**实验九 SpringMVC项目搭建**

学生姓名： 张小锋 学 号： 8002119102 专业班级： 计软192班

实验类型：□ 验证 □ 综合 □ 设计 □ 创新 实验日期：2021.12.1 实验成绩：

1. 实验项目名称

SpringMVC项目搭建

1. 实验目的

根据步骤搭建一个SpringMVC+Mybatis+Spring项目

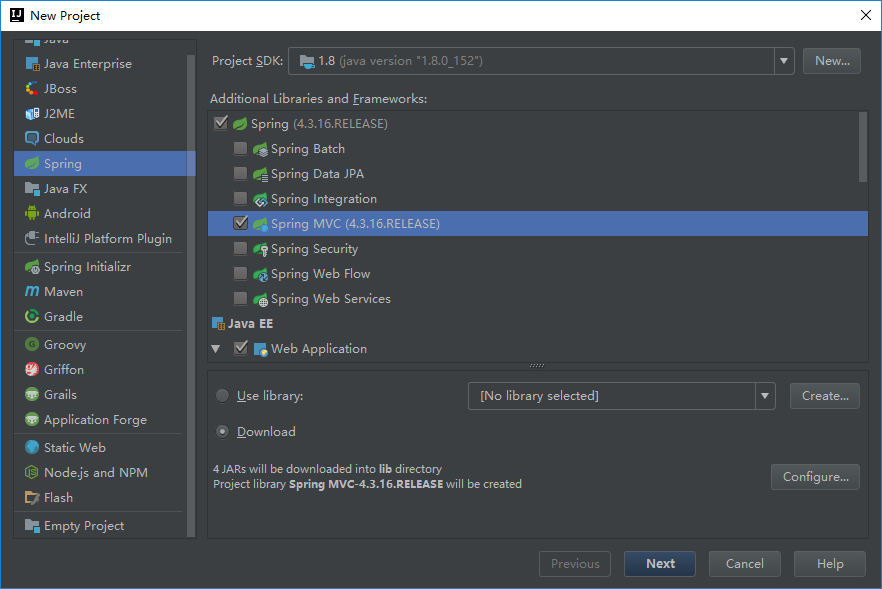
1. 实验基本原理
2. SpringMVC框架知识
3. Mybatis框架知识
4. Spring框架知识
5. 主要仪器设备及耗材

计算机、Idea、Tomcat9.0、JDK1.8

五、实验步骤（完整内容见光盘）

1.安装SpringMVC，创建SpringMVC项目并完善目录结构，所

以idea为例



选择Spring以及SpringMVC

完善项目目录结构，创建controller、dao、entity、service等目录

（1）src下的项目分层结构

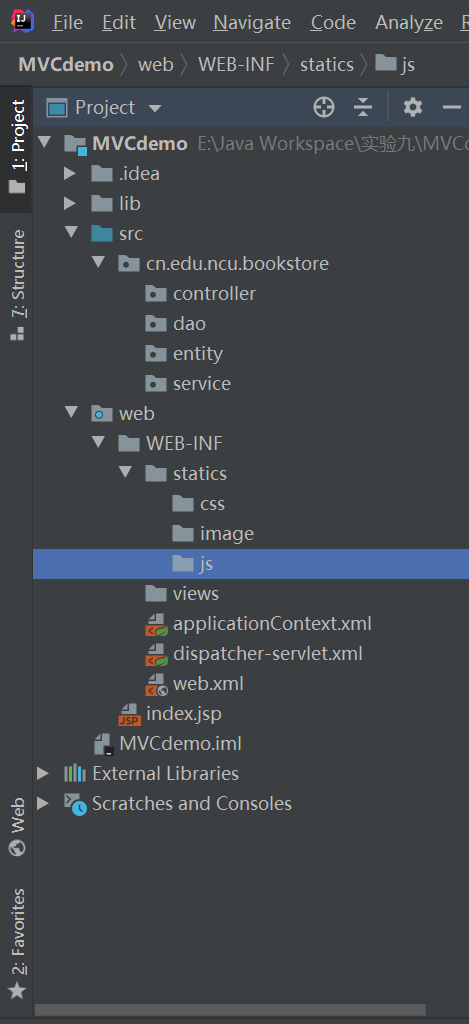
（2）WEB-INF中view的文件夹

（3）资源目录

（4）测试代码目录

（5）静态资源(css、js、image)

完善后的目录结构如下所示：



2.修改配置文件

（1）在web.xml中加载Spring和SpringMVC，web.xml内容如下所示：

1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"
5. version="4.0">
6. <context-param>
7. <param-name>contextConfigLocation</param-name>
8. <param-value>/WEB-INF/applicationContext.xml</param-value>
9. </context-param>
10. <listener>
11. <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
12. </listener>
13. <servlet>
14. <servlet-name>dispatcher</servlet-name>
15. <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
16. <load-on-startup>1</load-on-startup>
17. </servlet>
18. <servlet-mapping>
19. <servlet-name>dispatcher</servlet-name>
20. <url-pattern>/</url-pattern>
21. </servlet-mapping>
22. </web-app>

Springmvc 没有指定xml路径，默认会去查找xxx-servlet.xml的文件，如果需要指定可以添加<init-param>进行相应的指定。

同时可以设置多个dispathcher来处理不同url的请求，来区分不同的应用场景。

dispatcher-servlet.xml

添加依赖注入的注解方式（通过指定目录实现）

扩充注解驱动，可以将请求参数绑定到控制器参数

静态资源配置

显示层配置

dispatcher-servlet.xml中的内容如下所示：

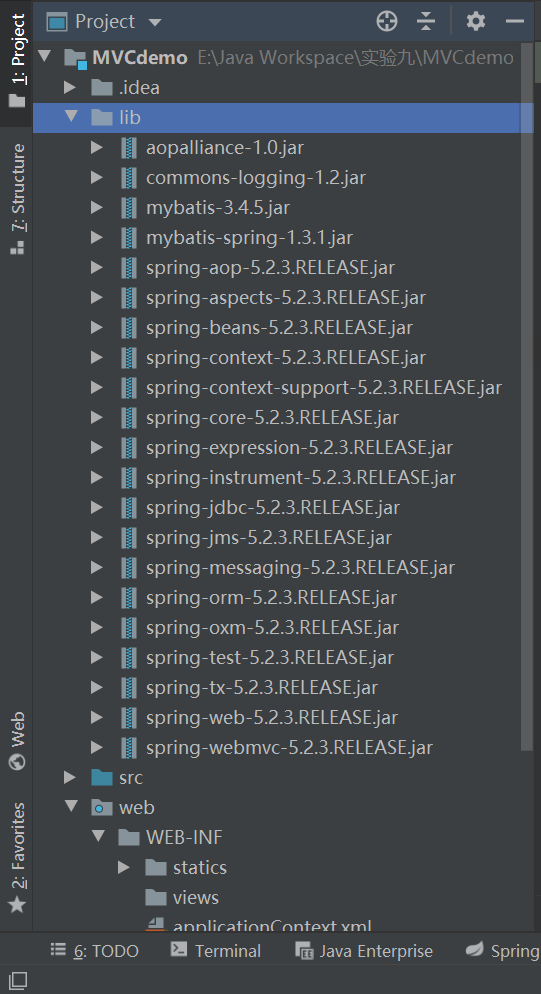
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:p="http://www.springframework.org/schema/p"
5. xmlns:tx="http://www.springframework.org/schema/tx"
6. xmlns:context="http://www.springframework.org/schema/context"
7. xmlns:aop="http://www.springframework.org/schema/aop"
8. xmlns:mvc="http://www.springframework.org/schema/mvc"
9. xsi:schemaLocation="http://www.springframework.org/schema/beans
10. http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
11. http://www.springframework.org/schema/mvc
12. http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd
13. http://www.springframework.org/schema/context
14. http://www.springframework.org/schema/context/spring-context-3.1.xsd
15. http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.1.xsd
16. http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.1.xsd">
17. *<!--RequestMappingHandlerAdapter-->*
18. <mvc:annotation-driven />
19. *<!-- 打开使用注解自动检测功能自动注册Bean，扫描@Controller -->*
20. <context:component-scan base-package="cn.edu.ncu.bookstore.controller">
21. </context:component-scan>
22. <mvc:resources mapping="/css/\*\*" location="/WEB-INF/statics/css/"/>
23. <mvc:resources mapping="/js/\*\*" location="/WEB-INF/statics/js/"/>
24. <mvc:resources mapping="/images/\*\*" location="/WEB-INF/statics/images/"/>
25. <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
26. <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
27. <property name="prefix" value="/WEB-INF/"></property>
28. <property name="suffix" value=".jsp"></property>
29. </bean>
30. </beans>

applicationContext.xml中的内容如下所示：

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd">
6. <context:component-scan base-package="cn.edu.ncu.bookstore">
7. <context:exclude-filter type="annotation"   expression="org.springframework.stereotype.Controller"/>
8. </context:component-scan>
9. </beans>

3. 配置MyBatis环境

（1）添加两个jar包mybatis.jar和mybatis-spring.jar，如下所示



（2）配置SqlSessionFactory

a. 整合后，不需要单独的mybatis配置文件，全部的配置内容都可以在spring的上下文当中进行。

b. myBatis框架主要是围绕SqlSessionFactory进行的

* + 定义一个configuration对象，其中包括数据源、mapper文件、事务和影响数据库行为属性设置。
  + 通过配置对象，创建一个SqlSessionFactoryBuilder对象
  + 通过SqlSessionFactoryBuilder 获得SqlSessionFactory实例
  + SqlSessionFactoryBuilder实例可以获得操作数据的SqlSession实例，通过该实例对数据库进行操作

修改dataSource中的driverClassName和url，建立相关的目录和文件，得到如下applicationContext.xml中的代码：

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd">
6. <context:component-scan base-package="cn.edu.ncu.bookstore">
7. <context:exclude-filter type="annotation"   expression="org.springframework.stereotype.Controller"/>
8. </context:component-scan>
9. <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
10. <property name="driverClassName" value="org.sqlite.JDBC" />
11. *<!-- 指定连接数据库的URL -->*
12. <property name="url" value="jdbc:sqlite:E:/Java Workspace/实验九/bookstore.db" />
13. </bean>
14. <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
15. <property name="dataSource" ref="dataSource"/>
16. <property name="mapperLocations" value="classpath:cn/edu/ncu/bookstore/mapper/\*.xml"/>
17. </bean>
18. <bean id="CategoryDao" class="org.mybatis.spring.mapper.MapperFactoryBean">
19. <property name="mapperInterface" value="cn.edu.ncu.bookstore.dao.CategoryDao"/>
20. <property name="sqlSessionFactory" ref="sqlSessionFactory"/>
21. </bean>
22. </beans>

设置为批量扫描方法：

在Spring的配置文件中增加如下bean即可实现批量扫描：

1. <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
2. <property name="basePackage" value="cn.edu.ncu.bookstore.dao"/>
3. <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
4. </bean>

4. Mapper（Dao）开发

添加Dao接口，使用myBatis不需要实现接口，需要编写相应的xml文件

1. @Repository
2. public interface CategoryDao {
3. List<Category> findAll();
4. }

在entity中创建相对应的实体类：

1. package cn.edu.ncu.bookstore.entity;
2. public class Category {
3. String id;
4. String name;
5. String description;
6. public String getId() {
7. return id;
8. }
9. public void setId(String id) {
10. this.id = id;
11. }
12. public String getName() {
13. return name;
14. }
15. public void setName(String name) {
16. this.name = name;
17. }
18. public String getDescription() {
19. return description;
20. }
21. public void setDescription(String description) {
22. this.description = description;
23. }
24. }

CategoryMapper.xml文件中的内容如下所示：

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE mapper
3. PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4. "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5. <mapper namespace="cn.edu.ncu.bookstore.dao.CategoryDao">
6. <resultMap type="cn.edu.ncu.bookstore.entity.Category" id="result\_map">
7. <id column="id" property="id"/>
8. <result column="name" property="name"/>
9. <result column="description" property="description"/>
10. </resultMap>
11. <select id="findAll" resultMap="result\_map">
12. select id,name,description
13. from category
14. </select>
15. </mapper>

编写xml文件中的sql语句时，应注意以下几点：

1、namespace要正确，否则无法对应到相关的mapper

2、resultMap为返回的对象，其中property的值需要和指定的entity一致

3、如需要传入变量则使用#{parm}的方式

SELECT id,name,description

FROM category

WHERE id = #{id}

5．Service开发

新建CategoryService类，将Mapper(CategoryDao）接口注入到由Spring管理的bean对象中

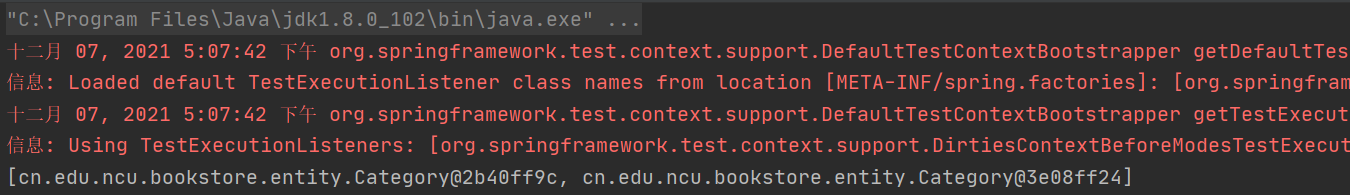
1. @Service
2. public class CategoryService {
3. @Autowired
4. private CategoryDao categoryDao;
5. public List<Category> findAll(){
6. return categoryDao.findAll();
7. }
8. }

6. Service单元测试

编写测试类如下所示：

1. @RunWith(SpringJUnit4ClassRunner.class)
2. @ContextConfiguration(locations = {"file:web/WEB-INF/applicationContext.xml"})
3. public class CategoryServiceTest {
4. @Autowired
5. private CategoryService service;
6. @Test
7. public void findAll(){
8. System.out.println(service.findAll().toString());
9. }
10. }

测试结果如下所示：



7. Controller开发

增加注解@Controller交由SpringMVC管理

@RequestMapping注解在类上，表明该路径下所有访问由此类处理

注解在方法上的和注解在类上的路径组合/category/view

注：return “main” 在前面的MVC配置中设置了前缀和后缀。

添加的控制器如下所示：

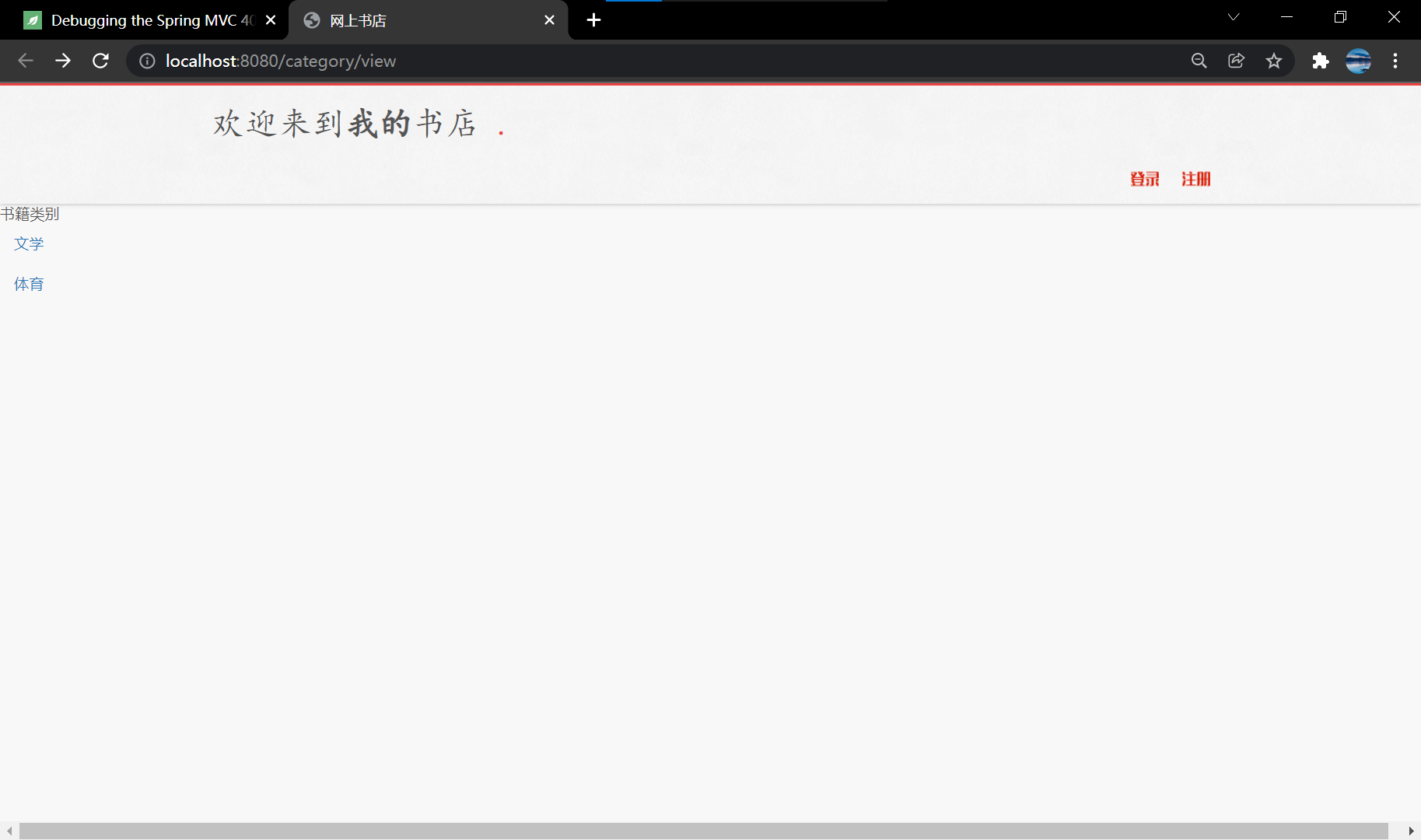
1. @Controller
2. @RequestMapping("/category")
3. public class CategoryController {
4. @Autowired
5. private CategoryService categoryService;
6. @RequestMapping(value = "/view")
7. public String viewCategory(Model model){
8. List<Category> list = categoryService.findAll();
9. model.addAttribute("category",list);
10. return "main";
11. }
12. }

8. view开发

修改（或新建）main.jsp，修改类别显示部分代码

使用JSTL处理，呈现数据显示。

如下：



main.jsp代码如下：

1. <%@ page import="cn.edu.ncu.bookstore.entity.Category" %>
2. <%@ page contentType="text/html;charset=UTF-8" language="java" %>
3. <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4. <%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
5. <html>
6. <head>
7. <link href="/css/bootstrap.min.css" rel="stylesheet"/>
8. <link href="/css/style.css" rel="stylesheet"/>
9. <title>网上书店</title>
10. </head>
11. <body>
12. <div class="modal fade" id="myModal">
13. <div class="modal-dialog">
14. <div class="modal-content">
15. <div class="modal-header"></div>
16. <div class="modal-body"></div>
17. <div class="modal-footer"></div>
18. </div>
19. </div>
20. </div>
21. <div class="header">
22. <div class="container">
23. <div class="row">
24. <div class="login span4">
25. <h1><a href=""> 欢迎来到<strong>我的</strong>书店</a>
26. <span class="red">.</span></h1>
27. </div>
28. <div class="links span8">
29. <a class="login" href="login.html" rel="tooltip" data-placement="bottom" data-toggle="modal" data-target="#myModal"></a>
30. <a class="register" href="" rel="tooltip" data\_placement="bottom"></a>
31. </div>
32. </div>
33. </div>
34. </div> <%--header--%>
35. <div class="row"> <%--下方左右div控制--%>
36. <div class="col-md-3"> <%--左侧菜单div控制--%>
37. <ul class="nav nav-list" id="list">
38. <li class="nav-header">书籍类别</li>
39. <c:forEach items="${category}" var="category">
40. <li>
41. <a href="#">${category.name}</a>
42. </li>
43. </c:forEach>
44. </ul>
45. </div><%--左侧菜单div控制--%>
46. <div class="col-md-9" id="book"><%--右侧书本div控制--%>
47. </div><%--右侧书本div控制--%>
48. </div><%--下方左右div控制--%>
49. <script src="/js/jquery.min.js" ></script>
50. <script src="/js/bootstrap.min.js"></script>
51. <script src="/js/commons.js"></script>
52. </body>
53. </html>

9. 修改Controller，使用modelandview

在控制器中添加如下函数：

1. @RequestMapping(value="/modelview")
2. public ModelAndView showCategory(ModelAndView model){
3. List<Category> list = categoryService.findAll();
4. model.setViewName("main");
5. model.addObject("category",list);
6. return model;
7. }

访问页面得到与上图一样的运行结果。

10. 使用参数的查询

1) 新建Book实体

代码如下所示：

1. package cn.edu.ncu.bookstore.entity;
2. public class Book {
3. private int id;
4. private String name;
5. private String author;
6. private double price;
7. private String image;
8. private String description;
9. private String category\_id;
10. public int getId() {
11. return id;
12. }
13. public void setId(int id) {
14. this.id = id;
15. }
16. public String getName() {
17. return name;
18. }
19. public void setName(String name) {
20. this.name = name;
21. }
22. public String getAuthor() {
23. return author;
24. }
25. public void setAuthor(String author) {
26. this.author = author;
27. }
28. public double getPrice() {
29. return price;
30. }
31. public void setPrice(double price) {
32. this.price = price;
33. }
34. public String getImage() {
35. return image;
36. }
37. public void setImage(String image) {
38. this.image = image;
39. }
40. public String getDescription() {
41. return description;
42. }
43. public void setDescription(String description) {
44. this.description = description;
45. }
46. public String getCategory\_id() {
47. return category\_id;
48. }
49. public void setCategory\_id(String category\_id) {
50. this.category\_id = category\_id;
51. }
52. }

2) 新建Mapper(BookDao的接口)，使用@Param注解

1. @Repository
2. public interface BookDao {
3. List<Book> findBookByCategoryId(@Param("id") String id);
4. }

3) 新建Mapper XML文件

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE mapper
3. PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4. "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5. <mapper namespace="cn.edu.ncu.bookstore.dao.BookDao">
6. <resultMap type="cn.edu.ncu.bookstore.entity.Book" id="result\_map">
7. <id column="id" property="id"/>
8. <result column="name" property="name"/>
9. <result column="author" property="author"/>
10. <result column="price" property="price"/>
11. <result column="image" property="image"/>
12. <result column="description" property="description"/>
13. <result column="category\_id" property="category\_id"/>
14. </resultMap>
15. <select id="findBookByCategoryId" resultMap="result\_map">
16. select id,name,author,price,image,description,category\_id
17. from book
18. where category\_id = #{id,jdbcType=VARCHAR}
19. </select>
20. </mapper>

4) 新建BookService

1. package cn.edu.ncu.bookstore.service;
2. import cn.edu.ncu.bookstore.dao.BookDao;
3. import cn.edu.ncu.bookstore.entity.Book;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import java.util.List;
6. @BookService
7. public class BookService {
8. @Autowired
9. private BookDao bookDao;
10. public List<Book> getBookByCategoryId(String id){
11. return bookDao.findBookByCategoryId(id);
12. }
13. }

5) 新建BookController

1. package cn.edu.ncu.bookstore.controller;
2. import cn.edu.ncu.bookstore.service.BookService;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.stereotype.Controller;
5. import org.springframework.ui.Model;
6. import org.springframework.web.bind.annotation.PathVariable;
7. import org.springframework.web.bind.annotation.RequestMapping;
8. import org.springframework.web.bind.annotation.RequestMethod;
9. import org.springframework.web.bind.annotation.RequestParam;
10. import javax.servlet.http.HttpServletRequest;
11. @Controller
12. @RequestMapping("/book")
13. public class BookController {
14. @Autowired
15. private BookService bookService;
16. @RequestMapping("")
17. public String showBookByCategoryId(@RequestParam("id")String id, Model model){
18. model.addAttribute("book",bookService.getBookByCategoryId(id));
19. return "main";
20. }
21. @RequestMapping(value="{id}",method = RequestMethod.GET)
22. public String getBookByCategoryId(@PathVariable("id")String id, Model model){
23. model.addAttribute("book",bookService.getBookByCategoryId(id));
24. return "main";
25. }
26. @RequestMapping(value="view")
27. public String findBookByCategoryId(HttpServletRequest request){
28. String id = request.getParameter("id");
29. request.setAttribute("book",bookService.getBookByCategoryId(id));
30. return "main";
31. }
32. }

前面两个是不同路径风格的处理方式

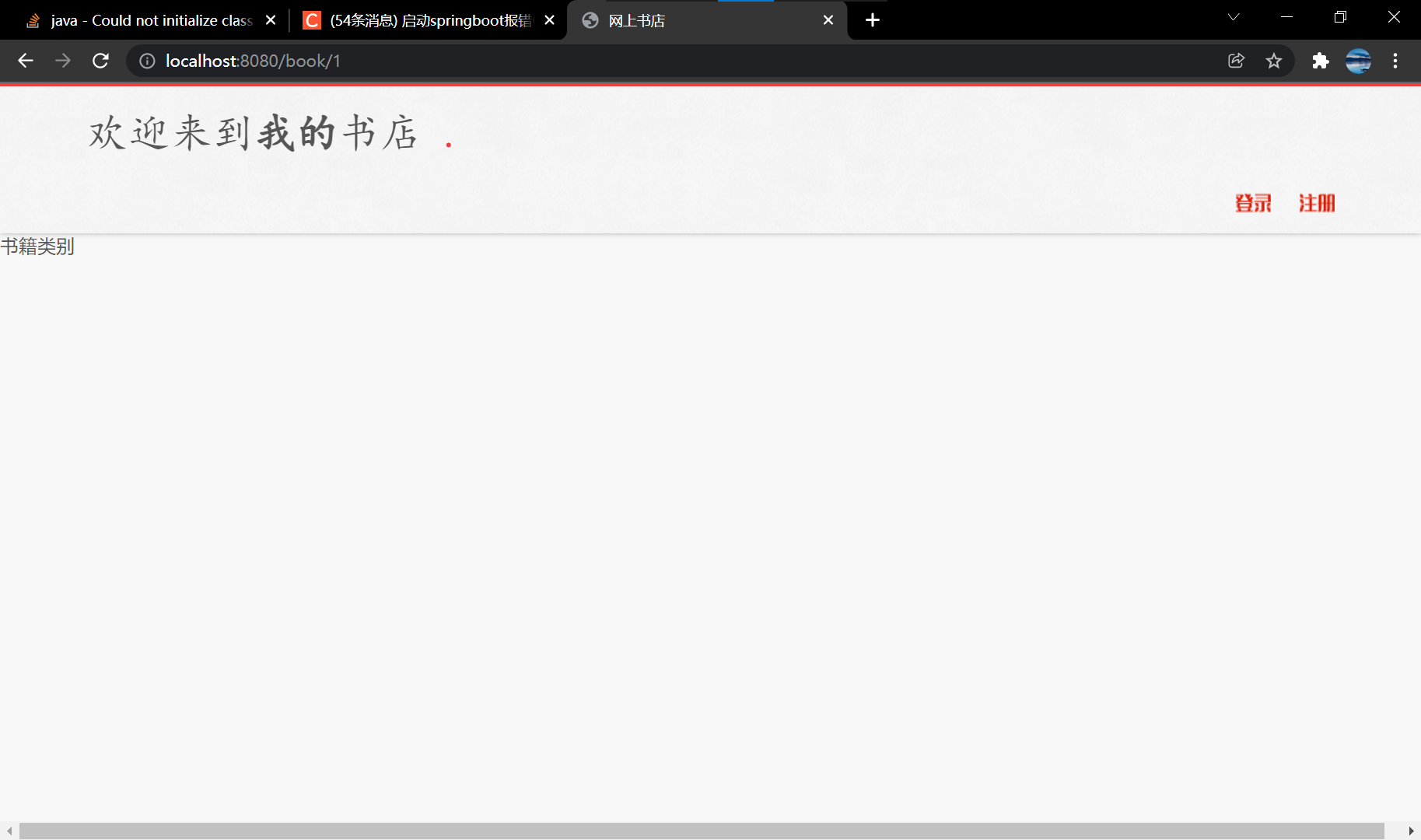
最后一个是通过传统的ServletRequest方式获取传入值。

http://localhost:8080/book?id=1

http://localhost:8080/book/1

http://localhost:8080/book/view?id=1

访问结果如下（三个地址效果一致）：



11. 使用json格式传递数据

使用@ ResponseBody处理数据

作用：

该注解用于将Controller的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的body数据区。

使用时机：

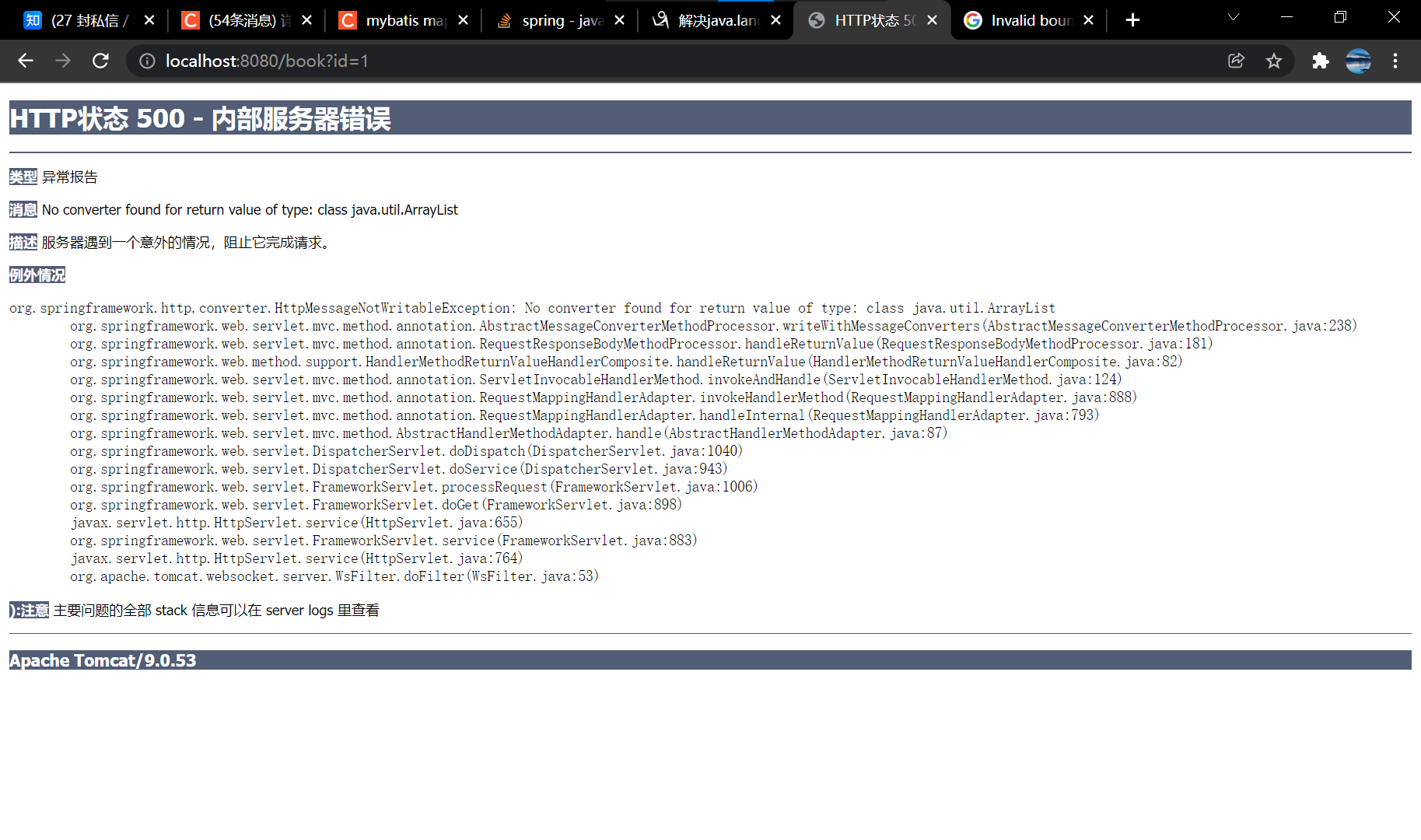
返回的数据不是html标签的页面，而是其他某种格式的数据时（如json、xml等）使用

注：HttpMessageConverter 早期的SpringMVC需要通过配置SpringMVC的xml文件进行相应的映射

本例中的SpringMVC可以自动转JSON格式，无需配置，只需导入SpringMVC支持的json工具包

完成后访问相应路径。

编写完成之后，访问出现以下错误：



经过资料查询得知，是由于没有导入jackson包，需要导入该包将对象转化为json数据返回

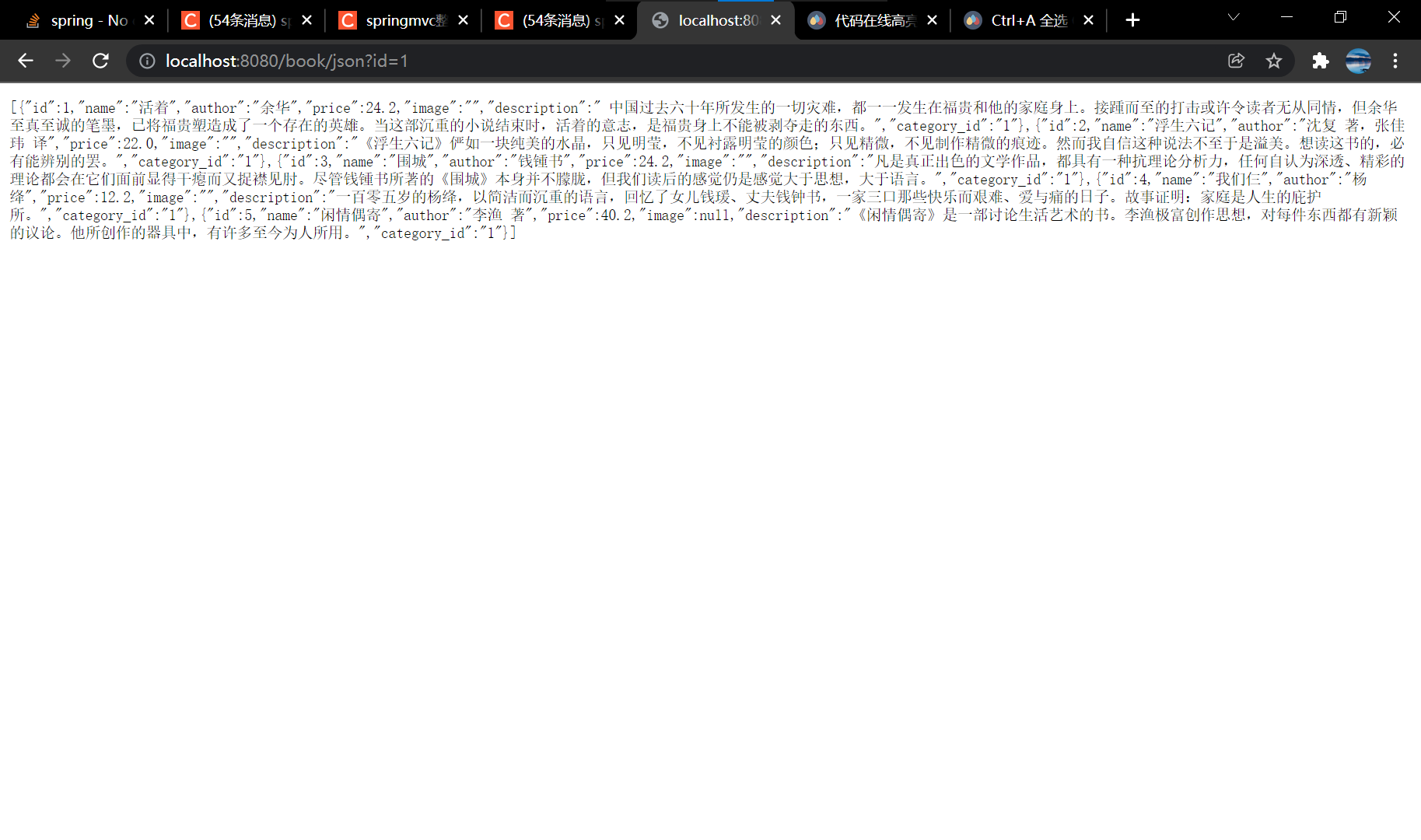
在Spring容器中配置如下bean

1. <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping" />
2. <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">
3. <property name="messageConverters">
4. <list>
5. <bean class="org.springframework.http.converter.StringHttpMessageConverter">
6. <property name="supportedMediaTypes">
7. <list>
8. <value>text/html; charset=UTF-8</value>
9. <value>application/json;charset=UTF-8</value>
10. </list>
11. </property>
12. </bean>
13. <bean class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">
14. <property name="supportedMediaTypes">
15. <list>
16. <value>text/html; charset=UTF-8</value>
17. <value>application/json;charset=UTF-8</value>
18. </list>
19. </property>
20. </bean>
21. </list>
22. </property>
23. </bean>

在控制器中添加如下方法：

1. @RequestMapping("json")
2. @ResponseBody
3. public List<Book> jsonBookByCategoryId(@RequestParam("id")String id, Model model){
4. return bookService.getBookByCategoryId(id);
5. }

直接访问结果如下所示：



六、思考讨论题或体会或对改进实验的建议

1. 修改实验7和实验8中的项目使用springMVC框架。

在上述项目的基础上进行修改：

将BookController中的代码修改成如下所示：

1. package cn.edu.ncu.bookstore.controller;
2. import cn.edu.ncu.bookstore.entity.Book;
3. import cn.edu.ncu.bookstore.service.BookService;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import org.springframework.stereotype.Controller;
6. import org.springframework.ui.Model;
7. import org.springframework.web.bind.annotation.\*;
8. import javax.servlet.http.HttpServletRequest;
9. import java.util.List;
10. @Controller
11. @RequestMapping("/book")
12. public class BookController {
13. @Autowired
14. private BookService bookService;
15. @RequestMapping("")
16. public String showBookByCategoryId(){
17. return "main";
18. }
19. @RequestMapping("json")
20. @ResponseBody
21. public List<Book> jsonBookByCategoryId(@RequestParam("id")String id, Model model){
22. return bookService.getBookByCategoryId(id);
23. }
24. }

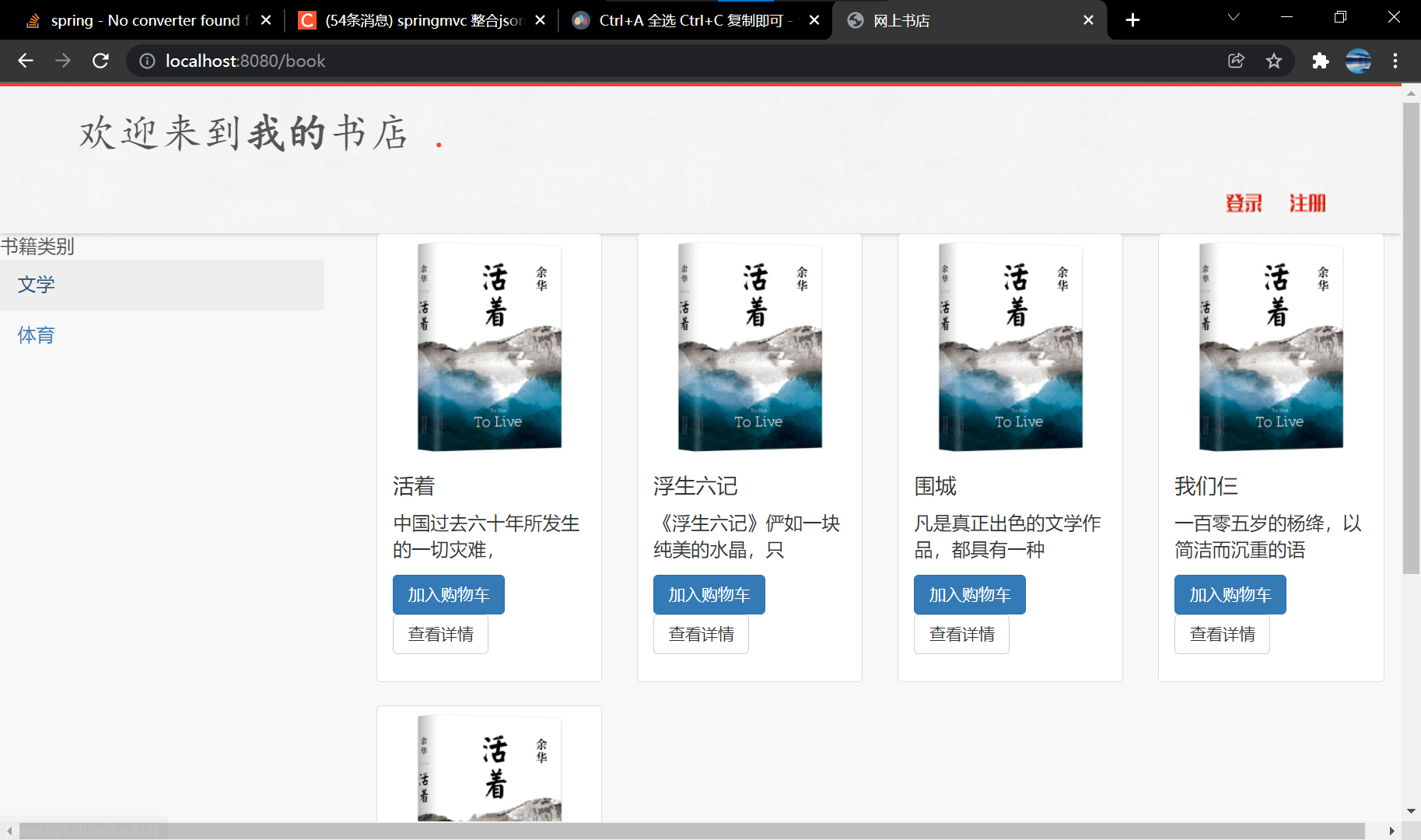
将main.jsp中的内容修改成如下所示：

1. <%@ page import="java.util.List" %>
2. <%@ page contentType="text/html;charset=UTF-8" language="java" %>
3. <html>
4. <head>
5. <link href="css/bootstrap.min.css" rel="stylesheet"/>
6. <link href="css/style.css" rel="stylesheet"/>
7. <title>网上书店</title>
8. </head>
9. <body onload="initAJAX();showCategory()">
10. <div class="modal fade" id="myModal">
11. <div class="modal-dialog">
12. <div class="modal-content">
13. <div class="modal-header"></div>
14. <div class="modal-body"></div>
15. <div class="modal-footer"></div>
16. </div>
17. </div>
18. </div>
19. <div class="header">
20. <div class="container">
21. <div class="row">
22. <div class="login span4">
23. <h1><a href=""> 欢迎来到<strong>我的</strong>书店</a>
24. <span class="red">.</span></h1>
25. </div>
26. <div class="links span8">
27. <a class="login" href="login.html" rel="tooltip" data-placement="bottom" data-toggle="modal" data-target="#myModal"></a>
28. <a class="register" href="" rel="tooltip" data\_placement="bottom"></a>
29. </div>
30. </div>
31. </div>
32. </div> <%--header--%>
33. <div class="row"> <%--下方左右div控制--%>
34. <div class="col-md-3"> <%--左侧菜单div控制--%>
35. <ul class="nav nav-list" id="categorylist">
36. </ul>
37. </div><%--左侧菜单div控制--%>
38. <div class="col-md-9" id="book"><%--右侧书本div控制--%>
39. </div><%--右侧书本div控制--%>
40. </div><%--下方左右div控制--%>
41. <script src="js/jquery.min.js" ></script>
42. <script src="js/bootstrap.min.js"></script>
43. <script src="js/commons.js"></script>
44. <script language="JavaScript">
45. function showCategory() {
46. xmlHttp.open("GET","/category/view",true);
47. xmlHttp.onreadystatechange = function () {
48. if(xmlHttp.readyState==4) {
49. var data = xmlHttp.responseText;
50. var obj = JSON.parse(data);
51. var listHtml = "<li class=\"nav-header\">书籍类别</li>\n" ;
52. for(var i in obj){
53. listHtml += "<li>\n" +
54. " <a href=\"javascript:showBook("+ obj[i].id + ")\">"+obj[i].name+"</a>\n" +
55. "           </li>"
56. }
57. document.getElementById("categorylist").innerHTML=listHtml;
58. }
59. }
60. xmlHttp.send();
61. }
62. function showBook(categoryID) {
63. xmlHttp.open("GET", "book/json?id="+categoryID, true);
64. xmlHttp.onreadystatechange = function () {
65. if (xmlHttp.readyState == 4) {
66. var data = xmlHttp.responseText;
67. var obj = JSON.parse(data);
68. var listbook = '';
69. for(var i in obj){
70. var bookname = obj[i].name;
71. var desc = obj[i].description;
72. if(desc.length > 20) {
73. desc = desc.substring(0,17)
74. }
75. listbook += `<div class="col-sm-9 col-md-3"><div class="thumbnail" ><img src="images/book.jpg"><div class="caption">  <h4>`
76. + bookname +
77. `</h4><p>`+desc+
78. `</p><p><a href="#" class="btn btn-primary" role="button">加入购物车</a> <a href="#" class="btn btn-default" role="button">查看详情`+
79. `</a></p></div></div> </div>`;
80. }
81. document.getElementById("book").innerHTML = listbook;
82. }
83. }
84. xmlHttp.send();
85. }
86. </script>
87. </body>
88. </html>

当地址栏访问/book时，后台将转发main.jsp页面，然后再main.jsp中使用ajax向后台请求数据，访问/book/json返回json格式书籍信息，访问/category/view返回json格式的书目类别信息，前端成功读取数据后再将其渲染出来。

main.jsp页面只需要修改ajax请求的地址即可，由于请求的数据都是同意以json格式返回的，所以前端不会被后端的修改所影响，修改请求的url即可。

修改之后运行结果如下：



七、参考资料

1.JavaWeb程序设计