

# 目 录

目 录.....	I
第一章 绪论.....	2
1.1 系统的目的和意义.....	2
第二章 需求分析.....	2
2.1 引言.....	2
2.2 需求分析.....	
2.3 游客需求分析.....	2
2.4 用户需求分析.....	3
2.5 管理员需求分析.....	3
第三章 系统结构分析与设计.....	3
3.1 引言.....	3
3.2 系统模块结构图.....	6
3.3 模块一.....	6
3.4 模块二.....	6
3.5 模块三.....	
第四章 关键技术研究.....	7
4.1 实现技术路线.....	7
4.2 关键技术研究.....	7
4.2.1 关键技术一.....	7
4.2.1 关键技术二.....	8
4.2.1 关键技术三.....	10
4.2.1 关键技术四.....	12
第五章 系统实现.....	16
5.1 系统实现介绍.....	16
5.2 系统实现的不足.....	19

# 第一章 绪论

## 1.1 系统的目的和意义

书籍是人类进步的阶梯。当今社会，每个人都需要从书籍中获取大量的知识，而如今传统互联网与移动互联网快速且强烈地改变了人们的生活。任何行业几乎都不能脱离互联网而独立存在，它也给人们的生活带来的极大的便利。对于传统的书店而言，现代繁忙又快节奏的生活使得人们在逛实体书店这项活动上花费的时间越来越少，随之网上书城应运而生。

网上书城不仅能让人们随时随地的浏览古今中外，丰富多彩的各类书籍，还能时时关注第一手资讯，并且支持网上付款，送货到家，可以说是秀才不出门，便知天下事。

该类型的网站通过前台与后台的系统相互协作来完成。前台宣传书城的书籍并将其展现给用户，让用户能自由浏览和选购自己所需的书籍，购买及确定下单。管理员通过后台系统来管理订单、用户、书籍来维护系统正常运行。

# 第二章 需求分析

## 2.1 引言

该系统的使用者主要包括 3 个种类：游客、用户、管理员。

对于游客和用户而言，需要前台提供良好的书籍展示界面，能够帮助其方便地找到或检索出自己需要的书籍，并且能够选择查看自己所需书籍的详细信息。相应地，在获取书籍信息之外，用户们还需要查看自己的个人历史订单。与此同时，系统应当保证用户顺利的选择所需的书籍加入购物车，并且进行结算。

对系统的后台管理员而言，对系统中的用户及书籍信息进行维护与操作。因此，在后台管理页面的设计上，应当更加注重实用性和易用性。管理员用户对于系统中信息应当具有最高的管理权限，后台系统除管理员外，其他人无权限使用。

## 2.2 游客需求分析

1. 浏览书籍
2. 根据关键词查找书籍

## 2.3 用户需求分析

1. 注册:用户填写注册资料, 即可进行注册。
2. 登录:在用户登录界面, 输入用户名和登录密码, 实现用户登录。
3. 用户查看书籍详细信息。
4. 用户添加书籍入购物车并结算。
5. 用户查看购物车, 可以将书籍从购物车中删除, 还可以修改数量, 或者想直接购买时, 可以点击下单。
6. 用户查看个人历史订单。
7. 用户可以通过搜索框或者首页下方的列表查找自己需要的书籍, 有满意的书籍后, 可以加入购物车。

## 2.4 管理员需求分析

1. 用户管理
  - 1) 查看所有用户的详细信息。
  - 2) 管理员可以添加用户, 并确定用户的权限等级。,
  - 3) 对于已经存在的用户, 可以修改用户的信息或者删除用户。
2. 书籍管理
  - 1) 查看所有已存在书籍的详细信息。
  - 2) 更新书籍的类型和种类等。
  - 3) 添加新的书籍及其详细信息
3. 查看实时在线人数

# 第三章 系统结构分析与设计

## 3.1 引言

### 3.1.1 可行性研究

#### 1. 技术可行性

该系统在设计过程中所应用到的技术, 负责前台显示的为 JSP 和 JavaBean, 后台管理涉及的 Servlet 与 MySQL 技术都是已经发展成熟、且被广泛使用的开发技术。通过采用 MVC 三层设计模式将这几种开发技术相结合的方式, 不但降低了网络系统的开发成本和程序的复杂度, 而且使得开发出的软件系统具有跨平台性、较强可移植性以及强可扩展性等特点。利用现有的技术条件, 开发者完全可

以开发出满足用户多种需求的网上书城。

## 2. 经济可行性

本系统主要采用的开发工具，如 Apache Tomcat 服务器，Eclipse 集成开发环境以及 MySQL 数据库管理系统均属于免费软件。从成本的角度考虑，并不需要太多的资金投入，完全可以实现低成本开发。

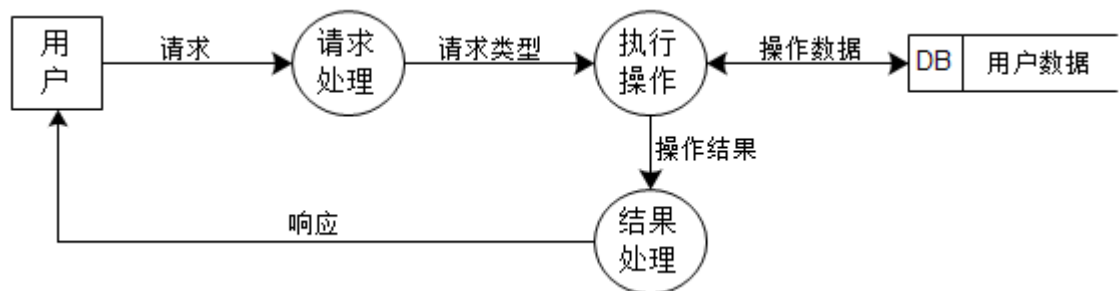
从市场角度来看，移动互联网的发展日趋迅猛，对生活的需求的便利性，快速性的要求也日益提高。网上书城作为一个面向各个年龄层提供书籍购买的平台，给生活节奏较快，缺少闲暇时间逛实体书店的人提供了极大的便利，迎合了广大人民群众的需要，具有巨大的市场潜力。

## 3. 操作可行性

对系统的设计与开发者而言，在细致地进行了分析研究和设计的基础上，通过合理的分模块实现，完全能够成功地完成最终的系统。另一方面，对于使用者，无论是普通用户还是管理员用户，该系统将提供一套完整、简洁的界面帮助其找到并实现自己需要的功能，并不需要他们具备太多的专业知识即可轻松地完成操作。同时，由于本系统采用 B/S 开发模式，基于 web 平台的软件系统能够很好地实现跨平台、跨区域进行使用，为用户提供最大程度的便利。

### 3.1.2 数据流图

该系统的所有用户向服务器发送请求并被服务器成功接受之后，服务器程序首先判断用户的请求类型，由请求类型决定接下来需要执行的操作。确定执行的操作之后，开始进行对目标数据的处理，其中涉及将对数据库信息的操作；处理操作完成后，对操作的返回结果进行包装；最后服务器端将请求处理的结果形成响应发送回用户端。



### 3.1.3 系统流程图

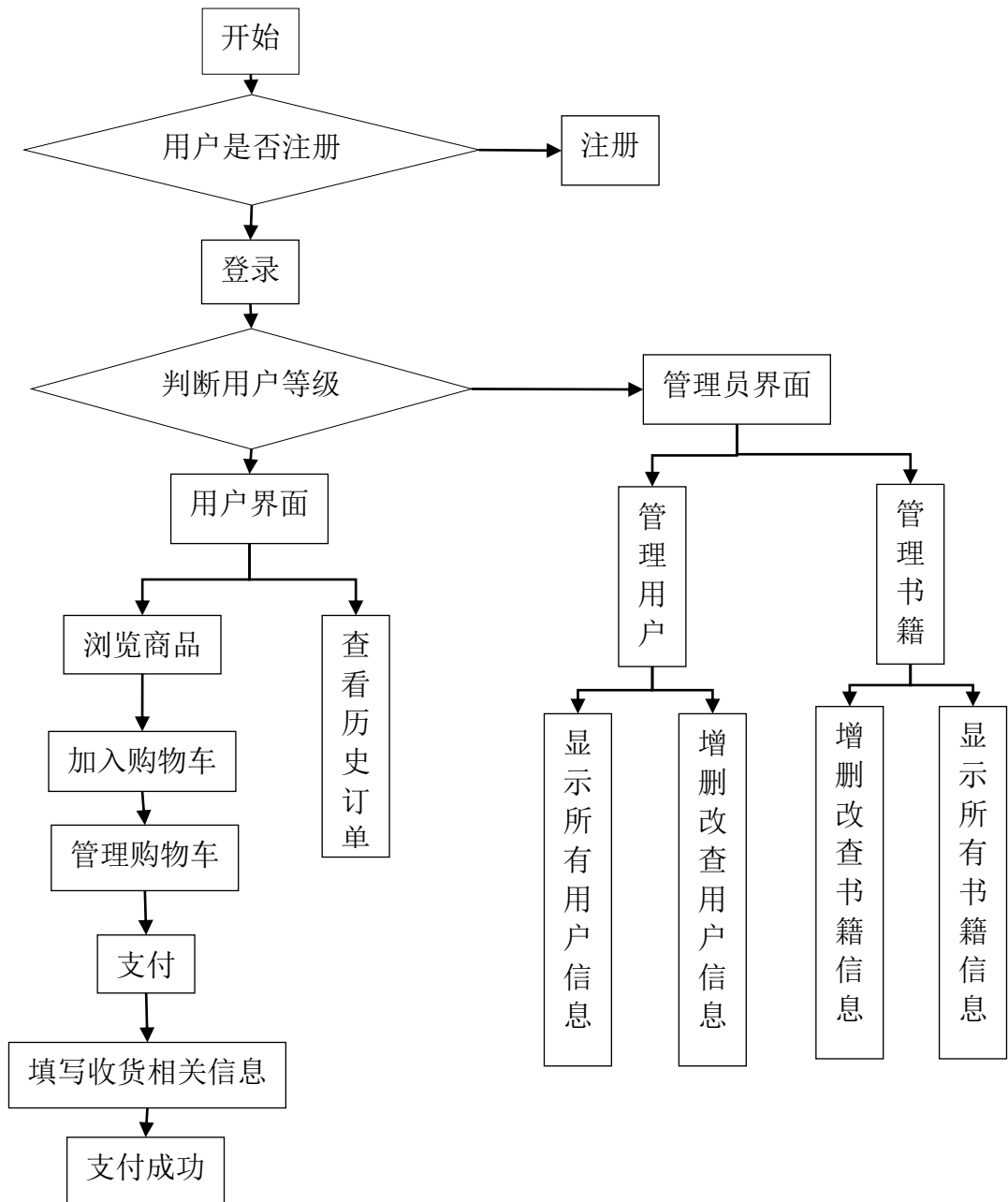
该系统在建设过程中，将使用者划分为了包括普通用户和管理员这两大类

别。

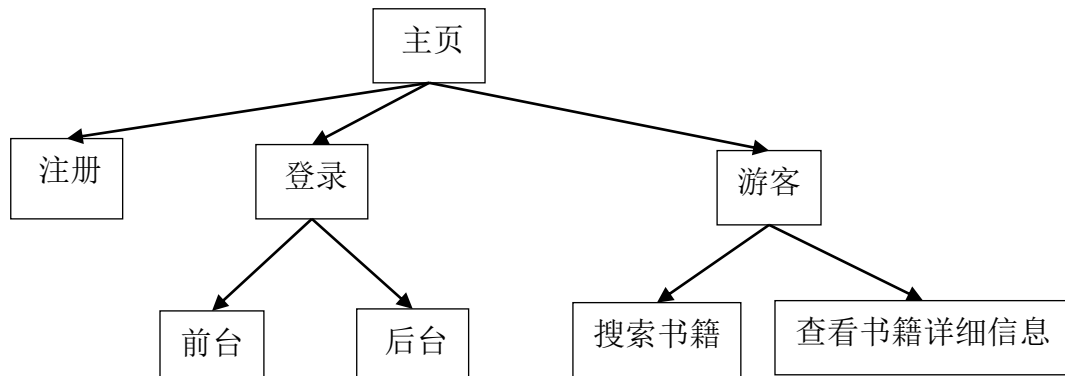
对普通用户提供前台操作界面，其包含的主要功能有：注册帐号、查看个人历史订单、搜索书籍内容、浏览书籍并购买。

当用户进入系统后，若该用户尚未注册，仍可以对网站所有书籍进行浏览、并可以搜索相关信息信息，但不能加入购物车，也不能进行购买；若其已经成为注册用户，可以直接进行登录，进入书城主页。

对于管理员用户而言，由于其工作将影响系统的正常运行，因此不向其提供注册功能，而是通过管理者手动添加用户信息到数据库的方法实现用户添加；管理员用户在登陆后进入系统的后台管理界面。而后可对所有用户或书籍进行相应的管理。

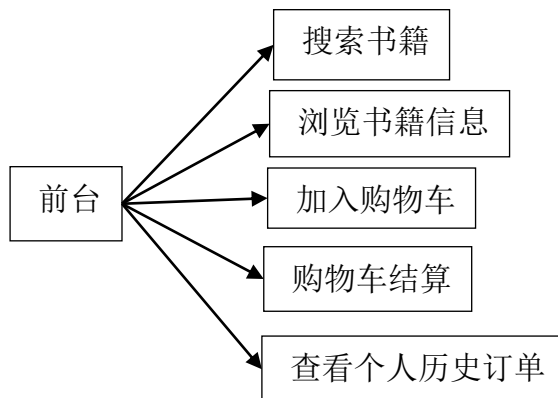


## 3.2 系统模块结构图



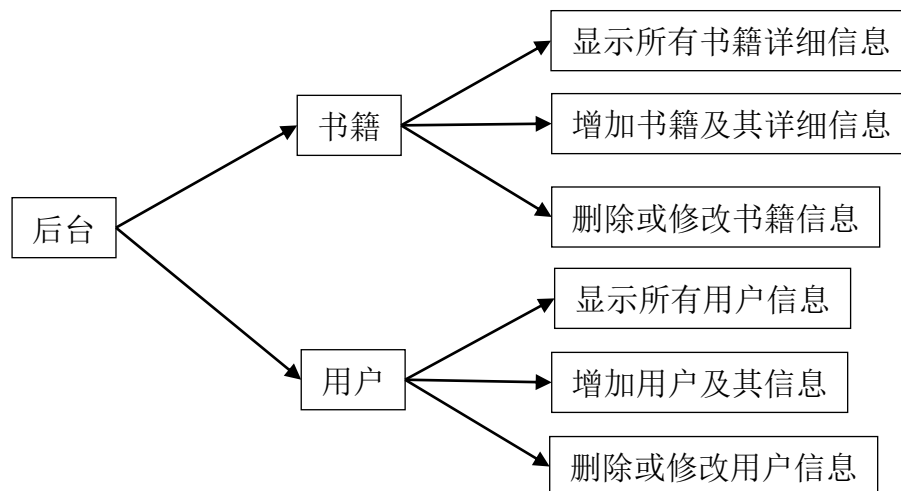
## 3.3 模块一

前台模块



## 3.4 模块二

后台模块



## 第四章 关键技术研究

### 4.1 实现技术路线

该系统采用了目前较为流行的 B/S(浏览器 / 服务器)结构模式, 主要使用 Java 语言开发, MYSQL 作为后台数据库, 前端页面使用 HTML、CSS、JavaScript 与 JSP 相结合实现页面展示。总体采用了 MVC 设计模式。MVC 三层模式的使用将服务端的代码与客户端的代码分离, 三层代码互不干扰, 即可以对各层进行分别操控, 便于修改和维护, 更利于了各层的重用, 提高开发效率和系统的灵活性。连接数据库上采用了数据库连接池来提高效率。

开发环境:

操作系统: Windows10

数据库: MYSQL

开发语言: JSP、HTML + CSS + JavaScript

服务器: Tomcat

编辑器: Eclipse

### 4.2 关键技术研究

其中主要研究实现的技术有: 统计实时在线人数、禁止未登录用户查看后台系统、实现分页查看、图片的上传。

#### 4.2.1 关键技术一: 监听器: 统计实时在线人数

在 JavaWeb 应用开发中, 需要统计当前在线的用户数, 可以使用监听器技术来实现这个功能。通过监听 HttpSession 域对象的创建和销毁来近似地确定实时在线人数。

##### 1) 分析

HttpSessionListener 接口用于监听 HttpSession 对象的创建和销毁。

①用户每一次访问时, 服务器创建 session: 创建一个 Session 时, 激发 sessionCreated(HttpSessionEvent se)方法;

②如果用户的 session30 分钟没有用或者可以自主在 web.xml 里面也可以配置 session 的失效时间: 销毁一个 Session 时, 激发 sessionDestroyed(HttpSessionEvent se)方法。

##### 2) 实现

编写一个 CountNumListener 类, 实现 HttpSessionListener 接口, 监听 HttpSession 对象的创建和销毁, 并且要在 web.xml 文件中注册监听器。

```
public class CountNumListener implements HttpSessionListener {
    @Override
    public void sessionCreated(HttpSessionEvent se) {
        //得到 ServletContext
        ServletContext context = se.getSession().getServletContext();
        Integer count = (Integer) context.getAttribute("count");
        if (count == null) {
            count = 1;
            context.setAttribute("count", count);
        } else {
            count++;
            context.setAttribute("count", count);
        }
    }
    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        //得到 ServletContext
        ServletContext context = se.getSession().getServletContext();
        Integer count = (Integer) context.getAttribute("count");
        count--;
        context.setAttribute("count", count);
    }
}
<listener>
    <listener-class> listener.CountNumListener</listener-class>
</listener>
```

## 4.2.2 关键技术二：过滤器：禁止未登录用户查看后台系统

为了提高系统运行时的数据的安全性，后台系统将禁止游客或是普通用户进行相关的操作。因而使用到了过滤器来过滤没有相关权限的用户访问后台系统

### 1. 分析

在这个系统中 jsp 页面数量还是比较多，但需要后台系统只占其中的一小部分且都存放在同一个文件夹中，即可采用 URL 级别的权限访问控制。

编写一个 Filter 可以在用户请求目标资源执行之前，进行登录检查，检查用户有无登录，如有登录则放行，如没有，则拒绝访问并重定向到登录界面；

### 2. 实现

1) 编写 Java 类实现 Filter 接口，并实现其 doFilter 方法；



---

```
public class LoginFilter extends HttpFilter{
    //1. 从 web.xml 文件中获取 sessionKey, redirectUrl, uncheckedUrls
    private String sessionKey;
    private String redirectUrl;
    private String uncheckedUrls;
    @Override
    protected void init() {
        ServletContext servletContext =
getFilterConfig().getServletContext();
        sessionKey = servletContext.getInitParameter("userSessionKey");
        redirectUrl = servletContext.getInitParameter("redirectPage");
        uncheckedUrls =
servletContext.getInitParameter("uncheckedUrls");
    }
    @Override
    public void doFilter(HttpServletRequest request,
        HttpServletResponse response, FilterChain filterChain)
        throws IOException, ServletException {
        //1. 获取请求的 servletPath
        String servletPath = request.getServletPath();
        //2. 检查 1 获取的 servletPath 是否为不需要检查的 URL 中的一个,
若是, 则直接放行. 方法结束
        List<String> urls = Arrays.asList(uncheckedUrls.split(","));
        if(urls.contains(servletPath)){
            filterChain.doFilter(request, response);
            return;
        }
        //3. 从 session 中获取 sessionKey 对应的值, 若值不存在, 则重定
向到 redirectUrl
        Object user = request.getSession().getAttribute(sessionKey);
        if(user == null){
            response.sendRedirect(request.getContextPath() +
redirectUrl);
            return;
        }
        //4. 若存在, 则放行, 允许访问.
        filterChain.doFilter(request, response);
    }
}
```

---

```
}
```

2) 在 web.xml 文件中使用<filter>和<filter-mapping>元素对编写的 filter 类进行注册，并设置它所能拦截的资源。

```
<context-param>
    <param-name>redirectPage</param-name>
    <param-value>/login.jsp</param-value>
</context-param>
<context-param>
    <param-name>uncheckedUrls</param-name>
    <param-value>/admin/index.jsp,/shop_list.jsp,/single-product.jsp</param-value>
</context-param>
<filter>
    <filter-name>LoginFilter</filter-name>
    <filter-class>web.filter.LoginFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>LoginFilter</filter-name>
    <url-pattern>/admin/*</url-pattern>
</filter-mapping>
```

### 4.2.3 关键技术三：实现分页查看

#### 1. 技术

采用了 JSP+servle+JavaBean 来实现分页，从而在前台显示了所有的书籍信息。

#### 2. 分析

将分页操作分为了首页，上一页，下一页，末页，还需要从数据库统计总记录数，设置每页的记录数，从而得到总页数。

将当前的页面设为 newPage

- |              |       |                         |                   |
|--------------|-------|-------------------------|-------------------|
| (1) 当前页      | ----- | 打开网页时看到的页面              |                   |
| (2) 首页       | ----- | 第一页                     | newPage=1         |
| (3) 上一页      | ----- | 当前页-1                   | newPage-1         |
| (4) 下一页      | ----- | 当前页+1                   | newPage+1         |
| (5) 末页       | ----- | 当前页==总页数                | countPage=newPage |
| (6) 总记录数     | ----- | select count(*) from 表名 |                   |
| (7) 总页数      | ----- | 总记录数%每页显示的记录数           |                   |
| (8) 显示当前页的分析 |       |                         |                   |

根据网页的界面布局，设定每页显示 6 条记录

---

第 1 页: newpage=1	起始记录为 0	6
第 2 页: newpage=2	起始记录 6	12
第 3 页: newpage=3	起始记录 12	18
第 n 页 newpage=n	(newpage-1)*pageSize	

### (9) 查询指定的页面

第一页: Select \* from test limit 0,6 从 0 开始查询, 每页显示 6 条记录

第 n 页: Select \* from test limit (newpage-1)\*pageSize, pagesize

### 3. 实现

- 1) 每本书为一个独立的个体, 封装所有信息为一个 JavaBean: Paging。
- 2) DAO: 连接数据库, 获取对应 production 表的总记录数、计算总页数和查询所有书籍的信息。

```
public interface PagingDao {
    //显示总的记录条数
    Integer getCountRecord();
    //根据当前页到结束页的查询
    List<Paging> findLimitPage(Integer newPage);
    //总的页数
    Integer getCountPage();
}
```

其中查询根据用户选择的哪一页来确定在数据库的查询语句的限制数。

// 根据传过来的数值条件查询

```
public List<Paging> findLimitPage(Integer newPage) {
    // 修改返回值
    List<Paging> entities = new ArrayList<Paging>();
    // 获取连接
    // 定义 SQL 语句
    String sql = "select id,name,address from test limit ?,?";//
```

参数为(newPage - 1) \* pageSize 和 pageSize

```
try {
    // 创建预处理对象
    pstmt = conn.prepareStatement(sql);
    // 为占位符赋值
    int index = 1;
    pstmt.setObject(index++, (newPage - 1) * pageSize);
    pstmt.setObject(index++, pageSize);
    // 执行更新
    rs = pstmt.executeQuery();
```

```
// 判断
while (rs.next()) {
    Paging entity = new Paging();
    entity.setId(rs.getInt("id"));
    entity.setName(rs.getString("name"));
    entity.setAddress(rs.getString("address"));
    entities.add(entity);
}
// 释放资源
if (rs != null) {
    rs.close();
}
if (pstmt != null) {
    pstmt.close();
}
} catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
return entities;
}
```

3) 最后由 servlet 进行双向传递, 将从 JSP 获取的页面数传到 DAO 层, 经过 DAO 层从数据库获取信息, 传回 servlet, 再在 jsp 页面呈现。

#### 4.2.4 关键技术四：图片的上传

因为书籍信息的需要, 管理员添加新的书籍及其信息时, 需要上传其封面图, 所以采用图片上传的技术, 将图片上传到指定的文件夹, 再将其存储的路径存入数据库的相应位置。需要导入来源组件 commons-fileupload.jar 和 commons-io-2.2.jar。

##### ① 分析

在实现这个功能的时候, 要注意的地方有许多, 比如:

- (1) 上传的图片名字为乱码。
- (2) 提交表单的类型一定要为 multipart/form-data。
- (3) 确定上传文件的保存目录。
- (4) 限制上传文件的格式, 因为在此系统中要求上传的是书籍的相关图片, 所以必须要求为图片的对应格式, 不能为文本文件或其他。

##### ② 实现

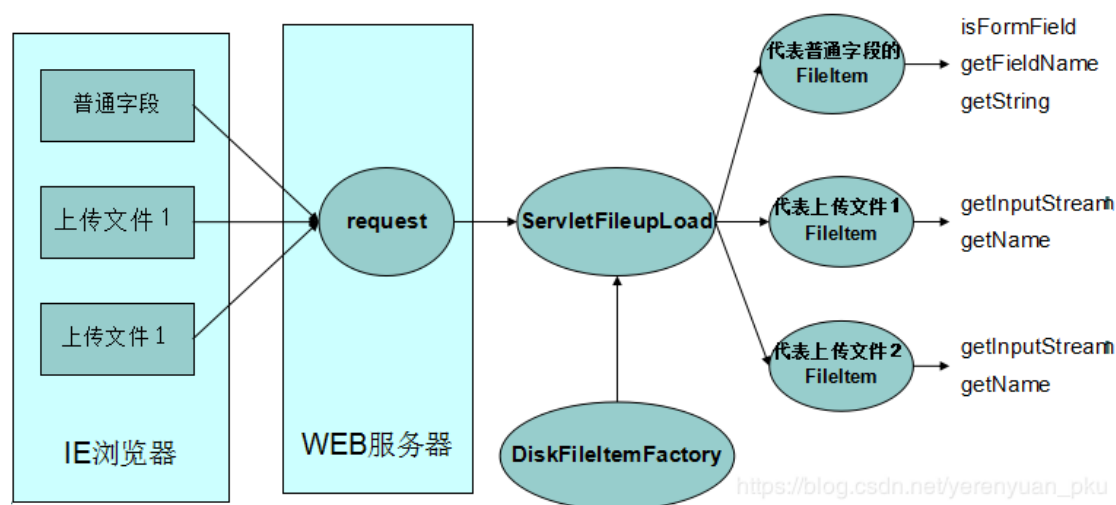
## (1) 图片上传的表单

```

<form action="${pageContext.request.contextPath}/FileUploadServlet"
        enctype="multipart/form-data" method="post">
    img: <input type="file" name="file"><br/>
    <input type="submit" value="submit" />
</form>

```

## (2) Servlet 利用组件读取图片上传的数据, fileupload 组件工作流程如下图所示:



## (3) Servlet 处理图片上传部分

- 创建 `DiskFileItemFactory` 对象, 可能需要设置缓冲区大小和临时文件目录, 也即创建解析工厂;
- 使用 `DiskFileItemFactory` 对象创建 `ServletFileUpload` 对象, 可能需要设置上传文件的大小限制, 也即创建解析器;
- 调用 `ServletFileUpload.parseRequest` 方法解析 `request` 对象, 得到一个保存了所有上传内容的 `List` 对象;
- 对 `List` 集合进行迭代, 每迭代一个 `FileItem` 对象, 调用其 `isFormField` 方法判断是否是上传文件;
  - 1) 为普通表单字段, 则调用 `getFieldName`、`getString` 方法得到字段名和字段值;
  - 2) 为上传文件, 则调用 `getInputStream` 方法得到数据输入流, 从而读取上传数据。

```

public class FileUploadServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        List<String> types = Arrays.asList(".jpg", ".png");
        String savepath;
        try {

```

---

```

        //创建解析工厂
        DiskFileItemFactory factory = new DiskFileItemFactory();
        //内存缓冲区的大小，默认值为 10K
        factory.setSizeThreshold(1024 * 1024);
        //设置内存缓冲区的大小为 1M
        factory.setRepository(new
File(this.getServletContext().getRealPath("/img/temp")));
        //创建解析器
        ServletFileUpload upload = new ServletFileUpload(factory);
        //限制上传文件的大小
        upload.setFileSizeMax(1024 * 1024 * 5);
        //只要超出 5M，for 循环在解析的时候就会抛异常
        //提交的表单类型不是 multipart/form-data，则没必要用解析器进
行解析数据，按照传统方式获取表单数据即可
        if (!upload.isMultipartContent(request)) {
            //按照传统方式获取表单数据
            request.getParameter("username");
            //balabala.....
            return;
        }
        //解决上传文件的中文名称的中文乱码问题，设置解析器的编码，
        upload.setHeaderEncoding("UTF-8");
        //调用解析器解析 request，得到保存了所有上传数据的 List 集合
        List<FileItem> list = upload.parseRequest(request);
        //迭代 List 集合，拿到封装了每一个输入项数据的 FileItem 对象
        for (FileItem item : list) {
            //判断 FileItem 的类型，如果是普通字段，则直接获取数据，如果
是上传文件，            则调用流获取数据写到本地硬盘
            if (item.isFormField()) {
                //为普通输入项的数据
                String inputName = item.getFieldName();
                String inputValue = item.getString("UTF-8");
                //解决普通输入项的中文乱码问题
                System.out.println(inputName + "=" + inputValue);
            } else {
                //代表当前处理的 item 里面封装的是上传文件
                String                filename                =
item.getName().substring(item.getName().lastIndexOf("\\") + 1);

```

---

```

        if (filename == null || filename.trim().equals("")) {
            continue;
        }
        //拿到文件的扩展名
        String ext =
filename.substring(filename.lastIndexOf(".") + 1);
        if (!types.contains(ext)) {
            request.setAttribute("message", "本系统不支持" +
ext + "这种类型文件的上传");
            request.getRequestDispatcher("/message.jsp").forward(request,
response);

            return;
        }
        InputStream in = item.getInputStream();
        int len = 0;
        byte[] buffer = new byte[1024];
        //得到保存在服务器中唯一的文件名
        String saveFileName = generateFileName(filename);
        //产生文件的保存目录
        savepath =
generateSavePath(this.getServletContext().getRealPath("/img"),
saveFileName);

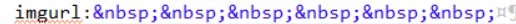
        request.setAttribute("imgurl", savepath);
        FileOutputStream out = new FileOutputStream(savepath
+ File.separator + saveFileName);
        while ((len = in.read(buffer)) > 0) {
            out.write(buffer, 0, len);
        }
        out.close();
        in.close();
        item.delete();
        //删除临时文件，必须位于流关闭之后
    }
}

} catch (FileUploadBase.FileSizeLimitExceededException e) {
    request.setAttribute("meassage", "文件大小不能超过 5M");
    request.getRequestDispatcher("/message.jsp").forward(request,
response);

```

```
        return;
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
request.getRequestDispatcher("/admin/add_production.jsp").forward(request, response);
}
```

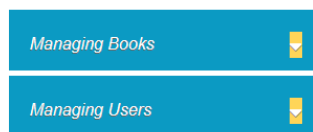
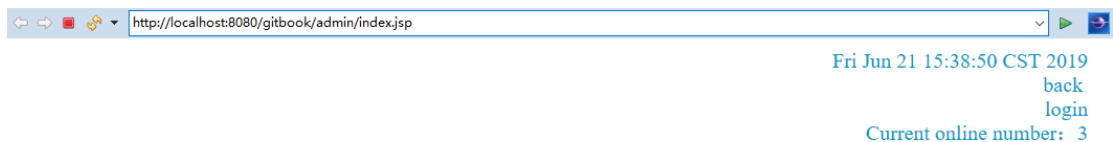
- (4) 将图片保存的路径存到 attribute 对象，在 jsp 页面获取，赋为 imgurl 的值

```
<form action="${pageContext.request.contextPath}/FileUploadServlet" enctype="multipart/form-data">
    <input type="file" name="file">
    <input type="submit" value="submit" class="button"/>
</form>

    <input type="text" name="imgurl" value="<%=request.getAttribute("imgurl")%"/><br/>
```

## 第五章 系统实现

### 5.1 系统实现介绍

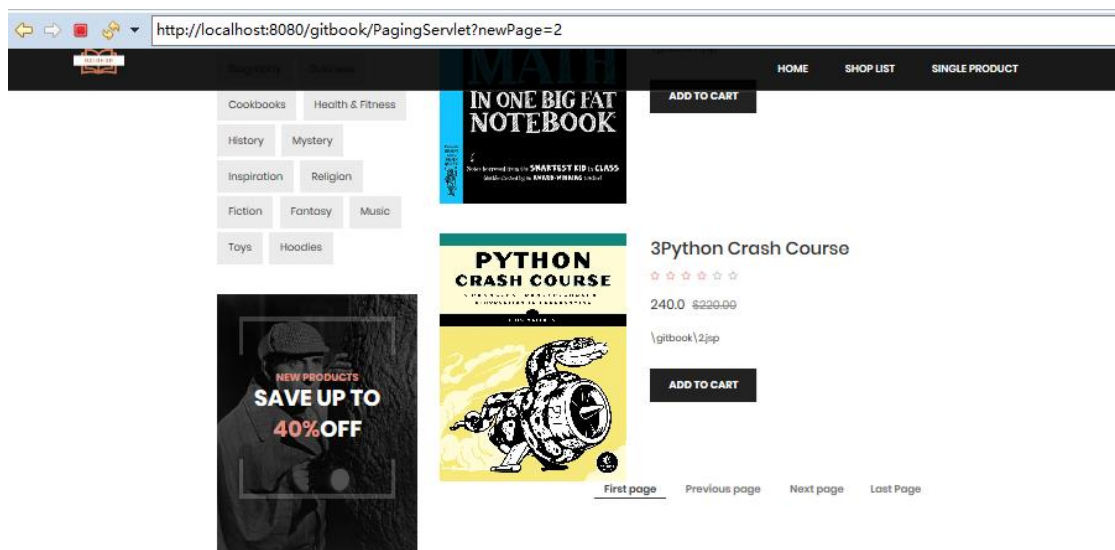
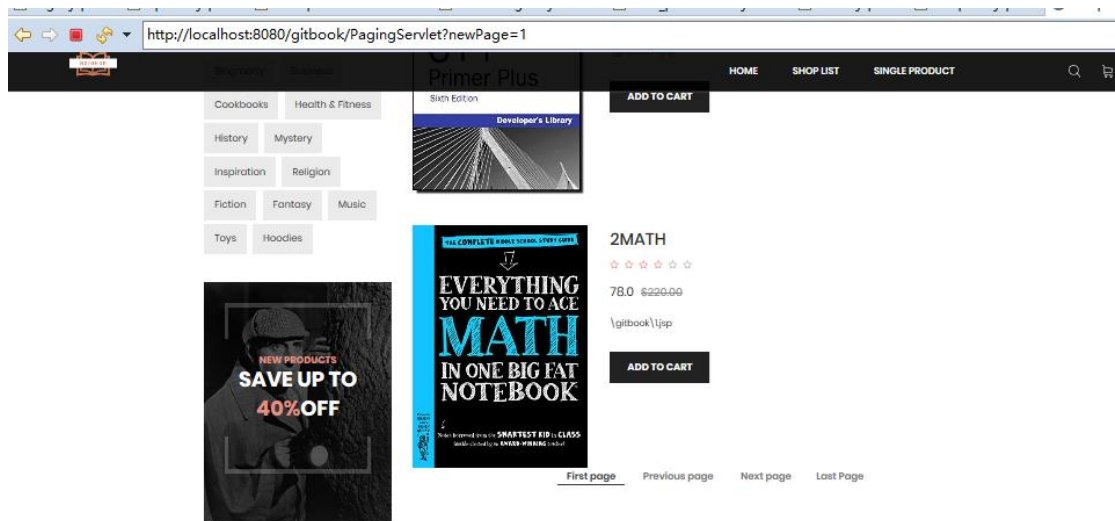
统计在线人数



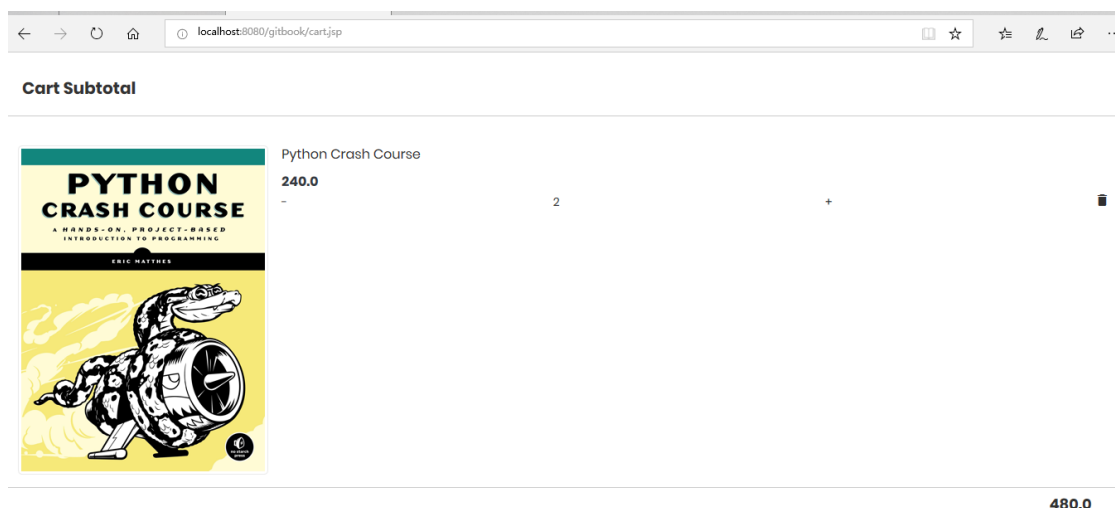
分页实现



[在此处键入]

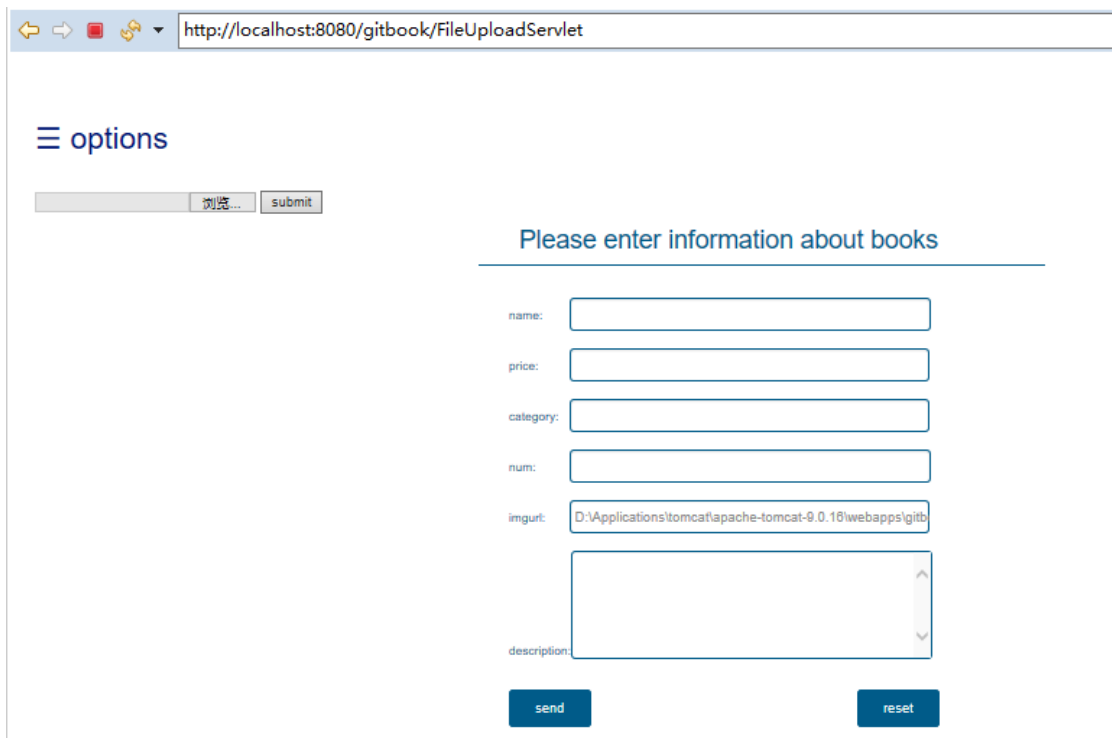


购物车修改数量或删除相关书籍



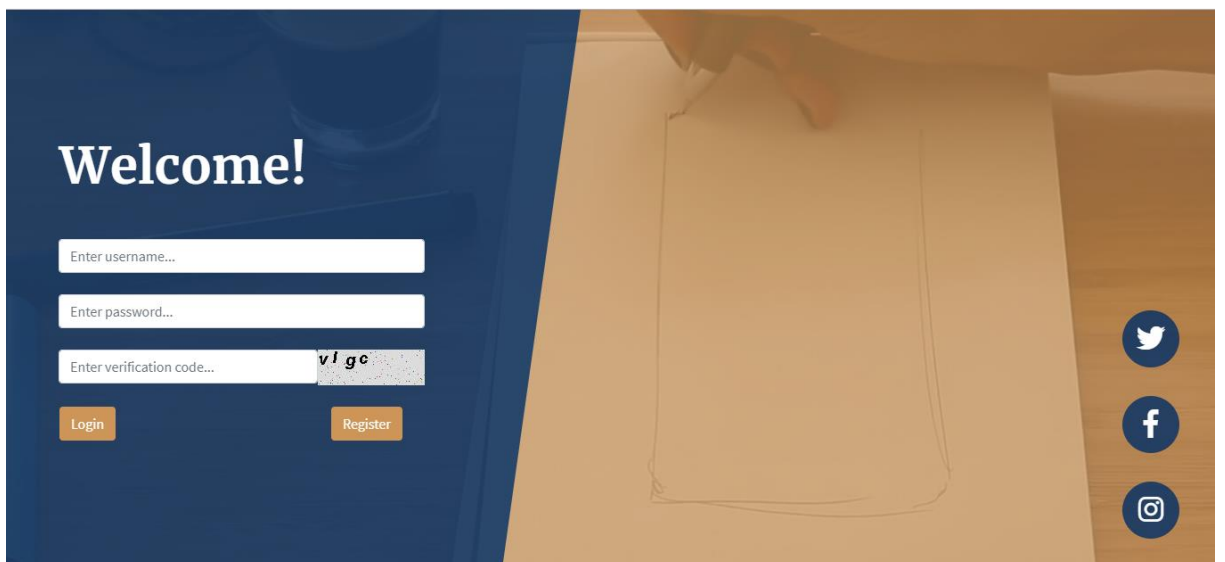
图片上传的使用效果：在管理员添加书籍时，可选择书籍图片进行上传，并返回其路径。

[在此处键入]



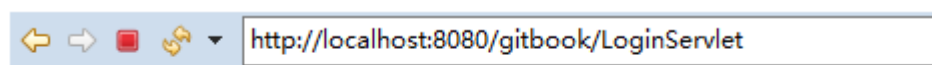
The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/gitbook/FileUploadServlet`. The page has a header with a hamburger menu icon and the text "options". Below the header, there is a file upload section with a "浏览..." (Browse...) button and a "submit" button. To the right, there is a section titled "Please enter information about books" with several input fields: "name:", "price:", "category:", "num:", "imgurl:" (containing the path `D:\Applications\tomcat\apache-tomcat-9.0.16\webapps\gitb`), and "description:". At the bottom of this section, there are two buttons: "send" and "reset".

## 登录验证码的使用



The screenshot shows a login page with a dark blue background. On the left, there is a "Welcome!" heading followed by three input fields: "Enter username...", "Enter password...", and "Enter verification code...". The verification code field contains the code "v1gc". Below these fields are two buttons: "Login" and "Register". On the right side of the page, there is a large, light brown rectangular area that looks like a piece of paper or a bag. In the bottom right corner, there are three circular social media icons: Twitter, Facebook, and Instagram.

点击登陆后，servlet 先检查验证码是否输入正确，正确则根据用户权限转到相应主页，否则输出错误信息，5 秒后转到登录界面。



The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/gitbook/LoginServlet`. Below the address bar, there is a message: "Verification code is invalid!After 5 seconds, transfer to the login page."

## 5.2 系统实现的不足

1. 在统计实时在线人数的部分，值为一个近似值。若是统一用户用若干个的浏览器同时登录，则会显示为若干个人。
2. 图片上传后存在了工程项目的某个文件目录下，但存在数据库中的信息为本地的绝对路径，项目移植后会报错，应为当前工程的相对路径。
3. 购物车修改商品数量时，点击按钮后会提交到 servlet 后返回结果，导致反复刷新页面，可以尝试采用 Ajax 异步刷新。
4. 用户提交订单后没有支付功能。
5. 数据库表的设计过于简单，缺少记录某一订单中具体的商品及其数量，导致显示用户历史订单时，显示的内容意义不太，无法查看购买的具体书籍及其数量。