

iOS开发之App间账号共享与SDK封装

2016-09-05 青玉伏案 CocoaChina

▲点击上方“CocoaChina”关注即可免费学习iOS开发

原文链接: <http://www.cnblogs.com/ludashi/p/5827156.html>

本文会封装一个登录用的SDK, 该登录SDK中包括登录、注册、忘记密码等功能, 当然该SDK中包括这些功能的UI、数据验证等业务逻辑、网络请求、数据存储等等。当然此处的登录SDK是一个简化版的, 真正的登录SDK比这个考虑的东西要多的多, 如果在加上多个App进行登录账号的共享的话, 那么考虑的东西就更为复杂了。

首先封装一个LoginSDK, 然后将该SDK植入到两个App中(一个暂且叫做“App One”, 另一个暂且称为“App Two”)。当App One登录成功后, 当你在打开App Two进行登录时, 我们封装的LoginSDK会从KeyChain中取出App One的账号进行登录。前提是这两个App设置了Keychain Share。

一、功能总述

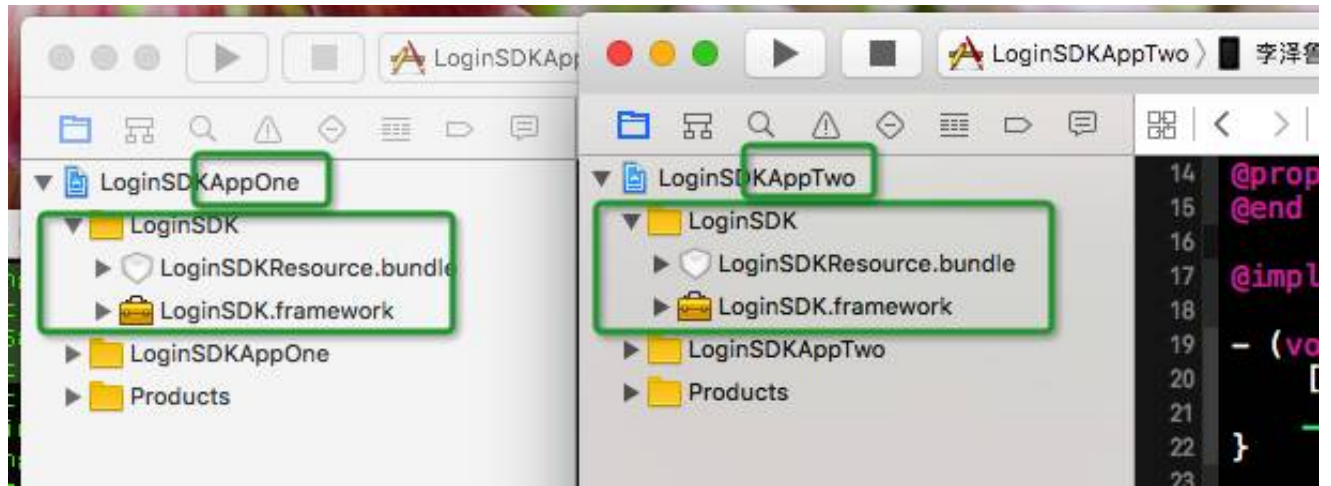
我们先来看一下我们最终要实现的效果。下图中所表述的就是我们今天博客中要做的事情, 下方的App One和App Two都植入了我们将要封装的LoginSDK, 两个App中都设置了Keychain Share。当App One通过我们的LoginSDK登录后, 在启动App Two时, 会去检索是否有账号以及在分享的Keychain中存储了, 如果有的话, 那么不会弹出“登录”界面, 直接进行隐式登录。当然上述这些工作都是在我们的LoginSDK中进行做的事情。



首次登录，进入到LoginSDK的登录页面



本部分算是本篇博客的一个综述吧，从下方截图中，我们能清楚的看到上述的两个App中都植入了我们接下来要封装的SDK。LoginSDK.framework就是我们封装的登录静态库，其中提供了用户所调用的API。



下方这个截图中的内容就是用户所调用LoginSDK的API。因为我们做的只是一个Demo，所以下方的API接口比较简单，如果你要和现实App中真正的需求和业务逻辑整合到一块，那么封装一个登录用的SDK是非常麻烦的。因为我考虑过把我们团队所开发的几个App中的登录模块封装成SDK，仔细考虑了一下，东西还是蛮多的。扯远了，不过今天这个Demo还是可以提供一个大体思路的。

下方API的对象是通过单例来获取的，如果是首次登录的话，就需要调用 `getLoginViewController` 这个方法来获取登录页面，并且这个函数需要提供一个Block参数，这个Block参数用来处理登录成功后的事件。而登录失败等事件就在我们SDK中自行处理了。

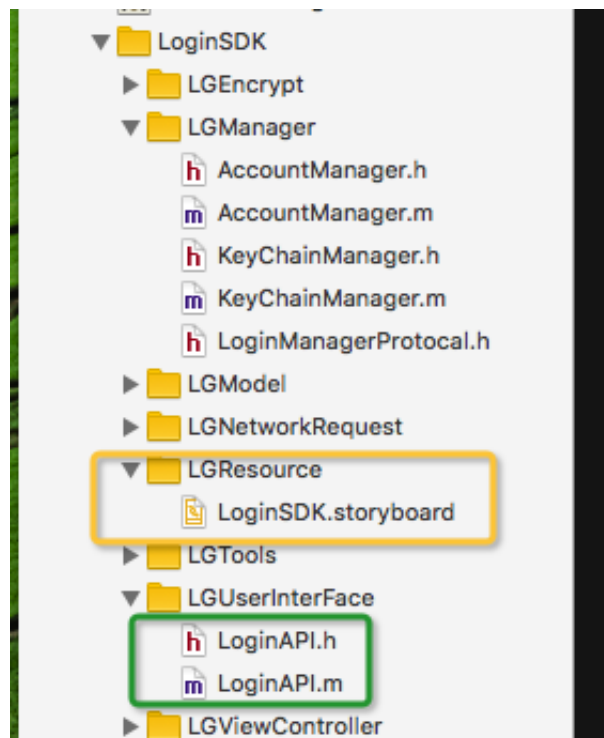
`checkHaveLogin` 方法是用来检查是否已经有账号登录过，该方法需要提供两个Block，一个是登录成功要执行的Block，一个是没有已登录账号时执行的Block。当执行该方法时，如果之前有账号登录过的话，就直接进行隐式登录，登录成功后执行 `loginSuccessBlock`。之前如果没有账号在此设备上登录就执行 `noAccountBlock`，来处理首次登录的事件。

```
4 //
5 // Created by Mr.LuDashi on 16/8/26.
6 // Copyright © 2016年 ZeluLi. All rights reserved.
7 //
8
9 #import <Foundation/Foundation.h>
10 #import "LoginManagerProtocal.h"
11 #import <UIKit/UIKit.h>
12
13 @interface LoginAPI : NSObject<LoginManagerProtocal>
14 @property (strong, nonatomic) NSString * token;
15 /**
16  * 获取账户管理的单例
17  *
18  * @return 返回账户管理的Manager
19  */
20 + (instancetype)shareManager;
21
22 /**
23  * 首次登录，获取登录页面
24  *
25  * @param loginBlock 登录成功后的Block
26  *
27  * @return 返回LoginViewController
28  */
29 - (UIViewController *)getLoginViewController: (LoginBlock)loginBlock;
30
31
32 /**
33  * 检查是否有账号登录过
34  *
35  * @param loginSuccessBlock 登录成功回调
36  * @param noAccountBlock 无账号登录
37  */
38 - (void)checkHaveLogin: (LoginBlock)loginSuccessBlock
39                      noAccountBlock: (NoAccountLoginBlock) noAccountBlock;
40
41 /**
42  * 注销
43  */
44 - (void)logout;
45
46 /**
47  * 判断是否已注销
48  *
49  * @return YES - 已注销, NO - 已登录
50  */
51 - (Boolean)isLogout;
52 @end
```

该部分先聊这么多，接下来会根据上述的知识点详细的展开。

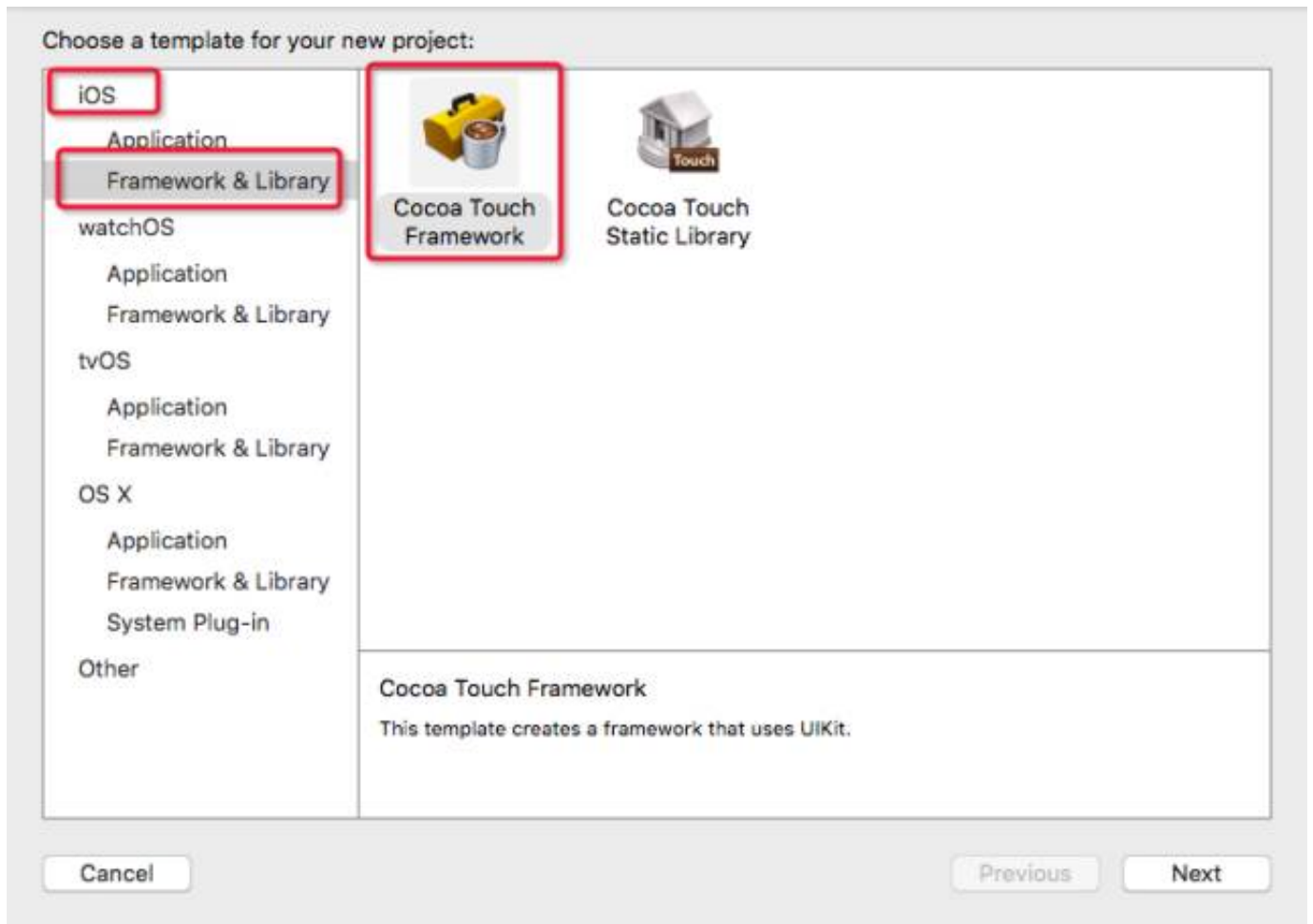
二、LoginSDK的封装

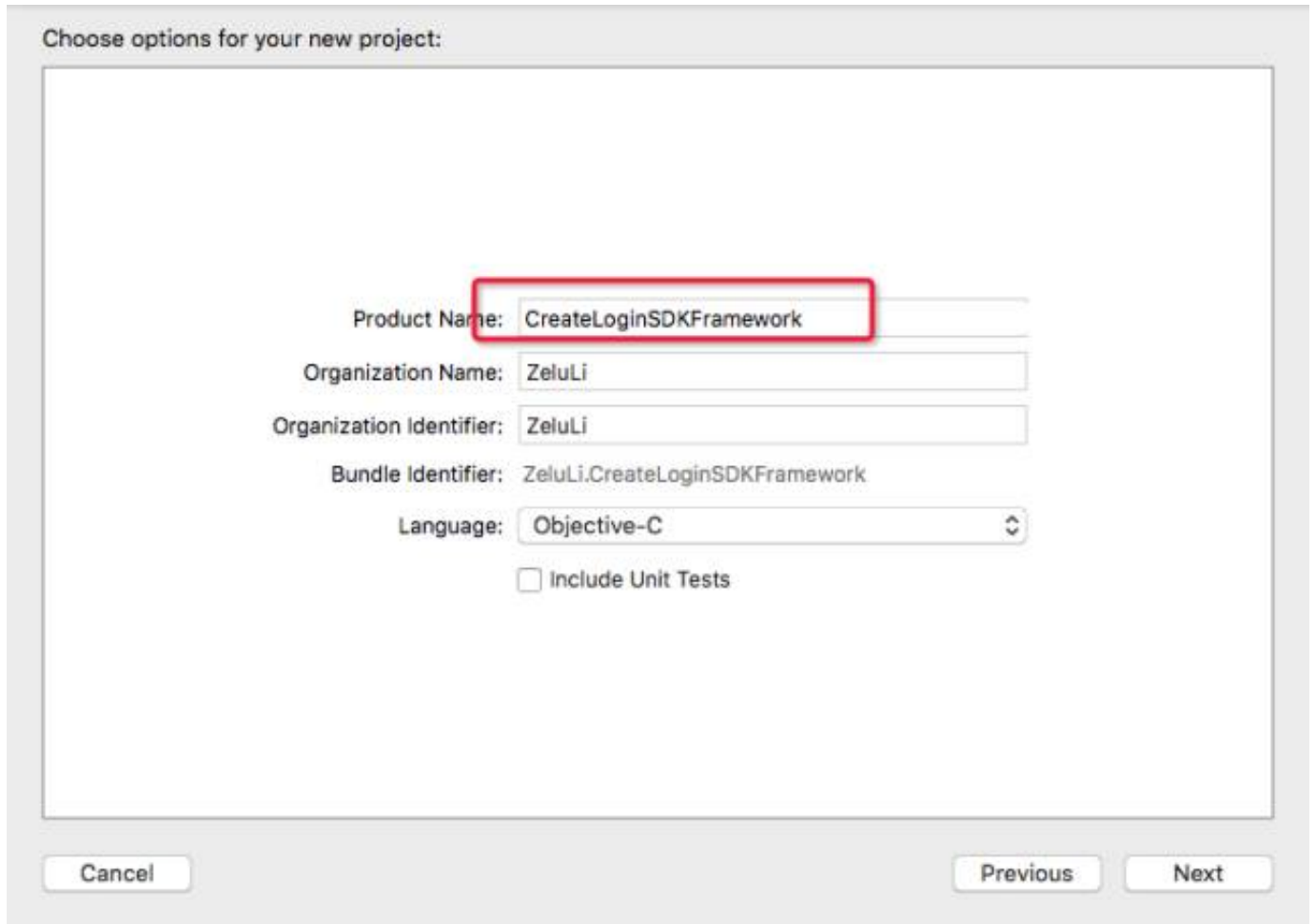
在封装LoginSDK之前呢，SDK的源代码以及所依赖的资源得准备好对吧。下方截图就是我们LoginSDK的源代码，下方绿框中的部分是留给用户使用的API，而黄框中的部分就是我们这个SDK所依赖的资源了，虽然此处只用一个Storyboard，我们还是有必要将该资源文件打包成Bundle文件提供给用户的。而其他源代码SDK的用户是看不到的。源码准备好，测试完毕后，接下来我们就要进行SDK的封装了。



1. 创建iOS Framework工程

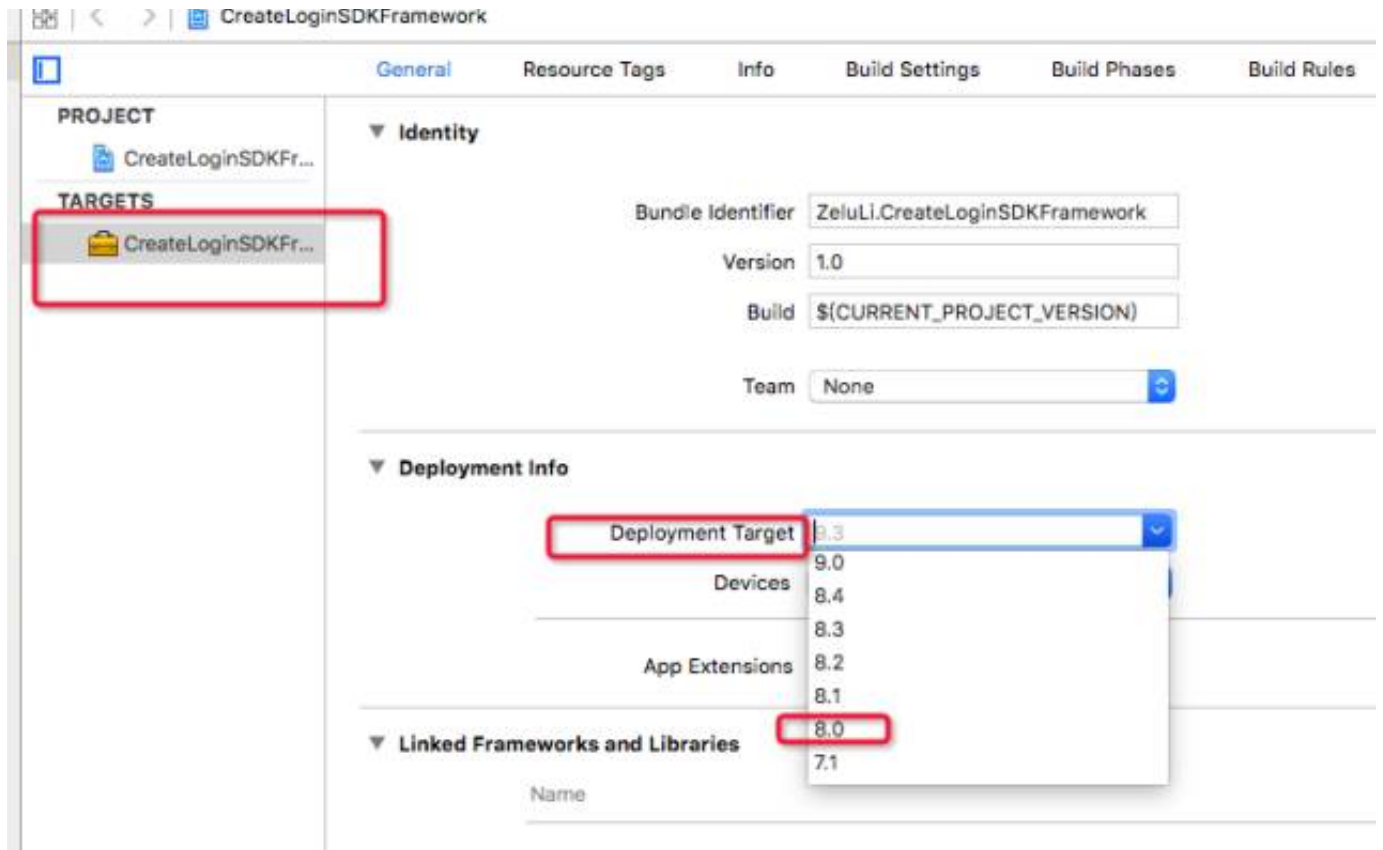
首先我们需要创建一个iOS的CocoaTouch工程，点击Next，输入我们Framework的名字即可。下方我们暂且将该Framework的名字命名为“CreateLoginSDKFramework”。如下所示：





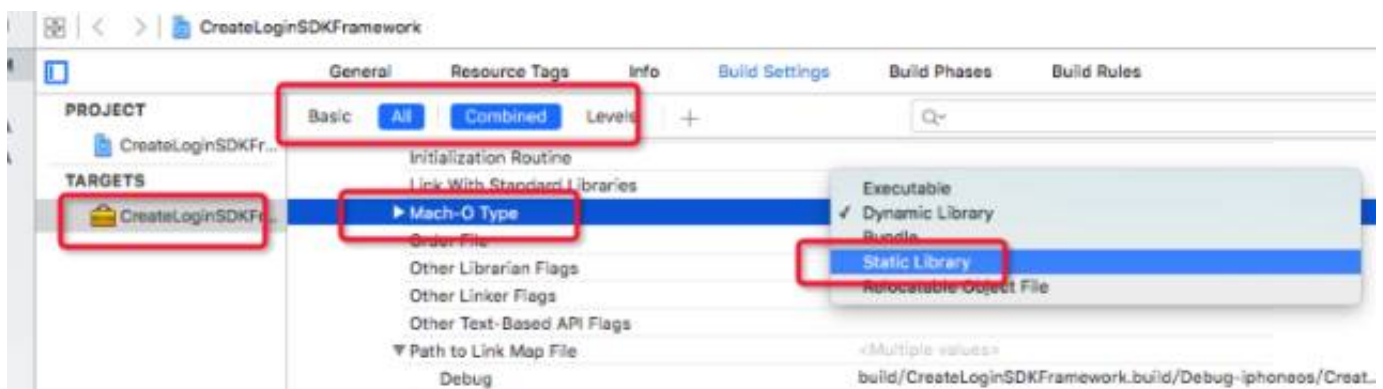
2. 设定兼容版本

创建完工程后，我们要选择“Deployment Target”，此处我们选择的是8.0。也就是说此处我们封装的SDK所支持的iOS系统版本是iOS8.0+。



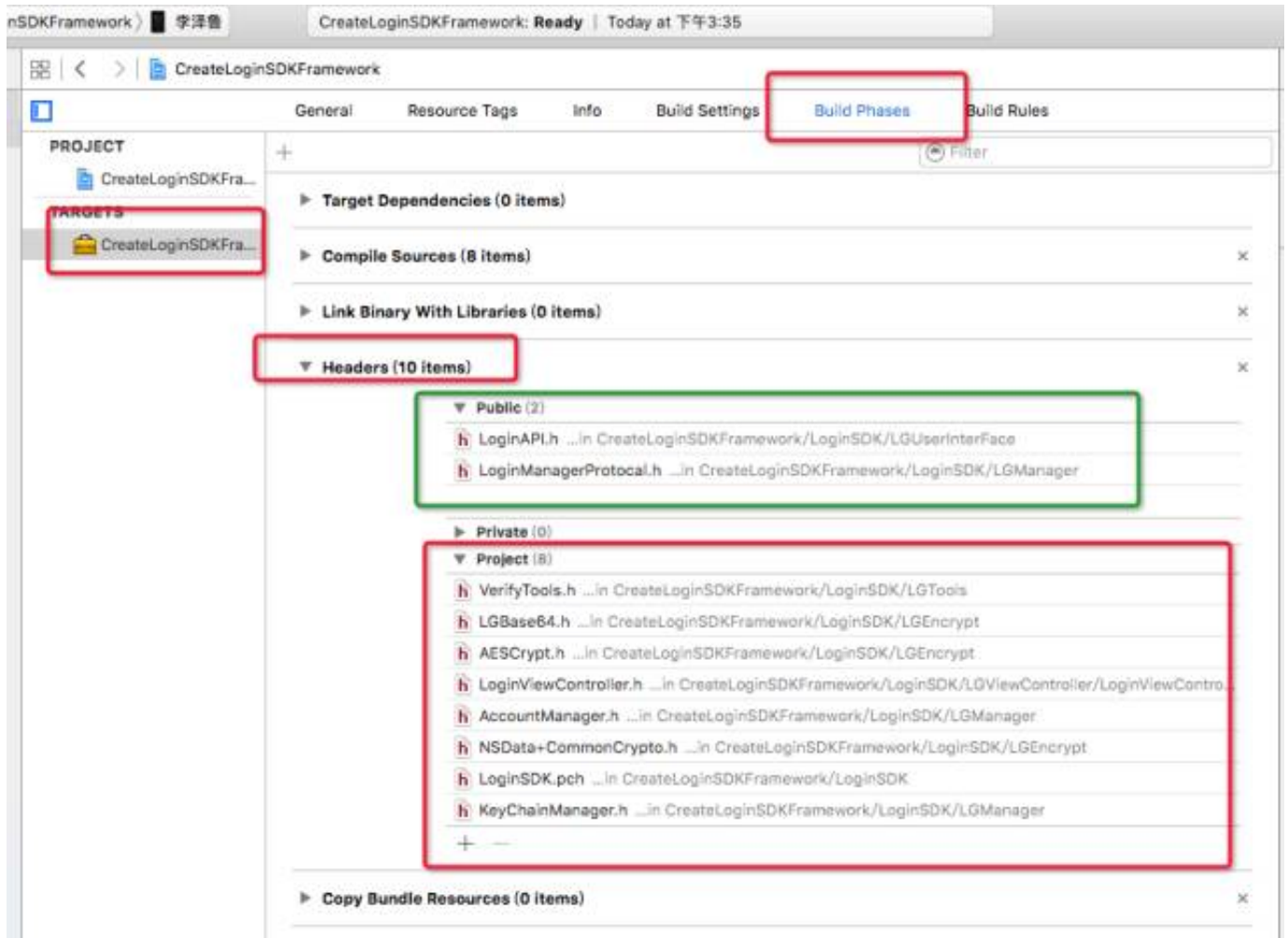
3.选择“静态库”

我们创建的framework默认是动态库，所以我们要讲Mach-O Type设置为静态库“Static Library”，如下所示。



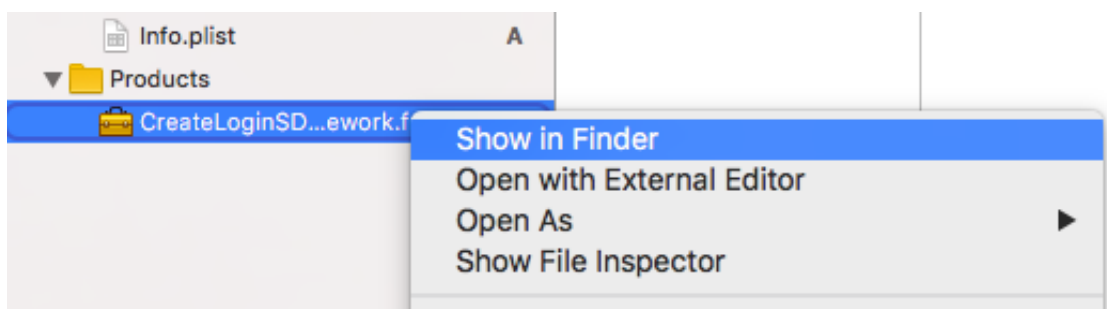
4.引入源代码并进行编译

配置好上述选项后，接下来我们就需要将我们事先准备好的SDK源代码引入到我们的Framework的工程中进行编译了，在编译之前我们要选择SDK用户可以看到的文件。下方截图中就是在Build Phases下的Headers中进行设置的。将用户可以看到的头文件房子Public中，用户看不到的放在Project中。如下所示。



5.编译

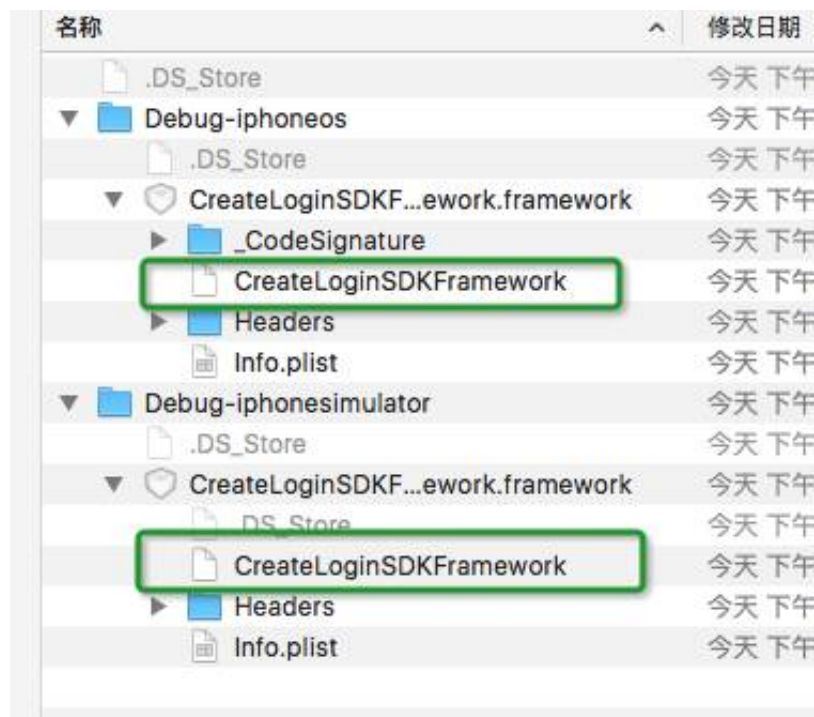
上述设置和配置完毕后，我们就要对我们的Framework工程进行编译了。先选择模拟器进行编译，然后选择真机进行编译。编译完后，在Products下会生成相应的Framework, 然后通过 Show in Finder进行查看即可。查看时，如果想看“模拟器”和“真机”的framework的话，在Show in finder后，需要前往上层文件夹查看。具体如下所示。



6.Framework的合并

因为在模拟器下编译会生成模拟器下使用的Framework，在真机下编译会生成真机使用的Framework。如果想我们生成的Framework既可以在真机下使用，也可以在模拟器下使用，那么我们需要将两个Framework进行合并。

下方截图中，这两个framework一个是真机生成的，另一个是模拟器生成的，我们做的事情就是将下方绿框中的两个文件进行合并。然后使用合并后的文件将下方的文件替换即可。替换后的framework就可以在模拟器和真机下进行使用了。



我们使用“lipo -create 模拟器framework路径 真机framework路径 -output 新的文件”命令将上述两个文件进行合并。下方就是合并上述两个文件的执行命令，执行完下方命令后会生成合并后的文件，将上述文件进行替换即可。经过上述步骤，我们的Framework至此就封装完毕了。

```
[bogon:MergeSDK lizelu$ lipo -create ./Debug-iphoneos/CreateLoginSDKFramework.framework/CreateLoginSDKFramework ./Debug-iphonesimulator/CreateLoginSDKFramework.framework/CreateLoginSDKFramework -output ./CreateLoginSDKFramework
```

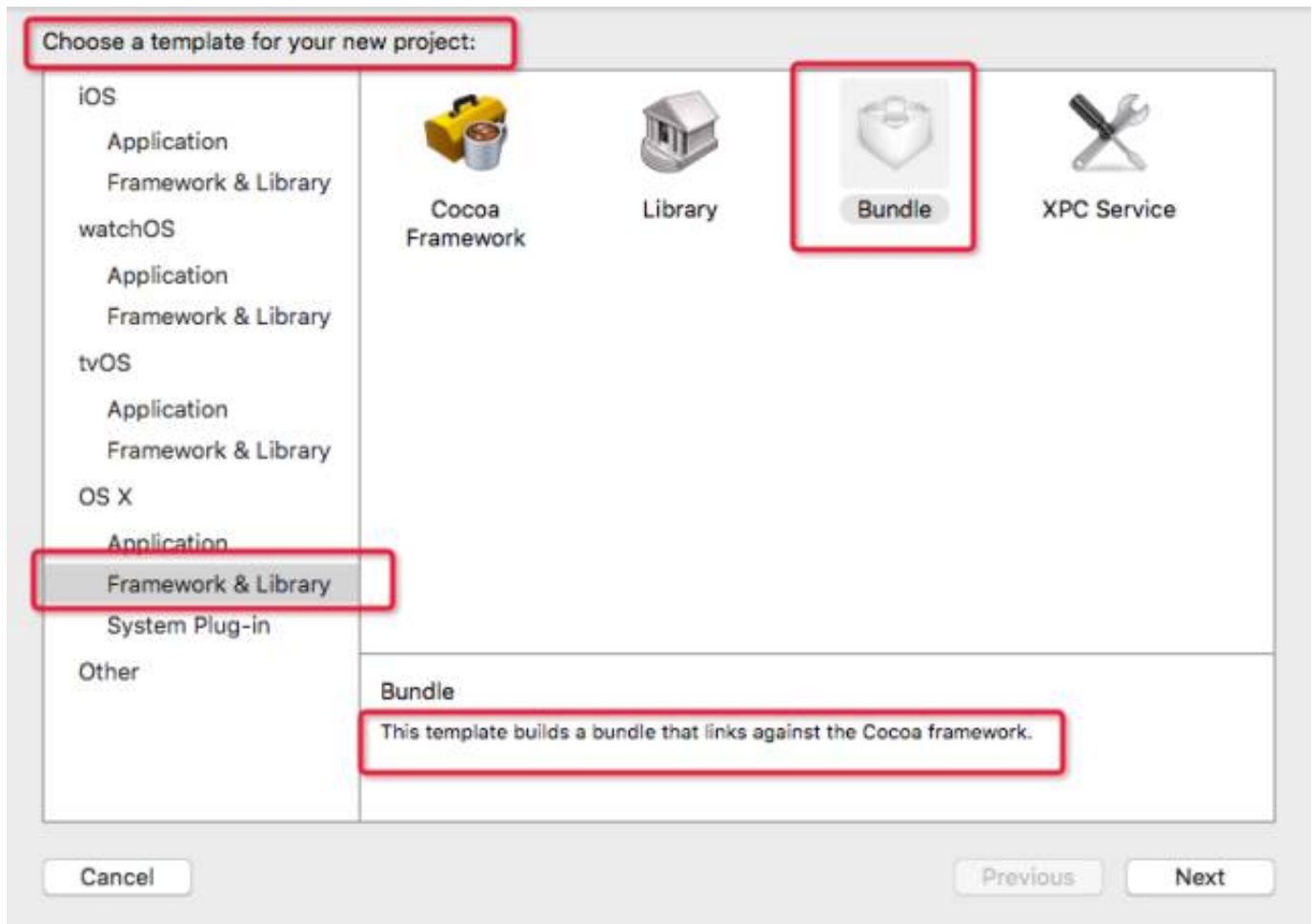
三、封装Bundle

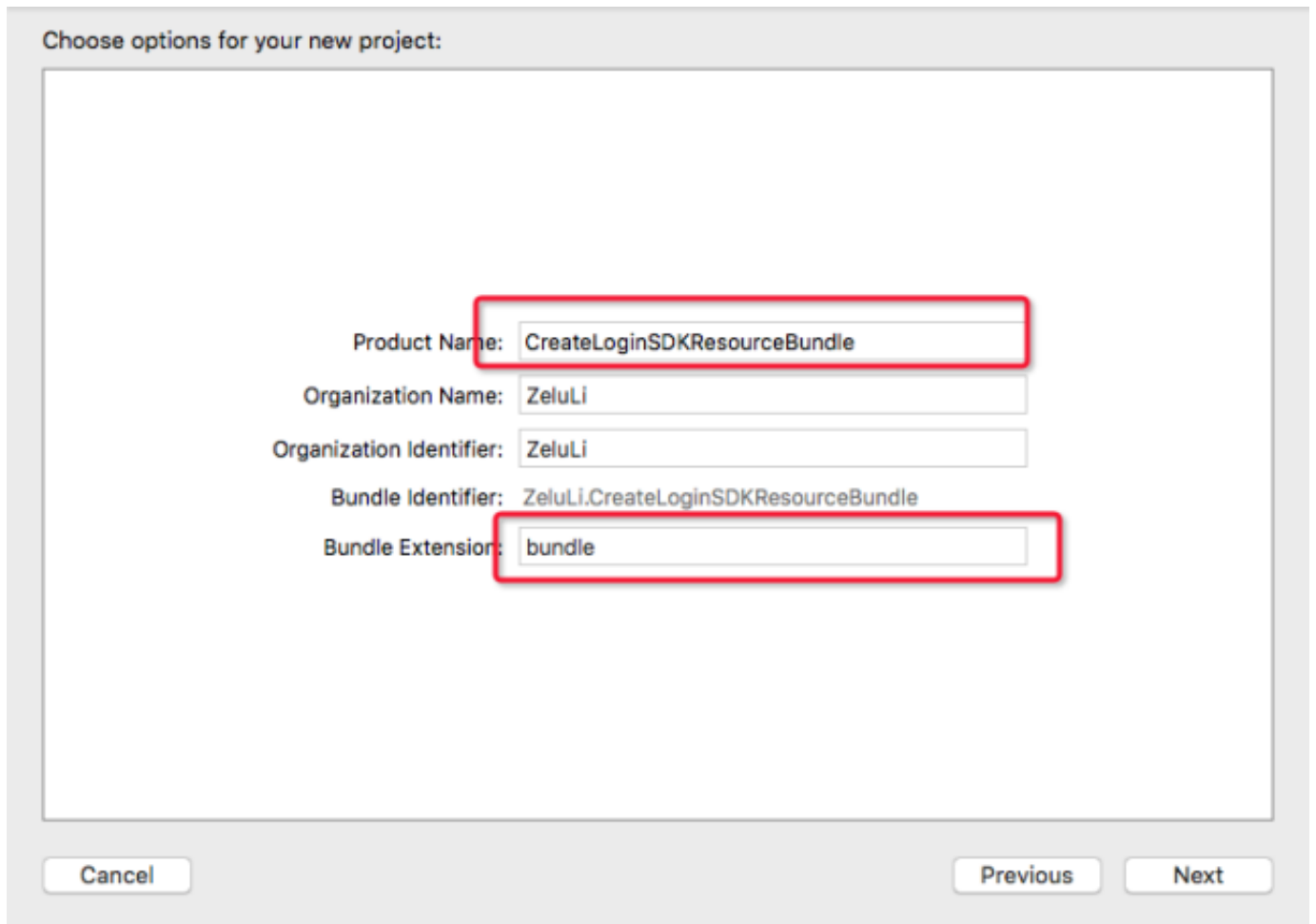
封装完Framework后，接下来我们要对Framework依赖的资源文件进行打包了。因为我们SDK中的界面是使用Storyboard做的，所以需要将Storyboard打包成Bundle资源文件与上述的Framework一起使用。如果我们SDK中需要一些图片资源的话，也可以进行一并打包。接下来

我们就要对资源文件进行打包。

1.Bundle工程的创建

首先我们像创建Framework工程一样创建一个Bundle工程，因为iOS工程下方没有Bundle类型的工程，所以我们需要在OS X -> Framework & Library -> Bundle下面来创建我们的Bundle工程。选择完后，输出我们的Bundle文件的名称即可，如下所示：





Choose options for your new project:

Product Name: CreateLoginSDKResourceBundle

Organization Name: ZeluLi

Organization Identifier: ZeluLi

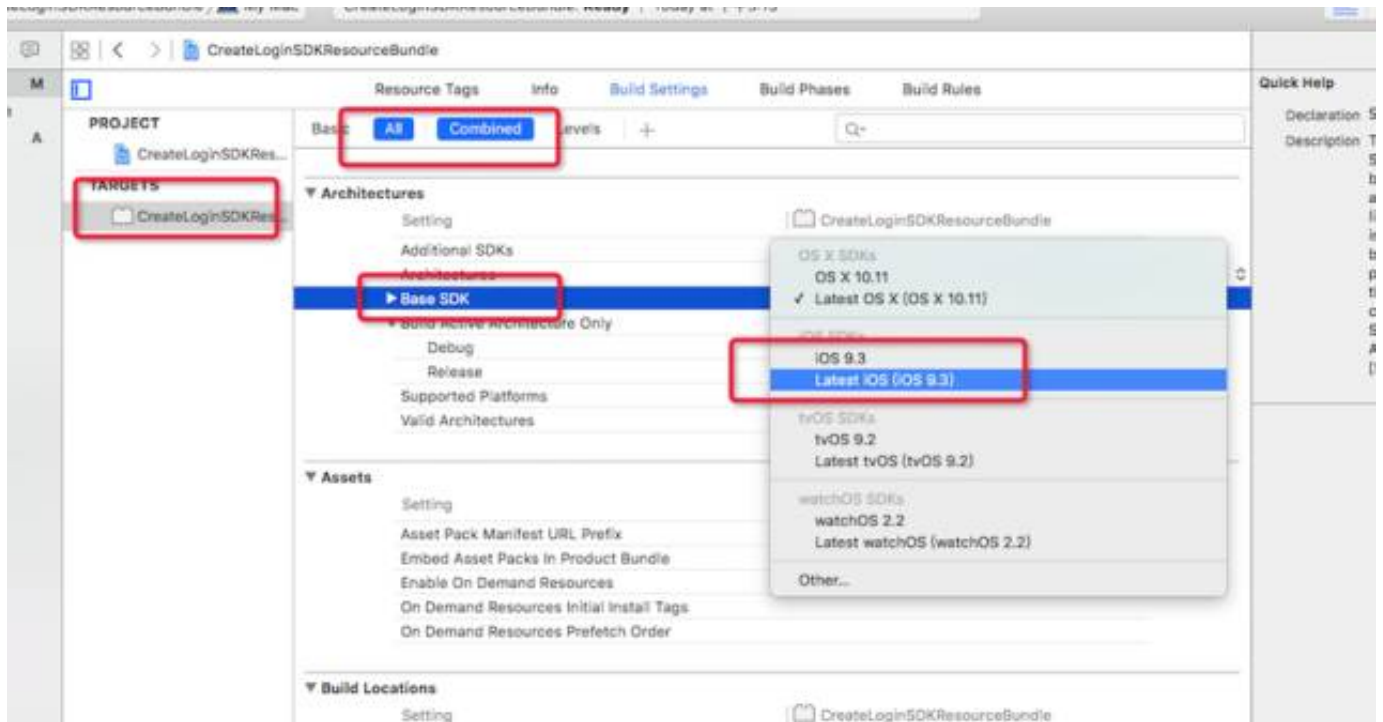
Bundle Identifier: ZeluLi.CreateLoginSDKResourceBundle

Bundle Extension: bundle

Cancel Previous Next

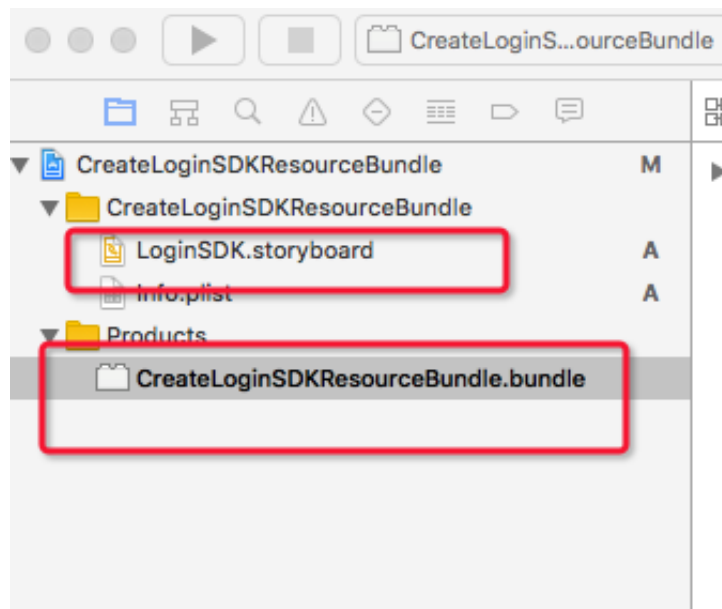
2. Bundle工程的配置

创建完Bundle工程后，我们要对其进行相应的配置。因为我们是选择OS X创建的Bundle，默认的Bundle是不能在iOS中使用的，所以我们得将Base SDK进行设置，选择相应的iOS版本即可，如下所示。选择完Base SDK后，我们还要像上面Framework的封装一样，设置一下要兼容的iOS版本（iOS Deployment Target），在此就不做过多赘述了。



3.引入资源，进行编译

进行上述配置完后，接下来就是引入资源文件进行编译了，下方引入的资源文件就是我们的 LoginSDK.storyboard。引入资源后，进行编译，编译后会在Products下面生成相应的Bundle资源文件，该文件就可以和我们的Framework进行使用了。



4.Bundle资源的加载

生成完Bundle资源文件后，我们在SDK的源代码中，要从Bundle资源文件中进行资源的加载。

下方代码就是加载相应Bundle的代码。通过下方的宏定义，就可以通过“Bundle”的名字来加载Bundle。下方的LOGIN_SDK_BUNDLE就是我们要使用的Bundle资源文件的对象。

```
#define LOGIN_SDK_BUNDLE_NAME    @"LoginSDKResource.bundle"
#define LOGIN_SDK_BUNDLE_PATH    [[[NSBundle mainBundle] resourcePath]
stringByAppendingPathComponent: LOGIN_SDK_BUNDLE_NAME]#define
LOGIN_SDK_BUNDLE                [NSBundle bundleWithPath: LOGIN_SDK_BUNDLE_PATH]
```

下方代码就是从上述Bundle对象中加载相应的Storyboard。与我们之前的代码不同，之前我们是从MainBundle中加载的Storyboard，而现在我们是从指定的Bundle中来加载Storyboard。具体代码如下所示。

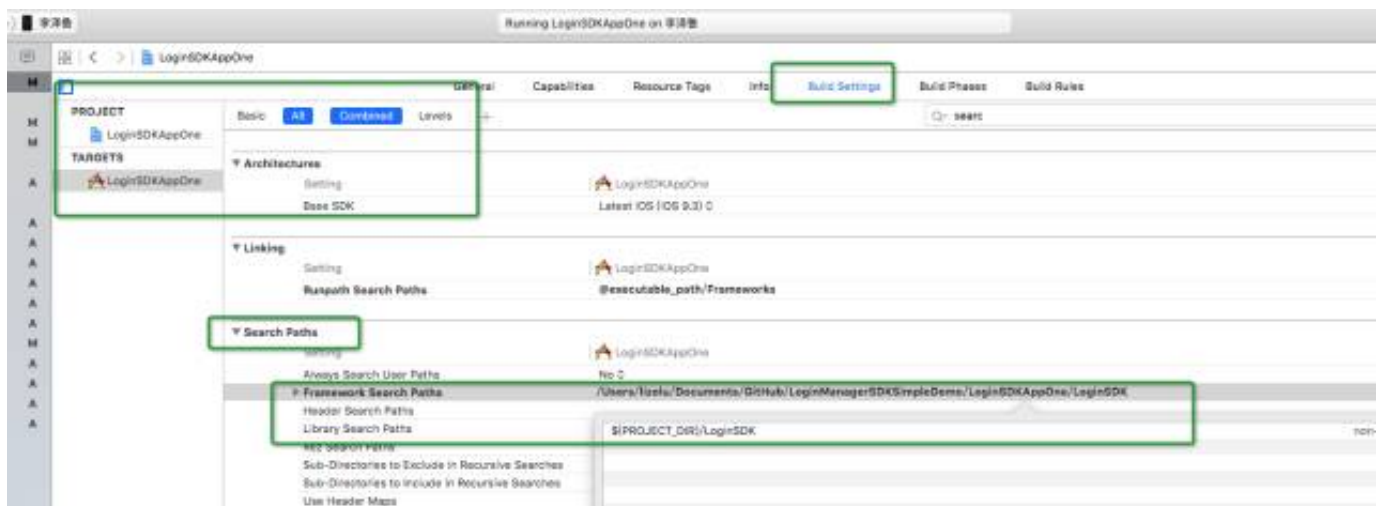
```
100 - (UIViewController *)getVCFromLoginSDKBundle {
101     NSLog(@"%g", [NSBundle allBundles]);
102     UIStoryboard *loginStoryboard = [UINavigationController storyboardWithName:LOGIN_STORYBOARD_NAME bundle:LOGIN_SDK_BUNDLE];
103     return [loginStoryboard instantiateViewControllerWithIdentifier:LOGIN_VIEWCONTROLLER_NAME];
104 }
105
```

四、SDK的引入

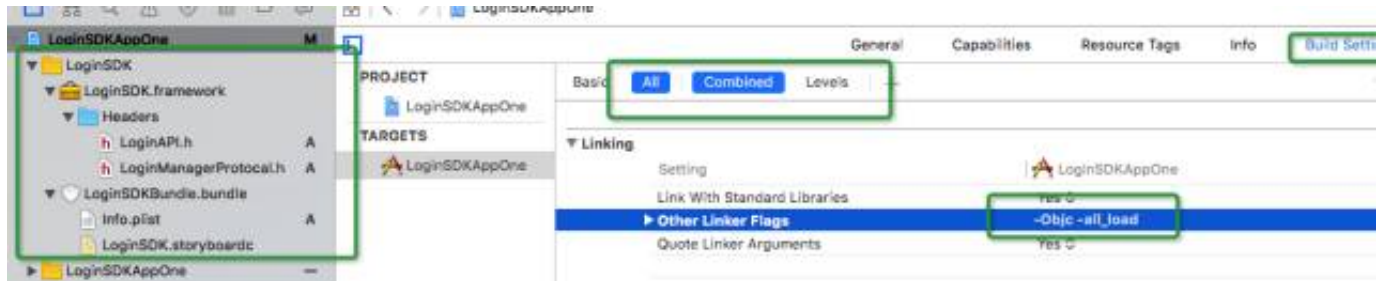
SDK已经依赖的资源文件封装完毕后，接下来就是在其他App中使用了。在第一部分中的App One和App Two都引入了上述我们封装的LoginSDK。引入SDK步骤也是比较简单的，这和引入友盟，个推，微信支付，支付宝等等SDK的步骤差不多。下方就是我们引入SDK的步骤。

1.导入SDK并进行相关配置

导入SDK到我们的App工程后，我们要对其进行相应的配置。首先我们要对Framework Search Paths进行配置，也就是说告诉编译器我们的第三方SDK所在的位置。下方这个配置项在引入SDK后就默认存在的，如果没有的话就进行配置即可。



配置完路径后，接下来我们要在Other Linker Flags添加上-Objc和-all_load选项。这两个选项在之前的博客中也不止一次的提到过。-Objc这个flag告诉链接器把库中定义的Objective-C类和Category都加载进来。而-all_load会强制链接器把目标文件都加载进来，即使没有objc代码。根据上面介绍的，下方即使不添加-Objc这个选项，下方的工程也是可以正常运行的。



2.SDK的使用

配置完毕后，接下来就是在我们App中使用该SDK了。下方代码就是我们上述LoginSDK的使用方式，首先获取单例，然后检查是否登录，登录成功后根据Block回调跳转到首页，如果未登录，就通过LoginAPI获取登录页面进行登录。具体如下所示。

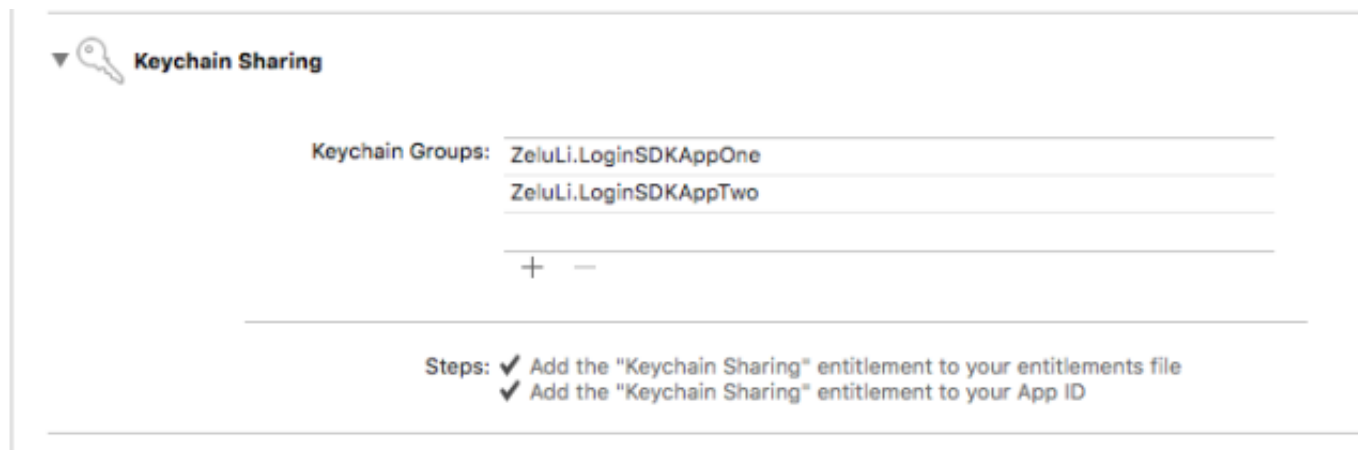
```

6 // Copyright © 2016年 ZeluLi. All rights reserved.
7 //
8
9 #import "GuidViewController.h"
10 #import "MainViewController.h"
11
12 @interface GuidViewController ()
13
14 @property (nonatomic, strong) LoginAPI *loginAPI;
15 @end
16
17 @implementation GuidViewController
18
19 - (void)viewDidLoad {
20     [super viewDidLoad];
21     _loginAPI = [LoginAPI sharedInstance]; //获取LoginAPI单例
22 }
23
24 - (IBAction)tapLogin: (id)sender {
25     [self checkHaveLogin:YES];
26 }
27
28 //检查是否已经登录
29 - (void)checkHaveLogin: (BOOL)isTapButton {
30     if (_loginAPI != nil) {
31         weak typeof(self) weak self = self;
32         [_loginAPI checkHaveLogin:^(NSString *token) {
33             [weak_self presentMainViewControllerWithText:token]; //二次登录, 成功后直接进入首页
34         } noAccountBlock:^(
35             if (isTapButton) {
36                 [weak_self presentLoginViewController]; //首次登录, 获取登录页面, 进行登录
37             }
38         )];
39     }
40 }
41
42 //通过loginAPI获取登录页面, 并对登录成功后的事件进行处理
43 - (void)presentLoginViewController {
44     weak typeof(self) weak self = self;
45     UIViewController *vc = [_loginAPI getLoginViewController:^(NSString *token) {
46         [weak_self presentMainViewControllerWithText:token];
47     }];
48     [self presentViewController:vc animated:YES completion:^(
49 }
50

```

五、Keychain共享

关于Keychain共享的东西，我们可以看一下上一篇博客的介绍《iOS逆向工程之KeyChain与Snoop-it》。而在本篇博客中，是对keychain共享的应用，在植入上述LoginSDK后，如果想多个App间进行账号共享的话，要在相应的App上添加Keychain Share的标示了。下方截图就是我们第一部分那两个App中所设置的Keychain共享的配置项了。具体如下所示。



经过上面的所有步骤，我们封装了一个简单的LoginSDK, 并在多个App中进行植入，并且进行了账号共享。

github分享地址：<https://github.com/lizelu/LoginManagerSDKSimpleDemo>



最近发现一个好玩的测试，互联网从业者都应该来试下，我得了 85 分，看看你能不能超过我！长按图片识别二维码或者点击阅读原文就可以参与。

微信号：[CocoaChinabbs](#)



**▲长按二维码“识别”关注即可免费学习 iOS 开发
月薪十万、出任CEO、赢娶白富美、走上人生巅峰不是梦**

商务合作QQ: 2408167315

投稿邮箱: support@cocoachina.com