会写代码的猪

没谱青年，资深80后，大爱加菲猫，不炫技会死！

iOS开发中的单元测试（二）

作者：gaosboy | 时间：June 30, 2013 | 分类：

本文首发于 InfoQ中文站

上一篇文章简单介绍了OCUnit和GHUnit两款iOS开发中较为常见的单元测试框架，本文进一步介绍单元测试中的另一利器——匹配引擎（Matcher Engine）。匹配引擎可以替代断言方法，配合单元测试引擎使用，测试用例可以更多样化，更细致。

传统断言提供的方法数量和功能都有限，以导读中提到的两款框架为例，即使是断言相对丰富的GHUnit也只是提供了38种断言方法，范围仅涵盖了逻辑比较，异常和出错等少数几方面，仍然很单一。而使用匹配引擎代替断言，可能性就大大丰富了，除了普通断言支持的规则，一般的引擎还默认提供了包含，区间，继承关系等。更重要的是，使用匹配引擎开发者可以自行开发匹配规则，引入与业务相关的逻辑判断。

本文要介绍两款匹配引擎，一款就是Hamcrest的Objective-C实现——OCHamcrest，另一款则是专为Objective-C/Cocoa而生的后来者——Expecta。接下来将结合GHUnitTest，介绍两款匹配引擎如何在单元测试中发挥作用（有关GHUnitTest参考《iOS开发中的单元测试（一）》。

OCHamcrest

介绍匹配引擎必须要提Hamcrest，几乎已经成为匹配引擎的代名词。官网首页上的一句话表明了它的身世：“Born in Java, Hamcrest now has implementations in a number of languages.”。这款诞生于Java的匹配引擎现在还支持除Java的Python、Ruby、PHP、Erlang和Objective-C。

- 加入工程

在iOS工程中使用OCHamcrest需要先获取OCHamcrestIOS.framework，可以从Quality Coding直接下载，或在Github上获取源码编译。注意：Github上托管的OCHamcrest工程以Submodule的形式关联源代码，因此如果使用命令行方式clone工程，需要执行“git submodule update --init”。

下载源码后，进入Source目录，执行MakeDistribution.sh脚本，将会在Source/build/Release下生成OCHamcrest.framework、OCHamcrestIOS.framework和OCHamcrest.framework.dSYM，OCHamcrestIOS.framework就是iOS工程中需要用到的框架，如图1。

```
# git clone https://github.com/hamcrest/OCHamcrest.git
Cloning into 'OCHamcrest'...
remote: Counting objects: 8724, done.
remote: Compressing objects: 100% (3365/3365), done.
remote: Total 8724 (delta 5189), reused 8655 (delta 5125)
```

图1，从源码编译生成 OCHamcrestIOS.framework

打开已经安装了GHUnitTest的工程，把OCHamcrestIOS.framework添加到单元测试的Target中。在需要使用匹配引擎的用例中，定义“HC_SHORTHAND”并导入“<OCHamcrestIOS/OCHamcrestIOS.h>”（如图2）。

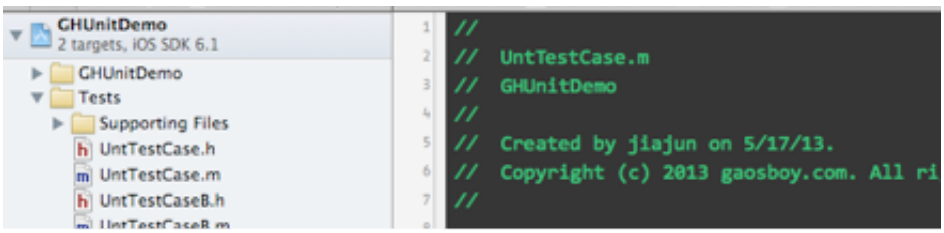

图2，把OCHamcrestIOS.framework导入工程

至此OCHamcrest已经安装完成，可以再测试用例中使用匹配规则代替GHUnitTest的断言方法。

- 预定义规则

OCHamcrest针对不同的数据类型提供了大量的预定义匹配规则，大大丰富了断言的类型。支持的数据类型包括：对象、容器、数值和文本，此外还提供了专门的逻辑匹配规则。

以文本（一般就是NSString）为例，OCHamcrest提供了6种针对对象的匹配规则：

> IsEqualIgnoringCase，该文本是否与给出的文本相同（忽略大小写）；
>
> IsEqualIgnoringWhiteSpace，该文本是否与给出的文本相同（忽略空格）；
>
> StringContains，该文本是否包含给出的文本片段；
>
> StringContainsInOrder，该文本是否按照先后顺序包含给出的若干文本片段；
>
> StringEndsWith，该文本是否以给出的文本片段结尾；
>
> StringStartsWith，该文本是否以给出的文本片段开头。

另外，再举OCHamcrest为对象（NSObject和NSObject的子类）预定义的8条规则：

> ConformsToProtocol，该对象是否遵循了给出的协议，或者说是否实现了给出的Delegate；
>
> HasDescription，允许使用文本规则对给出的一段文本与该对象的描述进行匹配；
>
> HasProperty，该对象是否含有给出的属性；
>
> IsInstanceOf，是给出的类的实例，或是给出的类子类的实例；
>
> IsTypeOf，是给出的类的实例，不同于IsInstanceOf，无法匹配子类实例；
>
> IsNil，为空；
>
> IsSame，与给出的对象是同一个实例。

- 撰写用例

OCHamcrest提供了匹配规则和相应的断言方法，配合单元测试框架（本文以GHUnit为例， 在《iOS开发中的单元测试（一）》中已经介绍了如何安装GHUnit框架并撰写用例）的驱动机制即可撰写用例。本文以联合使用上述提到的StringStartsWith和HasDescription规则为例。

首先，定义一个用于示例的类“Man”（如图3），有属性friends，当friends为空，其description为“Man without any friend, so sorry.”，反之为“Nice persion with [friends count] friend(s).”。（使用Foo或Bar这样的示例会显得很没情怀吧 ;-|）

```
@interface Man : NSObject

@property (strong, nonatomic)   NSMutableArray  *friends;

@end

@implementation Man

- (NSString *)description
{
    if (nil == self.friends || 0 >= [self.friends count]) {
        return @"Man without any friend, so sorry.";
    }
    return [NSString stringWithFormat:@"Nice persion with %d friend(s).", [self.friends count]];
}

@end
```

图3，用于测试的类：Man

用例中判断某Man实例的description是否以Nice开头（这是不是一个友善的人），如图4。

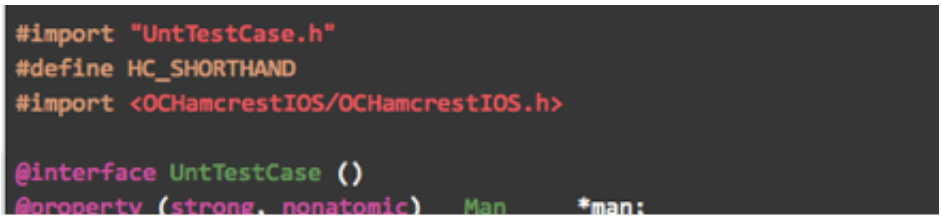

图4，测试用例两则

UntTestCase是GHTestCase的子类，引入<OCHamcrestIOS/OCHamcrestIOS.h>并定义HC_SHORTHAND表示使用OChamcrest。setUp方法在每个测试方法执行之前初始化一个Man实例；testANiceMan方法向Man实例的friends属性中加入两个值，因此该实例的description将返回“Nice man ....”；使用OCHamcrest提供的断言方法assertThat与匹配规则配合，判断该实例的description是否以“Nice”开头；testNotANiceMan方法则直接测试一个未经过加入friends的实例测试。

上述测试，testANiceMan方法顺利通过，testNotANiceMan不会通过，直接报出错误堆栈，并打印在匹配规则中预先定义好的出错信息（如图5）。

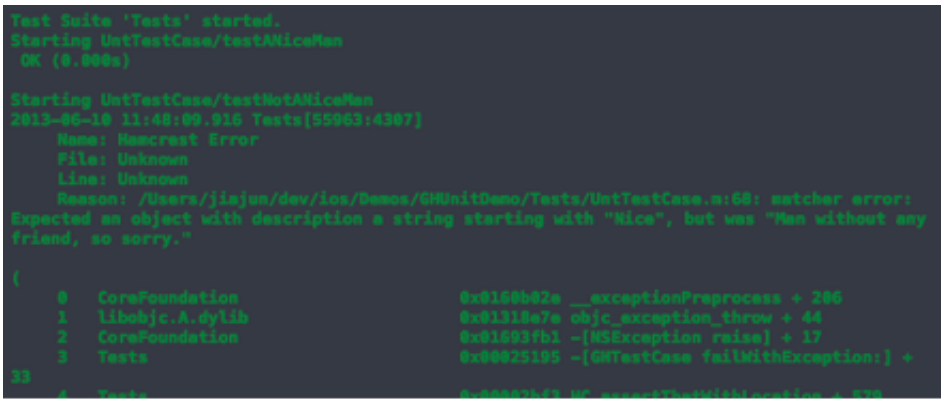

图5，测试结果

- 辅助方法

Syntactic Sugar是一种提高匹配规则和断言可读性的方案，让一个匹配和断言看起来更像是一句自然语言的话，而非多个函数的堆砌，对实际的匹配运算不产生任何影响。例如，没有加Sugar的匹配：

```
assertThat(foo, equalTo(bar));
```

加Sugar可以是：

```
assertThat(foo, is(equalTo(bar)));
```

除了Sugar，OCHamcrest还提供了describedAs方法，用于辅助断言方法，自定义出错文案，例如：

```
assertThat(foo, describedAs(@"foo should be equal to bar", equalTo(bar), nil));
```

- 自定义规则

OCHamcrest官方给出的自定义匹配规则示例是：onASaturday，判断一个NSDateComponents实例是否为星期六。本文以上一节使用的Man对象为例，匹配某Man实例是否有一个名为“Joe”的好友，规则命名为：hasAFriendJoe。

自定义匹配规则包括两部分，一个Macher类和一个用OBJC_EXPORT方式定义的函数。

自定义Macher类都是HCBaseMatcher的子类（如图6），接口中定义的类初始化方法供匹配方法hasAFriendJoe调用，其实现则通过调用接口中定义的另一个实例方法。

```
#import <OCHamcrestIOS/HCBaseMatcher.h>
#import <objc/objc-api.h>

@interface HasAFriend : HCBaseMatcher

@property (strong, nonatomic)   NSString        *name;
```

图6，HasAFriend接口和匹配方法hasAFriendJoe定义

在HasAFriend中需要引入<OCHamcrestIOS/HCDescription.h>，并重写父类中的matches:和describeTo:方法（如图7）。在maches:方法中实现匹配逻辑，匹配成功则返回YES，否则返回NO；describeTo是失败后的描述；hasAFriendJoe方法只需要调用类方法初始化匹配规则类即可。匹配规则定义后，可以配合断言方法使用，如上一节所示的assertThat方法：

assertThat(self.man, hasAFriendJoe());

图7，规则实现

Expecta

Expecta专为Objective-C/Cocoa而生，相比OCHamcrest，其优化了匹配的语法，测试用例的可读性更高。此外，Expecta对匹配对象类型没有强制要求，允许任意类型的数据进行匹配。在OCHamcrest中每一条匹配规则都是一个方法，规则联合使用也需要以参数形式传递。在Expecata中联合规则的语法是以点号连接，借助Sugar介词可以把一个联合规则拼装成一句符合自然语法的句子，例如：

```
OCHamcrest -- assertThat(@"foo", is(equalTo(@"foo")));
Expecta -- expect(@"foo").to.equal(@"foo");
```

- 加入工程

Expecta提供了CocoaPods的源，可以通过定义依赖引入：

```
dependency 'Expecta', '~> 0.2.1'

dependency 'Specta','~>0.1.7'#specta bdd framework
```

或者从github上获取源代码，编译出Library，引入XCode工程。下载源码后，进入工程目录，运行rake，编译工程。编译完成后，把products目录拷贝到工程中（如图8），在iOS/MacOSX工程中加入响应的.a文件（如图9）。在Build Settings的Other Linker Flags中加入-ObjC参数（在《iOS开发中的单元测试（一）》中添加GHUnit一节介绍了如何添加-ObjC的参数）。与OCHamcrest类似，在测试用例中定义EXP_SHORTHAND，并引入“Expecta.h”（如图10）。

图8，加入编译后的头文件列表

图9，加入Library文件

图10，用例中引入Expecta

- 预定义规则

Expecta提供的预定义规则只有20条，远远少于OCHamcrest提供的预定义规则。由于Expecta的匹配规则对匹配对象没有要求，因此没有提供像OCHamcrest中针对某种对象的特定规则。

在Expecta的github首页可以看到全部预定义规则列表。举几个较有特点的规则为例：

```
expect(x).to.beCloseToWithin(y, z)，x距离y的距离小于z
expect(x).to.beTruthy()， x是否为真（或非空）；
expect(x).to.beFalsy()，x是否为假（或空/零）；
expect(^{ /* code */ }).to.raiseAny()，该Block是否抛出异常；
expect(^{ /* code */ }).to.raise(@"ExceptionName")，该Block是否抛出给定名字异常。
```

此外，通过.notTo或.toNot对规则取反进行匹配，如：expect(x).notTo.equal(y)。

通过.will或.willNot进行异步匹配，即在超时时间（默认超时1秒，也可通过[Expecta setAsynchronousTestTimeout:x]设定）之前满足匹配规则即可，如：expect(x).will.beNil()。

- 撰写用例

Expecta不使用类似assertThat类似的辅助断言方法，而是直接使用expecta.语法匹配。

仍然以GHUnit测试用例为例，测试一个数字n，是否在5附近，距离小于2，即处在[3,7]区间内（如图11）。

图11，Expecta测试用例

Expecta不支持匹配规则的联合使用。

- 辅助方法

Expecta也有语法Sugar：to。

- 自定义规则

Expecta的自定义规则有两种方式，静态规则和动态规则。

定义静态规则：

Expecta的匹配规则不是一个类，是通过框架提供的宏定义来实现的，操作比定义OCHamcrest规则简单不少。仍以OCHamcrest中的判断一个Man实例是否有名为“Joe”的好友。

通过EXPMatcherInterface()方法定义，该方法有两个参数，规则名和规则参数列表。示例如图12。

图12，扩展规则hasAFriendJoe定义

EXPMacherInterface第二个参数允许通过这样的方式定义列表：(NSString *Foo, int bar)。

在实现中，用EXPMatcherImplementationBegin和EXPMatcherImplementationEnd标示规则实现的头尾。并定义prerequisite、match、failureMessageForTo和failureMessageForNotTo四个Block，分别返回与判断结果，匹配结果，正向匹配出错原因和反相匹配出错原因（如图13）。由于Expecta框架不支持ARC，因此需要在Build Settings中对该.m文件添加 -fno-objc-arc参数。在测试用例中可以通过如下语法调用：

```
expect(self.man).hasAFriend(@"Joe");
```

或反相匹配：

```
expect(self.man).notTo.hasAFriend(@"Joe");
```

图13，Expecta自定义规则实现

定义动态规则：

动态规则是本质上并不是一段逻辑匹配，而是通过Expecta的语法对匹配对象的属性进行是否为真的断言。例如：

```
@interface LightSwitch : NSObject
@property (nonatomic, assign, getter=isTurnedOn) BOOL turnedOn;
@end
@implementation LightSwitch
@synthesize turnedOn;
@end
```

可以写出如下断言：

```
expect([lightSwitch isTurnedOn]).to.beTruthy();
```

建立动态规则：

```
EXPMatcherInterface(isTurnedOn, (void));
```

就可以通过以下断言判断turendOn属性的真假：

```
expect(lightSwitch).isTurnedOn();
```

总结

整体看两款匹配引擎，Expecta小巧，敏捷，提供了多种灵活的匹配方式，OCHamcrest从Hamecrest体系继承而来，形式更加中庸，提供的机制更完善。从开发者的角度看，Expecta更好玩，而OCHamcrest更实用，在实验性的项目中我会偏向选择Expecta，而较正式的项目则会使用OCHamcrest。

OCHamcrest结合上一篇《iOS开发中的单元测试（一）》中介绍的单元测试框架GHUnit可以给开发者提供一个完整的单元测试方案，建议开发者在自己的项目中引入这样的质量自控机制，写出健壮的代码。

通过两篇文章介绍了单元测试框架和匹配引擎的一些基础知识，在接下来的文章中，我将结合一个项目，从实战角度详细记述如何开发带有单元测试的iOS项目。

标签：ios

添加新评论

称呼 *

Email *

网站

http://

内容 *

提交评论