设计模式 For iOS

第 01 式

观察者模式

译者：BeyondVincent(破船)

时间：2013.05.06

版本： 1.0

关于破船

程序猿砌墙于云南昆明!

长期扎根移动软件开发!

爱跑步爱打篮球爱运动!

命中无大富大贵之面相!

愿健康与平淡相随一生!

你可以发邮件与破船取得联系: BeyondVincent@gmail.com

还可以关注破船的微博: 腾讯微博和新浪微博。

这里是破船的个人博客，欢迎光临：破船之家

关于设计模式 For iOS 的整理

本系列文章，主要是学习设计模式在 iOS 中的实现过程中，写出来的。期间参考了许多互联网上的资料。如有不正确的地方，还请读者指正。

本系列全部文章和相关代码都可以在下面的链接中下载到：

https://github.com/BeyondVincent/ios_patterns

破船祝你阅读愉快！

目录

第 01 式 观察者模式

1.0. 简介

1.0.1. 什么是观察者模式

什么是观察者模式？你曾经订阅过报纸吗？在订阅报纸的时候，你不用去任何地方，只需要将你的个人地址信息以及订阅信息告诉出版社，出版社就知道如何将相关报纸传递给你。这种模式的第二个名称叫做发布/订阅模式。

在 GoF 中是这样描述观察者模式的——观察者模式定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题对象。这个主题对象在状态上发生变化时，会通知所有观察者对象，使它们能够自动更新自己。

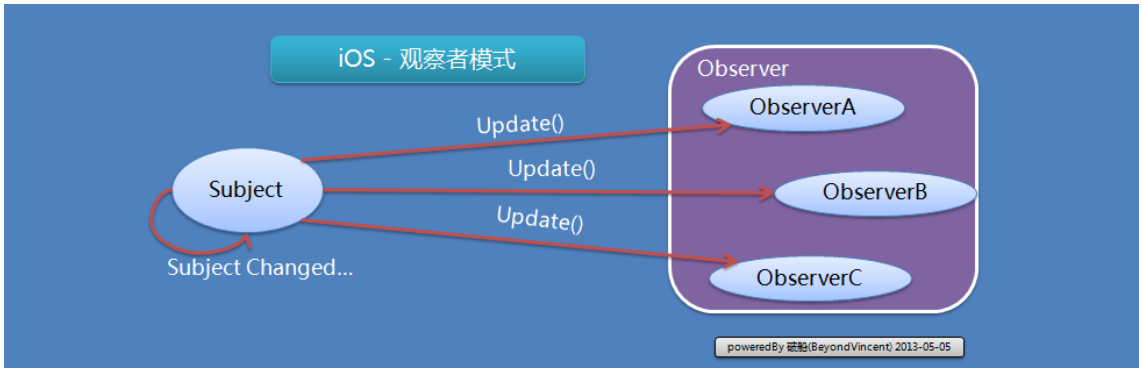
观察者模式的的思想非常简单，Subject（主题）允许别的对象——观察者（这些对象实现了观察者接口）对这个 Subject 的改变进行订阅和取消订阅。当 Subject 发生了变化——那么 Subject 会将这个变化发送给所有的观察者，观察者就能对 Subject 的变化做出更新。在这里，Subject 是报纸的出版社，而观察者则是订阅报纸的我和你，当 Subject 发生变化——有新的报纸，会做出通知——将报纸发送给

所有的订阅者。

1.0.2. 什么时候使用观察者模式？

1)当你需要将改变通知所有的对象时，而你又不知道这些对象的具体类型，此时就可以使用观察者模式。

2)改变发生在同一个对象中，并在别的地方需要将相关的状态进行更新。



[点击图片看大图]

1.1.iOS 中观察者模式的实现方法

在 iOS 中观察者模式的实现有三种方法：

1.1.1. NOTIFICATION

Notification - NotificationCenter 机制使用了操作系统的功能。通过 NSNotificationCenter 可以让对象之间进行进行通讯，这些对象相互间可以不认识。当你用一个并行的流来推送通知，或者刷新数据库，并希望在界面中能够看到时，

这非常有用。

NotificationCenter 发布消息的方法如下所示：

```
NSNotification * broadcastMessage = [ NSNotification notificationWithName: AnyNotification object: Self ];
NSNotificationCenter * notificationCenter = [ NSNotificationCenter defaultCenter];
[NotificationCenter postNotification: broadCastMessage];
```

上面的代码中，创建了一个 NSNotification 类型的对象，并指定名称为”broadcastMessage”，然后通过 notificationCenter 来发布这个消息。

要订阅感兴趣的对象中的相关事件，可以按照如下方法进行：

```
NSNotificationCenter * notificationCenter = [ NSNotificationCenter defaultCenter];
[NotificationCenter addObserver: Self selector: @ selector (update:) name: AnyNotification object: nil ];
```

如上代码所示：订阅了一个事件，并通过@selector 指定了一个 update:方法。

```
// 收到通知中心发来的通知
-(void)update:(NSNotification *) notification
{
    if ([[notification name] isEqualToString:AnyNotification])
        NSLog (@"成功收到通知中心发来的名为%@的通知", AnyNotification);
}
```

下面是运行上面代码，在控制台输出的内容：

```
2013-05-05 23:43:15.570 ObserverPattern[1738:c07] 成功收到通知中心发来的名为 broadcastMessage 的通知
```

1.1.2. KVO

通过 KVO，某个对象中的特定属性发生了改变，别的对象可以获得通知。苹果官方文档对 KVO 有了很好的解释：Key-Value Observing Programming Guide。下面两种方法都可以改变对象中属性的值：

```
kvoSubj.changeableProperty = @"新的一个值";
[kvoSubj setValue:@"新的一个值" forKey:@"changeableProperty"];
```

上面这种值改变的灵活性可以让我们对键值进行观察。

下面是新建的一个类 KVOSubject，这个类中有一个属性 changeableProperty：

```
@interface KVOSubject : NSObject

@property (nonatomic, strong) NSString *changeableProperty;

@end
```

```
@implementation KVOSubject

@end
```

接着新建了另外一个类 KVOObserver，通过该类可以监听 changeableProperty 属性值的改变。

```
@interface KVOObserver : NSObject
@end

@implementation KVOObserver
```

```
-(void) observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary *)change
context:(void *)context
{
    NSLog(@"KVO:值发生了改变");
}

@end
```

如上代码所示，KVOObserver 类只有一个方法 observeValueForKeyPath。当 changeableProperty 属性值的改变时，这个方法会被调用。下面是测试的代码：

```
- (IBAction)btnKVOObservationTest:(id)sender {
    KVOSubject *kvoSubj = [[KVOSubject alloc] init];
```

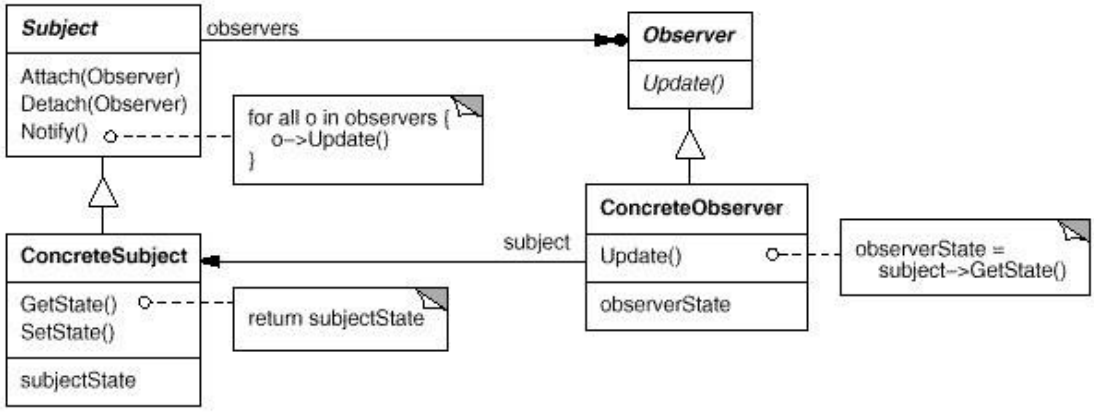
```
    KVOObserver *kvoObserver = [[KVOObserver alloc] init];

    [kvoSubj addObserver:kvoObserver forKeyPath:@"changeableProperty"
                 options:NSKeyValueObservingOptionNew context:nil];

    kvoSubj.changeableProperty = @"新的一个值";

    [kvoSubj setValue:@"新的一个值" forKey:@"changeableProperty"];

    [kvoSubj removeObserver:kvoObserver forKeyPath:@"changeableProperty"];
}
```

执行上面的代码，可以看到控制台输出如下结果：

```
2013-05-05 23:10:20.789 ObserverPattern[1358:c07] KVO:值发生了改变
2013-05-05 23:10:20.790 ObserverPattern[1358:c07] KVO:值发生了改变
```

1.1.3. 标准方法

先来看看 Gof 中对观察者模式定义的结构图：



标准方法的实现是这样的：Subject（主题）知道所有的观察者，但是不知道它们的类型。下面我们就从创建 Subject 和 Observer（观察者）的协议（protocol）开始。

```
@protocol StandardObserver
```

```
-(void) valueChanged:(NSString *)valueName newValue:(NSString *) newValue;
@end
```

```
@protocol StandardSubject
-(void) addObserver:(id) observer;
-(void) removeObserver:(id) observer;
-(void) notifyObjects;
@end
```

下面，我们来创建一个 Subject 的 implementation （实现）

```
@interface StandardSubjectImplementation : NSObject
{
    @private NSString *_valueName;
    @private NSString *_newValue;
}
@property (nonatomic, strong) NSMutableSet *observerCollection;
-(void)changeValue:(NSString *)valueName andValue:(NSString *) newValue;
@end
```

```
@implementation StandardSubjectImplementation

-(NSMutableSet *) observerCollection
{
    if (_observerCollection == nil)
        _observerCollection = [[NSMutableSet alloc] init];

    return _observerCollection;
}

-(void) addObserver:(id)observer
{
    [self.observerCollection addObject:observer];
}

-(void) removeObserver:(id)observer
{
    [self.observerCollection removeObject:observer];
}

-(void) notifyObjects
{
    for (id observer in self.observerCollection) {
        [observer valueChanged: _valueName newValue:_newValue];
    }
}
```

```
-(void)changeValue:(NSString *)valueName andValue:(NSString *) newValue
{
    _newValue = newValue;
    _valueName = valueName;
    [self notifyObjects];
}
@end
```

接下来是 Observer 的 implementation （实现）:

```
@interface SomeSubscriber : NSObject
@end
```

```
@implementation SomeSubscriber
-(void) valueChanged:(NSString *)valueName newValue:(NSString *)newValue
{
    NSLog(@"SomeSubscriber 输出: 值 %@ 已变为 %@", valueName, newValue);
}
@end

@interface OtherSubscriber : NSObject

@end

@implementation OtherSubscriber

-(void) valueChanged:(NSString *)valueName newValue:(NSString *)newValue
{
    NSLog(@"OtherSubscriber 输出: 值 %@ 已变为 %@", valueName, newValue);
}
@end
```

下面是演示的代码：

```
StandardSubjectImplementation * subj = [[StandardSubjectImplementation alloc] init];
SomeSubscriber * someSubscriber = [[SomeSubscriber alloc] init];
OtherSubscriber * otherSubscriber = [[OtherSubscriber alloc] init];

[Subj addObserver: someSubscriber];
[Subj addObserver: otherSubscriber];

[subj changeValue:@"version" andValue:@"1.0.0"];
```

上面代码运行的 log 如下所示：

```
2013-05-05 23:19:04.662 ObserverPattern[1459:c07] OtherSubscriber 输出: 值 version 已变为 1.0.0
2013-05-05 23:19:04.664 ObserverPattern[1459:c07] SomeSubscriber 输出: 值 version 已变为 1.0.0
```

1.2. 代码下载地址

点击如下图标，浏览并下载本文全部代码。

1.3. 参考

本文参考了如下文章：

1)When to use Delegation, Notification, or Observation in iOS

2)ios-patterns-observer

3)维基百科：观察者模式

感谢你的阅读！

如果对这篇文章有什么想法，可以与破船联系，破船的联系方式在文章开头。

破船