Spring Boot的Web开发

学习目的与要求

本章首先介绍Spring Boot的Web开发支持，然后介绍Thymeleaf视图模板引擎技术，最后介绍Spring Boot的Web开发技术(JSON数据交互、文件上传与下载、异常统一处理以及对JSP的支持)。通过本章的学习，掌握Spring Boot的Web开发技术。

本章主要内容

- Thymeleaf模板引擎。
- Spring Boot处理JSON数据。
- Spring Boot的文件上传与下载。
- Spring Boot的异常处理。

Web开发是一种基于B/S架构(即浏览器/服务器)的应用软件开发技术，分为前端(用户接口)和后端(业务逻辑和数据)，前端的可视化及用户交互由浏览器实现，即以浏览器作为客户端，实现客户与服务器远程的数据交互。Spring Boot的Web开发内容主要包括内嵌Servlet容器和Spring MVC。

5.1 Spring Boot的Web开发支持

Spring Boot提供了spring-boot-starter-web依赖模块，该依赖模块包含Spring Boot预定义的Web开发常用依赖包，为Web开发者提供了内嵌的Servlet容器(Tomcat)以及Spring MVC的依赖。如果开发者希望开发Spring Boot的Web应用程序，可以在Spring

Boot 项目的 pom.xml 文件中添加如下依赖配置：


视频讲解

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Spring Boot 将自动关联 Web 开发的相关依赖，如 tomcat、spring-webmvc 等，进而对 Web 开发提供支持，并对相关技术的配置实现自动配置。

另外，开发者也可以使用 Spring Tool Suite 集成开发工具快速创建 Spring Starter Project，在 New Spring Starter Project Dependencies 对话框中添加 Spring Boot 的 Web 依赖，如图 5.1 所示。

图 5.1 添加 Spring Boot 的 Web 依赖

5.2 Thymeleaf 模板引擎


视频讲解

在 Spring Boot 的 Web 应用中，建议开发者使用 HTML 完成动态页面。Spring Boot 提供了许多模板引擎，主要包括 FreeMarker、Groovy、Thymeleaf、Velocity 和 Mustache。因为 Thymeleaf 提供了完美的 Spring MVC 支持，所以在 Spring Boot 的 Web 应用中推荐使用 Thymeleaf 作为模板引擎。

Thymeleaf 是一个 Java 类库，是一个 XML/XHTML/HTML5 的模板引擎，能够处理 HTML、XML、JavaScript 以及 CSS，可以作为 MVC Web 应用的 View 层显示数据。

5.2.1 Spring Boot 的 Thymeleaf 支持

在 Spring Boot 1.X 版本中，spring-boot-starter-thymeleaf 依赖包含了 spring-boot-starter-web 模块。但是，在 Spring 5 中，WebFlux 的出现对于 Web 应用的解决方案将不再唯一。所以，spring-boot-starter-thymeleaf 依赖不再包含 spring-boot-starter-web 模块，需要开发人员自己选择 spring-boot-starter-web 模块依赖。下面通过一个实例，讲解如何创建基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_1。

【例 5-1】 创建基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_1。

具体实现步骤如下。

1. 创建 Spring Starter Project

选择菜单 File|New|Spring Starter Project，打开 New Spring Starter Project 对话框，在该对话框中选择和输入相关信息，如图 5.2 所示。

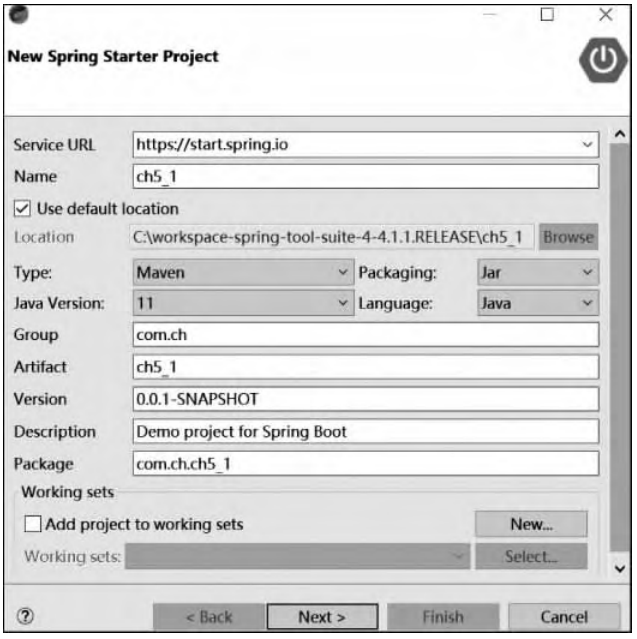

图 5.2 创建基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_1

2. 选择依赖

单击图 5.2 中的 Next 按钮，打开 New Spring Starter Project Dependencies 对话框，选择 Spring Web Starter 和 Thymeleaf 依赖，如图 5.3 所示。

3. 打开项目目录

单击图 5.3 中的 Finish 按钮，创建如图 5.4 所示的基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_1。

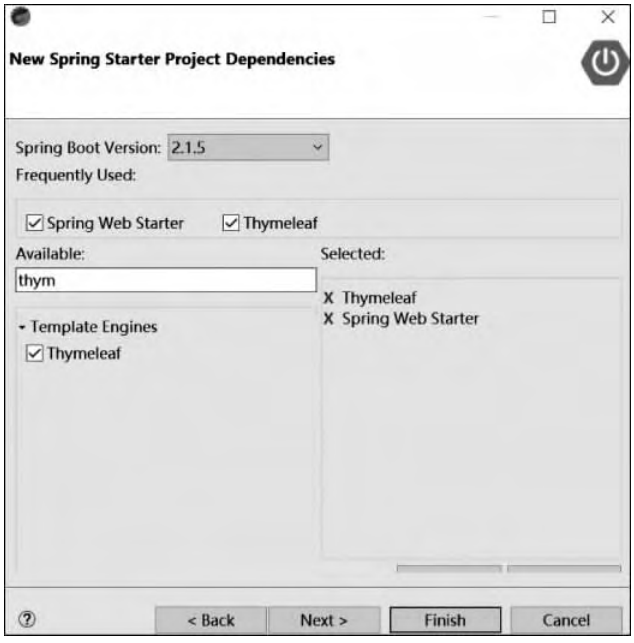

图 5.3 选择 Spring Web Starter 和 Thymeleaf 依赖

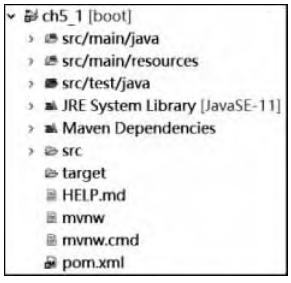

图 5.4 基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_1

Tymeleaf 模板默认将 JS 脚本、CSS 样式、图片等静态文件放置在 src/main/resources/static 目录下，将视图页面放在 src/main/resources/templates 目录下。

4. 创建控制器类

创建一个名为 com.ch.ch5_1.controller 的包。并在该包中创建控制器类 TestThymeleafController，代码如下：

```
package com.ch.ch5_1.controller;
import org.springframework.stereotype.Controller;
@Controller
public class TestThymeleafController {
    @RequestMapping("/")
    public String test(){
        //根据 Tymeleaf 模板，默认将返回 src/main/resources/templates/index.html
        return "index";
    }
}
```

5. 新建 index.html 页面

在 src/main/resources/templates 目录下新建 index.html 页面，代码如下：

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset = "UTF - 8">
<title> Insert title here</title>
</head>
<body>
测试 Spring Boot 的 Thymeleaf 支持
</body>
</html>
```

6. 运行测试

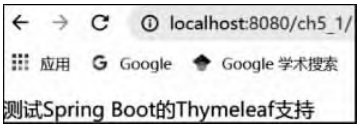

图 5.5 例 5-1 运行结果

首先，运行 Ch51Application 主类。然后，访问 http://localhost:8080/ch5_1/(因为配置文件中配置了 Web 应用的上下文路径为 ch5_1)。运行效果如图 5.5 所示。

5.2.2 Thymeleaf 基础语法

1. 引入 Thymeleaf

首先，将 View 层页面文件的 html 标签修改如下：

```
<html xmlns:th = "http://www.thymeleaf.org">
```

然后，在 View 层页面文件的其他标签里，使用 th:* 动态处理页面，示例代码如下：

```
<img th:src = "'images/' + ${aBook.picture}"/>
```

其中，${aBook.picture}获得数据对象 aBook 的 picture 属性。

2. 输出内容

使用 th:text 和 th:utext(对 HTML 标签解析)将文本内容输出到所在标签的 body 中。假如在国际化资源文件 messages_en_US.properties 中有消息文本“test.myText = Test International Message ”，那么在页面中可以使用如下两种方式获得消息文本：

```
<p th:text = "#{test.myText}"></p>
<!-- 不对 HTML 标签解析，即输出<strong> Test International Message </strong> -->
<p th:utext = "#{test.myText}"></p>
<!-- 对 HTML 标签解析，即输出加粗的"Test International Message" -->
```

3. 基本表达式

1) 变量表达式：${…}

变量表达式用于访问容器上下文环境中的变量，示例代码如下：

```
<span th:text = "${information}">
```

2) 选择变量表达式：*{…}

选择变量表达式计算的是选定的对象(th:object 属性绑定的对象)，示例代码如下：

```
<div th:object = "${session.user}">
    name: <span th: text = "*{firstName}"></span><br>
    <!-- firstName 为 user 对象的属性 -->
    surname: <span th: text = "*{lastName}"></span><br>
    nationality: <span th: text = "*{nationality}"></span><br>
</div>
```

3) 信息表达式：#{…}

信息表达式一般用于显示页面静态文本，将可能需要根据需求而整体变动的静态文本放在 properties 文件中以便维护(如国际化)，通常与 th:text 属性一起使用，示例代码如下：

```
<p th:text = "#{test.myText}"></p>
```

4. 引入 URL

Thymeleaf 模板通过@{…}表达式引入 URL，示例代码如下：

```
<!-- 默认访问 src/main/resources/static 下的 css 文件夹 -->
<link rel = "stylesheet" th:href = "@{css/bootstrap.min.css}" />
<!-- 访问相对路径 -->
<a th:href = "@{/}">去看看</a>
<!-- 访问绝对路径 -->
<a th:href = "@{http://www.tup.tsinghua.edu.cn/index.html(param1 = '传参')}">去清华大学出版社</a>
<!-- 默认访问 src/main/resources/static 下的 images 文件夹 -->
<img th:src = "'images/' + ${aBook.picture}"/>
```

5. 访问 WebContext 对象中的属性

Thymeleaf 模板通过一些专门的表达式从模板的 WebContext 获取请求参数、请求、会话和应用程序中的属性，具体如下：

${xxx}将返回存储在 Thymeleaf 模板上下文中的变量 xxx 或请求 request 作用域中的属性 xxx。

${param.xxx}将返回一个名为 xxx 的请求参数(可能是多个值)。

${session.xxx}将返回一个名为 xxx 的 HttpSession 作用域中的属性。

${application.xxx}将返回一个名为 xxx 的全局 ServletContext 上下文作用中的属性。

与 EL 表达式一样，使用${xxx}获得变量值，使用${对象变量名.属性名}获取 JavaBean 属性值。但需要注意的是，${}表达式只能在 th 标签内部有效。

6. 运算符

在 Thymeleaf 模板的表达式中可以使用+、-、*、/、%等各种算术运算符，也可以使用>、<、<=、>=、==、!=等各种逻辑运算符。示例代码如下：

```
<tr th:class = "(${row} == 'even')? 'even' : 'odd'">…</tr>
```

7. 条件判断

1) if 和 unless

只有在 th:if 条件成立时才显示标签内容；th:unless 与 th:if 相反，只有在条件不成立

时才显示标签内容。示例代码如下：

```
<a href = "success.html" th:if = " ${user != nul}">成功</a>
<a href = "success.html" th:unless = " ${user = nul}">成功</a>
```

2）switch 语句

Thymeleaf 模板也支持多路选择 switch 语句结构，默认属性 default 可用“*”表示。示例代码如下：

```
<div th:switch = " ${user.role}">
    <p th:case = "'admin'"> User is an administrator </p>
    <p th:case = "'teacher'"> User is a teacher </p>
    <p th:case = " * "> User is a student </p>
</div>
```

8. 循环

1）基本循环

Thymeleaf 模板使用 th:each="obj,iterStat: ${objList}"标签进行迭代循环，迭代对象可以是 java.util.List、java.util.Map 或数组等。示例代码如下：

```
<!-- 循环取出集合数据 -->
<div class = "col - md - 4 col - sm - 6" th:each = "book: ${books}">
    <a href = "">
        <img th:src = "'images/' + ${book.picture}" alt = "图书封面" style = "height: 180px;
width: 40 % ;"/>
    </a>
    <div class = "caption">
        <h4 th:text = " ${book.bname}"></h4>
        <p th:text = " ${book.author}"></p>
        <p th:text = " ${book.isbn}"></p>
        <p th:text = " ${book.price}"></p>
        <p th:text = " ${book.publishing}"></p>
    </div>
</div>
```

2）循环状态的使用

在 th:each 标签中可以使用循环状态变量，该变量有如下属性。

index：当前迭代对象的 index(从 0 开始计数)。

count：当前迭代对象的 index(从 1 开始计数)。

size：迭代对象的大小。

current：当前迭代变量。

even/odd：布尔值，当前循环是否是偶数/奇数(从 0 开始计数)。

first：布尔值，当前循环是否是第一个。

last：布尔值，当前循环是否是最后一个。

使用循环状态变量的示例代码如下：

```
<!-- 循环取出集合数据 -->
<div class = "col - md - 4 col - sm - 6" th:each = "book,bookStat: ${books}">
    <a href = "">
        <img th:src = "'images/' + ${book.picture}" alt = "图书封面" style = "height: 180px;
width: 40%;"/>
    </a>
    <div class = "caption">
        <!-- 循环状态 bookStat -->
        <h3 th:text = "${bookStat.count}"></h3>
        <h4 th:text = "${book.bname}"></h4>
        <p th:text = "${book.author}"></p>
        <p th:text = "${book.isbn}"></p>
        <p th:text = "${book.price}"></p>
        <p th:text = "${book.publishing}"></p>
    </div>
</div>
```

9. 内置对象

在实际 Web 项目开发中，经常传递列表、日期等数据。所以，Thymeleaf 模板提供了很多内置对象，可以通过 # 直接访问。这些内置对象一般都以 s 结尾，如 dates、lists、numbers、strings 等。Thymeleaf 模板通过 ${#…} 表达式访问内置对象。常见的内置对象如下。

#dates：日期格式化的内置对象，操作的方法是 java.util.Date 类的方法。

#calendars：类似于 #dates，但操作的方法是 java.util.Calendar 类的方法。

#numbers：数字格式化的内置对象。

#strings：字符串格式化的内置对象，操作的方法参照 java.lang.String。

#objects：参照 java.lang.Object。

#bools：判断 boolean 类型的内置对象。

#arrays：数组操作的内置对象。

#lists：列表操作的内置对象，参照 java.util.List。

#sets：Set 操作的内置对象，参照 java.util.Set。

#maps：Map 操作的内置对象，参照 java.util.Map。

#aggregates：创建数组或集合的聚合的内置对象。

#messages：在变量表达式内部获取外部消息的内置对象。

假如有如下控制器方法：

```
@RequestMapping("/testObject")
public String testObject(Model model) {
    //系统时间 new Date()
    model.addAttribute("nowTime", new Date());
    //系统日历对象
    model.addAttribute("nowCalendar", Calendar.getInstance());
    //创建 BigDecimal 对象
    BigDecimal money = new BigDecimal(2019.613);
    model.addAttribute("myMoney", money);
```

```
        //字符串
        String tsts = "Test strings";
        model.addAttribute("str", tsts);
        //boolean 类型
        boolean b = false;
        model.addAttribute("bool", b);
        //数组(这里不能使用 int 定义数组)
        Integer aint[] = {1,2,3,4,5};
        model.addAttribute("mya", aint);
        //List 列表 1
        List<String> nameList1 = new ArrayList<String>();
        nameList1.add("陈恒 1");
        nameList1.add("陈恒 3");
        nameList1.add("陈恒 2");
        model.addAttribute("myList1", nameList1);
        //Set 集合
        Set<String> st = new HashSet<String>();
        st.add("set1");
        st.add("set2");
        model.addAttribute("mySet", st);
        //Map 集合
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("key1", "value1");
        map.put("key2", "value2");
        model.addAttribute("myMap", map);
        //List 列表 2
        List<String> nameList2 = new ArrayList<String>();
        nameList2.add("陈恒 6");
        nameList2.add("陈恒 5");
        nameList2.add("陈恒 4");
        model.addAttribute("myList2", nameList2);
        return "showObject";
    }
```

那么,可以在 src/main/resources/templates/showObject.html 视图页面文件中使用内置对象操作数据。showObject.html 的代码如下:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    格式化控制器传递过来的系统时间 nowTime
    <span th:text="${#dates.format(nowTime, 'yyyy/MM/dd')}"></span>
    <br>
    创建一个日期对象
    <span th:text="${#dates.create(2019,6,13)}"></span>
    <br>
    格式化控制器传递过来的系统日历 nowCalendar:
```

```
    <span th:text = " ${ #calendars.format(nowCalendar, 'yyyy - MM - dd')}"></span>
    <br>
    格式化控制器传递过来的 BigDecimal 对象 myMoney:
    <span th:text = " ${ #numbers.formatInteger(myMoney,3)}"></span>
    <br>
    计算控制器传递过来的字符串 str 的长度:
    <span th:text = " ${ #strings.length(str)}"></span>
    <br>
    返回对象,当控制器传递过来的 BigDecimal 对象 myMoney 为空时,返回默认值 9999:
    <span th:text = " ${ #objects.nullSafe(myMoney, 9999)}"></span>
    <br>
    判断 boolean 数据是否是 false:
    <span th:text = " ${ #bools.isFalse(bool)}"></span>
    <br>
    判断数组 mya 中是否包含元素 5:
    <span th:text = " ${ #arrays.contains(mya, 5)}"></span>
    <br>
    排序列表 myList1 的数据:
    <span th:text = " ${ #lists.sort(myList1)}"></span>
    <br>
    判断集合 mySet 中是否包含元素 set2:
    <span th:text = " ${ #sets.contains(mySet, 'set2')}"></span>
    <br>
    判断 myMap 中是否包含 key1 关键字:
    <span th:text = " ${ #maps.containsKey(myMap, 'key1')}"></span>
    <br>
    将数组 mya 中的元素求和:
    <span th:text = " ${ #aggregates.sum(mya)}"></span>
    <br>
    将数组 mya 中的元素求平均:
    <span th:text = " ${ #aggregates.avg(mya)}"></span>
    <br>
    如果未找到消息,则返回默认消息(如"??msgKey_zh_CN??"):
    <span th:text = " ${ #messages.msg('msgKey')}"></span>
</body>
</html>
```

5.2.3 Thymeleaf 的常用属性

通过 5.2.2 节的学习,我们发现 Thymeleaf 语法的使用都是通过在 html 页面的标签中添加 th:xxx 关键字来实现模板套用,且其属性与 html 页面标签基本类似。常用属性有以下几种。

1. th:action

th:action 定义后台控制器路径,类似于< form >标签的 action 属性。示例代码如下:

```
<form th:action = "@{/login}">…</form>
```

2. th:each

th:each 用于集合对象遍历,功能类似于 JSTL 标签< c:forEach >。示例代码如下:

```
<div class = "col - md - 4 col - sm - 6" th:each = "gtype: ${gtypes}">
    <div class = "caption">
        <p th:text = " ${gtype.id}"></p>
        <p th:text = " ${gtype.typename}"></p>
    </div>
</div>
```

3. th:field

th:field 常用于表单参数绑定,通常与 th:object 一起使用。示例代码如下:

```
<form th:action = "@{/login}" th:object = " ${user}">
    <input type = "text" value = "" th:field = " *{username}"></input>
    <input type = "text" value = "" th:field = " *{role}"></input>
</form>
```

4. th:href

th:href 用于定义超链接,类似于< a >标签的 href 属性。value 形式为@{/logout},示例代码如下:

```
<a th:href = "@{/gogo}"></a>
```

5. th:id

th:id 用于 id 的声明,类似于 html 标签中的 id 属性。示例代码如下:

```
<div th:id = "stu + ( ${rowStat.index} + 1)"></div>
```

6. th:if

th:if 用于条件判断,如果为否则标签不显示。示例代码如下:

```
<div th:if = " ${rowStat.index} == 0">… do something …</div>
```

7. th:fragment

th:fragment 声明定义该属性的 div 为模板片段,常用于头文件、页尾文件的引入,常与 th:include、th:replace 一起使用。

假如在 ch5_1 的 src/main/resources/templates 目录下声明模板片段文件 footer.html,具体代码如下:

```
<! DOCTYPE html >
< html xmlns:th = "http://www.thymeleaf.org">
< head >
< meta charset = "UTF - 8">
< title > Insert title here </title >
```

```
</head>
<body>
    <!-- 声明片段 content -->
    <div th:fragment="content">
        主体内容
    </div>
    <!-- 声明片段 copy -->
    <div th:fragment="copy">
        ©清华大学出版社
    </div>
</body>
</html>
```

那么，可以在 ch5_1 的 src/main/resources/templates/index.html 文件中引入模板片段，具体代码如下：

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    测试 Spring Boot 的 Thymeleaf 支持<br>
    引入主体内容模板片段:
    <div th:include="footer::content"></div>
    引入版权所有模板片段:
    <div th:replace="footer::copy"></div>
</body>
</html>
```

8. th:object

th:object 用于表单数据对象绑定，将表单绑定到后台 controller 的一个 JavaBean 参数，常与 th:field 一起使用，进行表单数据绑定。下面通过实例讲解表单提交及数据绑定的实现过程。

【例 5-2】 表单提交及数据绑定的实现过程。

具体实现步骤如下。

1）创建实体类

在 Web 应用 ch5_1 的 src/main/java 目录下，创建 com.ch.ch5_1.model 包，并在该包中创建实体类 LoginBean，代码如下：

```
package com.ch.ch5_1.model;
public class LoginBean {
    String uname;
    String urole;
    //省略 set 和 get 方法
}
```

2）创建控制器类

在 Web 应用 ch5_1 的 com.ch.ch5_1.controller 包中，创建控制器类 LoginController，具体代码如下：

```
package com.ch.ch5_1.controller;
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 com.ch.ch5_1.model.LoginBean;
@Controller
public class LoginController {
    @RequestMapping("/toLogin")
    public String toLogin(Model model) {
        /* loginBean 与 login.html 页面中的 th:object = "${loginBean}"相同，类似于 Spring
        MVC 的表单绑定。 */
        model.addAttribute("loginBean", new LoginBean());
        return "login";
    }
    @RequestMapping("/login")
     public String greetingSubmit(@ModelAttribute LoginBean loginBean) {
        /* @ModelAttribute LoginBean loginBean 接收 login.html 页面中的表单数据，并将
        loginBean 对象保存到 model 中返回给 result .html 页面显示。 */
        System.out.println("测试提交的数据:" + loginBean.getUname());
        return "result";
    }
}
```

3）创建页面表示层

在 Web 应用 ch5_1 的 src/main/resources/templates 目录下，创建页面 login.html 和 result.html。

页面 login.html 的代码如下：

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h1>Form</h1>
    <form action="#" th:action="@{/login}" th:object="${loginBean}" method="post">
        <!-- th:field="*{uname}"的 uname 与实体类的属性相同，即绑定 loginBean 对象  -->
        <p>Uname: <input type="text" th:field="*{uname}" th:placeholder="请输入用户
名" /></p>
        <p>Urole: <input type="text" th:field="*{urole}" th:placeholder="请输入角色" />
</p>
        <p><input type="submit" value="Submit" /> <input type="reset" value="Reset" />
</p>
    </form>
```

```
</body>
</html>
```

页面 result.html 的代码如下：

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h1>Result</h1>
    <p th:text="'Uname: ' + ${loginBean.uname}" />
    <p th:text="'Urole: ' + ${loginBean.urole}" />
    <a href="toLogin">继续提交</a>
</body>
</html>
```

4）运行

首先，运行 Ch51Application 主类。然后，访问 http://localhost:8080/ch5_1/toLogin。运行结果如图 5.6 所示。

在图 5.6 的文本框中输入信息后，单击 Sumbit 按钮，打开如图 5.7 所示的页面。

图 5.6 页面 login.html 的运行结果

图 5.7 页面 result.html 的运行结果

9. th:src

th:src 用于外部资源引入，类似于< script >标签的 src 属性。示例代码如下：

```
<img th:src="'images/' + ${aBook.picture}" />
```

10. th:text

th:text 用于文本显示，将文本内容显示到所在标签的 body 中。示例代码如下：

```
<td th:text="${username}"></td>
```

11. th:value

th:value 用于标签赋值，类似于标签的 value 属性。示例代码如下：

```
<option th:value = "Adult"> Adult </option>
<input type = "hidden" th:value = " ${msg}" />
```

12. th:style

th:style 用于修改标签 style。示例代码如下：

```
<span th:style = "'display:' + @{( ${sitrue} ? 'none' : 'inline - block')}"></span>
```

13. th:onclick

th:onclick 用于修改单击事件，示例代码如下：

```
<button th:onclick = "'getCollect()'"></button>
```

5.2.4 Spring Boot 与 Thymeleaf 实现页面信息国际化

在 Spring Boot 的 Web 应用中实现页面信息国际化非常简单，下面通过实例讲解国际化的实现过程。

【例 5-3】 国际化的实现过程。

具体实现步骤如下。

1. 编写国际化资源属性文件

1）编写管理员模块的国际化信息

在 ch5_1 的 src/main/resources 目录下创建 i18n/admin 文件夹，并在该文件夹下创建 adminMessages. properties、adminMessages_en_US. properties 和 adminMessages_zh_CN. properties 资源属性文件。adminMessages. properties 表示默认加载的信息；adminMessages_en_US. properties 表示英文信息(en 代表语言代码，US 代表国家地区)；adminMessages_zh_CN. properties 表示中文信息。

adminMessages. properties 的内容如下：

```
test. admin = \u6D4B\u8BD5\u540E\u53F0
admin = \u540E\u53F0\u9875\u9762
```

adminMessages_en_US. properties 的内容如下：

```
test. admin = test admin
admin = admin
```

adminMessages_zh_CN. properties 的内容如下：

```
test. admin = \u6D4B\u8BD5\u540E\u53F0
admin = \u540E\u53F0\u9875\u9762
```

2）编写用户模块的国际化信息

在 ch5_1 的 src/main/resources 目录下创建 i18n/before 文件夹，并在该文件夹下创建 beforeMessages. properties、beforeMessages_en_US. properties 和 beforeMessages_zh_CN

. properties 资源属性文件。

beforeMessages. properties 的内容如下：

```
test. before = \u6D4B\u8BD5\u524D\u53F0
before = \u524D\u53F0\u9875\u9762
```

beforeMessages_en_US. properties 的内容如下：

```
test. before = test before
before = before
```

beforeMessages_zh_CN. properties 的内容如下：

```
test. before = \u6D4B\u8BD5\u524D\u53F0
before = \u524D\u53F0\u9875\u9762
```

3）编写公共模块的国际化信息

在 ch5_1 的 src/main/resources 目录下创建 i18n/common 文件夹，并在该文件夹下创建 commonMessages. properties、commonMessages_en_US. properties 和 commonMessages_zh_CN. properties 资源属性文件。

commonMessages. properties 的内容如下：

```
chinese. key = \u4E2D\u6587\u7248
english. key = \u82F1\u6587\u7248
return = \u8FD4\u56DE\u9996\u9875
```

commonMessages_en_US. properties 的内容如下：

```
chinese. key = chinese
english. key = english
return = return
```

commonMessages_zh_CN. properties 的内容如下：

```
chinese. key = \u4E2D\u6587\u7248
english. key = \u82F1\u6587\u7248
return = \u8FD4\u56DE\u9996\u9875
```

2. 添加配置文件内容，引入资源属性文件

在 ch5_1 应用的配置文件中，添加如下内容，引入资源属性文件：

```
spring. messages. basename = i18n/admin/adminMessages, i18n/before/beforeMessages,
i18n/common/commonMessages
```

3. 重写 localeResolver 方法配置语言区域选择

在 ch5_1 应用的 com. ch. ch5_1 包中，创建配置类 LocaleConfig，该配置类实现 WebMvcConfigurer 接口，并配置语言区域选择。LocaleConfig 的代码如下：

```
package com. ch. ch5_1;
import java. util. Locale;
import org. springframework. boot. autoconfigure. EnableAutoConfiguration;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.LocaleResolver;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.i18n.LocaleChangeInterceptor;
import org.springframework.web.servlet.i18n.SessionLocaleResolver;
@Configuration
@EnableAutoConfiguration
public class LocaleConfig implements WebMvcConfigurer {
    /**
     * 根据用户本次会话过程中的语义设定语言区域
     *(如用户进入首页时选择的语言种类)
     * @return
     */
    @Bean
    public LocaleResolver localeResolver() {
        SessionLocaleResolver slr = new SessionLocaleResolver();
        //默认语言
        slr.setDefaultLocale(Locale.CHINA);
        return slr;
    }
    /**
     * 使用 SessionLocaleResolver 存储语言区域时,
     * 必须配置 localeChangeInterceptor 拦截器
     * @return
     */
    @Bean
    public LocaleChangeInterceptor localeChangeInterceptor() {
        LocaleChangeInterceptor lci = new LocaleChangeInterceptor();
        //选择语言的参数名
        lci.setParamName("locale");
        return lci;
    }
    /**
     * 注册拦截器
     */
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(localeChangeInterceptor());
    }
}
```

4. 创建控制器类 I18nTestController

在 ch5_1 应用的 com.ch.ch5_1.controller 包中,创建控制器类 I18nTestController,具体代码如下:

```
package com.ch.ch5_1.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
@RequestMapping("/i18n")
public class I18nTestController {
```

```
    @RequestMapping("/first")
    public String testI18n(){
        return "/i18n/first";
    }
    @RequestMapping("/admin")
    public String admin(){
        return "/i18n/admin";
    }
    @RequestMapping("/before")
    public String before(){
        return "/i18n/before";
    }
}
```

5. 创建视图页面,并获得国际化信息

在ch5_1应用的src/main/resources/templates目录下,创建文件夹i18n;并在该文件夹中创建admin.html、before.html和first.html视图页面,并在这些视图页面中使用th:text="#{xxx}"获得国际化信息。

admin.html的代码如下:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <span th:text="#{admin}"></span><br>
    <a th:href="@{/i18n/first}" th:text="#{return}"></a>
</body>
</html>
```

before.html的代码如下:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <span th:text="#{before}"></span><br>
    <a th:href="@{/i18n/first}" th:text="#{return}"></a>
</body>
</html>
```

first.html的代码如下:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
```

```
<title>Insert title here</title>
</head>
<body>
    <a th:href="@{/i18n/first(locale='zh_CN')}" th:text="#{chinese.key}"></a>
    <a th:href="@{/i18n/first(locale='en_US')}" th:text="#{english.key}"></a>
    <br>
    <a th:href="@{/i18n/admin}" th:text="#{test.admin}"></a><br>
    <a th:href="@{/i18n/before}"  th:text="#{test.before}"></a><br>
</body>
</html>
```

6. 运行

首先，运行 Ch51Application 主类。然后，访问 http://localhost:8080/ch5_1/i18n/first。运行效果如图 5.8 所示。

单击图 5.8 中的“英文版”，打开如图 5.9 所示的页面效果。

图 5.8　程序入口页面

图 5.9　英文版效果

5.2.5　Spring Boot 与 Thymeleaf 的表单验证

本节使用 Hibernate Validator 对表单进行验证，注意它和 Hibernate 无关，只是使用它进行数据验证。在 Spring MVC 的 Web 应用中，需要加载 Hibernate Validator 所依赖的 jar 包。而在 Spring Boot 的 Web 应用中，不再需要加载 Hibernate Validator 所依赖的 jar 包。这是因为 spring-boot-starter-web 中已经依赖了 hibernate-validator 的 jar 包。

使用 Hibernate Validator 验证表单时，需要利用它的标注类型在实体模型的属性上嵌入约束。

1. 空检查

@Null：验证对象是否为 null。

@NotNull：验证对象是否不为 null，无法检查长度为 0 的字符串。

@NotBlank：检查约束字符串是不是 null，还有被 trim 后的长度是否大于 0，只针对字符串，且会去掉前后空格。

@NotEmpty：检查约束元素是否为 null 或者是 empty。

示例代码如下：

```
@NotBlank(message = "{goods.gname.required}") //goods.gname.required 为属性文件的错误代码
private String gname;
```

2. booelan 检查

@AssertTrue：验证 boolean 属性是否为 true。

@AssertFalse：验证 boolean 属性是否为 false。

示例代码如下：

```
@AssertTrue
private boolean isLogin;
```

3. 长度检查

@Size(min=, max=)：验证对象(Array、Collection、Map、String)长度是否在给定的范围之内。

@Length(min=, max=)：验证字符串长度是否在给定的范围之内。

示例代码如下：

```
@Length(min = 1,max = 100)
private String gdescription;
```

4. 日期检查

@Past：验证 Date 和 Calendar 对象是否在当前时间之前。

@Future：验证 Date 和 Calendar 对象是否在当前时间之后。

@Pattern：验证 String 对象是否符合正则表达式的规则。

示例代码如下：

```
@Past(message = "{gdate.invalid}")
private Date gdate;
```

5. 数值检查

@Min：验证 Number 和 String 对象是否大于等于指定的值。

@Max：验证 Number 和 String 对象是否小于等于指定的值。

@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=)：验证字符串是否符合指定格式的数字，integer 指定整数精度，fraction 指定小数精度。

@Range(min=, max=)：检查数字是否介于 min 和 max 之间。

@Valid：对关联对象进行校验，如果关联对象是一个集合或者数组，那么对其中的元素进行校验；如果关联对象是一个 map，则对其中的值部分进行校验。

@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，通过验证。

示例代码如下：

```
@Range(min = 0,max = 100,message = "{gprice.invalid}")
private double gprice;
```

下面通过实例讲解使用 Hibernate Validator 验证表单的过程。

【例 5-4】 使用 Hibernate Validator 验证表单的过程。

具体实现步骤如下。

1. 创建表单实体模型

在 ch5_1 应用的 com.ch.ch5_1.model 包中，创建表单实体模型类 Goods，在该类使用 Hibernate Validator 的标注类型进行表单验证。代码如下：

```
package com.ch.ch5_1.model;
import javax.validation.constraints.NotBlank;
import org.hibernate.validator.constraints.Length;
import org.hibernate.validator.constraints.Range;
public class Goods {
    @NotBlank(message = "商品名必须输入")
    @Length(min = 1, max = 5, message = "商品名长度在 1 到 5 之间")
    private String gname;
    @Range(min = 0,max = 100,message = "商品价格在 0 到 100 之间")
    private double gprice;
    //省略 set 和 get 方法
}
```

2. 创建控制器

在 ch5_1 应用的 com.ch.ch5_1.controller 包中，创建控制器类 TestValidatorController。在该类中有两个处理方法，一个是界面初始化处理方法 testValidator，一个是添加请求处理方法 add。在 add 方法中，使用@Validated 注解使验证生效。代码如下：

```
package com.ch.ch5_1.controller;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import com.ch.ch5_1.model.Goods;
@Controller
public class TestValidatorController {
    @RequestMapping("/testValidator")
    public String testValidator(@ModelAttribute("goodsInfo") Goods goods){
        goods.setGname("商品名初始化");
        goods.setGprice(0.0);
        return "testValidator";
    }
```

```
    @RequestMapping(value = "/add")
    public String add(@ModelAttribute("goodsInfo") @Validated Goods goods,BindingResult rs){
        //@ModelAttribute("goodsInfo")与 th:object = " ${goodsInfo}"相对应
        if(rs.hasErrors()){           //验证失败
              return "testValidator";
           }
        //验证成功,可以到任意地方,在这里直接到 testValidator 界面
        return "testValidator";
    }
}
```

3. 创建视图页面

在 ch5_1 应用的 src/main/resources/templates 目录下,创建视图页面 testValidator.html。在视图页面中,直接读取到 ModelAttribute 里面注入的数据,然后通过 th:errors="*{xxx}"获得验证错误信息。代码如下:

```
<!DOCTYPE html>
<html xmlns:th = "http://www.thymeleaf.org">
<head>
<meta charset = "UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h2>通过 th:object 访问对象的方式</h2>
    <div th:object = " ${goodsInfo}">
        <p th:text = " *{gname}"></p>
        <p th:text = " *{gprice}"></p>
    </div>
    <h1>表单提交</h1>
    <!-- 表单提交用户信息,注意表单参数的设置,直接是 *{} -->
    <form  th:action = "@{/add}" th:object = " ${goodsInfo}" method = "post">
        <div><span>商品名</span><input type = "text" th:field = " *{gname}"/><span th:errors
      = " *{gname}"></span></div>
        <div><span>商品价格</span><input type = "text" th:field = " *{gprice}"/><span
      th:errors = " *{gprice}"></span></div>
          <input type = "submit" />
    </form>
</body>
</html>
```

4. 运行

首先,运行 Ch51Application 主类。然后,访问 http://localhost:8080/ch5_1/testValidator。测试效果如图 5.10 所示。

图 5.10　表单验证

5.2.6　基于 Thymeleaf 与 BootStrap 的 Web 开发实例

在本书的后续 Web 应用开发中，尽量使用前端开发工具包 BootStrap、JavaScript 框架 jQuery 和 Spring MVC 框架。BootStrap 和 jQuery 的相关知识，请读者自行学习。下面通过一个实例，讲解如何创建基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_2。

【例 5-5】 创建基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_2。

具体实现步骤如下。

1. 创建基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_2

选择菜单 File|New|Spring Starter Project，打开 New Spring Starter Project 对话框，在该对话框中选择和输入相关信息，如图 5.11 所示。

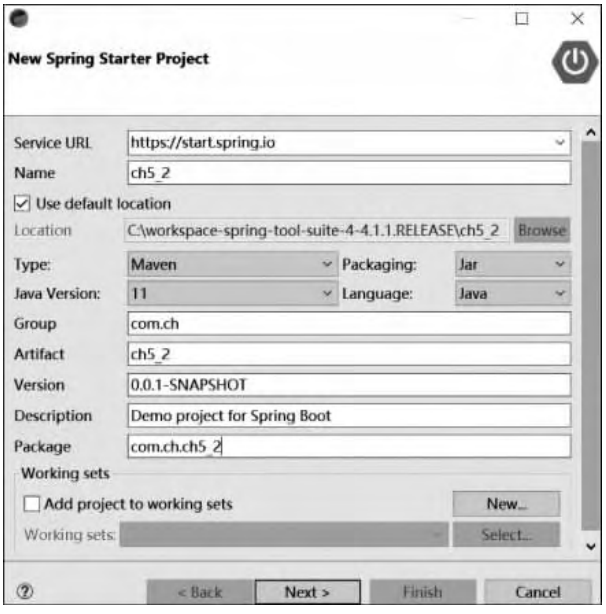

图 5.11　创建基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_2

单击图 5.11 中的 Next 按钮，打开 New Spring Starter Project Dependencies 对话框，选择 Spring Web Starter 和 Thymeleaf 依赖，如图 5.12 所示。

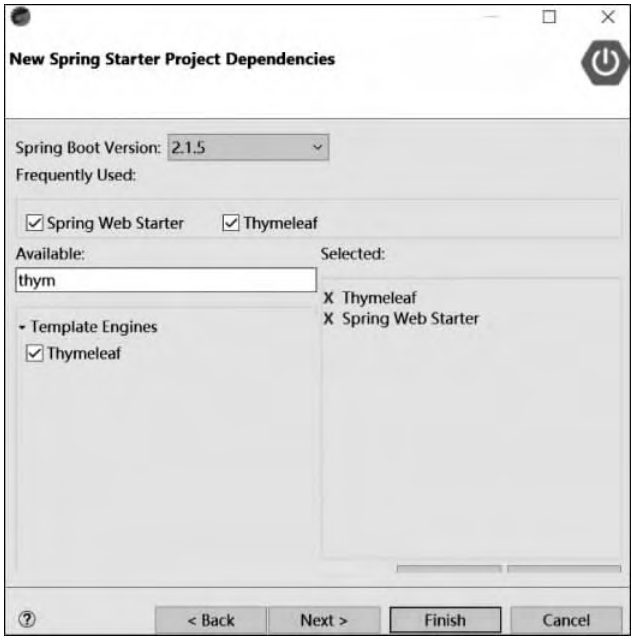

图 5.12 选择 Spring Web Starter 和 Thymeleaf 依赖

单击图 5.12 中的 Finish 按钮，完成创建基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_2。

2. 设置 Web 应用 ch5_2 的上下文路径

在 ch5_2 的 application.properties 文件中配置如下内容：

```
server.servlet.context-path = /ch5_2
```

3. 创建实体类 Book

创建名为 com.ch.ch5_2.model 的包，并在该包中创建名为 Book 的实体类，此实体类用在模板页面展示数据。代码如下：

```
package com.ch.ch5_2.model;
public class Book {
    String isbn;
    Double price;
    String bname;
    String publishing;
    String author;
    String picture;
    public Book(String isbn, Double price, String bname, String publishing, String author,
String picture) {
        super();
```

```
        this.isbn = isbn;
        this.price = price;
        this.bname = bname;
        this.publishing = publishing;
        this.author = author;
        this.picture = picture;
    }
    //省略 set 和 get 方法
}
```

4. 创建控制器类 ThymeleafController

创建名为 com.ch.ch5_2.controller 的包，并在该包中创建名为 ThymeleafController 的控制器类。在该控制器类中，实例化 Book 类的多个对象，并保存到集合 ArrayList<Book>中。代码如下：

```
package com.ch.ch5_2.controller;
import java.util.ArrayList;
import java.util.List;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import com.ch.ch5_1.model.Book;
@Controller
public class ThymeleafController {
    @RequestMapping("/")
    public String index(Model model) {
        Book teacherGeng = new Book(
                "9787302464259",
                59.5,
                "Java 2 实用教程(第 5 版)",
                "清华大学出版社",
                "耿祥义",
                "073423-02.jpg"
                );
        List<Book> chenHeng = new ArrayList<Book>();
        Book b1 = new Book(
                "9787302529118",
                69.8,
                "Java Web 开发从入门到实战(微课版)",
                "清华大学出版社",
                "陈恒",
                "082526-01.jpg"
                );
        chenHeng.add(b1);
        Book b2 = new Book(
                "9787302502968",
                69.8,
            "Java EE 框架整合开发入门到实战——Spring + Spring MVC + MyBatis(微课版)",
                "清华大学出版社",
```

```
                "陈恒",
                "079720-01.jpg");
        chenHeng.add(b2);
        model.addAttribute("aBook", teacherGeng);
        model.addAttribute("books", chenHeng);
        //根据 Thymeleaf 模板,默认将返回 src/main/resources/templates/index.html
        return "index";
    }
}
```

5. 整理脚本样式静态文件

JS 脚本、CSS 样式、图片等静态文件默认放置在 src/main/resources/static 目录下,ch5_2 应用引入了 BootStrap 和 jQuery,结构如图 5.13 所示。

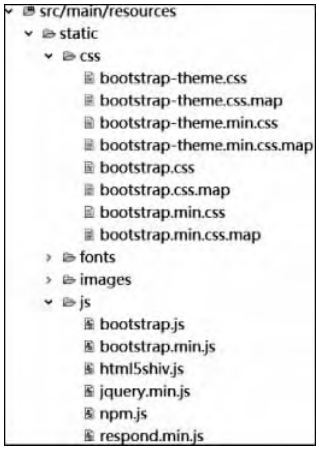

图 5.13 静态文件位置

6. View 视图页面

Thymeleaf 模板默认将视图页面放在 src/main/resources/templates 目录下。因此,在 src/main/resources/templates 目录下新建 html 页面文件 index.html。在该页面中,使用 Thymeleaf 模板显示控制器类 TestThymeleafController 中的 model 对象数据。代码如下:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<link rel="stylesheet" th:href="@{css/bootstrap.min.css}" />
<link rel="stylesheet" th:href="@{css/bootstrap-theme.min.css}" />
</head>
<body>
    <!-- 面板 -->
    <div class="panel panel-primary">
        <!-- 面板头信息 -->
        <div class="panel-heading">
            <!-- 面板标题 -->
            <h3 class="panel-title">第一个基于 Thymeleaf 模板引擎的 Spring Boot Web 应用
            </h3>
        </div>
    </div>
    <!-- 容器 -->
    <div class="container">
        <div>
        <h4>图书列表</h4>
    </div>
        <div class="row">
            <!-- col-md 针对桌面显示器,col-sm 针对平板 -->
```

```
            <div class = "col - md - 4 col - sm - 6">
                <a href = "">
                    <img th:src = "'images/' +  ${aBook.picture}" alt = "图书封面"
                    style = "height: 180px; width: 40 % ;"/>
                </a>
                <!-- caption 容器中放置其他基本信息,例如标题、文本描述等 -->
                <div class = "caption">
                    <h4 th:text = " ${aBook.bname}"></h4>
                    <p th:text = " ${aBook.author}"></p>
                    <p th:text = " ${aBook.isbn}"></p>
                    <p th:text = " ${aBook.price}"></p>
                    <p th:text = " ${aBook.publishing}"></p>
                </div>
            </div>
            <!-- 循环取出集合数据 -->
            <div class = "col - md - 4 col - sm - 6" th:each = "book: ${books}">
                <a href = "">
                    <img th:src = "'images/' +  ${book.picture}" alt = "图书封面"
                    style = "height: 180px; width: 40 % ;"/>
                </a>
                <div class = "caption">
                    <h4 th:text = " ${book.bname}"></h4>
                    <p th:text = " ${book.author}"></p>
                    <p th:text = " ${book.isbn}"></p>
                    <p th:text = " ${book.price}"></p>
                    <p th:text = " ${book.publishing}"></p>
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```

7. 运行

首先,运行 Ch52Application 主类。然后,访问 http://localhost:8080/ch5_2/。运行效果如图 5.14 所示。

图 5.14　例 5-5 运行结果

5.3 Spring Boot 处理 JSON 数据


视频讲解

在 Spring Boot 的 Web 应用中，内置了 JSON 数据的解析功能，默认使用 Jackson 自动完成解析(不需要加载 Jackson 依赖包)。当控制器返回一个 Java 对象或集合数据时，Spring Boot 自动将其转换成 JSON 数据，使用起来很方便简洁。

Spring Boot 处理 JSON 数据时，需要用到两个重要的 JSON 格式转换注解，分别是@RequestBody和@ResponseBody。

- @RequestBody：用于将请求体中的数据绑定到方法的形参中，该注解应用在方法的形参上。
- @ResponseBody：用于直接返回 JSON 对象，该注解应用在方法上。

下面通过一个实例讲解 Spring Boot 处理 JSON 数据的过程，该实例针对返回实体对象、ArrayList 集合、Map < String，Object >集合以及 List < Map < String，Object >>集合分别处理。

【例 5-6】 Spring Boot 处理 JSON 数据的过程。

具体实现步骤如下。

1. 创建实体类

在 ch5_2 应用的 com.ch.ch5_2.model 包中，创建实体类 Person。具体代码如下：

```
package com.ch.ch5_2.model;
public class Person {
    private String pname;
    private String password;
    private Integer page;
    //省略 set 和 get 方法
}
```

2. 创建视图页面

在 ch5_2 应用的 src/main/resources/templates 目录下，创建视图页面 input.html。在 input.html 页面中，引入 jQuery 框架，并使用它的 ajax 方法进行异步请求。具体代码如下：

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<link rel="stylesheet" th:href="@{css/bootstrap.min.css}" />
<!-- 默认访问 src/main/resources/static 下的 css 文件夹 -->
<link rel="stylesheet" th:href="@{css/bootstrap-theme.min.css}" />
<!-- 引入 jQuery -->
<script type="text/javascript" th:src="@{js/jquery.min.js}"></script>
<script type="text/javascript">
```

```
    function testJson() {
        //获取输入的值 pname 为 id
        var pname = $("#pname").val();
        var password = $("#password").val();
        var page = $("#page").val();
        alert(password);
        $.ajax({
            //发送请求的 URL 字符串
            url : "testJson",
            //定义回调响应的数据格式为 JSON 字符串,该属性可以省略
            dataType : "json",
            //请求类型
            type : "post",
            //定义发送请求的数据格式为 JSON 字符串
            contentType : "application/json",
            //data 表示发送的数据
            data : JSON.stringify({pname:pname,password:password,page:page}),
            //成功响应的结果
            success : function(data){
                if(data != null){
                    //返回一个 Person 对象
                    //alert("输入的用户名:" + data.pname + ",密码:" + data.password +
                    ",年龄:" + data.page);
                    //ArrayList<Person>对象
                    /** for(var i = 0; i < data.length; i++){
                        alert(data[i].pname);
                    } **/
                    //返回一个 Map<String, Object>对象
                    //alert(data.pname);      //pname 为 key
                    //返回一个 List<Map<String, Object>>对象
                    for(var i = 0; i < data.length; i++){
                        alert(data[i].pname);
                    }
                }
            },
            //请求出错
            error:function(){
                alert("数据发送失败");
            }
        });
    }
</script>
</head>
<body>
    <div class = "panel panel - primary">
        <div class = "panel - heading">
            <h3 class = "panel - title">处理 JSON 数据</h3>
        </div>
    </div>
    <div class = "container">
        <div>
```

```
        <h4>添加用户</h4>
        </div>
        <div class="row">
            <div class="col-md-6 col-sm-6">
                <form class="form-horizontal" action="">
                    <div class="form-group">
                        <div class="input-group col-md-6">
                            <span class="input-group-addon">
                                <i class="glyphicon glyphicon-pencil"></i>
                            </span>
                            <input class="form-control" type="text"
                             id="pname" th:placeholder="请输入用户名"/>
                        </div>
                    </div>
                    <div class="form-group">
                        <div class="input-group col-md-6">
                            <span class="input-group-addon">
                                <i class="glyphicon glyphicon-pencil"></i>
                            </span>
                            <input class="form-control" type="text"
                             id="password" th:placeholder="请输入密码"/>
                        </div>
                    </div>
                    <div class="form-group">
                        <div class="input-group col-md-6">
                            <span class="input-group-addon">
                                <i class="glyphicon glyphicon-pencil"></i>
                            </span>
                            <input class="form-control" type="text"
                             id="page" th:placeholder="请输入年龄"/>
                        </div>
                    </div>
                    <div class="form-group">
                        <div class="col-md-6">
                            <div class="btn-group btn-group-justified">
                                <div class="btn-group">
                                    <button type="button" onclick="testJson()"
                                        class="btn btn-success">
                                        <span class="glyphicon glyphicon-share"></span>
                                         测试
                                    </button>
                                </div>
                            </div>
                        </div>
                    </div>
                </form>
            </div>
        </div>
    </div>
</body>
</html>
```

3. 创建控制器

在 ch5_2 应用的 com.ch.ch5_2.controller 包中，创建控制器类 TestJsonController。在该类中有两个处理方法，一个是界面导航方法 input，一个是接收页面请求的方法。具体代码如下：

```
package com.ch.ch5_2.controller;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import com.ch.ch5_2.model.Person;
@Controller
public class TestJsonController {
    /**
     * 进入视图页面
     */
    @RequestMapping("/input")
    public String input() {
        return "input";
    }
    /**
     * 接收页面请求的 JSON 数据
     */
    @RequestMapping("/testJson")
    @ResponseBody
    /*@RestController 注解相当于@ResponseBody + @Controller 合在一起的作用。
    ①如果只是使用@RestController 注解 Controller，则 Controller 中的方法无法返回 jsp 页面或者 html，返回的内容就是 return 的内容。
    ②如果需要返回到指定页面，则需要用@Controller 注解。如果需要返回 JSON，XML 或自定义 mediaType 内容到页面，则需要在对应的方法上加上@ResponseBody 注解。
     */
    public List<Map<String, Object>> testJson(@RequestBody Person user) {
        //打印接收的 JSON 格式数据
        System.out.println("pname=" + user.getPname() +
                ", password=" + user.getPassword() + ",page=" + user.getPage());
        //返回 Person 对象
        //return user;
        /** ArrayList<Person> allp = new ArrayList<Person>();
        Person p1 = new Person();
        p1.setPname("陈恒 1");
        p1.setPassword("123456");
        p1.setPage(80);
        allp.add(p1);
```

```
        Person p2 = new Person();
        p2.setPname("陈恒 2");
        p2.setPassword("78910");
        p2.setPage(90);
        allp.add(p2);
        //返回 ArrayList<Person>对象
        return allp;
        **/
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("pname", "陈恒 2");
        map.put("password", "123456");
        map.put("page", 25);
        //返回一个 Map<String, Object>对象
        //return map;
        //返回一个 List<Map<String, Object>>对象
        List<Map<String, Object>> allp = new ArrayList<Map<String, Object>>();
        allp.add(map);
        Map<String, Object> map1 = new HashMap<String, Object>();
        map1.put("pname", "陈恒 3");
        map1.put("password", "54321");
        map1.put("page", 55);
        allp.add(map1);
        return allp;
    }
}
```

4. 运行

首先，运行 Ch52Application 主类。然后，访问 http://localhost:8080/ch5_2/input。运行效果如图 5.15 所示。

图 5.15 input.html 运行效果

5.4 Spring Boot 文件上传与下载


视频讲解

文件上传与下载是 Web 应用开发中常用的功能之一。本节将讲解如何在 Spring Boot 的 Web 应用开发中实现文件的上传与下载。

在实际的 Web 应用开发中，为了成功上传文件，必须将表单的 method 设置为 post，并将 enctype 设置为 multipart/form-data。只有这样设置，浏览器才能将所选文件的二进制数据发送给服务器。

从 Servlet 3.0 开始，就提供了处理文件上传的方法，但这种文件上传需要在 Java Servlet 中完成，而 Spring MVC 提供了更简单的封装。Spring MVC 是通过 Apache Commons FileUpload 技术实现一个 MultipartResolver 的实现类 CommonsMultipartResolver 完成文件上传的。因此，Spring MVC 的文件上传需要依赖 Apache Commons FileUpload 组件。

Spring MVC 将上传文件自动绑定到 MultipartFile 对象中，MultipartFile 提供了获取上传文件内容、文件名等方法，并通过 transferTo 方法将文件上传到服务器的磁盘中。MultipartFile 的常用方法如下。

- byte[] getBytes()：获取文件数据。
- String getContentType()：获取文件 MIME 类型，如 image/jpeg 等。
- InputStream getInputStream()：获取文件流。
- String getName()：获取表单中文件组件的名字。
- String getOriginalFilename()：获取上传文件的原名。
- long getSize()：获取文件的字节大小，单位为 b(byte)。
- boolean isEmpty()：是否有(选择)上传文件。
- void transferTo(File dest)：将上传文件保存到一个目标文件中。

Spring Boot 的 spring-boot-starter-web 已经集成了 Spring MVC，所以使用 Spring Boot 实现文件上传更加便捷，只需引入 Apache Commons FileUpload 组件依赖即可。

下面通过一个实例讲解 Spring Boot 文件上传与下载的实现过程。

【例 5-7】 Spring Boot 文件上传与下载。

具体实现步骤如下。

1. 引入 Apache Commons FileUpload 组件依赖

在 Web 应用 ch5_2 的 pom.xml 文件中，添加 Apache Commons FileUpload 组件依赖。代码如下：

```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <!-- 由于 commons-fileupload 组件不属于 Spring Boot,所以需要加上版本 -->
    <version>1.3.3</version>
</dependency>
```

2. 设置上传文件大小限制

在Web应用ch5_2的配置文件application.properties中,添加如下配置限制上传文件大小。

```
#上传文件时,默认单个上传文件大小是1MB,max-file-size设置单个上传文件大小
spring.servlet.multipart.max-file-size=50MB
#默认总文件大小是10MB,max-request-size设置总上传文件大小
spring.servlet.multipart.max-request-size=500MB
```

3. 创建选择文件视图页面

在ch5_2应用的src/main/resources/templates目录下,创建选择文件视图页面uploadFile.html。该页面中有一个enctype属性值为multipart/form-data的form表单。具体代码如下:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<link rel="stylesheet" th:href="@{css/bootstrap.min.css}" />
<!-- 默认访问 src/main/resources/static 下的 css 文件夹 -->
<link rel="stylesheet" th:href="@{css/bootstrap-theme.min.css}" />
</head>
<body>
<div class="panel panel-primary">
        <div class="panel-heading">
            <h3 class="panel-title">文件上传示例</h3>
        </div>
    </div>
    <div class="container">
        <div class="row">
            <div class="col-md-6 col-sm-6">
                <form class="form-horizontal" action="upload"
                method="post" enctype="multipart/form-data">
                    <div class="form-group">
                        <div class="input-group col-md-6">
                            <span class="input-group-addon">
                                <i class="glyphicon glyphicon-pencil"></i>
                            </span>
                            <input class="form-control" type="text"
                             name="description" th:placeholder="文件描述"/>
                        </div>
                    </div>
                    <div class="form-group">
                        <div class="input-group col-md-6">
                            <span class="input-group-addon">
                                <i class="glyphicon glyphicon-search"></i>
                            </span>
```

```
                    <input class="form-control" type="file"
                     name="myfile" th:placeholder="请选择文件"/>
                </div>
            </div>
            <div class="form-group">
                <div class="col-md-6">
                    <div class="btn-group btn-group-justified">
                        <div class="btn-group">
                            <button type="submit" class="btn btn-success">
                                <span class="glyphicon glyphicon-share"></span>
                                 上传文件
                            </button>
                        </div>
                    </div>
                </div>
            </div>
        </form>
    </div>
  </div>
</body>
</html>
```

4. 创建控制器

在 ch5_2 应用的 com.ch.ch5_2.controller 包中，创建控制器类 TestFileUpload。在该类中有 4 个处理方法，一个是界面导航方法 uploadFile，一个是实现文件上传的 upload 方法，一个是显示将要被下载文件的 showDownLoad 方法，一个是实现下载功能的 download 方法。具体代码如下：

```
package com.ch.ch5_2.controller;
import java.io.File;
import java.io.IOException;
import java.net.URLEncoder;
import javax.servlet.http.HttpServletRequest;
import org.apache.commons.io.FileUtils;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.http.ResponseEntity.BodyBuilder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;
@Controller
public class TestFileUpload {
    /**
     * 进入文件选择页面
     */
```

```
@RequestMapping("/uploadFile")
public String uploadFile() {
    return "uploadFile";
}
/**
 * 上传文件自动绑定到MultipartFile对象中,
 * 在这里使用处理方法的形参接收请求参数。
 */
@RequestMapping("/upload")
public String upload(
        HttpServletRequest request,
        @RequestParam("description") String description,
        @RequestParam("myfile") MultipartFile myfile)
        throws IllegalStateException, IOException {
    System.out.println("文件描述:" + description);
    //如果选择了上传文件,将文件上传到指定的目录uploadFiles
    if(!myfile.isEmpty()) {
        //上传文件路径
        String path = request.getServletContext().getRealPath("/uploadFiles/");
        //获得上传文件原名
        String fileName = myfile.getOriginalFilename();
        File filePath = new File(path + File.separator + fileName);
        //如果文件目录不存在,创建目录
        if(!filePath.getParentFile().exists()) {
            filePath.getParentFile().mkdirs();
        }
        //将上传文件保存到一个目标文件中
        myfile.transferTo(filePath);
    }
    //转发到一个请求处理方法,查询将要下载的文件
    return "forward:/showDownLoad";
}
/**
 * 显示要下载的文件
 */
@RequestMapping("/showDownLoad")
public String showDownLoad(HttpServletRequest request, Model model) {
    String path = request.getServletContext().getRealPath("/uploadFiles/");
    File fileDir = new File(path);
    //从指定目录获得文件列表
    File filesList[] = fileDir.listFiles();
    model.addAttribute("filesList", filesList);
    return "showFile";
}
/**
 * 实现下载功能
 */
@RequestMapping("/download")
public ResponseEntity<byte[]> download(
        HttpServletRequest request,
        @RequestParam("filename") String filename,
```

```
            @RequestHeader("User-Agent") String userAgent) throws IOException {
        //下载文件路径
        String path = request.getServletContext().getRealPath("/uploadFiles/");
        //构建将要下载的文件对象
        File downFile = new File(path + File.separator + filename);
        //ok 表示 HTTP 中的状态是 200
        BodyBuilder builder =  ResponseEntity.ok();
        //内容长度
        builder.contentLength(downFile.length());
        //application/octet-stream:二进制流数据(最常见的文件下载)
        builder.contentType(MediaType.APPLICATION_OCTET_STREAM);
        //使用 URLEncoder.encode 对文件名进行编码
        filename = URLEncoder.encode(filename,"UTF-8");
        /**
         * 设置实际的响应文件名,告诉浏览器文件要用于"下载"和"保存"。
         * 不同的浏览器,处理方式不同,根据浏览器的实际情况区别对待。
         */
        if(userAgent.indexOf("MSIE") > 0) {
            //IE 浏览器,只需要用 UTF-8 字符集进行 URL 编码
            builder.header("Content-Disposition", "attachment; filename=" + filename);
        }else {
            /** 非 IE 浏览器,如 FireFox、Chrome 等浏览器,则需要说明编码的字符集
             * filename 后面有一个 * 号,在 UTF-8 后面有两个单引号
             */
            builder.header("Content-Disposition", "attachment; filename* = UTF-8''" +
filename);
        }
        return builder.body(FileUtils.readFileToByteArray(downFile));
    }
}
```

5. 创建文件下载视图页面

在 ch5_2 应用的 src/main/resources/templates 目录下,创建文件下载视图页面 showFile.html。具体代码如下:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<link rel="stylesheet" th:href="@{css/bootstrap.min.css}" />
<!-- 默认访问 src/main/resources/static 下的 css 文件夹 -->
<link rel="stylesheet" th:href="@{css/bootstrap-theme.min.css}" />
<body>
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h3 class="panel-title">文件下载示例</h3>
        </div>
    </div>
    <div class="container">
```

```
        <div class = "panel panel - primary">
            <div class = "panel - heading">
                <h3 class = "panel - title">文件列表</h3 >
            </div >
            <div class = "panel - body">
                <div class = "table table - responsive">
                    <table class = "table table - bordered table - hover">
                        <tbody class = "text - center">
                            <tr th:each = "file,fileStat: ${filesList}">
                                <td >
                                    <span th:text = " ${fileStat.count}"></span >
                                </td >
                                <td >
                                <!-- file.name 相当于调用 getName()方法获得文件名称  -->
                                    <a th:href = "@{download(filename = ${file.name})}">
                                        <span th:text = " ${file.name}"></span >
                                    </a >
                                </td >
                            </tr >
                        </tbody >
                    </table >
                </div >
            </div >
        </div >
    </div >
</body >
</html >
```

6. 运行

首先，运行 Ch52Application 主类。然后，访问 http://localhost:8080/ch5_2/uploadFile。运行效果如图 5.16 所示。

图 5.16 文件选择界面

在图 5.16 中输入文件描述，并选择上传文件后，单击“上传文件”按钮，实现文件上传。文件上传成功后，打开如图 5.17 所示的下载文件列表页面。

单击图 5.17 中的文件名即可下载文件，至此，文件上传与下载示例演示完毕。

图 5.17　下载文件列表页面

5.5　Spring Boot 的异常统一处理


视频讲解

在 Spring Boot 应用的开发中，不管是对底层数据库操作，对业务层操作，还是对控制层操作，都会不可避免地遇到各种可预知的、不可预知的异常需要处理。如果每个过程都单独处理异常，那么系统的代码耦合度高、工作量大且不好统一，以后维护的工作量也很大。

如果能将所有类型的异常处理从各层中解耦出来，则既保证了相关处理过程的功能较单一，也实现了异常信息的统一处理和维护。幸运的是，Spring 框架支持这样的实现。本节将从自定义 error 页面、@ExceptionHandler 注解以及@ControllerAdvice 3 种方式讲解 Spring Boot 应用的异常统一处理。

5.5.1　自定义 error 页面

在 Spring Boot Web 应用的 src/main/resources/templates 目录下添加 error.html 页面，访问发生错误或异常时，Spring Boot 将自动找到该页面作为错误页面。Spring Boot 为错误页面提供了以下属性。

- timestamp：错误发生时间；
- status：HTTP 状态码；
- error：错误原因；
- exception：异常的类名；
- message：异常消息(如果这个错误是由异常引起的)；
- errors：BindingResult 异常里的各种错误(如果这个错误是由异常引起的)；
- trace：异常跟踪信息(如果这个错误是由异常引起的)；
- path：错误发生时请求的 URL 路径。

下面通过一个实例讲解在 Spring Boot 应用的开发中，如何使用自定义 error 页面。

【例 5-8】　自定义 error 页面。

具体实现步骤如下。

1. 创建基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_3

参照 5.2.6 节的例 5-5，创建基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_3。

2. 设置 Web 应用 ch5_3 的上下文路径

在 ch5_3 的 application.properties 文件中配置如下内容：

```
server.servlet.context-path=/ch5_3
```

3. 创建自定义异常类 MyException

创建名为 com.ch.ch5_3.exception 的包，并在该包中创建名为 MyException 的异常类。具体代码如下：

```
package com.ch.ch5_3.exception;
public class MyException extends Exception {
    private static final long serialVersionUID = 1L;
    public MyException() {
        super();
    }
    public MyException(String message) {
        super(message);
    }
}
```

4. 创建控制器类 TestHandleExceptionController

创建名为 com.ch.ch5_3.controller 的包，并在该包中创建名为 TestHandleExceptionController 的控制器类。在该控制器类中，有 4 个请求处理方法，一个是导航到 index.html，另外 3 个分别抛出不同的异常（并没有处理异常）。具体代码如下：

```
package com.ch.ch5_3.controller;
import java.sql.SQLException;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import com.ch.ch5_3.exception.MyException;
@Controller
public class TestHandleExceptionController {
    @RequestMapping("/")
    public String index() {
        return "index";
    }
    @RequestMapping("/db")
    public void db() throws SQLException {
        throw new SQLException("数据库异常");
    }
    @RequestMapping("/my")
    public void my() throws MyException {
        throw new MyException("自定义异常");
```

```
    }
    @RequestMapping("/no")
    public void no() throws Exception {
        throw new Exception("未知异常");
    }
}
```

5. 整理脚本样式静态文件

JS 脚本、CSS 样式、图片等静态文件默认放置在 src/main/resources/static 目录下，ch5_3 应用引入了与 ch5_2 一样的 BootStrap 和 jQuery。

6. View 视图页面

Thymeleaf 模板默认将视图页面放在 src/main/resources/templates 目录下。因此，我们在 src/main/resources/templates 目录下新建 html 页面文件 index.html 和 error.html。

在 index.html 页面中，有 4 个超链接请求，3 个请求在控制器中有对应处理，另一个请求是 404 错误。具体代码如下：

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>index</title>
<link rel="stylesheet" th:href="@{css/bootstrap.min.css}" />
<!-- 默认访问 src/main/resources/static 下的 css 文件夹 -->
<link rel="stylesheet" th:href="@{css/bootstrap-theme.min.css}" />
</head>
<body>
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h3 class="panel-title">异常处理示例</h3>
        </div>
    </div>
    <div class="container">
        <div class="row">
            <div class="col-md-4 col-sm-6">
                <a th:href="@{db}">处理数据库异常</a><br>
                <a th:href="@{my}">处理自定义异常</a><br>
                <a th:href="@{no}">处理未知错误</a>
                <hr>
                <a th:href="@{nofound}">404 错误</a>
            </div>
        </div>
    </div>
</body>
</html>
```

在 error.html 页面中，使用 Spring Boot 为错误页面提供的属性显示错误消息。具体代码如下：

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>error</title>
<link rel="stylesheet" th:href="@{css/bootstrap.min.css}" />
<!-- 默认访问 src/main/resources/static 下的 css 文件夹 -->
<link rel="stylesheet" th:href="@{css/bootstrap-theme.min.css}" />
</head>
<body>
  <div class="panel-l container clearfix">
        <div class="error">
            <p class="title"><span class="code" th:text="${status}"></span>非常抱歉,
没有找到您要查看的页面</p>
            <div class="common-hint-word">
                <div th:text="${#dates.format(timestamp,'yyyy-MM-dd HH:mm:ss')}"></div>
                <div th:text="${message}"></div>
                <div th:text="${error}"></div>
            </div>
        </div>
    </div>
</body>
</html>
```

7. 运行

首先，运行 Ch53Application 主类。然后，访问 http://localhost:8080/ch5_3/打开 index.html 页面，运行效果如图 5.18 所示。

单击图 5.18 中的超链接时，Spring Boot 应用将根据链接请求，到控制器中找对应的处理。例如，单击图 5.18 中的“处理数据库异常”链接时，将执行控制器中的 public void db() throws SQLException 方法，而该方法仅仅抛出了 SQLException 异常，并没有处理异常。当 Spring Boot 发现有异常抛出且没有处理时，将自动在 src/main/resources/templates 目录下找到 error.html 页面显示异常信息，效果如图 5.19 所示。

图 5.18 index.html 页面

图 5.19 error.html 页面

从上述例 5-8 的运行结果可以看出，使用自定义 error 页面并没有真正处理异常，只是将异常或错误信息显示给客户端，因为在服务器控制台上同样抛出了异常，如图 5.20 所示。

图 5.20　异常信息

5.5.2　@ExceptionHandler 注解

在 5.5.1 节中使用自定义 error 页面并没有真正处理异常，在本节可以使用@ExceptionHandler 注解处理异常。如果在 Controller 中有一个使用@ExceptionHandler 注解修饰的方法，那么当 Controller 的任何方法抛出异常时，都由该方法处理异常。

下面通过实例讲解如何使用@ExceptionHandler 注解处理异常。

【例 5-9】 使用@ExceptionHandler 注解处理异常。

具体实现步骤如下。

1. 在控制器类中添加使用@ExceptionHandler 注解修饰的方法

在例 5-8 的控制器类 TestHandleExceptionController 中，添加一个使用@ExceptionHandler 注解修饰的方法，具体代码如下：

```
@ExceptionHandler(value = Exception.class)
public String handlerException(Exception e) {
    //数据库异常
    if (e instanceof SQLException) {
        return "sqlError";
    } else if (e instanceof MyException) {        //自定义异常
        return "myError";
    } else {                                      //未知异常
        return "noError";
    }
}
```

2. 创建 sqlError、myError 和 noError 页面

在 ch5_3 的 src/main/resources/templates 目录下，创建 sqlError、myError 和 noError 页面。当发生 SQLException 异常时，Spring Boot 处理后，显示 sqlError 页面；当发生 MyException 异常时，Spring Boot 处理后，显示 myError 页面；当发生未知异常时，Spring Boot 处理后，显示 noError 页面。具体代码略。

3. 运行

再次运行 Ch53Application 主类后，访问 http://localhost:8080/ch5_3/打开 index.html页面，单击“处理数据库异常”链接时，执行控制器中的 public void db() throws SQLException 方法，该方法抛出了 SQLException，这时 Spring Boot 会自动执行使用@ExceptionHandler 注解修饰的方法 public String handlerException(Exception e)进行异常处理并打开 sqlError.html 页面，同时观察控制台有没有抛出异常信息。注意单击“404

错误”链接时，还是由自定义 error 页面显示错误信息，这是因为没有执行控制器中抛出异常的方法，进而不会执行使用@ExceptionHandler 注解修饰的方法。

从例 5-9 可以看出，在控制器中添加使用@ExceptionHandler 注解修饰的方法才能处理异常。而一个 Spring Boot 应用中往往存在多个控制器，不太适合在每个控制器中添加使用@ExceptionHandler 注解修饰的方法进行异常处理。可以将使用@ExceptionHandler 注解修饰的方法放到一个父类中，然后所有需要处理异常的控制器继承该类即可。例如，可以将例 5-9 中使用@ExceptionHandler 注解修饰的方法移到一个父类 BaseController 中，然后让控制器类 TestHandleExceptionController 继承该父类即可处理异常。

5.5.3 @ControllerAdvice 注解

使用 5.5.2 节中父类 Controller 进行异常处理，也有其自身的缺点，那就是代码耦合性太高。可以使用@ControllerAdvice 注解降低这种父子耦合关系。

@ControllerAdvice 注解，顾名思义，是一个增强的 Controller。使用该 Controller，可以实现 3 个方面的功能：全局异常处理、全局数据绑定以及全局数据预处理。本节将学习如何使用@ControllerAdvice 注解进行全局异常处理。

使用@ControllerAdvice 注解的类是当前 Spring Boot 应用中所有类的统一异常处理类，该类中使用@ExceptionHandler 注解的方法统一处理异常，不需要在每个 Controller 中逐一定义异常处理方法，这是因为对所有注解了@RequestMapping 的控制器方法有效。

下面通过实例讲解如何使用@ControllerAdvice 注解进行全局异常处理。

【例 5-10】 使用@ControllerAdvice 注解进行全局异常处理。

具体实现步骤如下。

1. 创建使用@ControllerAdvice 注解的类

在 ch5_3 的 com.ch.ch5_3.controller 包中，创建名为 GlobalExceptionHandlerController 的类。使用@ControllerAdvice 注解修饰该类，并将例 5-9 中使用@ExceptionHandler 注解修饰的方法移到该类中，具体代码如下：

```
package com.ch.ch5_3.controller;
import java.sql.SQLException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import com.ch.ch5_3.exception.MyException;
@ControllerAdvice
public class GlobalExceptionHandlerController {
    @ExceptionHandler(value = Exception.class)
    public String handlerException(Exception e) {
        //数据库异常
        if (e instanceof SQLException) {
            return "sqlError";
        } else if (e instanceof MyException) {          //自定义异常
            return "myError";
        } else {                                        //未知异常
            return "noError";
        }
```

```
    }
}
```

2. 运行

再次运行 Ch53Application 主类后，访问 http://localhost:8080/ch5_3/打开 index.html页面测试即可。

5.6 Spring Boot 对 JSP 的支持


视频讲解

尽管 Spring Boot 建议使用 HTML 完成动态页面，但也有部分 Java Web 应用使用 JSP 完成动态页面。遗憾的是 Spring Boot 官方不推荐使用 JSP 技术，但考虑到是常用的技术，本节将介绍 Spring Boot 如何集成 JSP 技术。

下面通过实例讲解 Spring Boot 如何集成 JSP 技术。

【例 5-11】 Spring Boot 集成 JSP 技术。

具体实现步骤如下。

1. 创建 Spring Boot Web 应用 ch5_4

选择菜单 File|New|Spring Starter Project，打开 New Spring Starter Project 对话框，在该对话框中选择和输入相关信息，如图 5.21 所示。

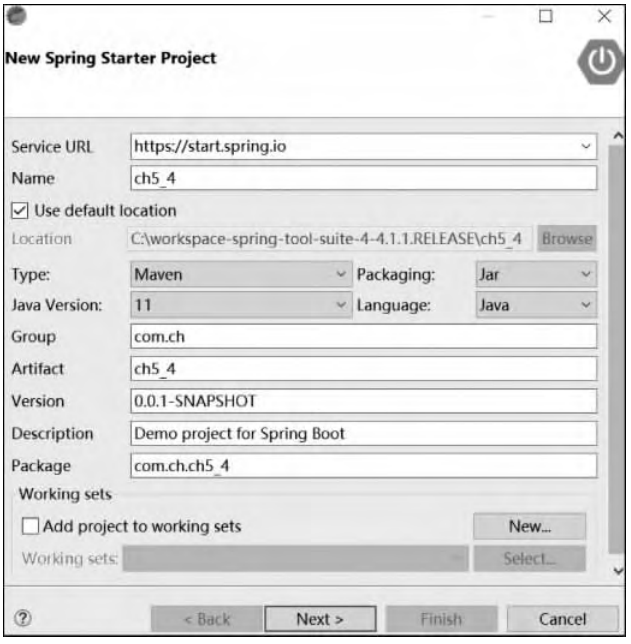

图 5.21 创建 Spring Boot Web 应用 ch5_4

单击图 5.21 中的 Next 按钮，打开 New Spring Starter Project Dependencies 对话框，在该对话框中，选择 Spring Web Starter 依赖，如图 5.22 所示。

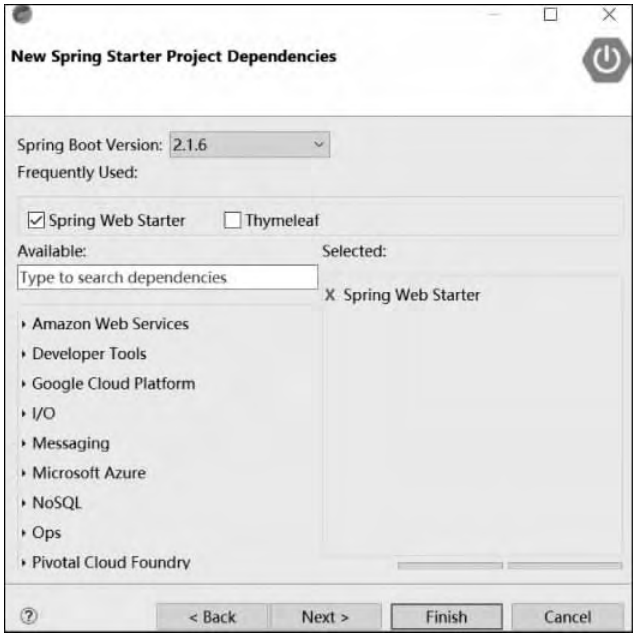

图 5.22 选择 Spring Web Starter 依赖

单击图 5.22 中的 Finish 按钮，完成创建 Spring Boot Web 应用 ch5_4。

2. 修改 pom.xml 文件，添加 Servlet、Tomcat 和 JSTL 依赖

因为在 JSP 页面中使用 EL 和 JSTL 标签显示数据，所以在 pom.xml 文件中，除了添加 Servlet 和 Tomcat 依赖外，还需要添加 JSTL 依赖。具体代码如下：

```
<!-- 添加 Servlet 依赖 -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <scope>provided</scope>
    <!-- provided 被依赖包理论上可以参与编译、测试、运行等阶段，相当于 compile，但是在打包阶
    段做了 exclude 的动作。适用场景：例如，如果我们在开发一个 Web 应用，在编译时需要依赖
    servlet-api.jar，但是在运行时不需要该 jar 包，因为这个 jar 包已由应用服务器提供，此时需
    要使用 provided 进行范围修饰。 -->
</dependency>
<!-- 添加 Tomcat 依赖 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>
<!-- Jasper 是 Tomcat 使用的引擎，使用 tomcat-embed-jasper 可以将 Web 应用在内嵌的 Tomcat
下运行 -->
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
```

```
    <scope>provided</scope>
</dependency>
<!-- 添加 JSTL 依赖 -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <!-- 如果没有指定 scope 值,该元素的默认值为 compile。被依赖包需要参与到当前项目的编
    译、测试、打包、运行等阶段。打包的时候通常会包含被依赖包。 -->
</dependency>
```

3. 设置 Web 应用 ch5_4 的上下文路径及页面配置信息

在 ch5_4 的 application.properties 文件中配置如下内容：

```
server.servlet.context-path=/ch5_4
#设置页面前缀目录
spring.mvc.view.prefix=/WEB-INF/jsp/
#设置页面后缀
spring.mvc.view.suffix=.jsp
```

4. 创建实体类 Book

创建名为 com.ch.ch5_4.model 的包，并在该包中创建名为 Book 的实体类。此实体类用在模板页面展示数据，代码与例 5-5 中的 Book 一样，不再赘述。

5. 创建控制器类 ThymeleafController

创建名为 com.ch.ch5_4.controller 的包，并在该包中创建名为 ThymeleafController 的控制器类。在该控制器类中，实例化 Book 类的多个对象，并保存到集合 ArrayList<Book>中。代码与例 5-5 中的 ThymeleafController 一样，不再赘述。

6. 整理脚本样式静态文件

JS 脚本、CSS 样式、图片等静态文件默认放置在 src/main/resources/static 目录下，ch5_4 应用引入的 BootStrap 和 jQuery 与例 5-5 中的一样，不再赘述。

7. View 视图页面

从 application.properties 配置文件中可知，将 JSP 文件路径指定到/WEB-INF/jsp/目录。因此，我们需要在 src/main 目录下创建目录 webapp/WEB-INF/jsp/，并在该目录下创建 JSP 文件 index.jsp。代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!-- 引入 JSTL 标签 -->
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    String path = request.getContextPath();
    String basePath = request.getScheme() + "://" + request.getServerName() + ":" +
request.getServerPort() + path + "/";
%>
```

```
<!DOCTYPE html>
<html>
<head>
<base href="<% =basePath %>">
<meta charset="UTF-8">
<title>JSP 测试</title>
<link href="css/bootstrap.min.css" rel="stylesheet">
<link href="css/bootstrap-theme.min.css" rel="stylesheet">
</head>
<body>
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h3 class="panel-title">第一个基于 JSP 技术的 Spring Boot Web 应用</h3>
        </div>
    </div>
    <div class="container">
        <div>
        <h4>图书列表</h4>
    </div>
        <div class="row">
            <div class="col-md-4 col-sm-6">
                <!-- 使用 EL 表达式 -->
                <a href="">
        <img src="images/${aBook.picture}" alt="图书封面" style="height: 180px; width:
40%;"/>
                </a>
                <div class="caption">
                    <h4>${aBook.bname}</h4>
                    <p>${aBook.author}</p>
                    <p>${aBook.isbn}</p>
                    <p>${aBook.price}</p>
                    <p>${aBook.publishing}</p>
                </div>
            </div>
            <!-- 使用 JSTL 标签 forEach 循环取出集合数据 -->
            <c:forEach var="book" items="${books}">
                <div class="col-md-4 col-sm-6">
                <a href="">
            <img src="images/${book.picture}" alt="图书封面" style="height: 180px;
width: 40%;"/>
                </a>
                <div class="caption">
                    <h4>${book.bname}</h4>
                    <p>${book.author}</p>
                    <p>${book.isbn}</p>
                    <p>${book.price}</p>
                    <p>${book.publishing}</p>
                </div>
            </div>
            </c:forEach>
        </div>
    </div>
</body>
</html>
```

8. 运行

首先，运行 Ch54Application 主类。然后，访问 http://localhost:8080/ch5_4/。运行效果如图 5.23 所示。

图 5.23　例 5-11 运行结果

5.7　本章小结

本章首先介绍了 Spring Boot 的 Web 开发支持，然后详细讲述了 Spring Boot 推荐使用的 Thymeleaf 模板引擎，包括 Thymeleaf 的基础语法、常用属性以及国际化。同时，本章还介绍了 Spring Boot 对 JSON 数据的处理、文件上传下载、异常统一处理和对 JSP 的支持等 Web 应用开发的常用功能。

习题 5

使用 Hibernate Validator 验证如图 5.24 所示的表单信息，具体要求如下：

(1) 用户名必须输入，并且长度范围为 5～20。

(2) 年龄范围为 18～60。

(3) 工作日期在系统时间之前。

图 5.24　输入页面