# 16.MVC设计思想

## 16.1.MVC设计思想

Model-View-Controller，软件编程的通用的设计思想。mvc设计思想认为，任何软件都可以分为: 负责程序控制的控制器、负责封装数据处理数据的模型、负责展示数据的视图来组成的。MVC设计思想要求一个符合MVC设计思想的软件，应该尽量的让这三者互相独立，互补干扰，每个模块只做自己该做的事情，一个模块的变化不应该影响其他模块。好处是 软件的结构更加的清晰，便于开发维护，模块可以实现复用。

## 16.2.案例：重构EasyMall项目

### 16.2.1.项目环境重构

1.分包： （注意，分包之前可以先备份一份）

cn.tedu.web

cn.tedu.service

cn.tedu.dao

cn.tedu.domain/bean

cn.tedu.utils

cn.tedu.exception

cn.tedu.factory(后面会用工厂设计模式进行层与层之间的解耦，暂时用不到)

2.导入开发包：

~jstl（JavaEE5.0及以后的版本不需要）

c3p0.jar

mysql-connector-java.jar

3.配置文件：

c3p0-config.xml

### 16.2.2.注册功能重构

代码如下：

|  |
| --- |
| 将RegistServlet改造为如下内容:  注意：用到BeanUtils的时候需要导包，在课前资料中BeanUtils目录下的两个包放到lib目录中。  //1.处理请求参数问题（post）  request.setCharacterEncoding("utf-8");  //处理响应正文乱码  response.setContentType("text/html;charset=utf-8");    //2.校验验证码  String valistr = request.getParameter("valistr");  **if**(WebUtils.*isNull*(valistr)){  //将提示存入request域中，通过转发将消息带到regist.jsp中  request.setAttribute("msg", "验证码不能为空！");  request.getRequestDispatcher("/regist.jsp").forward(request, response);  **return**;  }  //>>验证码是否正确  //从session中获取验证码内容  String code = (String) request.getSession().getAttribute("code");  **if** (!valistr.equalsIgnoreCase(code)) {  //将提示存入request域中，通过转发将消息带到regist.jsp中  request.setAttribute("msg", "验证码不正确！");  request.getRequestDispatcher("/regist.jsp").forward(request, response);  **return**;  }    //3.获取请求参数,并将数据封装到JavaBean中  User user = **new** User();  //可以用BeanUtils包来封装数据，比较快捷，需要先导入jar包，在引入包的时候，注意要使用org.apache开头的包  **try** {  BeanUtils.*populate*(user, request.getParameterMap());  } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException();  }    //4.调用JavaBean的checkData方法校验数据  **try** {  user.checkData();  //5.实现注册(将用户信息保存进数据库)  UserService service = **new** UserService();  service.registUser(user);  } **catch** (MsgException e) {  //将MsgException中的异常信息存入request域中  request.setAttribute("msg", e.getMessage());  //跳转回注册页面, 显示提示消息  request.getRequestDispatcher("/regist.jsp").forward(request, response);  **return**;  }    //6.提示用户注册成功，3秒后跳转到首页。  response.getWriter().write("<h1 style='color:red;text-align:center'>恭喜您注册成功，3秒之后跳转回首页...</h1>");  response.setHeader("refresh", "3;url="+request.getContextPath()+"/index.jsp"); |
| 在第四步，调用JavaBean的checkData方法时，在User类中添加如下代码：  先加入password2属性：  **private** String password2;  并添加get和set方法  然后将添加如下代码：  **public** **void** checkData() **throws** MsgException {  //>>非空校验  **if**(WebUtils.*isNull*(username)){  **throw** **new** MsgException("用户名不能为空！");  }  **if**(WebUtils.*isNull*(password)){  **throw** **new** MsgException("密码不能为空！");  }  **if**(WebUtils.*isNull*(password2)){  **throw** **new** MsgException("确认密码不能为空！");  }  **if**(WebUtils.*isNull*(nickname)){  **throw** **new** MsgException("昵称不能为空！");  }  **if**(WebUtils.*isNull*(email)){  **throw** **new** MsgException("邮箱不能为空！");  }    //>>两次密码是否一致  **if**(!password.equals(password2)){  **throw** **new** MsgException("两次密码不一致！");  }    //>>邮箱格式校验  //abc@sina.com.cn  **if**(!email.matches("^\\w+@\\w+(\\.\\w+)+$")){  **throw** **new** MsgException("邮箱格式不正确！");  } |
| 在exception包下, 创建MsgException类, 继承Exception类, 代码如下:  **public** MsgException() {  }  **public** MsgException(String msg) {  **super**(msg);  } |
| 在service包下创建UserService，并添加如下内容：  **private** UserDao dao = **new** UserDao();  /\*\*  \* 实现注册  \* **@param** user  \* **@throws** MsgException  \*/  **public** **void** registUser(User user) **throws** MsgException {  //1.检查用户名是否存在  **boolean** result = dao.findUserByUsername(user.getUsername());  **if**(result){//用户名已存在  **throw** **new** MsgException("用户名已存在");  }  //2.实现注册（保存数据到数据库）  dao.addUser(user);  } |
| 在dao包下创建UserDao，并添加如下内容：  /\*\*  \* 根据用户名查询用户是否存在  \* **@param** username 用户名  \* **@return** true表示用户名存在，false表示不存在  \*/  **public** **boolean** findUserByUsername(String username) {  Connection conn = **null**;  PreparedStatement ps = **null**;  ResultSet rs = **null**;  **try** {  conn = JDBCUtils.*getConnection*();  String sql = "select \* from user where username=?";  ps = conn.prepareStatement(sql);  ps.setString(1, username);  rs = ps.executeQuery();  **return** rs.next(); //true表示存在  } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException();  }**finally**{  JDBCUtils.*close*(conn, ps, rs);  }  }    /\*\*  \* 将用户注册信息保存进数据库中  \* **@param** user User对象  \*/  **public** **void** addUser(User user) {  Connection conn = **null**;  PreparedStatement ps = **null**;  ResultSet rs = **null**;  **try** {  conn = JDBCUtils.*getConnection*();  String sql = "insert into user values(null,?,?,?,?)";  ps = conn.prepareStatement(sql);  ps.setString(1, user.getUsername());  ps.setString(2, user.getPassword());  ps.setString(3, user.getNickname());  ps.setString(4, user.getEmail());    //执行sql  ps.executeUpdate();  } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException();  }**finally**{  JDBCUtils.*close*(conn, ps, rs);  }  } |

### 16.2.3.登录功能重构

代码如下：

|  |
| --- |
| 将LoginServlet改造为如下内容:  //1.处理请求乱码  request.setCharacterEncoding("utf-8");  //2.获取请求参数  String username = request.getParameter("username");  String password = request.getParameter("password");  String remname = request.getParameter("remname");    //3.调用service层的方法进行登录  UserService service = **new** UserService();  User user = service.loginUser(username,password);  **if**(user != **null**){//用户名密码正确  //4.实现记住用户名功能  **if**("true".equals(remname)){  Cookie cookie = **new** Cookie("remname", URLEncoder.*encode*(username,"utf-8"));  cookie.setPath(request.getContextPath()+"/");  cookie.setMaxAge(60\*60\*24\*30);//设置30天  response.addCookie(cookie);  }**else**{//取消记住用户名（删除cookie）  Cookie cookie = **new** Cookie("remname","");  cookie.setPath(request.getContextPath()+"/");  cookie.setMaxAge(0);//设置30天  response.addCookie(cookie);  }  //5.进行登录  request.getSession().setAttribute("user", user);    //6.重定向到主页  response.sendRedirect(request.getContextPath()+"/index.jsp");    }**else**{//用户名密码不正确  request.setAttribute("msg","用户名或密码不正确！");  request.getRequestDispatcher("/login.jsp").forward(request, response);  } |
| 在UserService中添加如下代码:  /\*\*  \* 实现登录  \* **@param** username 用户名  \* **@param** password 密码  \* **@return** User对象  \*/  **public** User loginUser(String username, String password) {  **return** dao.findUserByUsernameAndPassword(username,password);  } |
| 在UserDao中添加如下代码:  /\*\*  \* 根据用户名密码查询用户  \* **@param** username 用户名  \* **@param** password 密码  \* **@return** User对象  \*/  **public** User findUserByUsernameAndPassword(String username, String password) {  Connection conn = **null**;  PreparedStatement ps = **null**;  ResultSet rs = **null**;  **try** {  conn = JDBCUtils.*getConnection*();  String sql = "select \* from user where username=? and password=?";  ps = conn.prepareStatement(sql);  ps.setString(1, username);  ps.setString(2, password);  rs = ps.executeQuery();  **if**(rs.next()){//用户名密码正确  //将结果集中的第一条记录封装到JavaBean并返回  User user = **new** User();  user.setId(rs.getInt("id"));  user.setUsername(rs.getString("username"));  user.setPassword(rs.getString("password"));  user.setNickname(rs.getString("nickname"));  user.setEmail(rs.getString("email"));    **return** user;  }**else**{//用户名或密码错误  **return** **null**;  }  } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException();  }**finally**{  JDBCUtils.*close*(conn, ps, rs);  }  } |

### 16.2.4.Ajax检查用户名

代码如下：

|  |
| --- |
| 将AjaxCheckUsernameServlet改造为如下内容:  //1.处理乱码（load方法为post提交）  request.setCharacterEncoding("utf-8");  response.setContentType("text/html;charset=utf-8");  //2.获取请求参数（用户名）  String username = request.getParameter("username");  //3.调用service层的方法检查用户名是否存在  UserService service = **new** UserService();  **boolean** result = service.hasUser(username);  **if**(result){  response.getWriter().write("用户名已存在！");  }**else**{  response.getWriter().write("恭喜，用户名可以使用！");  } |
| 在UserService中添加如下代码:  /\*\*  \* 根据用户名查询用户是否存在  \* **@param** username 用户名  \* **@return** true表示用户名已存在，false表示不存在  \*/  **public** **boolean** hasUser(String username) {  **return** dao.findUserByUsername(username);  } |

## 16.3.EasyMall解耦

### 16.3.1.概述

在EasyMall中引入接口(面向接口编程)

接口本质上是一套规范，从接口的定义方面来说, 接口其实就是类和类之间的一种协定,一种约束.

在软件开发中，某一层可能会被修改或者替换，在完全替换某一层时，要保证新的层和被替换的层，虽然具体实现不同，但是必须得实现相同的功能。这样才能尽可能的不影响到其他层。

### 16.3.2.耦合和解耦

**耦合**

所谓的耦合指的是在软件开发中，在层与层之间产生了某种紧密的关系，这种关系可能会导致，在我们修改或者是替换某一层时会影响其他的层，像这种情况我们就称之为层与层之间产生了耦合。

由于耦合（层与层之间的紧密关系）可能会导致我们在修改某一层时影响到其他的层，而这严重违反了我们对软件进行分层的最初设想 -- 软件各层之间应该相互独立、互不干扰，在修改或者是替换某一层时，应该做到不影响其他层。

**解耦**

去除耦合的过程，称为解耦。

首先应该在开发程序时保证不要胡乱传递只属于某一层特有的对象或跨层调用只属于某一层特有的方法，从而避免产生不必要的耦合。但是无论如何小心，层与层之间最终还是会有关系，对于这种无法避免的耦合，我们应该想办法管理起来。

解耦： 接口+配置文件+工厂

### 16.3.3.案例：EasyMall解耦

使用 接口+配置文件+工厂 进行EasyMall项目中层与层之间的解耦

**（1）引入接口**

将UserDao和UserService类改为UserDaoImpl和UserServiceImpl类, 在对应包下创建UserDao和UserService接口, 并让UserDaoImpl和UserServiceImpl分别实现UserDao和UserService接口，在接口中分别添加对应的方法。

**（2）引入配置文件**

在src下创建conf.properties文件, 添加如下配置:

|  |
| --- |
| UserService=cn.tedu.service.UserServiceImpl  UserDao=cn.tedu.dao.UserDaoImpl |

**（3）引入工厂**

**版本一：分别创建UserService和UserDao的工厂类，例如：**

|  |
| --- |
| 在factory包下创建UserDaoFactory 类，并添加如下代码：  **private** **static** UserDaoFactory *factory* = **new** UserDaoFactory();  **private** **static** Properties *prop* = **new** Properties();    **static**{  **try** {  String path = UserDaoFactory.**class**.getClassLoader().getResource("config.properties").getPath();  *prop*.load(**new** FileInputStream(path));  } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException();  }  }    /\*\*  \* 私有化构造方法 禁止外界new 工厂  \*/  **private** UserDaoFactory(){  }    /\*\*  \* 将工厂单例 从这个方法获取唯一的工厂  \* **@return**  \*/  **public** **static** UserDaoFactory getFactory(){  **return** *factory*;  }    /\*\*  \* 通过工厂 根据配置文件配置 获取UserDao的实现  \* **@return**  \* **@throws** Exception  \*/  **public** UserDao getInstance(){  **try** {  String userDaoStr = *prop*.getProperty("UserDao");  Class<?> clz = Class.*forName*(userDaoStr);  **return** (UserDao) clz.newInstance();  } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException();  }  } |
| 在需要new UserDaoImpl的地方只需要调用工厂类即可，例如：  **private** UserDao dao = UserDaoFactory.*getFactory*().getInstance(); |

UserServiceFactory与UserDaoFactory 代码类似，发现每个实现类都需要创建一个自己的factory，而代码又非常类似，因此，可以创建一个通用的工厂类。

**版本二：创建通用的BasicFactory工厂类**

|  |
| --- |
| 在factory包下创建BasicFactory类，并添加如下代码：  **private** **static** BasicFactory *factory* = **new** BasicFactory();  **private** **static** Properties *prop* = **new** Properties();    **static**{  **try** {  String path = BasicFactory.**class**.getClassLoader().getResource("config.properties").getPath();  *prop*.load(**new** FileInputStream(path));  } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException();  }  }    /\*\*  \* 私有化构造方法 禁止外界new 工厂  \*/  **private** BasicFactory(){}    /\*\*  \* 将工厂单例 从这个方法获取唯一的工厂  \* **@return**  \*/  **public** **static** BasicFactory getFactory(){  **return** *factory*;  }    /\*\*  \* 通过工厂 根据配置文件配置 获取UserDao的实现  \* **@return**  \* **@throws** Exception  \*/  **public** <T> T getInstance(Class<T> clazz){  **try** {  String className = *prop*.getProperty(clazz.getSimpleName());  Class clz = Class.*forName*(className);  **return** (T) clz.newInstance();  } **catch** (Exception e) {  e.printStackTrace();  **throw** **new** RuntimeException();  }  } |
| 然后修改UserServiceImpl中创建UserDao的方法，改为：  **private** UserDao dao=BasicFactory.*getFactory*().getInstance(UserDao.**class**);  分别修改RegistServlet、LoginServlet、AjaxCheckUsernameServlet中创建UserService 的方法，改为：  UserService service = BasicFactory.*getFactory*().getInstance(UserService.**class**); |

## 16.4.作业练习

1. MVC 的各个部分都有那些技术来实现?

答：View：视图层用jsp，javascript

作用--显示数据，接受用户输入数据

Controller：控制层用servlet

作用--接收视图层数据，传输给业务逻辑层（即模型层）

Model：模型层用普通java class

作用--业务类的实现

1. MVC的流程？

答：用户发送一个请求，首先由控制器接收，控制器根据请求去调用哪个业务逻辑层处理并返回数据，接着控制层调用相应的视图层返回给用户。

1. 耦合的概念？

答：所谓的耦合指的是在软件开发中，在层与层之间产生了某种紧密的关系，这种关系可能会导致，在我们修改或者是替换某一层时会影响其他的层，像这种情况我们就称之为层与层之间产生了耦合。

1. 写出几种常见的单例实现

答：单例模式可以分为懒汉式和饿汉式：

　　懒汉式单例模式：在类加载时不初始化。

　　饿汉式单例模式：在类加载时就完成了初始化，所以类加载比较慢，但获取对象的速度快。

懒汉式：（线程不安全）

public class SingletonDemo1 {

private static SingletonDemo1 instance;

private SingletonDemo1(){}

public static SingletonDemo1 getInstance(){

if (instance == null) {

instance = new SingletonDemo1();

}

return instance;

}

}

饿汉式：（线程安全）

public class SingletonDemo2 {

private static SingletonDemo2 instance = new SingletonDemo2();

private SingletonDemo2(){}

public static SingletonDemo2 getInstance(){

return instance;

}

}