设计模式 For iOS

第 03 式

工厂方法

整理：BeyondVincent(破船)

时间：2013.05.16

目录

第 03 式 工厂方法

1.0. 简介

1.0.1. 什么是工厂方法

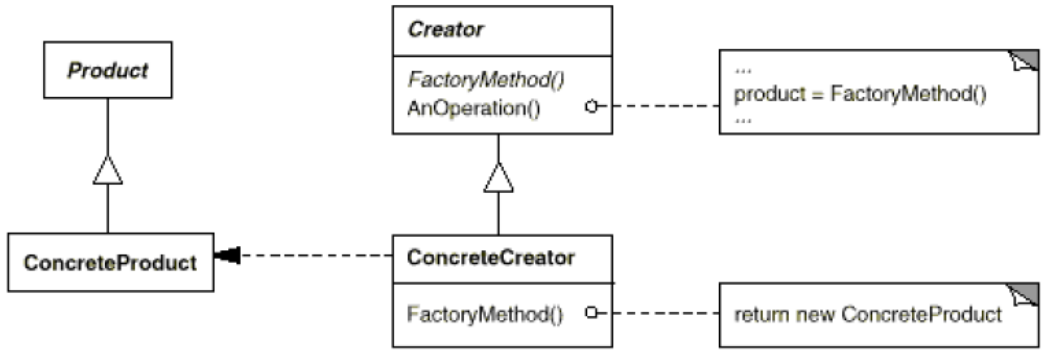
GOF 是这样描述工厂模式的：

Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

在基类中定义创建对象的一个接口，让子类决定实例化哪个类。工厂方法让一个类的实例化延迟到子类中进行。

工厂方法要解决的问题是对象的创建时机，它提供了一种扩展的策略，很好地符合了开放封闭原则。工厂方法也叫做虚构造器（Virtual Constructor）。

如下图所示，是工厂方法的类结构图：



1.0.2. 什么时候使用工厂方法

当是如下情况是，可以使用工厂方法：

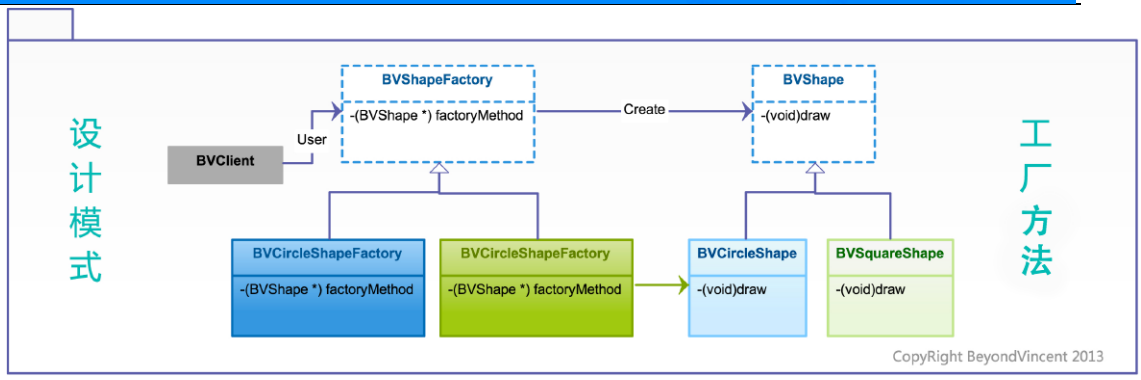
1.一个类不知道它所必须创建的对象的类时

2.一个类希望有它的子类决定所创建的对象时

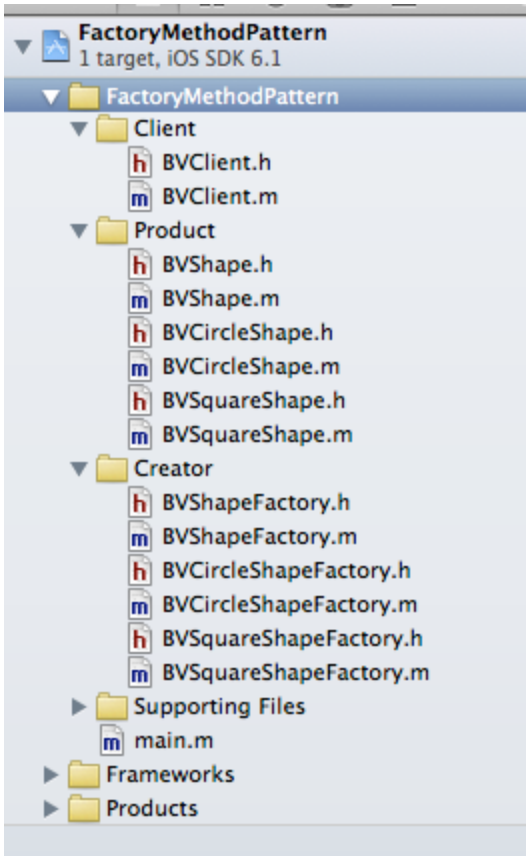
更多关于工厂方法的介绍，可以参考本文最后给出的参考内容。下面我们就来看看在 iOS 中工厂方法的一种实现方法。

1.1. iOS 中工厂方法的实现

如下类图所示，该图描述了下面即将实现的工厂方法（利用工厂方法，创建出不同的形状）。其中 BVShapeFactory 为工厂方法的基类，BVShape 为形状的基类，BVClient 通过 BVShapeFactory，利用 BVShapeFactory 的子类（BVCircleShapeFactory 和 BVSquareShapeFactory）分别创建出 BVCircleShape 和 BVSquareShape。



如下图，是在 Xcode 中创建的相关文件



具体实现步骤如下：

1. 创建一个形状基类 BVShape

该类中定义了形状的基本行为和属性，如下代码所示：

BVShape.h

```
//
//  BVShape.h
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//

#define BV_Exception_Format @"在%@的子类中必须 override:%@方法"

@interface BVShape : NSObject

@property (nonatomic, weak)NSString *name;

// 子类必须重写这个 draw 方法，否则会抛出异常错误
-(void)draw;

@end
```

BVShape.m

```
//
//  BVShape.m
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//

#import "BVShape.h"

@implementation BVShape

-(id)init
{
    self = [super init];
     if (self) {
          // 做一些初始化任务
     }
     return self;
}

-(void)draw
{
     // 如果是通过 BVShape 的实例调用此处的 draw，则绘制一个 BVShape 图形
     if ([self isMemberOfClass:[BVShape class]]) {
```

```
        NSLog(@"绘制一个 BVShape 图形");
    } else {
        // 如果是通过 BVShape 子类的实例调用了此处的 draw，则抛出一个异常：表明子类并没有重写 draw 方法。
        // 注：在 OC 中并没有 abstract class 的概念，只有 protocol，如果在基类中只定义接口(没有具体方法的实现)，
        //      则可以使用 protocol，这样会更方便。
        [NSException raise:NSInternalInconsistencyException
                    format:BV_Exception_Format, [NSString stringWithUTF8String:object_getClassName(self)], NSStringFromSelector(_cmd)];
    }
}

@end
```

在上面的代码中定义了一个 draw 方法，为了让子类必须实现该方法，在 BVShape 中做了特殊处理，具体内容可以看上面的代码，已经有注视了。

2.子类化形状基类

首先子类化一个圆形类：BVCircleShape。

BVCircleShape.h

```
//
//  BVCircleShape.h
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//

#import "BVShape.h"

@interface BVCircleShape : BVShape

@end
```

BVCircleShape.m

```
//
//  BVCircleShape.m
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//
```

```
#import "BVCircleShape.h"

@implementation BVCircleShape

-(void)draw
{
    NSLog(@"绘制一个 BVCircleShape 图形");
}

@end
```

在上面的子类中，重写了基类的 draw 方法。同样，我们再子类化一个正方形类，并重写 draw 方法，如下代码所示：

BVSquareShape.h

```
//
//  BVSquareShape.h
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年  BeyondVincent. All rights reserved.
//

#import "BVShape.h"

@interface BVSquareShape : BVShape

@end
```

BVSquareShape.m

```
//
//  BVSquareShape.m
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年  BeyondVincent. All rights reserved.
//

#import "BVSquareShape.h"

@implementation BVSquareShape

-(void)draw
{
    NSLog(@"绘制一个 BVSquareShape 图形");
```

```
}

@end
```

3.创建一个工厂方法的基类 BVShapeFactory

BVShapeFactory.h

```
//
//  BVShapeFactory.h
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年  BeyondVincent. All rights reserved.
//

#import "BVShape.h"

@interface BVShapeFactory : NSObject

-(BVShape *) factoryMethod;

@end
```

BVShapeFactory.m

```
//
//  BVShapeFactory.m
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年  BeyondVincent. All rights reserved.
//

#import "BVShapeFactory.h"

@implementation BVShapeFactory

-(BVShape *)factoryMethod
{
    // 在此处，子类必须重写 factoryMethod 方法。当然，在工厂模式中，也可以在此处返回一个默认的 Product。
    // 如果是通过 BVShapeFactory 子类的实例调用了此处的 factoryMethod，则抛出一个异常：表明子类并没有重写 factoryMethod 方法。
    [NSException raise:NSInternalInconsistencyException
                format:BV_Exception_Format, [NSString stringWithUTF8String:object_getClassName(self)], NSStringFromSelector(_cmd)];

    // 下面这个 return 语句只是为了消除警告，实际上永远都不会执行到这里。
    return nil;
```

```
}

@end
```

在上面的代码中，定义了一个 factoryMethod，该类的子类必须实现该方法，通过实现该方法，返回一个具体的形状对象。下面来看看该类的子类化。

4.子类化工厂方法的基类

首先子类化一个圆形工厂方法 BVCircleShapeFactory：

BVCircleShapeFactory.h

```
//
//  BVCircleShapeFactory.h
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//

#import "BVShapeFactory.h"
#import "BVCircleShape.h"

@interface BVCircleShapeFactory : BVShapeFactory

@end
```

BVCircleShapeFactory.m

```
//
//  BVCircleShapeFactory.m
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//

#import "BVCircleShapeFactory.h"

@implementation BVCircleShapeFactory

-(BVShape *)factoryMethod
{
    return [[BVCircleShape alloc] init];
}
```

```
@end
```

如上代码所示，重写了 factoryMethod，返回一个 BVCircleShape 实例。下面来看看另外一个子类 BVSquareShapeFactory：

BVSquareShapeFactory.h

```
//
//  BVSquareShapeFactory.h
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//

#import "BVShapeFactory.h"
#import "BVSquareShape.h"

@interface BVSquareShapeFactory : BVShapeFactory

@end
```

BVSquareShapeFactory.m

```
//
//  BVSquareShapeFactory.m
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//

#import "BVSquareShapeFactory.h"

@implementation BVSquareShapeFactory

-(BVShape *)factoryMethod
{
    return [[BVSquareShape alloc] init];
}

@end
```

该子类返回的是一个 BVSquareShape 实例。

5.工厂方法的使用

定义一个 BVClient 类，在该类中演示工厂方法的使用。代码如下：

BVClient.h

```
//
//  BVClient.h
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//

@interface BVClient : NSObject

-(void)doSomething;

@end
```

BVClient.m

```
//
//  BVClient.m
//  FactoryMethodPattern
//
//  Created by BeyondVincent on 13-5-15.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//

#import "BVClient.h"

#import "BVShapeFactory.h"
#import "BVCircleShapeFactory.h"
#import "BVSquareShapeFactory.h"

#import "BVShape.h"
#import "BVCircleShape.h"
#import "BVSquareShape.h"

@implementation BVClient

-(void)doSomething
{
    // 工厂方法的实例化
    BVShapeFactory *circleShapefactory = [[BVCircleShapeFactory alloc] init];
    BVShapeFactory *squareShapefactory = [[BVSquareShapeFactory alloc] init];

    // 通过工厂方法实例化对应的形状
    BVShape *circleShape = [circleShapefactory factoryMethod];
```

```
    BVShape *squareShape = [squareShapefactory factoryMethod];

    // 调用形状的方法
    [circleShape draw];
    [squareShape draw];
}

@end
```

如上代码所示，首先实例化两个工厂方法，并通过工厂方法创建出对应的形状，最后调用形状的 draw 方法进行测试。会在控制台窗口输出如下内容：

```
2013-05-16 10:12:46.292 FactoryMethodPattern[2845:c07]  绘制一个 BVCircleShape 图形
2013-05-16 10:12:46.295 FactoryMethodPattern[2845:c07]  绘制一个 BVSquareShape 图形
```

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

在学习工厂方法时，参考了如下一些文章和视频：

工厂方法 wiki 上对工厂方法的介绍

Class Factory Methods 苹果官网对类工厂方法的介绍

iOS Patterns. Factory Method(一篇介绍工厂方法的文章，俄文，需要楼梯)

What Is the Factory Method Pattern?（Pro Objective-C Design Patterns for iOS 书中对工厂方法的介绍）

.NET 设计模式（5）：工厂方法模式（Factory Method）

java 实现工厂方法

JAVA: Factory Method Design Patternyoutube 上一个关于工厂方法的视频介绍

iphone-sdk-difference-between-iskindofclass-and-ismemberofclass iskindofclass 和 ismemberofclass 区别

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

感谢你的阅读！

如果对这篇文章有问题和建议，可以与我联系：

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