## 实验八、Struts2或Spring MVC框架

### 一、实验目的

1．理解Struts2或SpringMVC体系结构核心组件及运行机制，控制器组件、模型组件、视图组件的设计方法，Struts2或SpringMVC 的工作流程；

2．掌握Struts2或SpringMVC 开发环境的构建；

3．掌握Struts2或SpringMVC配置方法及常用标记库的使用方法。

4、\*了解SpringMVC 与 Spring 、Mybatis集成技术（SSM）或Spring与Struts、Hibernate集成技术（SSH）

重点：SpringMVC框架及SSM（SSH）集成技术（SSH）。

### 二、实验内容与要求

1、基于Struts2或SpringMVC自行设计实现一个面向学校教学管理应用背景的个人信息管理Web系统，包括包括基于MyBaits的对用户信息的增删查改等基本操作；从而加深对MVC架构的理解，并掌握Struts2或SpringMVC的框架应用和开发技巧。

2、基本概念和原理：

A、Struts2是一个基于MVC设计模式的Web应用框架，它本质上相当于一个servlet，在MVC设计模式中，Struts2作为控制器(Controller)来建立模型与视图的数据交互。Struts 2以WebWork为核心，采用拦截器的机制来处理用户的请求，这样的设计也使得业务逻辑控制器能够与ServletAPI完全脱离开，Struts2的8个核心文件为：

struts2-core-2.x.x.jar :Struts 2框架的核心类库

xwork-2.x.x.jar : XWork类库，Struts 2在其上构建

ognl-2.6.x.jar : 对象图导航语言（Object Graph Navigation Language），Struts 2框架使用的一种表达式语言

freemarker-2.3.x.jar : Struts 2的UI标签的模板使用FreeMarker编写

commons-logging-1.1.x.jar: ASF出品的日志包，Struts 2框架使用这个日志包来支持Log4J和JDK 1.4+的日志记录。

其余的核心类库还包括commons-fileupload－1.2.2.jar、commons-lang-2.x.xjar、commons-io-2.X.X.jar、javassist-3.x.x.ar

B、Spring MVC所有的请求都经过DispatcherServlet来统一分发。DispatcherServlet将请求分发给Controller之前，需要借助于Spring MVC提供的HandlerMapping定位到具体的Controller。HandlerMapping接口负责完成客户请求到Controller映射。Controller接口将处理用户请求，这和Java Servlet扮演的角色是一致的。一旦Controller处理完用户请求，则返回ModelAndView对象给DispatcherServlet前端控制器，ModelAndView中包含了模型（Model）和视图（View）。从宏观角度考虑，DispatcherServlet是整个Web应用的控制器；从微观考虑，Controller是单个Http请求处理过程中的控制器，而ModelAndView是Http请求过程中返回的模型（Model）和视图（View）。ViewResolver接口（视图解析器）在Web应用中负责查找View对象，从而将相应结果渲染给客户。

Spring的4个核心JAR包、commons-logging的JAR包以及两个Web相关的JAR包（spring-web-5.0.2.RELEASE.jar和spring-webmvc-5.0.2.RELEASE.jar）。另外，在Spring MVC应用中使用注解时，别忘记添加spring-aop-5.0.2.RELEASE.jar包。

3、实验要求：

（1）能够基于MVC模式进行系统设计；

（2）能够根据系统设计对涉及到的Model、View和Controller进行编程实现

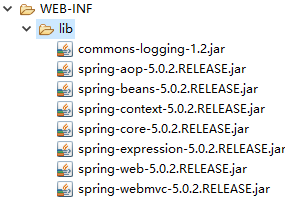
（3）\*能够利用数据库技术实现数据的存储和读取

### 三、实验步骤

1、第一个Spring MVC应用

（1） 创建Web应用并引入JAR包

在Eclipse中，创建一个名为ch9的Web应用，在ch9的lib目录中添加Spring MVC程序所需要的JAR包，包括Spring的4个核心JAR包、commons-logging的JAR包以及两个Web相关的JAR包（spring-web-5.0.2.RELEASE.jar和spring-webmvc-5.0.2.RELEASE.jar）。另外，在Spring MVC应用中使用注解时，别忘记添加spring-aop-5.0.2.RELEASE.jar包。添加后的JAR包如图所示。



1. 在web.xml文件中部署DispatcherServlet

在开发Spring MVC应用时，需要在web.xml中部署DispatcherServlet

<!--部署DispatcherServlet-->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 表示容器在启动时立即加载servlet -->

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<!-- 处理所有URL-->

<url-pattern>/</url-pattern>

</servlet-mapping>

上述DispatcherServlet的servlet对象springmvc初始化时，将在应用程序的WEB-INF目录下查找一个配置文件（见后第5步），该配置文件的命名规则是“servletName-servlet.xml”，如：springmvc-servlet.xml。

1. 创建Web应用首页

页面核心内容为用户动作的模拟，包括注册和登录两个动作：

<body>

没注册的用户，请<a href="${pageContext.request.contextPath }/register">注册</a>！<br>

已注册的用户，去<a href="${pageContext.request.contextPath }/login">登录</a>！

</body>

1. 创建Controller类

在src目录下，创建包controller，并在该包中创建“RegisterController”和“LoginController”两个传统风格的控制器类（实现了Controller接口），分别处理首页中“注册”和“登录”超链接请求。

RegisterController.java：

package controller;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.ModelAndView;

import org.springframework.web.servlet.mvc.Controller;

public class RegisterController implements Controller{

@Override

public ModelAndView handleRequest(HttpServletRequest arg0, HttpServletResponse arg1) throws Exception {

//return new ModelAndView("/WEB-INF/jsp/register.jsp");

return new ModelAndView("register");

}

}

LoginController.java

package controller;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.ModelAndView;

import org.springframework.web.servlet.mvc.Controller;

public class RegisterController implements Controller{

@Override

public ModelAndView handleRequest(HttpServletRequest arg0, HttpServletResponse arg1) throws Exception {

//return new ModelAndView("/WEB-INF/jsp/register.jsp");

return new ModelAndView("register");

}

}

1. 创建Spring MVC配置文件并配置Controller映射信息

传统风格的控制器定义后，需要在Spring MVC配置文件中部署它们（学习基于注解的控制器后，不再需要部署控制器）。在WEB-INF目录下，创建名为springmvc-servlet.xml的配置文件（文件名命名规则，见第2步）

springmvc-servlet.xml：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<!--LoginController控制器类，映射到“/login” -->

<bean name="/login" class="controller.LoginController"/>

<!--RegisterController控制器类，映射到“/register” -->

<bean name="/register" class="controller.RegisterController"/>

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"

id="internalResourceViewResolver">

<!-- 前缀 -->

<property name="prefix" value="/WEB-INF/jsp/" />

<!-- 后缀 -->

<property name="suffix" value=".jsp" />

</bean>

</beans>

1. 创建视图

RegisterController控制器处理成功后，跳转到“/WEB-INF/jsp/register.jsp”视图；LoginController控制器处理成功后，跳转到“/WEB-INF/jsp/login.jsp”视图。因此，应用的“/WEB-INF/jsp”目录下应有“register.jsp”和“login.jsp”页面，此两个JSP页面代码如下：

login.jsp：

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

</head>

<body>

登录页面。

</body>

</html>

register.jsp

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

</head>

<body>

注册页面。

</body>

</html>

（7）发布并运行Spring MVC应用

在Eclipse中第一次运行Spring MVC应用时，需要将应用发布到Tomcat。可以选中应用名称后单击右键，选择【Run As】/【Run on Server】打开对话框，在对话框中单击【Finish】即完成发布并运行。

2、SpringMVC编程详解：

（1）Controller模块编程：基于注解的控制器模块编程，使用基于注解的控制器，具有如下两个优点：A．在基于注解的控制器类中，可以编写多个处理方法，进而可以处理多个请求（动作）。这就允许将相关的操作编写在同一个控制器类中，从而减少控制器类的数量，方便以后的维护。B．基于注解的控制器不需要在配置文件中部署映射，仅需要使用RequestMapping注释类型注解一个方法进行请求处理。在Spring MVC中，最重要的两个注解类型是Controller和RequestMapping，具体做法如下。

indexController.java（使用类别级注解）：

package controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

@Controller

@RequestMapping("/index")

public class IndexController {

@RequestMapping("/login")

public String login() {

return "login";//跳转到“/WEB-INF/jsp/login.jsp”

}

@RequestMapping("/register")

public String register() {

return "register";

}

}

UserController.java（使用类别级注解）

package controller;

import javax.servlet.http.HttpSession;

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.RequestMapping;

import pojo.UserForm;

import service.UserService;

@Controller

@RequestMapping("/user")

public class UserController {

//得到一个用来记录日志的对象，这样打印信息的时候能够标记打印的是那个类的信息

private static final Log logger = LogFactory.getLog(UserController.class);

//将服务依赖注入到属性userService

@Autowired

public UserService userService;

/\*\*

\* 处理登录

\*/

@RequestMapping("/login")

public String login(UserForm user, HttpSession session, Model model) {

if(userService.login(user)){

session.setAttribute("u", user);

logger.info("成功");

return "main";//登录成功，跳转到main.jsp

}else{

logger.info("失败");

model.addAttribute("messageError", "用户名或密码错误");

return "login";

}

}

/\*\*

\*处理注册

\*/

@RequestMapping("/register")

public String register(@ModelAttribute("user") UserForm user) {

if(userService.register(user)){

logger.info("成功");

return "login";//注册成功，跳转到login.jsp

}else{

logger.info("失败");

//使用@ModelAttribute("user")与model.addAttribute("user", user)功能相同

//在register.jsp页面上可以使用EL表达式${user.uname}取出ModelAttribute的uname值

return "register";//返回register.jsp

}

}

}

（2）Model模块编程

用户表单对应pojo类UserForm.java：

package pojo;

public class UserForm {

private String uname;//与请求参数名称相同

private String upass;

private String reupass;

public String getUname() {

return uname;

}

public void setUname(String uname) {

this.uname = uname;

}

public String getUpass() {

return upass;

}

public void setUpass(String upass) {

this.upass = upass;

}

public String getReupass() {

return reupass;

}

public void setReupass(String reupass) {

this.reupass = reupass;

}

}

对应的login和register服务model设计：

UserService.java

package service;

import pojo.UserForm;

public interface UserService {

boolean login(UserForm user);

boolean register(UserForm user);

}

UserServiceImpl.java

package service;

import org.springframework.stereotype.Service;

import pojo.UserForm;

//注解为一个服务

@Service

public class UserServiceImpl implements UserService{

@Override

public boolean login(UserForm user) {

if("zhangsan".equals(user.getUname())

&& "123456".equals(user.getUpass()))

return true;

return false;

}

@Override

public boolean register(UserForm user) {

if("zhangsan".equals(user.getUname())

&& "123456".equals(user.getUpass()))

return true;

return false;

}

}

（3）view模块编程

login.jsp：

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>后台登录</title>

<style type="text/css">

table{

text-align: center;

}

.textSize{

width: 120px;

height: 25px;

}

\* {

margin: 0px;

padding: 0px;

}

body {

font-family: Arial, Helvetica, sans-serif;

font-size: 12px;

margin: 10px 10px auto;

background-image: url(${pageContext.request.contextPath }/images/bb.jpg);

}

</style>

<script type="text/javascript">

//确定按钮

function gogo(){

document.forms[0].submit();

}

//取消按钮

function cancel(){

document.forms[0].action = "";

}

</script>

</head>

<body>

<form action="${pageContext.request.contextPath }/user/login" method="post">

<table>

<tr>

<td colspan="2"><img src="${pageContext.request.contextPath }/images/login.gif"></td>

</tr>

<tr>

<td>姓名：</td>

<td><input type="text" name="uname" class="textSize"></td>

</tr>

<tr>

<td>密码：</td>

<td><input type="password" name="upass" class="textSize"></td>

</tr>

<tr>

<td colspan="2">

<input type="image" src="${pageContext.request.contextPath }/images/ok.gif" onclick="gogo()" >

<input type="image" src="${pageContext.request.contextPath }/images/cancel.gif" onclick="cancel()" >

</td>

</tr>

</table>

${messageError }

</form>

</body>

</html>

register.jsp

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<style type="text/css">

.textSize{

width: 100pt;

height: 15pt

}

</style>

<title>注册画面</title>

<script type="text/javascript">

//注册时检查输入项

function allIsNull(){

var name=document.registForm.uname.value;

var pwd=document.registForm.upass.value;

var repwd=document.registForm.reupass.value;

if(name==""){

alert("请输入姓名！");

document.registForm.uname.focus();

return false;

}

if(pwd==""){

alert("请输入密码！");

document.registForm.upass.focus();

return false;

}

if(repwd==""){

alert("请输入确认密码！");

document.registForm.reupass.focus();

return false;

}

if(pwd!=repwd){

alert("2次密码不一致，请重新输入！");

document.registForm.upass.value="";

document.registForm.reupass.value="";

document.registForm.upass.focus();

return false;

}

document.registForm.submit();

return true;

}

</script>

</head>

<body>

<form action="${pageContext.request.contextPath }/user/register" method="post" name="registForm">

<table

border=1

bgcolor="lightblue"

align="center">

<tr>

<td>姓名：</td>

<td>

<input class="textSize" type="text" name="uname" value="${user.uname }"/>

</td>

</tr>

<tr>

<td>密码：</td>

<td><input class="textSize" type="password" maxlength="20" name="upass"/></td>

</tr>

<tr>

<td>确认密码：</td>

<td><input class="textSize" type="password" maxlength="20" name="reupass"/></td>

</tr>

<tr>

<td colspan="2" align="center"><input type="button" value="注册" onclick="allIsNull()"/></td>

</tr>

</table>

</form>

</body>

</html>

（4）配置文件设置

web.xml:

<?xml version="1.0" encoding="UTF-8"?>

<web-app

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"

id="WebApp\_ID" version="3.1">

<!--部署DispatcherServlet-->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 表示容器在启动时立即加载servlet -->

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<!-- 处理所有URL-->

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

springmvc-servlet.xml:

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd">

<!-- 使用扫描机制，扫描控制器类 -->

<context:component-scan base-package="controller"/>

<context:component-scan base-package="service"/>

<mvc:annotation-driven />

<!-- annotation-driven用于简化开发的配置，

注解DefaultAnnotationHandlerMapping和AnnotationMethodHandlerAdapter -->

<!-- 使用resources过滤掉不需要dispatcher servlet的资源。

使用resources时，必须使用annotation-driven，不然resources元素会阻止任意控制器被调用。

如果不使用resources，则annotation-driven可以没有。 -->

<!-- 允许css目录下所有文件可见 -->

<mvc:resources location="/css/" mapping="/css/\*\*"></mvc:resources>

<!-- 允许html目录下所有文件可见 -->

<mvc:resources location="/html/" mapping="/html/\*\*"></mvc:resources>

<!--允许images目录下所有文件可见 -->

<mvc:resources location="/images/" mapping="/images/\*\*"></mvc:resources>

<!-- 配置视图解析器 -->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"

id="internalResourceViewResolver">

<!-- 前缀 -->

<property name="prefix" value="/WEB-INF/jsp/" />

<!-- 后缀 -->

<property name="suffix" value=".jsp" />

</bean>

</beans>

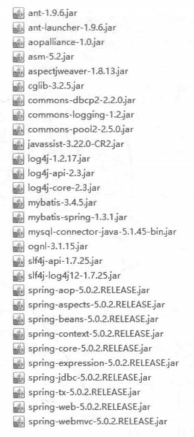
1. 扩展完成：
2. 将数据库引入上述项目中，实现基于用户信息在数据库的查询结果确定login的成功与否
3. 将数据库引入上述项目中，实现基于用户信息在数据库的更新实现register功能
4. 引入一个error视图，在login和register过程中出现错误时进入统一的错误信息提示界面。
5. SSM框架集成技术：

因为Spring MVC是Spring框架中的一个子模块，所以Spring与Spring MVC之间不存在整合的问题。实际上SSM框架的整合只涉及Spring与MyBatis的整合，以及Spring MVC与MyBatis的整合

SSM框架整合思路：以一个应用案例（根据用户名模糊查询用户信息，用户表是数据表user），使用SSM框架实现该案例的具体步骤如下：

1. SSM框架整合所需JAR包

实现SSM框架的整合，首先需要准备三个框架的JAR包以及其他整合所需要的JAR包。在第6章6.5节已经讲解了Spring与MyBatis框架整合所需要的JAR包，在本章只需再加入Spring MVC的相关JAR包（spring-web-5.0.2.RELEASE.jar和spring-webmvc-5.0.2.RELEASE.jar）即可。因此，SSM框架整合所需的JAR包如图所示。



1. 创建应用并导入相关JAR包

创建应用，将图所示的JAR包复制到应用的lib中。因为，案例中使用JSTL标签显示查询结果，所以还需要将JSTL标签相关的JAR包taglibs-standard-impl-1.2.5.jar和taglibs-standard-spec-1.2.5.jar复制到应用的lib中。

1. 创建信息输入页面

<body>

<form action="${pageContext.request.contextPath }/select" method="post">

输入用户名：<input type="text" name="uname"/><br>

<input type="submit" value="提交"/>

</form>

</body>

1. 创建持久化类

package com.po;

/\*\*

\*springtest数据库中user表的持久化类

\*/

public class MyUser {

private Integer uid;

private String uname;

private String usex;

//省略setter和getter方法

}

1. 创建DAO层

在src目录下，创建一个名为com.dao的包，并在该包中创建一个名为UserDao的接口，该接口使用@Mapper注解自动装配为MyBatis的映射接口

@Repository("userDao")

@Mapper

/\*使用Spring自动扫描MyBatis的接口并装配

（Spring将指定包中所有被@Mapper注解标注的接口自动装配为MyBatis的映射接口\*/

public interface UserDao {

/\*\*

\* 接口方法对应SQL映射文件UserMapper.xml中的id

\*/

public List<MyUser> selectUserByUname(MyUser user);

}

1. 创建Service层

在src目录下，创建一个名为com.service的包，并在该包中创建一个名为UserService的接口和该接口的实现类UserServiceImpl。

public interface UserService {

public List<MyUser> selectUserByUname(MyUser user);

}

@Service("userService")

@Transactional

/\*\*加上注解@Transactional,可以指定这个类需要受Spring的事务管理

注意@Transactional只能针对public属性范围内的方法添加，

本案例并不需要处理事务，在这里只是告诉读者如何使用事务\*/

public class UserServiceImpl implements UserService{

@Autowired

private UserDao userDao;

@Override

public List<MyUser> selectUserByUname(MyUser user) {

return userDao.selectUserByUname(user);

}

}

1. 创建Controller层

@Controller

public class UserController {

@Autowired

private UserDao userDao;

@RequestMapping("/select")

public String select(MyUser user, Model model) {

List<MyUser> list = userDao.selectUserByUname(user);

model.addAttribute("userList", list);

return "userList";

}

}

1. 创建用户信息显示页面

<body>

用户信息<br>

<c:forEach items="${userList}" var="user">

${user.uid}&nbsp;&nbsp;&nbsp;&nbsp;

${user.uname}&nbsp;&nbsp;&nbsp;&nbsp;

${user.usex}<br>

</c:forEach>

</body>

1. 创建相关配置文件

1）web.xml

在WEB-INF目录下，创建web.xml文件，并在该文件中实例化ApplicationContext容器、启动Spring容器、配置DispatcherServlet以及配置字符编码过滤器。

<!-- 实例化ApplicationContext容器 -->

<context-param>

<!-- 加载src目录下的applicationContext.xml文件 -->

<param-name>contextConfigLocation</param-name>

<param-value>

classpath:applicationContext.xml

</param-value>

</context-param>

<!-- 指定以ContextLoaderListener方式启动Spring容器 -->

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

2）springmvc-servlet.xml

在WEB-INF目录下，创建Spring MVC的核心配置文件springmvc-servlet.xml。该文件中仅配置控制器扫描包和视图解析器。

<!-- 使用扫描机制，扫描包 -->

<context:component-scan base-package="com.controller" />

<!-- 配置视图解析器 -->

<bean

class="org.springframework.web.servlet.view.InternalResourceViewResolver"

id="internalResourceViewResolver">

<!-- 前缀 -->

<property name="prefix" value="/WEB-INF/jsp/" />

<!-- 后缀 -->

<property name="suffix" value=".jsp" />

</bean>

3）log4j.properties

# Global logging configuration

log4j.rootLogger=ERROR, stdout

# MyBatis logging configuration...

log4j.logger.com.dao=DEBUG

# Console output...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

1. applicationContext.xml

在src目录下，创建Spring的配置文件applicationContext.xml。在该文件中配置数据源、添加事务支持、开启事务注解、配置MyBatis工厂、Mapper代理开发以及扫描包。

5）创建MyBatis的核心配置文件和SQL映射文件

src目录下，创建com.mybatis文件夹，在该文件夹中创建MyBatis的核心配置文件mybatis-config.xml。

<configuration>

<mappers><!-- 映射器，告诉 MyBatis到哪里去找映射文件-->

<mapper resource="com/mybatis/UserMapper.xml"/>

</mappers>

</configuration>

<mapper namespace="com.dao.UserDao">

<!-- 查询用户信息，id的值对应dao的接口方法 -->

<select id="selectUserByUname" resultType="com.po.MyUser" parameterType="com.po.MyUser">

select \* from user where 1=1

<if test="uname !=null and uname!=''">

and uname like concat('%',#{uname},'%')

</if>

</select>

</mapper>

1. 发布并运行应用

### 四、附录

### A、Struts2框架下的参考代码与配置文件

1、框架搭建：

新建Web Project ,命名为：Struts2Demo，Add struts Capabilities为该工程添加Struts支持（将Struts2的8个核心文件复制到项目的“WebRoot\WEB-INF\lib”目录中）。

2、修改“WebRoot\WEB-INF\web.xml”文件，配置Struts2过滤器，内容如下：

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" id="WebApp\_ID" version="3.0">

<display-name>Struts2Demo</display-name>

<filter>

<filter-name>struts2</filter-name>

<filter-class>

org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>\*.action</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>\*.jsp</url-pattern>

</filter-mapping>

<welcome-file-list>

。。。。。。。。。

</welcome-file-list>

</web-app>

3. 在webRoot下建立两个文件login.jsp,loginresult.jsp。

可直接将所给代码中的jsp拷贝到自己工程下的webroot下

(1) login.jsp代码：

<%@ page language="java" import="java.util.\*" pageEncoding="GBK"%>

<html>

<body>

<form action="login.action">

username:<input type="text" name="username"/><br>

password:<input type="password" name="password"/><br>

<input type="submit"/>

</form>

</body>

</html>

(2) loginresult.jsp代码：

<%@ taglib prefix="s" uri="/struts-tags" %>

<%@ page language="java" contentType="text/html; charset=GBK"

pageEncoding="GBK"%>

<html>

<body>

<p>

在线人数：

<s:property value="#application.onlineUserCount"/>

</p>

<p>

<a href="logout.action">退出</a>

</p>

</body>

</html>

4. 在src目录下建立包：test，在test包下建立Login.java

代码如下：（也可直接拷贝所给代码的中的test包到本机的src下）

package test;

import javax.servlet.ServletContext;

import org.apache.struts2.ServletActionContext;

import com.opensymphony.xwork2.ActionSupport;

public class Login extends ActionSupport{

private String username;

private String password;

public void setUsername(String username) {

this.username = username;

}

public String getUsername() {

return username;

}

public void setPassword(String password) {

this.password = password;

}

public String getPassword() {

return password;

}

public String login(){

if(username.equals("admin")&&password.equals("admin")){

int onlineUserCount = 0;

ServletContext servletContext = ServletActionContext.

getServletContext();

synchronized (servletContext) {

try {

onlineUserCount = (Integer) servletContext

.getAttribute("onlineUserCount");

} catch (Exception e) {

}

servletContext.setAttribute("onlineUserCount",

onlineUserCount + 1);

}

return "success";

}else{

return "input";

}

}

public String logout() {

//在此处编写退出代码，将在线人数减1后返回登录界面login.jsp。

return "success";

}

}

5. 编写struts.xml，可直接将所给代码中src目录下的struts.xml拷贝到自己工程下的src目下。代码如下：

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<constant name="struts.devMode" value="true" />

<package name="test" namespace="" extends="struts-default">

<action name="logininput">

<result>/login.jsp</result>

</action>

<action name="login" class="test.Login" method="login">

<result name="success">/loginresult.jsp</result>

<result name="input">/login.jsp</result>

</action>

</package>

</struts>

6. 请完成如下功能：

（1）. 填写退出部分的Login.java中的logout部分代码。

（2）. 填写退出部分的struts.xml中对动作logout的配置部分

（3）．部署程序到tomcat下，运行tomcat.打开浏览器，请根据自己的工程情况修改连接：<http://localhost:8082/myproject/logininput.action>，查看登录后在线人数是否变化。

（4）修改Login.java，让其继承ActionSupport子类并实现ModelDriven接口，用模型驱动方式完成同样的功能（需定义一个model即javabean）。

7、扩展编程：

使用ActionSupport类实现输入信息的校验：用户名输入admin，且密码输入sise后登录到welcome.jsp页面，如果没有输入用户名或密码将在相应字段上面显示提示信息，如果输入错误用户名或错误密码将在用户名字段上面显示提示信息。实验效果如下图所示：

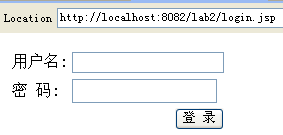
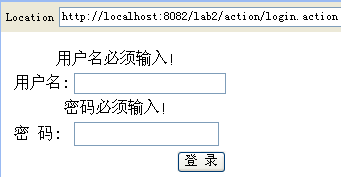
 

图1 登录页面 图2 用户名和密码为空

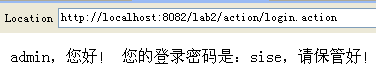
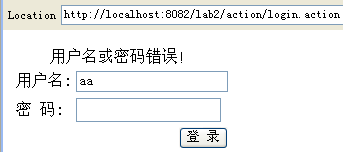


图3 用户名或密码不对 图4 用户名和密码输入正确

2、修改Struts2Demo项目，实现新闻的编辑、修改、浏览功能，其中新闻的实体类中，应该含有：新闻标题、新闻内容、作者、日期等信息。

使用validate()校验失败时，转到哪个逻辑视图？

如何将字段校验错误信息显示出来？

如何使用命名空间？

### B、Spring与Struts 2的整合步骤介绍（图书管理系统登录案例）

① 创建Web项目SpringStruts2。

② 添加Struts 2框架。添加8个Struts　Jar包：

配置web.xml，代码如下：

<web-app version="2.5”....... >

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

</web-app>

③ 创建login.jsp。

<%@ page language="java" pageEncoding="utf-8"%>

<%@ taglib uri="/struts-tags" prefix="s" %>

<html>

<head> <title>登录界面</title> </head>

<body>

<s:form action="login.action" method="post">

<s:textfield name="xh" label="学号"/>

<s:password name="kl" label="口令"/>

<s:submit value="登录"/>

</s:form>

</body>

</html>

④ 创建Action。

package cs2011.action;

import com.opensymphony.xwork2.ActionSupport;

public class LoginAction extends ActionSupport{

private String xh;

private String kl;

//setter/getter方法

public String execute() throws Exception {

return SUCCESS;

}

}

配置struts.xml文件：

<struts>

<include file="struts-default.xml"/>

<package name="default" extends="struts-default">

<action name="login" class="cs2011.action.LoginAction">

<result name="success">/login\_success.jsp</result>

</action>

</package>

</struts>

⑤ 创建login\_success.jsp。

<%@ page contentType="text/html;charset=gb2312" %>

<%@ taglib prefix="s" uri="/struts-tags" %>

<html>

<body >

<h2>您好! <s:property value=" xh"/>欢迎您登录成功 </h2>

</body>

</html>

⑥ 部署运行。

部署，测试Struts 2是否正常。运行http://localhost:8080/Struts\_Spring/login.jsp。

⑦ 添加Spring框架。

⑧ 添加Spring支持包struts2-spring-plugin.jar。注意，一定要加入该包，该包位于Struts 2的lib目录下。

⑨ 修改web.xml内容。

修改web.xml内容，增加对Spring的支持。

<web-app >

<filter>

</filter>

<filter-mapping>

</filter-mapping>

<listener >

<listener-class >

org.springframework.web.context.ContextLoaderListener

</listener-class >

</listener >

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>

<!--此路径为Spring配置文件applicationContext.xml路径-->

/WEB-INF/classes/applicationContext.xml

</param-value>

</context-param>

</web-app>

⑩ 创建消息包文件struts.properties。

在src文件夹下创建struts.properties文件,内容如下：

struts.objectFactory=spring

⑾修改applicationContext.xml。

<?xml version="1.0" encoding="UTF-8"?>

<beans >

<bean id="loginAction" class=“cs2011.action.LoginAction"></bean>

</beans>

⑿ 修改struts.xml。

使得struts 2的类的生成交给Spring完成，注意与步骤4中的不同点。

<struts>

<include file="struts-default.xml"/>

<package name="default" extends="struts-default">  
 <!--使用Spring生成的类对象 -->

<action name="login" class=" loginAction">

<result name="success">/login\_success.jsp</result>

</action>

</package>

</struts>

⒀发布Web应用与测试。

　　发布后，注意项目发布目录lib下的jar包是否有重复，保留Struts2版本2.3的文件，重复的包删除。

　　启动TOMCAT进行测试

### C、Spring、Struts、Hibernate整合步骤介绍（图书管理系统登录案例）：

（1）新建Web Project：Struts2HibernateSpringDemo;

（2）新建包：csexp.model、csexp.dao、csexp.service、sc2011.action;

（3）添加Struts2支持包；freemarker,ognl,struts2-core,xworks

（4）修改WEB-INF/web.xml,增加对Struts2支持：

<web-app>

....

<filter>

<filter-name>struts2</filter-name>

<filter-

class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-

class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

......

</web-app>

（5）添加Spring框架支持,配置文件为src/applicationContext.xml；

Spring Aop　、 Spring core　、 Spring Persistence core

Spring Persistence JDBC　、 Spring Web Liberaries

（6）添加Spring Struts2支持包struts2-spring-plugin.jar;

（7）修改WEB-INF/web.xml,增加对Spring的支持：

<web-app>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/classes/applicationContext.xml</param-value>

</context-param>

<listener>

<listener-

class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

......

</web-app>

（8）创建资源文件struts.properties,使得Struts2的类的生成交给Spring完成：

struts.objectFactory=spring

（9）添加Hibernate支持,选择使用"Spring configuration file",并选择“Existing Spring

configuration file”,配置文件为src/applicationContext.xml；

（10）选择数据源连接，使用csexp连接配置或新配置数据源信息，

Bean Id为dataSource；

（11）“Next",不创建SessionFactory,使用Spring为注入sessionFactory；

（12）修改applicationContext.xml，添加HibernateTemplate类bean hibTemplate:

...

<bean id="hibTemplate"

class="org.springframework.orm.hibernate3.HibernateTemplate">

<property name="sessionFactory">

<ref bean="sessionFactory"/>

</property>

</bean>

</beans>

（13）利用Hibernate逆向工程，创建csexp.model.User类（表为test.users);

（14） 新建csexp.dao包下的IUserDao接口类和实现类UserDaoImp:

//1.IUserDao.java

package csexp.dao;

import csexp.model.User;

public interface IUserDao {

// 查找指定用户

public Boolean find(User user);

// 查找指定用户

public User findUserByName(String username);

// 以ID查找用户

public User find(int id);

// 查找所有用户

public List<User> findAll();

// 添加用户

public Boolean insert(User user);

// 更新（修改）用户

public Boolean update(int id, User user);

// 删除指定用户

public Boolean delete(int id);

}

//2.UserDaoImp.java

public class UserDaoImp implements IUserDao {

private HibernateTemplate hibTemplate ;

//注入hibTemplate

public void setHibTemplate(HibernateTemplate hibTemplate) {

this.hibTemplate = hibTemplate;

}

// 查找指定用户

public Boolean find(User user) {

Boolean result = false;

String hql = "from User where name=? and password=?";

List<User> list = hibTemplate.find(hql,

new String[]{user.getName(), user.getPassword()});

if (list.size() >= 1)

result = true;

return result;

}

// 查找指定用户

public User findUserByName(String username) {

User user = null ;

String hql = "from User where name=?";

List<User> list = hibTemplate.find(hql, username);

if(list.size()>=1)

user = list.get(0);

return user;

}

// 以ID查找用户

public User find(int id) {

User user = null;

String hql = "from User where id=?";

//List<User> list = hibTemplate.find(hql, String.valueOf(id));

List<User> list = hibTemplate.find(hql, new Integer(id));

if(list.size()>=1)

user = list.get(0);

return user;

}

// 查找所有用户

public List<User> findAll() {

String hql = "from User";

List<User> list = hibTemplate.find(hql);

return list;

}

// 添加用户

public Boolean insert(User user) {

Boolean result = false;

if(findUserByName(user.getName()) == null){

hibTemplate.save(user);

result = true;

}

else

System.out.println("User is exist!");

return result ;

}

// 更新（修改）用户

public Boolean update(int id, User user) {

hibTemplate.update(user);

return true;

}

// 删除指定用户

public Boolean delete(int id) {

User user = find(id);

if(user != null)

hibTemplate.delete(user);

return true;

}

}

（15）编写csexp.service.IUserService接口和csexp.service.UserServiceImp类：

//1.IUserService.java

package csexp.service;

import csexp.model.User;

public interface IUserService {

/\* \* 查找指定用户 \*/

public Boolean findUser(User user);

/\* \* 以ID查找用户 \*/

public User findUser(int id);

/\* \* 以Name查找用户 \*/

public User findUserByName(String name);

/\* \* 查找所有用户 \*/

public List<User> findAllUser();

/\* \* 添加用户 \*/

public Boolean addUser(User user);

/\* \* 更新（修改）用户 \*/

public Boolean updateUser(int id, User user);

/\* \* 删除指定用户 \*/

public Boolean deleteUser(int id);

}

//2.UserServiceImpl.java

package csexp.service;

import csexp.dao.UserDao;

import csexp.dao.IUserDao;

import csexp.model.User;

public class UserServiceImp implements IUserService {

// 使用Hibernate实现持久层。

private IUserDao userDao ;

//注入userDao

public void setUserDao(IUserDao userDao) {

this.userDao = userDao;

}

// 查找指定用户

public Boolean findUser(User user) {

return userDao.find(user);

}

// 以ID查找用户

public User findUser(int id) {

return userDao.find(id);

}

/\* \* 以Name查找用户 \*/

public User findUserByName(String name){

return userDao.findUserByName(name);

}

// 查找所有用户

public List<User> findAllUser() {

return userDao.findAll();

}

// 添加用户

public Boolean addUser(User user) {

return userDao.insert(user);

}

// 更新（修改）用户

public Boolean updateUser(int id, User user) {

return userDao.update(id, user);

}

// 删除指定用户

public Boolean deleteUser(int id) {

return userDao.delete(id);

}

}

（16）.编写csexp.action.UserManagementAction类：

//UserManagementAction.java

package csexp.action;

import csexp.model.User;

import csexp.service.IUserService;

public class UserManagementAction extends ActionSupport implements

ServletRequestAware {

private IUserService service;

private User user;

private String message;

HttpServletRequest request;

private List<User> list = null;

private String oldpassword;// 旧口令

private String password; // 重复口令

private static final long serialVersionUID = 1L;

//注入service

public void setService(IUserService service) {

this.service = service;

}

public void setServletRequest(HttpServletRequest request) {

this.request = request;

}

/\*\*

\* 注册（添加）新用户 规则：（1）使用该Action方法的用户必须先登录,且用户为Admin； (2)输入口令不一致，添加新用户失败

\*

\* @return

\*/

public String register() {

System.out.println("======register======");

if (!password.trim().equals(user.getPassword().trim())) {

setMessage("输入口令不一致，添加新用户失败!");

return ERROR;

}

if (service.addUser(user)) {

setMessage("添加新用户成功!");

return SUCCESS;

} else {

setMessage("添加新用户失败!");

return ERROR;

}

}

/\*

\* 查找指定用户（登录）

\*/

public String logon() {

System.out.println("======UserName=" + user.getName());

if (service.findUser(user)) {

setMessage("用户登录成功!");

HttpSession session = request.getSession();

session.setAttribute("LOGONUSER", user);

return SUCCESS;

} else {

setMessage("用户登录失败!");

return ERROR;

}

}

/\* \* 查找指定用户 \*/

public String find() {

System.out.println("======find======" + user.getId());

user = service.findUser(user.getId());

if (user != null) {

setMessage("查找指定用户成功!");

return SUCCESS;

} else

setMessage("查找指定用户失败!");

return ERROR;

}

/\* \* 列表所有用户，用户登录后才可以使用本方法 \*/

public String list() {

System.out.println("===＝＝list＝＝＝＝＝");

if (!isLogon()) {

setMessage("用户必须登录后才能进行该操作!");

return ERROR;

}

list = service.findAllUser();

request.setAttribute("UserList", list);

return SUCCESS;

}

/\*\*

\* 更新用户资料 规则：（1）使用该Action方法的用户必须先登录； （2）如果用户没有输入新口令，则默认为不修改口令；

\* （3）如果输入的两次口令不一致，则更新失败 （4）Admin用户（id=1001）资料更新操作的限制，用户名不能更改

\*/

// 更新用户资料，需要加上对Admin用户资料更新操作的限制，如用户名不能改。

public String update() {

System.out.println("======Update＝====");

if (!isAdmin()) {

setMessage("必须是管理员用户才能进行该操作!");

return ERROR;

}

if ((user.getId() == 1001) && !user.getName().equals("Admin")) {

setMessage("Admin用户的用户名不能更改，更新资料失败!");

return ERROR;

}

if (user.getPassword().trim().length() == 0) {

// 如果用户没有输入新口令，则默认为不修改口令，使用原口令；

user.setPassword(oldpassword);

} else if (!password.trim().equals(user.getPassword().trim())) {

setMessage("输入口令不一致，更新资料失败!");

return ERROR;

}

if (service.updateUser(user.getId(), user)) {

setMessage("用户资料更新成功!");

return SUCCESS;

} else

setMessage("用户资料更新失败!");

return ERROR;

}

/\*

\* 删除指定用户资料，需要加上对Admin用户删除操作的限制，如该用户不能删除。 Admin用户（id=1001）

\*/

public String delete() {

System.out.println("===delete＝");

if (!isAdmin()) {

setMessage("必须是管理员用户才能进行该操作!");

return ERROR;

}

if (user.getId() == 1001) {

setMessage("Admin用户资料不能删除!失败!");

return ERROR;

}

if (service.deleteUser(user.getId())) {

setMessage("用户资料删除成功!");

return SUCCESS;

} else

setMessage("用户资料删除失败!");

return ERROR;

}

/\*

\* 判断当前登录用户为管理员？

\*/

protected boolean isAdmin() {

//HttpSession session = ServletActionContext.getRequest().getSession();

HttpSession session = request.getSession();

User u = (User) session.getAttribute("LOGONUSER");

System.out.println("User=" + u.getName());

if (u == null) {

return false;

}

if (!u.getName().equals("Admin"))

return false;

return true;

}

/\*

\* 判断当前登录用户为管理员？

\*/

protected boolean isLogon() {

HttpSession session = request.getSession();

User u = (User) session.getAttribute("LOGONUSER");

if (u == null) {

return false;

}

return true;

}

public User getUser() {

return user;

}

//注入user

public void setUser(User user) {

this.user = user;

}

public void setMessage(String message) {

this.message = message;

}

public String getMessage() {

return message;

}

public List<User> getList() {

return list;

}

public void setList(List<User> list) {

this.list = list;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getOldpassword() {

return oldpassword;

}

public void setOldpassword(String oldpassword) {

this.oldpassword = oldpassword;

}

}

（16）修改applicationContext.xml,配置bean:向UserDaoImp注入hibTemplate、UserServiceImp注入userDao、UserManagementAction注入service：

<beans>

.....

<bean id="userDao" class="csexp.dao.UserDaoImp" >

<property name="hibTemplate">

<ref bean="hibTemplate" />

</property>

</bean>

<bean id="service" class="csexp.service.UserServiceImp" >

<property name="userDao">

<ref bean="userDao" />

</property>

</bean>

<bean id="userManagementAction"

class="csexp.action.UserManagementAction">

<property name="service">

<ref bean="service" />

</property>

</bean>

</beans>

同时，向sessionFactory　bean的配置中，添加POJO类User与表test.users的映射配置：

<bean id="sessionFactory"　......>

......

<property name="mappingResources">

<list>

<value>csexp/model/User.hbm.xml</value>

</list>

</property>

</bean>

（17）创建struts.xml或修改“Struts2--用户管理的完整示例”中的struts.xml，注意与Struts2的例子中不同，class属性使用的不是类名，而是使用Spring生成的类对象（在applictionContext.xml）中配置的注入类对象bean:

userManagementAction

而不是csexp.action.UserMangementAction!！！

<struts>

<!-- Character encoding problem -->

<constant name="struts.i18n.encoding" value="GBK" />

<!-- or use

<constant name="struts.i18n.encoding" value="GB18030" />

-->

<package name="controller" namespace="/" extends="struts-default" >

<!-- 使用Spring生成的类对象（在applictionContext.xml）中配置的注入类对象bean:

userManagementAction

-->

<action name="logon" class="userManagementAction" method="logon" >

<result name="success" type="chain">

<param name="namespace">/</param>

<param name="actionName">list</param>

</result>

<result name="error">/Faild.jsp</result>

</action>

<action name="list" class="userManagementAction" method="list" >

<result name="success" >/ListUser.jsp</result>

<result name="error">/Faild.jsp</result>

</action>

<action name="delete" class="userManagementAction" method="delete" >

<result name="success" type="chain">

<param name="namespace">/</param>

<param name="actionName">list</param>

</result>

<result name="error">/Faild.jsp</result>

</action>

<action name="edit" class="userManagementAction" method="find" >

<result name="success" >/EditUser.jsp</result>

<result name="error">/Faild.jsp</result>

</action>

<action name="save" class="userManagementAction" method="update" >

<result name="success" >/Success.jsp</result>

<result name="error">/Faild.jsp</result>

</action>

<action name="register" class="userManagementAction" method="register" >

<result name="success" >/Success.jsp</result>

<result name="error">/Faild.jsp</result>

</action>

<!-- Add your actions here -->

</package>

</struts>