## 第一阶段：使用jQuery对注册页面进行验证

## 第二阶段：完成登录、注册功能

### 软件结构

生活案例：

**顾客 服务员 厨师 采购员 菜市场**

思考：

1. 当香菇炖排骨做好之后，可以由厨师或采购员给顾客上菜吗？
2. 假若厨师或采购员感觉工作辛苦想离职，那会影响其它岗位上同事的工作吗？

分析：

这里整个饭店分出三个岗位，分别是服务员、厨师、采购员。所以我们的软件结构的初始模型也可以这样划分一下。只不过要换一种说法：

顾客 ： 用户

服务员 ： 表现层(专门用来与用户进行交互的)

厨师 ： 业务逻辑层(主要用来处理业务)

采购员 ： 数据层(专门与数据库进行打交道的,比如数据的增删改查)

菜市场 ： 数据库

**》》软件的三层架构：**

数据库

持久化层

业务逻辑层

表现层

表现层：(表示层)

作用：用来与用户进行交互

体现：html文件、servlet

业务逻辑层：(service层)

作用：用来处理业务

体现：登录、注册、添加图书、购物等

持久化层：(dao层)

作用：主要用来操作数据库。

比如向数据库插入一条数据，或从数据库中删除一条数据等

思考：为什么要将项目分出这么多层来呢？

在实际项目开发过程中，一是不确定性因素太多，二是为了提高用户的体验，三是项目后期的优化及代码维护，造成代码改动频繁，尤其像业务逻辑层改动的概率是最大的，那这样划分之后，改动业务逻辑层会影响其它层吗？比如dao层吗？

这样做的目的就是为了解耦和复用。

注意：由于业务逻辑层与持久化层在实际开发中最容易发生变化，所以通常将其定义成页面接口的方式

### 项目结构



### 开发步骤

1. 创建一个动态web工程：

将阶段1工程中，webContent下的所有静态页面拷贝到当前工程中的webContent中

1. 创建数据库bookstore\_0410:
   1. 创建数据表bs\_users:
   2. Sql语句：

Create table bs\_users(

Id int(11) primary key auto\_increment,

Username varchar(20) not null unique,

Password varchar(20) not null,

Email varchar(50)

)

1. 创建User类：
   1. Integer id;
   2. String name; //用户名
   3. String password; //密码
   4. String email; //邮箱
2. 导入jar包：
   1. 导入mysql数据库驱动:mysql-connector-java-5.1.7-bin.jar
   2. 导入数据库连接池：c3p0-0.9.1.2.jar
   3. 引入c3p0的配置文件：c3p0-config.xml 放在src目录中
3. 创建连接池工具类：JDBCUtils.jave
   1. Connetion getConnection()
   2. Void releaseConnection()
4. 创建BaseDao<T>类(专门用于被其它Dao继承)
   1. 使用dbUtil工具实现对数据库表的操作
   2. 导入commons-dbutils-1.3.jar
   3. 定义通用方法：

Int update():对数据表中的数据进行增删改操作

T getBean():获取一个对象

List<T> getAll():获取所有对象

1. 创建UserDao接口及实现类
   1. Int save(User user):保存用户信息
   2. User getUserByusernameAndPassword(User user); 根据用户名与密码获取用户
2. 创建UserService接口及实现类
   1. Boolean regist(User user); 注册用户
   2. User login(User user); 处理用户登录功能
3. 创建RegistServlet 用来处理用户注册请求
4. 创建 LoginServlet 用来处理用户登录请求
5. 修改regist.html页面和 login.html页面
   1. 修改页面路径，将相对路径改为绝对路径(页面中可以使用base标签)
   2. 修改请求提交的服务器路径

## 第三阶段：优化项目

1. 将html文件修改成jsp文件
   1. 办法1：先创建同名jsp文件，再将html中的代码全部复制到jsp文件中，注意不要覆盖page指令标签
   2. 办法2：先将html文件中插入page指令保存，再修改html文件的扩展名为jsp
2. 提取页面中的重复代码部分
3. 完成登录页面与注册页面错误信息提示
4. 完成登录页面与注册页面表单信息的回显
5. 项目优化

问题1：目前是一个功能对应一个servlet，比如一个登录功能对应一个loginServlet，一个注册功能对应一个registServlet，那么我们后面添加图书、删除图书、添加购物车、结账、查询订单等等，也需要一个功能对应一个servlet，这样我们会有非常非常多的servlet出现，那这有没有问题存在呢？

问题2：通常情况在servlet中都会先获取请求参数信息，然后再封装数据，这个过程通常比较繁琐且降低工作效率，如需类型转换问题，比如获取到的都是String类型，而封装数据时却需要int类型，效率会更低，那有没有更好的解决方案呢？

解决方案：

问题1\_思路:

* 我们希望相关的功能封装在一个servlet中，比如与用户相关的登录注册功能我们可以封装在UserServlet中，这样我们维护起来非常方便。
* 具体可以通过传递一个请求参数标识来调用不同的操作，如我们传递method=regist代表调用注册操作，假如我们传递一个method=login代表调用登录操作。但是现在又出现类似之前servlet过多的问题，假若还修改用户、删除用户、查询用户等等，那我们这里的else if又将休止的添加，导致又淹没在if...else if..中
* 我们目前获取的请求参数标识符就是我们定义的方法名，我们可以通过反射的方式来调用不同的操作。注意一点，定义的方法结构必须与doGet或doPost方法结构一样，不然没办法获取到。
* 为实现代码复用，可以将反射部分抽取出来，放在一个BaseServlet类中，专门用来被其它servlet继承

问题2\_思路：

* 可以使用一个工具beanUtils帮我们解决这个问题。需要导包：

commons-beanutils-1.8.0.jar

commons-logging-1.1.1.jar

* 什么是javaBean?
  + javaBean必须有一个无参构造器
  + 属性私有化，并提供相应的getter|setter方法
  + javaBean的属性名指的是getter|setter方法名，去掉get|set后，首字母小写部分。如getName()去掉get后，首字母小写变为name
* 对servlet中获取请求参数部分进行优化，可以通过request.getParameterMap()方法获取一个Map对象。使用BeanUtils.populate()方法实现封装。这样问题就解决了，不再需要手动一个一个获取然后再封装及类型转换问题了。
* 实现代码复用。抽取工具类WEBUtils.params2Bean()获取请求参数封装成bean

## 第四阶段：使用EL表达式替换jsp表达式

* Login.jsp
* Regist.jsp
* base.jsp

## 第五阶段：图书的增删改查

### 第一部分：后台图书的管理（增删改查）



开发步骤：

1. 创建图书表bs\_books

Create table bs\_books(

Id int(11) primary key auto\_increment,

Title varchar(50) not null unique,

Author varchar(20) ,

Price double(11,2),

Sales int(11),

Stocks int(11),

Img\_path varchar(100)

)

1. 创建图书类Book

Integer id;

String title; 书名

String author; 作者

Double price; 价格

Int sales; 销量

Int stocks; 库存

String imgPath; 封面

1. 创建BookDao接口及实现类

Int saveBook(Book book) 保存图书

Int updateBook(Book book) 修改图书

Int deleteBook(String bookId) 根据id删除图书

Book getBookById(String bookId) 根据图书id查找一本图书

List<Book> getBookAll() 查找所有图书

1. 创建BookService接口及实现类

Int saveBook(Book book) 保存图书

Int updateBook(Book book) 修改图书

Int deleteBook(String bookId) 根据id删除图书

Book getBookById(String bookId) 根据图书id查找一本图书

List<Book> getBookAll() 查找所有图书

1. 创建BookManagerServlet
   1. addBook 添加图书请求

* 修改pages/manager/book\_edit.jsp页面中的表单项及提交到服务器的路径、请求方式
* 注意：表单项中的name属性值应与Book类的属性名保持一致

响应路径不可以直接返回pages/manager/book\_manager.jsp而是要重新查图书列表

* 1. deleteBook 删除图书请求
* 修改pages/manager/book\_manager.jsp页面
* 添加js确认删除功能
  1. updateBook 修改图书请求
* 修改pages/manager/book\_edit.jsp页面中的表单项的value属性
* 修改图书根据bookId进行操作。需要在页面中添加一个关于bookId的隐藏表单项
  1. bookList 获取图书列表
* 使用JSTL修改pages/manager/book\_manager.jsp页面
* 注意：需要导入JSTL包

1. 项目优化

当我们将图书数据导入图书表后，我们发现项目中会一次性将所有图书信息加载出来。这样会造成不良后果，一方面是用户体验非常差，另一方面有可能导致系统崩溃。所以在实际开发中一定不要将所有数据一次性全部查出，那怎么办呢？可以采用百度搜索效果，将结果分页分条显示出来。

怎么分条？比如就显示4条图书信息。可以使用如下sql语句：

SELECT id,title,author,price,sales,stocks,img\_path imgPath FROM bs\_books LIMIT 索引,条数

目前又引出新问题，后台仅传递图书列表信息已经不能满足需要，我们需要重新定义一个类来封装页面需要的数据，这个类我们定义成Page类



List<Book>

每页条数

索引

总条数

总页数

当前页

Page<T>：

List<T> list; 保存页面显示的数据,需要从数据库中查出

Int totalRecord ; 总记录数，需要从数据库中查出

Int pageNo ; 当前页，需要用户指定

Int pageSize; 每页条数，需要用户指定

Int totalPage; 总页数，需要计算得出

Int index; 索引，需要计算得出

实现分页步骤：

1. 在BaseDao中添加查询单个数据方法
   1. Object getSingle(String sql,Object ... Params)
2. 在BookDao接口中添加分页方法
   1. Page<Book> findBookList(Page<Book> page)
3. 在BookService接口中添加颁方法
   1. Page<Book> findBookList(String pageNo,String pageSize);
4. 在BookManagerServlet中添加方法
   1. findBookList()
5. 修改pages/manager/book\_manager.jsp
   1. 获取数据
   2. 实现页码跳转功能
   3. 实现分页功能

Question1：完善“上一页”与“下一页”功能？

* 方案1：

在页面中进行处理，如果当前页已经是第1页，可以隐藏“上一页”;

如果当前页已经是最后一页，可以隐藏“下一页”。但此方案并没有彻底解决问题，用户仍然可以通过直接输入地址的方法访问

* 方案2：

在后台控制这个问题，如果当前页已经是第1页，那么点击上一页始终返回第1页;如果当前页已经是最后1页，那么点击下一页始终返回最后一页。不轮点击多少次都是这个结果。我们可以对getPageNo()方法进行处理.

Question2:页码显示过多问题？



* 方案：可以采取百度分页策略

【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

4 5 【6】 7 8

5 6 【7】 8

我们可以通过控制开始与结束页码来实现分页。

当总页数小于5时，开始值为1，结束值为总页数

当当前页小于3时，开始值为1，结束值为5

其它情况，开始值为当前页-2，结束值为当前页+2

当第三种情况，结束值大于总页数时，开始值为总页数-4，结束值为总页数

1. 提取分页代码为page.jsp。使用静态包含指令引入页面

由于路径是变化的，我们可以在page类中定义一个path属性来保存

### 第二部分：前台图书的显示

前台页面的显示不需要重新写Dao层与service层，可以直接调用后台部分

我们只需要创建BookClientServlet来处理客户端的一些操作就可以啦！

一、前台页面显示操作步骤：

1. 创建BookClientServlet
   1. findBookList() 处理前台页面显示
2. 程序启动需要先访问BookClientServlet获取需要数据，所以需要修改目前的前台显示结构
   1. 将index.jsp移动pages/user/目录中改名home.jsp
   2. 在项目根目录下新创建index.jsp ，要求在index.jsp中通过jsp:forword转发到client/BookClientServlet?method=findBookList
3. 带条件分页查询图书操作
4. 在BookDao接口中创建带查询条件的分页方法：

Page<Book> findBookListByPrice(Page<Book>,int min,int max);

2. 在BookService接口中创建方法：

Page<Book> findBookListByPrice(String pageNoStr,String pageSizeStr,String minStr,String maxStr);

1. 在BookClientServlet中创建方法：

findBookListByprice()

1. 修改home.jsp页面

将页面中输入价格部分修改为form表单的形式

使用post请求有问题，需要修改为get提交方式

同时定义一个隐藏表单项：<input type=*"hidden"* name=*"method"* value=*"findBookListByPrice"*>

## 第六阶段：登录登出、验证码、购物车

### 第一部分：登录登出

* 怎么判断用户是否登录登出？可以将用户信息放入session域中，如果session域中有user信息说明登录;如果没有，说明登出
* 用户登出操作？可以直接让session失效

### 第二部分：验证码

验证码和我们的token指令的原理是一样的，由服务器创建字符串，然后一方面设置进session中，另一方面交给客户端，最终的目的是为了防止用户重复提交表单。不同的是，我们这里的验证码它是以图片形式显示的，需要用户手动输入字符。

验证码流程：

由服务器创建验证码字符串，并将字符串保存到session中

用户根据图片提示输入字符串

将图片设置进页面显示

将字符串转成图片

不一致

一致

表单重复提交不处理

进行表单注册

在Servlet中，获取用户提交的验证字符串，并从session中获取验证字符串进行比较

将字符串转成图片？我们可以使用第三方工具：kaptcha-2.3.2.jar

步骤：

1. 导包：kaptcha-2.3.2.jar
2. 在web.xml中进行配置

<servlet>

<servlet-name>KaptchaServlet</servlet-name>

<servlet-class>com.google.code.kaptcha.servlet.KaptchaServlet</servlet-class>

<init-param>

<param-name>kaptcha.session.key</param-name>

<param-value>code</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>KaptchaServlet</servlet-name>

<url-pattern>/code.jsp</url-pattern>

</servlet-mapping>

1. 在regist.jsp中引入验证图片
2. 修改client/UserServlet中的regist()方法

### 第三部分：购物车

* 购物车就是暂时保存图书的
* 实现购物车的方式：

1. 基于cookie的，显然这种方式已经过时，不使用
2. 基于session，不同浏览器间是不能共享的
3. 基于数据库，不同浏览器乃至不同设备都可以实现数据的共享



总金额

总数据

cartItem

CartItem类：

Book book ; 保存图书

Int count; 单品种数量

Double amount; 单品种金额

Cart类：

Map<String,CartItem> map; 定义购物项

Int totalCount; 总数量

Int totalAmount; 总金额

方法：

Void deleteBook(String bookId); 根据图书id删除单品种图书

Void clear() ; 清空购物车

addBook2Cart(Book book) ; 添加图书到购物车

updateCount(String bookId,int count); 修改指定图书的数量

* 实现购物车cart.jsp页面展示
* 实现添加图书到购物车

1. 创建CartServlet:

addBook2Cart()

1. 修改home.jsp页面

* 实现购物车中的删除操作

1. 在CartServlet中：
   1. deleteCartItem()
2. 修改cart.jsp页面删除链接
3. 在cart.jsp页面中对删除链接做再次确认js代码

* 清空购物车操作

1. 在CartServlet中：
   1. clear()

2.修改cart.jsp页面删除链接

* 修改购物项数量

1. 在CartServlet中：
   1. updateCount()
2. 修改cart.jsp页面

将文本修改为单行文本框，并绑定更改事件

## 第七阶段：结账、查询订单及订单详情



BookDao

OrderItemDao

OrderDao

访问OrderService的saveOrder()方法：

1. 生成订单号，并返回
2. 保存订单及订单详情
3. 修改图书表中库存和销量
4. 清空购物车

当点击“去结账”请求会发送给OrderServlet的checkout()方法:

1. 获取user
2. 判断user是否登录，没有登录去login.jsp，否则继续
3. 获取购物车cart，调用OrderService的saveOrder()
4. 返回订单生成页面checkout.jsp

结账步骤：

1. 创建Order类与OrderItem类

Order类：

String id; //订单号

Int totalCount ; 购物总数量

Double totalAmoutn ；购物总金额

Date createDate; 结账日期

Int state; //保存状态 0未发货，1已发货，2交易完成

int userId;

用户与订单的关系：1对多

订单号的特点：

1. 用来进行售后服务，易识别
2. 它有唯一性
3. 要尽可能少透漏商业信息

订单规则：System.currentTimeMillis()+””+userId

OrderItem类：

Int id;

String title;

String author;

Double price;

Int stocks;

Int sales;

String imgPath;

Int count;

Double amount;

String orderId;

订单项与订单关系：1对多

1. 创建bs\_orders表与bs\_order\_items表

Create table bs\_orders(

Id varchar(100) primary key,

Total\_count int(11),

Total\_amount double(11,2),

Create\_date datetime,

State int(2),

User\_Id int(11) ,

Foreign key (user\_id) references bs\_users(id)

)

Create table bs\_order\_items(

Id int(11) primary key auto\_increment,

Title varchar(20),

Author varchar(20),

Price double(11,2),

Sales Int(11),

Stocks Int(11),

Img\_path varchar(200),

Count Int(11),

Amount double(11,2),

Order\_id varchar(100),

Foreign key (order\_id) references bs\_orders(id)

)

1. 创建OrderDao接口及实现类
   1. Int saveOrder(Order order); //保存订单
   2. Int updateState(String orderId,int state) ; 修改订单状态
   3. List<order> getOrderList() ; 查询所有订单，提供给管理员使用
   4. List<order> getOrderListByUserId(int userid) ; 查询当前用户所有的订单，提供给用户使用
2. 创建OrderItemDao接口及实现类
   1. Int saveOrderItem(OrderItem item) ; 保存订单项
   2. List<orderItem> getInfo(String orderId); 根据订单id查询所有的订单项
3. 生成订单
   1. 创建OrderService接口及实现类

String saveOrder(Cart cart,User user); //保存订单

* 1. 创建OrderClientServlet

Checkout() //处理订单请求

* 1. 修改cart.jsp页面链接

1. 用户查看订单
   1. 在OrderService中创建方法：

List<Order> getOrderListByUserId(int UserId);查看当前用户的订单

* 1. 在OrderClientServlet中创建方法：

orderList() 处理当前用户查询订单请求

* 1. 修改order.jsp页面展示订单

1. 管理员查看订单
   1. 在OrderService中创建方法：

List<Order> getOrderList();查看系统所有订单

* 1. 创建OrderManagerServlet

OrderList(); 查找所有订单信息

* 1. 修改order\_manager.jsp展示订单

1. 修改订单的状态
   1. 在OrderService中创建方法：

Void send(String orderId);管理员发货，将状态设置为1

Void take(String orderId); 用户收货，状态设置为2

* 1. 在OrderManagerServlet

send(); 发货

* 1. 在OrderClientServlet

take(); 收货

* 1. 修改order\_manager.jsp链接发货
  2. 修改order.jsp链接收货

1. 查看订单详情
   1. 在orderService中创建方法：

List<OrderItem> getInfo(String orderId); 根据订单号查看订单详情

* 1. 在OrderManagerServlet

Info();查询订单详情

* 1. 在OrderClientServlet

Info();查询订单详情

* 1. 修改order\_manager.jsp链接查询订单详情
  2. 修改order.jsp链接查询订单详情

1. 代码优化

在orderService类的saveOrder()方法中，每遍历一次购物项就会操作订单项表与图书表各一次，即获取数据库连接和关闭数据库连接各2次，如果要遍历10次，就需要获取数据库连接20次，关闭数据库连接20次，而获取数据库连接是比较耗费资源降低性能的。通过观察我们发现所使用的sql语句是一样的，不同的是每次遍历的参数不同，这样我们可以采用批量方式对此进行优化。

* 在BaseDao类中添加批量方法：

Void batch(String sql ,Object[][] Params)

* 在BookDao接口中添加修改库存与销量的方法：

Void batchupdateSaleAndStock(int sale,int stock,int bookId);

* 在OrderItemDao接口中添加批量保存方法：

Void batchSaveOrderItem(Object[][] param)

* 对orderService类中的saveOrder进行优化

## 第八阶段：在项目中引入Filter

### 第一部分：用户登录验证

原则上orderClientServlet中的方法被访问，需要用户必须登录之后才可访问;而目前我们是在每一个方法中对用户是否登录作验证，这样带来代码冗余。我们希望有那么一个技术可以对用户是否登录作统一验证，怎么做？可能通过Filter来实现

步骤：

1. 定义一个LoginFilter的过滤器，专门过滤向/client/OrderClientServlet发送的请求
2. 在LoginFilter中检验用户是否登录，如果没有去登录页面，否则放行

### 第二部分：结账事务处理

场景：当保存订单的sql语句写错时，由于不能生成正确的订单，导致订单项也不能正确保存，但是图书的库存和销量却改变了。另外像保存订单，保存订单项及修改图书的库存和销量属于同一个业务，我们希望要么都成功，要么都失败。所以这里要使用到事务。

事务的常规格式：

Connection conn = null;

try{

//取消事务自动提交行为

conn.setAutoCommit(false);

//操作数据库

。。。。

//没有异常，手动提交事务

conn.commit();

}catch(Exception e){

//出现异常，回滚事务

conn.rollback();

}finally{

//关闭连接

conn.close();

}

问题1 ：我们这段代码写在哪里？因为要保证所有的事务操作要写在同一个try代码块中，加到Dao层、service层、servlet层都不合适，所以我们可以开辟一个新的领域，创建一个处理事务的过滤器TransactionFilter.

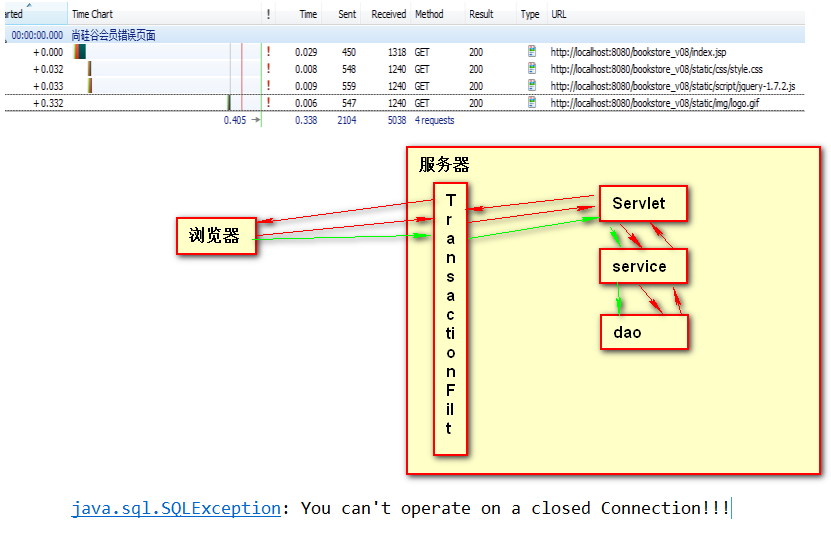
问题2：这里要求我们使用的Connection对象必须是同一个，而在我们的BaseDao中每个方法是单独获取Connection对象，导致我们需要对整个BaseDao进行大篇幅修改，将Connection对象以参数的形式传进来，不仅如此service层及servlet层都需要进行修改。这种方式不可取，即然目的是要保证connection对象是同一个，我们可以将其设计成单例模式。

过程1：并没有解决事务问题，为什么呢？我们在BaseDao层将异常捕获并处理了，所以这里应不应该在BaseDao中处理啊？不应该在这里处理而是统一传递到TransactionFilter中统一处理。

过程2：好像是解决问题了，但是却报出java.lang.reflect.InvocationTargetException。这个异常是通过反射调用目标方法时出现的异常。经查看BaseServlet对异常进行了捕获并拦截了下来，我们希望这里也向上抛出到TransactionFilter中统一处理

过程3：返回的是一个空白页面，我们可以定义一错误页面进行展示

过程4：当打开关闭连接之后，报出java.sql.SQLException: You can't operate on a closed Connection!!!



问题的关键是我们将connection对象设计成了单例模式，想一想这个问题是不是非常可怕，在实际开发中，connection对象坚决不能设计成单例模式。

经过分析，一个线程对应一个connection对象，同一时刻会有多个connection对象，所以我们可以定义一个Map来保存connection,键就是当前线程，值就是connection对象。Java中给我们提供了一个线程安全的ConcurrentHashMap.同时java还给我们提供了一个类ThreadLocal来简化ConcurrentHashMap.

## 第九阶段：在项目中引入AJAX

### 首页的加入购物车功能可以引入AJAX

* 修改CartServlet中的addBook2Cart方法
* 修改home.jso页面给button按钮绑定单击事件

//给按钮绑定单击事件

$("button").click(function(){

//准备请求参数

//client/CartServlet?method=addBook2Cart&bookId=${book.id }

var bookId = this.id;

var url = "client/CartServlet?method=addBook2Cart&bookId="+bookId;

//向服务器发送请求

$.get(url,function(data){

if(data){

$(".info").empty().append("<span>您的购物车中有"+data.totalCount+"件商品</span><br>")

.append("您刚刚将<span style='color: red'>"+data.title+"</span>加入到了购物车中");

}

},"json");

});

### 2.购物车页面可以使用AJAX

* 修改cart.jsp页面中单品种图书的数量文本框的改变事件。
* 修改CartServlet中updateCount()

### 注册页面可以使用AJAX来检查用户名是否存在

步骤：

1. 在userDao中新添方法并实现：User getUserByUsername(String username)
2. 在userService中添加方法并实现 ：boolean checkUsername(String username)
3. 在UserServlet中添加方法: checkUsername() //处理用户名是否可用请求
4. 在regist.jsp页面中对用户名文本框绑定改变事件

$("[name=username]").change(**function**(){

//准备参数

**var** username = **this**.value;

//发送请求

**var** url="${pageContext.request.contextPath}/client/UserServlet?method=checkUserName&username="+username;

$.get(url,**function**(data){

//获取响应，设置信息

**if**(data.flag){

$(".errorMsg").html("此用户名可以使用");

$("#sub\_btn").attr("disabled",**false**);

}**else**{

$(".errorMsg").html("此用户名已被占用");

$("#sub\_btn").attr("disabled",**true**);

}

},"json");

})

## 第十阶段：修改图书封面

* 对book\_edit.jsp进行修改。
* 添加file表单项
* 修改form标签的enctype属性
* 对BookManagerServlet中addBook()进行修改处理
* 导入commons-fileupload和commons-io包
* 进行解析