

wsh622827的专栏

目录视图

摘要视图

RSS 订阅

个人资料



wsh622827

访问: 52560次

积分: 722

等级:

BLOG > 3

排名: 千里之外

原创: 19篇 转载: 3篇

译文: 5篇 评论: 21条

文章搜索

文章分类

Algorithms (1)

Data Mining (2)

Database (1)

Design Patterns (4)

Java (6)

Node.JS (2)

文章存档

2012年06月 (1)

2012年05月 (1)

2010年01月 (2)

2009年12月 (4)

2009年11月 (4)

展开

阅读排行

java.util.Date和java.sql.

Spring 3.0参考手册之Sp

命令模式在MVC框架中的

Jquery UI Theme 切换

Spring 3.0参考手册之集

Spring AOP中两种动态什

Java实现观察者模式

CSDN学院讲师招募, 诚邀您加入!

博客Markdown编辑器上线啦

那些年我们追过的Wrox精品红皮计算机图书

PMBOK

第五版精讲视频教程

火星人敏捷开发1001问

命令模式在MVC框架中的应用

2009-11-02 21:22 4180人阅读 评论(4) 收藏 举报

mvc

框架

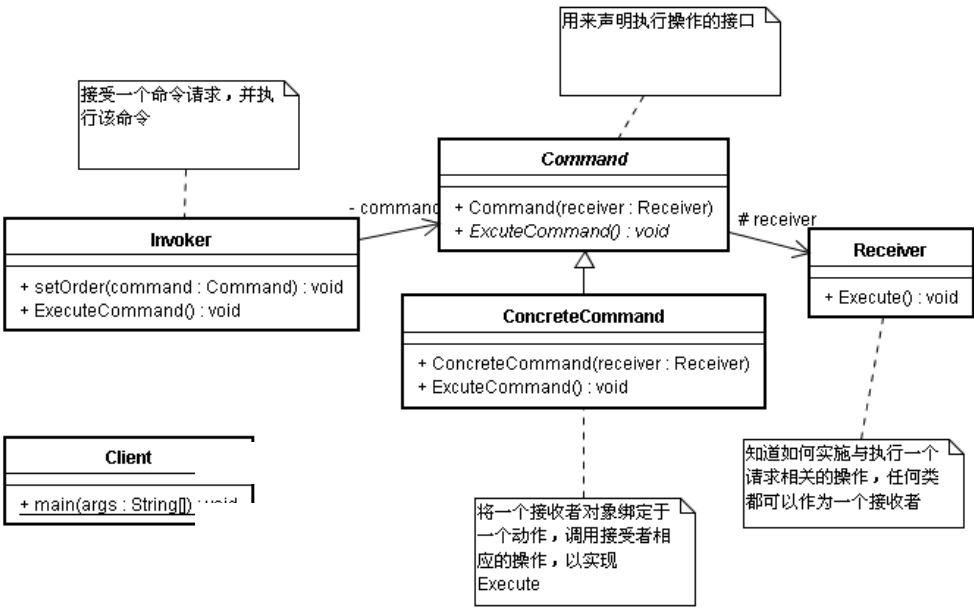
hashmap

command

action

servlet


命令模式:
定义: 把一个请求或者操作封装在命令对象中。命令模式允许系统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。



Invoker类 被客户端调用, 可以接受命令请求, 设计命令队列, 决定是否相应该请求,记录或撤销或重做命令请求, 记录日志等等。

[java]

01. public class Invoker {
02. private Command command;
03. public void setOrder(Command command) {
04. this.command = command;
05. }
06. }
07.



Command类, 将一个请求封装成一个对象, 将一个请求具体化, 方便对请求记录。

[java]

01. public abstract class Command {
02. }
03.

Java备忘录模式	(1082)
cache技术提高Web应用	(903)
常用Java Date方法	(588)

评论排行	
Spring 3.0参考手册之Sp	(5)
命令模式在MVC框架中的	(4)
java.util.Date和java.sql.T	(3)
Jquery UI Theme 切换	(3)
对区间的模糊排序	(2)
用对象图学习JavaScript	(1)
Spring 3.0参考手册之集J	(1)
用对象图学习JavaScript	(1)
结构型模式——适配器的	(1)
结构型模式——代理复习	(0)

推荐文章	
* 【ShaderToy】开篇	
* FFmpeg源代码简单分析：avio_open2()	
* 技能树之旅：从模块分离到测试	
* Qt5官方demo解析集36——Wiggly Example	
* Unity3d HDR和Bloom效果（高动态范围图像和泛光）	
* Android的Google官方设计指南	

最新评论	
命令模式在MVC框架中的应用 放逐在七号: 学习了。。。。	
命令模式在MVC框架中的应用 汤长海: 很好。核心控制器 ActionServlet也是Front Controller模式的体现	
Spring 3.0参考手册之集成Web S sunyuanfeng613: 例子不详细， 初学者看不懂	
结构型模式——适配器的两个应 hacke2: 你的博客真心不错	
java.util.Date和java.sql.Timestar 突破者cd: 谢谢了，遇到同样问 题。	
命令模式在MVC框架中的应用 淘娃: 谢谢博主~写得很好~	
java.util.Date和java.sql.Timestar Shut___up: 学习了，谢谢	
java.util.Date和java.sql.Timestar HaiPiaoYeZi: 刚刚有遇到同样的 问题，正好看了你写的，谢了。	
Spring 3.0参考手册之SpEL kuaifei1: 能发一份完整的spring 3.0给我吗？亲！ kuai_fly@163.com	
Jquery UI Theme 切换 tiantangqiu: 很好，嘿嘿~	

```
02.     protected Receiver receiver;
03.     public Command(Receiver receiver){
04.         this.receiver = receiver;
05.     }
06.     public abstract void ExecuteCommand();
07. }
```

ConcreteCommand类，可以将Receiver对象放到这个类里面，这个类具体实现了要怎么处理这个用户的请求。

```
[java]
01. public class ConcreteCommand extends Command {
02.     public ConcreteCommand(Receiver receiver){
03.         super(receiver);
04.     }
05.     @Override
06.     public void ExecuteCommand() {
07.         receiver.Execute();
08.     }
09. }
```

Receiver类，其实这个类可以没有，不过为了让设计看起来更整洁清楚。

```
[java]
01. public class Receiver {
02.     public void Execute(){
03.         System.out.println("Receiver excute!");
04.     }
05. }
```

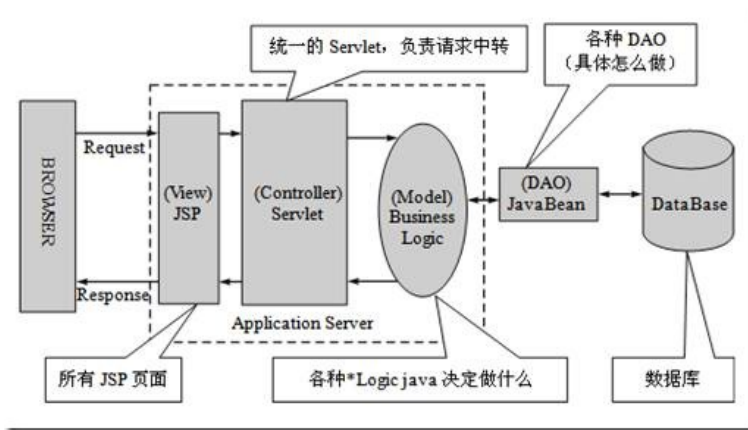
最后一个Client类。

```
[java]
01. public class Client {
02.     public static void main(String[] args) {
03.         Receiver r = new Receiver();
04.         Command c = new ConcreteCommand(r);
05.         Invoker i = new Invoker();
06.         i.setOrder(c);
07.         i.ExecuteCommand();
08.     }
09. }
```

命令模式在MVC中的应用：

Struts中，在模型层都要继承一个Action接口，并实现execute方法，其实这个Action就是命令类。为什么Struts会应用命令模式，是因为Struts的核心控制器ActionServlet只有一个，相当于Invoker，而模型层的类会随着不同的应用有不同的模型类，相当于具体的Command。这样，就需要在ActionServlet和模型层之间解耦，而命令模式正好解决这个问题。

MVC Model2 实现的Web框架示意图：



说明:

- 视图层采用JSP实现
- 控制器采用Servlet实现，整个框架采用同一个Servlet，以实现请求的中转
- 模型层采用Java实现，主要决定用来做什么
- 在模型层后添加了一个DAO，目的是将决定做什么和具体怎么做分开

整个Web框架大致的流程是：首先客户端发送请求，提交JSP页面给中转器（Servlet）；中转器根据客户的请求，选择相应的模型层，即Logic，Logic进行相应的逻辑处理；如果需要使用数据库，则通过DAO进行相应的数据库操作。

下面主要看一下控制层和模型层的设计，应用命令模式：

1.控制层设计

控制层主要用来转发从视图层传来的数据和请求到相对应的模型层，因此，实现它最好的方式莫过于使用Servlet了。当从视图层获取请求后，首先通过对web.xml文件的配置，使其转入Servlet，在Servlet中完成对页面中数据的封装和对相应模型的选择，然后再到相应的模型层进行数据处理；当在模型层数据处理完毕后，通过RequestDispatcher将处理后的数据返回相应的视图页面。

在Servlet中，将使用doPost()来处理相应的中转请求，如果开发人员使用get提交方式，则使用如下方式进行处理。示例代码如下：

```
[java]
01. public void doGet(HttpServletRequest req, HttpServletResponse res)
02.     throws ServletException, IOException {
03.     doPost(req, res);
04. }
05. //使用post提交方式
06. public void doPost(HttpServletRequest req, HttpServletResponse res)
07.     throws ServletException, IOException {
08.     do_dispatcher (req, res);
09. }
```

代码说明：

- 不论采用get还是post提交方式，都将执行do_dispatcher(req, res)方法。
- do_dispatcher(req, res)是用来处理视图层发送来的请求的方法。

如果直接使用request方式来获取从页面提交的数据，在要获取的数据比较多的情况下，会比较烦琐，而且直接将request传递给模型层不符合Model 2规范。所以，这里将对从页面传来的值进行封装，将其放在一个Map中，然后再传递给模型层，这样在模型层就可以直接使用Map中的值。示例代码如下：

```
[java]
01. private HashMap getRequestToMap(HttpServletRequest req) throws Exception {
02.     req.setCharacterEncoding("GBK");
03.     HashMap infoIn = new HashMap();
04.     for (Enumeration e = req.getParameterNames(); e.hasMoreElements ();)
05.         { //获取页面中所有元素名
06.             String strName = (String)e.nextElement();
07.             String[] values = (String[]) req.getParameterValues (strName);
08.             //根据名称获取对应的值
09.             if (values == null) { //假如没有值
10.                 infoIn.put(strName, "");
11.             } else if (values.length == 1) { //假如只有一个值
```

```

12.         infoIn.put(strName, values[0]);
13.     } else { //假如有多个值
14.         infoIn.put(strName, values);
15.     }
16. }
17. return infoIn;
18. }

```

代码说明:

- req.setCharacterEncoding("GBK"), 这里首先将从视图层传来的数据设定编码为GBK。
- HashMap infoIn = new HashMap(), 定义一个HashMap, 用来存放从request中获取的数据。
- req.getParameterNames(), 用来获取从页面中传来的所有元素。
- req.getParameterValues(), 用来根据元素名称来获取元素对应的值, 并将元素名称和值的对应关系存入HashMap中。如果元素的值为空, 则在HashMap中将元素名称对应的值置为空; 如果只有一个值, 则将该值存入; 如果有多个值, 则存入数组。

命令模式使用:

一个视图对应一个模型, 也可能一个视图对应多个模型, 但只有一个控制器, 所以, 为了实现一个控制器可以转发到多个模型中去, 就需要使用接口, 让所有模型都实现这个接口, 然后在控制器里, 仅仅是面对接口编程即可。这里定义一个接口Action.java, Action.java的示例代码如下:

```

[java]
01. //***** Action.java*****
02. import java.util.*;
03. public interface Action{
04.     public HashMap doAction(HashMap infoIn);
05. }

```

在控制器中只针对这个接口处理即可。示例代码如下:

```

[java]
01. Actionaction = (Action) Class.forName(getActionName(systemName,
02.     logicName)).newInstance();
03. HashMap infoOut = action.doAction(infoIn);

```

代码说明:

- getActionName()方法是获取实现接口Action的类的名称和所在的包。示例代码如下:

```

[java]
01. private String getActionName(String systemName ,String actionName)
02. throws IOException, Exception {
03.     return "com. " + systemName + ".action." + actionName;
04. }

```

使用RequestDispatcher返回视图层。示例代码如下:

```

[java]
01. req.setAttribute("infoOut", infoOut);
02. RequestDispatcher rd = req.getRequestDispatcher("/"+ systemName +
03.     "/jsp/" + forwardJsp+ ".jsp");
04. rd.forward(req, res);

```

代码说明:

- 这里表示JSP文件放在项目中系统名下的jsp文件夹下。

2. 模型层设计

假定有一个模型层类为WebExamAction.java, 主要用来负责在线考试系统的业务处理, 则这个类要实现Action

接口。示例代码如下：

```
[java]
01. //***** WebExamAction.java*****
02. import java.util.HashMap;
03. public class WebExamAction implements Action{
04.     //根据页面的请求，进行动作的转换
05.     public HashMap doAction(HashMap infoIn) {
06.         String action = (infoIn.get("action") == null) ? "" :
07. (String)infoIn.get("action");
08.         HashMap infoOut = new HashMap();
09.         if (action.equals("")) infoOut = this.doInit (infoIn);
10.         else if (action.equals("insert")) infoOut = this.doInsert (infoIn);
11.         return infoOut;
12.     }
13.     /**该方法设置用户登录时页面的初始信息
14.     * @param infoIn
15.     * @return HashMap
16.     */
17.     private HashMap doInit(HashMap infoIn) {
18.         HashMap infoOut = infoIn;
19.         int clerkId = (infoIn.get("clerkId") == null || "".equals
20. (infoIn.get("clerkId"))) ? -1 : Integer.parseInt((String)
21. infoIn.get ("clerkId"));
22.         try {
23.             //取得考生姓名
24.             Users user = new Users();
25.             String clerkName = user.getName(clerkId);
26.             infoOut.put("clerkName", clerkName);
27.         } catch(Exception e) {
28.             e.printStackTrace();
29.         } finally {
30.             return infoOut;
31.         }
32.     }
33.     /**该方法用来进行新增
34.     * @param infoIn
35.     * @return HashMap
36.     */
37.     private HashMap doInsert(HashMap infoIn) {
38.         HashMap infoOut = infoIn;
39.         int clerkId = (infoIn.get("clerkId") == null || "".equals
40. (infoIn.get("clerkId"))) ? -1 : Integer.parseInt((String)
41. infoIn.get ("clerkId"));
42.         try {
43.             //取得考生姓名
44.             Users user = new Users();
45.             String clerkName = user.getName(clerkId);
46.             infoOut.put("clerkName", clerkName);
47.         } catch(Exception e) {
48.             e.printStackTrace();
49.         } finally {
50.             return infoOut;
51.         }
52.     }
53. }
```

代码说明：

- 这里，在doAction中根据从页面传来的action进行动作请求的转换。
- 通过一个名为infoIn的HashMap，来负责传入页面中元素的值。
- 通过一个名为infoOut的HashMap，来负责将处理后的数据传出。

可以看出，如果模型层都实现接口Action，实现doAction方法，即可实现动作请求的转换。

命令模式要点：

1. Command模式的根本目的在于将“行为请求者”与“行为实现者”解耦，在面向对象语言中，常见的实现手段是“将行为抽象为对象”。
2. 实现Command接口的具体命令对象ConcreteCommand有时候根据需要可能会保存一些额外的状态信息。
3. 通过使用Compmosite模式，可以将多个命令封装为一个“复合命令”MacroCommand。
4. Command模式与C#中的Delegate有些类似。但两者定义行为接口的规范有所区别：Command以面向对象中的“接口-实现”来定义行为接口规范，更严格，更符合抽象原则；Delegate以函数签名来定义行为接口规范，

更灵活，但抽象能力比较弱。

5. 使用命令模式会导致某些系统有过多的具体命令类。某些系统可能需要几十个，几百个甚至几千个具体命令类，这会使命令模式在这样的系统里变得不实际。

适用性：

- 在下面的情况下应当考虑使用命令模式：
- 1. 使用命令模式作为"CallBack"在面向对象系统中的替代。"CallBack"讲的便是先将一个函数登记上，然后在以后调用此函数。
 - 2. 需要在不同的时间指定请求、将请求排队。一个命令对象和原先的请求发出者可以有不同的生命期。换言之，原先的请求发出者可能已经不在，而命令对象本身仍然是活动的。这时命令的接收者可以是在本地，也可以在网络的另外一个地址。命令对象可以在串行化之后传送到另外一台机器上去。
 - 3. 系统需要支持命令的撤消(undo)。命令对象可以把状态存储起来，等到客户端需要撤销命令所产生的效果时，可以调用undo()方法，把命令所产生的效果撤销掉。命令对象还可以提供redo()方法，以供客户端在需要时，再重新实施命令效果。
 - 4. 如果一个系统要将系统中所有的数据更新到日志里，以便在系统崩溃时，可以根据日志里读回所有的数据更新命令，重新调用Execute()方法一条一条执行这些命令，从而恢复系统在崩溃前所做的数据更新。
- 参考：
- 1. 《自己动手写Struts：构建基于MVC的Web开发框架》
 - 2. 《敏捷软件开发：原则、模式与实践》
 - 3. 《Head First设计模式》

上一篇 [Java实现观察者模式](#)

下一篇 [Ubuntu 9.10 安装mysql](#)

主题推荐 框架 mvc web框架 web开发 面向对象

猜你在找

与String相关的面试题汇总	spring整合memcached注意事项-poolname
DOM事件流	C++类静态成员变量和const常量的初始化方法
介绍java中listset和map 的区别	单点登录学习2CAS服务器端配置编程
Tomcat vs Apache	SciterHTMLLayout中元素的 增 删 改 查
在WCF服务中获取客户端的IP地址和端口号	js实现点击浏览器historygo返回上一页 刷新上一页

准备好了么？跳吧！ 更多职位尽在 CSDN JOB

Android Engineer / Android手机应用	我要跳槽	应用开发岗(手机银行方向)	我要跳槽
北京万桥达观信息技术有限公司	13-20K/月	江苏常熟农村商业银行股份有限公司	8-12K/月
应用开发岗(呼叫中心方向)	我要跳槽	资深Android应用开发工程师	我要跳槽
江苏常熟农村商业银行股份有限公司	8-12K/月	广州晶绮信息科技有限公司	5-9K/月

查看评论

4楼 放逐在七号 2014-07-31 10:24发表



学习了。。。

3楼 汤长海 2014-07-30 11:48发表



很好。核心控制器ActionServlet也是Front Controller模式的体现

2楼 淘娃 2013-12-01 19:09发表



谢谢博主~写得很好~

1楼 科學信仰偉大生物科學 2010-05-21 11:29发表



好啊 写得很好

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