# 第一阶段 登陆注册表单验证

jquery + 正则表达式

# 第二阶段 登陆注册功能

## 软件架构

表示层

直接与用户交互

用户界面以及交互

软件

数据库访问层

业务逻辑层

业务逻辑（代码）

分层的目的---》分模块—》解耦

1、表示层：和用户直接交互

1）、HTML页面，Servlet。

2、业务逻辑层：处理网站的业务功能

2）、编写的业务逻辑代码。

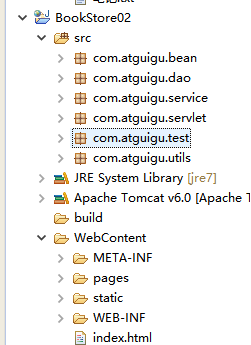
Service层：UserService login()

3）、数据库访问层：（持久化层）

Dao层：把所有操作数据库的类我们可以放在Dao层

模型层：创建与数据库数据对应的对象

## 项目架构



Dao层

操作数据库

Service层

业务代码

Servlet层

处理请求

HTML页面

WebContent

Bean层

所有的JavaBean，对象模型

1、用户查看HTML页面，发送请求（登陆、注册）

2、Servlet层的某个Servlet收到用户请求

3、调用Service层相关方法来处理

4、Service层可以直接调用Dao层来和数据库沟通

5、Dao层将查询到的数据封装为对象，系统以后就是使用对象就可以。

## 开发项目

1）、需求分析，建立模型

1、建立登陆注册相关模型（Bean层）

1-建立用户表

当数据表很多的时候，也需要分层，一般按照命名规则分层即可

auth\_menu

sys\_user

bbs\_content

bs\_user：

主键：自增

username:varchar

password:varchar

email:varchar

创建数据库和表

CREATE DATABASE bookstore\_0407;

CREATE TABLE bs\_user(

id INT(11) PRIMARY KEY AUTO\_INCREMENT,

username VARCHAR(100) NOT NULL UNIQUE,

PASSWORD VARCHAR(100) NOT NULL,

email VARCHAR(100)

)

2-建立对应的User类

**public** **class** User {

**private** Integer id;

**private** String username;

**private** String password;

**private** String email;

2、建立Dao层：操作数据库

1-JDBCUtils。用来获取数据库连接。使用c3p0获取连接

1、导包mysql-connector-java-5.1.7-bin.jar、c3p0-0.9.1.2.jar、commons-dbutils-1.3.jar

建立BaseDao，专门用来被其他的dao继承，BaseDao定义一些基本的操作

BaseDao<T>:

public T getBean()

public List<T> getBeanList()

public int update()

2-UserDao:操作User表的Dao类

面向接口编程

定义一个UserDao接口；定义UserDao有什么方法

UserDao接口：

User getUserByUserNameAndPassword(User user);

boolean registUser(User user);

3、建立Service层

1-UserService：主要就是完成用户的登陆注册功能

面向接口编程

UserService接口：

User login(User user);

boolean regist(User users);

4、建立Servlet层：处理用户的请求

1-LoginServlet：处理登陆请求

2-RegistServlet：处理注册请求

5、界面

修改响应的逻辑 html里面使用base标签

Servlet层

LoginServlet

RegistServlet

HTML页面

login.html

regist.html

Service层

UserService

用户登陆

用户注册

Dao层

UserDao

按照用户名密码查找用户

保存用户

Bean层

User

bs\_user

# 第三阶段 项目优化

## 1、将所有的HTML页面改为jsp

很多页面都有重复信息。

特别是操作菜单。我们也可以抽取出来。

1）、jsp页面和HTML就差了一个Page指令头

2）、相关链接也必须改

3）、引入页面（我们将相关的菜单抽取出来并引入）

<%@include file=”菜单的路径”%>

4）、在页面上，相对路径不靠谱，需要使用绝对路径

由于页面相对路径基本都写好了，所以我们可以使用Base标签给所有相对路径一个基准路径，所有页面要引入css文件，还要引入jquery文件。

我们将这些也可以抽取出来，再每个页面引入

base的href属性修改为动态获取

request.getScheme()：获取协议名

request.getServerName()：获取服务器名

request.getServerPort()：获取服务器端口

request.getContextPath()：获取项目路径

base标签，因为转发的情况下，可能导致页面上的相对路径失效。

<http://localhost:8080/LoginServlet---success.html>(所有相对路径可能失效)

写绝对路径可以保证不失效（绝对从服务器根开始，所以绝对路径加项目名）

base标签。为了让页面上的其他相对路径不受当前资源影响，受base指定的路径影响

<http://localhost:8080/BookStore02/> 所有的相对路径都是基于base指定的

所以页面上的相对路径解析出来以后<http://localhost:8080/BookStore02/>js/jquery.js

## 2、登陆，注册，如果发生错误应该有错误信息提示

### 错误提示：（登陆）

1）、用户输入用户名密码

2）、发送登陆请求

3）、登陆Servlet来处理登陆请求—》查用户

成功3.1）、重定向到页面

失败 3.2）、转发到登陆页面

4）、又会在浏览器上显示登陆页面。想要在登陆页面动态获取错误信息

在3.2转发之前，设置一个错误消息提示，把他放在域中。jsp就可以在域中取数据

### 错误提示：（注册）

1）、用户输入用户名密码其他信息

2）、发送注册请求

3）、注册Servlet来处理注册请求—》注册用户

成功3.1）、重定向到页面

失败 3.2）、转发到注册页面

4）、又会在浏览器上显示注册页面。想要在注册页面动态获取错误信息

在3.2转发之前，设置一个错误消息提示，把他放在域中。jsp就可以在域中取数据

## 3、表单回显

当登陆或者注册不成功的时候，回到登陆注册页面我们希望将之前信息保留（表单回显），方便我们的操作。提升用户体验。

### 登陆回显

1）、用户输入用户名密码（登陆回显）

2）、发送登陆请求

3）、登陆Servlet来处理登陆请求—》查用户

成功3.1）、重定向到页面

失败 3.2）、转发到登陆页面

4）、又会在浏览器上显示登陆页面。想要在登陆页面动态获取之前输入的值

在3.2转发之前，

1）、可以将获取到的数据放在request域中，再页面获取

request.setAttribute(“username”,username);

request.setAttribute(“password”,password);

之上不推荐

2）、直接在页面使用

request.getParameter("username")

### 注册回显

## 4、优化Servlet的逻辑

一个请求对应了一个Servlet。

UserService---》login(),regist(),xx();和用户相关的业务逻辑都放在了UserService。

能不能和用户相关的请求（登陆，注册,…）放在一个Servlet里面。

和用户相关的请求 UserServlet

和图书相关的请求 BookServlet

…..

分析：

注册，登陆 都被UserServlet 🡪所以注册登陆都请求同一个地址

可以为请求添加一个method字段，提交的时候，带上method值。

if(method.eqauls(“regist”))调用注册方法

if(method.eqauls(“login”))调用注册方法

问题：

上面看起好用了。

get请求，表单的数据会被带上，而之前请求地址中的数据会被表单数据覆盖

添加一个隐藏的表单项

<input type="hidden" name="method" value="login">

为了不陷入层层的if判断，我们利用反射来做这个事情

所有的Servlet都是通过反射来调用相应的方法，所以我们可以抽取出一个BaseServlet

String method = request.getParameter("method");

System.*out*.println(method);

// getDeclaredMethod(方法名，参数列表)

**try** {

Method method2 = **this**.getClass().getDeclaredMethod(method,

HttpServletRequest.**class**, HttpServletResponse.**class**);

//把方法权限设大

method2.setAccessible(**true**);

//invoke(对象，参数);

method2.invoke(**this**, request,response);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

String method = request.getParameter("method");

获取请求参数：

表单提交的：key-value

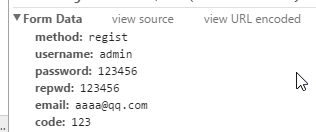
url地址后面的：

http://localhost:8080/BookStore03/UserServlet?method=login&username=admin&password=123

?前面是请求地址，？后面是请求参数

对于get请求请求request.getParameter是获取？后面的这些数据。

post请求：参数是带在请求体中。可以查看请求报文中的数据



上面是浏览器格式化后的。

元数据

method=regist&username=admin&password=123456&repwd=123456&email=aaaa%40qq.com&code=123

<a href=” http://localhost:8080/BookStore03/UserServlet?username=abc&pwd=123”>链接 </a>

## 5、参数封装优化

我们使用分层架构

Servlet -🡪 Service -🡪 Dao 🡪 bean: 在系统中数据都使用对象包装

注册：第一步：来到注册Servlet，获取数据封装对象

登陆：第一步：来到登陆Servlet，获取数据封装对象

xxx: 第一步，来到xxxServlet，获取数据封装对象

我们应该将获取数据，封装对象这个过程抽取出来。

因为封装对象，还牵扯到数据转型，很麻烦。。。

User user = Utils.param2Bean(t);

我们希望抽取为一个自动化的工具。根据我的需求，自动的封装对象。

能智能的帮我们封装数据。

反射…..

request里面带了好多参数…..，根据要封装的对象获取相应的参数进行封装

beanutils有丰富的操作javaBean的方法。

commons-beanutils-1.8.0.jar、commons-logging-1.1.1.jar

String username = request.getParameter("username");

String password = request.getParameter("password");

String email = request.getParameter("email");

由于request获取的数据都是String类型的，要封装成Bean的话，需要类型转换。

JavaBean的属性值的设置是按照getter setter方法定的

属性的真正的名字是get或者set后面的字符串

将map中的属性映射到JavaBean中

BeanUtils.populate(t, map);

为JavaBean的某个属性赋值

BeanUtils.setProperty(t, name, value);

# 第四阶段 使用El进行错误消息提示以及表单回显

以前都是使用jsp表达式

el🡪获取数据方便。如果没有属性就是什么都没有””。不用做三元运算

# 第五阶段 图书后台管理的增删改查以及前端显示

## 1、后端的图书管理功能

1）、显示层：页面

2）、控制层：Servlet。

3）、业务逻辑层：Service。

4）、持久化层：Dao

5）、模型层：Bean

6）、数据库对应的表；

实现步骤

6）、创建图书相关的表。哪些字段？



主键 自增 id

书名 varchar title

作者 varchar author

价格 double(11,2) price

销量 int sales

库存 int stock

封面图片 varchar img\_path

只保存图书封面的图片路径，绝对路径，base标签。保存图片相对于当前项目的路径 <img src=””>

CREATE TABLE bs\_book(

id INT(11) PRIMARY KEY AUTO\_INCREMENT,

title VARCHAR(100) NOT NULL,

author VARCHAR(100) NOT NULL,

price DOUBLE(11,2),

sales INT(11),

stock INT(11),

img\_path VARCHAR(200)

)

5）、对应的javaBean

book

Integer id;

String title;

String author;

double price;

int sales;

int stock;

String imgPath;

4）、开发数据库层

面向接口开发。

BookDao

List<Book> getAllBook() 显示所有图书

boolean addBook(Book book);增加图书

boolean delBook(Book book);删除图书

boolean updateBook(Book book);修改图书

Book getBook(Book book);找一本图书

BookDaoimpl

BookDaoimpl extends BaseDao<Book> implements BookDao

3）、业务逻辑层

增删改查 面向接口编程

添加图书

修改图书

删除图书

图书列表

BookService

add(Book book);添加图书

update(Book book);修改图书

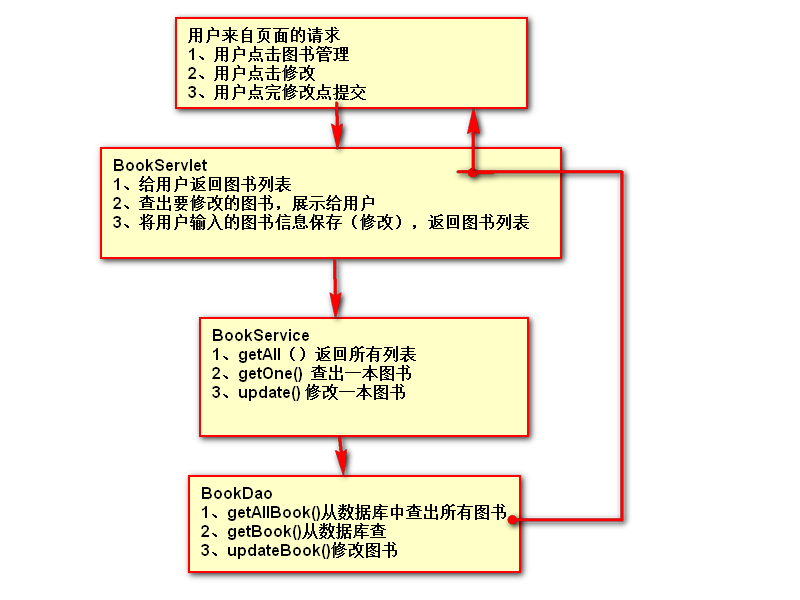
delete(Book book);删除图书

getOne(Book book);获取一本图书

getAll();获取所有图书

BookServiceImpl

2）、Servlet，控制器Controller，用来控制页面转发



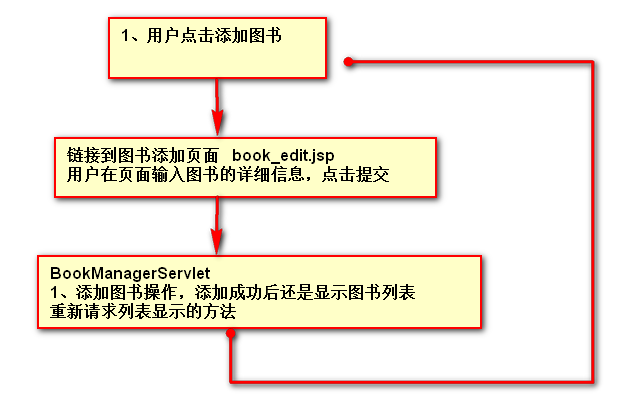
1、用户点击图书管理，显示所有的图书

点击图书管理—》交给Servlet---》查出数据---》交给页面显示

BookManagerServlet

list（）处理显示图书列表的请求

2、添加图书



3、删除图书  
 用户点击删除---》Servlet接受请求---》按照图书id删除图书---》还是回到图书列表页面

删除需要提示用户。

删除请求响应的servlet方法。

4、修改图书

1）、用户点击修改，先要去Servlet，查出要修改的图书详情，显示到页面。

问题：修改和添加都是一个页面，都指向同一个方法。点击提交。现在是添加

解决办法：

去编辑页面的时候带上参数m=update,m=add，在编辑页面取出参数即可

解决办法2：

同样都是在编辑页面点击提交。

method=add

将提交过来的数据封装

判断是否有id值，如果有id—》修改操作

没有id—》添加操作

5、问题存在：

现在所有的图书都在列表里面。

1）、一次查找所有的图书，页面很乱

2）、加重数据库负担

分页的显示。

一次取部分数据。而不是全部数据。

分页的原理：limit

select \* from bs\_book limit 起始的索引,要取出多少个。

分页：



作用：

显示部分信息。减轻服务器负担。

做分页功能：

分页的原理：limit

select \* from bs\_book limit 起始的索引,要取出多少个。

在管理员图书列表页面，显示分页数据。

显示层。

Servlet控制层。BookManagerServlet -🡪list

Service 层。 查出部分图书（分页要求的。1,2）

Dao层。 查出部分图书（分页查询 limit index,size）

Bean层。 图书。需要一个分页模型（分装当前分页信息，当前第几页，总页数，。。。）。

分页模型Page select \* from bs\_book limit index,size



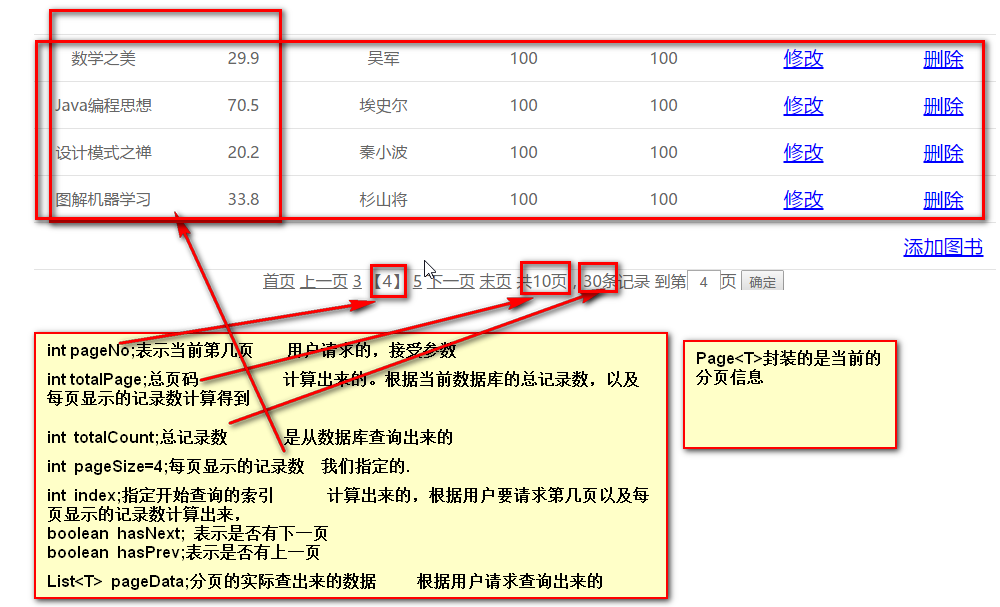
当前是第几页 int pageNo;

总页码 int totalPage;

总记录数 int totalCount;

每页显示多少条记录 int pageSize;

数据库查询的索引 int index;



分页模型；

1、创建Bean层的分页模型

2、dao层,应该返回原始数据

List<Book> getPageList(int index,int size);

获取总记录数的方法 int getTotalCount();

3、service层。

getPage(String pageNo,String pageSize)；特别注意，封装pageNo,pageSize,totalCount的顺序

先pageSize,totalCount 🡪 然后pageNo

4、Servlet层

第一请求显示图书列表，应该只第一页数据。

list();

page();用户请求的分页方法。



显示过多页码。

。。。。。

相当于把当前页码的前两页和当前页码的后两页显示出来

分情况来确定begin-end值

1、总页码五页以内，全部显示 begin=1 end=totalPage

1 2 3

2、总页码五页以上

优化。连续显示五页

当前1 【1】 2 3 4 5

当前2 1 【2】 3 4 5

当前3 1 2 【3】 4 5

当前4 2 3 【4】 5 6

1、当前页码小于3 显示1-5

2、当前页码如果接近尾页 显示5页，显示后五页。总页码-4 10

6 7【8】 9 10

6 7 8 【9】 10

6 7 8 9 【10】

## 2、图书前端的显示

显示层

控制器BookClientServlet -🡪page

service bookservice 🡪 getPage();

dao

bean

分页抽取

带条件的查询。



点击查询-🡪通过Servlet处理-🡪按照要求查询并返回页面显示

显示层

控制器BookClientServlet -🡪page,只是简单分页方法。基于原分页之上，带条件查询

service bookservice 🡪 getPage();--🡪带上查询条件查询

dao -🡪编写一个带条件查询的方法（价格区间）v

bean

# 第六阶段 登陆、登出（注销）、验证码、购物车

## 1、登陆

怎么实现？

将登陆的用户保存到session中，方便在其他资源共享。

在页面直接从session取出用户的详细信息

## 2、登出

用户退出系统，

直接将session设为不可用。

模型；

dao；

service；

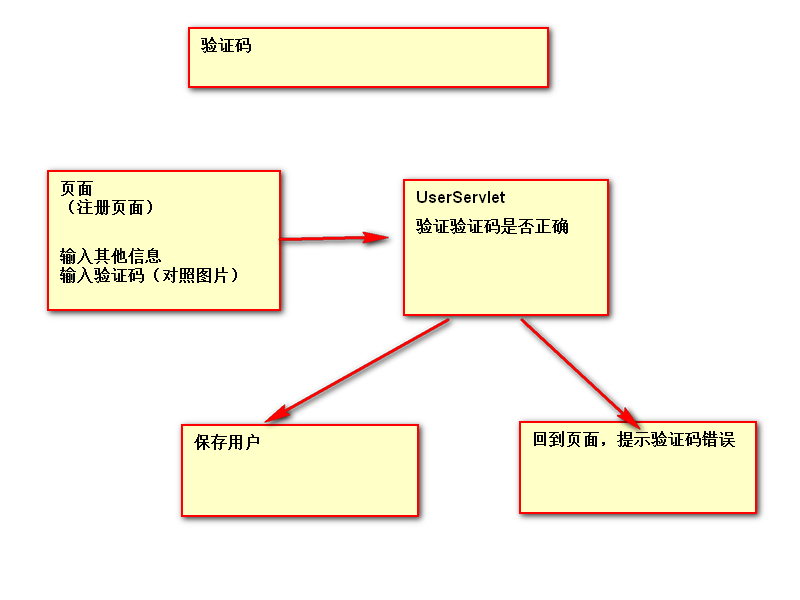
servlet；在servlet将session清空即可

logout

根据session中是否有user对象就可以知道用户是否已经登陆。

## 3、验证码

防止多次恶意请求。



和token机制一模一样

在页面生成验证码图片-🡪给session中保存验证码实际的内容-🡪用户提交表单带上验证码-🡪取出session中的内容以及页面带来的内容（验证码）进行匹配-🡪成功则提交-🡪否则重回注册页面

<img src=”请求kaptcha”>

## 4、购物车

购物车：

暂存物品信息。

实现：

1）、基于cookie，把购买的物品，都保存成cookie存于浏览器。

cookie--🡪都会携带。大量的数据被携带。

2）、基于session，把购买的物品，保存到session中（服务器）。浏览器关闭丢失购物车数据。v

3）、基于数据库，把购买的物品保存到数据库中。

模型层：



方法：

void addBook2Cart(Book book);//在页面直接传入图书id；

void deleteItem(String bookid);//删除一个购物项

List<CartItem> getAllItems();//获取所有购物项，把这个交给页面

void updateCount(String bookid,int count);//修改某一项的数量,传入修改后的数量

void clear();//清空购物车

double,float在进行运算的时候会产生精度问题。

1、大整数运算

2、浮点数运算

控制器：

添加图书到购物车：

用户在页面点击添加购物车，将图书id提交过去。在购物车中保存相应的条目

返回首页

删除某个购物项：

修改

Service：不用写

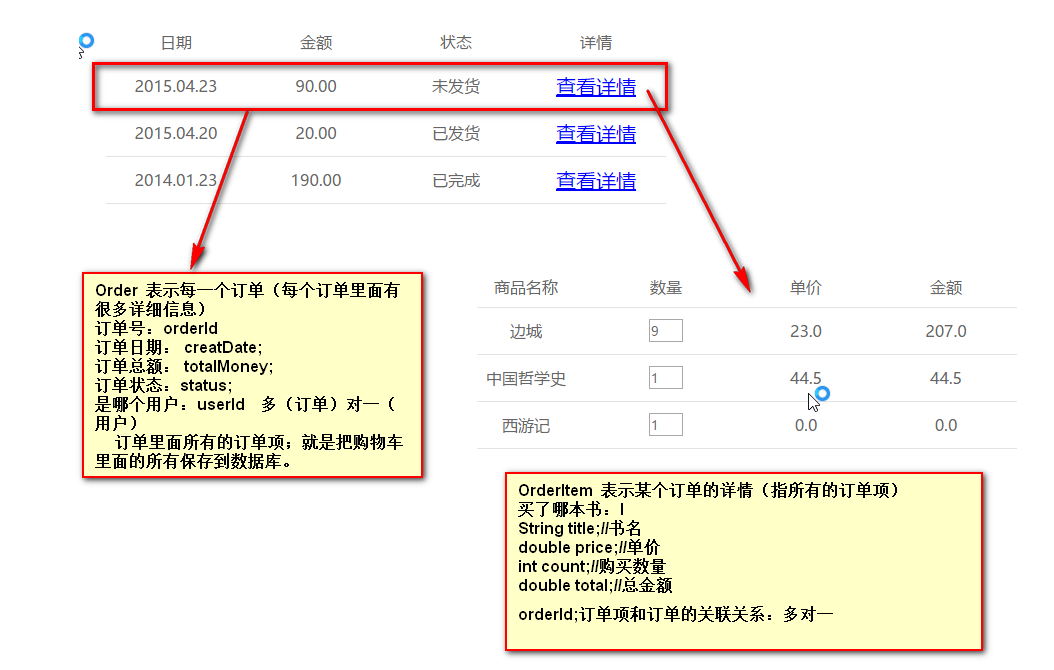
Dao；不用写

# 第七阶段 订单结算

## 1、结算逻辑

## 2、抽取模型

目的：在页面显示，在数据库保存



**Order类**

String orderId;订单号；//作为主键

UUID.randomString():不能用来做订单号。

订单号应该是一个有意义的标识。还必须唯一

时间戳+用户id：保证订单的唯一性

Date createDate; //订单日期

double totalMoney;//总金额

int status; 0:未发货 //订单状态

1：已发货

2：交易完成

Integer userId;//关联的用户，一个用户有很多订单，外键关联

**OrderItem**

**保存当时购买的图书的信息**

String title;//购买的书名

double price;//图书的价格

int count;//购买的数量

double total;//金额

String orderId;//关联的订单

建表

CREATE TABLE bs\_order(

order\_id CHAR(50) PRIMARY KEY,

create\_date DATETIME,

total\_money DOUBLE(11,2),

STATUS INT(2),

user\_id INT(11),

FOREIGN KEY (user\_id) REFERENCES bs\_user(id)

)

CREATE TABLE bs\_order\_item(

id INT(11) PRIMARY KEY AUTO\_INCREMENT,

title VARCHAR(200),

COUNT INT(11),

price DOUBLE(11,2),

total\_price DOUBLE(11,2),

order\_id CHAR(50),

FOREIGN KEY (order\_id) REFERENCES bs\_order(order\_id)

)

## 3、编写dao

OrderDao//操作订单

int saveOrder(Order order);//保存订单

int updateStatus(Order order);//修改订单状态

List<Order> getOrderList();//查出所有订单（管理员）

List<Order> getOrderByUserId(Integer userid);//查出某个用户的所有订单

OrderItemDao//

getOrderItems(String orderId);//根据订单号查询订单所有项

saveItems(OrderItem item);//保存订单项

## 4、编写service

OrderService;订单操作

String checkout(Cart cart);//(结账)保存订单,返回订单号

1、生成订单号

2、订单详情（总价，日期，状态）

3、保存订单项

4、修改库存及销量

update(String orderId,String status)//修改订单状态

getAllOrder();//管理员

getMyOrders(Integer userid);//获取某个用户的订单

OrderItemService;

getOrderItems(String orderid);//根据订单号查出所有的订单项

saveItem(OrderItem item);//保存某个订单项

## 5、编写servlet

OrderClientServlet

用户：

结算：

1、判定用户是否登陆

2、登陆才结算，否则返回登陆页面

查询自己的所有订单：

点击我的订单-🡪订单列表

查看订单详情：

确认收货:

OrderManagerServlet

管理员：

查看所有订单

点击订单管理🡪OrderManagerServlet查出所有订单-🡪返回到order\_manager.jsp进行显示

在页面对未发货的订单，可以点击发货

发货：

点击发货—》将对应的订单状态改为已发货-🡪回到列表页面

优化，我们将保存订单项应该作为批处理；

批量全部处理。

每保存一个订单项就执行

"insert into bs\_order\_item(title,count,price,total\_price,order\_id) values(?,?,?,?,?)";

"insert into bs\_order\_item(title,count,price,total\_price,order\_id) values(?,?,?,?,?)";

"insert into bs\_order\_item(title,count,price,total\_price,order\_id) values(?,?,?,?,?)";

因为BaseDao 🡪 update方法每次获取连接关闭连接…很浪费时间和资源。

我们希望将这些所有的sql攒起来一起执行。🡪批处理

在BaseDao中添加批处理方法；

String sql="insert into bs\_order\_item(title,count,price,"

+ "total\_price,order\_id) "

+ "values(?,?,?,?,?)";

//二维数组，第一维长度：代表sql执行几次

//第二维：存储数据，代表每个sql需要用到的参数。

Object[][] objs = **new** Object[params.size()][5];

//objs[0] = new Object[]{};

**int** count = 0;

**for** (OrderItem item : params) {

objs[count++] = **new** Object[]{item.getTitle(),

item.getCount(),item.getPrice(),

item.getTotalPrice(),item.getOrderId()};

}

batch(sql, objs);

# 第八阶段 优化（解决bug），引入filter

1）、销量和库存问题；

原因，我们要修改图书的库存，以及销量。从CartItem拿到的Book对象并不能代表图书现在的库存和销量。只代表放的那一刻的库存和销量。

解决：从数据库获取图书实际的库存和销量，再进行修改

2）、结账后购物车应该清空

3）、只要是用户订单的操作，应该先登陆。

我们只在结账验证了登陆。

使用filter验证登陆。关于订单的操作

事务：

对数据库一系列操作之后，才能完成一项功能。

只要一项失败，则整个功能失败，希望数据库不要有影响。

1、a表保存了几条数据 2、b表保存数据 3、c表修改了一些数据 4、d表查询一些数据。

只要任何一步失败，我们都应该回滚。。。

保持事务，最重要的是要使用同一个连接Connnection

connection.setAutoCommit(false);

1、a表保存了几条数据 2、b表保存数据 3、c表修改了一些数据 4、d表查询一些数据。

connection.rollback();

回滚机制：

禁止自动提交。改为手动提交。有错误则回滚。

问题演示：

可以将OrderItemDaoImpl. saveBatch sql语句写错

现象：

虽然订单项保存失败了，但是订单，图书信息都改了。

1、解决办法

把所有的代码写在一个dao的方法中。dao方法获取一个链接，这样只要有异常就回滚。

问题：

每个dao里面都来写。Dao🡪book,order,orderItem

这样的代码耦合度太高。x

2、解决办法

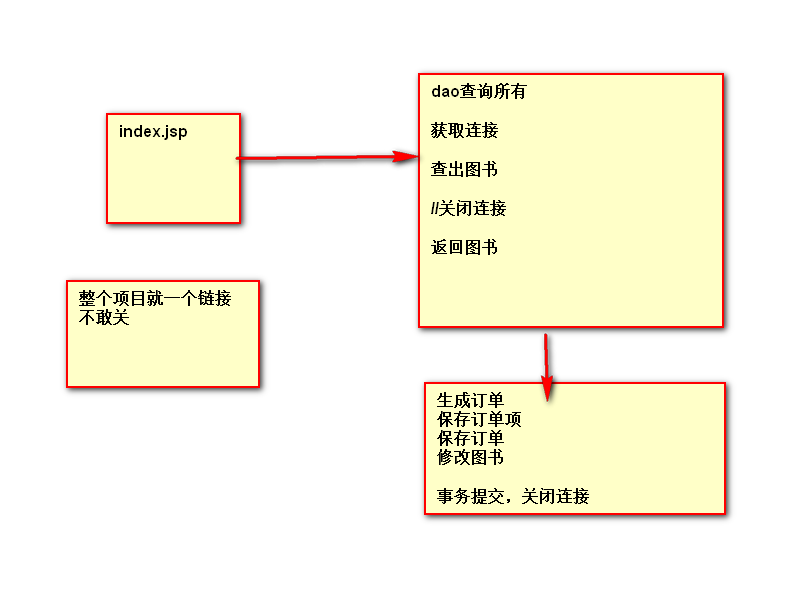
感觉只要保证所有获取的链接都是同一链接就可以。

首先获取的链接是同一链接（直接修改JDBCUtils.getConnection()）

在dao中不关闭链接。（最后一次再关）

You can't operate on a closed Connection!!!

整个项目使用一个链接（把链接对象做成单例）



3、解决办法。

但是每个功能模块应该使用一个链接。

每个dao方法都不要自动关链接，应该我们手动的去关链接（整个功能完成后关链接）

如果整个功能某一步失败就可以回滚了。

抓住这个特性：

一个请求到响应完成用的都是统一线程。

我们应该一个线程对应一个链接就行。当这个线程执行出问题就直接回滚。否则在提交

可以使用一个map来保存connnection key🡪就是线程号， value-🡪就是当前线程对应的链接

ThreadLocal<T> td

T t = td.get();

STFW