第 7 章 SpringMVC 概述

7.1 SpringMVC 基础

7.1.1 SpringMVC 简介

Spring MVC 的全程是 Spring Web MVC，它是 Spring 框架提供的支持 web 应用的表现层框架，与 struts2 类似。只不过现在 Spring MVC 应用更广。Spring MVC 的支持多种视图层技术，不限定是 JSP 技术，还可以是 Tiles、Velocity 和 iText 等，本书选择 JSP 技术实现视图层。

7.1.2 SpringMVC 的 MVC 架构

在 web 应用程序实现过程中，MVC 模式已经成为了不可获取的设计基础。MVC 是 Model、View 和 Controlller 的缩写，也代表着该模式的三项职责。

1.Model：模型，模型通常由 JavaBean 技术来实现，包括业务实体组件、业务逻辑组件、工具组件等。

2.View：视图，视图通常由 JSP 技术实现，用来收集用户请求，或者向用户传递服务器响应的结果，该层还可能设计客户端验证、国际化和样式等技术。

3.Controller：控制器，控制器负责将用户的请求数据移交对应模型处理，将处理结果再提交对应视图完成反馈。

在 Spring 的 web 设计中，主要的工作围绕 DispatcherServlet 展开。DispatcherServlet 是用来接受请求并将它们分发给相应控制器的中央 servlet。Spring MVC 的主要组成架构包括 DispatcherServlet、处理器映射器、处理器适配器、视图解析器、处理器和视图。

1.DispatcherServlet：前端控制器，由 Spring 框架提供。它相当于 Spring MVC 的中枢神经系统，降低了组件之间的耦合度。所有的请求都经过它来分发，当然在分发请求前还需要 HandlerMapping 映射到具体的 Handler。

2.HandlerMapping：处理器映射器，由 Spring 框架提供。完成请求到 Handler 的映射，查找 Handler 的方式有 xml 配置和注解方式。

3.HandlerAdapter：处理器适配器，由 Spring 框架提供。支持不同类型的 Handler。

4.Handler：处理器，由设计者开发，也常被称为 Controller，即后端控制器。它设计的规则是处理器适配器接口的规则，处理完成后返回 ModelAndView 对象至 DispatcherServlet。这个 ModelAndView 部分顾名思义，是请求过程中返回的 Model（模型）和 View（视图）。

5.ViewReslover：视图解析器，由 Spring 框架提供。主要是根据逻辑视图创建 View 对象。

6.View：视图，由设计者开发。将处理结果渲染进视图并呈现客户端。

7.1.3 SpringMVC 的工作原理

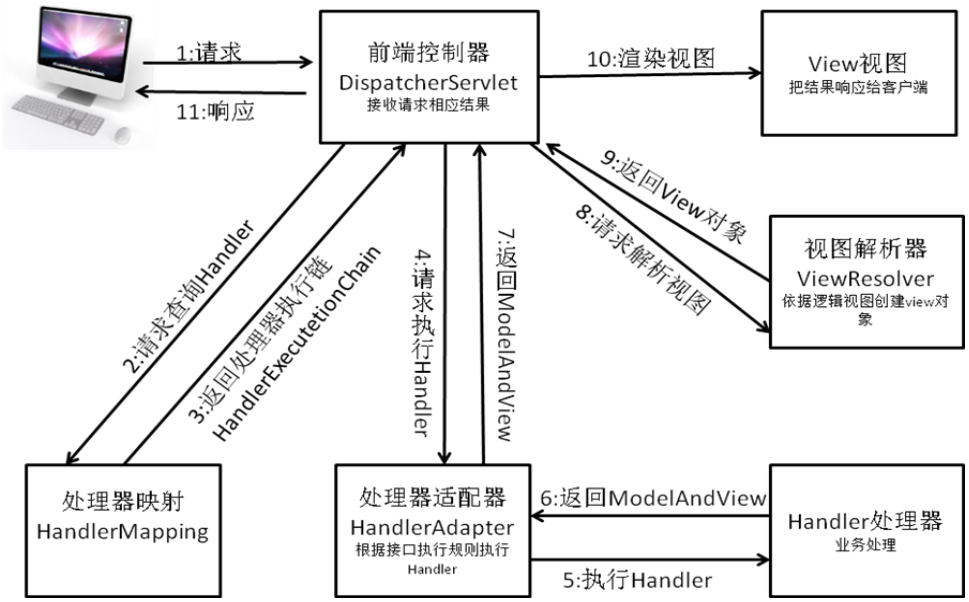

图 7.1 Spring MVC 的工作原理

Spring MVC 的工作原理如图 7.1 所示。

1.客户端发送请求到前端控制器：DispatcherServlet。

2. DispatcherServlet 收到请求后调用 HandlerMapping 查找 Handler。

3. HandlerMapping 根据 url 查找具体 Controller，生成处理器对象和处理器拦截器并返回 DispatcherServlet。

4. DispatcherServlet 依据处理器适配器调用处理器。

5.处理器适配器调用处理器，执行业务处理。

6.Handler 执行后返回 ModelAndView。

7.处理器适配器把接收到的 ModelAndView 返回 DispatcherServlet。

8.DispatcherServlet 寻找视图解析器，将 ModelAndView 传过去，视图解析器根据逻辑视图名解析视图。

9.视图解析器返回视图。

10. DispatcherServlet 对视图进行渲染。

11. DispatcherServlet 响应客户端，将进过渲染的视图呈现。

7.1.4 SpringMVC 的优势

Spring MVC 框架有如下优势。

1.Spring MVC 是优秀的 MVC 框架技术，角色与职责的划分也非常清晰，每个角色都有对应的对象实现；

2.Spring MVC 的视图技术实现灵活。可以采用 JSP 技术，也可以使用 Velocity 和 Tiles 等；

3.Spring MVC 是非侵入，可适配的；

4.Spring MVC 不需要复杂的配置过程，除了基本的配置内容，在编辑处理器类和方法时，可以通过注解修饰；

5.Spring MVC 作为 Spring 的子项目，与 Spring 无缝整合；

6.Spring MVC 具备可定制的绑定和验证。

7.Spring MVC 支持国际化和本地化，可以根据用户区域显示多国语言。

7.2 配置 SpringMVC 的运行环境

7.2.1 下载 SpringMVC 框架

Spring MVC 框架下载可以参考下面的网址，Spring MVC 框架与 Spring 框架的 jar 包是被整合在一起的，http://repo.spring.io/simple/libs-release-local/org/springframework/spring/。

Web 应用 Spring MVC 框架，至少需要用到以下 8 个包：

- Spring 的 4 个核心 jar 包：spring-beans-4.2.4.RELEASE.jar、spring-context-4.2.4.RELEASE.jar、spring-core-4.2.4.RELEASE.jar、spring-expression-4.2.4.RELEASE.jar；
- commons-logging 的 jar 包：commons-logging-1.1.1.jar；
- 与 web 相关的 2 个 jar 包：spring-web-4.2.4.RELEASE.jar、spring-webmvc-4.2.4.RELEASE.jar；
- AOP 相关 jar 包：spring-aop-4.2.4.RELEASE.jar。

7.2.2 创建空的 SpringMVC 配置文件（springmvc.xml）

1.创建 web.xml 配置 DispatcherServlet。

开展 Spring MVC web 项目，需要创建 web.xml，用来部署 DispatcherServlet。

```
<?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">
  <servlet>
    <!-- 配置servlet名称 -->
    <servlet-name>springmvc</servlet-name>
    <!--servlet-class中的value是前端控制器，用于接受所有请求 -->
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <!--  springmvc.xml文件如果在WEB-INF目录下，则不需要指明下面的init-param元素；如果放在其他位置，需要用到下面的init-param元素来配置位置，本书将该文件置于src目录下，故使用“classpath:springmvc.xml”作为位置信息。读者需根据自己的实际需要调整param-value子元素的值 -->
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <!--这里的springmvc.xml文件在放在了src路径下  -->
        <param-value>classpath:springmvc.xml</param-value>
    </init-param>
    <!-- 容器启动即可加载servlet -->
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <!--表示处理用户的所有请求-->
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

2.创建 springmvc.xml 配置 Handler，并将其置于 src 目录下。

```
<?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:aop="http://www.springframework.org/schema/aop"
                xmlns:tx="http://www.springframework.org/schema/tx"
                xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
                        http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-4.2.xsd
                        http://www.springframework.org/schema/aop
                        http://www.springframework.org/schema/aop/spring-aop-4.2.xsd
                        http://www.springframework.org/schema/tx
                        http://www.springframework.org/schema/tx/spring-tx-4.2.xsd">

</beans>
```

7.3 SpringMVC 实现 Hello World 入门

7.3.1 新建 Web 项目

新建 web 项目，架构设计如下图 7.2 所示。

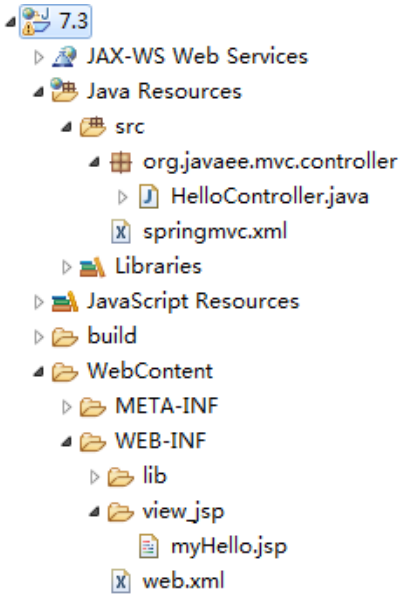

图 7.2 Hello World 入门程序的项目架构

7.3.2 添加 Spring MVC 框架依赖的文件

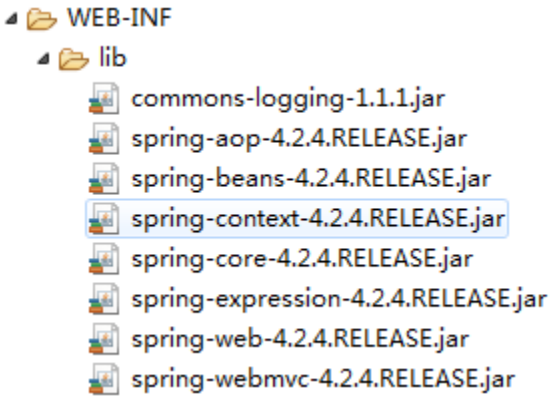

图 7.3 SpringMVC 框架依赖的 jar 包

7.3.3 前端控制器配置

```
<?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">
  <servlet>
    <!-- 配置servlet名称 -->
    <servlet-name>springmvc</servlet-name>
    <!--servlet-class中的value是前端控制器，用于接受所有请求 -->
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <!--  springmvc.xml文件如果在WEB-INF目录下，则不需要指明下面的init-param元素；
                如果放在其他位置，需要用到下面的init-param元素来配置位置。 -->
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <!--springmvc.xml文件在类加载路径下  -->
        <param-value>classpath:springmvc.xml</param-value>
    </init-param>
    <!-- 容器启动即可加载servlet -->
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <!--表示处理所有用户的URL-->
    <url-pattern>/</url-pattern>
  </servlet-mapping>
    </web-app>
```

7.3.4 完善 Spring MVC 配置文件

以 7.2.2 中所描述的 springmvc.xml 为基础，填充下列代码完善 Spring MVC 配置文件。

```
<!-- 扫描处理器类所在的包及其子包 -->
```

```
<context:component-scan base-package="org.javaee.mvc"></context:component-scan>
<!-- 在处理器返回的时候进行解析视图，prefix是前缀，suffix是后缀。 -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
id="internalResourceViewResolver">
<property name="prefix" value="/WEB-INF/view_jsp/"></property>
<property name="suffix" value=".jsp"></property>
    </bean>
```

7.3.5 创建业务控制器 HelloController 类

```
package org.javaee.mvc.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller//表示该类的实例是一个处理器
public class HelloController {
    /*RequestMapping注解将请求和处理方法对应起来，即有/hello的请求来时，
    处理结束，返回逻辑视图名:myHello。*/
    @RequestMapping("/hello")
    public String hello() {
        //这里的逻辑视图名需要根据springmvc.xml配置文件中配置的前缀、后缀找到物理视图。
        return "myHello";
    }
}
```

上面的 Controller 实现的方式是基于注解的，还有一种比较传统的方式实现 Controller，即定义处理器类让其实现 Controller 接口。这种方式需要在配置文件中部署映射关系，并且只能有一个处理方法，很不灵活，所以本节及后面的实例均采用注解的形式实现 Controller。

7.3.6 创建 JSP 页面

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>This is my first Spring-MVC programm</title>
</head>
<body>
<h1>Hello world.</h1>
</body>
</html>
```

7.3.7 部署测试项目

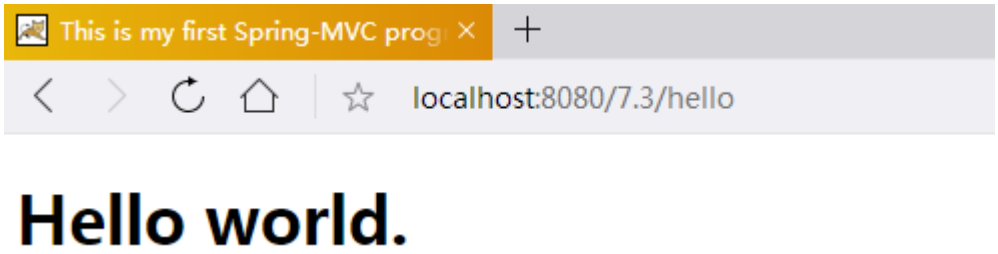

图 7.4 Hello World 入门程序的运行结果

7.4 Spring MVC 的验证框架

7.4.1 数据验证概述

数据验证是由于 WEB 应用的开放性而不得不去考虑的输入数据校验问题。数据验证的主要目的是过滤异常输入数据。

数据验证分为客户端验证和服务器验证。客户端验证主要是过滤正常用户的错误操作，实现技术多为 JavaScript；服务器验证整个应用对非法输入的最后防线，主要是编程实现。

7.4.2 实现 SpringMVC 的数据验证

Spring MVC 框架的数据验证方式有以下两种实现形式：

1. 基于 Validator 接口的 Spring 自带数据验证框架；
2. 基于注解的 JSR 303 标准的验证。

7.4.3 Spring 的 Validation 校验框架

Spring 提供了一个可用于应用程序所有层的验证器接口：Validator。在 Spring MVC 应用中，还可将其配置称为全局的 Validator 实例。

Validator 接口包含两个方法，一个是 supports 方法，它的功能是返回验证器是否可以处理指定的类；另一个是 validate 方法，该方法有两个形参：Object 类型对象 object 和 Errors 类型对象 errors，该方法是验证目标对象 object 并把验证错误消息存入 errors（通过 reject 方法或 rejectValue 方法）。

下面通过一个实例，演示 Spring 的 Vlaidation 数据验证。

1.创建项目架构，web.xml 与上例相同。

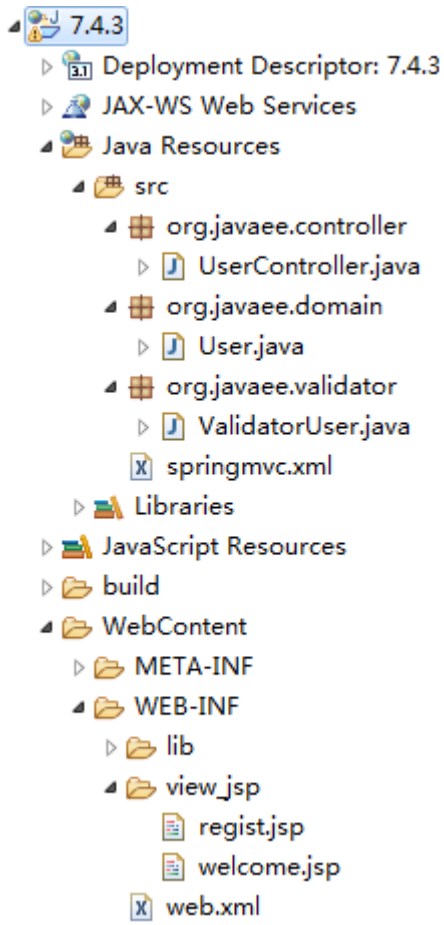

图 7.5 Spring Validator 验证实例程序架构

2.创建注册视图页面：regist.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<!--使用Spring表单标签必须引入标签库-->
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>regist page</title>
</head>
<body>
<h2>欢迎注册本系统</h2>
<form:form modelAttribute="user" action="${pageContext.request.contextPath }/regist"
method="post">
    用户名：<form:input type="text" path="username"/><br>
    密  &nbsp码：<form:input type="password" path="password"/><br>
    年  &nbsp龄：<form:input type="text" path="age"/><br>
    <input type="submit" value="注册"><br>
```

```
<!-- 列出验证错误,path的值为*表示显示所有错误消息；
也可以是某个指定错误消息，如path="username"则表示只显示与username相关的错误信息，
其他错误不显示。 -->
<span style="color:#F00; font-weight:bold"><form:errors path="*"/></span>
</form:form>
</body>
</html>
```

3.创建业务实体模型：User.java

```
package org.javaee.domain;
public class User {
	private String username;
	private String password;
	private Integer age;
	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 Integer getAge() {
		return age;
	}
	public void setAge(Integer age) {
		this.age = age;
	}
}
```

4.创建验证器类：ValidatorUser.java

```
package org.javaee.validator;

import org.javaee.domain.User;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

public class ValidatorUser implements Validator {
	@Override
	public boolean supports(Class<?> arg0) {
```

```
        // TODO Auto-generated method stub
        // 判断是否是需要检验的类，如果参数是 User 类型则返回 true
        return User.class.equals(arg0);
    }

    @Override
    public void validate(Object arg0, Errors arg1) {
        // TODO Auto-generated method stub
        // 将 arg0 转换为待检验类对象
        User user = (User) arg0;
        // 检验其 username 属性
        if (null == user.getUsername() || "".equals(user.getUsername())) {
            // rejectValue 方法向 Errors 对象存入错误信息，第一个参数是 field 名，第四参数是普通字符串型验证结果。
            arg1.rejectValue("username", null, null, "*用户名不能为空！");
        }
        // 检验其 password 属性
        if (null == user.getPassword() || "".equals(user.getPassword())) {
            arg1.rejectValue("password", null, null, "*密码不能为空！");
        }
        // 检验其 age 属性
        if (user.getAge() > 150 || user.getAge() < 0) {
            arg1.rejectValue("password", null, null, "*年龄只能介于 0-150 岁之间！");
        }
    }
}
```

5.创建控制器类：UserController.java

```
package org.javaee.controller;

import org.javaee.domain.User;
import org.javaee.validator.ValidatorUser;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.DataBinder;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class UserController {
    // 注册验证器
```

```
    @InitBinder
    public void initBainder(DataBinder binder) {
        binder.replaceValidators(new ValidatorUser());
    }

    // 这个方法主要是跳转到用户注册页面
    @RequestMapping(value = "/regist", method = RequestMethod.GET)
    public String regist(Model model) {
        // 在视图页面中，可以用EL表达式{$user.username}取出ModelAttribute的username
值
        model.addAttribute("user", new User());
        return "regist";
    }

    // 处理用户注册表单
    @RequestMapping(value = "/regist", method = RequestMethod.POST)
    // @Validated注解修饰的对象为待验证的对象， BindingResult为验证后存放的信息。
    public String regist(@Validated User user, BindingResult br) {
        //// br会被spring的验证器自动填充，如果存在错误，br.hasErrors的返回值会是true。
        if (br.hasErrors()) {
            return "regist";
        }
        return "welcome";
    }
}
```

6.创建配置文件：springmvc.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:aop="http://www.springframework.org/schema/aop"
                xmlns:tx="http://www.springframework.org/schema/tx"
                xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
                        http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-4.2.xsd
                        http://www.springframework.org/schema/aop
                        http://www.springframework.org/schema/aop/spring-aop-4.2.xsd
                        http://www.springframework.org/schema/tx
                        http://www.springframework.org/schema/tx/spring-tx-4.2.xsd">
    <!-- 扫描处理器类所在的包及其子包 -->
    <context:component-scan base-package="org.javaee"></context:component-scan>
    <!-- 在处理器返回的时候进行解析视图，prefix是前缀，suffix是后缀。 -->
```

```
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
    id="internalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/view_jsp/"></property>
    <property name="suffix" value=".jsp"></property>
    </bean>
</beans>
```

7.渲染视图：welcome.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
</head>
<body>
<h2>恭喜你成功注册。</h2>
<ul>
<li>用户名：${user.username}</li>
<li>密码：${user.password }</li>
<li>年龄：${user.age }</li>
</ul>
</body>
</html>
```

8.测试

将项目部署到 Tomcat 服务器之后，启动服务器，在浏览器地址栏输入：http://localhost:8080/7.4.3/regist

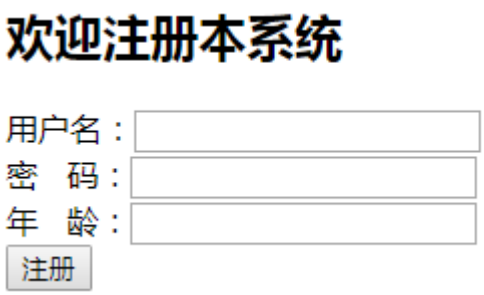

图 7.6 注册页面

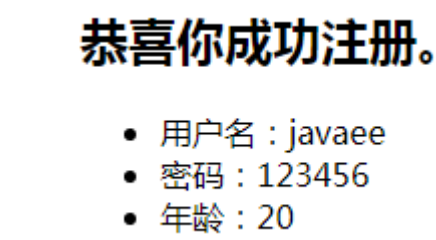

图 7.7 成功注册页面效果

当注册页面输入不合法时，会出现错误提示，例如不输入用户名和密码且年龄大于 150 岁，点击注册后页面显示如下图所示。

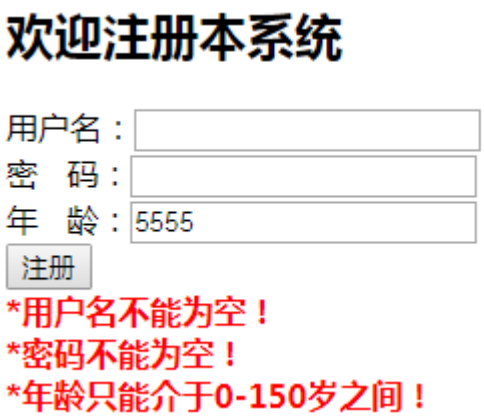

图 7.8 输入异常时页面效果

7.4.4 JSR 303 校验

JSR(Java Specification Requests)是 Java 规范提案。是指向 JCP(Java Community Process)提出新增一个标准化技术规范的正式请求，JSR 已成为 Java 中的重要标准。JSR303 是企业技术中的 Bean 验证标准，它有两个实现，分别为 Hibernate Validator 和 Apache BVal。本书以 Hibernate Validator 为主，读者需注意，这里与 Hibernate 的技术思想无关，只是用于数据验证。

读者可以登录网站：https://sourceforge.net/projects/hibernate/，下载对应版本的 Hibernate Validator，本书选择的版本为 5.3.5。将下载的压缩包解压，把需要的 jar 包复制到 \WEB-INF\lib 目录下面，下面会通过实例程序展示所需的 Jar 包。

JSR303 的使用需要通过一系列的标注类型来对待约束属性进行修饰。下面列举常见的标注类型。

1.空检查

@Null：验证对象是不是为 null；

@NotNull：验证对象是不是不为 null，无法检查字符串""；

@NotBlank：检查约束字符串是不是 null，还有被去掉两端空格后长度是否大于 0；

@NotEmpty 检查约束元素是不是 null 或者 empty。

2.布尔检查

@AssertTrue：验证 boolean 对象是不是 true；

@AssertFalse：验证 boolean 对象是不是 false。

3.长度检查

@Size(min=, max=)：验证对象长度是否在给定的范围内，这些对象可以是数组和集合等；

@Length(min=, max=)：验证字符串长度是不是在给定的范围内。

4.日期检查

@Past：验证 Date 和 Calendar 对象是否在当前时间之前，验证成立的话被注释的元素一定是一个过去的日期；

@Future：验证 Date 和 Calendar 对象是否在当前时间之后，验证成立的话被注释的元素一定是一个将来的日期；

@Pattern：验证 String 对象是否符合正则表达式的规则，被注释的元素符合制定的正则表达式。

5.数值检查

@Min：验证 Number 和 String 对象是否大等于指定的值；

@Max：验证 Number 和 String 对象是否小等于指定的值；

@DecimalMax：被标注的值必须不大于约束中指定的最大值，这个约束的参数是一个通过 BigDecimal 定义的最大值的字符串表示，小数存在精度；

@DecimalMin：被标注的值必须不小于约束中指定的最小值，这个约束的参数是一个通过 BigDecimal 定义的最小值的字符串表示，小数存在精度；

@Digits：验证 Number 和 String 的构成是否合法；

@Digits(integer=,fraction=)：验证字符串是否是符合指定格式的数字，interger 指定整数精度，fraction 指定小数精度；

@Range(min=, max=)：被指定的元素必须在合适的范围内；

@Valid：递归的对关联对象进行校验，如果关联对象是个集合或者数组,那么对其中的元素进行递归校验,如果是一个 map,则对其中的值部分进行校验.(是否进行递归验证)；

@CreditCardNumber：信用卡验证；

@Email：验证是否是邮件地址，如果为 null，不进行验证，算通过验证；

@URL(protocol=,host=, port=,regexp=, flags=)：验证是否是一个有效的 URL。

接下来通过一个实例程序介绍 JSR 303 校验的应用。

1.首先创建项目架构，并添加 jar 包。

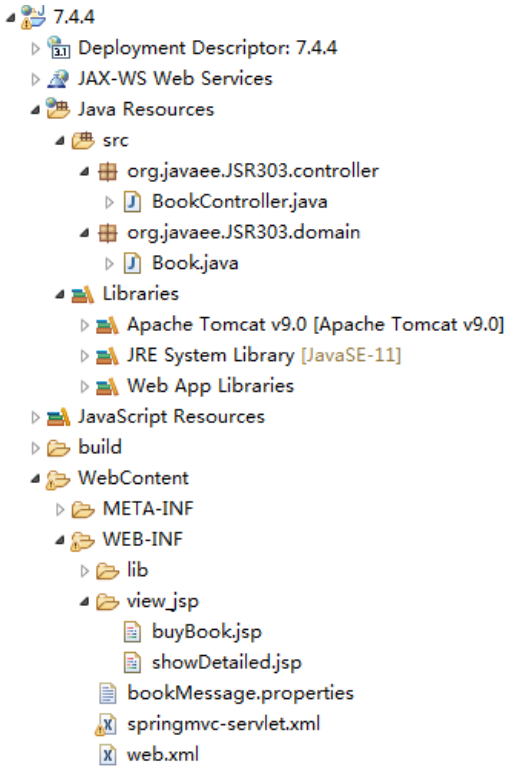

图 7.9 JSR 303 校验实例程序项目架构

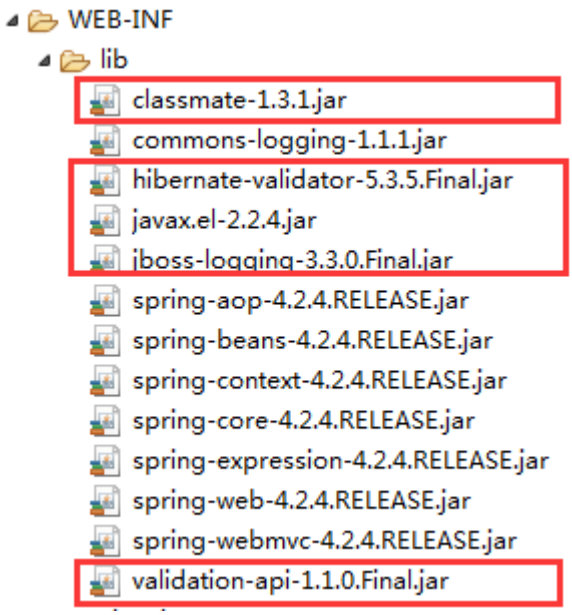

图 7.10 红线标注的是 JSR 303 校验所需 Hibernate Validator 支持要用的 jar 包

2.创建 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">
  <servlet>
    <!-- 配置servlet名称 -->
    <servlet-name>springmvc</servlet-name>
    <!--servlet-class中的value是前端控制器，用于接受所有请求 -->
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <!-- 容器启动即可加载servlet -->
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <!--表示处理所有用户的URL-->
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

3.创建属性文件：bookMessage.properties

```
book.bookName.fail=\u56FE\u4E66\u540D\u4E0D\u80FD\u4E3A\u7A7A
book.writer.fail=\u4F5C\u8005\u59D3\u540D\u5B57\u7B26\u4E32\u957F\u5EA6\u4ECB\u4E8E
2-50\u4E4B\u95F4
```

```
book.price.fail=\u56FE\u4E66\u4EF7\u683C\u4ECB\u4E8E1-1000\u4E4B\u95F4
book.publisher.fail=\u56FE\u4E66\u51FA\u7248\u793E\u4FE1\u606F\u4E0D\u80FD\u4E3A\u7A7A
book.pDate.fail=\u65F6\u95F4\u8BBE\u7F6E\u4E0D\u51C6\u786E
```

Eclipse 将属性文件中输入的中文字符进行了 Unicode 编码。定义了属性文件以后，需要配置框架从该属性文件中读取异常消息，则需要在配置文件中配置 ReloadableResourceBundleMessageSource 实例，在该实例中需指明属性文件路径。具体配置代码见后面的配置文件内容。

4.创建业务实体模型：Book.java

```
package org.javaee.JSR303.domain;
import java.util.Date;
import javax.validation.constraints.Past;
import org.hibernate.validator.constraints.Length;
import org.hibernate.validator.constraints.NotBlank;
import org.hibernate.validator.constraints.Range;
import org.springframework.format.annotation.DateTimeFormat;
public class Book {
    // book.bookName.fail就是属性文件中关于图书名异常消息的key，下面类同。
    @NotBlank(message = "{book.bookName.fail}")
    private String bookName;
    @Length(min = 2, max = 50, message = "{book.writer.fail}")
    private String writer;
    @Range(min = 1, max = 1000, message = "{book.price.fail}")
    private Double price;
    @NotBlank(message = "{book.publisher.fail}")
    private String publisher;
    // 采用日期格式化需配置FormattingConversionServiceFactoryBean
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    @Past(message = "{book.pDate.fail}")
    private Date pDate;

    public String getBookName() {
        return bookName;
    }

    public void setBookName(String bookName) {
        this.bookName = bookName;
    }

    public String getWriter() {
        return writer;
    }
```

```
	public void setWriter(String writer) {
		this.writer = writer;
	}

	public Double getPrice() {
		return price;
	}

	public void setPrice(Double price) {
		this.price = price;
	}

	public String getPublisher() {
		return publisher;
	}

	public void setPublisher(String publisher) {
		this.publisher = publisher;
	}

	public Date getpDate() {
		return pDate;
	}

	public void setpDate(Date pDate) {
		this.pDate = pDate;
	}

}
```

3.创建购买图书页面：buyBook.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"
	pageEncoding="utf-8"%>
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
</head>
<body>
<h2>请输入您要购买的图书信息</h2>
<table>
<form:form modelAttribute="book"
```

```
action="${pageContext.request.contextPath }/booksys/show">
	<tr><td>书名：</td><td><form:input type="text" path="bookName"/></td></tr>
	<tr><td>作者：</td><td><form:input type="text" path="writer"/></td></tr>
	<tr><td>价格：</td><td><form:input type="text" path="price"/></td></tr>
	<tr><td>出版社：</td><td><form:input type="text" path="publisher"/></td></tr>
	<tr><td>出版日期：</td><td><form:input type="text" path="pDate"/></td></tr>
	<tr><td><input type="submit" value="提交"></td></tr>
<span style="color:#F00; font-weight:bold"><form:errors path="*"/></span>
</form:form>
</table>
</body>
</html>
```

4.创建控制器类：BookController.java

```
package org.javaee.JSR303.controller;
import javax.validation.Valid;
import org.javaee.JSR303.domain.Book;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
@RequestMapping("/booksys")
public class BookController {
	@RequestMapping("/buy")
	public String buyBook(Model model) {
		model.addAttribute(new Book());
		return "buyBook";
	}

	@RequestMapping("/show")
	public String showDetailed(@Valid @ModelAttribute Book book, BindingResult br, Model
model) {
		if (br.hasErrors()) {
			return "buyBook";
		}
		return "showDetailed";
	}
}
```

5.创建显示图书明细页面：showDetailed.jsp

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

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
</head>
<body>
<h2>您购买的图书明细如下</h2>
<ul>
<li>图书名：${book.bookName}</li>
<li>图书作者：${book.writer }</li>
<li>图书价格：${book.price }</li>
<li>出版社：${book.publisher}</li>
<li>出版日期：${book.pDate}</li>
</ul>
</body>
</html>
```

6.创建配置文件：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-4.2.xsd
                                http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-4.2.xsd
                                http://www.springframework.org/schema/mvc
                                http://www.springframework.org/schema/mvc/spring-mvc-4.2.xsd">
      <!-- 扫描处理器类所在的包及其子包 -->
      <context:component-scan base-package="org.javaee"></context:component-scan>
      <!-- 在处理器返回的时候进行解析视图，prefix是前缀，suffix是后缀。 -->
      <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
      id="internalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/view_jsp/"></property>
      <property name="suffix" value=".jsp"></property>
      </bean>
      <!-- 配置用于显示异常消息的属性文件 -->
      <bean
class="org.springframework.context.support.ReloadableResourceBundleMessageSource"
      id="message">
      <property name="basename" value="/WEB-INF/bookMessage"></property>
```

```
<!-- 设值属性文件编码格式 -->
<property name="fileEncodings" value="utf-8"></property>
</bean>
<!-- 注册校验器，关联属性文件与Hibernate Validator -->
<bean class=" org.springframework.validation.beanvalidation.LocalValidatorFactoryBean"
id="validator">
<!-- 依赖注入hibernate校验器 -->
<property name="providerClass"
value="org.hibernate.validator.HibernateValidator"></property>
<!-- 将上面定义的属性文件bean依赖注入validator -->
<property name="validationMessageSource" ref="message"></property>
</bean>
 <!-- 注册格式化的转换器 -->
<bean id="conversionService"
class="org.springframework.format.support.FormattingConversionServiceFactoryBean"></bean>
<!-- 开启Spring的Valid功能 -->
<mvc:annotation-driven conversion-service="conversionService" validator="validator"/>
</beans>
```

7.部署测试。

部署并启动好服务器之后，在浏览器地址栏中输入：http://localhost:8080/7.4.4/booksys/buy，得到如下结果。

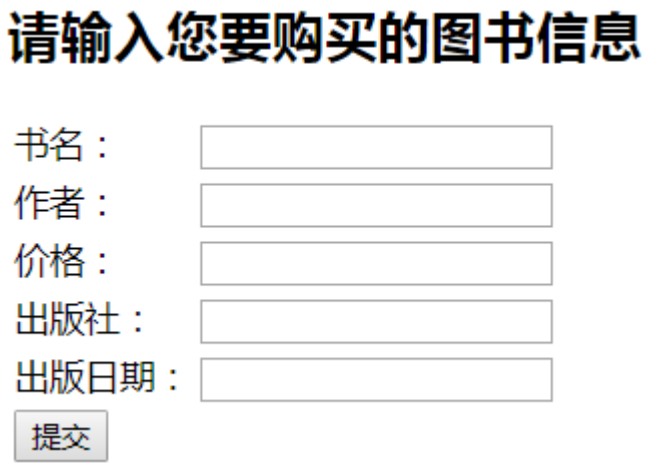

图 7.11 首次登录购买图书页面效果

成功进入购买图书页面后，如果录入异常信息（书名为空、作者长度为 1、价格为 0、出版社信息为空、日期超过当前时间），会得到如下结果：

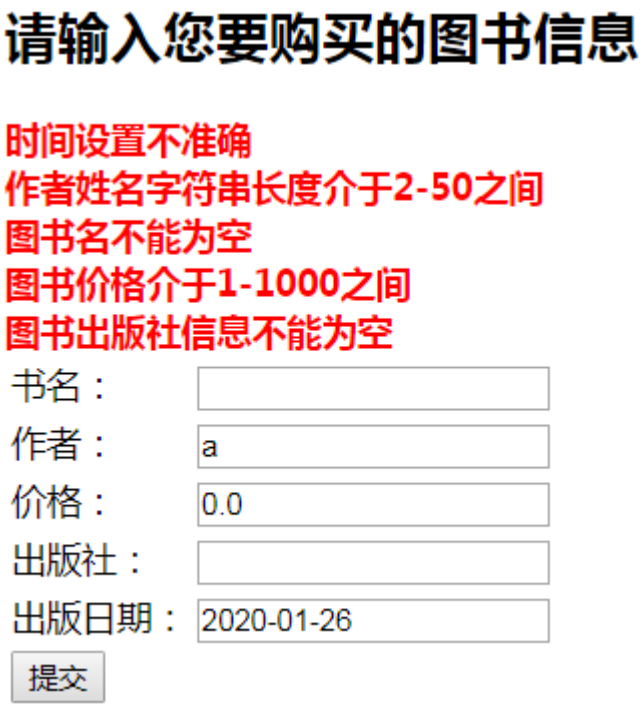

图 7.12 购书页面输入异常信息后进入的视图效果

7.5 小结

Spring MVC 应用主要的工作是设计 Controller。处理器的核心流程是接收请求参数，调用业务组件处理请求，返回逻辑结果名。

处理第一步：接受请求参数。接受请求参数的方式也有很多种，现给出常见的三种设置方式，如下列表内容所示。

1.通过实体对象接收请求参数，例如 public String processer(User user, Model model);//User 为实体类；

2.通过方法形参接收请求参数，例如 public String processer(String username, String password, String age, Model model);

3. 通过 @ModelAttribute 接收请求参数，例如 public String processer (@ModelAttribute("user") User user);

处理第二步：调用业务逻辑组件。本章的控制器均没有将业务内容抽取出来，这种方式是不合理的，所以实际项目中，需要将业务内容从控制器分离出来，定义成业务组件。业务组件类需要用@Service 注解修饰，同时还需要在配置文件中将业务组件类所在的包置于扫描元素，例如<context:component-scan base-package="业务组件类所在包">。如果定义了业务组件，可以利用@Autowired 来将业务组件注入控制器。例如下面的程序过程。

1.定义业务组件类

```
import org.springframework.stereotype.Service;
@Service
public class UserService{}
```

2.设置配置文件

```
<context:component-scan base-package="org.javaee.service"/>
```

3.定义控制器

```
@Controller
```

```
public class UserController{
    @Autowired
    public UserService userService;
    //.....
}
```

处理第三步：返回逻辑结果。在处理第二步结束后，处理器方法需要 return 一个逻辑结果。该结果可以转发到视图、重定向和转发到一个方法、某个具体资源。举例如下。

1.转发到视图。例如在处理器中使用 return "welcome";将结果转发到 spring 配置文件中设定的 InternalResourceViewResolver Bean 所定义的“前缀”目录中的 welcome.jsp。

2.重定向。例如 return "redirect:/abc";表示重定向到 abc 请求方法。

3.请求转发。例如 return "forward:/abc";表示转发到 abc 请求方法。

本章介绍了 Spring MVC 的初步应用技术，Spring MVC 的拦截器、国际化等将在下一章的 Spring MVC 高级应用中介绍。