本文章GitHub链接:https://github.com/3wz/ios-native-integration-unity
本文章适用于iOS采用Objective-C编写的应用, 如果是Swift编写可以直接参照GitHub链接: https://github.com/blitzagency/ios-unity5

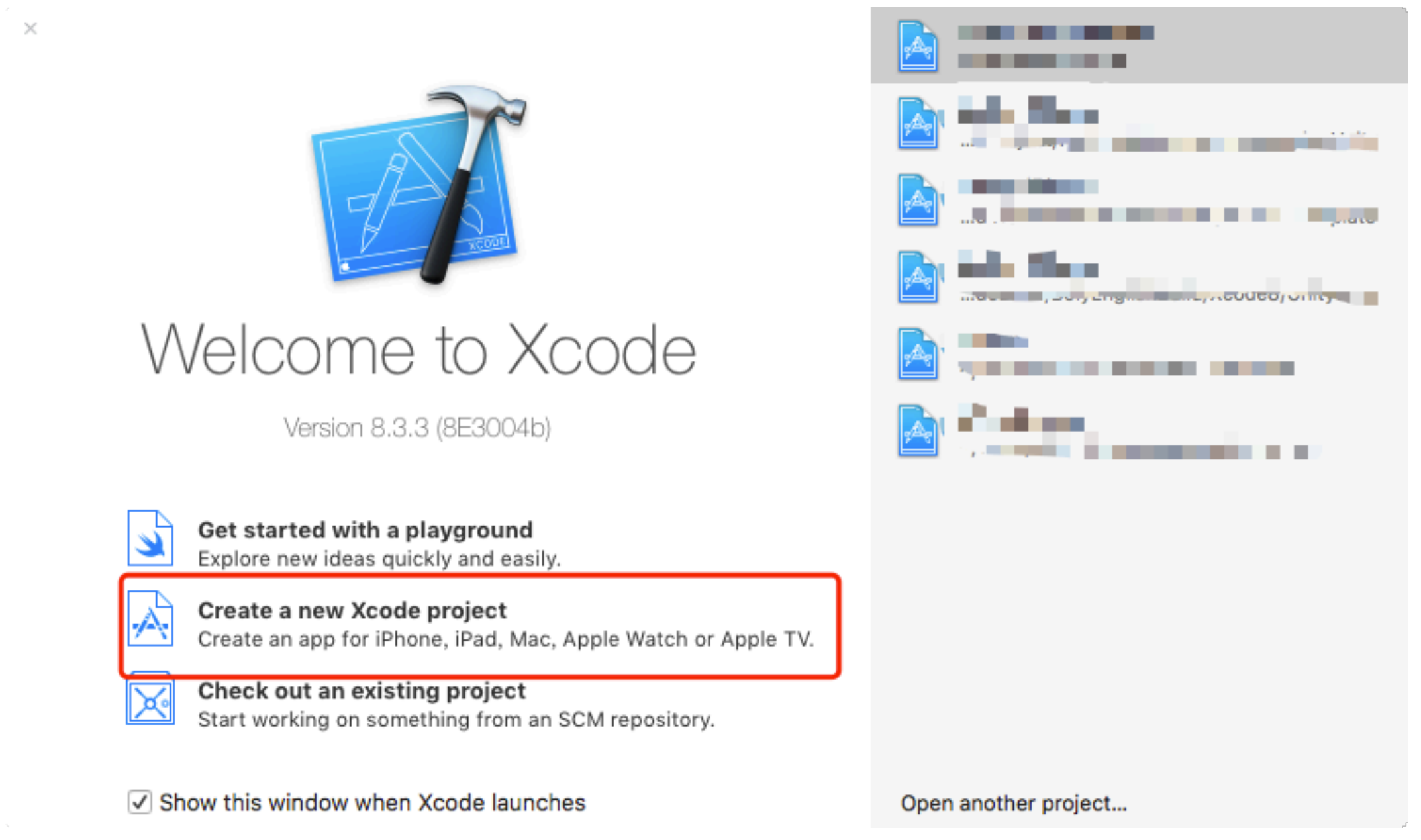
参考链接：
1. hthttps://the-nerd.be/2015/08/20/a-better-way-to-integrate-unity3d-within-a-native-ios-application/
2. https://github.com/blitzagency/ios-unity5
3. http://www.jianshu.com/p/deeacf3458fd

软件环境:
1. Xcode 8.3.3版本
2. Unity5.4.5f1版本. (最新的2017.1.0f3也试过, 但是需要在Other C Flag下增加 -DRUNTIME_IL2CPP=1, 否则会导致启动运行Crash)

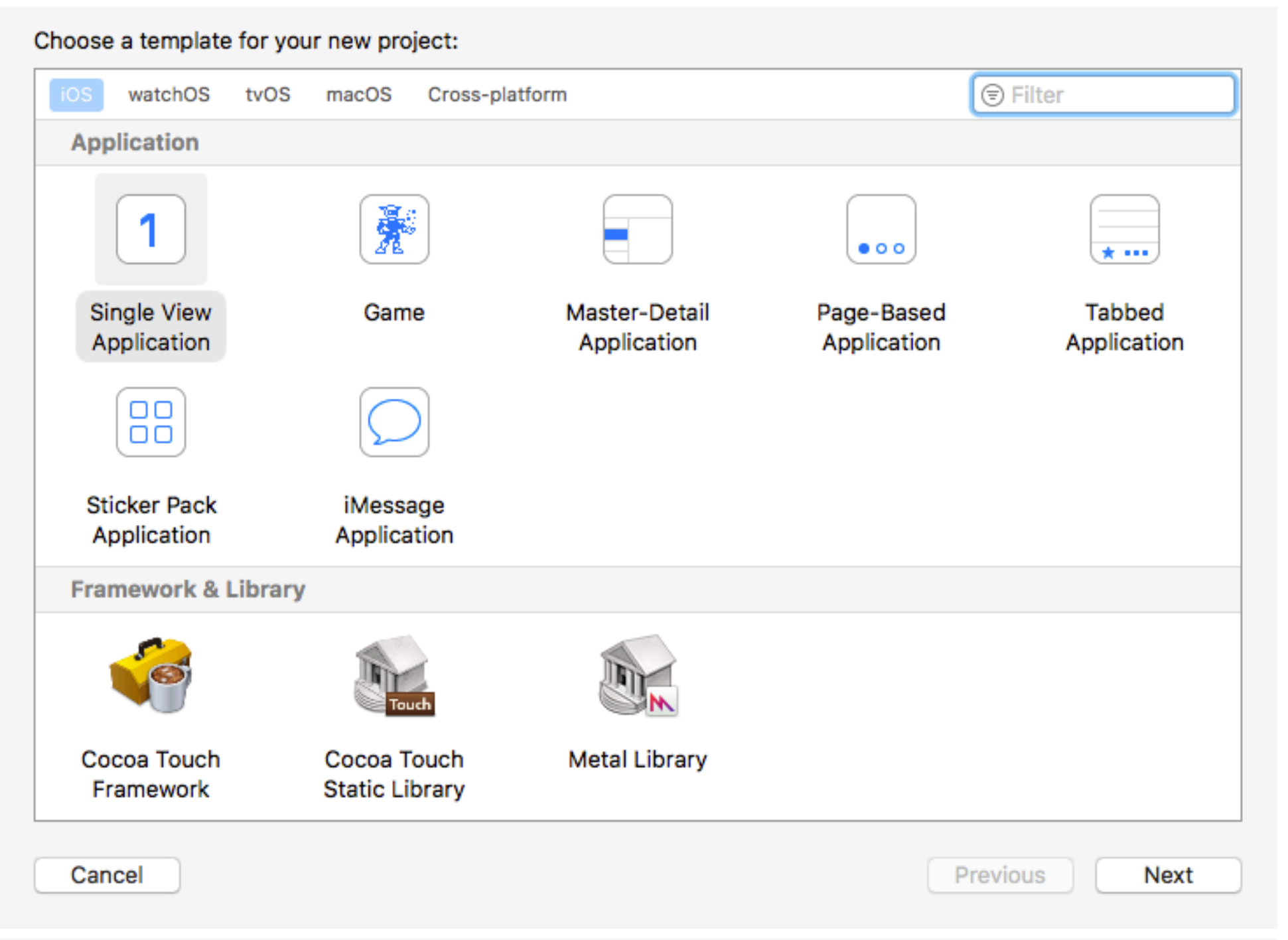
整体步骤:
1. 用Xcode创建原生iOS应用
2. 添加Unity.xcconfig文件到工程, 然后在Project中使用该配置文件
3. 在项目Target中Build Phases中添加run script, 并贴上相应代码
4. 导入Unity Build的Xcode工程
5. 重命名原生项目内的main.m文件后缀为main.mm. 切记
6. 在原生应用的AppDelegate中封装UnityAppController
7. 替换UnityAppController.h中的`GetAppController`方法
8. 在原生应用的一个ViewController中添加打开和返回Unity的界面测试方法
9. 当所有都配置完毕后, 如何更新原生工程内的Unity文件.

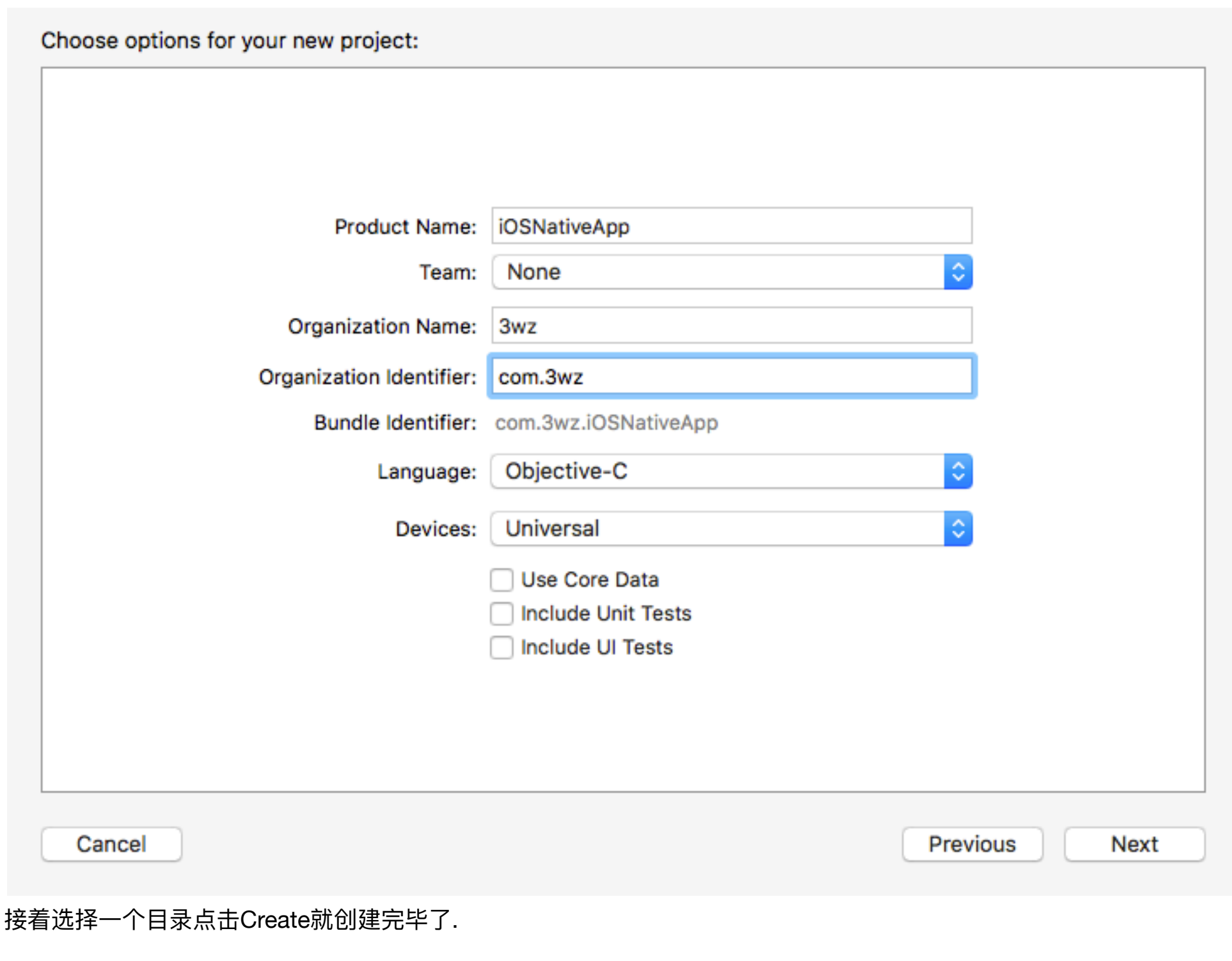
开始详细步骤：

利用Xcode创建一个原生测试应用
1.打开Xcode软件选择Create a new Xcode project



2.打开界面后, 默认选择第一个Single View Application即可, 点击Next下一步



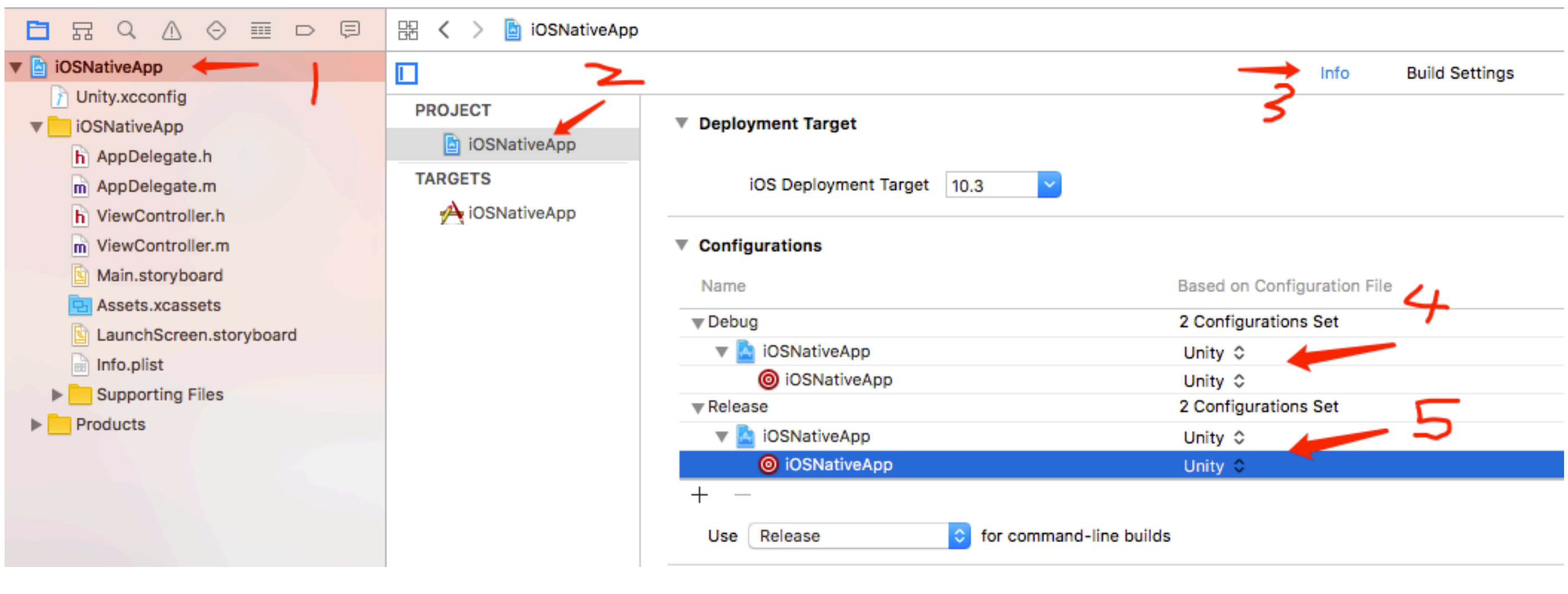


Choose options for your new project:

Product Name: iOSNativeApp
Team: None
Organization Name: 3wz
Organization Identifier: com.3wz
Bundle Identifier: com.3wz.iOSNativeApp
Language: Objective-C
Devices: Universal
☐ Use Core Data
☐ Include Unit Tests
☐ Include UI Tests

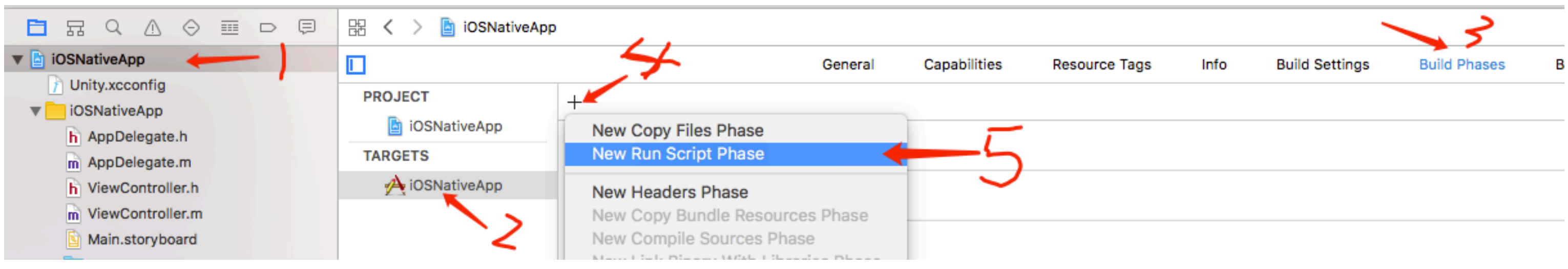
接着选择一个目录点击Create就创建完毕了.

2.拖拽Unity.xcconfig文件到原生Xcode工程中, 文件下载地址: https://github.com/3wz/ios-native-integration-unity
拖动到任何目录都行, 然后根据图片中步骤设置启用这个配置文件:



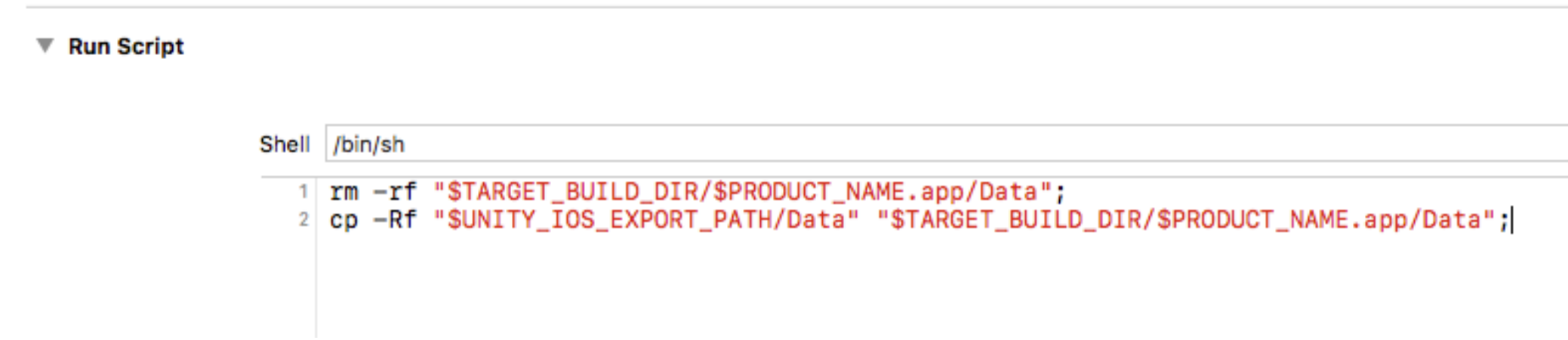
配置文件内容如下可根据自己情况调整, 调整部分已经在配置文件注释中写明.

3.添加run Script脚本, 按照如图所示的步骤添加.



添加后选择run script, 并复制以下内容进去:
rm -rf "$TARGET_BUILD_DIR/$PRODUCT_NAME.app/Data";
cp -Rf "$UNITY_IOS_EXPORT_PATH/Data" "$TARGET_BUILD_DIR/$PRODUCT_NAME.app/Data";

如图所示:

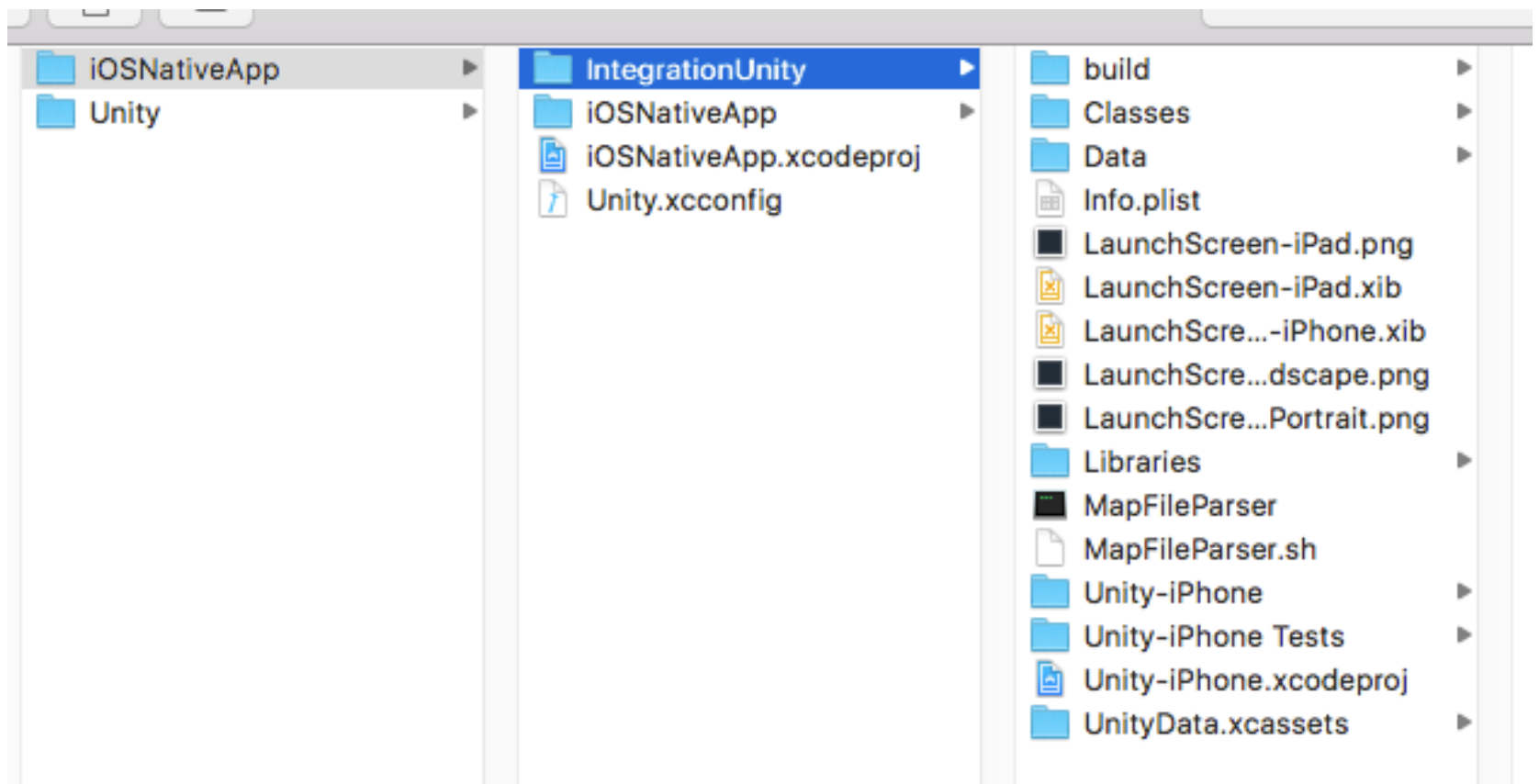


Run Script

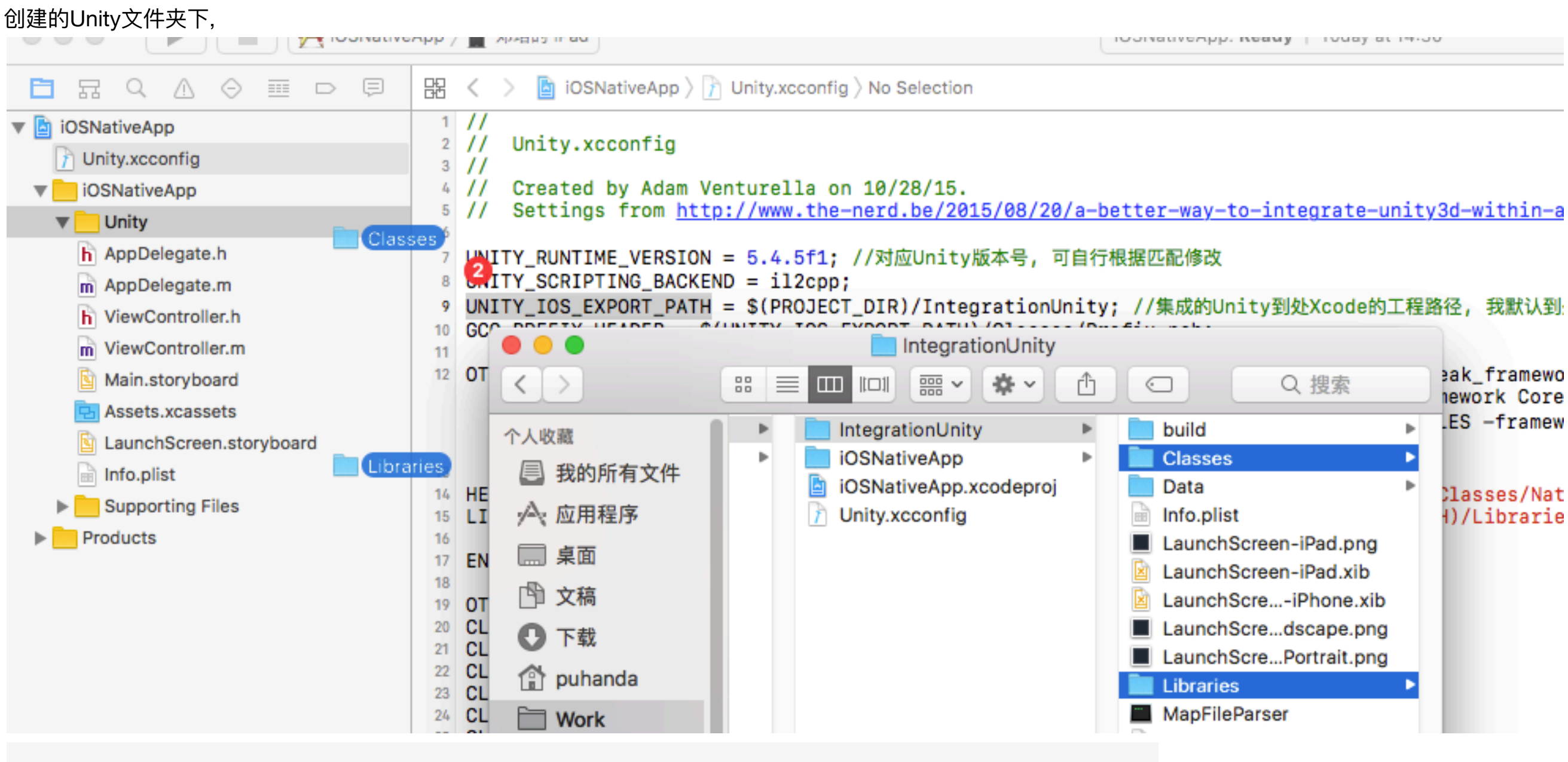
Shell /bin/sh

```
rm -rf "$TARGET_BUILD_DIR/$PRODUCT_NAME.app/Data";
cp -Rf "$UNITY_IOS_EXPORT_PATH/Data" "$TARGET_BUILD_DIR/$PRODUCT_NAME.app/Data";
```

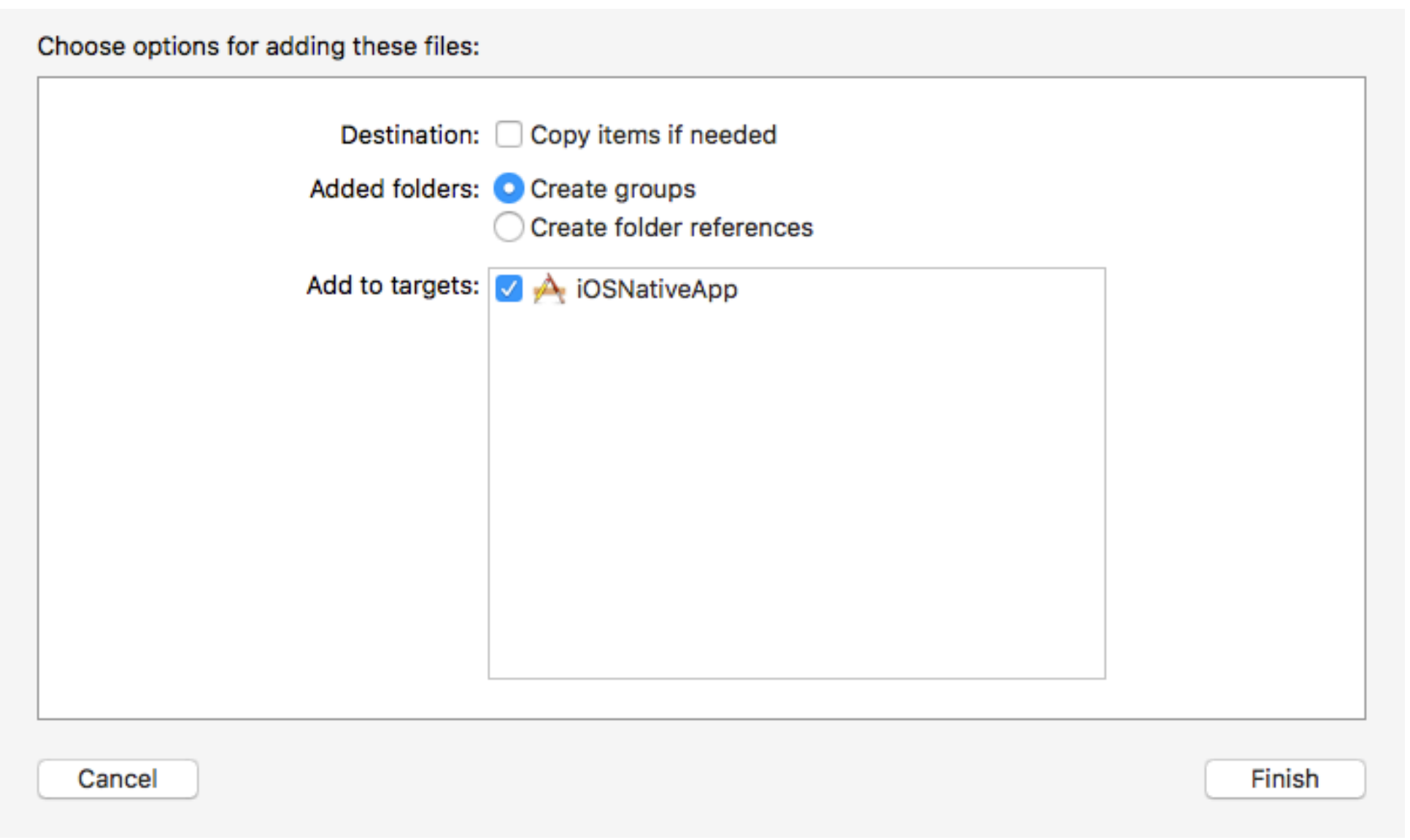
4.接下来就是准备好Unity导出的Xcode工程, 具体如何创建一个Unity工程并且导出, 我就不具体细说了.
导出的工程文件夹名字可默认为IntegrationUnity, 然后放入原生工程根目录, 如图所示:



接着在原生Xcode工程中创建一个名为Unity文件夹, 然后同时选择刚才的Unity导出的Xcode工程目录中的Classed和Libraries文件夹, 拖拽到刚才创建的Unity文件夹下,



```
//
//  Unity.xcconfig
//
//  Created by Adam Venturella on 10/28/15.
//  Settings from http://www.the-nerd.be/2015/08/20/a-better-way-to-integrate-unity3d-within-a
UNITY_RUNTIME_VERSION = 5.4.5f1; //对应Unity版本号, 可自行根据匹配修改
UNITY_SCRIPTING_BACKEND = il2cpp;
UNITY_IOS_EXPORT_PATH = $(PROJECT_DIR)/IntegrationUnity; //集成的Unity到处Xcode的工程路径, 我默认到
```

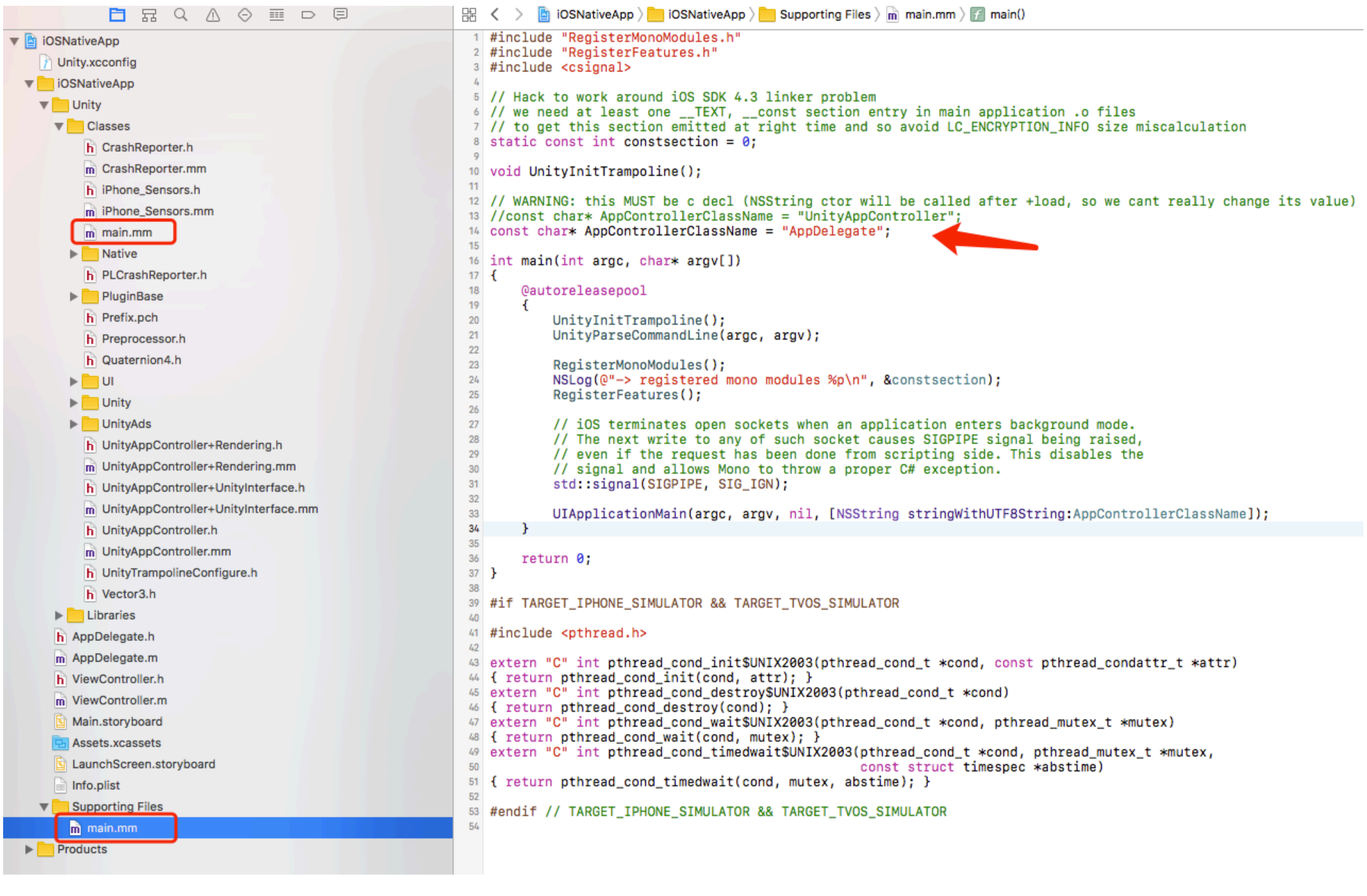


Choose options for adding these files:

Destination: ☐ Copy items if needed
Added folders: ● Create groups
○ Create folder references
Add to targets: ☑ iOSNativeApp

如图所示, Destination不需要勾选, 然后选择Create groups, 然后点击Finish后慢慢等待导入完毕, 文件比较多, 需要耐心等待一段时间.
参照的其他教程 此时这步骤之后还需要清理Classes下的Native中的.h文件, 和Librares下的libil2cpp整个文件夹, 但其实不需要, 没什么区别.

5.修改原生应用中的main.m后缀为main.mm, 然后复制Classes下的main.mm所有的内容, 到原生的main中, 并根据如图所示修改, 然后就可以把Classes下的main.mm文件给移除掉了.



```
#include "RegisterMonoModules.h"
#include "RegisterFeatures.h"
#include <csignal>

// Hack to work around iOS SDK 4.3 linker problem
// we need at least one __TEXT, __const section entry in main application .o files
// to get this section emitted at right time and so avoid LC_ENCRYPTION_INFO size miscalculation
static const int constsection = 0;

void UnityInitTrampoline();

// WARNING: this MUST be c decl (NSString ctor will be called after +load, so we cant really change its value)
//const char* AppControllerClassName = "UnityAppController";
const char* AppControllerClassName = "AppDelegate";

int main(int argc, char* argv[])
{
    @autoreleasepool
    {
        UnityInitTrampoline();
        UnityParseCommandLine(argc, argv);

        RegisterMonoModules();
        NSLog(@"-> registered mono modules %p\n", &constsection);
        RegisterFeatures();

        // iOS terminates open sockets when an application enters background mode.
        // The next write to any of such socket causes SIGPIPE signal being raised,
        // even if the request has been done from scripting side. This disables the
        // signal and allows Mono to throw a proper C# exception.
        std::signal(SIGPIPE, SIG_IGN);

        UIApplicationMain(argc, argv, nil, [NSString stringWithUTF8String:AppControllerClassName]);
    }

    return 0;
}

#if TARGET_IPHONE_SIMULATOR && TARGET_TVOS_SIMULATOR

#include <pthread.h>

extern "C" int pthread_cond_init$UNIX2003(pthread_cond_t *cond, const pthread_condattr_t *attr)
{ return pthread_cond_init(cond, attr); }
extern "C" int pthread_cond_destroy$UNIX2003(pthread_cond_t *cond)
{ return pthread_cond_destroy(cond); }
extern "C" int pthread_cond_wait$UNIX2003(pthread_cond_t *cond, pthread_mutex_t *mutex)
{ return pthread_cond_wait(cond, mutex); }
extern "C" int pthread_cond_timedwait$UNIX2003(pthread_cond_t *cond, pthread_mutex_t *mutex,
                                               const struct timespec *abstime)
{ return pthread_cond_timedwait(cond, mutex, abstime); }

#endif // TARGET_IPHONE_SIMULATOR && TARGET_TVOS_SIMULATOR
```

6.接着开始在APPDelegate中封装Unity的UnityAppController, 因为本身UnityAppConroller就是继承自`UIApplicationDelegate`协议, 和AppDelegate差不多, 所以需要在每个协议方法里重新调用以下`UnityAppController`里的对应协议方法.具体代码我直接贴在下面, 照着改一下就好.

首先AppDelegate.h:

```
#import <UIKit/UIKit.h>

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

- (void)firstInit;//第一次初始化启动Unity
- (void) startUnity;//启动Unity界面的方法
- (void) stopUnity;//停止Unity界面的方法;
@end
```

接着AppDelegate.m:

```
#import "AppDelegate.h"
#import "ViewController.h"
#import "UnityAppController.h"

@interface AppDelegate ()

@property (nonatomic, strong) UnityAppController *unityController;
@property(nonatomic, assign)BOOL isUnityRunning;
@property(nonatomic,strong) UIApplication* application;
@property(nonatomic,strong) NSDictionary * launchOptions;

@end

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    self.application = application;
    self.launchOptions = launchOptions;
    self.window.backgroundColor = [UIColor whiteColor];
    UINavigationController* naviVC = [[UINavigationController alloc] initWithRootViewController:[[ViewController alloc] init]];
    self.window.rootViewController = naviVC;

    return YES;
}

- (void)firstInit{
    self.unityController = [[UnityAppController alloc] init];
    [self.unityController application:_application didFinishLaunchingWithOptions:_launchOptions];
}

- (void)applicationWillResignActive:(UIApplication *)application {
    // Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an
incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.
    // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method to pause the game.

//    [self.unityController applicationWillResignActive:application];
}

- (void)applicationDidEnterBackground:(UIApplication *)application {
    // Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your
application to its current state in case it is terminated later.
    // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.

    [self.unityController applicationDidEnterBackground:application];
}

- (void)applicationWillEnterForeground:(UIApplication *)application {
    // Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on entering the background.

    [self.unityController applicationWillEnterForeground:application];
}

- (void)applicationDidBecomeActive:(UIApplication *)application {
    // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background,
optionally refresh the user interface.

//    [self.unityController applicationDidBecomeActive:application];
}

- (void)applicationWillTerminate:(UIApplication *)application {
    // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.

    [self.unityController applicationWillTerminate:application];
}

- (void) startUnity {
    if (!_isUnityRunning) {
        _isUnityRunning = true;
        [_unityController applicationDidBecomeActive:_application];
    }
}

- (void) stopUnity {
    if (_isUnityRunning) {
        [_unityController applicationWillResignActive:_application];
        _isUnityRunning = false;
    }
}

@end
```

7.找到Classes下的UnityAppController.h文件, 找到GetAppController()方法, 注释掉该方法, 然后替换以下方法:

```
NS_INLINE UnityAppController* GetAppController()
{
    NSObject<UIApplicationDelegate>* delegate = [UIApplication sharedApplication].delegate;
    UnityAppController* currentUnityController = (UnityAppController *)[delegate valueForKey:@"unityController"];
    return currentUnityController;
}
```

8.然后接着在默认的ViewController里面定义一个按钮用于跳转到Unity界面, 然后再定义一个按钮显示在Unity界面中用于从Unity界面调回来.
好了直接上代码更直观, 只需要修改ViewController.m文件即可:

```
#import "ViewController.h"
#import "AppDelegate.h"

@interface ViewController ()

@property (nonatomic,strong)UIView* unityView;
@property (nonatomic,strong)UIWindow* unityWindow;
@property (nonatomic,strong)AppDelegate* appDelegate;
@property (nonatomic,strong) UIButton *showButton;
@property (nonatomic,strong)UIButton* hideButton;
@property (nonatomic,assign)BOOL firstInit;

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    self.view.backgroundColor = [UIColor whiteColor];

    _appDelegate =(AppDelegate *)[UIApplication sharedApplication].delegate;

    [self createShowUnityButton];
}

- (void)createShowUnityButton{
    self.showButton = [UIButton buttonWithType:UIButtonTypeSystem];
    [self.showButton setTitle:@"加载Unity(游戏化英语)" forState:UIControlStateNormal];
    self.showButton.frame = CGRectMake(0, 0, 300, 44);
    self.showButton.center = self.view.center;
    self.showButton.backgroundColor = [UIColor greenColor];
    [self.showButton setTitleColor:[UIColor blackColor] forState:UIControlStateNormal];
    [self.showButton addTarget:self action:@selector(showUnity:) forControlEvents:UIControlEventTouchDown];
    [self.view addSubview:self.showButton];
}

- (void)showUnity:(id)sender {
    if(!_firstInit){
        _firstInit = YES;
        [_appDelegate firstInit];
        _unityView = UnityGetGLView();
        _unityWindow = UnityGetMainWindow();
        [self createHideUnityButton];
    }
    [_appDelegate startUnity];
    [_unityWindow makeKeyAndVisible];
}

- (void)createHideUnityButton{
    self.hideButton = [UIButton buttonWithType:UIButtonTypeSystem];
    [self.hideButton setTitle:@"隐藏Unity" forState:UIControlStateNormal];
    self.hideButton.frame = CGRectMake(0,0, 100, 44);
    self.hideButton.center = _unityView.center;
    self.hideButton.backgroundColor = [UIColor blueColor];
    [self.hideButton setTitleColor:[UIColor blackColor] forState:UIControlStateNormal];
    [self.hideButton addTarget:self action:@selector(hideUnity) forControlEvents:UIControlEventTouchDown];
    [_unityView addSubview:self.hideButton];
}

-(void)hideUnity{
    [_appDelegate stopUnity];
    [_appDelegate.window makeKeyAndVisible];
}

@end
```

9.当所有上面的东西都修改完毕后就可以尝试Build的以下看看有什么错误, 如果是2017.1.0f1版本则需要在Build Settings上加入Other C Flags -DRUNTIME_IL2CPP=1
都没问题后, 就可以考虑开始更新Unity导出的Xcode工程了, 在Unity大版本不变的情况下, Unity每次导出的工程其实变化的文件夹只有Classes下的Native文件夹, 和Libraries文件夹.
所以简单粗暴点就是每次把这两个文件夹删掉重新拖进去一遍, 其他文件可以都不用动. 但当Unity大版本更新的时候最好重新把Classes和Libraries都删了重新拖一遍, 然后记得把Classes下的main.mm文件内容如果和之前原生的的复制的没啥变化就直接删掉, 如果变化了,需要重新复制一遍然后再删掉, 然后就是UnityAppController.h中的GetController方法替换一下.其他的就没了.至此Unity嵌入原生应用的配置就全部搞定啦~