iOS开发中的单元测试（一）

作者：gaosboy | 时间：June 30, 2013 | 分类：猪在写代码

本文首发于 InfoQ中文站

导读：本文不讨论单元测试是什么，或者它之于一个工程的利弊，我认为单元测试是一个开发者保证产出代码质量的有效工具。本文从使用者的角度对比当下比较流行的两款单元测试框架，给大家提供一些选用建议。如果你还不甚了解单元测试在工程中所起到的作用，或者还不知道TDD的开发模式，可参考：Test-Driven Development和Unit Testing。

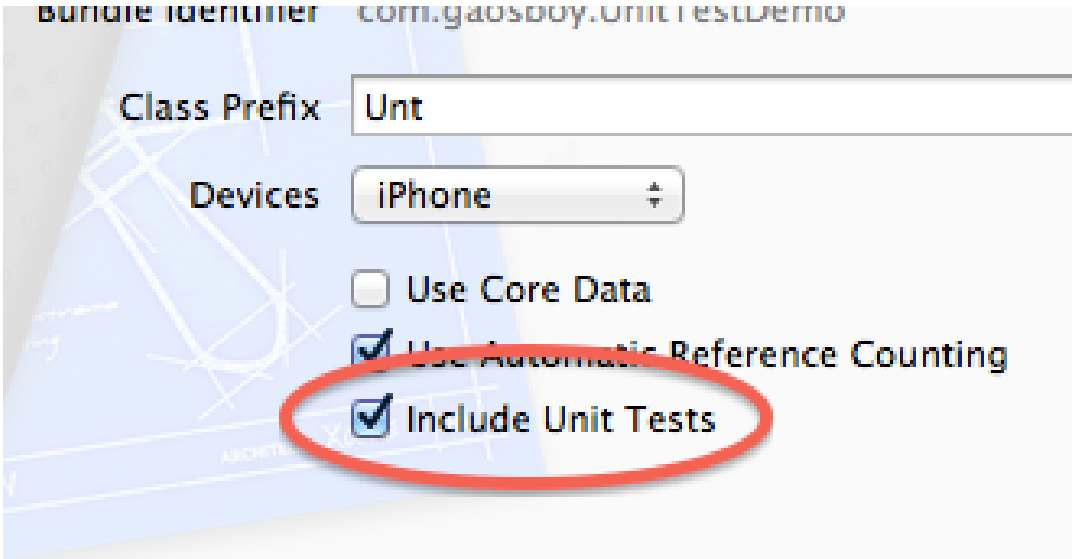
本文对比两个iOS开发中常见的单元测试框架：OCUnit，被官方集成进XCode 4.x版本中；GHUnit，被推荐最多的测试框架，带GUI界面。初窥两款测试框架非常相似，而上手使用就会发现其中的区别。细节上的区别使两款框架在不同角度各有优劣。

OCUnit

OCUnit是XCode 4.x集成的单元测试框架，OCUnit中的测试分为两类，一类称为Logic Tests，另一类称为Application Tests。Logic Tests更倾向于所谓的白盒测试，用于测试工程中较细节的逻辑；Application Tests更倾向于黑盒测试，或接口测试，用于测试直接与用户交互的接口。

• 添加单元测试

OCUnit是XCode集成的，所以其与工程的结合理应是最好的，添加到工程中的成本也理应最低。使用XCode创建新工程的流程中就有一个“Include Unit Tests”的选项（如图1），新的工程就会自动生成一个Logic Tests。



向已存在的工程中添加OCUnit Logic Tests也不复杂，只需要添加一个类型为：“Cocoa Touch Unit Testing Bundle”的Target即可（如图2）。

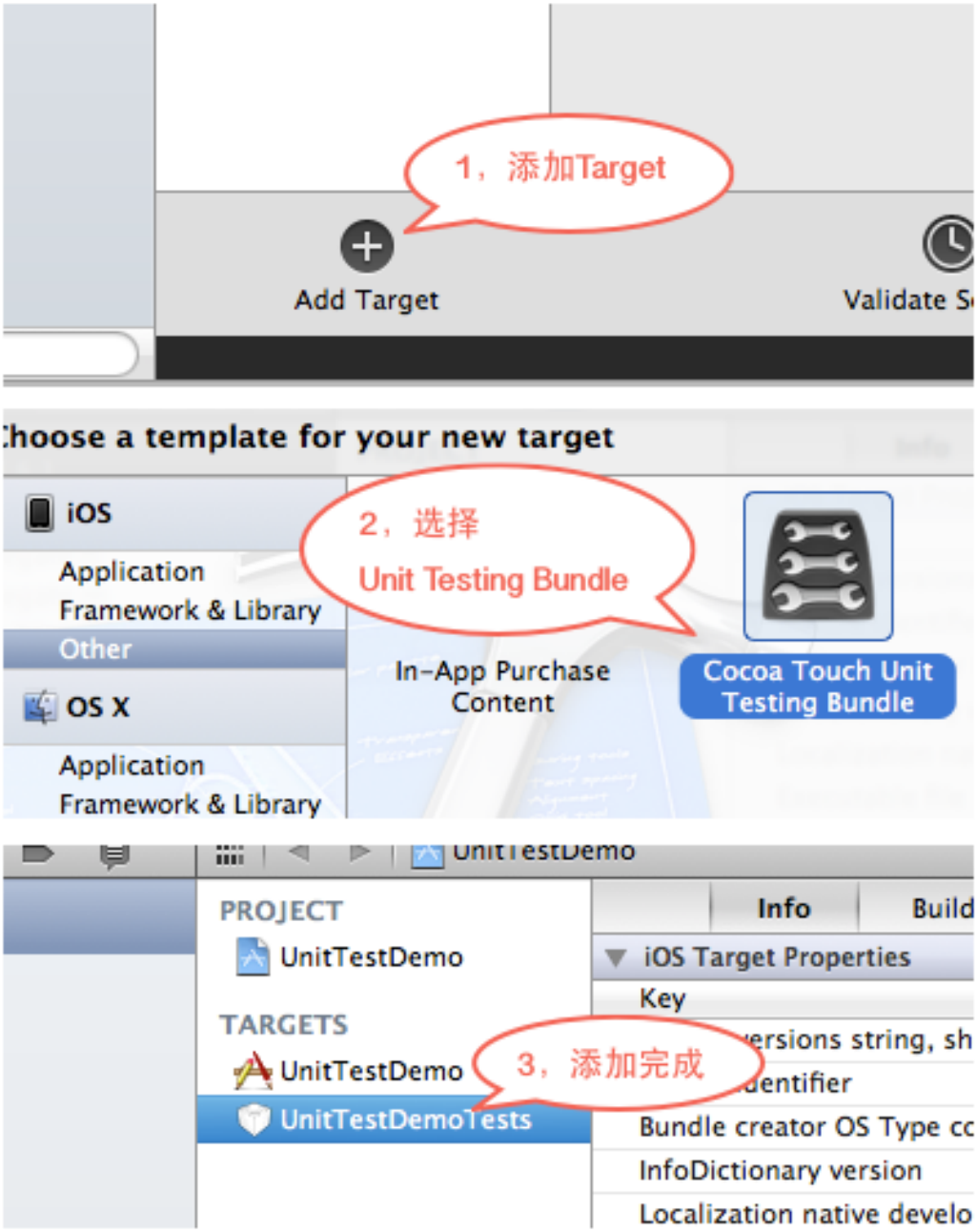


图2，向已存在的工程中添加OCUnit测试

向已有工程中添加一个测试Target时，XCode会自动生成一个Scheme，运行单元测试用例和Build原工程需要切换不同的Scheme。如果认为切换Scheme非常麻烦，也可以在添加Target之前，在“Manage Scheme”菜单中取消“Autocreate schemes”（如图3）。

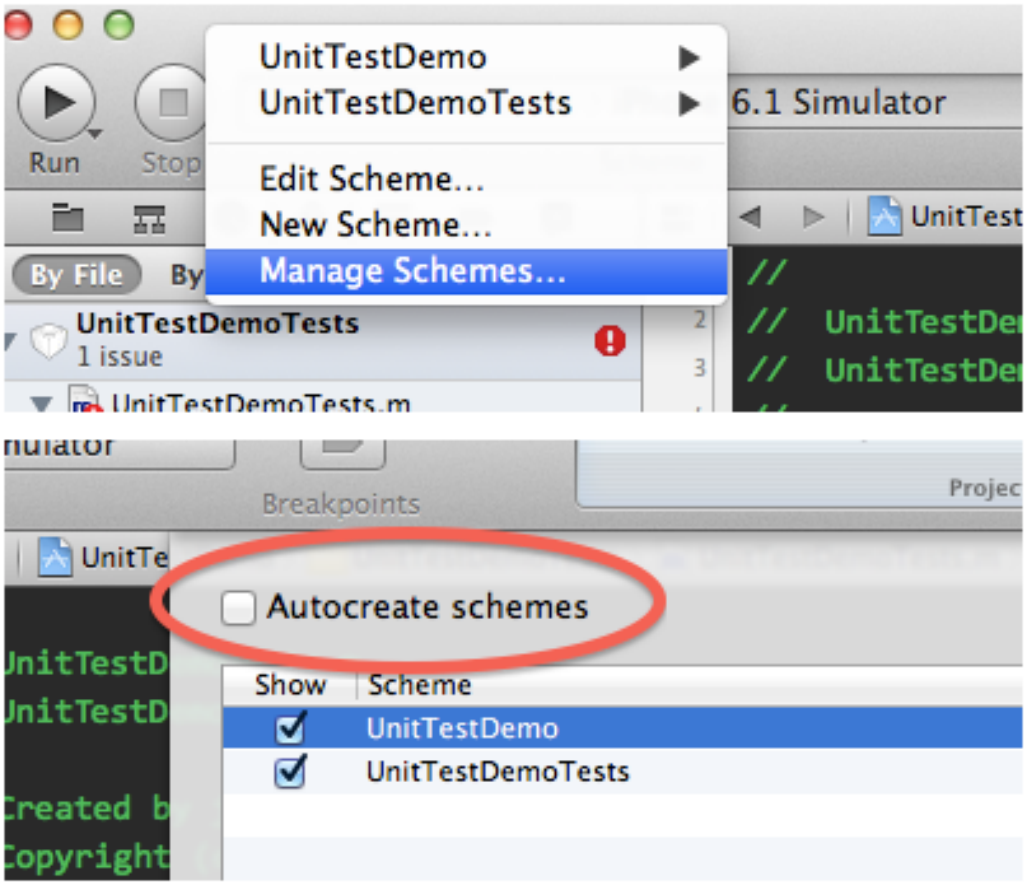


图3，添加Target不创建Scheme

Application Tests要基于Logic Tests做一些修改。一般来说一个工程既需要Logic Tests也需要Application Tests，所以建议按照上述方法添加一个单独的Target，然后执行以下操作（如图4）：

1. 在Build Settings中搜索“bundle loader”，设置为：
$(BUILT_PRODUCTS_DIR)/APP_NAME.app/APP_NAME（APP_NAME是应用名）

2. 再搜索“test host”，设置为：$(BUNDLE_LOADER)

3. 在Build Phases-Target Dependencies中添加依赖，选择主程序Target

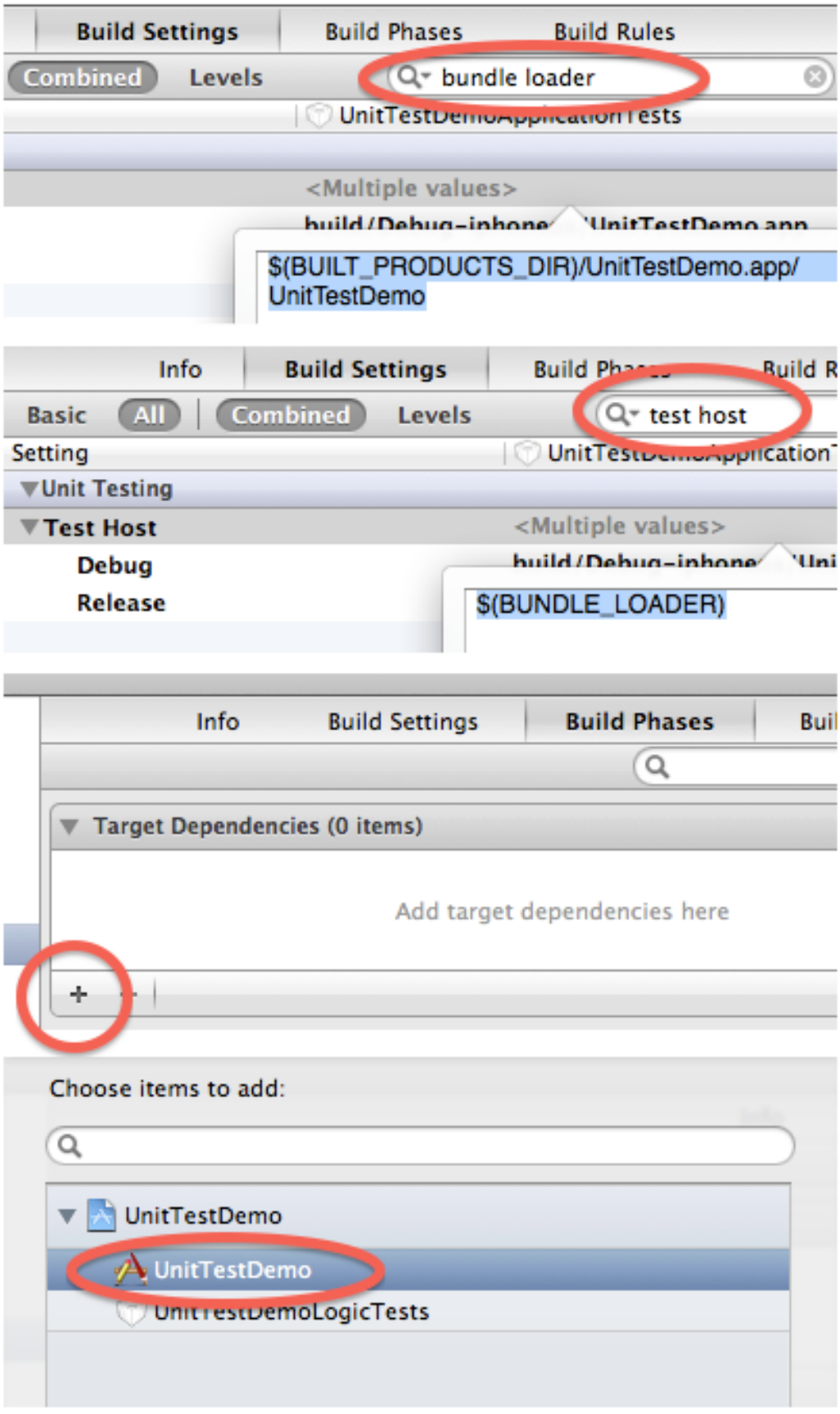


图4，添加一个Application Tests

• 创建测试用例

OCUnit的测试用例最常用的方法有三个

1. - (void)setUp：每个test方法执行前调用

2. - (void)tearDown：每个test方法执行后调用

3. - (void)testXXX：命名为XXX的测试方法

添加Target之时XCode已经自动创建了一个测试用例类：UnitTestDemoTests，其中UnitTestDemo是工程的名字，该类中已经包含了setUp，tearDown和testExample三个方法。

通过command+n，选择“Objective-C test case class”创建一个新的测试用例类（如图5）。通过XCode创建的测试用例类是一个继承自SenTestCase（OCUnit由SEN:TE公司开发，因此基类命名为SenTestCase）的空类，需要模仿UnitTestDemoTests编写测试方法。

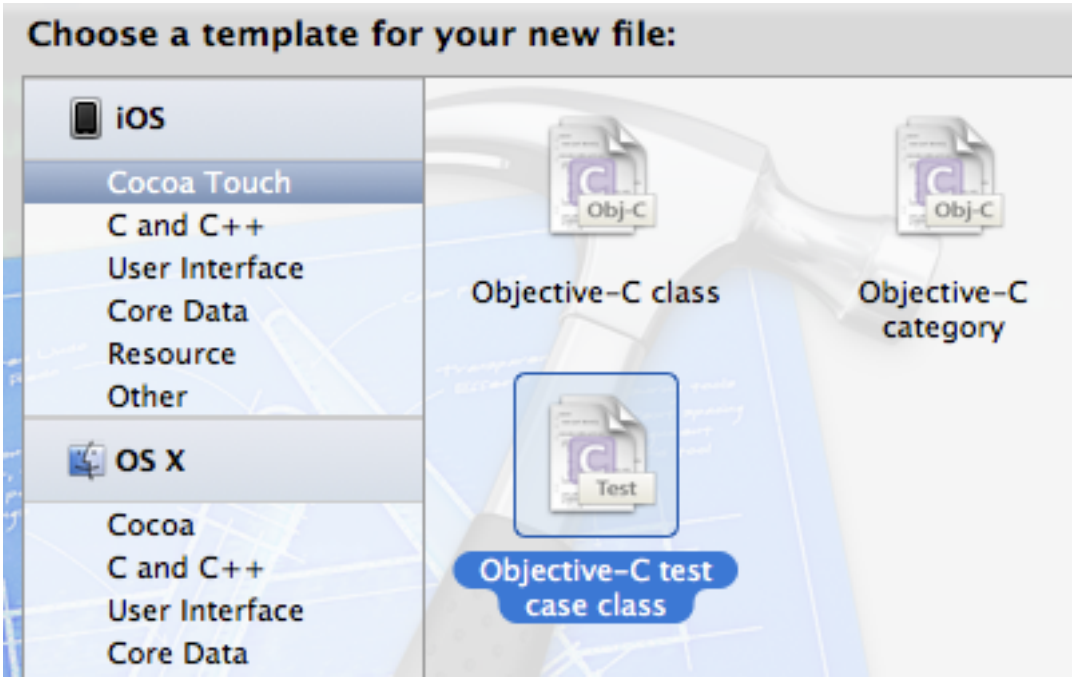


图5，创建一个测试用例类

开发者可以自己实现无返回值，且命名规则为testXXX的实例方法，并使用框架提供的大量断言方法。

Logic Tests与Application Tests的区别主要在setUp方法，Logic Tests只需在setUp方法中初始化一些测试数据，而Application Tests需要在setUp方法中获取主应用的AppDelegate，供test方法调用。

值得注意的是，OCUnit的test bundle是侵入主应用的，因此在使用过程中要十分注意，不要让单元测试的资源覆盖主应用资源，造成诡异的Bug。

• 运行测试

由于OCUnit是集成在XCode中的框架，因此在XCode中运行也比较方便。切换到单元测试的scheme（如果与工程共用scheme则无需切换），Product->Test（或直接使用快捷键command+u），框架会自动查找所有工程中SenTestCase的子类，运行其中全部命名类似testXXX的无返回值方法。

• 测试反馈

OCUnit的失败方法会通过Console和XCode Issues两个位置反馈，通过XCode Issues可以直接定位到出现错误的单元测试代码行。Issue的提示信息就是在单元测试断言方法中定义的description。

GHUnit

GHUnit是一款Objective-C的测试框架，除了支持iOS工程还支持OSX的工程，但OSX不在本文的讨论范围。GHUnit不同于OCUnit，它提供了GUI界面来操作测试用例，而且也不区分Logic Tests和Application Tests。

• 添加单元测试

与集成进XCode的OCUnit相比，GHUnit的添加过程略显复杂。首先在上下载GHUnit的框架包，当前的For iOS的最新版本是0.5.6，解压后是一个GHUnitIOS.framework的文件夹。

打开已经存在的工程，添加一个EmptyApplication Target，并在新Target中添加刚刚下载的GHUnitIOS.framework（如图6、7）。

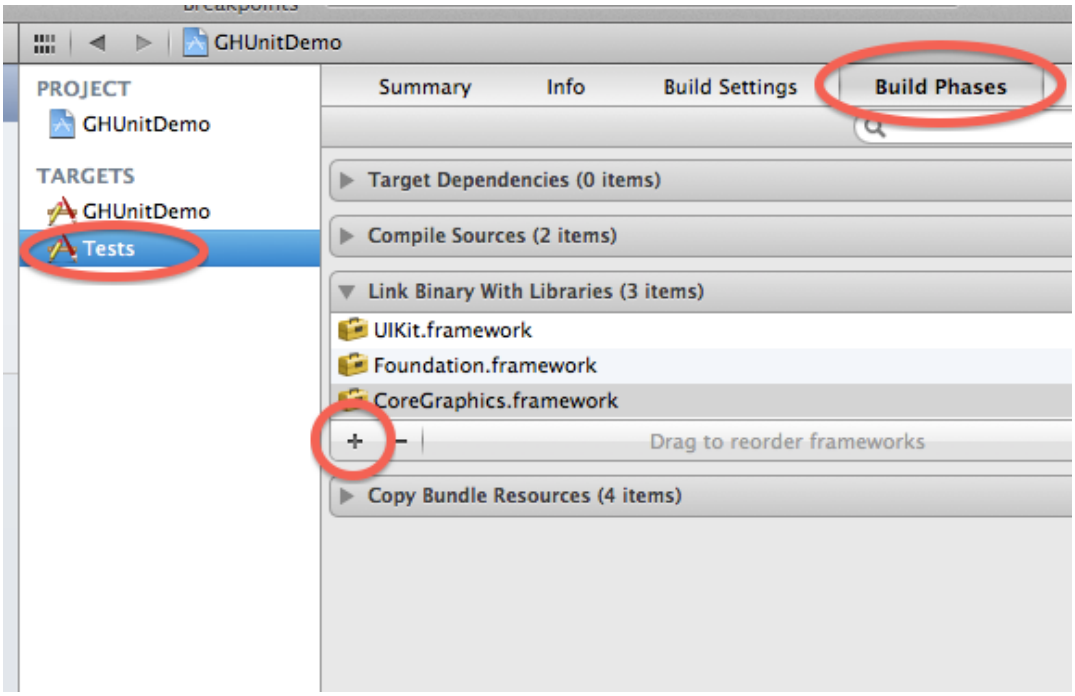


图6，在新Target中添加GHUnitIOS.framework

在Build Phases中添加非官方框架并不会把框架文件拷贝到工程目录，而是只做一个链接，所以建议在添加之前先把框架拷贝到工程目录下。

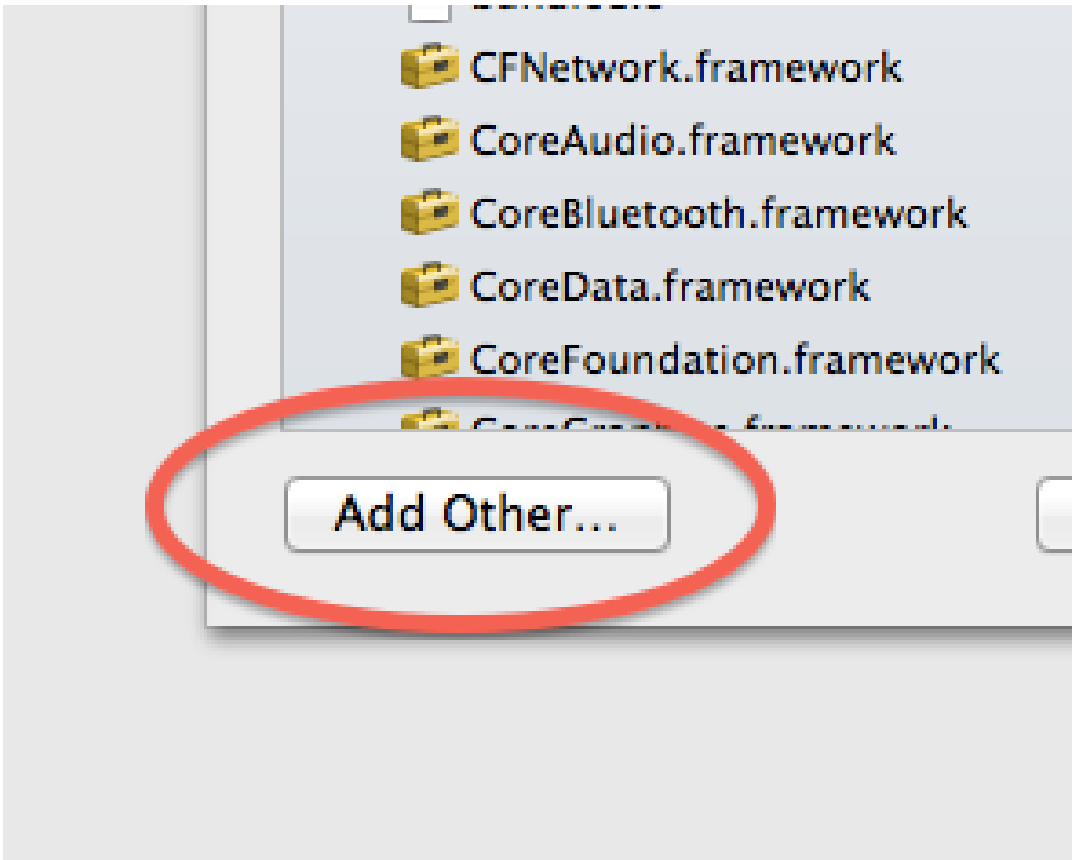


图7，选择GHUnitIOS.framework

接下来用相同的方法添加框架依赖的其他库：“QuartzCore.framework”。

在Build Settings中搜索“linker flags”，设置Other Linker Flags - Debug - 添加一个支持全架构和全版本SDK的标示“-ObjC -all_load”（如图8）。

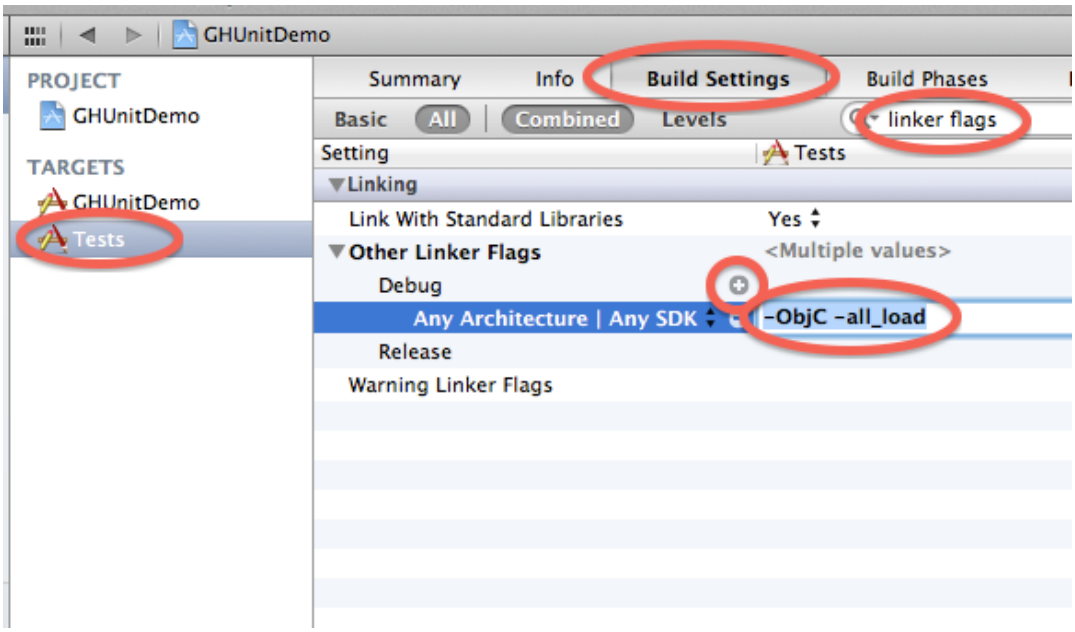


图8，设置linker flags

删除Tests Target中的AppDelegate（.h和.m一起删除）。修改main函数，支持GHUnitIOS，导入GHUnitIOSAppDelegate代替原来的AppDelegate，修改UIApplicationMain的参数（如图9）。

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

#import <GHUnitIOS/GHUnitIOSAppDelegate.h>

int main(int argc, char *argv[])
{
    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil,
                                 NSStringFromClass([GHUnitIOSAppDelegate class])
                                 );
    }
}
```

图9，修改main函数

至此已经完成了GHUnit的添加，选择新建Target同时创建的scheme，直接Build and Run即可在设备或Simulator中启动一个新的App（如图10），即该单元测试的App。

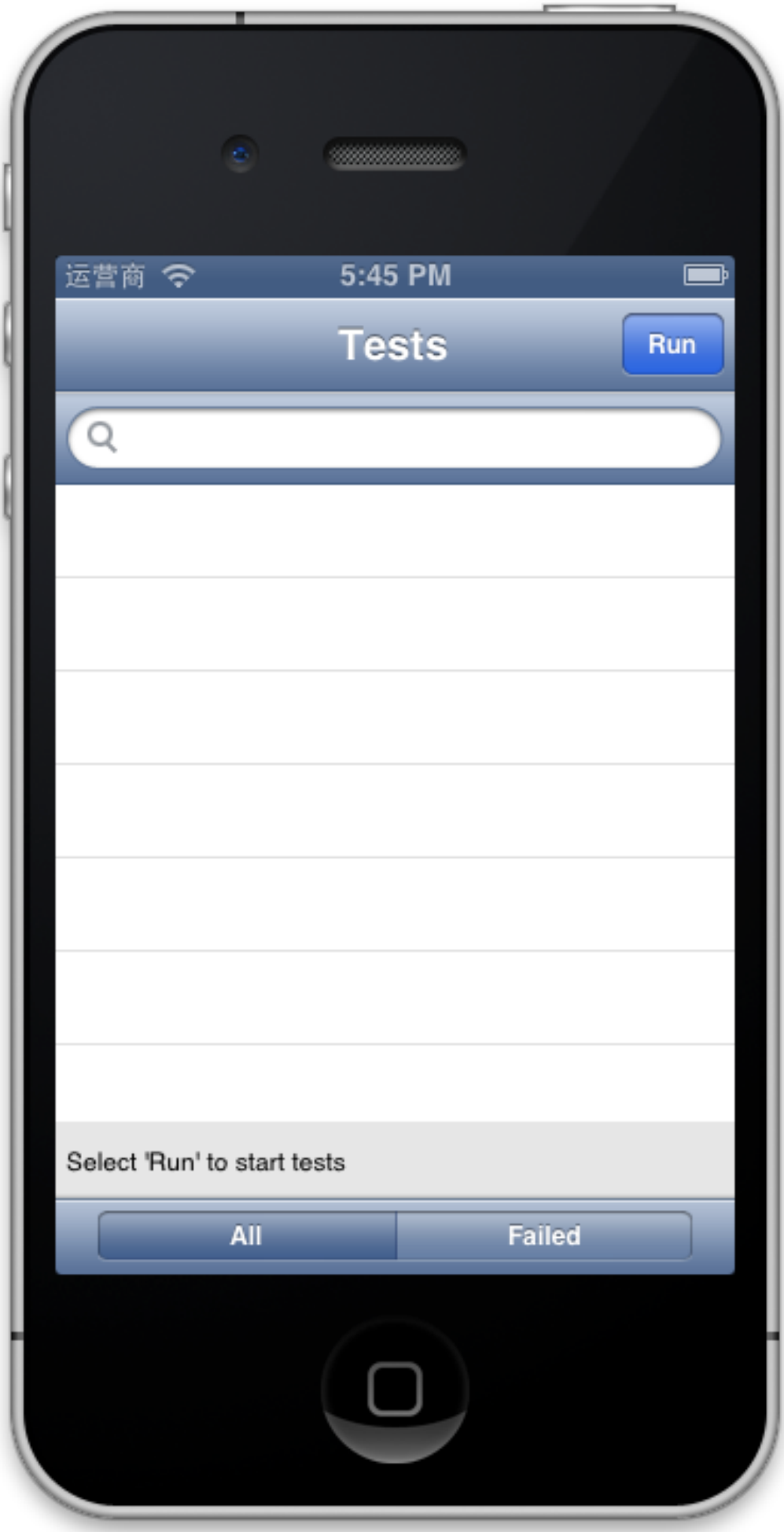


图10，单元测试App

• 创建测试用例

创建GHUnit测试用例与创建OCUnit测试用例相似。

新建一个Objective-C Class文件，继承自GHTestCase，在XCode生成的.h文件中不会导入GHUnit.h文件，需要开发者自行导入“#import <GHUnitIOS/GHUnit.h>”。

GHUnit框架提供断言方法比OCUnit更加丰富，开发用例也就可以做的更加细致，更有利查找/定位错误。

测试方法的命名规则与OCUnit一样，是以test开头的无返回值方法：- (void)testXXX。而常用的方法除了上述提到的setUp和tearDown，GHUnit还提供了setUpClass和tearDownClass两个方法，在该用例运行前和结束后调用。另外，刚刚提到GHUnit不区分Logic Tests和Application Tests，所以在setUp和tearDown方法中也就不存在设置的区分。

• 运行测试

运行GHUnit需要分两步，首先编译并安装单元测试App到设备或Simulator里（如图11），创建了两个用例，每个用例中分别有一个方法。

图11，两个用例的GHUnit App

在App中可以通过点击右上角的Run按钮运行全部用例，框架会查找所有以testXXX命名的无返回值方法，并执行。或点击TableView中的某个Cell运行单独的测试方法。

• 测试反馈

断言失败测试未通过的方法在App中会标记为红色，并给出每一个方法的运行时间。在Console中会打印出详细的出错信息，包括：异常类型，出错文件，位置，以及断言方法中指定的出错原因。更重要的是，出错时的程序堆栈内容（如图12）。

```
    Name: GHTestFailureException
    File: /Users/jiajun/dev/ios/Demos/GHUnitDemo/Tests/UntTestCase.m
    Line: 14
    Reason: 'a' should be equal to 'b'. a不等于b

(
    0   CoreFoundation                      0x015d302e __exceptionPreprocess + 206
    1   libobjc.A.dylib                     0x012e0e7e objc_exception_throw + 44
    2   CoreFoundation                      0x0165bfb1 -[NSException raise] + 17
    3   Tests                               0x0000d965 -[GHTestCase failWithException:] + 33
    4   Tests                               0x00002c71 -[UntTestCase testExample] + 1089
    5   libobjc.A.dylib                     0x012f4663 -[NSObject performSelector:] + 62
    6   Tests                               0x0000989d +[GHTesting runTestWithTarget:selector:exception:interval:reraiseExceptions:] + 450
    7   Tests                               0x00004c86 -[GHTest run:] + 278
    8   Tests                               0x0000743f -[GHTestGroup _run:] + 715
    9   Tests                               0x00007769 -[GHTestGroup run:] + 130
    10  Tests                               0x0000743f -[GHTestGroup _run:] + 715
    11  Tests                               0x00007769 -[GHTestGroup run:] + 130
    12  Tests                               0x00009c48 -[GHTestRunner runTests] + 257
    13  Tests                               0x00009db8 -[GHTestRunner _runInBackground] + 79
    14  Foundation                          0x00d2b805 -[NSThread main] + 76
    15  Foundation                          0x00d2b764 __NSThread__main__ + 1304
    16  libsystem_c.dylib                   0x9a0c15b7 _pthread_start + 344
    17  libsystem_c.dylib                   0x9a0abd4e thread_start + 34
)
 FAIL (0.000s)
```

图12，未通过测试的方法，Console中的内容

GHUnit通过Console中的内容给开发者提供帮助，可以快速定位程序出错的位置，这一点比OCUnit做的要好。

总结

GHUnit在安装上确实显得有些麻烦，无法跟集成在XCode里的OCUnit相比。 但从开发者的角度讲，我更喜欢GHUnit带来的体验，GUI的操作界面可以脱离IDE单独运行，支持运行单一测试方法和运行全部用例的，打印出错堆栈可以更快定位到问题所在。

本文简单介绍了两款框架的安装与入门，可以初步了解其各自特点，在接下来的文章中将会更加详细的介绍如何使用框架进行单元测试，以及框架中的一些高级功能。此外，后续还将向大家介绍另外的与这两款框架区别更加明显的单元测试框架。

标签：ios

添加新评论

称呼 *

Email *

网站

http://

内容 *

提交评论