第 5 章 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、文件的上传与下载、异常统一处理以及对 JSP 的支持)。通过本章的学习，读者应该掌握 Spring Boot 的 Web 开发技术。

本章主要内容

- Thymeleaf 模板引擎
- 使用 Spring Boot 处理 JSON 数据
- Spring Boot 中文件的上传与下载
- Spring Boot 的异常处理
- Spring Boot 对 JSP 的支持

Web 开发是一种基于 B/S(即浏览器/服务器)架构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：

```
<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，并实现相关技术的自动配置。

另外，开发者也可以使用 IDEA 集成开发工具快速创建 Spring Initializr，在 New Project 对话框中添加 Spring Boot 的 Web 依赖。

5.2 Thymeleaf 模板引擎

在 Spring Boot 的 Web 应用中，建议开发者使用 HTML 开发动态页面。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/HTML 5 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，需要开发者自己选择 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。

其具体实现步骤如下。

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

在 IDEA 中选择 File→New→Project 命令，打开 New Project 对话框；在 New Project 对话框中选择和输入相关信息，然后单击 Next 按钮，打开新的界面；在新的界面中选择 Thymeleaf、Lombok 和 Spring Web 依赖，单击 Create 按钮，即可创建 ch5_1 应用。

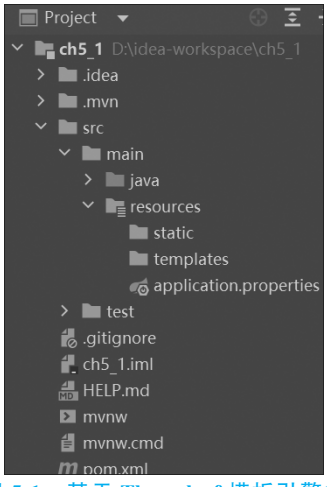

图 5.1　基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_1

❷ 打开项目目录

打开已经创建的基于 Thymeleaf 模板引擎的 Spring Boot Web 应用 ch5_1，如图 5.1 所示。

Thymeleaf 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。

❸ 设置 Web 应用 ch5_1 的上下文路径

在 ch5_1 应用的 application.properties 文件中配置如下内容：

```
server.servlet.context-path=/ch5_1
```

❹ 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，具体代码如下：

```
package com.ch.ch5_1.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
@Controller
public class TestThymeleafController {
    @GetMapping("/")
    public String test(){
        //根据 Thymeleaf 模板，默认返回 src/main/resources/templates/index.html
        return "index";
    }
}
```

❺ 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>
```

❻ 运行

首先运行 Ch51Application 主类，然后访问“http://localhost:8080/ch5_1/”，运行结果如图 5.2 所示。

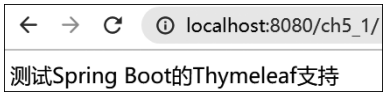

图 5.2 例 5-1 的运行结果

▶5.2.2 Thymeleaf 的基础语法

❶ 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中使用 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。

❷ 输出内容

使用 th:text 和 th:utext(不对文本转义，正常输出)将文本内容输出到所在标签的 body 中。如果在国际化资源文件 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>
<!-- 对文本转义，即输出<strong>Test International Message</strong> -->
<p th:utext="#{test.myText}"></p>
<!-- 对文本不转义，即输出加粗的“Test International Message” -->
```

❸ 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>
```

❹ 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}"/>
```

❺ 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}：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}：返回一个名为 xxx 的请求参数（可能是多个值）。
- ${session.xxx}：返回一个名为 xxx 的 HttpSession 作用域中的属性。
- ${application.xxx}：返回一个名为 xxx 的全局 ServletContext 上下文作用域中的属性。

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。需要注意的是，${}表达式只能在 th 标签内部有效。

❻ 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>
```

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

❽ 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，当前循环是否为偶数/奇数(从 0 开始计数)。
- first：布尔值，当前循环是否为第一个。
- last：布尔值，当前循环是否为最后一个。

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>
```

❾ 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。

例如，有如下控制器方法：

```
@GetMapping("/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 数据是否为 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>
```

扫一扫 视频讲解

▶5.2.3　Thymeleaf 的常用属性

通过 5.2.2 节的学习,发现 Thymeleaf 语法都是通过在 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。Thymeleaf 的常用属性如下。

❶ th:action

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

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

❷ th:each

th:each 用于集合对象的遍历,功能类似 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>
```

❸ th:field

th:field 用于表单参数的绑定,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>
```

❹ th:href

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

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

❺ th:id

th:id 用于 div 的 id 声明,类似 HTML 标签中的 id 属性。示例代码如下:

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

❻ 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>
```

❼ th:fragment

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

例如，在 ch5_1 应用的 src/main/resources/templates 目录下声明模板片段文件 footer.html，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 应用的 src/main/resources/templates/index.html 文件中引入模板片段，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>
```

❽ th:object

th:object 用于表单数据对象的绑定，将表单绑定到后台 controller 的一个 JavaBean 参数，通常与 th:field 一起使用。下面通过一个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。

其具体实现步骤如下。

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;
import lombok.Data;
@Data
```

```
public class LoginBean {
    String uname;
    String urole;
}
```

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，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.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import com.ch.ch5_1.model.LoginBean;
@Controller
public class LoginController {
    @GetMapping("/toLogin")
    public String toLogin(Model model) {
        /* loginBean 与 login.html 页面中的 th:object="${loginBean}"相同 */
        model.addAttribute("loginBean", new LoginBean());
        return "login";
    }
    @PostMapping("/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。

login.html 页面的代码如下：

```
<!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>
```