IOS_Bowen

iOS封装功能生成 .framework

前言

如果你想将你开发的控件与别人分享，一种方法是直接提供源代码文件。然而，这种方法并不是很优雅。它会**暴露所有的实现细节**，而这些实现你可能并不想开源出来。此外，开发者也可能并不想看到你的所有代码，因为他们可能仅仅希望将你的这份漂亮代码的一部分植入自己的应用中。

另一种方法是将你的代码编译成静态库（library），让其他开发者添加到自己的项目中。然而，这需要你一并公布所有的公开的头文件，实在是非常不方便。

你需要一种简单的方法来编译你的代码，这种方法应该使得你的代码易分享，并且在多个工程中易复用。你需要的是一种方法来打包你的静态库，将所有的头文件放到一个单元中，这样你就可以立刻将其加入到你的项目中并使用。

OS X完美地支持这一点，因为Xcode就提供了一个项目模板，包含着默认构建目标（target）和可以容纳类似于图片、声音、字体等资源的文件。你可以为iOS创建Framework，不过这是一个比较复杂的手工活，如果你跟着教程走，你将学到怎么样跨过路障，顺利地完成Framework的创建。

比较

可以参考这篇文章.a和.framework.a和.framework的区别。

一、什么是库?

库是共享程序代码的方式，一般分为静态库和动态库。

二、静态库与动态库的区别?

静态库：链接时完整地拷贝至可执行文件中，被多次使用就有多份冗余拷贝。

动态库：链接时不复制，程序运行时由系统动态加载到内存，供程序调用，系统只加载一次，多个程序共用，节省内存

三、iOS里静态库形式?

.a和.framework

四、iOS里动态库形式?

.dylib和.framework

五、framework为什么既是静态库又是动态库?

系统的.framework是动态库，我们自己建立的.framework是静态库。

六、a与.framework有什么区别?

.a是一个纯二进制文件，.framework中除了有二进制文件之外还有资源文件。

.a文件不能直接使用，至少要有.h文件配合，.framework文件可以直接使用。

.a + .h + sourceFile = .framework。

建议用.framework.

七、为什么要使用静态库?

方便共享代码，便于合理使用。

实现iOS程序的模块化。可以把固定的业务模块化成静态库。

和别人分享你的代码库，但不想让别人看到你代码的实现。

开发第三方sdk的需要。

我们可以看出.a的封装和.framework的封装差不多，也有模拟器和真机合并的过程，通过上边的图片我们可以看出.a 和.framework的区别，就是.a+.h+soureFile=.framework。可以看出我们直接封装.framework其实是最好的。那么我们就来看看framework怎么封装的。

另外关于.a的封装大家可以参考iOS如何生成.a文件

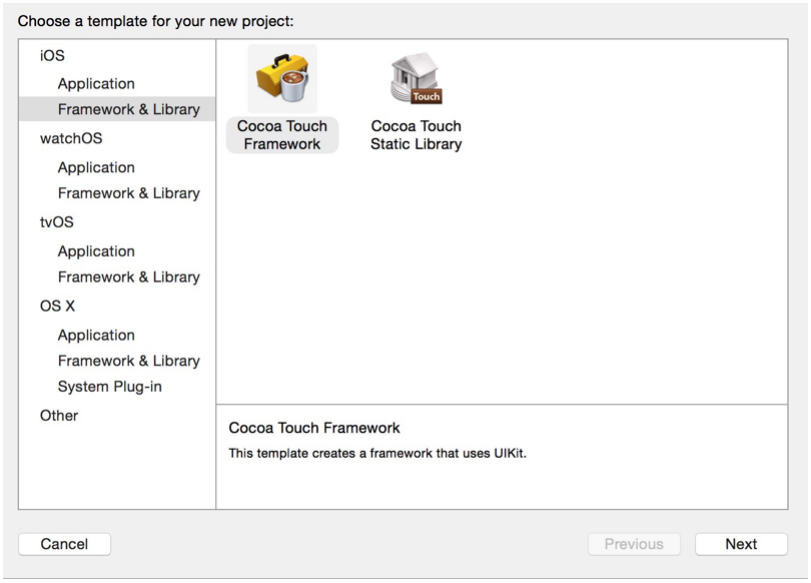
目标

本文将基于Xcode7创建一个简单的工程，通过两种方法来教大家如何制作一个自己的framework，目的就是**简单易学的制作framework**。这种方法可以使得你的代码**易分享**，在多个工程中复用，并且可以**隐藏实现细节**，**控制公开的头文件**。

步骤

1、打开Xcode，新建工程。

不要选择"Application"，选择"**Framework & Library**"。选择第一个，然后Next。



2、创建功能类。

这里我创建一个继承自NSObject的SayHello类

3、实现功能。

在新创建的类里面声明方法并实现。这里我写一个sayHello的方法，以便后面测试使用。

SayHello.h

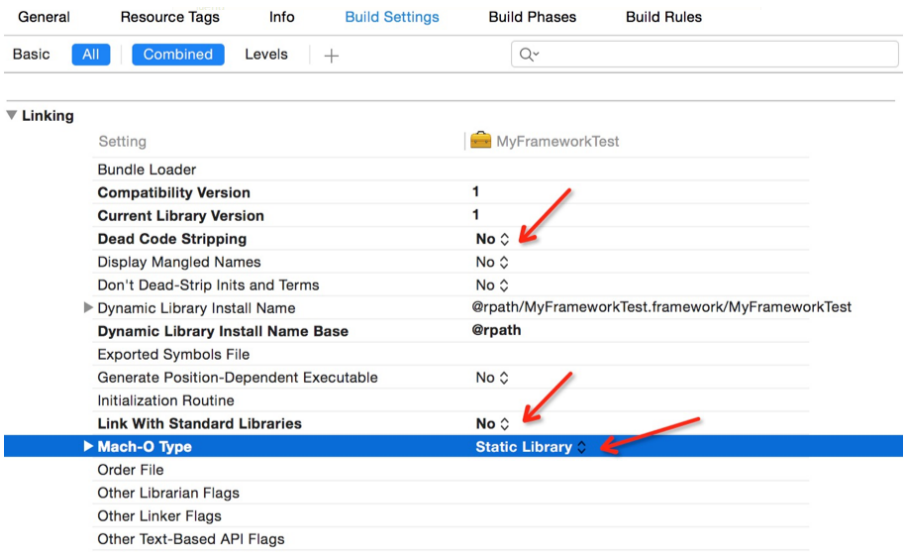
```
#import <Foundation/Foundation.h>
@interface SayHello : NSObject
-(void)sayHello;
@end
```

SayHello.m

```
 #import "SayHello.h"
@implementation SayHello
-(void)sayHello
 {
  NSLog(@"你好，第一次见面，请多关照");
 }
 @end
```

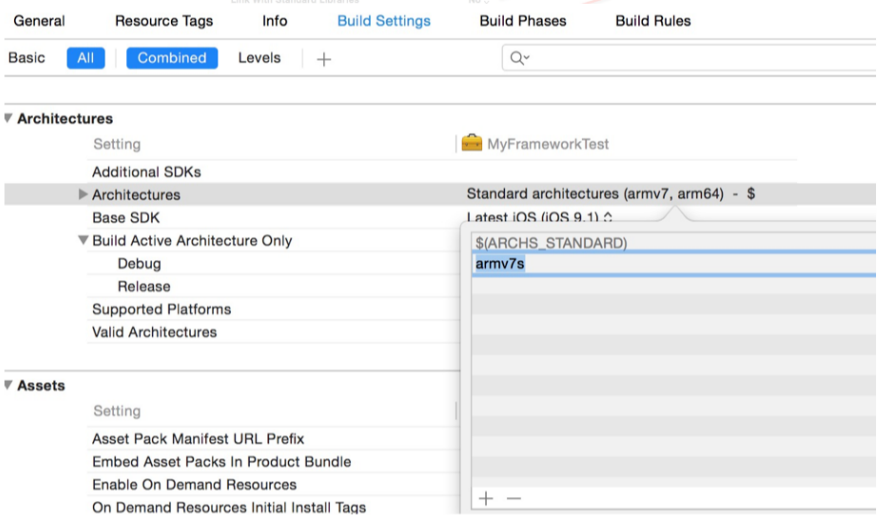
4、**更改参数**

在TARGETS下选中工程，在Build Settings下更改几个参数。



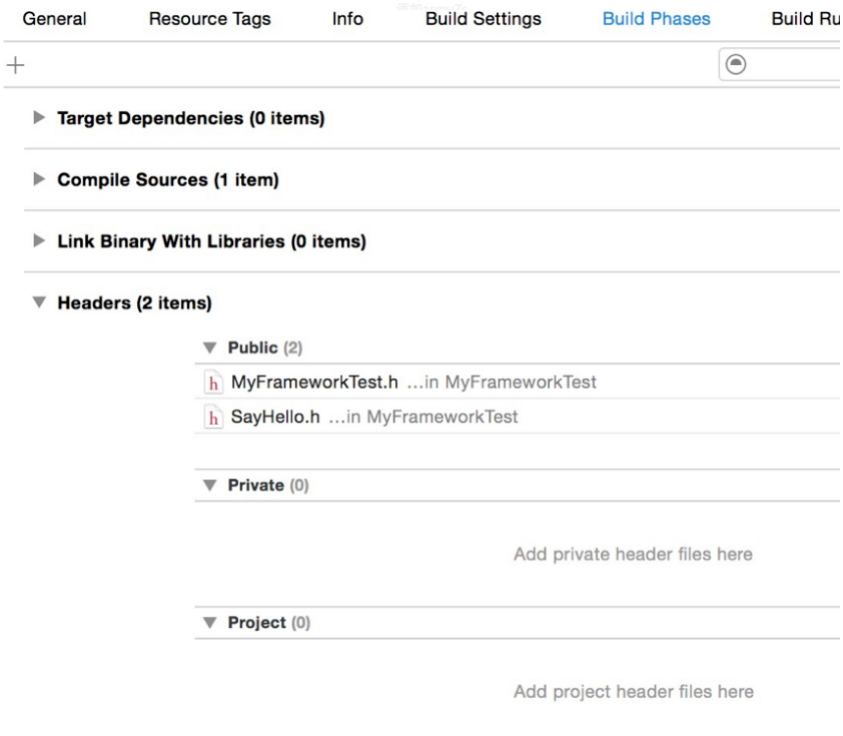
5、增加armv7s

在Architectures下增加armv7s，并选中。将Build Active Architecture Only 设置为NO。



6、设置Headers

将你要公开的头文件**拖至**Public下，要隐藏的放在Private或者Project下，当然，隐藏的头文件就无法再被引用。



然后需要在Test.h（必须是公开的，否则无法引用）中将你所有要公开的.h引入。

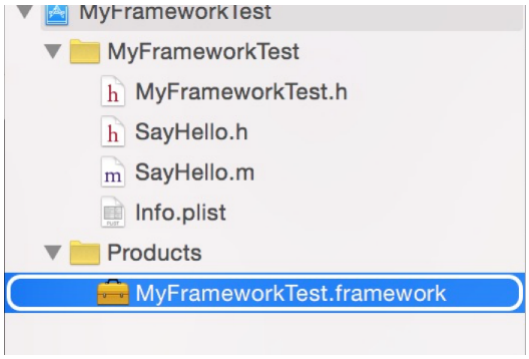
打包Framework

第一种方法

1.选中模拟器，编译程序

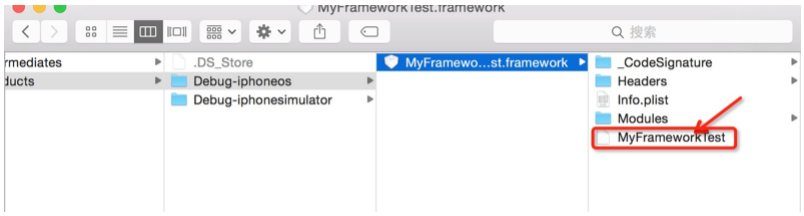
2.选中测试机，编译程序

3.在finder中找到framework文件



选中图中所标示的framework，然后右键show in finder。

找到下图中所示的Test文件，一个是Debug-iphoneos（真机）下的，一个是Debug-iphonesimulator(模拟器)下的。



4.通过终端命令将两个framework合为一个模拟器和真机都可使用的framework。

打开控制台输入 lipo -create iphoneos下frameworkTest的路径 simulator下frameworkTest的路径 -output 新的路径，这样就完成了模拟器和真机版本的合并，新路径下的frameworkTest就是你合并后的文件，将这个文件名字改成和你未合并之前的Test一样的名字，放到framework文件夹下，替换掉原来的frameworkTest文件。

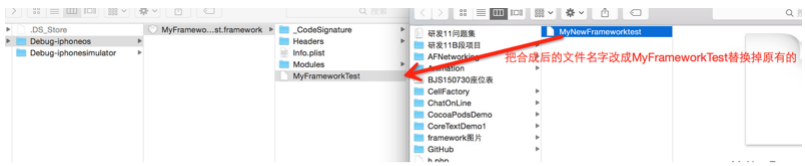
上边说的乱糟糟的，看不清楚，这里给大家解释一下，看下边的图：**打开终端，手动输入**画红线的**lipo -create**命令，然后绿线是iphoneos下frameworkTest的路径（找到iphoneos下frameworkTest的文件，**拖拽进来**），会自动有空格，紫线是simulator下frameworkTest的路径（同样找到simulator下frameworkTest的文件，**拖拽进来**），也会自动有空格，然后**输入-output**，然后敲空格，在引入一个新的路径（**拖拽进一个新的路径**），最后敲回车。这样就完成合并了。

```
[redacted]$ lipo -create /Users/chenxiangyang/Li
brary/Developer/Xcode/DerivedData/MyFrameworkTest-cazpdrvnhmfxwybteitjdaakblot/B
uild/Products/Debug-iphoneos/MyFrameworkTest.framework/MyFrameworkTest /Users/ch
enxiangyang/Library/Developer/Xcode/DerivedData/MyFrameworkTest-cazpdrvnhmfxwybt
eitjdaakblot/Build/Products/Debug-iphonesimulator/MyFrameworkTest.framework/MyFr
ameworkTest -output /Users/chenxiangyang/Desktop/New/MyNewFrameworktest
```

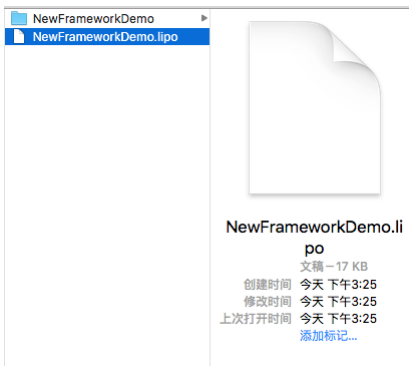
上面这段命令就是把真机和模拟器的frameworkTest合并成一个MyNewFrameworktest文件并存放在桌面上的New文件夹下。

这里我们合并的时候会遇到一个error，这是啥原因还真不知道，但是会在和我们－output的文件夹路径并列的地方生成一个.lipo文件，这个.lipo文件我们下边会说到。

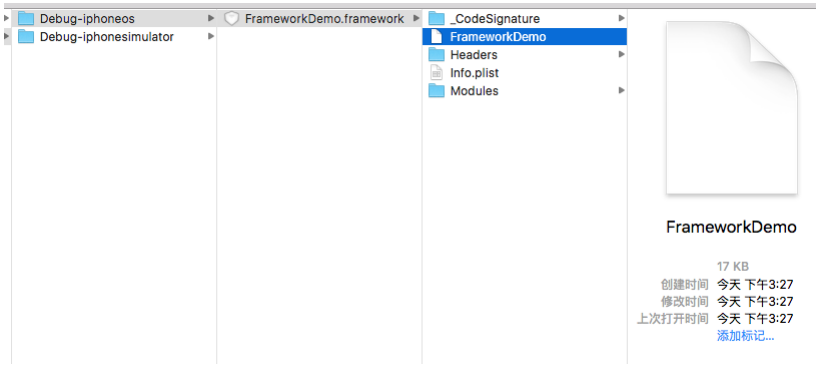
```
[redacted]iMac:~ hjm$ lipo -create /Users/hjm/Library/Developer/Xcode/DerivedDat
a/FrameworkDemo-dookmwebnbfhvdarnialalymubgk/Build/Products/Debug-iphoneos/Frame
workDemo.framework/FrameworkDemo /Users/hjm/Library/Developer/Xcode/DerivedData/
FrameworkDemo-dookmwebnbfhvdarnialalymubgk/Build/Products/Debug-iphonesimulator/
FrameworkDemo.framework/FrameworkDemo -output /Users/hjm/Desktop/未命名文件夹\ 2

error: /Applications/Xcode 3.app/Contents/Developer/Toolchains/XcodeDefault.xcto
olchain/usr/bin/lipo: can't move temporary file: /Users/hjm/Desktop/未命名文件夹
 2 to file: /Users/hjm/Desktop/未命名文件夹 2.lipo (Is a directory)
[redacted]iMac:~ hjm$
```



注意：合并完成后会出现一个如下图的.lipo格式的文件。



这TM是啥，不是应该出现一个类似下图的吗？不应该后缀什么也没有吗？怎么后缀会是.lipo，这是什么文件啊？！



我们的操作是按照人家说的**把合成后的文件名字改成MyFrameworkTest替换原来的**。而且，**把后缀.lipo去掉**！

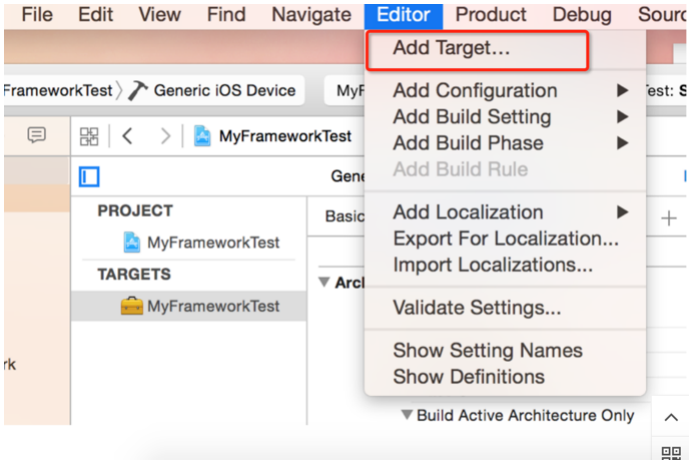
在按照上述说的，替换了原来的。

然后就可以进行下一步了。

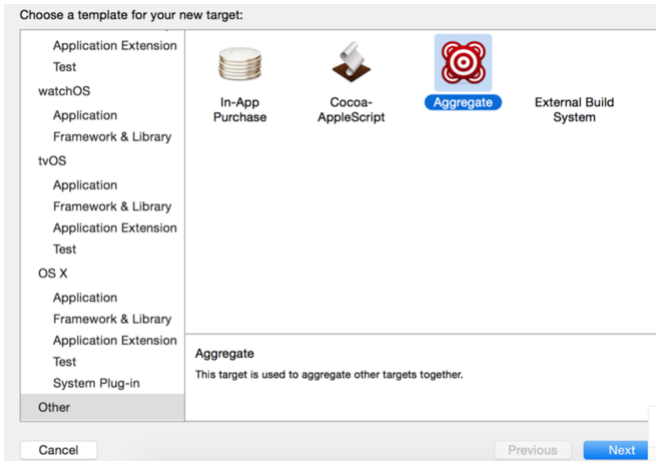
5.将修改后的framework拷贝出来保存，这就是我们最终制作的framework。

第二种方法

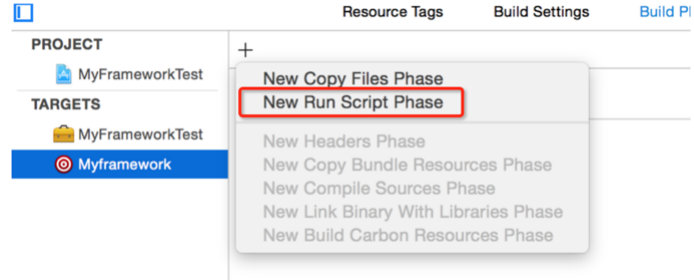
1、选中TARGETS下的工程，点击上方的Editor,选择Add Target创建一个Aggregate.



2、选择Other下的Aggregate，点击Next创建。



3、嵌入脚本。选中刚刚创建的Aggregate,然后选中右侧的Build Phases,点击左下方加号，选择New Run Script Phase



将这段脚本复制进去：

```
# Sets the target folders and the finalframework product.# 如果工程名称和Framework的Target名称不一样的话，要自定义FMKNAME# 例如: FMK_NAME = "MyFramework"FMK_NAME=${PROJECT_NAME}# Install dir will be the final output tothe framework.# The following line create it in the rootfolder of the current project.  INSTALL_DIR=${SRCROOT}/Products/$    {FMK_NAME}.framework# Working dir will be deleted after theframework creation.  WRK_DIR=build  DEVICE_DIR=${WRK_DIR}/Release-iphoneos/$                {FMK_NAME}.framework SIMULATOR_DIR=${WRK_DIR}/Release-  iphonesimulator/${FMK_NAME}.framework# -configuration ${CONFIGURATION}# Clean and Building both architectures.xcodebuild -configuration"Release"-target"${FMK_NAME}"-sdk iphoneos clean build  xcodebuild -configuration"Release"-target"${FMK_NAME}"-sdk iphonesimulator clean        build# Cleaning the oldest.if[-d"${INSTALL_DIR}"]thenrm -rf"${INSTALL_DIR}"fimkdir -p"${INSTALL_DIR}"cp -R"${DEVICE_DIR}/""${INSTALL_DIR}/"# Uses the Lipo Tool to merge both binaryfiles (i386 + armv6/armv7) into one    Universal final product.  lipo -create"${DEVICE_DIR}/${FMK_NAME}""${SIMULATOR_DIR}/${FMK_NAME}"-output"$                {INSTALL_DIR}/${FMK_NAME}"rm -r"${WRK_DIR}"open"${INSTALL_DIR}"
```

这里有一个误区，就是复制上边的这段脚本的时候，会在我们期望的效果里面多了几个回车，这几个回车是致命的，如果不删除回车，会报出如下的错误：

```
/Users/hjm/Library/Developer/Xcode/DerivedData/FrameworkDemo2-ffwipmbfosaqofdykzfsxhepejuz/Build/Intermediates/FrameworkDemo2.build/Debug-iphoneos/
  FrameworkDemo.build/Script-16E2437F1D50DEC500999D0A.sh: line 11: framework: command not found
/Users/hjm/Library/Developer/Xcode/DerivedData/FrameworkDemo2-ffwipmbfosaqofdykzfsxhepejuz/Build/Intermediates/FrameworkDemo2.build/Debug-iphoneos/
  FrameworkDemo.build/Script-16E2437F1D50DEC500999D0A.sh: line 15: Release-iphonesimulator/FrameworkDemo2.framework: No such file or directory
Build settings from command line:
    SDKROOT = iphoneos9.2
```

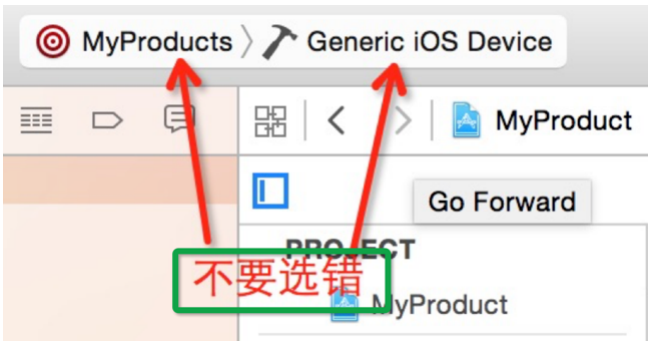
```
fatal error: lipo: can't open input file: build//FrameworkDemo2 (No such file or directory)
Command /bin/sh emitted errors but did not return a nonzero exit code to indicate failure
```

最后的格式如下图，尽量一个回车也不能错：

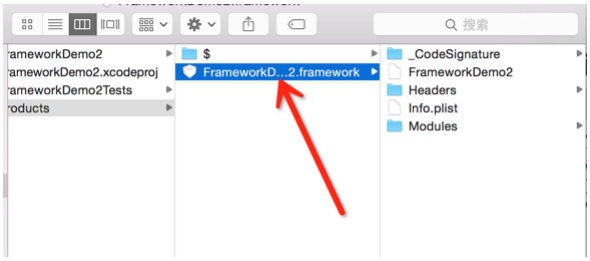
```
# Sets the target folders and the final framework product.
# 如果工程名称和Framework的Target名称不一样的话，要自定义FMKNAME
# 例如: FMK_NAME = "MyFramework"
FMK_NAME=${PROJECT_NAME}
# Install dir will be the final output to the framework.
# The following line create it in the root folder of the current project.
INSTALL_DIR=${SRCROOT}/Products/${FMK_NAME}.framework
# Working dir will bedeleted after the framework creation.
WRK_DIR=build
DEVICE_DIR=${WRK_DIR}/Release-iphoneos/${FMK_NAME}.framework
SIMULATOR_DIR=${WRK_DIR}/Release-iphonesimulator/${FMK_NAME}.framework
# -configuration ${CONFIGURATION}
# Clean and Building both architectures.
xcodebuild -configuration "Release" -target "${FMK_NAME}" -sdk iphoneos
    clean build
xcodebuild -configuration "Release" -target "${FMK_NAME}" -sdk
    iphonesimulator clean build
# Cleaning the oldest.
if [ -d "${INSTALL_DIR}" ]
then
rm -rf "${INSTALL_DIR}"
fi
mkdir -p "${INSTALL_DIR}"
cp -R "${DEVICE_DIR}/" "${INSTALL_DIR}/"
# Uses the Lipo Tool to merge both binary files (i386 + armv6/armv7) into
    one Universal final product.
lipo -create "${DEVICE_DIR}/${FMK_NAME}" "${SIMULATOR_DIR}/${FMK_NAME}" -
    output "${INSTALL_DIR}/${FMK_NAME}"
rm -r "${WRK_DIR}"
open "${INSTALL_DIR}"
```

通过第一种方法中**“把真机和模拟器的frameworkTest合并成一个”**的过程和上边的脚本语言比较，我们可以发现其实两者异路同归，两个方法里面同时用到了“lipo -create xxx”和“-output xxx”，不同的地方是第一种方法需要我们自己真机和模拟器分别变异一遍，而且需要我们把framework的路径拖进去，相比而言第二种方法比较简单。

4、编译。如图所示，command+B编译。这里Generic iOS Device的意思是“iOS通用设备”，大概就是说模拟器和真机都能用。

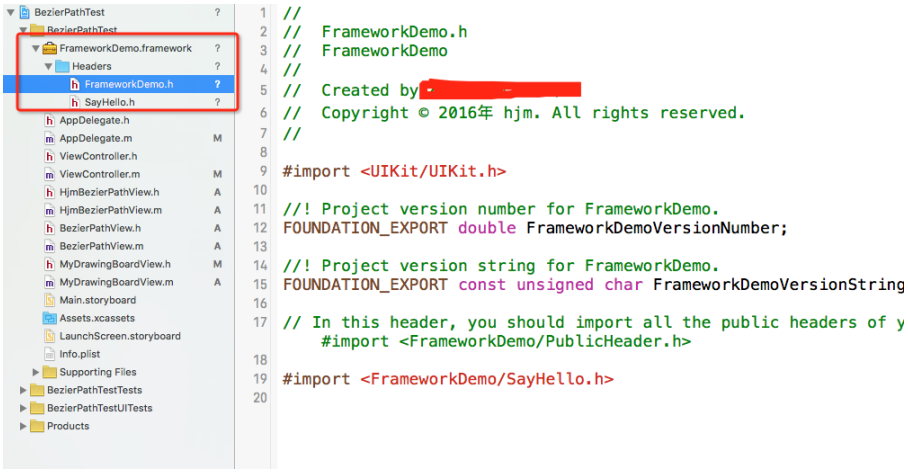


5、编译成功后会自动跳出一个finder，保存这个.framework，这就是我们需要的framework。



至此，两种打包framework的方法介绍完成！

最后就是用我们的Framework了，倒入另一个Xcode中，我们打开这个framework看看，发现只有Headers，里面有两个.h，其中一个是我们之前添加的FrameworkDemo.h文件，另一个就是我们的SayHello.h 。



然后引入头文件：

```
#import "ViewController.h"
#import "HjmBezierPathView.h"
#import "BezierPathView.h"
#import "MyDrawingBoardView.h"
#import <FrameworkDemo/FrameworkDemo.h>
#define WIDTH  [[UIScreen mainScreen] bounds].size.width
#define HEIGHT  [[UIScreen mainScreen] bounds].size.height
#define pi M_PI
#define   DEGREES_TO_RADIANS(degrees)  ((pi * degrees)/ 180)
```

由于我们测试的方法是实例方法，那么我们实例化一个实例对象，然后就可以让这个实例对象调取相应的方法了：

```
@implementation ViewController
{
    double add;
}

- (void)viewDidLoad {
    [super viewDidLoad];
    [self createBezierPathView];

    SayHello *say=[[SayHello alloc]init];
    [say sayHello];

    self.view.backgroundColor=[UIColor grayColor];
}
```


```
2016-08-01 15:52:01.366 BezierPathTest[2783:1260924] 你好，第一次见面，请多关照
```

至此，完成Framework的制作和使用。

总结

最后需要注意的是：

1、.h文件的外漏一定要保证是自己的想要外漏的。不想外漏的就别外漏了。

2、开始打包的时候，一定要在选中模拟器和选中真机上边分别编译一次， 我觉得之前在家里没有真机的时候编译的好像不对。

3、在终端上边合并的时候可能是error并生成一个.lipo文件，不要怕，大胆修改成同名的不挂后缀的同名文件。

4、调用的时候分清楚是类方法还是实例方法，方便调用。

5、在制作framework或者lib的时候，如果使用了category，则使用改FMWK的程序运行时会crash，此时需要在该工程中 other linker flags添加两个参数 -ObjC -all_load。（这点没有亲测）

6、带有图片资源的需要把图片打包成Bundle文件，和framework一起拷贝到相应的项目中。

7、公开的类中如果引用的private的类，打包以后对外会报错，找不到那个private的类，可以把那个private的.h放到（也没亲测）

8、**namespace 冲突**。静态库用了某第三方库，项目也用了同样的第三方库，在编译的时候就会有 duplicate symbol 错误，因为有两份同样的第三方库。解决办法就是把用到的第三方库加上自定义前缀，包括类名、delegate 协议、常量名，尤其需要注意 Category 的方法名要修改。

9、封装静态库的时候应尽量避免引入重量级第三方库，**多自己进行封装**。

10、一个静态库要**有自己独有的前缀**，所有类名、常量等都要加同样的前缀。

11、**真机+模拟器支持**。（和第2条意思一样）Xcode 默认只会用当前环境(真机或模拟器)生成静态库，这样的 SDK 不方便其他项目开发时调试。解决办法就是通过脚本生成一份通用库，build_universal_library.sh，via SO.

12、**文档**。静态库的方便是使用者直接拿你提供的方法来用，无需关注具体实现；不方便在于看不到实现，出现问题无法排查，因此需要把 SDK 的版本、更新历史、使用、FAQ 等写成文档，方便使用，也显得 SDK 比较正式规范。

13、图片等资源文件用 **bundle** 方式打包。一个简单制作 bundle 的方法：新建文件夹，重命名为 YourSDK.bundle，然后 Show Package Contents 打开，加入图片。使用图片的时候需要指明 bundle: [UIImage imageNamed:@"YourSDK.bundle/img.png"]。也可以用 Target 方式制作 bundle，比如 iOS Library With Resourceshttp://www.galloway.me.uk/tutorials/ios-library-with-resources/.

14、如果 SDK 有用到 **Category**，注意项目设置 Other Linker Flags 添加 -ObjC。（后边介绍了－ObjC的作用）

补充

编译过程：

从C代码到可执行文件经历的步骤是：源代码 > 预处理器 > 编译器 > 汇编器 > 机器码 > 链接器 > 可执行文件

在最后一步需要把.o文件和C语言运行库链接起来，这时候需要用到ld命令。源文件经过一系列处理以后，会生成对应的.obj文件，然后一个项目必然会有许多.obj文件，并且这些文件之间会有各种各样的联系，例如函数调用。链接器做的事就是把这些目标文件和所用的一些库链接在一起形成一个完整的可执行文件。Other linker flags设置的值实际上就是ld命令执行时后面所加的参数

下面逐个介绍3个常用参数：

－ObjC：加了这个参数后，链接器就会把静态库中所有的Objective-C类和分类都加载到最后的可执行文件中

－all_load：会让链接器把所有找到的目标文件都加载到可执行文件中，但是千万不要随便使用这个参数！假如你使用了不止一个静态库文件，然后又使用了这个参数，那么你很有可能会遇到ld: duplicate symbol错误，因为不同的库文件里面可能会有相同的目标文件，所以建议在遇到-ObjC失效的情况下使用-force_load参数。

-force_load：所做的事情跟-all_load其实是一样的，但是-force_load需要指定要进行全部加载的库文件的路径，这样的话，你就只是完全加载了一个库文件，不影响其余库文件的按需加载

后期会试着把贝塞尔画饼的demo封装成framework，另外可能会增加Bundle文件的生成方法。

参考自1、iOS－制作Framework（最新）

2、iOS－－创建你自己的Framework

最后，哪里不对的地方可以给我留言，我会及时改进的，谢谢大家。

作者：和珏猫
链接：http://www.jianshu.com/p/87dbf57cfe4a
來源：简书

我虐代码千百遍，代码待我如初恋！

分类：杂谈

IOS_Bowen
关注 - 13
粉丝 - 124
+加关注

0 0

« 上一篇：这样好用的ReactiveCocoa，根本停不下来
» 下一篇：Xcode下的中文乱码问题

posted on 2017-09-05 14:20 IOS_Bowen 阅读(3058) 评论(0) 编辑 收藏