iOS开发～制作同时支持armv7,armv7s,arm64,i386,x86\_64的静态库.a

原文地址：http://www.cocoachina.com/ios/20170623/19619.html

**armv7**,**armv7s**,**arm64**,**i386**,**x86\_64** 详解

一、概要

平时项目开发中，可能使用第三方提供的静态库**.a**，如果**.a**提供方技术不成熟，使用的时候就会出现问题，例如：

在真机上编译报错：**No architectures to compile for** (**ONLY\_ACTIVE\_ARCH=YES**, **active arch=x86\_64**, **VALID\_ARCHS=i386**)**.**

在模拟器上编译报错：**No architectures to compile for** (**ONLY\_ACTIVE\_ARCH=YES**, **active arch=armv7s**, **VALID\_ARCHS=armv7 armv6**)**.**

要解决以上问题，就要了解一下**Apple**移动设备处理器指令集相关的一些细节知识。

二、几个重要概念

**1**、**ARM**

**ARM**处理器，特点是体积小、低功耗、低成本、高性能，所以几乎所有手机处理器都基于**ARM**，在嵌入式系统中应用广泛。

**2**、**ARM**处理器指令集

**armv6**｜**armv7**｜**armv7s**｜**arm64**都是**ARM**处理器的指令集，这些指令集都是向下兼容的，例如**armv7**指令集兼容**armv6**，只是使用**armv6**的时候无法发挥出其性能，无法使用**armv7**的新特性，从而会导致程序执行效率没那么高。

还有两个我们也很熟悉的指令集：**i386**｜**x86\_64** 是**Mac**处理器的指令集，**i386**是针对**intel**通用微处理器**32**架构的。**x86\_64**是针对**x86**架构的**64**位处理器。所以当使用**iOS**模拟器的时候会遇到**i386**｜**x86\_64**，**ios**模拟器没有**arm**指令集。

**3**、目前**iOS**移动设备指令集

**arm64**：**iPhone5S**｜ **iPad Air**｜ **iPad mini2**(**iPad mini with Retina Display**)

**armv7s**：**iPhone5**｜**iPhone5C**｜**iPad4**(**iPad with Retina Display**)

**armv7**：**iPhone3GS**｜**iPhone4**｜**iPhone4S**｜**iPad**｜**iPad2**｜**iPad3**(**The New iPad**)｜**iPad mini**｜**iPod Touch 3G**｜**iPod Touch4**

**armv6** 设备： **iPhone**, **iPhone2**, **iPhone3G**,第一代、第二代 **iPod Touch**（一般不需要去支持）

**4**、**Xcode**中指令集相关选项（**Build Setting**中）

（**1**）**Architectures**

**Space-separated list of identifiers. Specifies the architectures** (**ABIs**, **processor models**) **to which the binary is targeted. When this build setting specifies more than one architecture**, **the generated binary may contain object code for each of the specified architectures.**

指定工程被编译成可支持哪些指令集类型，而支持的指令集越多，就会编译出包含多个指令集代码的数据包，对应生成二进制包就越大，也就是**ipa**包会变大。

（**2**）**Valid Architectures**

**Space-separated list of identifiers. Specifies the architectures for which the binary may be built. During the build**, **this list is intersected with the value of ARCHS build setting**; **the resulting list specifies the architectures the binary can run on. If the resulting architecture list is empty**, **the target generates no binary.**

限制可能被支持的指令集的范围，也就是**Xcode**编译出来的二进制包类型最终从这些类型产生，而编译出哪种指令集的包，将由**Architectures**与**Valid Architectures**（因此这个不能为空）的交集来确定，例如：

比如，你的**Valid Architectures**设置的支持**arm**指令集版本有：**armv7/armv7s/arm64**，对应的**Architectures**设置的支持**arm**指令集版本有：**armv7s**，这时**Xcode**只会生成一个**armv7s**指令集的二进制包。

再比如：将**Architectures**支持**arm**指令集设置为：**armv7**,**armv7s**，对应的**Valid Architectures**的支持的指令集设置为：**armv7s**,**arm64**，那么此时，**XCode**生成二进制包所支持的指令集只有**armv7s**

在**Xcode6.1.1**里的 **Valid Architectures**设置里，默认为 **Standard architectures**(**armv7**,**arm64**),如果你想改的话，自己在**other**中更改。

原因解释如下：

使用 **standard architectures** (**including 64-bit**)(**armv7**,**arm64**)参数，则打的包里面有**32**位、**64**位两份代码，在**iPhone5s**（ **iPhone5s**的**cpu**是**64**位的）下，会首选运行**64**位代码包，其余的**iPhone**（其余**iPhone**都是**32**位的,**iPhone5c**也是**32**位），只能运行**32**位包，但是包含两种架构的代码包，只有运行在**ios6**，**ios7**系统上。

这也就是说，这种打包方式，对手机几乎没要求，但是对系统有要求，即**ios6**以上。

而使用 **standard architectures** (**armv7**,**armv7s**)参数，则打的包里只有**32**位代码， **iPhone5s**的**cpu**是**64**位，但是可以兼容**32**位代码，即可以运行**32**位代码。但是这会降低**iPhone5s**的性能。其余的**iPhone**对**32**位代码包更没问题，而**32**位代码包，对系统也几乎也没什么限制。

所以总结如下：

要发挥**iPhone5s**的**64**位性能，就要包含**64**位包，那么系统最低要求为**ios6**。如果要兼容**ios5**以及更低的系统，只能打**32**位的包，系统都能通用，但是会丧失**iPhone5s**的性能。

（**3**）**Build Active Architecture Only**

指定是否只对当前连接设备所支持的指令集编译

当其值设置为**YES**，这个属性设置为**yes**，是为了**debug**的时候编译速度更快，它只编译当前的**architecture**版本，而设置为**no**时，会编译所有的版本。编译出的版本是向下兼容的，连接的设备的指令集匹配是由高到低（**arm64 > armv7s > armv7**）依次匹配的。比如你设置此值为**yes**，用**iphone4**编译出来的是**armv7**版本的，**iphone5**也可以运行，但是**armv6**的设备就不能运行。所以，一般**debug**的时候可以选择设置为**yes**，**release**的时候要改为**no**，以适应不同设备。

**1**）

**Architectures:  armv7**, **armv7s**, **arm64**

**ValidArchitectures:  armv6**, **armv7s**, **arm64**

生成二进制包支持的指令集： **arm64**

**2**）

**Architectures: armv6**, **armv7**, **armv7s**

**Valid Architectures:  armv6**, **armv7s**, **arm64**

生成二进制包支持的指令集： **armv7s**

**3**）

**Architectures: armv7**, **armv7s**, **arm64**

**Valid Architectures: armv7**，**armv7s**

这种情况是报错的，因为允许使用指令集中没有**arm64**。

注：如果你对**ipa**安装包大小有要求，可以减少安装包的指令集的数量，这样就可以尽可能的减少包的大小。当然这样做会使部分设备出现性能损失，当然在普通应用中这点体现几乎感觉不到，至少不会威胁到用户体检。

三、制作静态库**.a**是指令集选择

现在回归到正题，如何制作一个“没有问题”的**.a**静态库，通过以上信息了解到，当我们做**App**的时候，为了追求高效率，并且减小包的大小，**Build Active Architecture Only**设置成**YES**，**Architectures**按**Xcode**默认配置就可以，因为**arm64**向前兼容。但制作**.a**静态库就不同了，因为要保证兼容性，包括不同**iOS**设备以及模拟器运行不出错，所以结合当前行业情况，要做到最大的兼容性。

1. ValidArchitectures设置为：armv7｜armv7s｜arm64｜i386｜x86\_64
2. Architectures设置不变（或根据你需要）:  armv7｜arm64

然后分别选择**iOS**设备和模拟器进行编译，最后找到相关的**.a**进行合包，使用**lipo -create** 真机库**.a**的路径模拟器库**.a**的的路径－**output** 合成库的名字**.a**（详情可以参考**http://blog.csdn.NET/lizhongfu2013/article/details/12648633**）

这样就制作了一个通用的静态库**.a**

**.a** 与 **FrameWork** 合并技巧

查看**.a** 和 **FrameWork** 的适配机型

lipo -create Debug-iphoneos/libMJRefresh.a Debug-iphonesimulator/libMJRefresh.a -output libMJRefresh.a

lipo -create /Users/harvey/Library/Developer/Xcode/DerivedData/FMDB-ctegiztcjikewoeprxxtmryzetfa/Build/Products/Release-iphoneos/libFMDB.a /Users/harvey/Library/Developer/Xcode/DerivedData/FMDB-ctegiztcjikewoeprxxtmryzetfa/Build/Products/Release-iphonesimulator/libFMDB.a -output /Users/harvey/Desktop/libFMDB.a

lipo -create /Users/harvey/Library/Developer/Xcode/DerivedData/FMDB-clvayfrjgytqrbdkyqrtcjkxfeuz/Build/Products/Release-iphonesimulator/FMDB.framework/FMDB /Users/harvey/Library/Developer/Xcode/DerivedData/FMDB-clvayfrjgytqrbdkyqrtcjkxfeuz/Build/Products/Release-iphoneos/Release-iphoneos.framework/FMDB -output /Users/harvey/Desktop/FMDB

Architectures in the fat file: /Users/kunkkaqi/Desktop/TestFrameWork.framework/TestFrameWork.lipo are: i386 armv7 armv7s x86\_64 arm64