下的 .dll，Linux 下的 .so，Mac 下的 .dylib/.tbd）。与静态库相反，动态库在编译时并不会被拷贝到目标程序中，目标程序中只会存储指向动态库的引用。等到程序运行时，动态库才会被真正加载进来。

动态库的优点是，不需要拷贝到目标程序中，不会影响目标程序的体积，而且同一份库可以被多个程序使用（因为这个原因，动态库也被称作**共享库**）。同时，编译时才载入的特性，也可以让我们随时对库进行替换，而

不需要重新编译代码。动态库带来的问题主要是，动态载入会带来一部分性能损失，使用动态库也会使得程序依赖于外部环境。如果环境缺少动态库或者库的版本不正确，就会导致程序无法运行（Linux 下喜闻乐见的 lib not found 错误）。

iOS Framework

除了上面提到的 .a 和 .dylib/.tbd 之外，Mac OS/iOS 平台还可以使用 Framework。Framework 实际上是一种打包方式，将库的二进制文件，头文件和有关的资源文件打包到一起，方便管理和分发。

在 iOS 8 之前，iOS 平台不支持使用动态 Framework，开发者可以使用的 Framework 只有苹果自家的 UIKit.Framework，Foundation.Framework 等。这种限制可能是出于安全的考虑（见这里的讨论）。换一个角度讲，因为 iOS 应用都是运行在沙盒当中，不同的程序之间不能共享代码，同时动态下载代码又是被苹果明令禁止的，没办法发挥出动态库的优势，实际上动态库也就没有存在的必要了。

由于上面提到的限制，开发者想要在 iOS 平台共享代码，唯一的选择就是打包成静态库 .a 文件，同时附上头文件（例如微信的SDK）。但是这样的打包方式不够方便，使用时也比较麻烦，大家还是希望共享代码都能能像 Framework 一样，直接扔到工程里就可以用。于是人们想出了各种奇技淫巧去让 Xcode Build 出 iOS 可以使用的 Framework，具体做法参考这里和这里，这种方法产生的 Framework 还有“伪”(Fake) Framework 和“真”(Real) Framework 的区别。

iOS 8/Xcode 6 推出之后，iOS 平台添加了动态库的支持，同时 Xcode 6 也原生自带了 Framework 支持（动态和静态都可以），上面提到的的奇技淫巧也就没有必要了（新的做法参考这里）。为什么 iOS 8 要添加动态库的支持？唯一的理由大概就是 Extension 的出现。Extension 和 App 是两个分开的可执行文件，同时需要共享代码，这种情况下动态库的支持就是必不可少的了。但是这种动态 Framework 和系统的 UIKit.Framework 还是有很大区别。系统的 Framework 不需要拷贝到目标程序中，我们自己做出来的 Framework 哪怕是动态的，最后也还是要拷贝到 App 中（App 和 Extension 的 Bundle 是共享的），因此苹果又把这种 Framework 称为 Embedded Framework。

Swift 支持

跟着 iOS8 / Xcode 6 同时发布的还有 Swift。如果要在项目中使用外部的代码，可选的方式只有两种，一种是把代码拷贝到工程中，另一种是用动态 Framework。使用静态库是不支持的。

造成这个问题的原因主要是 Swift 的运行库没有被包含在 iOS 系统中，而是会打包进 App 中（这也是造成 Swift App 体积大的原因），静态库会导致最终的目标程序中包含重复的运行库（这是苹果自家的解释）。同时拷贝 Runtime 这种做法也会导致在纯 ObjC 的项目中使用 Swift 库出现问题。苹果声称等到 Swift 的 Runtime 稳定之后会被加入到系统当中，到时候这个限制就会被去除了（参考这个问题 的问题描述，也是来自苹果自家文档）。

CocoaPods 的做法

在纯 ObjC 的项目中，CocoaPods 使用编译静态库 .a 方法将代码集成到项目中。在 Pods 项目中的每个 target 都对应这一个 Pod 的静态库。不过在编译过程中并不会真的产出 .a 文件。如果需要 .a 文件的话，可以参考这里，或者使用 CocoasPods-Packager 这个插件。

当不想发布代码的时候，也可以使用 Framework 发布 Pod，CocoaPods 提供了 `vendored_framework` 选项来使用第三方 Framework，具体的做法可以参考这里和这里。

对于 Swift 项目，CocoaPods 提供了动态 Framework 的支持。通过 `use_frameworks!` 选项控制。对于 Swift 写的库来说，想通过 CocoaPods 引入工程，必须加入 `use_frameworks!` 选项。具体原因参见上一节对于 Swift 部分的介绍。

更多有关代码分发的扩展资料可以参考这篇博客：
http://geeklu.com/2014/02/objc-lib/

参考资料

- https://stackoverflow.com/questions/2649334/difference-between-static-and-shared-libraries

- https://stackoverflow.com/questions/25080914/will-ios-8-support-dynamic-linking
- https://stackoverflow.com/questions/6245761/difference-between-framework-and-static-library-in-xcode4-and-how-to-call-them
- http://blog.cocoapods.org/CocoaPods-0.36/