

[登录](#) | [注册](#)

卡奴达摩的专栏

海洋之大，源于点滴之水的积累。

[目录视图](#)[摘要视图](#)[RSS](#) [订阅](#)

个人资料



卡奴达摩



访问：1387582次

积分：12258

等级：[BLOG > 7](#)

排名：第775名

原创：52篇

转载：23篇

译文：0篇

评论：1329条

文章搜索

博客专栏



设计模式

文章：25篇

阅读：1018146

文章分类

[数据库](#) (14)

[疑难问题](#) (12)

[SSH框架](#) (2)

[Linux](#) (2)

[程序人生](#) (7)

[javascript](#) (2)

[设计模式](#) (27)

[字符串处理](#) (1)

[web前台](#) (5)

[算法](#) (1)

[【CSDN技术主题月】深度学习框架的重构与思考](#) [深度学习代码专栏](#) [【观点】有了深度学习，你还学传统机器学习算法么？](#)
[【知识库】深度学习知识图谱上线啦](#)

23种设计模式（10）：命令模式

标签：[设计模式](#) [command](#) [class](#) [敏捷开发](#) [扩展](#) [action](#)

2012-05-09 17:41

28824人阅读

[评论\(15\)](#)

[收藏](#)

[举报](#)

分类：

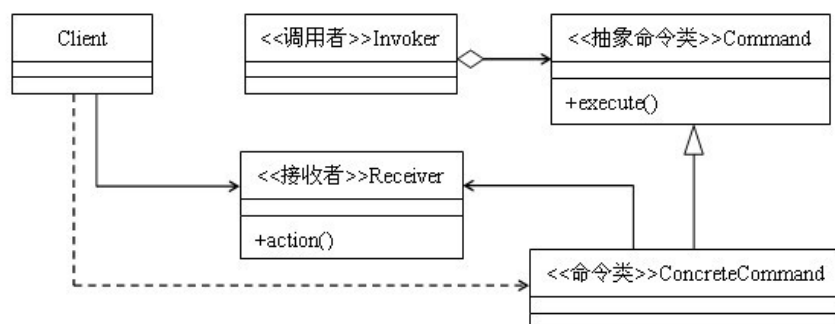
[设计模式 \(26\)](#)

版权声明：本文为博主原创文章，未经博主允许不得转载。

定义：将一个请求封装成一个对象，从而让你使用不同的请求把客户端参数化，对请求排队或者记录请求日志，可以提供命令的撤销和恢复功能。

类型：行为类模式

类图：



命令模式的结构

顾名思义，命令模式就是对命令的封装，首先来看一下命令模式类图中的基本结构：

- **Command类**：是一个抽象类，类中对需要执行的命令进行声明，一般来说要对外公布一个execute方法来执行命令。
- **ConcreteCommand类**：Command类的实现类，对抽象类中声明的方法进行实现。
- **Client类**：最终的客户端调用类。

以上三个类的作用应该算是比较好理解的，下面我们重点说一下Invoker类和Receiver类。

- **Invoker类**：调用者，负责调用命令。
- **Receiver类**：接收者，负责接收命令并且执行命令。

所谓对命令的封装，说白了，无非就是把一系列的操作写到一个方法中，然后供客户端调用就行了，反映到类图上，只需要一个ConcreteCommand类和Client类就可以完成对命令的封装，即使再进一步，为了增加灵

项目管理 (0)
项目总结 (1)
国学 (0)
工作流 (0)
JVM (1)
JAVA源码 (4)
C++系列 (0)
网络 (1)
高性能Web (0)

文章存档

2013年07月 (1)
2013年02月 (1)
2012年07月 (2)
2012年06月 (6)
2012年05月 (5)

展开

阅读排行

23种设计模式 (3) : 抽 (109008)
23种设计模式 (1) : 单 (90063)
23种设计模式 (2) : 工 (71407)
设计模式六大原则 (1) : (64266)
设计模式六大原则 (2) : (62718)
设计模式六大原则 (6) : (52397)
设计模式六大原则 (3) : (52133)
23种设计模式 (5) : 原 (46954)
设计模式六大原则 (5) : (43998)
单例模式讨论篇: 单例模 (42019)

评论排行

设计模式六大原则 (6) : (97)
设计模式六大原则 (1) : (94)
单例模式讨论篇: 单例模 (91)
设计模式六大原则 (3) : (84)
23种设计模式 (1) : 单 (77)
23种设计模式 (3) : 抽 (68)
迈出从3K到1W的重要一 (67)
设计模式六大原则 (2) : (66)
详解java类的生命周期 (59)
23种设计模式 (2) : 工 (54)

推荐文章

* 2016 年最受欢迎的编程语言是什么?
* Chromium扩展 (Extension) 的页面 (Page) 加载过程分析
* Android Studio 2.2 来啦
* 手把手教你做音乐播放器 (二) 技术原理与框架设计
* JVM 性能调优实战之: 使用阿里开源工具 TProfiler 在海量业务代码中精确定位性能代码

活性,可以再增加一个Command类进行适当地抽象,这个调用者和接收者到底是什么作用呢?

其实大家可以换一个角度去想:假如仅仅是简单地把一些操作封装起来作为一条命令供别人调用,怎么能称为一种模式呢?命令模式作为一种行为类模式,首先要做到低耦合,耦合度低了才能提高灵活性,而加入调用者和接收者两个角色的目的也正是为此。命令模式的通用代码如下:

```
[java]
01. class Invoker {
02.     private Command command;
03.     public void setCommand(Command command) {
04.         this.command = command;
05.     }
06.     public void action(){
07.         this.command.execute();
08.     }
09. }
10.
11. abstract class Command {
12.     public abstract void execute();
13. }
14.
15. class ConcreteCommand extends Command {
16.     private Receiver receiver;
17.     public ConcreteCommand(Receiver receiver){
18.         this.receiver = receiver;
19.     }
20.     public void execute() {
21.         this.receiver.doSomething();
22.     }
23. }
24.
25. class Receiver {
26.     public void doSomething(){
27.         System.out.println("接受者-业务逻辑处理");
28.     }
29. }
30.
31. public class Client {
32.     public static void main(String[] args){
33.         Receiver receiver = new Receiver();
34.         Command command = new ConcreteCommand(receiver);
35.         //客户端直接执行具体命令方式 (此方式与类图相符)
36.         command.execute();
37.
38.         //客户端通过调用者来执行命令
39.         Invoker invoker = new Invoker();
40.         invoker.setCommand(command);
41.         invoker.action();
42.     }
43. }
```

通过代码我们可以看到,当我们调用时,执行的时序首先是调用者类,然后是命令类,最后是接收者类。也就是说一条命令的执行被分成了三步,它的耦合度要比把所有的操作都封装到一个类中要低的多,而这也正是命令模式的精髓所在:把命令的调用者与执行者分开,使双方不必关心对方是如何操作的。

命令模式的优缺点

首先,命令模式的封装性很好:每个命令都被封装起来,对于客户端来说,需要什么功能就去调用相应的命令,而无需知道命令具体是怎么执行的。比如有一组文件操作的命令:新建文件、复制文件、删除文件。如果把这三个操作都封装成一个命令类,客户端只需要知道有这三个命令类即可,至于命令类中封装好的逻辑,客户端则无需知道。

其次,命令模式的扩展性很好,在命令模式中,在接收者类中一般会对操作进行最基本的封装,命令类则通过对这些基本的操作进行二次封装,当增加新命令的时候,对命令类的编写一般不是从零开始的,有大量的接收者类可供调用,也有大量的命令类可供调用,代码的复用性很好。比如,文件的操作中,我们需要增加一个剪切文件的命令,则只需要把复制文件和删除文件这两个命令组合一下就行了,非常方便。

最新评论

- 23种设计模式（10）：命令模式12期-赵尽朝:
@z15732621582:C++和Java的区别?
- 23种设计模式（10）：命令模式12期-赵尽朝: Invoker 和 Command的关系是聚合？怎么看着有点不熟悉？请教楼主！
- 23种设计模式（3）：抽象工厂模o对酒当歌: @dijr_2012:可以参考下
http://blog.csdn.net/superbeck/art...
- 23种设计模式（3）：抽象工厂模Tsing Teng: 看了评论，才发现感觉不太对的不是我一个人。
- 设计模式六大原则（2）：里氏替AriesUITL: 楼主可以从寄存器角度讲下吗？里氏替换原则有没有造成内存泄漏？有没有释放内存？
- 23种设计模式（2）：工厂方法模Oversdownload: 要是再把Icar写出来就更容易懂了
- 23种设计模式（13）：迭代器模: 一个有梦想的咸鱼: 感谢博主写出这么好的文章
- 23种设计模式（13）：迭代器模: 一个有梦想的咸鱼: 4楼说的有道理
- 23种设计模式（2）：工厂方法模qq_小倩的C博客: 下面的例子特别好懂
- 设计模式六大原则（6）：开闭原qq_小倩的C博客: 一进来就是最后一篇，直接下载啦，哈哈，谢谢楼主

最后说一下命令模式的缺点，那就是命令如果很多，开发起来就要头疼了。特别是很多简单的命令，实现起来就几行代码的事，而使用命令模式的话，不用管命令多简单，都需要写一个命令类来封装。

命令模式的适用场景

对于大多数请求-响应模式的功能，比较适合使用命令模式，正如命令模式定义说的那样，命令模式对实现记录日志、撤销操作等功能比较方便。

总结

对于一个场合到底用不用模式，这对所有的开发人员来说都是一个很纠结的问题。有时候，因为预见到需求上会发生的某些变化，为了系统的灵活性和可扩展性而使用了某种设计模式，但这个预见的的需求偏偏没有，相反，没预见到的需求倒是来了不少，导致在修改代码的时候，使用的设计模式反而起了相反的作用，以至于整个项目组怨声载道。这样的例子，我相信每个程序设计者都遇到过。所以，基于敏捷开发的原则，我们在设计程序的时候，如果按照目前的需求，不使用某种模式也能很好地解决，那么我们就不要引入它，因为要引入一种设计模式并不困难，我们大可以在真正需要用到时候再对系统进行一下，引入这个设计模式。

拿命令模式来说吧，我们开发中，请求-响应模式的功能非常常见，一般来说，我们会把对请求的响应操作封装到一个方法中，这个封装的方法可以称之为命令，但不是命令模式。到底要不要把这种设计上升到模式的高度就要另行考虑了，因为，如果使用命令模式，就要引入调用者、接收者两个角色，原本放在一处的逻辑分散到了三个类中，设计时，必须考虑这样的代价是否值得。

顶

46

踩

2

上一篇

详解java类的生命周期

下一篇

23种设计模式（11）：责任连模式

我的同类文章

设计模式 (26)

• 23种设计模式 (15) : 备忘录模式

2012-06-27

阅读 36537

• 23种设计模式 (13) : 迭代器模式

2012-05-28

阅读 22283

• 23种设计模式 (11) : 责任链模式

2012-05-15

阅读 14346

• 23种设计模式 (9) : 访问者模式

2012-04-23

阅读 36233

• 23种设计模式 (7) : 中介者模式

2012-04-05

阅读 23592

• 23种设计模式 (14) : 解释器模式

2012-06-15

阅读 19957

• 23种设计模式 (12) : 策略模式

2012-05-28

阅读 20060

访问者模式讨论篇: java的访问者模式

2012-04-25

阅读 12870

• 23种设计模式 (8) : 观察者模式

2012-04-18

阅读 25746

• 23种设计模式 (6) : 模板方法模式

2012-03-29

阅读 18404

更多文章

参考知识库



敏捷知识库

1210 关注 | 138 收录



Java EE知识库

6799 关注 | 690 收录



Java SE知识库

14039 关注 | 459 收录



Java 知识库

15796 关注 | 1288 收录

猜你在找

【技术公开课】SCRUM敏捷开发	23种设计模式10命令模式
设计模式	23种设计模式10命令模式
设计模式课程	23种设计模式10命令模式
iOS8开发视频教程Swift语言版-Part 7:iOS常用设计模式	23种设计模式10命令模式
火星敏捷开发1001问 (第二季)	23种设计模式10命令模式

查看评论

11楼 [12期-赵尽朝](#) 3天前 16:47发表



Invoker 和 Command的关系是聚合? 怎么看着有点不熟悉? 请教楼主!

Re: [12期-赵尽朝](#) 3天前 16:48发表



回复z15732621582: C++和Java的区别?

10楼 [zhaoyu_android4311](#) 2014-02-07 20:54发表



楼主:

UML图中action方法的位置是不是画错地了, 感觉和代码不对着

9楼 [houchangren](#) 2014-01-10 17:42发表



那就是命令如果很多, 开发起来就要头疼了。特别是很多简单的命令, 实现起来就几行代码的事, 而使用命令模式的话, 不用管命令多简单, 都需要写一个命令类来封装。

---其实这句话我觉得着有点不太赞同, 比如你可以等到真正操作的时候再去实现这个操作就可以了比如

```
invoker.setCommand(new Command() {  
    @Override  
    public void execute() {  
        //因为命令很简单所以这个直接可以写进去, 以后回调直接调用它的execute函数。操作很简单啊。  
        //syso...  
    }  
});  
invoker.action();
```

不知道对不对, 也可能是对command的模式的理解不透彻。

8楼 [快乐的Android小小鸟](#) 2013-08-30 14:08发表



把一个命令的执行分成三个层次, 乍一看有点冗余的感觉, 不过我写过一个测试例子后感觉就不一样了:

在Command下面增加Receiver: Receiver的作用是封装原子操作, 这样添加新命令时更加灵活 (就像lz举的例子)

在Command上面增加Invoker: Invoker的作用是封装执行命令的方式, 比如说许多客户端并发请求服务器执行某些命令, 这时服务器可以有不同的执行方式 (基于优先级, 先来先执行等)

不知道对不对, 欢迎指正

7楼 [可少](#) 2013-04-21 00:01发表



百度百科的实际例子, 协助大家理解:

电视机是请求的接收者, 遥控器是请求的发送者, 遥控器上有一些按钮, 不同的按钮对应电视机的不同操作。抽象命令角色由一个命令接口来扮演, 有三个具体的命令类实现了抽象命令接口, 这三个具体命令类分别代表三种操作: 打开电视机、关闭电视机和切换频道。显然, 电视机遥控器就是一个典型的命令模式应用实例。

6楼 [lijihong0723](#) 2012-10-25 15:00发表



一般场景: Client请求(FE), 在BE根据请求类型建立相应的Command实例, 执行相应的Receiver方法, 我觉得Invoke所起作用不大明显。而Command在这里起到了媒介的作用, 真正做事的是Receiver。

5楼 [桐桐-Dragon](#) 2012-10-23 16:00发表




抓紧写啊, 村头厕所都没纸了

4楼 [TalkingRabbit](#) 2012-09-11 14:18发表



mvc好像命令的扩展

3楼 [yangnianbing110](#) 2012-05-29 16:41发表




没有涉及到命令模式的撤销和恢复功能，能不能再举个列子呢？

2楼

kangxingang


2012-05-15 07:54发表



LZ，目前是不是只写到十？

Re: 卡奴达摩


2012-05-15 14:57发表



回复kangxingang：嗯，目前只写到10，打算把23种模式都总结一下。

Re: kangxingang

2012-05-16 20:23发表




回复zhengzhib：谢谢楼主总结了。。正在学习中。。

1楼

简洁是智慧的灵魂


2012-05-09 21:32发表



lz终于更新了。。哈哈

Re: 卡奴达摩

2012-05-09 21:57发表



回复wangpeifeng669：呵呵，不好意思啊，五一期间请了三天假，回了趟老家。

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题

Hadoop

AWS

移动游戏

Java

Android

iOS

Swift

智能硬件

Docker

OpenStack

VPN

Spark

ERP

IE10

Eclipse

CRM

JavaScript

数据库

Ubuntu

NFC

WAP

jQuery

BI

HTML5

Spring

Apache

.NET

API

HTML

SDK

IIS

Fedora

XML

LBS

Unity

Splashtop

UML

components

Windows Mobile

Rails

QEMU

KDE

Cassandra

CloudStack

FTC

coremail

OPhone

CouchBase

云计算

iOS6

Rackspace

Web App

SpringSide

Maemo

Compuware

大数据

aptech

Perl

Tornado

Ruby

Hibernate

ThinkPHP

HBase

Pure

Solr

Angular

Cloud Foundry

Redis

Scala

Django

Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服

杂志客服

微博客服

webmaster@csdn.net

400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持

京 ICP 证 09002463 号 | Copyright © 1999-2016, CSDN.NET, All Rights Reserved

