iOS单元测试初探以及OCMock使用入门

这段时间在工作之余研究了一下iOS的单元测试，试图在项目中引入开发自己写的白盒测试，积攒一些用例来减少之后修改代码后引发的缺陷。

一、为什么需要单元测试

写代码的过程中，我们发现有一些很基础功能的接口，测试在黑盒测试中很难发现缺陷。假如我们在开发过程中书写好一些用例，不仅能提高代码的质量，也能保证在之后的改动中及时发现改动会带来的错误。

二、选择什么框架来做单元测试

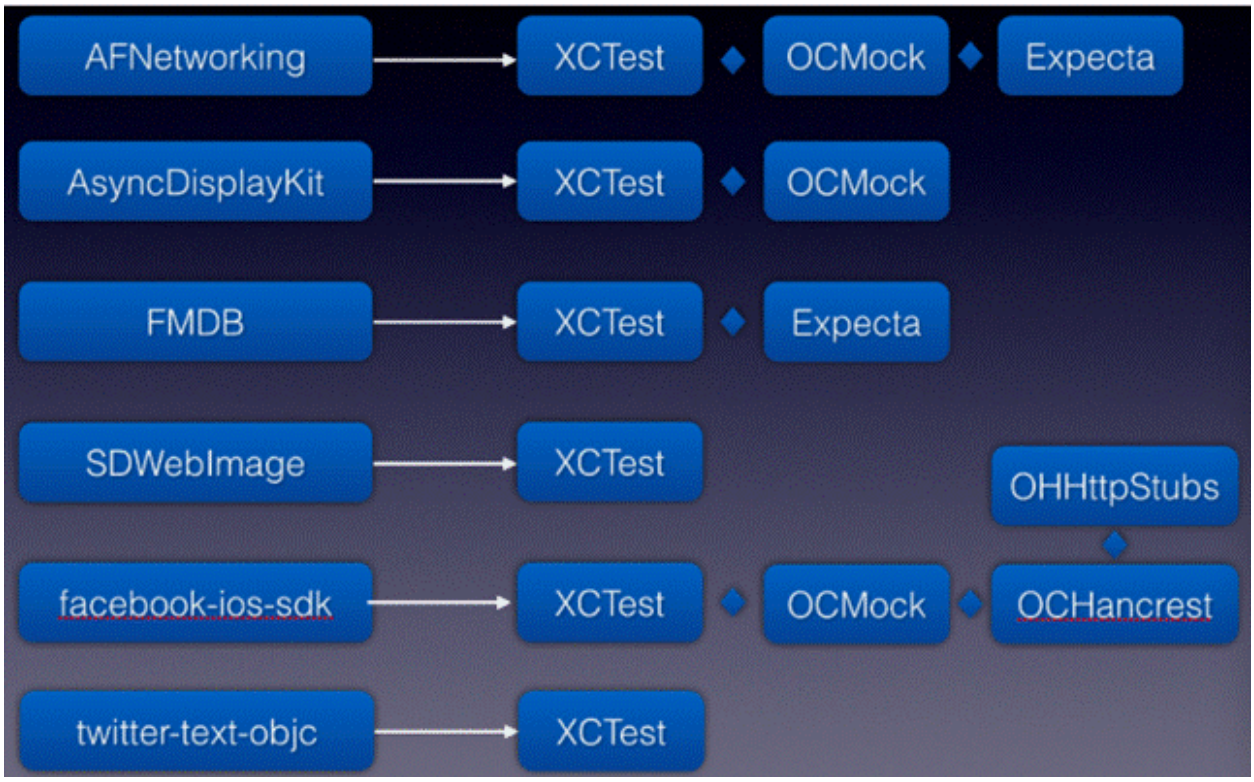
比较流行的有两种单元测试的思路模式，Test Driven Development(TDD)和Behavior Driven Development(BDD)。

`TDD`：先根据需求或者接口情况编写测试，然后再根据测试来编写业务代码，这也就必然导致所有代码的public部分都会需要必要的测试。

`BDD`：通过Given - When - Then三个流程化的条件来帮助开发确定应该测试什么

从一开始就舍弃掉了所有TDD的方案，因为就项目的现状来说实行起来效率会很低。对比了一下现有的BDD框架，还是想选用XCTest + OCMock的方案。原因主要是希望更好适应和Apple之后的更新，基本能满足需求，也更容易上手。

选择上也参考了一些现在主要采用同样方案的开源库：



三、配置运行XCTest需要的环境

Test target是和主Target在同一个project文件下，运行的另一个Target，设置编译和Target Radio同样的文件加上自己加上的测试文件

工程中使用了Cocoapods来管理第三方库的话，Test target也需要添加响应的第三方库依赖，这样所有在工程中能使用到的代码，也能在测试用例用使用。

四、如何搭建OCMock的运行环境

在搭建好XCTest的情况下，按照http://ocmock.org/ios/能很快配置好OCMock的运行环境。

五、如何使用XCTest + Mock来完成单元测试

（5.1）XCTest简介

1. 整个工程中应该有多个XCTest文件，每一个XCTests都是只有.m文件

的，继承自XCTestCase

2. 每个XCTests.m文件中，必然包含一个setUp方法和一个tearDown方法
3. 每个单元测试方法执行之前，XCTest会先执行setUp方法，所以可以把一些测试代码需要用的初始化代码写在这个方法里
4. 每个单元测试方法执行完毕后，XCTest会执行tearDown方法，所以可以把需要测试完成后销毁的内容写在这个里，以便保证下面的测试不受本次测试影响

```
#import <XCTest/XCTest.h>
#import "NSNumber+Sugar.h"

@interface FMNSNumberSugarTests : XCTestCase

@end

@implementation FMNSNumberSugarTests

- (void)setUp {
    [super setUp];
    // Put setup code here. This method is called before the invocation of each test
        method in the class.
}

- (void)tearDown {
    // Put teardown code here. This method is called after the invocation of each test
        method in the class.
    [super tearDown];
}
```

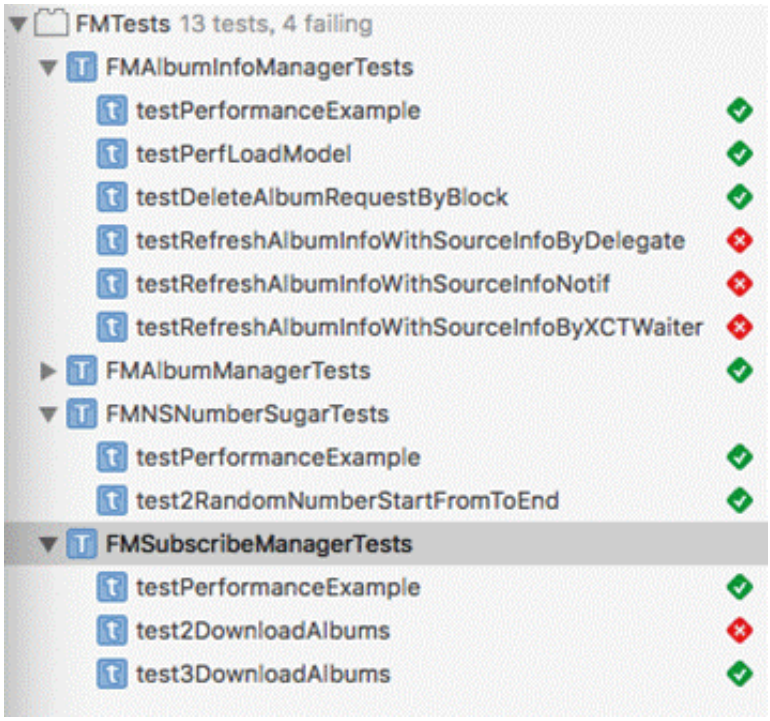
(5) 只执行test开头的用例，可以用前面加DISABLE_xxx 来表示废弃该用例

(6) command + U顺序运行测试用例

(7) measureBlock用来测试性能

```
- (void)DISABLE_testExample {
    // This is an example of a functional test case.
    // Use XCTAssert and related functions to verify your tests produce the correct
        results.
}
```

(8) 运行结果如下图，成功和失败的错误用例显示在右侧，还可以选择导出代码的覆盖率



（5.2）XCTest + OCMock能够帮助我们完成的事

5.2.1. 基本断言的逻辑测试

XCTest定义了一系列的断言来帮助代码的正确性，详细种类列举在断言列表

- 例：有一个函数目的是生成在[base, end]之间的随机数

```
//获得[base, top]之间的随机数
+ (int)randomNumberStartFrom: (int)base ToEnd: (int)top;
```

```
+ (int)randomNumberStartFrom: (int)base ToEnd: (int)top {
    if(base >= top){
        return base;
    }
    return (arc4random() % (top - base + 1)) + base;
}
```

```
-(void)test2RandomNumberStartFromToEnd {
    int base = 4;
    int top = 100;

    int testCount = 110;
    for(int i = 0; i < testCount; i++){
        int temp = [NSNumber randomNumberStartFrom:base ToEnd:top];
        if(temp < base || temp > top){
            XCTFail(@"invalid number = %d", temp);
        }
    }
}
```

5.2.2. 异步测试

代码中会有很多异步的场景需要验证，例如一些操作需要在不同的线程，在delegate method，或是callback中执行的操作。XCTest中主要使用XCTestExpectation来进行异步是否完成期望的测试。一般有两种方式完成异步的测试Block和Delegate：

By Block: 如果超时或者是遇到断言的失败，该用例会失败。

```
//删除专辑
- (void)sendDeleteAlbumRequest:(FMDeleteAlbumCompleteBlock)completeBlock;
```

```
#pragma -mark 异步测试用例
- (void)testDeleteAlbumRequestByBlock {
    self.exp1 = [[XCTestExpectation alloc] initWithDescription:@"Block异步测试：删除专辑数
        据"];
    [self.manager sendDeleteAlbumRequest:^(NSError *error, id response) {
        if(response){
            //add more condition
            [self.exp1 fulfill];
        }
    }];
    [self waitForExpectations:@[self.exp1] timeout:2];
}
```

By Delegate: 使用delegate的方式会对原有代码产生影响。

```
@protocol FMAlbumInfoManagerTestsDelegate <NSObject>
-(void)asyncRefreshAlbumInfoFullfill:(BOOL)succ;
-(void)asyncRefreshAlbumInfoNotEmpty:(BOOL)empty;
-(void)asyncRefreshAlbumInfoNotifFullfill;
@end
```

```
- (void)testRefreshAlbumInfoWithSourceInfoByDelegate {
    self.manager.testDelegate = self;
    self.exp1 = [[XCTestExpectation alloc] initWithDescription:@"Delegate异步测试：测试拉取专辑数据，请求是否成功"];
    self.exp1.expectedFulfillmentCount = 2;
    [self.manager refreshAlbumInfoWithSourceInfo:@""];
    [self waitForExpectations:@[self.exp1, self.exp2] timeout:2];
}

#pragma -mark test helpers
-(void)asyncRefreshAlbumInfoFullfill:(BOOL)succ {
    if(succ){
        [self.exp1 fulfill];
    }else{
        XCTFail(@"拉取出错了");
    }
}

-(void)asyncRefreshAlbumInfoNotEmpty:(BOOL)empty {
    if(!empty){
        [self.exp2 fulfill];
    }else{
        XCTFail(@"拉取到了空专辑");
    }
}
```

failed - 拉取出错了

[WWDC 2017NEW] XCTWaiter

WWDC2017中引入了XCTWaiter，简单来说就是通过delegate的方式把处理XCTExpectation的方法解耦，可以在delegate中处理超时，中断等异步测试用例的异常，关于Notification, Predicate, KVO的Expectation可以根据实际情况使用

```
//XCTWaiter factor out logic by delegate
- (void)testRefreshAlbumInfoWithSourceInfoByXCTWaiter {
    self.manager.testDelegate = self;
    XCTWaiter *waiter = [[XCTWaiter alloc] initWithDelegate:self];
    self.exp1 = [[XCTestExpectation alloc] initWithDescription:@"XCTWaiter异步测试：测试拉
        取专辑数据，请求是否成功"];
    self.exp2 = [[XCTestExpectation alloc] initWithDescription:@"XCTWaiter异步测试：测试拉
        取专辑数据，专辑节目数是否为空"];
    [self.manager refreshAlbumInfoWithSourceInfo:@""];
    //by using the new XCTWaiter class with a nil XCTWaiterDelegate, you may receive a
        result enum instead of failure
    XCTWaiterResult result = [waiter waitForExpectations:@[self.exp1, self.exp2]
        timeout:2];
    XCTAssertEqual(result, XCTWaiterResultIncorrectOrder);
}

-(void)waiter:(XCTWaiter *)waiter didTimeoutWithUnfulfilledExpectations:(NSArray<
    XCTestExpectation *> *)unfulfilledExpectations {
    //time out
}
```

5.2.3. 加上OCMock完成的Mock测试

我们要测试的方法会引用很多外部依赖的对象，而我们没法控制这些外部依赖的对象。为了解决这个问题，我们需要用到Stub和Mock来模拟这些外部依赖的对象,从而控制它们。单独依靠XCTest难以完成Mock或者Stub，但是结合OCMock可以在测试代码中实现这以下功能。

*** Mock: 创建一个模拟对象，我们可以验证，修改它的行为**

*** Stub: Mock对象的函数返回特定的值**

*** Partial Mock: 重写Mock对象的方法**

例：简单声明了一个类，再使用不同方式对它进行Mock：

```
@interface FMTestExample : NSObject

-(int)example1:(NSNumber *)input;
-(int)callExample1;

@end
```

```
#import "FMTestExample.h"
@implementation FMTestExample

-(int)example1:(NSNumber *)input {
    return (int)[input integerValue];
}

-(int)callExample1 {
    return [self example1:@(1)];
}
@end
```

(5.2.3.1)Method Mock:

```
-(void)testMethodMock {
    id exMock = [OCMockObject mockForClass:[FMTestExample class]];
    [[[exMock stub] andReturnValue:OCMOCK_VALUE((int){10})] example1:[OCMArg any]];
    int mockRet = [exMock example1:@(10)]; //mockRet = 10
    [exMock stopMocking];
}
```

以上代码表示这个exMock对象，的example1方法无论接受什么参数，都只会返回10

(5.2.3.2)Partial Mock or Protocol Mock

```
-(void)testPartialMock {
    FMTestExample *ex = [[FMTestExample alloc] init];
    id exMock = [OCMockObject partialMockForObject:ex];
    [[[exMock stub] andReturnValue:OCMOCK_VALUE((int){10})] example1:[OCMArg any]];

    int mockRet = [exMock example1:@(10)]; //mockRet = 10
    int objRet = [ex example1:@(10)];      //objRet = 10;
    [exMock stopMocking];
}
```

上面代码表示这个exMock对象，的example1方法无论接受什么参数，都只会返回10，并且实际的对象ex调用example1也只会返回10

*也可以创造遵循某个protocol的mock对象

(5.2.3.3)Delegate to Other Method or Block

```
-(void)testDelegateMethod {
    id exMock = [OCMockObject mockForClass:[FMTestExample class]];
    [[[exMock stub] andCall:@selector(__switchExample1:) onObject:self] example1:[OCMArg
        any]];
    int mockRet = [exMock example1:@(10)]; // mockRet = 20;
    [exMock stopMocking];
}

-(int)__switchExample1:(NSNumber *)input {
    return (int)[input integerValue] * 2;
}
```

上代码表示这个exMock对象的example1方法已经用__switchExample1方法替换掉了，所以当调用[exMock example1:@(10)]的时候返回值为20

(5.2.3.4)是三种OCMock主要的Mock方法

利用它们我们可以在用例中排除一些外部类的干扰因素，构造自己的用例来进行验证，需要注意的是 `mock` 的对象在用例结束后要 `stopMocking` ，避免由于单例或者 `property` 导致的用例之间相互影响

(5.2.3.5) Expect-run-verify

```
-(void)test4 {
    //setup mock
    FMTestExample *ex = [[FMTestExample alloc] init];
    id exMock = [OCMockObject partialMockForObject:ex];

    //expect
    [[exMock expect] example1:[OCMArg any]];

    //innovation
    [exMock callExample1];
    //verify
    [exMock verify];
    //stop
    [exMock stopMocking];
}
```

OCMock有一个Expect-Run-Verify的模式，能够帮助我们验证是否期望的方法有被调用。上面的代码中，exMock期望example1会被调用，没有对参数做任何的限制。而在callExample1中假如像期望一样调用了example1，该用例会通过，否则失败。

这个模式在某些场景会适用到，比如验证app启动后是否进行一些必须做的一些操作和配置等。

六、Reference

(6.1)[iOS单元测试系列]单元测试框架选型:

http://zixun.github.io/blog/2015/04/11/iosdan-yuan-ce-shi-xi-lie-dan-yuan-ce-shi-kuang-jia-xuan-xing/

(6.2)XCTest Documentation:
https://developer.apple.com/documentation/xctest

(6.3)XCTest实战: https://objccn.io/issue-15-2/

(6.4)Migrate to XCTWaiter for async tests?:
https://github.com/Instagram/IGListKit/issues/610

(6.5)What's New in Testing:
https://developer.apple.com/videos/play/wwdc2017/409/

(6.6)Kiwi 使用进阶Mock, Stub, 参数捕获和异步测试:
https://onevcat.com/2014/05/kiwi-mock-stub-test/

(6.7)OCMock Documentation: http://ocmock.org/