sjzLovecj
菜鸟小码农

博客园 首页 新随笔 联系 管理

随笔- 27 文章- 0 评论- 0

ios设计模式——工厂方法

工厂方法模式是抽象工厂模式的精简版本。

工厂方法模式为虚构造器，也是用于这种情况：一个类无法预期需要生成那个类的对象，但以其子类来指定所生成的对象。

工厂方法模式，定义创建一个对象的接口，让子类决定实例化哪一个类。工厂方法使得一个类的实例化延迟到其子类。

例如使用工厂方法：

1. 编译时无法准确预期需要创建的对象的类
2. 类想让其子类决定应创建对创建的小品
3. 类希望子辅助类为其子类，指定哪些实现哪个子类这一信息局部化

那么工厂方法是如何设计的呢？

工厂方法是通过子类可以重承自工厂方法创建的对象类型一些共同的行为，所以这类型活动中引入新的具体产品不需要修改客户端代码，因为这回的任何具体对象的接口都要那产品一直在用的公共的接口相同。

下面我们为创建产品类：

People类用来产属性和一个可能的无方法，同时，这个类也是其他产品类的父类。

```
//  People.h

#import <Foundation/Foundation.h>

@interface People : NSObject

@property (nonatomic, copy) NSString * name;
@property (nonatomic, copy) NSString * age;

- (id)initPeopleWithName:(NSString *)name andAge:(NSString *)age;

@end


//  People.m

#import "People.h"

@implementation People

- (id)initPeopleWithName:(NSString *)name andAge:(NSString *)age
{
    self = [super init];
    if (self)
    {
        self.name = name;
        self.age = age;
    }

    return self;
}

+ (NSString *)description
{
    return [NSString stringWithFormat:@"%@, 我的年龄是:%@", self.name, self.age];
}

@end
```

Student类继承People类，添加了一个新的属性StudentDescription，重写了初始化方法：-(id)initPeopleWithName:(NSString *)name andAge:(NSString *)age

```
//  Student.h
//
//  Created by ibeethoven.
//  Copyright (c) 2014年 ibeethoven. All rights reserved.
//

#import "People.h"

@interface Student : People
{
}
@property (nonatomic, copy) NSString * StudentDescription;

@end


//  Student.m
//
//  Created by ibeethoven.
//  Copyright (c) 2014年 ibeethoven. All rights reserved.
//

#import "Student.h"

@implementation Student

- (id)initPeopleWithName:(NSString *)name andAge:(NSString *)age
{
    self = [super initPeopleWithName:name andAge:age];
    if (self)
    {
        _StudentDescription = @"我是一个学生";
    }

    return self;
}

+ (NSString *)description
{
    return [NSString stringWithFormat:@"%@, 我的年龄是:%@, 我的学习是:%@", self.StudentDescription, self.name, self.age];
}

@end
```

Programmer类继承自People类，添加了两个属性ProgrammerDescription和workType，并重写了初始化方法：-(id)initPeopleWithName:(NSString *)name andAge:(NSString *)age，在初始化方法中，初始化方法中调用了属性赋值

```
//  Programmer.h
//
//  Created by ibeethoven.
//  Copyright (c) 2014年 ibeethoven. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "People.h"

@interface Programmer : People
{
}
@property (nonatomic, copy) NSString * ProgrammerDescription;
@property (nonatomic, copy) NSString * workType;

@end


//  Programmer.m
//
//  Created by ibeethoven.
//  Copyright (c) 2014年 ibeethoven. All rights reserved.
//

#import "Programmer.h"

@implementation Programmer

- (id)initPeopleWithName:(NSString *)name andAge:(NSString *)age
{
    self = [super initPeopleWithName:name andAge:age];

    if (self)
    {
        _ProgrammerDescription = @"我的职业工作";
        _workType = @"编程工作是我的乐趣";
    }

    return self;
}

+ (NSString *)description
{
    return [NSString stringWithFormat:@"%d, %d, 我的年龄是:%d, 我的学习是:%d", self.ProgrammerDescription, self.workType, self.name, self.age];
}

@end
```

下面我们来创建生成器方法：

Creator类为People的生成器对象，同时也也是其他生成器的父类。

```
//  Creator.h
//
//  Created by ibeethoven.
//  Copyright (c) 2014年 ibeethoven. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "People.h"

@interface Creator : NSObject

+ (People *)createPeopleWithName:(NSString *)name andAge:(NSString *)age;

@end


//  Creator.m
//
//  Created by ibeethoven.
//  Copyright (c) 2014年 ibeethoven. All rights reserved.
//

#import "Creator.h"

@implementation Creator

+ (People *)createPeopleWithName:(NSString *)name andAge:(NSString *)age
{
    return [[People alloc] initPeopleWithName:name andAge:age];
}

@end
```

我们可以为每类产品创建一个生成器：所以我们生成了CreateStudent和CreateProgrammer两个生成器分别用来生成Student和Programmer对象

CreateStudent类继承自Creator

```
//  CreateStudent.h
//
//  Created by ibeethoven.
//  Copyright (c) 2014年 ibeethoven. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "Creator.h"

@interface CreateStudent : Creator
{
}

@end


//  CreateStudent.m
//
//  Created by ibeethoven.
//  Copyright (c) 2014年 ibeethoven. All rights reserved.
//

#import "CreateStudent.h"
#import "Student.h"

@implementation CreateStudent

+ (People *)createPeopleWithName:(NSString *)name andAge:(NSString *)age
{
    return [[Student alloc] initPeopleWithName:name andAge:age];
}

@end
```

CreateProgrammer同样也继承自Creator

```
//  CreateProgrammer.h
//
//  Created by ibeethoven.
//  Copyright (c) 2014年 ibeethoven. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "Creator.h"

@interface CreateProgrammer : Creator
{
}

@end


//  CreateProgrammer.m
//
//  Created by ibeethoven.
//  Copyright (c) 2014年 ibeethoven. All rights reserved.
//

#import "CreateProgrammer.h"
#import "Programmer.h"

@implementation CreateProgrammer

+ (People *)createPeopleWithName:(NSString *)name andAge:(NSString *)age
{
    return [[Programmer alloc] initPeopleWithName:name andAge:age];
}

@end
```

这种情况的调用：

```
    People * people = [[Creator alloc] createPeopleWithName:@"师人" andAge:@"22"];
    NSLog(@"%@", people);

    People * peopleS = [[CreateStudent alloc] createPeopleWithName:@"学学生" andAge:@"21"];
    NSLog(@"%@", peopleS);

    People * peopleP = [[CreateProgrammer alloc] createPeopleWithName:@"师员工" andAge:@"23"];
    NSLog(@"%@", peopleP);
```

生成结果：

```
2015-08-24 18:47:39.365 工厂方法[1348:126261] 师人, 我的年龄是:22
2015-08-24 18:47:39.368 工厂方法[1348:126261] 学学生, 我的年龄是:21, 我的学习是:我是一个学生
2015-08-24 18:47:39.369 工厂方法[1348:126261] 我的职业工作, 编程工作是我的乐趣, 师员工, 我的年龄是:23, 我的年龄是:23
```

上面的方法，每次扩展产品的时候，都需要创建一个生成器子类，这样就起来有些麻烦，我们也可以为每一类产品的子类创建一个公用的生成器。

使用生成Creator类：

```
//  Creator.h
//
//  Created by ibeethoven.
//  Copyright (c) 2014年 ibeethoven. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "People.h"

@interface Creator : NSObject

+ (People *)createPeopleWithName:(NSString *)name andAge:(NSString *)age withType:(NSString *)initType;

@end


//  Creator.m
//
//  Created by ibeethoven.
//  Copyright (c) 2014年 ibeethoven. All rights reserved.
//

#import "Creator.h"

@implementation Creator

+ (People *)createPeopleWithName:(NSString *)name andAge:(NSString *)age withType:(NSString *)initType
{
    if (initType isEqualToString:@"stu"){
        return [[Student alloc] initPeopleWithName:name andAge:age];
    }else if (initType isEqualToString:@"pro"){
        return [[Programmer alloc] initPeopleWithName:name andAge:age];
    }
    else
    {
        return nil;
    }
}

@end
```

调用代码：

```
    People * people = [[CreatePeople alloc] createPeopleWithName:@"学学生" andAge:@"21" withType:@"stu"];
    NSLog(@"%@", people);
    People * peopleP = [[CreatePeople alloc] createPeopleWithName:@"师员工" andAge:@"23" withType:@"pro"];
    NSLog(@"%@", peopleP);
```

输出结果：

```
2015-08-24 11:15:44.391 工厂方法[1348:77] 我是一个学生, 我的年龄是:21, 我的学习是:我是一个学生
2015-08-24 11:15:44.391 工厂方法[1348:77] 我的职业工作, 编程工作是我的乐趣, 师员工, 我的年龄是:23, 我的年龄是:23
```

工厂方法的优势：

我们可以用对象分为两个步骤创建：1. 分配空间，2. 初始化，例如：

```
NSArray * arr = [[NSArray alloc] initWithCount:[NSUIntegerMax]];
```

这有一些比较麻烦的方法来创建初始对象（第方法，以a开头的方法），例如：

```
NSArray * arr = [NSArray arrayWithCount:[NSUIntegerMax]];
```

这些比较便利创建对象的方法，是在类级别引包装那个真实工厂方法实现，它们称作类工厂方法。

工厂方法是面向对象程序设计中应用非常常见的设计模式。工厂方法从共同中消除了对应用程序特有类的耦合。

分享：ios设计模式

关注博主 支持博主 分享文章

sjzLovecj
粉丝 - 4
关注 - 2

0 推荐 0 反对

上一篇 ios设计模式--原型模式
下一篇 ios设计模式--抽象工厂

posted @ 2015-08-21 22:23 sjzLovecj 阅读(221) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部

登录后才能发表评论，立即 注册 或 登录。 访问 网站首页

可能感兴趣的一些推荐信息

【推荐】Nexus - 开源模型、工具、流集、CANDIS 为云计算Paas + 微服务基于容器
【推荐】有目标的，更好为你——博客园的产品管理进展，期望你的意见
【推荐】博客园UX改版，欢迎参与测试，欢迎反馈，意见优选
【推荐】了不起的开发者，挡不住的诱惑，现于墨天轮上线，聚焦开发者成长
【推荐】由阿里巨擘倾力开源最成熟最稳定的Java开发规范
【推荐】本地测试部署，最终调试环境，声明文件自打包功
【推荐】第一代Node应用，即编即用，Serverless可视开发工具

广告
灵感精选 闪耀华礼
现代简约美学与实用主义
LEFOOD x 设计
LAMY
Lamy Safari 钢笔 ¥ 1,100
Lamy Safari 钢笔 ¥ 1,100
DOMIN
DOMIN 多明诺 ¥ 1,100
DOMIN 多明诺 ¥ 1,100
现代简约美学与实用主义
闪耀华礼 闪耀华礼100
最新 IT 资讯 | 精选 IT 技术文章 | 关注 IT 行业动向 | 关注 IT 人物
· 苹果 iOS 9 新功能 Apple News 将支持中国大陆
· 苹果 iOS 9 新功能 Siri 支持多任务处理
· 特斯拉宣布入局太阳能领域，加州电池工厂
· 7 款最有用一代最实惠最流行的 Chrome 扩展
· 微信、QQ 等主流 App 即将不支持 Android 2.3 及更早版本
· 一夜之间的 ... 更多资讯 ...

Copyright © 2020 sjzLovecj
Powered by .NET 5.0 on Kubernetes