**自动化测试**

**一、测试种类**

1、白盒测试

白盒测试又称结构测试、逻辑驱动测试或基于程序代码内部结构的测试。白盒测试的时候，测试人员是可以直接接触待测试App的源代码的。白盒测试更多的是单元测试，测试人员针对各个单元进行各种可能的输入分析，然后测试其输出。白盒测试的测试代码通常由iOS开发编写。

2、黑盒测试

黑盒测试又叫功能测试、数据驱动测试或给予需求规格说明书的功能测试。这种测试注重于测试软件的功能性需求。黑盒测试的时候，测试人员不需要接触源代码。是从App层面对其行为以及UI的正确性进行验证，黑盒测试由软件测试工程师完成。

按照业务来说又可分为：

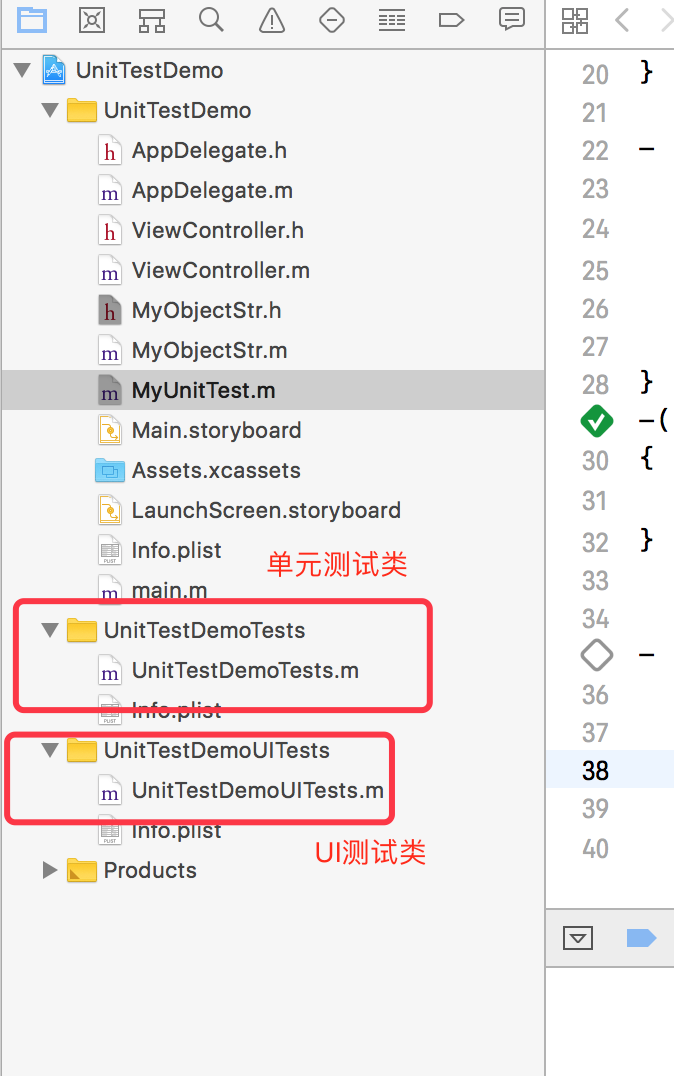
1、单元测试：单元测试（英语：Unit Testing）又称为模块测试, 是针对程序模块的最小单位来进行正确性检验的测试工作

2、UI测试：从业务层的角度保证各个业务可以正常工作

**二、常用框架**

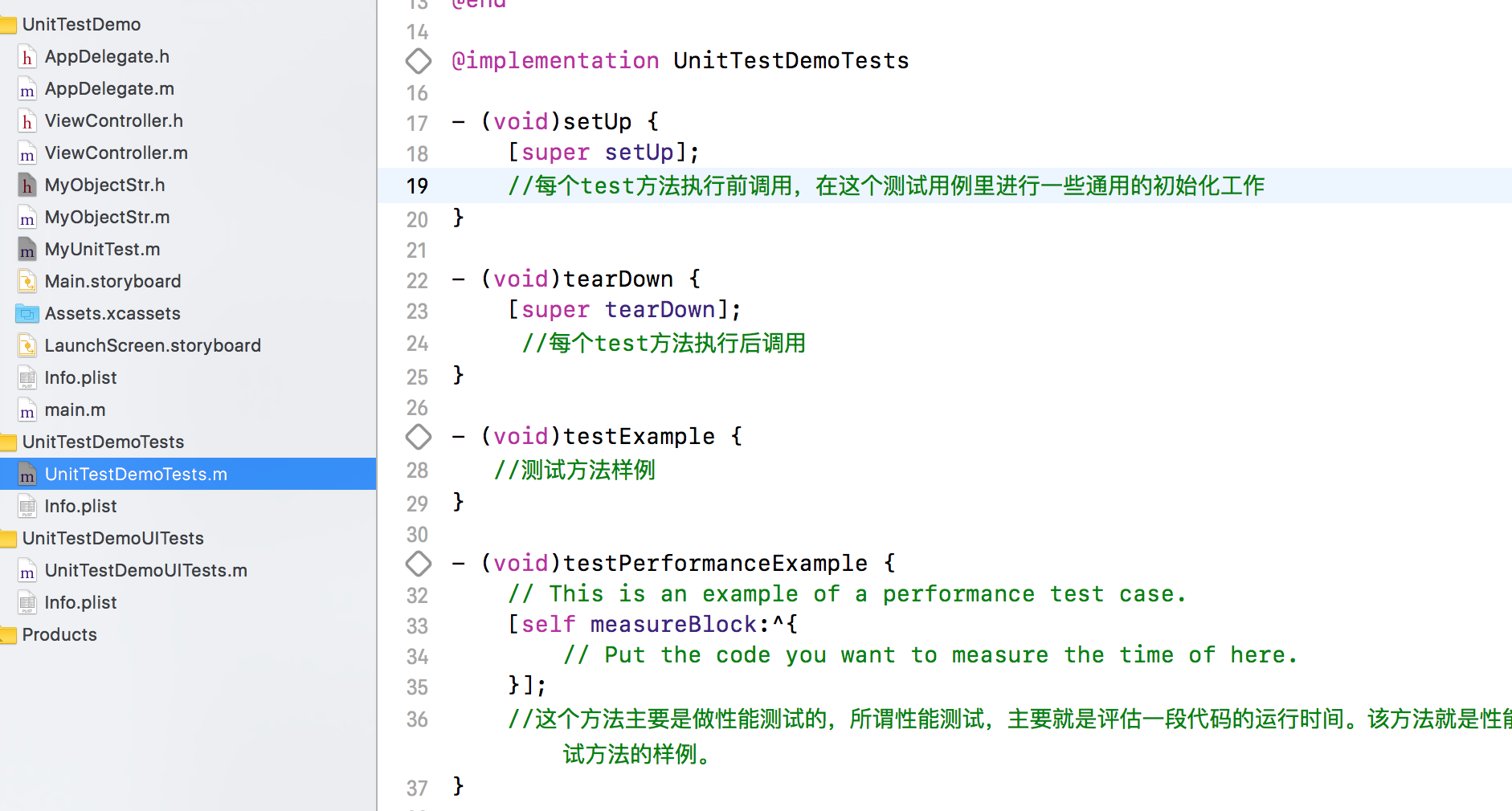
1、XCTest框架

XCTest框架是Xcode7系统自带的框架， 使用Xcode7 以上的软件新建的项目都会默认生成一个单元测试类和UI测试类。如下图



1.1 单元测试

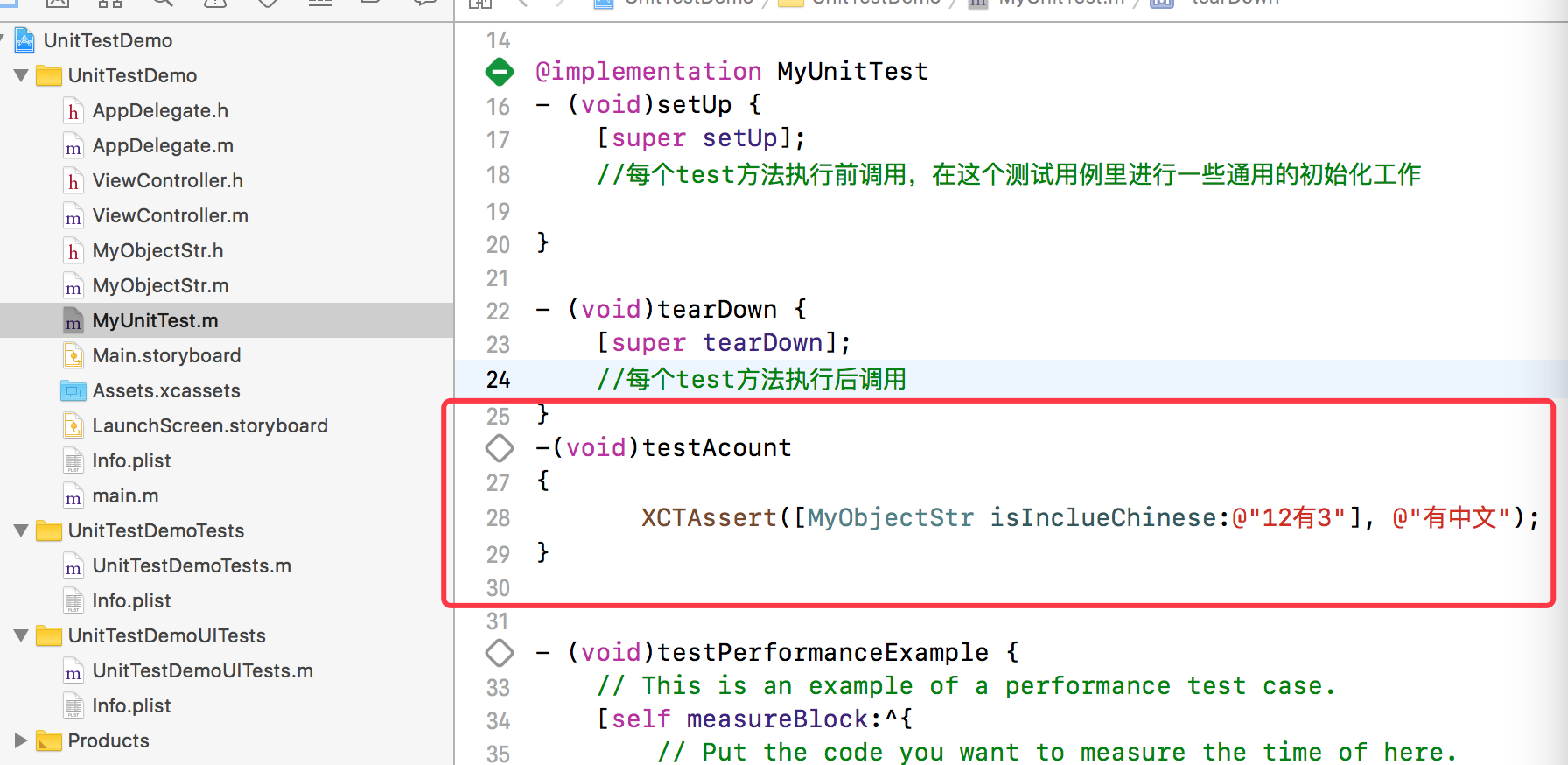
单元测试类中含有四个方法，每个方法都有各自的含义，如下图



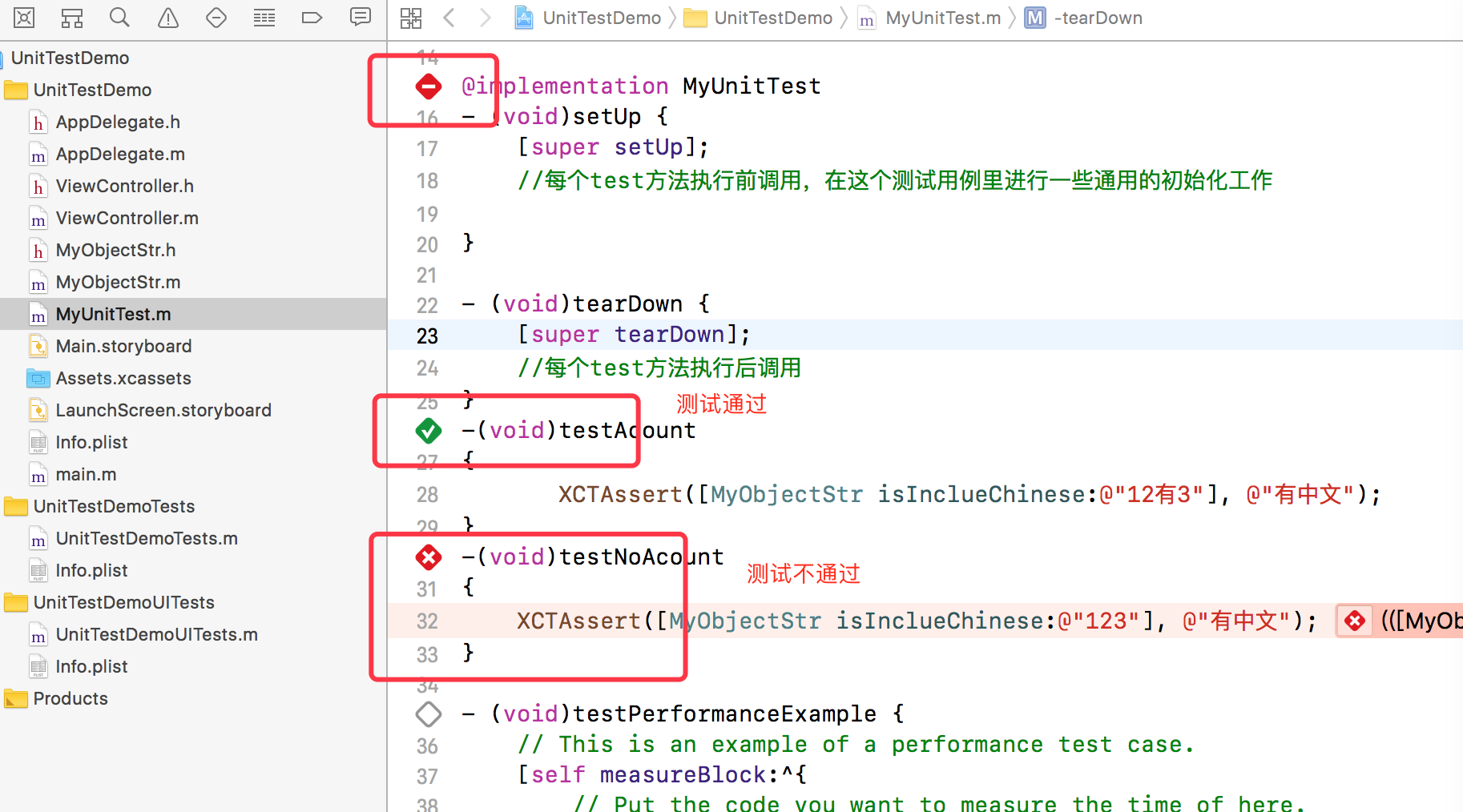
1）、对某一个类创建对应的单元测试类

Xcode的File->New->File->Source选择Unit Test Case Class来新建一个UserTests,注意要继承XCTestCase类。

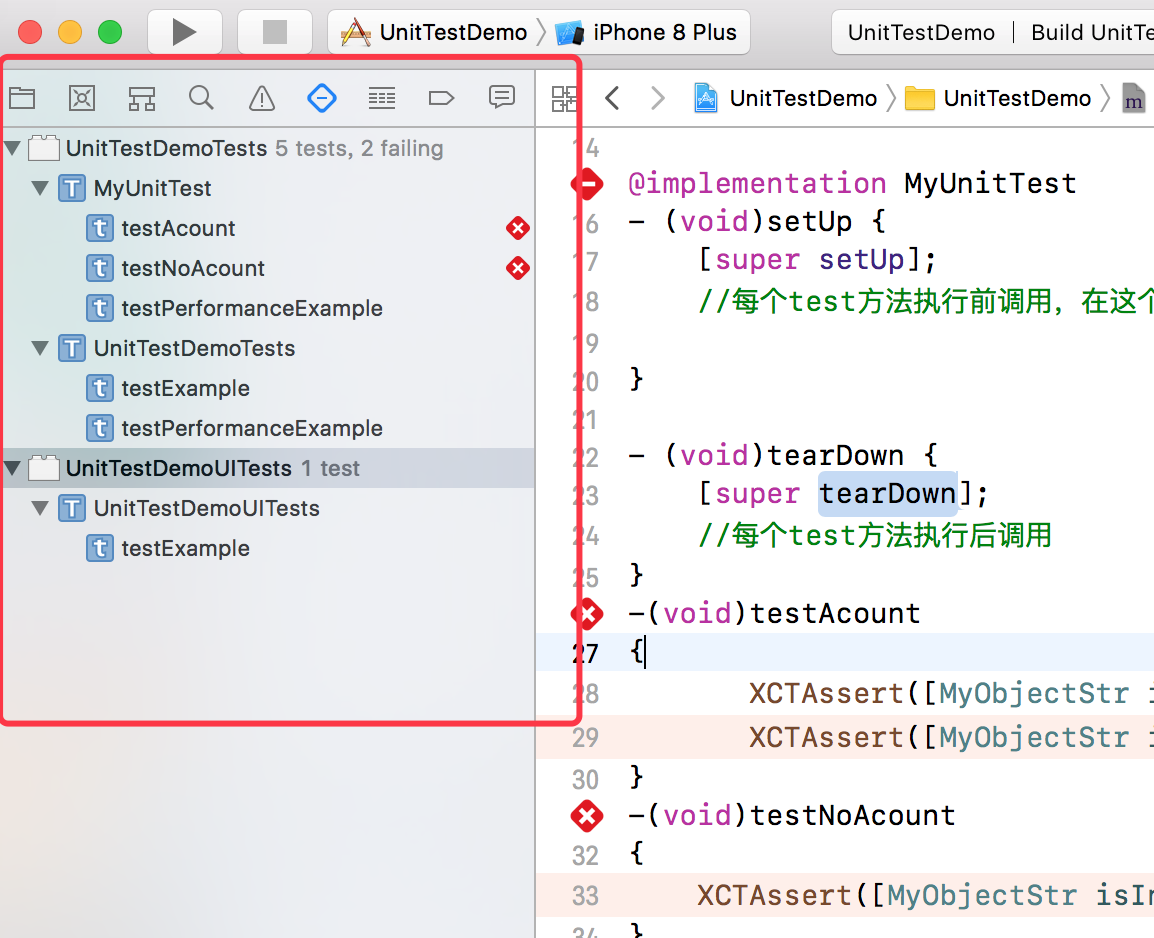
2）、对应的测试方法只要是以“test”开头就行，如下图



3）、 选择对应的模拟器之后，然后点击左侧棱形就可以进行单元测试,测试通过显示绿色的勾勾，测试不通过显示红色的叉叉。



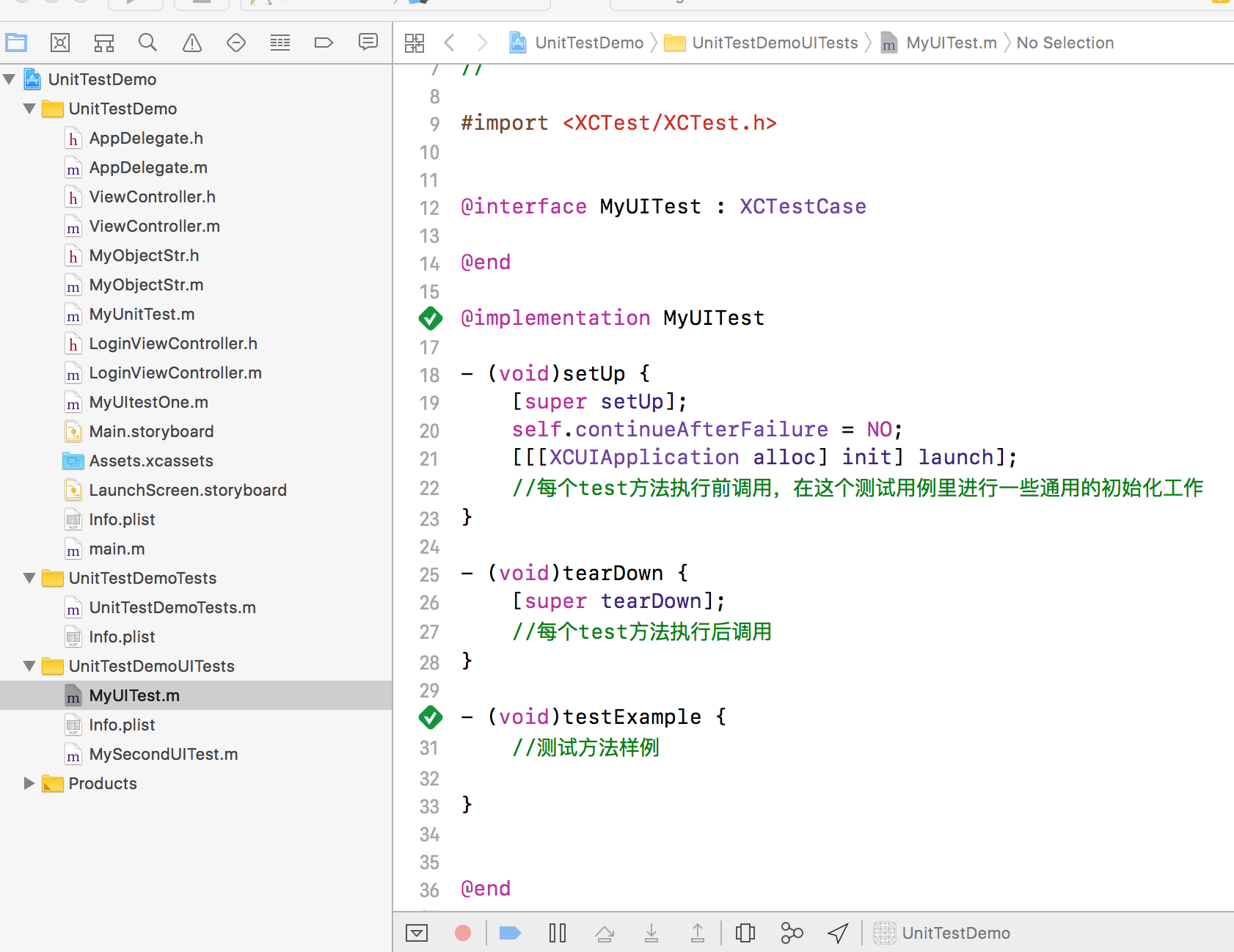
4）、所有的测试类和方法都可直观的看到，如下图



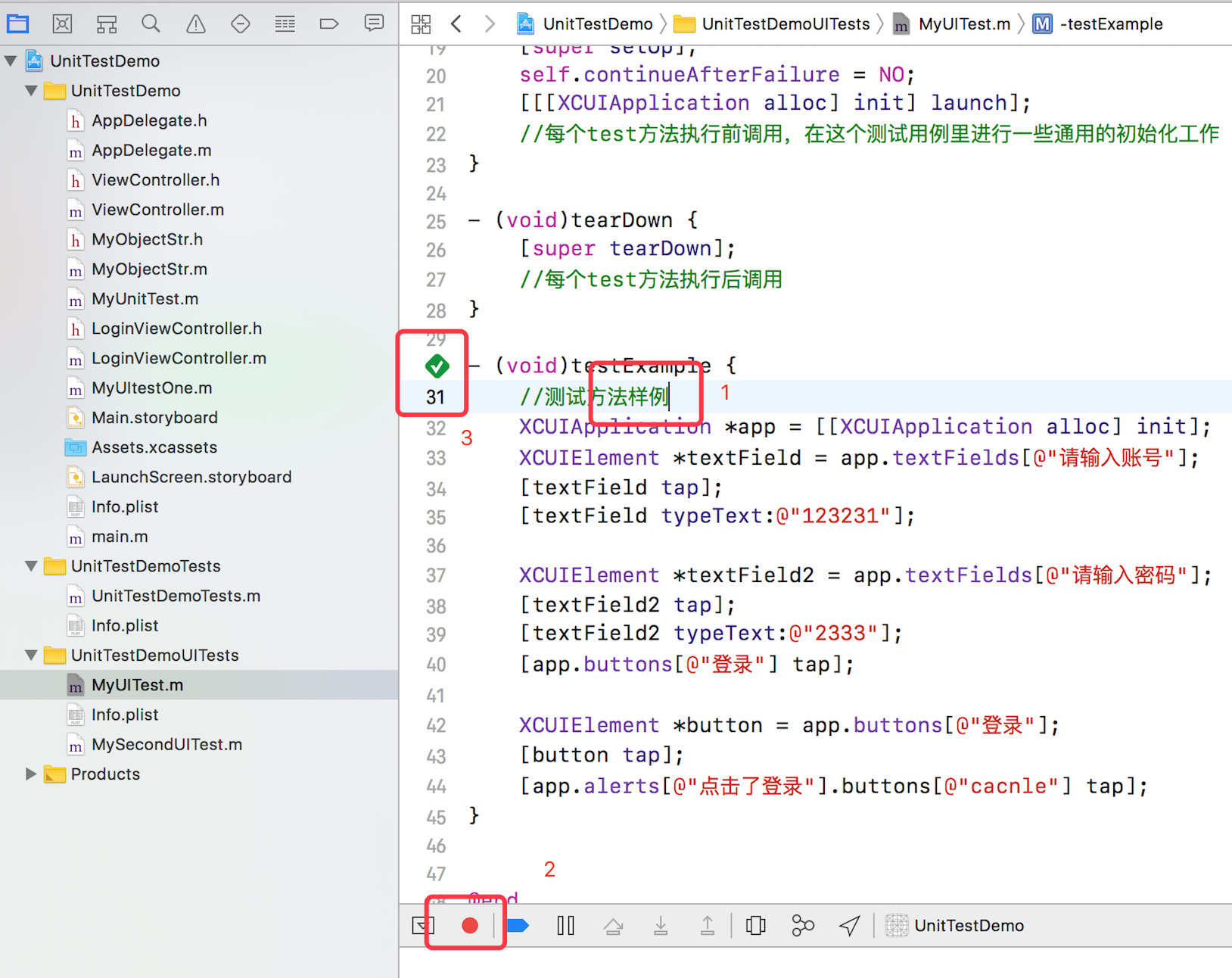
1.2、UI测试

1）、为某一个类创建对应的UI测试类，并将文件放到特定的文件夹

Xcode的File->New->File->Source选择UI Test Case Class来新建一个MyUITest,注意要继承XCTestCase类。



2)、将光标点击到 testExample方法中，然后点击做下角的红色录制按钮。然后在模拟器中进行操作，在testExample方法中会自动生成UI测试的初步代码，然后再进行对应的UI测试需求修改，就可以点击左侧的棱形进行自动化的UI测试。如下图



注：常用断言

XCTFail(format…) //生成一个失败的测试；

XCTAssertNil(a1, format...) //为空判断，a1为空时通过，反之不通过；

XCTAssertNotNil(a1, format…) //不为空判断，a1不为空时通过，反之不通过；

XCTAssert(expression, format...) //当expression求值为TRUE时通过；

XCTAssertTrue(expression, format...) //当expression求值为TRUE时通过；

XCTAssertFalse(expression, format...) //当expression求值为False时通过；

XCTAssertEqualObjects(a1, a2, format...) //判断相等，[a1 isEqual:a2]值为TRUE时通过，其中一个不为空时，不通过；

XCTAssertNotEqualObjects(a1, a2, format...) //判断不等，[a1 isEqual:a2]值为False时通过；

XCTAssertEqual(a1, a2, format...) //判断相等（当a1和a2是 C语言标量、结构体或联合体时使用,实际测试发现NSString也可以）；

XCTAssertNotEqual(a1, a2, format...) //判断不等（当a1和a2是 C语言标量、结构体或联合体时使用）；

XCTAssertEqualWithAccuracy(a1, a2, accuracy, format...) //判断相等，（double或float类型）提供一个误差范围，当在误差范围（+/-accuracy）以内相等时通过测试；

XCTAssertNotEqualWithAccuracy(a1, a2, accuracy, format...) //判断不等，（double或float类型）提供一个误差范围，当在误差范围以内不等时通过测试；

XCTAssertThrows(expression, format...) //异常测试，当expression发生异常时通过；反之不通过；

XCTAssertThrowsSpecific(expression, specificException, format...) //异常测试，当expression发生specificException异常时通过；反之发生其他异常或不发生异常均不通过；

XCTAssertThrowsSpecificNamed(expression, specificException, exception\_name, format...) //异常测试，当expression发生具体异常、具体异常名称的异常时通过测试，反之不通过；

XCTAssertNoThrow(expression, format…) //异常测试，当expression没有发生异常时通过测试；

XCTAssertNoThrowSpecific(expression, specificException, format...) //异常测试，当expression没有发生具体异常、具体异常名称的异常时通过测试，反之不通过；

XCTAssertNoThrowSpecificNamed(expression, specificException, exception\_name, format...) //异常测试，当expression没有发生具体异常、具体异常名称的异常时通过测试，反之不通过

2、Kiwi框架

测试思想

1）行为驱动开发（英语：Behavior-driven development，缩写BDD）是一种敏捷软件开发的技术，BDD的重点是通过与利益相关者的讨论取得对预期的软件行为的清醒认识。它通过用自然语言书写非程序员可读的测试用例扩展了测试驱动开发方法。

2）测试驱动开发（英语：Test-driven development，缩写为TDD）是一种软件开发过程中的应用方法，由极限编程中倡导，以其倡导先写测试程序，然后编码实现其功能得名。测试驱动开发是戴两顶帽子思考的开发方式：先戴上实现功能的帽子，在测试的辅助下，快速实现其功能；再戴上重构的帽子，在测试的保护下，通过去除冗余的代码，提高代码质量。测试驱动着整个开发过程：首先，驱动代码的设计和功能的实现；其后，驱动代码的再设计和重构。

Kiwi是一套以BDD(行为驱动开发)为思想的可读性良好的测试框架，其底层由XCTest框架实现，与XCode结合度很高。