

目 录

目 录.....	I
第一章 简介.....	2
1.1 项目内容.....	2
1.2 项目目标.....	2
第二章 功能需求分析.....	2
2.1 xxx 功能.....	2
2.2 xxx 功能.....	2
2.3 xxx 功能.....	3
第三章 系统结构设计.....	6
3.1 系统模块结构图.....	6
3.2 xxx 模块.....	8
3.3 xxx 模块.....	10
3.4 xxx 模块.....	12
第四章 系统实现.....	20
5.1 系统实现介绍.....	20
5.2 系统实现的不足.....	26
第五章 总结与展望.....	27

第一章 简介

1.1 项目内容

【做什么系统，为什么要做】

一个简单的网上书城系统,用户能够在系统实现基本的交互.该系统能够有效的将前后端的知识相结合以达到锻炼的目的.

1.2 项目目标

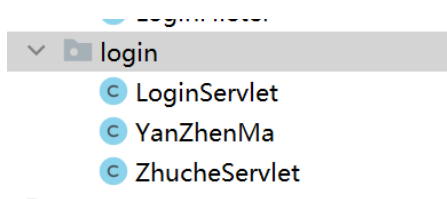
【项目要实现的目标】

书城中的信息全部从后端数据库中获取,同时能够实现用户书城账号的注册与登录,用户能够将自己喜爱的书籍放入购物车中进行购买.用户也能够进入后端系统对数据库进行增删改查的操作.前端页面中要实现分页以及搜索的功能.为了实现增删改查,需要文件或图片的上传功能.整个系统应该采用更符合趋势的前后端分离模式进行开发,运用 ajax 技术,实现基于 mvc 设计模式的软件架构.

第二章 功能需求分析

2.1 登录注册功能

用户需要能够注册并登录书城账号,登录时需要提供验证码识别功能.同时为使得登录功能的存在有意义,服务器端提供识别是否为登录用户的安全监测 Filter 来避免绕过登录模块直接访问的非正常访问.



2.2 操作数据库功能

用户登录成功后,能够进入操作数据库模块,对数据库进行增删改查等一系列修改.

- ▼ book
 - AddServlet
 - DatabaseReader
 - Dataitem
 - DeleteServlet
 - exit
 - xiugaiservlet
 - ...

2.3 Filter 监测过滤功能

过滤器能够避免绕过登录模块直接访问的非正常访问

- ▼ filter
 - CorsFilter
 - LoginFilter

2.4 文件或图片的上传功能

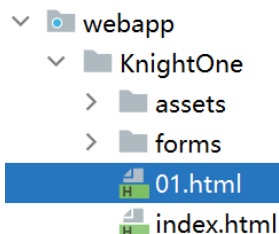
此功能是实现了对数据库增加与修改的必要功能.因此包含在对数据库的增加与修改功能中

1 个用法 DEZREMNACUI

```
private String saveImageToServer(byte[] imageBytes) throws IOException {  
    // 获取服务器上的存储目录，你可以根据需要修改  
    String uploadPath = getServletContext().getRealPath(s: "uploads");  
  
    // 确保目录存在  
    File uploadDir = new File(uploadPath);  
    if (!uploadDir.exists()) {  
        uploadDir.mkdir();  
    }  
  
    // 生成唯一的文件名  
    String fileName = System.currentTimeMillis() + ".jpg";  
    String filePath = uploadPath + File.separator + fileName;  
  
    // 写入文件到服务器文件系统  
    Files.write(new File(filePath).toPath(), imageBytes);  
  
    // 返回文件路径  
    return fileName;  
}
```

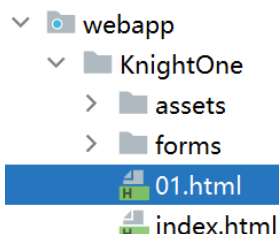
2.5 购物车功能

通过 `HttpSession` 实现,用户能够在书城挑选自己喜爱的书籍,被挑选的书籍将会在购物车中显示.



2.6 记录用户访问次数功能

通过 `HttpSession`,用户访问书城页面



2.7 前后端分离模式

运用 Ajax 异步技术实现前后端分离,通过引入 `axios` 的库,通过 `axios` 在前端向后端发送请求并获取后端的响应.

```
<script>
|   axios.get("http://10.69.219.123:8080/JAVA_FINAL_WORK/DatabaseReader", {
|       params: {
|           searchType: "book",
|           filter: "",
|           page: 1 // 使用当前输入的页码
|       }
|   })
|   .then(function (response) {
|       for (var i = 0; i < document.getElementsByClassName("img-fluid").length; i++) {
|           document.getElementsByClassName("img-fluid")[i].src = response.data[i].image;
|           document.getElementsByClassName("details-link")[i].href = "portfolio-details.html?book=" + encodeURIComponent(response.data[i].content) +
|               "&price=" + encodeURIComponent(response.data[i].price) +
|               "&author=" + encodeURIComponent(response.data[i].author) +
|               "&nation=" + encodeURIComponent(response.data[i].nation) +
|               "&image=" + encodeURIComponent(response.data[i].image);
|       }
|       var element = document.querySelector('li.filter-active[data-filter="*"]');
|       window.addEventListener('load', element.click());
|   })
|   </script>
```

2.8 美化客户端界面

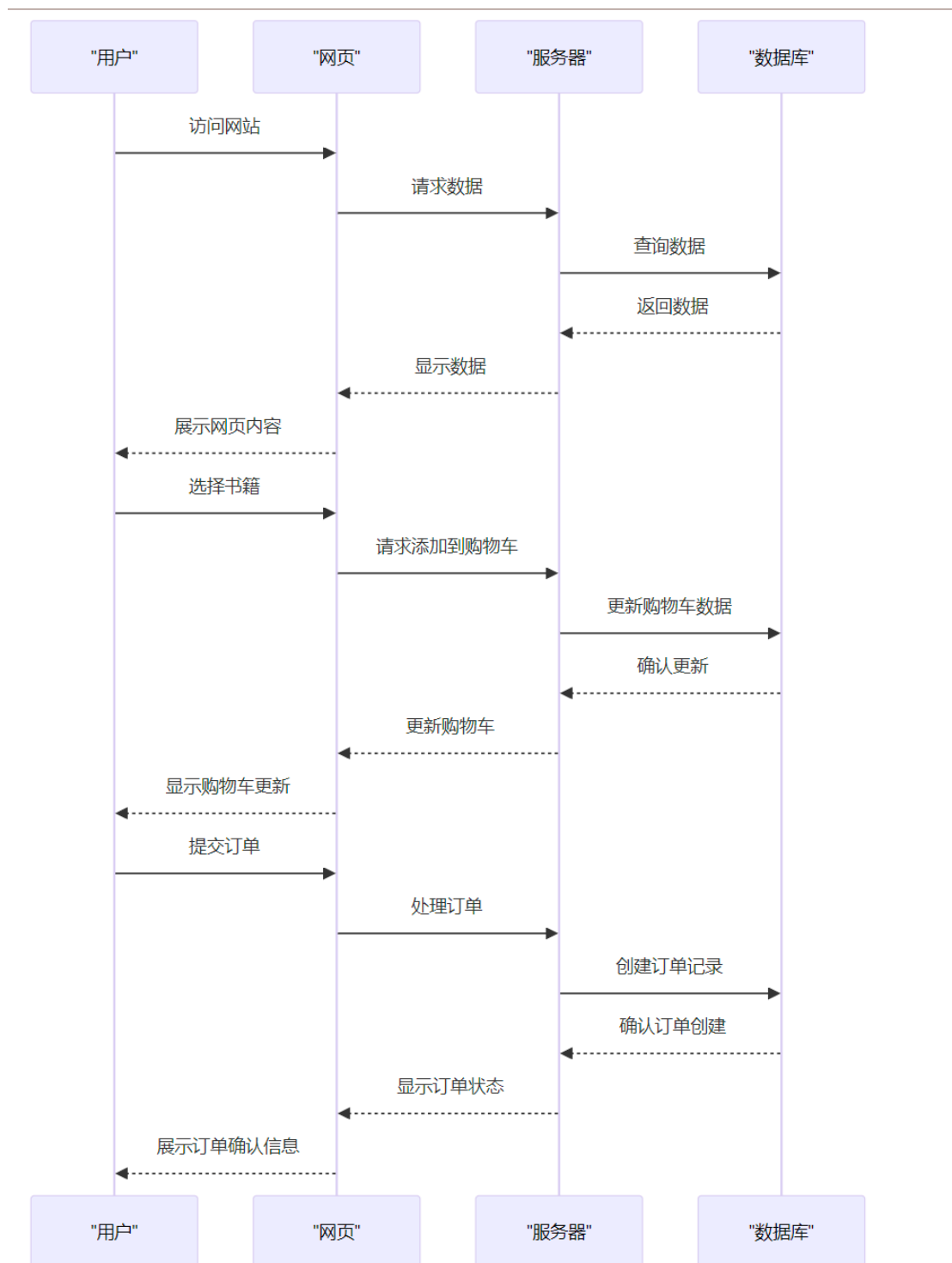
通过引入 bootstrap 的库来实现前端界面的美化,同时通过前面的 ajax 来保证功能的完整性.

.....

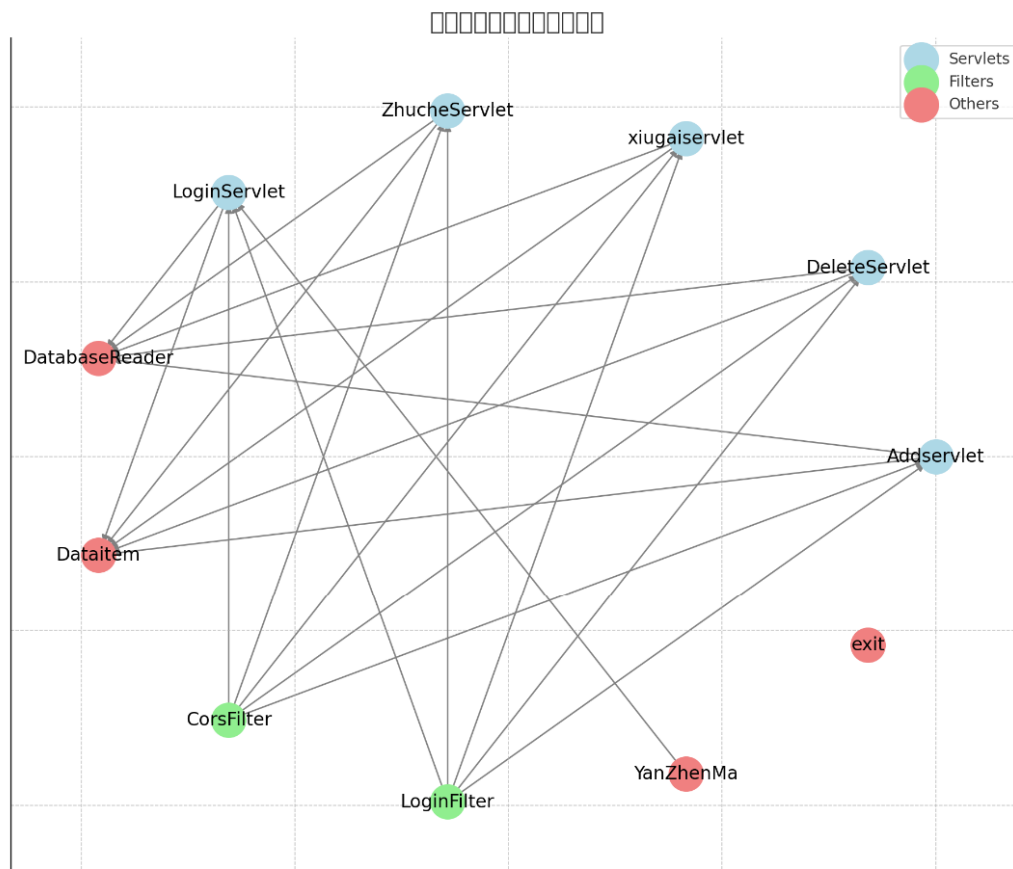
第三章 系统结构设计

3.1 系统模块结构图

【给出一个系统模块的组合起来的结构框图】

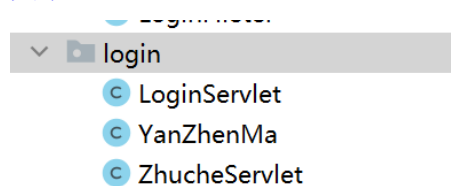


后端部分各模块关系:



3.2 登录注册模块

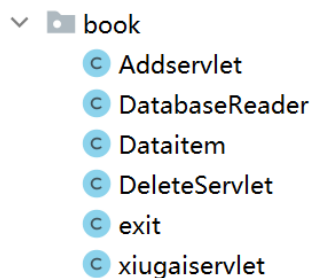
【根据功能分析得出的结论，划分模块，分别设计各个模块结构，理清各个模块之间的交互关系】



- **LoginServlet.java (登录 Servlet)**
- 功能
- **处理登录请求:** 通过 `doPost` 方法接收 `username` 和 `password`。
- **Cookie 验证:** 检查是否存在与提交的用户名和密码相匹配的 Cookie。
- **验证码验证:** 从请求中获取用户输入的验证码 (`captchaInput`)，并与会话中存储的验证码 (`captcha`) 进行比对。
- **登录逻辑:**
 - 如果用户名和密码的 Cookie 存在且验证码正确，用户登录成功。
 - 如果任一验证失败，显示错误信息并在 5 秒后重定向到登录界面。
- 关键代码
- `Cookie[] cookies = req.getCookies();` 获取请求中的所有 Cookie。
- `String userInput = req.getParameter("captchaInput");` 获取用户输入的验证码。

- **HttpSession session = req.getSession();**: 创建或获取当前的会话。
- **resp.sendRedirect("LoginSuccess.html");**: 重定向到登录成功页面。
- **YanZhenMa.java (验证码 Servlet)**
 - 功能
 - **生成验证码:** 使用 **CaptchaUtil** 创建一个圆形验证码。
 - **存储和发送验证码:**
 - 将生成的验证码存储在会话 (**session**) 中。
 - 将验证码图片作为响应发送给客户端。
 - 关键代码
 - **CircleCaptcha captcha = CaptchaUtil.createCircleCaptcha(100, 40, 4, 20);**: 创建一个新的圆形验证码。
 - **req.getSession().setAttribute("captcha", captcha.getCode());**: 将验证码文本存储在会话中。
 - **ImageIO.write(image, "png", resp.getOutputStream());**: 将验证码图片写入响应。
- **ZhucheServlet.java (注册 Servlet)**
 - 功能
 - **处理注册请求:** 通过 **doGet** 方法接收 **username**、**password** 和 **tel**。
 - **创建和存储 Cookie:**
 - 为用户名、密码和电话号码创建 **Cookie**。
 - 设置 **Cookie** 的最大存活时间为一天。
 - 将这些 **Cookie** 添加到响应中。
 - **重定向:** 完成注册后重定向到首页 (**index.html**)。
 - 关键代码
 - **Cookie username1 = new Cookie("username", username);**: 为用户名创建一个新的 **Cookie**。
 - **username1.setMaxAge((24*60*60));**: 设置 **Cookie** 的有效期为一天。
 - **resp.addCookie(username1);**: 将 **Cookie** 添加到响应中。
- **总结**
- **登录流程** 首先验证用户名和密码是否存储在 **Cookie** 中, 然后验证用户输入的验证码是否与会话中的验证码匹配。
- **注册流程** 接收用户的信息并将其存储在 **Cookie** 中, 然后重定向用户到首页

3.3 数据库操作模块



- **AddServlet.java（添加数据 Servlet）**
- 功能
- **接收书籍信息：** 通过 **doPost** 方法从请求中获取书籍信息，包括标题、作者、国籍、内容和价格。
- **处理图像文件：** 接收并处理上传的图像文件。
- **存储图像：** 将图像文件存储到服务器文件系统，并返回文件路径。
- **数据库操作：** 调用 **Dataitem** 类的 **addDataToDatabase** 方法将书籍信息和图像路径添加到数据库。
- 关键代码
- **Part imagePart = req.getPart("image");**：获取上传的图像部分。
- **Dataitem.addDataToDatabase(...);**：添加书籍信息到数据库。
- **resp.getWriter().write("{\"success\": true}");**：返回成功响应。

- **DatabaseReader.java（数据库读取 Servlet）**
- 功能
- **读取数据库信息：** 根据请求中的参数（如搜索类型和过滤条件），从数据库中读取数据。
- **数据转换为 JSON：** 将数据库中的数据转换为 JSON 格式，并包含图像的 URL。
- 关键代码
- **List<Dataitem> dataList = Dataitem.fetchDataFromDatabase(...);**：从数据库获取数据。
- **convertDataitemToJSON(dataItem);**：将 **Dataitem** 转换为 JSON 字符串。

- **Dataitem.java（数据模型和数据库操作类）**
- 功能
- **表示数据模型：** 包含书籍的属性，如标题、作者、国籍等。
- **数据库操作：** 提供方法从数据库中获取、添加、更新和删除数据。
- 关键代码
- **Dataitem.fetchDataFromDatabase(...);**：从数据库获取数据。
- **Dataitem.addDataToDatabase(...);**：添加数据到数据库。
- **Dataitem.updateDataInDatabase(...);**：更新数据库中的数据。
- **Dataitem.deleteDataFromDatabase(...);**：从数据库删除数据。

- **DeleteServlet.java（删除数据 Servlet）**

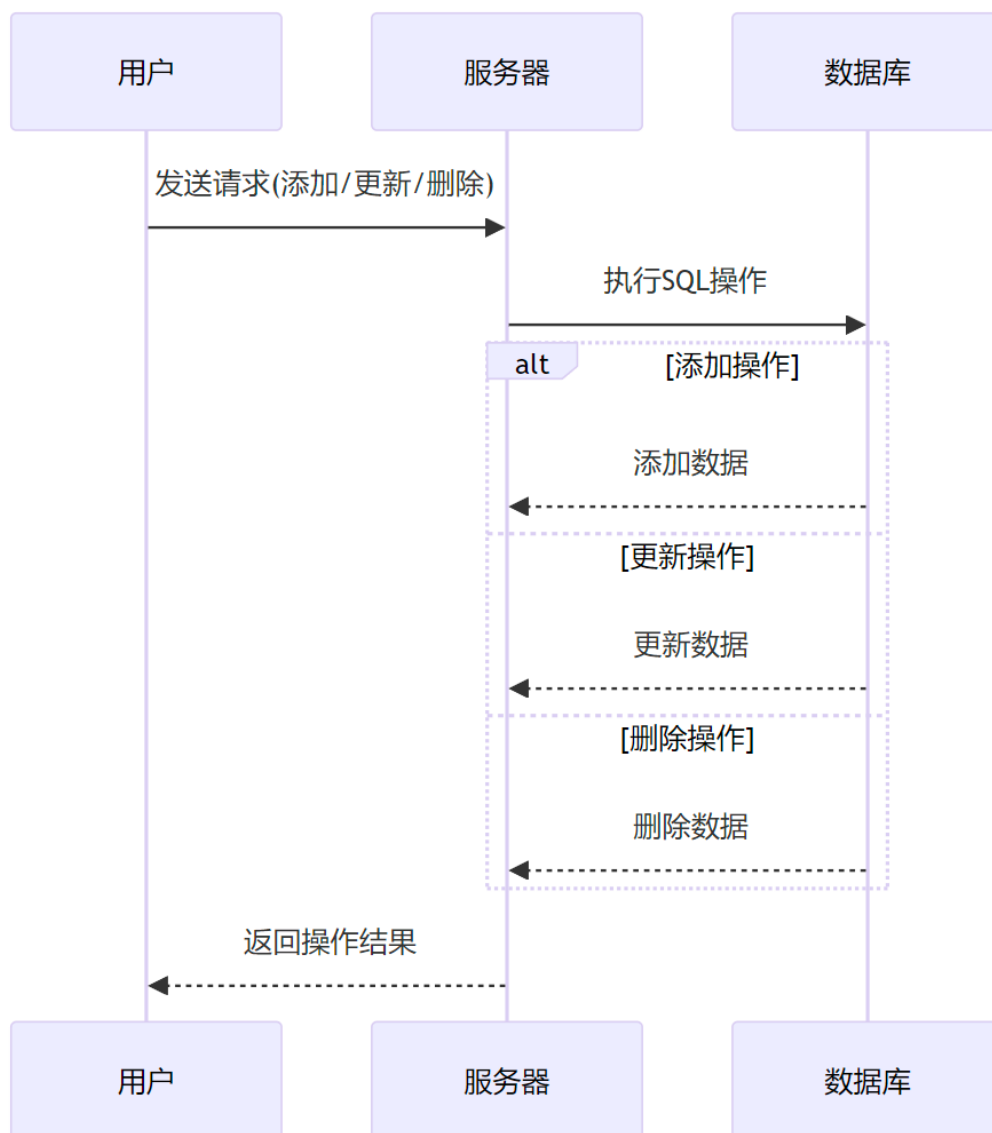
- 功能
- **处理删除请求：** 接收需要删除的书籍信息，并调用 **Dataitem** 类的方法从数据库中删除相应的记录。
- **删除相关图像文件：** 从文件系统中删除与书籍相关联的图像文件。
- 关键代码
- **Dataitem.deleteDataFromDatabase(...);** 删除数据库中的数据。
- **Files.delete(...);** 删除文件系统中的图像文件。

- **xiugaiservlet.java（修改数据 Servlet）**

- 功能
- **接收和处理修改请求：** 接收需要修改的书籍信息和新图像文件。
- **更新数据库：** 调用 **Dataitem** 类的方法更新数据库中的记录。
- **处理新图像文件：** 存储新的图像文件并更新数据库中的图像路径。
- 关键代码
- **Dataitem.updateDataInDatabase(...);** 更新数据库中的数据。
- **saveImageToServer(imageBytes);** 存储新的图像到服务器。

- **总结**

这些组件协同工作，提供了一个完整的数据操作流程，包括添加、读取、修改和删除数据库中的书籍记录。这些操作通过 **Dataitem** 类与数据库直接交互，而 **Addservlet**、**DatabaseReader**、**DeleteServlet** 和 **xiugaiservlet** 类则处理来自用户的 HTTP 请求，并调用 **Dataitem** 类的方法进行数据库操作。这样的架构分离了数据处理逻辑和 Web 请求处理逻辑，使得系统更加模块化和易于维护。



3.4 Filter 监测过滤模块

filter

- CorsFilter
- LoginFileter

- **CorsFilter.java**（跨域资源共享过滤器）
- 功能
- **设置跨域资源共享（CORS）策略：** 允许来自所有来源的请求访问，支持常见的 HTTP 方法，允许特定的头信息。
- 关键代码
- **httpResponse.setHeader("Access-Control-Allow-Origin", "*");** 允许所有来源的跨域请求。
- **httpResponse.setHeader("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE, OPTIONS");** 定义允许的 HTTP 方法。

- `httpResponse.setHeader("Access-Control-Allow-Headers", "Content-Type");`: 允许 **Content-Type** 头信息。
- 作用
- 这个过滤器为所有请求设置了 CORS 策略，它是为了解决前端应用在不同域名或端口下访问后端 API 时可能遇到的跨域问题。
- **LoginFileter.java**（登录状态过滤器）
- 功能
- **保护特定资源**: 该过滤器用于确保只有已登录的用户才能访问 **LoginSuccess.html** 页面。
- **检查会话状态**: 检查用户是否有有效的会话和登录状态。
- 关键代码
- `HttpSession session = httpRequest.getSession(false);`: 获取当前请求的会话，但不创建新会话。
- `boolean loggedIn = session != null && session.getAttribute("loginStatus") != null;`: 检查会话是否存在且登录状态不为空。
- `httpServletResponse.sendRedirect("index.html");`: 如果用户未登录，重定向到登录页面。
- 作用
- 该过滤器确保只有登录用户可以访问受保护的资源（在这个例子中是 **LoginSuccess.html**），如果用户没有登录，他们将被重定向到登录页。这是一种基本的访问控制，用于增强应用的安全性。
- 总结
- **CorsFilter** 主要用于处理 Web 应用中的跨域请求，确保前端应用可以从不同的源安全地访问后端服务。
- **LoginFileter** 用于保护特定的 Web 资源，确保只有验证过的用户可以访问这些资源，从而提高应用的安全性。

这两个过滤器在 Web 应用中扮演着重要的角色，**CorsFilter** 处理跨域问题，而 **LoginFileter** 管理访问控制。

3.5 文件或图片的上传功能

AddServlet

AddServlet.java 类是一个用于处理书籍信息添加的 Servlet，它包括处理图像文件上传的功能。

- 文件和图片上传功能
- 1. 接收图像文件
- **获取图像部分**: 使用 `req.getPart("image")` 从请求中获取名为 "image" 的部分，这部分是上传的图像文件。

```
Part imagePart = req.getPart("image");
```

- 2. 读取图像数据
- **读取图像流：** 从 `imagePart` 获取 `InputStream`，然后读取所有字节到一个字节数组 (`imageBytes`)。这个字节数组包含了图像文件的数据。

```
InputStream imageStream = imagePart.getInputStream();
byte[] imageBytes = imageStream.readAllBytes();
```

- 3. 存储图像到服务器
- **确定存储路径：** 使用 `getServletContext().getRealPath("uploads")` 获取服务器上用于存储上传文件的目录的实际路径。
- **创建目录：** 如果该目录不存在，则创建它。
- **生成文件名和路径：** 为了避免文件名冲突，使用当前时间戳生成唯一的文件名，并构建文件的完整路径。
- **写入文件系统：** 使用 `Files.write` 将图像字节写入构建的文件路径。



```
String uploadPath = getServletContext().getRealPath("uploads");
File uploadDir = new File(uploadPath);
if (!uploadDir.exists()) {
    uploadDir.mkdir();
}
String fileName = System.currentTimeMillis() + ".jpg";
String filePath = uploadPath + File.separator + fileName;
Files.write(new File(filePath).toPath(), imageBytes);
```

- 4. 返回文件路径
- **返回文件名：** 函数 `saveImageToServer` 返回存储的图像文件的文件名，该文件名可以用于在数据库中与其他书籍信息一起存储或以其他方式使用。

总结

这个上传功能的实现是典型的文件上传处理流程，在 Web 应用中非常常见。它接收上传的文件，读取文件数据，将数据存储在服务器的文件系统中，并返回文件路径或文件名。这个流程对于处理用户上传的图像或其他文件类型是非常重要和实用的。

3.6 购物车功能

 01.html
 index.html

- **功能概述**
- **展示购物车内容：** 购物车模块负责显示用户添加到购物车中的书籍信息。
- **移除功能：** 允许用户从购物车中移除特定的书籍。
- **本地存储：** 购物车的数据存储在 `sessionStorage` 中，这意味着购物车的内容仅在当前会话期间有效。
- **关键实现细节**

- 1. 显示购物车内容
- **获取存储的书籍数据：** 从 **sessionStorage** 中读取存储的书籍信息（JSON 格式）。
- **动态生成 HTML 内容：** 根据获取的书籍数据动态生成 HTML 元素，并插入到 **bookTable** 元素中。

```
var books = JSON.parse(sessionStorage.getItem('books')) || [];
```

```
var bookTable = document.getElementById("bookTable");
```

```
// 动态添加书籍信息到 DOM 中
```

- 2. 移除功能
- **移除书籍：** 用户可以通过点击“删除”按钮来移除特定的书籍。这是通过 **removeBookFromCart** 函数实现的，它从 **sessionStorage** 中移除相应的书籍并更新显示。

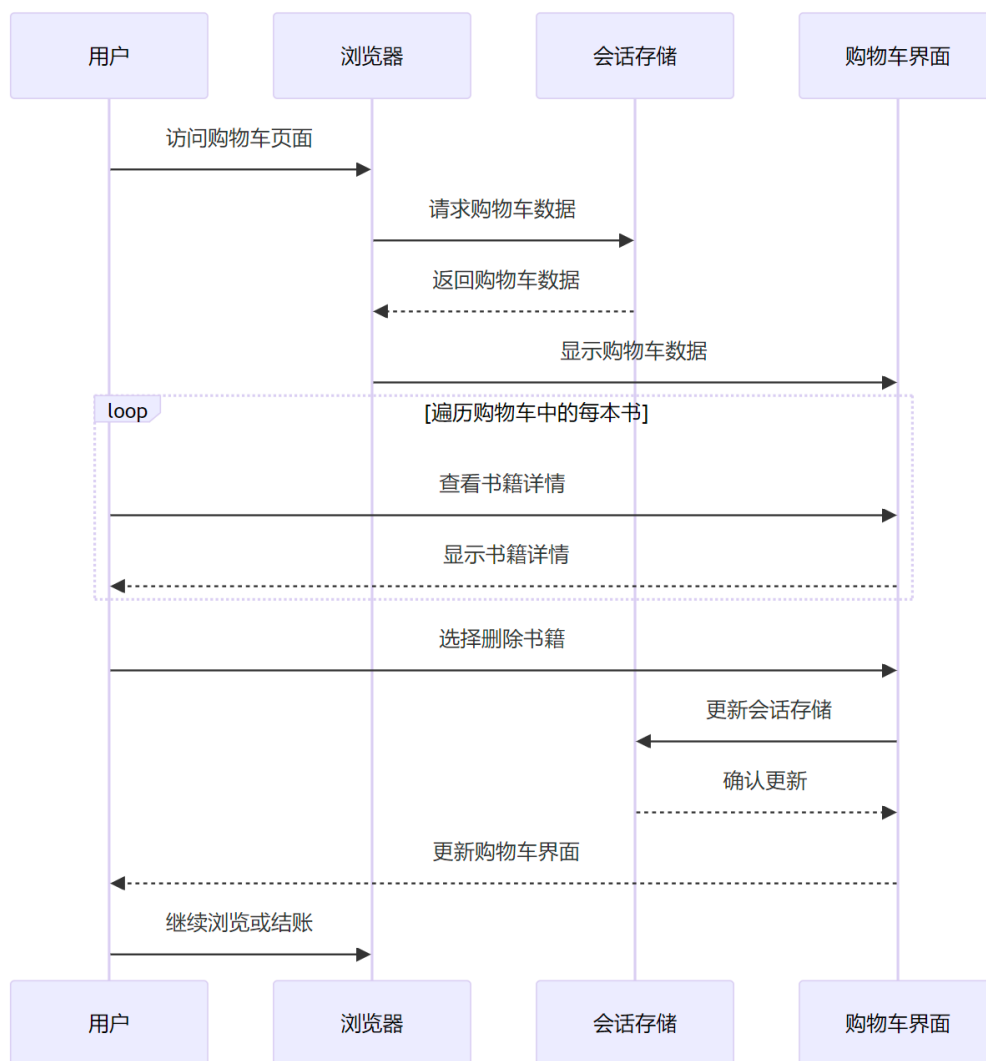
```
function removeBookFromCart(index) {  
    // 移除指定索引的书籍  
}
```

- 3. 页面加载时展示内容
- **加载事件处理：** 当页面加载完成后，调用 **displayBooks** 函数来显示存储在 **sessionStorage** 中的购物车内容。

```
document.addEventListener('DOMContentLoaded', displayBooks);
```

总结

这个购物车模块允许用户在浏览网站时添加和移除书籍。使用 **sessionStorage** 进行存储意味着购物车的内容只在当前浏览器会话期间保持，一旦会话结束（如关闭浏览器标签或窗口），购物车内容将被清除。这个模块利用 JavaScript 动态地更新页面内容，提供了一个交互性良好的用户体验。



3.7 记录用户访问次数功能

index.html

- 功能概述
- 记录访问次数：每当用户访问页面时，页面上的 JavaScript 代码会更新存储在 `sessionStorage` 中的访问次数。
- 关键实现细节
- 1. 获取和设置 `sessionStorage`
- 读取存储的访问次数：代码首先尝试从 `sessionStorage` 中获取键为 `'visitCount'` 的值，这个值代表了用户在当前会话中访问页面的次数。

```
var visitCount = sessionStorage.getItem('visitCount');
```

更新访问次数： 如果该值存在，则将其转换为数字并加一；如果不存在，则初始化为 1。

```
if (visitCount) {
```



```

    visitCount = Number(visitCount) + 1;
  } else {
    visitCount = 1;
  }

```

保存更新后的访问次数： 然后，更新后的访问次数被保存回 **sessionStorage**。

```
sessionStorage.setItem('visitCount', visitCount);
```

- 2. 显示访问次数
- **在页面上显示访问次数：** 更新后的访问次数被显示在页面上的指定元素中。


```
document.getElementById('visitCountDisplay').textContent = '访问次数: ' + visitCount;
```

总结

这个功能利用了 **sessionStorage** 来存储在用户当前会话期间的页面访问次数。与 **localStorage** 不同，**sessionStorage** 中的数据在页面会话结束时（如关闭浏览器标签页）会被清除。这意味着每次用户在新会话中访问页面时，计数会重新开始。

通过 JavaScript 在每次页面加载时更新和显示访问次数，这个功能提供了一个简单的方式来跟踪和展示用户在当前会话中对页面的访问频率。

3.8 前后端分离模式

 index.html

• 前后端分离模式概述

前后端分离是一种网站架构模式，其中前端（通常是用户界面）和后端（服务器和数据库）是独立的，它们通过 API（应用程序编程接口）进行通信。这种模式提供了更好的灵活性和可维护性，同时也支持更复杂的用户交互和数据处理。

- **关键实现细节**
 - 1. 前端设计
 - **静态资源：** 前端包括 HTML、CSS 和 JavaScript，用于创建用户界面和交互。
 - **动态内容加载：** JavaScript（通常结合 AJAX 技术或使用库如 Axios）用于向后端请求数据，并动态地将这些数据呈现在页面上。
 - 2. 后端接口
 - **数据 API：** 后端提供 API 接口，前端通过这些 API 获取或发送数据。这通常通过 HTTP 请求实现。
 - 3. 分离的优势
 - **灵活性：** 前端和后端可以独立开发和部署，只要保持 API 接口的兼容性。
 - **易于维护和扩展：** 更容易对前端或后端进行更新和扩展，而不影响系统的其他部分。
 - **支持多平台：** 相同的后端可以支持网站、移动应用等多个前端平台。

前端使用 Axios 库从后端获取数据：

```

axios.get("http://10.69.219.123:8080/JAVA_FINAL_WORK/DatabaseReader", {
  params: {
    searchType: "book",

```

```

        filter: "",
        page: 1
    }
})


```

- 这段代码向后端的 **DatabaseReader** 接口发送 GET 请求，获取书籍数据。
- 获取到的数据随后被用来动态生成 HTML 内容，展示在网页上。

总结

前后端分离模式通过将用户界面（前端）和数据处理（后端）的职责分开，提高了应用程序的灵活性和可维护性。这种模式特别适合大型、复杂的应用程序，其中前端和后端的开发可能由不同的团队进行，并且需要频繁地独立更新和扩展。在现代 Web 开发中，前后端分离已成为一种流行和有效的架构模式。

3.9 客户端界面的搜索与分页功能

 index.html

1. 搜索功能

2. 实现步骤

3. **用户输入：** 用户在搜索框（**id="search"**）中输入搜索关键字。
4. **事件触发：** 当用户在搜索框中按下键盘按键时，触发 **searchTable** 函数。
5. **发送请求：** 使用 Axios 库，函数 **searchTable** 向后端的 **DatabaseReader** 发送 GET 请求，包含用户输入的搜索关键字作为参数。
6. **处理响应：** 一旦收到响应，页面上的内容会根据返回的数据动态更新。
7. 关键代码

```

function searchTable() {
    axios.get("http://10.69.219.123:8080/JAVA_FINAL_WORK/DatabaseReader", {
        params: {
            searchType: "book",
            filter: document.getElementById('search').value,
            page: document.getElementById('currentPage').value
        }
    })
    .then(function (response) {
        // 更新页面内容的代码
    });
}

```

1. 分页功能

2. 实现步骤

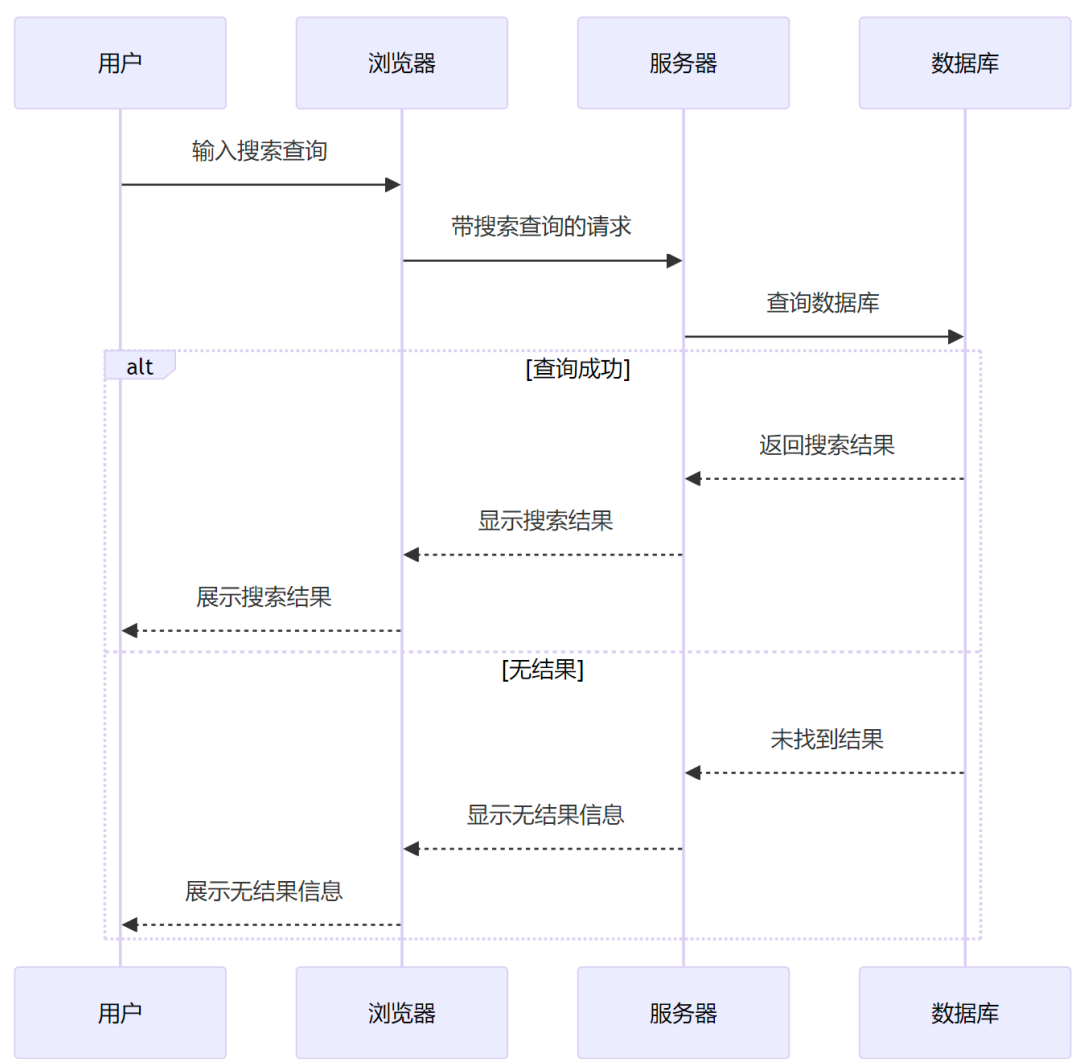
3. **分页控件：** 用户界面包含用于分页的输入框（**id="currentPage"**）和按钮（上一页/下一页）。
4. **事件触发：** 当用户点击分页按钮或更改页码时，触发 **changePage** 函数。
5. **更新页码：** 根据用户的操作（前进或后退），更新当前页码。

6. **发送请求：** 使用更新后的页码重新调用 **searchTable** 函数，向后端请求新一页的数据。
7. **处理响应：** 页面上的内容会根据新一页的数据进行更新。
8. 关键代码

```
function changePage(offset) {  
    currentPage += offset;  
    if (currentPage < 1) {  
        currentPage = 1;  
    }  
    document.getElementById('currentPage').value = currentPage;  
    searchTable();  
}
```

- **综合分析**
- **动态内容加载：** 页面不会在每次搜索或分页操作时进行完全的重新加载，而是通过 JavaScript 动态更新内容，提高用户体验。
- **与后端交互：** 所有数据的获取都是通过与后端 API 的交互完成的。这样的设计使得前端与后端能够灵活、独立地进行开发和维护。
- **用户界面友好性：** 分页和搜索功能的实现使得用户能够方便地浏览和查找大量数据，提高了网站的可用性。

这种实现方式是现代 Web 应用中常见的模式，特别是在涉及大量动态数据的情况下。通过前端 JavaScript 与后端 API 的结合使用，可以创建响应快速、用户友好的 Web 应用。



第四章 系统实现

4.1 系统实现介绍

【本章只是对系统实现内容，实现程度，实现效果的介绍，不是满篇贴代码，主要针对系统的关键功能，关键技术，技术难点实现结果的说明，可以粘贴部分系统程序的界面做说明和展示】

1.登录与注册

欢迎登录

还没有账号? [请注册](#)

用户名:

密码:

验证码: 

[登录](#)

欢迎注册

已有账号? [登录](#)

用户名

密码

手机号

[提交](#)

- 登录流程 首先验证用户名和密码是否存储在 Cookie 中, 然后验证用户输入的验证码是否与会话中的验证码匹配。
- 注册流程 接收用户的信息并将其存储在 Cookie 中, 然后重定向用户到首页

2.操作数据库

书籍信息表格(控制数据库,不要轻易删除)						
搜索: <input type="text"/>						
书名	内容	价格	作者	国籍	图片	操作
铁壁之围 一战中的德国和奥匈帝国	第一次世界大战开始时, 同盟国认为己方有望轻松获胜。但是, 随着德国对法作战计划战败, 奥匈帝国的军队蒙受沉重损失, 速胜的希望很快落空了。对于同盟国来说, 这场战争现在成了一个规模空前的包围战。英国实施了无情的封锁包围。德国和奥匈帝国无法获得军需物资和食品。他们的士兵面对着装备更好的敌军, 两国平民面临着饥饿。征服和劫掠、陆上攻势、潜艇战都无法对抗或打破封锁包围。同盟国陷入了协约国构筑的不断收紧的铁壁之围当中。本书是	101.0	亚历山大·沃森	英国		修改 删除

战争与和平（套装全4册）	以四大贵族的经历为主线，完整勾勒出1805年至1820年间俄国的重大历史事件。 在战争与和平两种生活的交织中，透过一群年轻的心灵，展现了一个惊心动魄的时代……	89.9	列夫·托尔斯泰	俄国		修改 删除
普希金诗集	普希金被誉为“俄罗斯诗歌的太阳”。这些优秀诗作歌颂了爱情和友谊，呼唤自由和公正，思索人生和社会，为读者广为吟诵，经久不衰	11.7	普希金	俄国		修改 删除

当前页: 1 [上一页](#) [下一页](#)

[添加](#) [退出系统](#)

修改书籍信息

新书名:

新内容:

第一次世界大战开始时，同盟国认为己方有望轻松获胜，但是，随着德国对法作战计划战败，奥匈帝国的军队受沉重损失，连胜利的希望很快落空了。对于同盟国来说，这场战争现在成了一个被粮食包围的包围战。英国实施了无休止的封锁包围，德国和奥匈帝国无法获得军需物资和食品，他们的士兵面对有武备更好的敌军，两国平民面临着饥饿、征服和劫掠、陆上攻

新价格:

新作者:

新国籍:

新图片:
 未选择任何文件

添加书籍信息

书名:

内容:

价格:

作者:

国籍:

选择图片:
 未选择任何文件

这些组件协同工作，提供了一个完整的数据操作流程，包括添加、读取、修改和删除数据库中的书籍记录。这些操作通过 **Dataitem** 类与数据库直接交互，而 **Addservlet**、**DatabaseReader**、**DeleteServlet** 和 **xiugaiservlet** 类则处理来自用户的 HTTP 请求，并调用 **Dataitem** 类的方法进行数据库操作。这样的架构分离了数据处理逻辑和 Web 请求处理逻辑，使得系统更加模块化和易于维护。

3. 文件或图片的上传功能

添加书籍信息

书名:

内容:

价格:

作者:

国籍:

选择图片:


选择文件

未选择任何文件

添加

这个上传功能的实现是典型的文件上传处理流程，在 Web 应用中非常常见。它接收上传的文件，读取文件数据，将数据存储在服务器的文件系统中，并返回文件路径或文件名。

4.购物车功能




书名: 铁壁之围 一战中的德国和奥匈帝国

作者: 亚历山大·沃森

国籍: 英国

价格: 101.0

Buy Now




书名: 尼采诗集

作者: 尼采

国籍: 德国

价格: 41.7

Buy Now



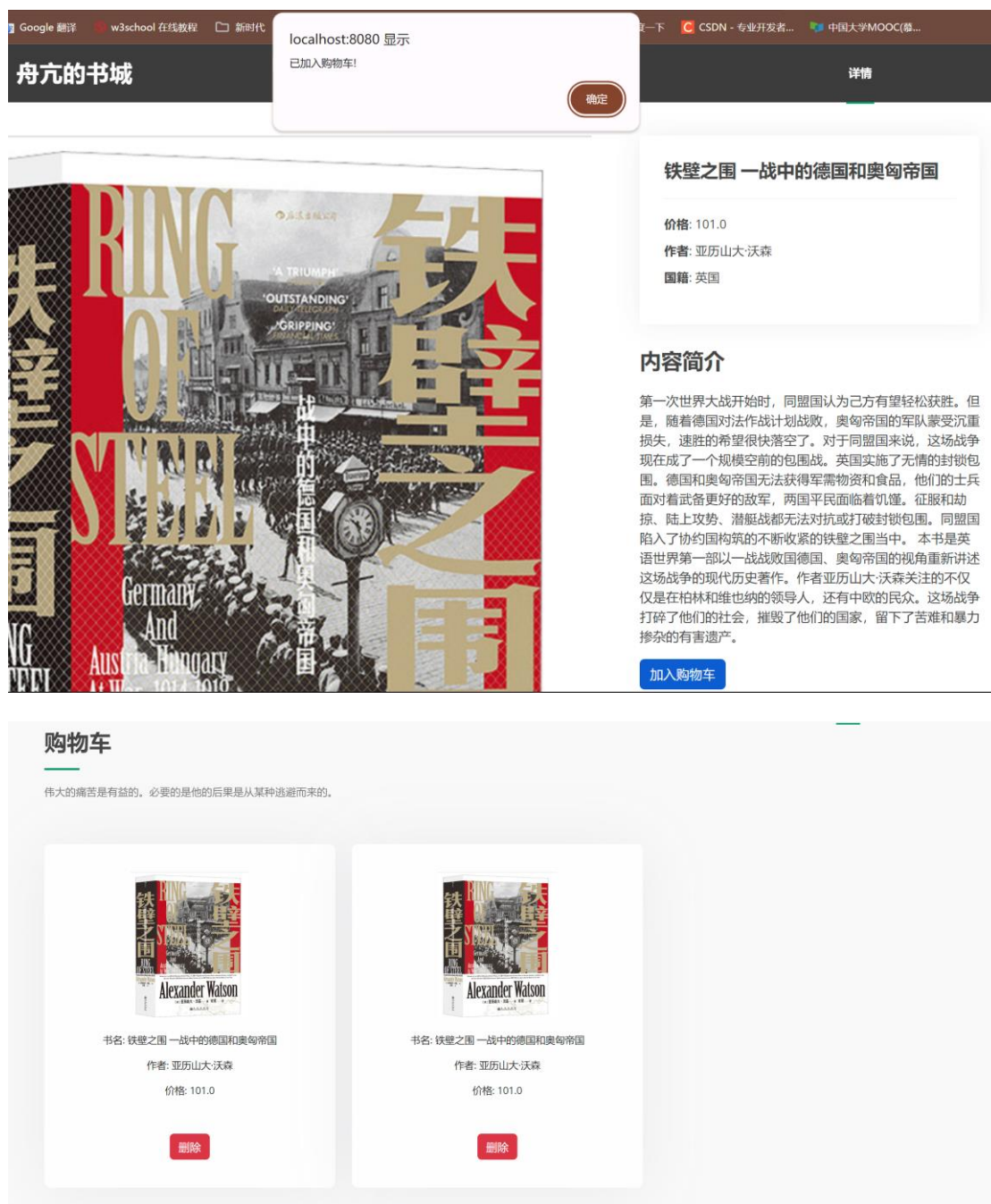
书名: 新民说 宋神宗与王安石: 变法时代 精装上下册

作者: 吴钩

国籍: 中国

价格: 99.0

Buy Now



这个购物车模块允许用户在浏览网站时添加和移除书籍。使用 **sessionStorage** 进行存储意味着购物车的内容只在当前浏览器会话期间保持，一旦会话结束（如关闭浏览器标签或窗口），购物车内容将被清除。这个模块利用 **JavaScript** 动态地更新页面内容，提供了一个交互性良好的用户体验。

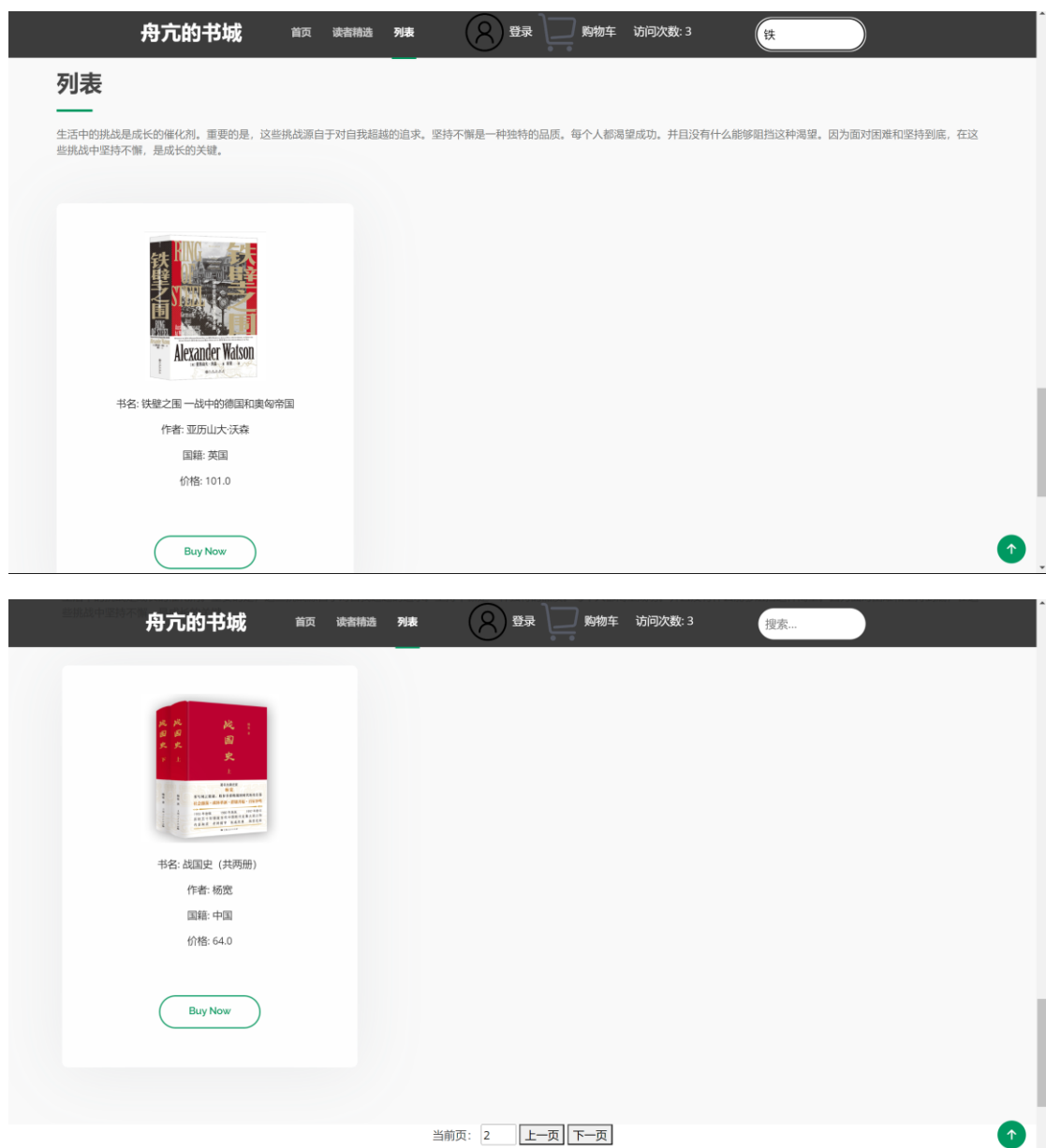
5. 记录用户访问次数功能



这个功能利用了 **sessionStorage** 来存储在用户当前会话期间的页面访问次数。与 **localStorage** 不同，**sessionStorage** 中的数据在页面会话结束时（如关闭浏览器标签页）会被清除。这意味着每次用户在新会话中访问页面时，计数会重新开始。

通过 JavaScript 在每次页面加载时更新和显示访问次数，这个功能提供了一个简单的方式来跟踪和展示用户在当前会话中对页面的访问频率。

6.客户端的搜索与分页功能



- 动态内容加载：页面不会在每次搜索或分页操作时进行完全的重新加载，而是通过 JavaScript 动态更新内容，提高用户体验。
- 与后端交互：所有数据的获取都是通过与后端 API 的交互完成的。这样的设计使得前端与后端能够灵活、独立地进行开发和维护。
- 用户界面友好性：分页和搜索功能的实现使得用户能够方便地浏览和查找大量数据，提高了网站的可用性。

这种实现方式是现代 Web 应用中常见的模式，特别是在涉及大量动态数据的情况下。通过前端 JavaScript 与后端 API 的结合使用，可以创建响应快速、用户友好的 Web 应用。

4.2 系统实现的不足

在用户的登录与注册中,我将用户名和密码存储在 Cookie 中，这不是最安全的做法。通常，密码不应直接存储，而应使用加密或哈希处理。此外，存储敏感信息在 Cookie 中可能会增加安全风险。

购物车使用 `sessionStorage` 进行存储意味着购物车的内容只在当前浏览器会话期间保持，一旦会话结束（如关闭浏览器标签或窗口），购物车内容将被清除。

用户的信息与购物车中的内容没有与数据库相连因此无法实现信息的长久存储

。。。。

第五章 总结与展望

JavaWeb 学习总结

技术栈掌握

Servlet 技术： 通过使用 Servlets 处理 HTTP 请求与响应，展示了对 JavaWeb 核心组件的理解。

CSS 和 HTML： 结合 CSS 和 HTML，展示了在 Web 应用中创建美观内容的能力。

前后端分离： 的项目采用前后端分离的方式，前端使用 HTML、CSS 和 JavaScript，后端则侧重于 Java Servlets，展现了现代 Web 应用的开发模式。

数据库操作： 通过 JDBC 连接数据库，执行查询和更新操作，显示了对数据库基础的掌握。

项目实践

登录注册功能

操作数据库功能

Filter 监测过滤功能

文件或图片的上传功能

购物车功能

记录用户访问次数功能

前后端分离模式

美化客户端界面, 并且具有搜索与分页

设计与实现

模块化设计： 代码展示了模块化设计思想，每个功能如登录、注册、图片处等都被细分为不同模块。

数据驱动的界面： 利用 Ajax 技术和后端数据交互，实现了数据驱动的 Web 界面，提高了用户体验。

展望

技术深化

框架学习： 考虑学习如 Spring 和 Hibernate 等流行框架，以提升开发效率和项目质量。

安全加强： 深入学习 Web 安全知识，如 SQL 注入防御、跨站脚本攻击(XSS)防御等。

新技术探索

1. **微服务架构：** 了解并实践微服务架构，以适应大型复杂应用的开发需求。
2. **容器化与云部署：** 学习 Docker、Kubernetes 等容器技术，并尝试云端部署和管理应用。

项目实战

1. **完整项目开发：** 参与或独立开发完整项目，从需求分析到部署上线，以获得全面的实战经验。
2. **开源贡献：** 参与开源项目，提升编码能力，了解开源社区的协作和开发流程。

持续学习

1. **关注技术动态：** 持续关注 Java 和 Web 技术的最新动态和趋势。
2. **深入理论与实践：** 平衡理论学习与项目实践，不断提升解决复杂问题的能力。