JavaEE开发模式

2019年3月2日 14:05

1. 纯Servlet开发

   只用Servlet开发，html内容混在Java代码中不利于开发维护。

2. 纯JSP开发

   比起纯Servlet开发，JSP改为了在HTML编写java代码，便于页面开发。大量的java代码嵌在HTML中，同样非常混乱，不利于开发维护。

3. JSP+JSP标签技术开发

   将JSP中的java代码通过标签技术替代，减少了jsp页面中的java代码，jsp更加简洁。大量的自定义标签开发非常麻烦，大量的自定义标签混在html标签中，不利于开发维护。

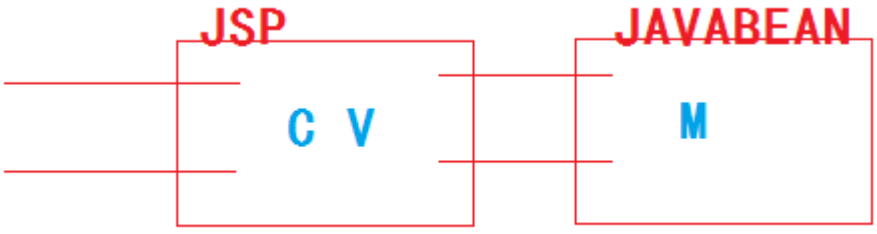
4. JSP + JavaBean(模式一)

   JSP负责接收请求 封装数据到Bean 调用Bean中的业务逻辑代码获取结果 展示数据

   JavaBean负责封装数据 处理数据业务逻辑

   初步实现了业务逻辑和数据展示的分离，结构变的跟加清晰，便于开发维护。

   JSP仍然在负责处理页面展示以外的其他功能，仍然不可避免的要编写java代码



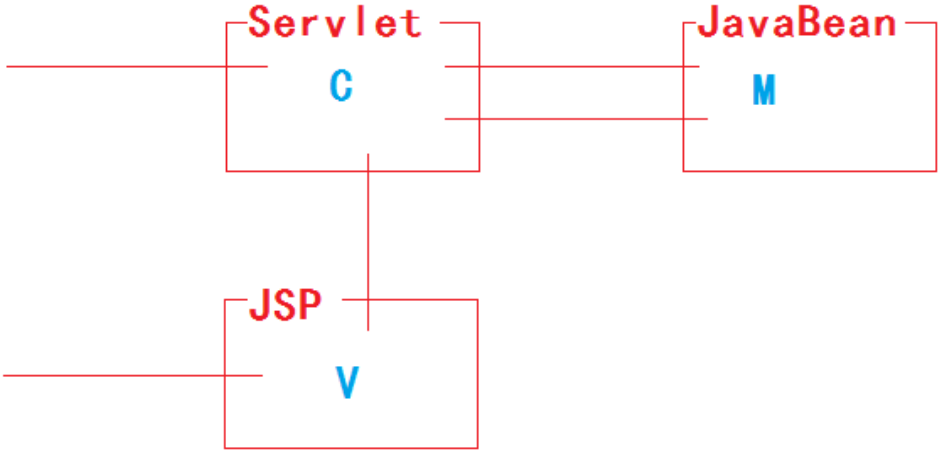
5. Servlet + JavaBean +JSP(模式二)

   Servlet负责接收请求，封装数据到Bean，调用Bean中的业务逻辑代码 获取结果数据 转发到JSP中展示

JavaBean负责封装数据 处理数据业务逻辑

JPS负责展示数据

每个部分只负责自己最擅长的工作，结构清晰，便于开发维护。



6. MVC设计思想

MVC设计模式认为，任何软件都可以分为负责程序执行过程的控制器(Controller) 负责业务数据分装处理的模型(Model) 和负责用户交互的视图(View)三部分组成。MVC设计思想要求软件在设计的过程中，应该将这三部分互相独立，使软件结构更加清晰，便于开发维护。

MVC设计思想不是JavaEE特有的思想，普遍应用在软件开发领域，指导软件的开发设计。

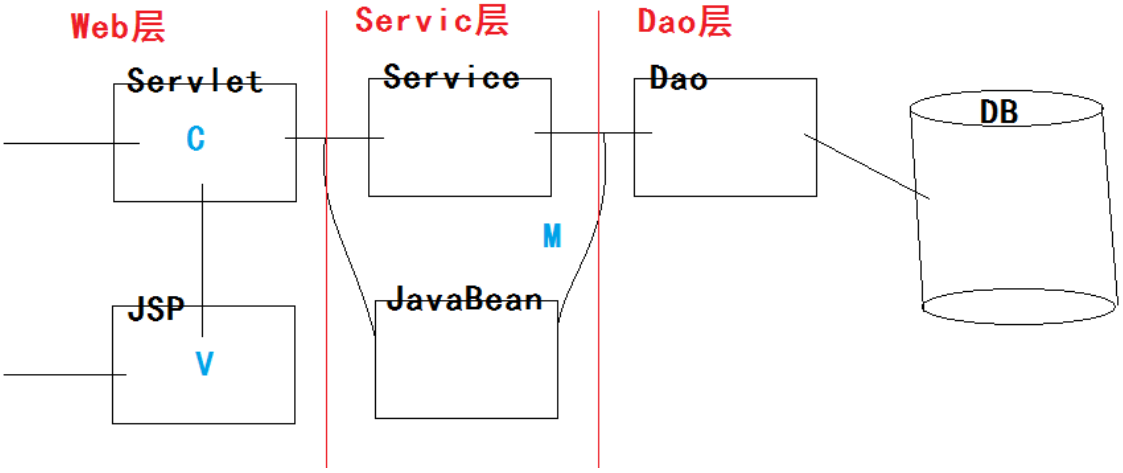
7. JavaEE的经典三层架构

在JavaEE的经典三层架构中，将JavaBean的功能进一步细分，JavaBean只负责数据封装，Service负责业务逻辑处理，Dao负责数据库的访问。

JavaEE的经典三层架仍然符合MVC设计模式，可以说是模式二的进一步发展。比起模式二，模块划分跟加清晰，各司其职，便于开发维护。

目前JavaEE开发中主要采用的就是这种开发模式。

JavaEE的经典三层架构

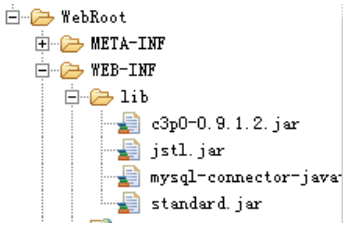


基于JavaEE的经典三层架构改造EasyMall

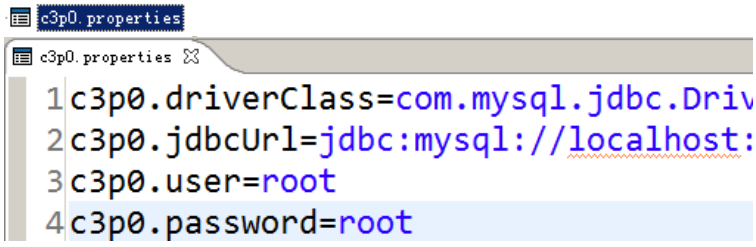
2019年3月2日 14:27

1. 创建EasyMall项目

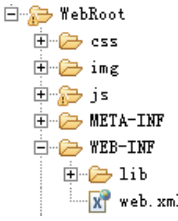
a. 导入第三方开发包



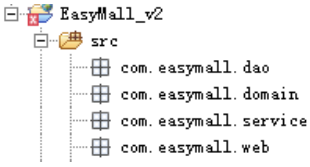
b. 导入配置文件



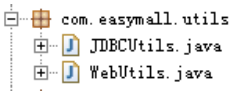
c. 导入其他资源文件



d. 创建出对应包结构

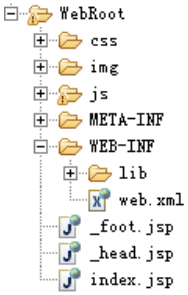


e. 导入相关工具类



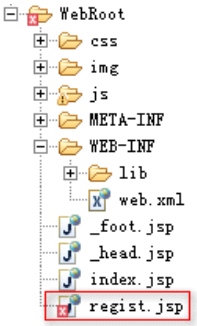
2. 开发主页

直接将原项目中的主页拷贝即可



3. 开发注册功能

a. 导入注册页面



b. 开发UserBean

```
package com.easymall.domain;

public class User {
	private int id;
	private String username;
	private String password;
	private String nickname;
	private String email;

	public User() {
	}

	public User(int id, String username, String password, String nickname, String email) {
		this.username = username;
		this.password = password;
		this.nickname = nickname;
		this.email = email;
	}

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public String getUsername() {
		return username;
	}
	public void setUsername(String username) {
		this.username = username;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
```

```
    }
    public String getNickname() {
        return nickname;
    }
    public void setNickname(String nickname) {
        this.nickname = nickname;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
}
```

c. 开发RegistServlet

在web包下创建RegistServlet，将原来的RegistServlet导入，将其中处理业务逻辑和数据访问的代码删除，改为调用Service中的业务方法

在这个过程中用到了Service中不存在的方法，则先创建出这个方法，后续再补。

```
package com.easymall.web;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.easymall.domain.User;
import com.easymall.exception.MsgException;
import com.easymall.service.UserService;
import com.easymall.utils.WebUtils;

public class RegistServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        //0.乱码处理
        //>>响应数据乱码处理
        response.setContentType("text/html;charset=utf-8");
        //>>请求数据乱码处理
        request.setCharacterEncoding("utf-8");

        //1.获取请求参数
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        String password2 = request.getParameter("password2");
```

```
String nickname = request.getParameter("nickname");
String email = request.getParameter("email");
String valistr = request.getParameter("valistr");

//验证码 session
//从session中取出ValistrSevlet生成的验证码文本
String code = (String) request.getSession().getAttribute("code");
//获取用户输入的验证码 --- 上方已经获取过了↑
//判断用户输入的数据和valistrServlet中的文本是否相同
if(!code.equalsIgnoreCase(valistr)){
    //验证码正确不需要任何操作，如果错误应该在页面中提示错误信息
    request.setAttribute("msg", "验证码不正确");
    request.getRequestDispatcher("/regist.jsp").forward(request, response);
    return;
}
//2.非空校验
if(WebUtils.isNull(username)){//利用工具类来校验
    //request可以用作域对象，
    //利用域对象身上的域将数据由当前servlet传递到regist.jsp中
    request.setAttribute("msg", "用户名不能为空");
    request.getRequestDispatcher("/regist.jsp").forward(request, response);
    return;
}
if(WebUtils.isNull(password)){//利用工具类来校验
    //request可以用作域对象，
    //利用域对象身上的域将数据由当前servlet传递到regist.jsp中
    request.setAttribute("msg", "密码不能为空");
    request.getRequestDispatcher("/regist.jsp").forward(request, response);
    return;
}
if(WebUtils.isNull(password2)){//利用工具类来校验
    //request可以用作域对象，
    //利用域对象身上的域将数据由当前servlet传递到regist.jsp中
    request.setAttribute("msg", "确认密码不能为空");
    request.getRequestDispatcher("/regist.jsp").forward(request, response);
    return;
}
if(WebUtils.isNull(nickname)){//利用工具类来校验
    //request可以用作域对象，
    //利用域对象身上的域将数据由当前servlet传递到regist.jsp中
    request.setAttribute("msg", "昵称不能为空");
    request.getRequestDispatcher("/regist.jsp").forward(request, response);
    return;
}
if(WebUtils.isNull(email)){//利用工具类来校验
    //request可以用作域对象，
    //利用域对象身上的域将数据由当前servlet传递到regist.jsp中
    request.setAttribute("msg", "邮箱不能为空");
```

```
            request.getRequestDispatcher("/regist.jsp").forward(request, response);
            return;
        }
        if(WebUtils.isNull(valistr)){//利用工具类来校验
            //request可以用作域对象，
            //利用域对象身上的域将数据由当前servlet传递到regist.jsp中
            request.setAttribute("msg", "验证码不能为空");
            request.getRequestDispatcher("/regist.jsp").forward(request, response);
            return;
        }
        //3.密码一致性校验
        if(password.trim() !="" && password2.trim() != ""
            && !password.trim().equals(password2.trim())){
            request.setAttribute("msg", "两次密码不一致");
            request.getRequestDispatcher("/regist.jsp").forward(request, response);
            return;
        }
        //4.邮箱格式校验
        String reg = "\\w+@\\w+(\\.\\w+)+";
        if(!email.matches(reg)){
            request.setAttribute("msg", "邮箱格式不正确");
            request.getRequestDispatcher("/regist.jsp").forward(request, response);
            return;
        }

        //5.调用Service完成用户注册功能
        //--封装数据到bean
        User user = new User(0,username,password,nickname,email);
        //--调用Service中的相关方法
        UserService service = new UserService();
        try{
            service.registUser(user);
        }catch (MsgException e) {
            //--底层出问题要提示用户 将错误信息展示回原来的页面
            request.setAttribute("msg", e.getMessage());
            request.getRequestDispatcher("/regist.jsp").forward(request, response);
            return;
        }

        //6.利用定时刷新返回首页
        response.getWriter().write("<h1 align='center'><font color='red'>恭喜注册成功，3秒后跳转
        回首页！</font></h1>");
        response.setHeader("refresh", "3;url=http://www.easymall.com");
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request, response);
```

```
    }

}
```

d. 开发UserService中的registUser方法

经过分析，注册用户时业务逻辑为，不允许注册重名用户，如果存在重名则抛出异常，如果不存在重名，则调用dao完成注册。

在其中用到了UserDao中未定义的方法，则先创建出该方法，后续再补。

```
package com.easymall.service;

import com.easymall.dao.UserDao;
import com.easymall.domain.User;
import com.easymall.exception.MsgException;

public class UserService {
    private UserDao userDao = new UserDao();
    /**
     * 注册用户的方法
     * @param user 封装了用户信息的bean
     * 业务逻辑：检查用户名是否已经存在，如果已经存在，则抛异常，如果不存在正常注册
     */
    public void registUser(User user) {
        User findUser = userDao.findUserByUserName(user.getUsername());
        if(findUser != null){
            //用户名已经存在，抛出异常
            throw new MsgException("用户名已存在！");
        }else{
            //用户名不存在，正常注册用户
            userDao.addUser(user);
        }

    }

}
```

在其中用到了自定义异常，自定义MsgException，为了用起来方便，将其定义为RuntimeException，并提供了可以接受信息的构造方法。

```
package com.easymall.exception;

public class MsgException extends RuntimeException{
    public MsgException() {
    }
    public MsgException(String msg) {
        super(msg);
    }
}
```

e. 开发UserDao中的方法

开发过程中用到了相关工具类，直接从原来的项目中拷贝即可

```
package com.easymall.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import com.easymall.domain.User;
import com.easymall.utils.JDBCUtils;

public class UserDao {

	/**
	 * 根据用户名查询用户
	 * @param username 要查找的用户名
	 * @return 找到的用户bean 如果没找到返回null
	 */
	public User findUserByUserName(String username) {
		//1.注册数据库驱动 - 有连接池，可以不用
		Connection conn = null;;
		PreparedStatement ps = null;
		ResultSet rs = null;
		try {
			//2.获取数据库连接 - 通过连接池获取
			conn = JDBCUtils.getConnection();
			//3.获取传输器
			ps = conn.prepareStatement("select * from user where username = ?");
			ps.setString(1,username);
			//4.传输sql执行
			rs = ps.executeQuery();
			//5.获取结果
			if(rs.next()){
				//查到了信息
				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{
				//没查到
				return null;
			}
		} catch (SQLException e) {
```

```
                e.printStackTrace();
                throw new RuntimeException(e);
            }finally{
                //6.关闭了连接 - 将conn还回连接池
                JDBCUtils.close(conn, ps, null);
            }
        }

        /**
         * 向数据库中User表新增一条User信息
         * @param user 封装了用户信息的bean
         * @throws SQLException
         */
        public void addUser(User user) {
            //1.注册数据库驱动 - 有连接池，可以不用
            Connection conn = null;;
            PreparedStatement ps = null;
            try {
                //2.获取数据库连接 - 通过连接池获取
                conn = JDBCUtils.getConnection();
                //3.获取传输器
                ps = conn.prepareStatement("insert into user values (null,?,?,?,?)");
                ps.setString(1, user.getUsername());
                ps.setString(2, user.getPassword());
                ps.setString(3, user.getNickname());
                ps.setString(4,user.getEmail());
                //4.传输sql执行
                ps.executeUpdate();
                //5.获取结果
            } catch (SQLException e) {
                e.printStackTrace();
                throw new RuntimeException(e);
            }finally{
                //6.关闭了连接 - 将conn还回连接池
                JDBCUtils.close(conn, ps, null);
            }
        }

}
```

4. 开发登录功能

a. 作业1：自己完成登录功能开发

b. 作业2：更新主页，根据登录状态不同，提示不同信息

在_head.jsp中通过cif判断用户是否登录，提示不同内容

```
<c:if test="${sessionScope.user == null}">
    <a href="${pageContext.request.contextPath}/login.jsp">登录</a>  |  
    <a href="${pageContext.request.contextPath}/regist.jsp">注册</a>
```

```
</c:if>
<c:if test="${sessionScope.user != null}">
    欢迎回来，${sessionScope.user.username }！    <a href="#">注销</a>  
</c:if>
```

c. 作业3：开发注销功能

修改_head.jsp中的注销链接到LogoutServlet

开发LogoutServlet

```
package com.easymall.web;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class LogoutServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        if(request.getSession(false) != null){
            request.getSession().invalidate();
        }
        response.sendRedirect("/index.jsp");
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request, response);

    }

}
```

d. 作业4：配置EasyMall全站友好错误提示页面