

网 上 书 城

尚硅谷 java 研究院

版本：V1.0

第一阶段：登录、注册的验证

1. 使用 jQuery 技术对登录中的用户名、密码进行非空验证
2. 使用 jQuery 技术和正则表达式对注册中的用户名、密码、确认密码、邮箱进行格式验证，对验证码进行非空验证

第二阶段：实现登录、注册

1. 软件的三层架构

1.1 生活中的三层架构



- 服务员：只管接待客人。
- 厨师：只管做客人点的菜。
- 采购员：只管按客人点菜的要求采购食材。

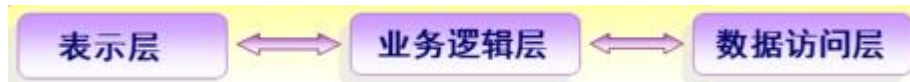
他们各负其职，服务员不用了解厨师如何做菜，不用了解采购员如何采购食材；厨

师不用知道服务员接待了哪位客人，不用知道采购员如何采购食材；同样，采购员不用

【更多 Java - Android 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

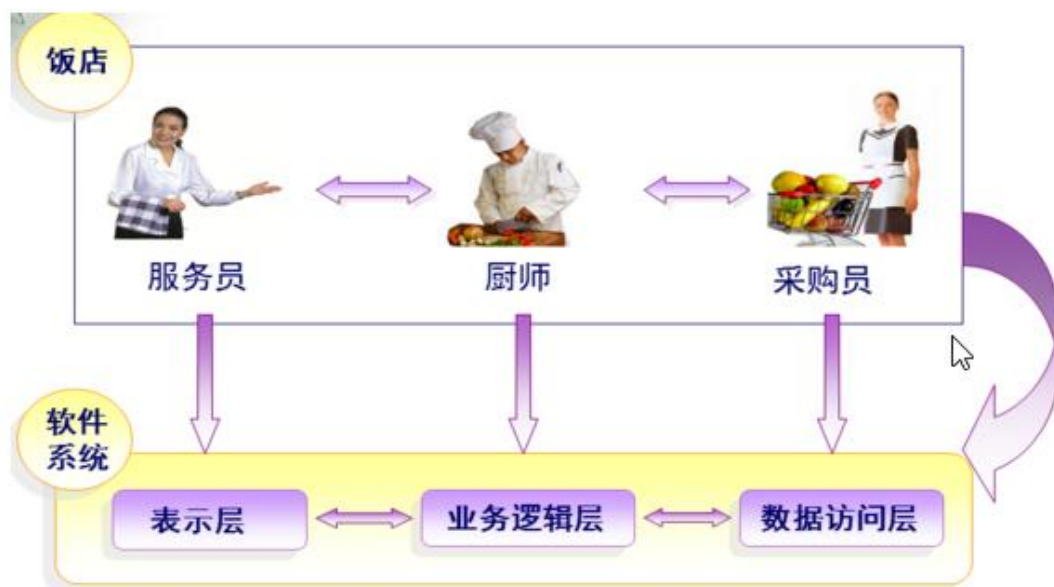
知道服务员接待了哪位客人，不用知道厨师如何做菜。

1.2 软件中的三层架构



- ✚ **UI (User Interface layer 表示层):**主要是指与用户交互的界面。用于接收用户输入的数据和显示处理后用户需要的数据。
- ✚ **BLL (Business Logic Layer 业务逻辑层):**UI 层和 DAL 层之间的桥梁。实现业务逻辑。业务逻辑具体包含：验证、计算、业务规划等等。
- ✚ **DAL (Data access layer 数据访问层):**与数据库打交道。主要实现对数据的增、删、改、查。将存储在数据库中的数据提交给业务层，同时将业务层处理的数据保存到数据库。（当然这些操作都是基于 UI 层的，用户的需求反映给界面 UI，UI 反映给 BLL，BLL 反映给 DAL，DAL 进行数据的操作，操作后再一一返回，直到将用户所需数据反馈给用户）。

1.3 为什么使用三层架构



✚ 使用三层架构的目的：**解耦!!!**

✚ 同样拿上面饭店的例子来讲：

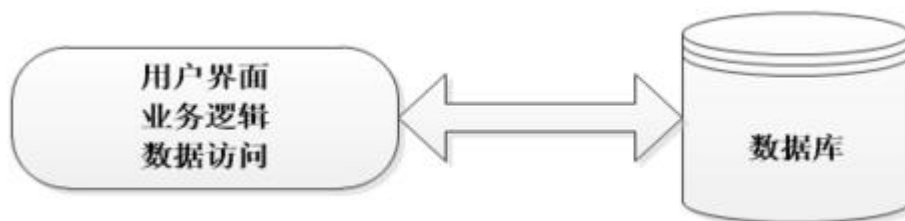
■ 服务员（UI 层）辞职——另找服务员；厨师（BLL 层）辞职——招聘另一个厨师；采购员（DAL）辞职——招聘另一个采购员。

■ 顾客反映：店里服务态度不好——服务员的问题，开除服务员；店里菜里有虫子——厨师的问题，换厨师。

✚ 任何一层发生变化都不会影响到另外一层!!!

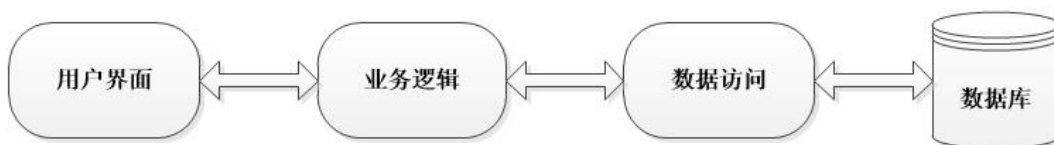
1.4 三层与一层的区别

一层：



✚ 当任何一个地方发生变化时，都需要重新开发整个系统。“多层”放在一层，分工不明确，**耦合度高**——难以适应需求变化，可维护性低、可扩展性低。

三层：



✚ 发生在哪一层的变化，只需更改该层，不需要更改整个系统。层次清晰，分工明确，每层之间**耦合度低**——提高了效率，适应需求变化，可维护性高，可扩展性高。

1.5 三层架构的优势和劣势

✚ 优势：

1) 结构清晰、耦合度低

【更多 Java - Android 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

- 2) 可维护性高，可扩展性高
- 3) 利于开发任务同步进行
- 4) 容易适应需求变化

劣势：

- 1) 降低了系统的性能。这是不言而喻的。如果不采用分层式结构，很多业务可以直接造访数据库，以此获取相应的数据，如今却必须通过中间层来完成。
- 2) 有时会导致级联的修改。这种修改尤其体现在自上而下的方向。如果在表示层中需要增加一个功能，为保证其设计符合分层式结构，可能需要在相应的业务逻辑层和数据访问层中都增加相应的代码。
- 3) 增加了代码量，增加了工作量。


2. 书城的三层架构


2.1 表示层（表现层）


 HTML、Servlet

 接受用户的请求，调用业务逻辑层处理用户请求，显示处理结果

2.2 业务逻辑层（Service 层）


 Service


 调用数据访问层处理业务逻辑

 采用面向接口编程的思想，先定义接口，再创建实现类

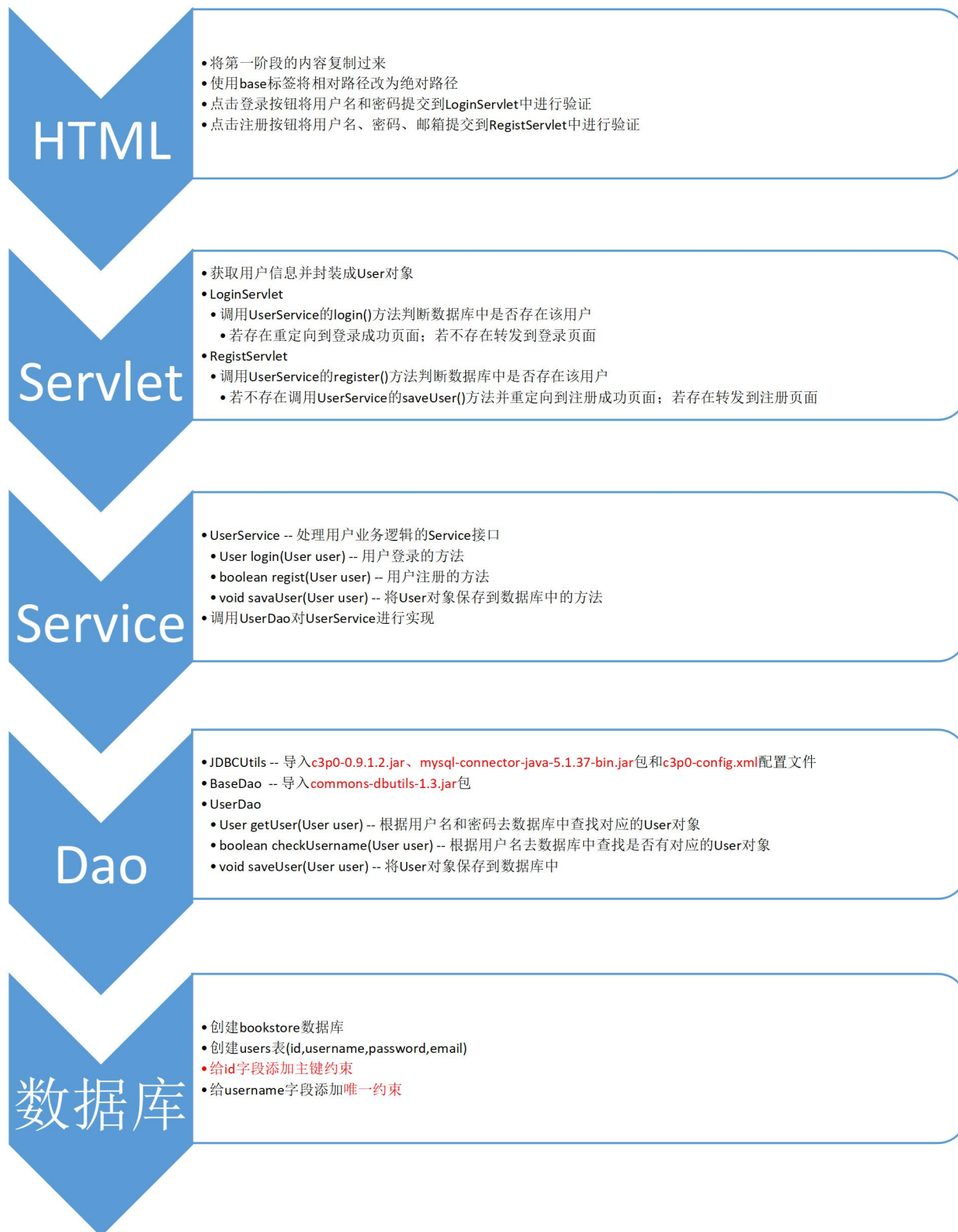
2.3 数据访问层（持久化层）

 Dao

 用来操作数据库，对数据库进行增删改查

 采用面向接口编程的思想，先定义接口，再创建实现类

3. 书城的三层架构具体实现



第三阶段：动态书城的开始及局部优化

1. 将所有 HTML 页面改为 JSP 页面

- ◆ 复制一份 BookStore02，重命名为 BookStore03，仅此还不够！还需要在 BookStore03 工程上右键→Properties→搜索 Web→单机 Web Project Settings→修改 Context root 为 BookStore03。

- ◆ 在每一个 HTML 页面的第一行添加 JSP 的 page 指令：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
```

然后将 HTML 页面的文件扩展名改为 jsp，即将.html 改为.jsp。

注意：一定要先在 HTML 页面中添加 jsp 指令再修改扩展名，否则页面中会出现乱码现象。

- ◆ 动态获取 base 标签

```
<base
```

```
href="<%=request.getScheme() %>://<%=request.getServerName() %>:<%=request.getServerPort() %><%=request.getContextPath() %>/">
```

✚ 获取 `http: request.getScheme()`

✚ 获取服务器名：`request.getServerName()`

✚ 获取端口号：`request.getServerPort()`

✚ 获取项目虚拟路径：`request.getContextPath()`

- ◆ 将每个页面都使用的那些重复的部分使用 JSP 指令中的 include 指令进行静态包含。

✚ 创建一个 base.jsp 页面，删除除 page 指令之外的所有内容；将 base 标签和 link 标签复制到 base.jsp 页面中

✚ 为了不让用户能够直接访问 base.jsp 页面，在 WEB-INF 下创建一个 include 目录，将 base.jsp 页面放入该目录中。

2. 显示错误消息及表单回显

- ◆ 登录、注册失败时我们分别转发到了登录、注册页面，但是并没有给用户一个错误消息，用户体验差，现在我们分别为登录、注册失败时提供一个错误消息并在页面显示：
 - ✚ 登录、注册失败时我们将错误消息放到 `request` 域中，通过**转发**将错误消息带到登录、注册页面，然后在页面中通过 `request` 域对象获取错误消息。
- ◆ 登录、注册失败时我们输入的用户名等信息会被清空，为了让用户有更好的体验，让用户看到之前输入的内容，需要对表单进行回显：
 - ✚ 给需要回显的表单项设置 `value` 属性，`value` 值通过 `request.getParameter("属性值")` 来获取。

3. 局部优化

3.1 优化一（整合 Servlet）：

- ◆ 现在我们的项目是一个请求对应一个 Servlet，**我们能不能将多个请求对应一个 Servlet 呢？**比如用户登录、注册的 `LoginServlet`、`RegistServlet` 使用一个 `UserServlet` 代替：
 - ✚ 在用户发送请求的 URL 地址后面添加一个 `method` 请求参数，在 `UserServlet` 中添加两个方法 `login()` 和 `regist()`。但是，两个方法的格式与 `doGet` 和 `doPost` 保持一致，可以复制一个 `doGet` 或 `doPost` 方法，将方法名改为 `login` 或 `regist`。
 - 登录表单提交到 `UserServlet?method=login`，注册表单提交到 `UserServlet?method=regist`。在 `UserServlet` 的 `doGet` 方法中获取请求参数，在 `doPost` 方法中调用 `doGet` 方法；在 `doGet` 方法中使用 `if else` 语句判断 `method` 的值，如果是 `login` 就调用 `login` 方法，如果是 `regist` 就调用 `regist` 方法。
 - 在 `UserServlet` 中创建 `UserService` 对象，将 `LoginServlet`、`RegistServlet` 的 `doPost` 方法中的内容分别复制到 `UserServlet` 的 `login` 方法和 `regist`

【更多 Java – Android 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

方法中即可实现之前的登录注册功能。

3.2 优化二（动态调用方法）：

- ◆ 在优化一中实现了多个请求对应一个 Servlet 的需求，但是我们在 UserServicelet 中每添加一个方法就要在 doGet 方法中添加一个 else if 语句！而且我们发现我们获取的请求参数的值正好是 UserServicelet 中的方法名，**那么我们能不能通过请求参数动态的调用对应的方法呢？**

✚ 通过**反射**的形式动态获取方法对象，这样我们再添加新的方法时就不需要再写 if else 语句了，但是新添加的方法的格式仍需要与 doGet 或者 doPost 的格式一致。

➤ 根据请求方式获取方法名

- `String methodName = request.getParameter("method");`

➤ 根据方法名获取方法的对象，`getDeclaredMethod()` 方法用来获取当前类中的某一个方法；第一个参数是方法名，第二个参数是方法的参数类型

- `Method method = this.getClass().getDeclaredMethod(methodName, HttpServletRequest.class, HttpServletResponse.class);`

➤ 设置访问权限

- `method.setAccessible(true);`

➤ 调用方法。`invoke()` 方法用来调用一个方法；第一个参数是要调用哪个对象的方法，第二个参数是调用方法需要传的参数

- `method.invoke(this, request, response);`

3.3 优化三（创建 BaseServlet）：

- ◆ 在优化二中我们实现了动态调用对应方法的需求，但是以后每次创建 Servlet 的时候都需要将以上代码重写一遍，那么我们为何不把以上代码单独放到一个

Servlet 中，我们给它命名为 BaseServlet，以后再创建 Servlet 时直接让它继承 BaseServlet 即可。

✚ 创建一个专门用来被其他 Servlet 继承的 BaseServlet

✚ 但是不可以重写 BaseServlet 中的 doGet 和 doPost 方法

4. 使用 EL 替换项目中的所有 JSP 表达式

第四阶段：图书的增删改查

1. 后台图书的增删改查

1.1 创建图书类 Book



✚ private Integer id;

✚ private String title; //书名

✚ private String author; //作者

✚ private double price; //价格

✚ private Integer sales; //销量

✚ private Integer stock; //库存

✚ private String imgPath = "static/img/default.jpg"; //封面图片的路径

【更多 Java – Android 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

➤ 指定一个默认值 static/img/default.jpg

1.2 创建图书表 books

1.3 创建 BookDao 接口及实现类

- ✚ public void saveBook(Book book); 向数据库中插入一本图书
- ✚ public void deleteBook(String bookId); 根据图书的 ID 删除一本图书
- ✚ public void updateBook(Book book); 更新一本图书
- ✚ public Book getBookById(String bookId); 根据图书的 ID 查询一本图书
- ✚ public List<Book> getBookList(); 获取所有的图书

1.4 创建 BookService 接口及实现类

- ✚ public void saveBook(Book book); 向数据库中插入一本图书
- ✚ public void deleteBook(String bookId); 根据图书的 ID 删除一本图书
- ✚ public void updateBook(Book book); 更新一本图书
- ✚ public Book getBookById(String bookId); 根据图书的 ID 查询一本图书
- ✚ public List<Book> getBookList(); 获取所有的图书

1.5 创建 BookManagerServlet

- ✚ bookList() → 获取图书列表的方法
- ✚ ~~addBook()~~ → 添加图书的方法
- ✚ deleteBook() → 删除图书的方法
- ✚ editBook() → 编辑图书的方法
- ✚ saveOrUpdate() → 保存或者更新图书的方法
- ✚ 修改图书操作

➤ 当我们点击修改图书时，首先我们需要根据当前图书的 ID 去数据库中将该图书的信息查询出来显示到编辑图书的页面 book_edit.jsp。

- `editBook()` → 通过 ID 从数据库中查询到该图书显示到 `book_edit.jsp` 页面。
- 当修改完图书之后我们需要把修改之后的图书信息提交到一个更新图书的方法中去，不过这个时候我们添加图书和修改图书的页面都是 `book_edit.jsp`，那么我们怎么判断我们是在添加图书还是在修改图书呢？
- 方式一：我们可以将添加图书和修改图书分离，再创建一个 `jsp` 页面
- 方式二：添加图书没有 ID，但是修改图书有 ID，所有我们可以通过是否有 ID 来判断当前是添加图书的操作还是修改图书的操作，不过我们需要把图书的 ID 放到一个隐藏域中。
- 将之前的 `addBook()` 方法改为 `saveOrUpdate()` 方法，在该方法中通过判断获取的图书 ID 是否为 null 来决定是调用 `BookService` 的 `saveBook` 方法还是 `update()` 方法。

1.6 添加分页

- ◆ 至此，我们管理端图书的增删改查已经完成，但是我们在获取图书列表的方法中是将数据库中的所有图书都查询了出来，如果数据库中的图书过多在页面显示时就会出现滚动条，不但用户体验不好而且一次性从数据库中查询这么多条记录性能也差。所以我们需要对图书进行分页显示：
- ✚ 在 `sql` 语句中使用 `limit` 关键字进行分页
- ✚ 我们知道我们从数据库中查询出来的图书放到了一个 `List` 集合中，但是如果添加分页操作我们需要知道总页数、当前页码、总记录数等信息，但是 `List` 中并没有这些信息，所有我们需要创建一个类来封装分页的信息。



Java编程思想	70.5	埃史尔	100	100	修改	删除
设计模式之禅	20.2	秦小波	100	100	修改	删除
图解机器学习	33.8	杉山将	100	100	修改	删除
艾伦图灵传	47.2	安德鲁	100	100	修改	删除
教父	29.0	马里奥普佐	100	100	修改	删除
添加图书						

分页控件: 首页 上一页 [4] 5 下一页 末页 共10页 30条记录 到第 4 页 确定

◆ Page<T>类

```
private List<T> list; //从数据库中查询的集合  
public static final int PAGE_SIZE = 5; //每页显示多少条记录  
private int pageNo; //当前页  
private int totalPageNo; //总页数  
private int totalRecord; //总记录数
```

◆ 修改 BookDao: 添加一个分页的方法

```
Page<Book> getPageBook(Page<Book> page) → 分页查找图书的方法
```

➤ 该方法由 BookService 调用, 传过来一个带有 pageNo 的 Page<Book> 类型的 page 参数, 通过该方法从数据库中先获取总的记录数, 然后通过 page 的 (pageNo-1)*PAGE_SIZE 和 PAGE_SIZE 获取一个带分页的 List 集合。给 page 设置了总的记录数和 List 后再将 page 返回。

◆ 修改 BookService: 同样添加一个分页的方法

```
Page<Book> getPageBook(String pageNo) → 分页查找图书的方法
```

➤ 该方法由 Servlet 调用, 接受的请求参数都是 String 类型的, 所以传入的页码 pageNo 设置为了字符串类型。实现该方法是首先要创建一个 Page<Book> 对象, 然后设置 pageNo 属性, 不过这里需要将获取的字符串类型的 pageNo 转换为 int 类型。**注意: 有可能会出现转换异常 (例如用**

户输入的是字符串 abc)

- ◆ 在 BookManagerServlet 中添加 getPageBook()方法
- ◆ 这时我们发现我们原来的 bookList()已经没用了……
 - ✚ 就会导致原来添加图书、修改图书、删除图书出现 bug。
 - ✚ 改完 bug 之后考虑一个问题：目前我们执行添加图书、删除图书、修改图书之后都回到了图书列表的首页，我们能不能让它回到执行之前的那个页面呢？
(提示：我们可以通过请求头中的 Referer 获取之前的地址)
- ◆ 关于分页页码的问题
 - ✚ 目前我们是将所有的页面都显示了出来，如果页面过多则用户体验不好，我们能不能让它只显示 5 个页面呢？
 - ✚ [\[1\]](#) [2](#) [3](#) [4](#) [5](#)
 - ✚ [1](#) [\[2\]](#) [3](#) [4](#) [5](#)
 - ✚ [1](#) [2](#) [\[3\]](#) [4](#) [5](#)
 - ✚ [2](#) [3](#) [\[4\]](#) [5](#) [6](#)
 - ✚ [3](#) [4](#) [\[5\]](#) [6](#) [7](#)
 - ✚ 综上所述：一共有三种情况
 - 1) 总页数小于 5 时
 - 2) 总页数等于 5，当前页小于等于 3 时
 - 3) 总页数大于 5，当前页大于 3 时

<!-- 设置两个变量 begin 和 end -->

<!-- 注意：当 end 大于总页数时这种情况 -->

<c:choose>

<c:when test="\${page.totalPageNo < 5 }">

<c:set var="begin" value="1"></c:set>

<c:set var="end" value="\${page.totalPageNo }"></c:set>

</c:when>

<c:when test="\${page.pageNo <=3 }">

<c:set var="begin" value="1"></c:set>

```
<c:set var="end" value="5"></c:set>

</c:when>

<c:otherwise>

    <c:set var="begin" value="${page.pageNo - 2 }"></c:set>

    <c:set var="end" value="${page.pageNo + 2 }"></c:set>

    <c:if test="${end > page.totalPageNo }">

        <c:set var="begin" value="${page.totalPageNo - 4 }"></c:set>

        <c:set var="end" value="${page.totalPageNo }"></c:set>

    </c:if>

</c:otherwise>

</c:choose>
```

◆ 分页完成之后如何复用分页

- ✚ 使页码超链接的请求地址动态显示
 - 获取请求地址
 - 获取查询字符串
 - 将请求地址与查询字符串使用?拼接
 - 给 Page 类添加一个 path 属性,将拼接好的地址设置到 page 对象的 path 属性中

2. 前台图书的显示

2.1 创建 BookClientServlet

- ✚ 添加 getPageBook()方法获取分页图书信息。

2.2 根据用户输入的价格查询图书

- ✚ 在 BookDao 中添加一个根据价格查询图书记录的方法
 - Page<Book> getPageBookByPrice(Page<Book> page,double min,double max);

- ✚ 在 BookService 中添加一个根据价格查询图书记录方法
 - Page<Book> getPageBookByPrice(String pageNo,String min,String max);
- ✚ 在 BookClientServlet 中添加 getPageBookByPrice()方法

2.3 完成了查询按钮之后我们发现一个问题：

- ✚ 点击下一页时查询条件丢失，如何实现带查询条件的分页呢：
 - 由于我们查询操作的表单使用的是 Post 请求，请求参数在请求体中，所以获取请求参数时获取不到最低价格和最高价格。
 - 如果我们将 post 请求改为 **get 请求**，那么我们提交表单时 action 属性中的 method 请求参数将被覆盖，即 method=getPageBookByPrice 将丢失，所有我们决定将 **getPageBookByPrice** 请求参数放到一个隐藏域中

2.4 修改 index.jsp 页面

- ✚ 将首页里的内容复制到另一个页面中
- ✚ 删除首页的内容使用<jsp:forward>标签转发到获取图书列表的 servlet

第五阶段：登录、登出 、验证码、购物车

1. 登录、登出

1.1 登录

- ✚ 登录成功后将用户保存到 Session 域中，根据 Session 域中是否含有用户信息来判断用户是否登录。

1.2 登出

- ✚ 在 UserServlet 中添加一个 logout 方法，在该方法中获取 Session 对象然后强制
- 【更多 Java – Android 资料下载，可访问尚硅谷（中国）官网 www.atguigu.com 下载区】

Session 失效即可。

2. 验证码

2.1 使用 Google 提供的第三方 jar 包

- ✚ 导入 kaptcha-2.3.2.jar 包
- ✚ 在 web.xml 文件中注册 Servlet
- ✚ 在 Session 对象中获取验证码，**获取之后不要忘记移除**

3. 购物车

3.1 购物车实现的三种方式：

- ✚ 基于 Cookie 的，将购物信息都放入到 Cookie 中，由浏览器保存。
- ✚ 基于 Session 的，将购物信息都放入到 Session 中，由服务器保存。✓
- ✚ 基于数据库表的，将购物信息都放入数据库中，由数据库保存。

商品名称	数量	单价	金额	操作
解忧杂货店	2	27.2	54.4	删除
边城	1	23.0	23.0	删除
中国哲学史	1	44.5	44.5	删除

购物车中共有 4 件商品 总金额 121.9 元 [继续购物](#) [清空购物车](#) [去结账](#)



```
graph TD
    A[购物车] --> B[购物车]
    C[删除] --> D[购物项]
```

3.2 创建购物项类 CartItem

- ✚ private Book book; //图书信息

✚ private int count; //图书的数量

✚ private double amount; // 购物项中图书的金额，通过计算得到

3.3 创建购物车类 Cart

✚ private Map<String, CartItem> map = new LinkedHashMap<String, CartItem>();

- 保存购物项的 Map，key 是 bookId，value 是 CartItem
- 使用 LinkedHashMap 是为了保证购物车中添加图书的顺序

✚ private int totalCount;

- 购物车中商品的总数量，通过计算得到

✚ private double totalAmount;

- 购物车中商品的总金额，通过计算得到

3.4 另外还需要在购物车类中添加一些方法：

✚ public List<CartItem> getCartItems();

- 获取购物车中所有的购物项，以便以后遍历

✚ public void addBook2Cart(Book book);

- 向购物车中添加图书
 - 注意：需要判断购物车中是否已经存在该图书：如果存在，让数量加一；
如果不存在，首先需要创建一个购物项，然后在购物项中 setBook，设置数量为 1，最后将该购物项添加到购物车的 map 中。

✚ public void delCartItem(String bookId);

- 删除一个购物项

✚ public void updateCartItem(String bookId, String countStr);

- 更新购物项
 - 根据图书 ID 和用户输入的数量更新购物项中图书的数量

✚ public void emptyCart();

- 清空购物车

3.5 创建处理购物车请求的 CartServlet

里面需要四个方法：添加图书、更新购物项、删除购物项、清空购物车

第六阶段：结账、添加事务、使用 Ajax

1. 结账

- ◆ 结账就是把我们的购物车中的东西转换成订单，我们之前将购物车中的东西都放在了 Session 对象中，但是订单中的东西我们就需要**保存在数据库**中了，所以我们就需要创建类和表。

1.1 订单

订单号	日期	数量	金额	状态	详情
14733841007541	2016-9-9 9:21:40	2	99.8	交易完成	查看详情
14733838864371	2016-9-9 9:18:06	10	272.0	确认收货	查看详情
14733838756481	2016-9-9 9:17:55	56	4480.0	未发货	查看详情

1.2 查看详情

封面	书名	作者	价格	数量	金额
	解忧杂货店	东野圭吾	27.2	10	272.0

1.3 和购物车类似，我们需要创建订单类和订单项类

1.3.1 订单类 Order

- `private String id;` // 订单号
- `private Date orderTime;` // 生成订单的时间
- `private int totalCount;` // 商品总数量
- `private double totalAmount;` // 商品总金额
- `private int state;` // 订单状态 0: 未发货 1: 已发货 2: 交易完成
- `private int userId;` // 订单所属的用户
- 对于订单号的要求:
 - 不能重复，唯一
 - 尽量少体现出订单信息
- 订单和用户的关系
 - 多对一的关系：一个用户可以有多个订单，一个订单只能属于一个用户

1.3.2 订单项类 OrderItem

- `private Integer id;` // 订单项 id
- `private int count;` // 买了多少本图书
- `private double amount;` // 买了 count 本图书花费的钱数
- `private String title;` // 所买图书的书名
- `private String author;` // 图书的作者
- `private double price;` // 图书的价格
- `private String imgPath;` // 图书封面
- `private String orderId;` // 订单项所属的订单
- 订单项和订单的关系
 - 多对一的关系：一个订单可以包含多个订单项，一个订单项属于一个

订单

1.4 操作订单的接口 OrderDao

- `public void saveOrder(Order order);` // 向数据库中添加订单
- `public List<Order> getOrders();` // 获取所有订单
- `public List<Order> getOrdersByUserId(int userId);` // 根据用户 id 查询用户的所有订单
- `public void updateOrderState(String orderId, int state);` // 根据订单号更新订单状态

1.5 操作订单项的接口 OrderItemDao

- `public void saveOrderItem(OrderItem orderItem);` // 向数据库中添加订单项
- `public List<OrderItem> getOrderItemsByOrderId(String orderId);` // 根据订单号查询所有订单项

1.6 处理订单相关业务的接口 OrderService

- `public String createOrder(Cart cart, User user);` // 生成订单，返回订单号
- **注意：这个方法里面的业务逻辑比较多**
 - 1) 根据当前时间和用户 id 拼接订单号
 - 2) 创建订单对象，将购物车中的信息封装到订单对象中并将订单保存到数据库中
 - 3) 创建订单项对象，将购物车中购物项的信息封装到订单项中，其中不要忘记更新图书的销量和库存，然后将订单项保存到数据库中
 - 4) 最后还需要清空购物车
- `public List<Order> getOrders();` // 获取所有的订单
- `public List<Order> getOrdersByUserId(int userId);` // 根据用户 ID 获取该用户的订

单

public List<OrderItem> getOrderItemsByOrderId(String orderId); //根据订单号获取所有订单项

public void sendOrTakeOrder(String orderId,int state); //管理员发货或者用户确认收货

不管是发货还是收货，本质就是更新订单的状态

1.7 客户端订单管理

- 结账、我的订单、订单详情、确认收货
 - 以上操作都需要判断用户是否登录

1.8 管理端订单管理

- 订单管理、发货

2. 添加事务

- ◆ 如果我们将 saveOrder 的 SQL 语句故意写错，模拟一个数据库出现异常的情况，订单将不能正常插入到数据库中，同时，由于订单号不存在，订单项也将不能插入进数据库。但是，图书的库存和销量却发生了改变。
- ◆ 所以我们要在项目中添加事务控制
- ◆ 我们在结账的操作中分别调用了三个 DAO，OrderDao、OrderItemDao、BookDao。对数据库做操作，这三次操作就是一个事务，三个操作要么都成，要么都失败。
- ◆ 事务控制的关键：在同一个事务中，不同的操作要求使用 **同一个的数据库连接**

2.1 使用 ThreadLocal<T>对象保证一个线程对应一个数据库连接

- ◆ 在 Java 中有一个类叫 ThreadLocal，在 ThreadLocal 的内部实际维护着一个 Map，这个 Map 的 Key 就是当前线程对象，值就是你想存的的对象，所以我们一般

使用 `ThreadLocal` 来在同一个线程中共享数据，他的作用有点像域对象。

◆ `ThreadLocal<T>` 中常用的方法

- `void set(T t)` → 向 `ThreadLocal` 保存一个对象
- `T get()` → 获取当前线程中保存的对象
- `void remove()` → 移除当前线程中保存的对象

2.2 使用过滤器 `Filter` 控制事务

- ◆ 我们在项目中添加一个 `TransactionFilter`，用于控制事务，我们在 `Filter` 中先获取数据库连接，然后开启事务，接着放行请求，当响应回来的时候，还会回到 `Filter`，这时我们的 `Filter` 可以对异常进行捕获，如果出现异常则回滚事务，如果没出现则提交事务。
- ◆ 如果添加 `TransactionFilter`，那么所有的数据库连接的关闭操作都应该在 `Filter` 中统一处理，而不应该再在 `DAO` 中操作，所以 `BaseDao` 中所有释放数据库连接的代码，全都应该注释掉。
- ◆ 我们在 `Filter` 统一处理事务，那么就希望异常可以一直抛到 `Filter` 中，然后 `Filter` 一旦捕获到异常就可以自动的回滚。但是我们发现异常在 `BaseDao` 中都被 `try...catch` 捕获了，如果直接 `throws` 向上抛异常那么它的父类也需要 `throws`，比较麻烦！
- ◆ 所以我们就需要在 `BaseDao` 中将所有捕获到 `SQLException` 转换为 `RuntimeException` 向上抛。
- ◆ 当我们在 `BaseDao` 中将异常向上抛以后，又出现了这么一个异常：
 - `java.lang.reflect.InvocationTargetException`
 - 这个异常一般在通过反射调用一个方法时，当那个方法有异常时，会导致抛出该异常。
- ◆ 我们在 `BaseServlet` 中通过反射调用 `Servlet` 中的方法，当方法有异常以后，又被 `BaseServlet` 捕获了，所以我们需要在 `BaseServlet` 中继续将异常向上抛。
- ◆ 当我们在 `BaseServlet` 中将异常向上抛以后，的确出现变化了，但是页面并没

有转到错误页面，也是说异常并没有被 Filter 捕获到，而是出现 500，证明异常被服务器捕获了。

- ◆ **原因：**我们在 Filter 中捕获的是 SQLException，但是异常过来的时候是 RuntimeException，所以我们 Filter 没有捕获该异常，这里我们需要修改 Filter 捕获的异常类型为 Exception。

3. 使用 Ajax

3.1 注册页面使用 Ajax 验证用户名是否可用

3.2 修改购物车中图书数量使用 Ajax