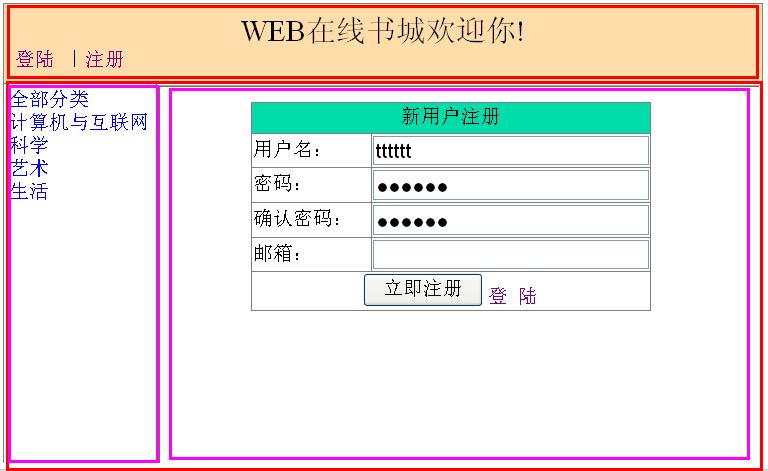
尚硅谷WEB书城

# WEB知识目录

1. [HTML/CSS/JavaScript/JS DOM/jQuery](#_<<WEB_QuickStart>>之后完成)
2. WEB QuickStart
3. XML
4. [Servlet](#_2._<<Servlet>>之后完成)
5. [JSP](#_3._<<JSP>>之后完成)
6. [EL表达式](#_4._<<EL表达式>>之后完成)
7. [JSTL](#_5._<<JSTL>>之后完成)
8. [cookie与session](#_6._<<cookie与session>>之后完成)
9. [My tag](#_7._<<My_tag>>之后完成)
10. [Filter](#_8._<<filter>>之后完成)
11. [Listener](#_10._<<文件上传下载>>之后完成)
12. [文件上传下载](#_10._<<文件上传下载>>之后完成)
13. [I18N](#_11._<<I18N>>之后完成)
14. [Ajax](#_12._<<Ajax>>之后完成)

# <<HTML/CSS/JavaScript >>之后完成



## 1.1. 完成应用的前台主页面: index.html

1). 使用DIV+CSS完成整体布局

2). 使用<iframe>标签实现主内容界面的动态显示

## 1.2. 完成注册页面: regist.html

1). 用户注册的Form表单:

username/password/password2/email/注册提交按钮

## 1.3. 完成登陆页面: login.html

1). 用户登陆的Form表单:

username / password /登陆提交按钮

## 1.4. 完成注册表单的javascript验证(使用Jquery)

1). 用户名: 长度为6到12位, 只能输入数字和英文和下划线

**/^\w{6,12}$/**

2). 密码: 长度为6位, 只能输入数字和英文

**/^[a-zA-Z0-9]{6}$/**

3). 确认密码: 必须与密码相同

4). 邮箱: 格式必须正确, 可以不输入

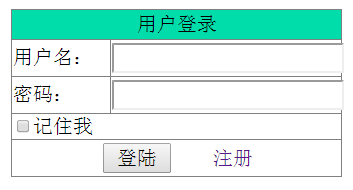
**/^([a-zA-Z0-9\_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})+$/**

## 1.5.总结

HTML+CSS+Javascript

1. DIV iframe form table input a
2. CSS:
   1. 选择器: 标签名/id/class
   2. 效果: width/height/backgroudcolor/font-size/float/border text-decoration /text-align
3. Jquery
   1. 选择器
   2. 操作(CRUD, 事件)
   3. 正则表达式

# <<Servlet>>之后完成





## 2.1. 分析web应用的结构

Regist.html

Mysql

bookstore

users

UserService

Boolean regist(user){

getUserByName(n)

//判断

addUser(user)

}

UserDao

getUserByName(name)

addUser(user)

浏览器

Servlet

doPost()

us.regist(u)

注册

Success.html

error.html

1. 获取请求参数username
2. 封装成一个user对象
3. us.regist(user)🡪boolean
4. 根据结果转向到成功或失败的页面

Struts2

Spring

db

Dao

Service

Servlet

JDBCUtils

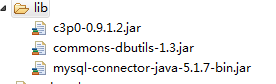
页面(html/jsp)

实体类

Hibernate

## 2.2. 搭建应用

1). 导入需要的jar包



2). 创建应用的包结构

|  |
| --- |
| com.atuigu.bookstore.servlet.client  RegistServlet  LoginServlet  com.atuigu.bookstore.servlet.manager  com.atuigu.bookstore.service  UserSerivce  com.atuigu.bookstore.service.impl  UserServiceImpl  com.atuigu.bookstore.dao  UserDao  com.atuigu.bookstore.dao.impl  UserDaoImpl  BaseDao  com.atuigu.bookstore.domain  User  com.atuigu.bookstore.util  JDBCUtils  junit.test  UserServiceTest |

## 2.3. 创建定义数据库和表的sql(bookstore.sql)

1). 创建数据库 : bookstore

2). 创建表 : users

|  |
| --- |
| #users 用户表  **CREATE** **TABLE** users  (  id **VARCHAR**(40) **PRIMARY** **KEY**, #主键  username **VARCHAR**(20) **UNIQUE**, #用户名  PASSWORD **VARCHAR**(30), #密码  email **VARCHAR**(40) #邮箱  ); |

## 2.4. 创建users表对应的实体类User

|  |
| --- |
| **public** **class** User {  **private** String id;  **private** String username;  **private** String password;  **private** String email;  } |

## 2.5. 创建c3p0连接池的的配置文件(src/c3p0-config.xml)

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <c3p0-config>  <named-config name=*"webDataSource"*>  <property name=*"jdbcUrl"*>jdbc:mysql://localhost:3306/web\_bookstore</property>  <property name=*"driverClass"*>com.mysql.jdbc.Driver</property>  <property name=*"user"*>root</property>  <property name=*"password"*>root</property>    <!-- 最小的连接数 -->  <property name=*"minPoolSize"*>5</property>  <!-- 最大的连接数 -->  <property name=*"maxPoolSize"*>30</property>  <!-- 初始化的连接数 -->  <property name=*"initialPoolSize"*>10</property>  <!-- 需要时一次性创建的连接数 -->  <property name=*"acquireIncrement"*>5</property>  <!-- 缓存多少个Statement对象 -->  <property name=*"maxStatements"*>15</property>  </named-config>  </c3p0-config> |

## 2.6. 创建访问数据库的帮助类JDBCUtils(使用连接池)

## 2.7. 创建操作表数据的所有DAO类的基类BaseDao

## 2.8. 完成注册功能

### 1). View层:

1.1). regist.html: 用户注册信息表单

1.2). login.html: 注册成功后自动跳转到此页

1.3). regist\_error.html: 注册失败后自动跳转到此页面

### 2).Controller层:

2.1). Severlet类: RegistServlet

2.2). 配置: url-pattern=/client/RegistServlet

### 3). Service层:

3.1). 接口: UserService(定义处理用户相关业务的方法)

boolean regist(User user)

3.2). 实现类: UserServiceImpl(实现处理用户相关业务的方法)

在regist()方法中调用UserDao的对象操作users表数据

### 4). DAO层:

4.1). 接口: UserDao(定义操作users表数据的方法)

User getByName(String username)

void insert(User user);

4.2). 实现类: UserDao(实现操作users表数据的方法)

getByName(): 调用BaseDAO的query()

insert(): 调用BaseDAO的update()

## 2.9. 完成登陆功能

### 1). View层:

1.1). login.html: 用户登陆信息表单

1.2). login\_success.html: 登陆成功后自动跳转到此页面,显示: 登陆成功

1.3). login\_error.html: 登陆失败后自动跳转到此页面, 显示: 用户名或密码不正确

### 2). Controller层:

2.1). 处理登陆请求的Servlet: LoginServlet

2.2). 配置: url-pattern=/client/LoginServlet

### 3). Service层:

3.1). 接口: UserService

boolean login(String name, String pwd)

3.2).实现类: UserServiceImpl

在login()方法中调用UserDao的对象操作users表数据

### 4). DAO层:

4.1). 接口: UserDao

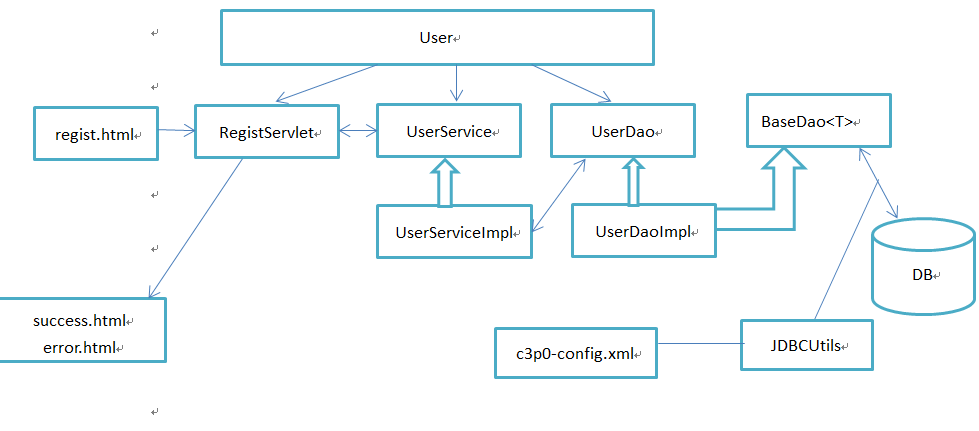
User getByNameAndPwd(String name, String pwd)

4.2). 实现类: UserDao

在getByNameAndPwd()中调用BaseDAO的query()

## 2.10. 总结:

### 注册功能结构图

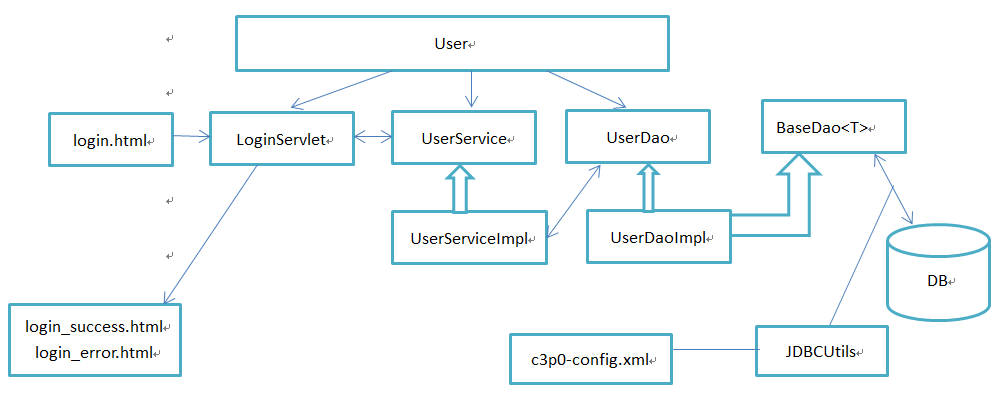


**View** 视图 HTML/JSP result.jsp

**Controller** 控制器 Servlet

**Model** Domain Service Dao

### 登陆结构图



### 接口与类的定义

UserService

boolean regist(User user)

boolean login(Sting username, String password)

UserDao

User getByName(String name)

void insert(User user)

User getByNameAndPwd(String name, String password)

BaseDao<T> 所有dao实现类的基类

update(String sql, Object… params) //CUD DML

T query(String sql, Object… params) //R DQL

List<T>queryForList(String sql, Object… params) //R DQL

JDBCUtils

Conection getConection()

releaseConn(Connection conn)

# 3. <<JSP>>之后完成

## 3.1. 将html页面改为jsp页面

1).欢迎页面: index.jsp(自动跳转到client/client.jsp)

2).应用前台主面: client.jsp

3).注册页面: regist.jsp

4).登陆页面: login.jsp

5).登陆成功页面: login\_success.jsp

## 3.2. 改善请求处理后的跳转及显示

### 1). 注册功能:

1.1). 成功: 转发到login.jsp, 并显示: 注册成功, 可以登陆了!

1.2). 失败: 转发到regist.jsp, 显示: 注册失败, 此用户名已存在!

### 2). 登陆功能:

2.1). 成功: 重定向到login\_success.jsp, 并显示: 登陆成功!

2.2). 失败: 转发到login.jsp, 显示: 登陆失败, 用户名或密码错误!

### 3). 失败返回页面后, 输入框的自动回显

## 3.3. 实现一个Servlet处理多个请求

分析:

|  |
| --- |
| 用户 UserServlet doPost struts2  注册 /bookstore/ UserServlet?method=regist  登陆 /bookstore/ UserServlet? method =login  显示用户列表 /bookstore/ UserServlet? method =”list”  书 BookServlet  books  订单 OrderServlet |

1). 应用中的所有对Servlet的请求都需要添加一个请求参数method=xxx

2). 创建一个应用中的所有Servlet类的基类: BaseServlet,利用反射调用method参数值所对应的方法

|  |
| --- |
| public void xxx(HttpServletRequest req, HttpServlet resp) {} |

3). 在应用的Servlet类中, 只需要定义处理请求的方法:

|  |
| --- |
| public class UserServlet extends BaseServlet {  public void xxx(request, response){ }  } |

注意:

**方法必须是public的,且与method请求参数值同名,**

**BaseServlet不需要在web.xml文件中配置**

## 3.4. 实现对请求参数到实体类对象的自动收集封装WebUtils

<T> T request2Bean(request, Class<T> beanClass)

## 总结 :

### href="<%=request.getContextPath%>/client/user/regist.jsp"

### 页面如果需要动态显示一些信息

1. 请求的转发(不能重定向)

2. 转发之前: request.setAttribute("msg", "提示的内容")

3. 在Jsp中: <%=req.getAttribute("msg")==null ? "" : req.getAttribute("msg")%>

### 页面回显

<input name="username"

value="<%=req.getParameter("username")==null ? "" :req.getParameter("username")%>">

### 一个Servlet处理多个请求

1. 页面的请求: UserServlet?method=xxx(方法名)

2. abstract class BaseServlet extends HttpServlet {

String methodName = req.getparameter("method");

//通过反射调用实现子类的对应的处理请求的方法

}

3. class UserServlet extends BaseServlet {

//处理注册的请求

public void regist(req, resp) {

}

//处理登陆的请求

public void login(req, resp) {

}

}

### 自动收集请求参数并封装成实体类对象?

beanUtils---->copyProperty(bean, propertyName, value)

### 区别: 类对象与 Class对象

class Person {} Person.class

**Class.forName(“com.atguigu.bean.Person”).newInstance()**🡪

1. 在内存找对应的Class对象(通过全类名)?
   1. 🡪找到了, clazz.newInstance()
   2. 🡪找不到, 去classpath下找对应的class文件?
      1. 🡪找到了
         1. 将对应的class文件加载到内存中
         2. 生成包含class文件所有信息的一个对象(Class)
         3. clazz.newInstance();
      2. 🡪没有找到
         1. 抛出ClassNotFoudException

Class Field Construtor Method

c.getMethod(…).invoke()

# 4. <<EL表达式>>之后完成

## 4.1. 使用EL表达式完善页面

1). 显示提示信息

2). 失败返回界面后, 输入框的自动回显

## 4.2. 搭建应用的后台主页面(/manager/manager.jsp)

1). 与前台client.jsp的结构相同

## 4.3. 实现后台图书分类的CRUD功能

设计表:

|  |
| --- |
| CREATE TABLE `categorys` (   `id` varchar(40) PRIMARY KEY,   `NAME` varchar(30) default NULL ) |

设计整体结构

**Controller(Servlet)**

CategoryServlet

add()

list()

update()

delete()

**DAO**

CategoryDao

insert(Category c)

getAll()

update(Category c)

deleteById(id)

**Service**

CategoryService

add(Category c)

getAll()

update(Category c)

delete(String id)

**View(jsp)**

add.jsp

list.jsp

result.jsp

1). 分类的添加

2). 分类的列表显示

3). 分类名称的修改

4). 分类的删除

## 4.4. 转发请求到页面显示提示信息功能的封装WebUtils

1). 转发到/result.jsp页面, 并显示以message为name的指定信息

void forwordMessageUI(request, response, msg)

2). 转发请求到指定页面,并在请求域中保存指定属性

void forword(request,response,path,name,value)

# 5. <<JSTL>>之后完成

## 5.1. 使用JSTL: 完善图书分类列表的显示

## 5.2. 实现后台图书信息的CRUD操作:

设计表:

|  |
| --- |
| #books 图书表  **CREATE** **TABLE** books  (  id **VARCHAR**(40) **PRIMARY** **KEY** , #主键  NAME **VARCHAR**(50), #书名  author **VARCHAR**(20), #作者  price **FLOAT**, #价格  salesamount **INT**(11) **NOT** **NULL** **DEFAULT** 0,#已售数量  storenumber **INT**(11) **NOT** **NULL**, #库存数量  imagepath **VARCHAR**(100), #图书照片存储路径  categorysid **VARCHAR**(40), #此图书对应的分类ID(是一个外键)  **CONSTRAINT** categorysid\_FK **FOREIGN** **KEY**(categorysid) **REFERENCES** categorys(id)  ); |

设计整体结构

JavaBean : Book

**Service**

**BookService**

void addBook( book)

geAllBooks()

Book getBook(id)

updateBook(Book)

deleteBook(id)

**Controller(Servlet)**

BookServlet

toAddUI()

add()

list()

toUpdateUI()

update()

delete()

**DAO**

BookDao

insert(Book b)

getAll()

getById(String id)

update(Book b)

deleteById(id)

books

categorys

**View(jsp)**

add.jsp

list.jsp

update.jsp

CategoryDao

getAll()

CategoryService

List<Category> getAllCategorys()

1). 图书信息的添加(暂时不提交图片)

2). 所有图书信息的列表显示

3). 图书信息的修改

4). 图书信息的删除

## 5.3. 前台图书列表的分页显示:

Service

BookService

Controller

BookServlet

getAllCategorys()

getBooks()

View

list.jsp



Page

List<Book>//显示的图书对象的集合

int pageNum //第几页

int totalPageCount;//总页数

int totalRecord; //总记录

int bookCount; //页面显示几本书

int pageIndexCount; //连续显示的页码数据

int firstPageIndex //连续显示的第一个页码

int lastPageIndex //连续显示的最后一页码

### 1). 分页信息的封装: Page<T>

### 2). 查询条件信息的封装: ConditionBook

### 3). 完成多条件查询分页的功能:

#### 3.1). View层: books.jsp

取出请求域中的page对象数据, 利用JSTL+EL+HTML标签来显示

#### 3.2). Controller层: BookServlet

void getBooks(req, resp): 得到请求参数数据, 调用bookService的getPage()得到当前页数据的page对象

#### 3.3). Service层: BookService / BookServiceImpl

Page getPage(ConditionBook cb)

#### 3.4). Dao层: BookDao / BookDaoImpl

long getTotalRecord(int minPrice, int maxPrice, String categorysid)

List<Book> getBooks(int first, int count, int minPrice, int maxPrice, String categorysid)

# 6. <<cookie与session>>之后完成

## 6.1. 使用Cookie技术: 显示图书浏览记录

BookServlet

showBook()

getHistory()

clearHistory()

View

book/list.jsp

history.jsp

book/book.jsp



Cookie

book\_id1---name1

book\_id2—name2

book\_id3---name3

区别查看书的cookie和别的所有cookie











1). 显示图书的浏览记录(最多显示4条, 最后浏览的在前面)

2). 可以清空浏览记录

## 6.2. 使用session技术: 三个相关功能

### 1). 用户的登陆与登出

### 2). 注册的一次性验证码

### 3). 购物车的操作

CartService

addBook()

updateCartItem()

deleteCartItem()

clearCart()

CartServlet

add()

updateCount()

delete()

clear()

Cart对象

CartItem

Cart

Session对象

cart

3.1). 添加一本书到购物车

3.2). 显示购物车中所有的购物项

3.3). 修改某个购物项的书的数量

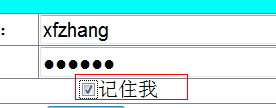
3.4). 删除某个购物项

## 6.3.综合应用: 用户的自动登陆

jsessionid=sid

Sesssion(“Suser”,user)

jsessionid=sid

Session(“sUser”, user)

cUser=user信息

Cookie c = new Cookie(“cUser”, user信息)

c.setMaxage(正数)

res.addCookie(c);

cUser=user信息

新的Session对象

从request中获取以cUser为name的cookie

得到它value(user信息)

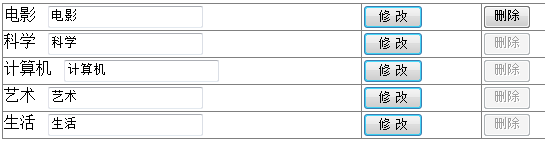
创建一个user对象

放到session对象(“sUser”, user)

为什么关闭浏览器不可能得到原来的Session对象

# 7. <<My tag>>之后完成

## 7.1. 利用自定义标签实现功能:



功能描述: ***分类的删除按钮在该分类下有图书时可以操作,否则不能操作***.

# 8. <<filter>>之后完成

## 8.1. 实现前台的订单操作:

1). 在购物车页面: 去结算

2). 在地址页面: 添加当前用户的收货人地址信息

|  |
| --- |
| #address 地址表  **CREATE** **TABLE** address  (  id **VARCHAR**(40) **PRIMARY** **KEY**,#主键  NAME **VARCHAR**(20),#收货人姓名  location **VARCHAR**(100),#地址  cellphone **VARCHAR**(20),#电话  usersid **VARCHAR**(40),#所对应用户的id  **CONSTRAINT** usersid\_FK **FOREIGN** **KEY**(usersid) **REFERENCES** users(id)  ); |

3). 在地址页面: 展示当前用户的所有收货地址

4). 在地址页面: 生成订单

|  |
| --- |
| #orders 订单表 cart  **CREATE** **TABLE** orders  (  id **VARCHAR**(40) **PRIMARY** **KEY**, #主键  number **VARCHAR**(50),#订单号  STATUS **BOOLEAN** **DEFAULT** **FALSE**,#状态, 标识是否发货  price **FLOAT**,#价格  ordertime **TIMESTAMP**,#下单时间  usersid **VARCHAR**(40),#对应用户的ID  addressid **VARCHAR**(40),#对应的地址ID  **CONSTRAINT** usersid\_FK1 **FOREIGN** **KEY**(usersid) **REFERENCES** users(id),  **CONSTRAINT** addressid\_FK **FOREIGN** **KEY**(addressid) **REFERENCES** address(id)  ); |

|  |
| --- |
| # orderitems 订单项 cartitem  **CREATE** **TABLE** orderitems  (  id **VARCHAR**(40) **PRIMARY** **KEY**,#主键  quantity **INT**,#数量  price **FLOAT**,#单价  ordersid **VARCHAR**(40),#对应订单的ID  booksid **VARCHAR**(40),#对应书的ID  **CONSTRAINT** ordersid\_FK **FOREIGN** **KEY**(ordersid) **REFERENCES** orders(id),  **CONSTRAINT** booksid\_FK **FOREIGN** **KEY**(booksid) **REFERENCES** books(id)  ); |

users/categorys/books/address/order/orderitem

两种事物的关联关系哪几种?

一对一: 人:头 人:身份证

一对多 : 学生:老师 父亲:孩子

多对多 :

BookDao

batchUpdateBook(Object[][] params)

OrderService

makeOrder(Cart,userid, aid)

makeOrder()

OrderDao

insert(Order order)

OrderItemDao

batchInsert(Object[][] params)

## 8.2. 实现后台的订单操作

1). 根据订单的状态显示订单列表

2). 对未发货的订单发货

## 8.3. 应用Filter

1). 实现对用户是否登陆的统一检查

2). 实现数据库事务的统一管理

过滤请求请求, 满足要求:放行, 不满足要求: 拦截(请求没有到达Servlet)

1. Filter的使用
2. 事务处理(3步)
   1. 1. 设置事务手动提交
   2. 2. 如果工作全部正常完成, 就提交事务
   3. 3. 如果在执行过程中去异常被捕获到了, 就回滚事务
3. 如何实现将conn与用户对应的Thread缓存起来使用
   1. Servlet是单例多线程运行
   2. Map<Thread, Connection> map
4. 异常的处理(自定义异常)
   1. 异常从下向上抛出, 在Filter中处理(也可在Servlet中处理)
5. ThreadLocal<T>

它相当于是一个缓存结构(内部是一个Map<Thread,T>)

Dao

Thread1—conn1

Thread1—conn1

Filter

T2—conn2

T2=conn2

T3—conn3

T3—conn3

## 8.4. 面向接口编程: (高级扩展部分)

1). 使用工厂模式和Properties文件实现Service类和DAO类的可配置

Service

Dao

Servlet

# 9. <<Listener>>之后完成

## 9.1. 使用Listener技术: 实现前台在线登陆用户的列表显示

1). 显示登陆用户和匿名访客人数

2). 显示登陆用户的用户名列表

3). 显示匿名访客的ip列表

## 9.2. 利用动态代理模式和注解技术完善事务的管理(高级扩展部分)

1). 利用注解技术实现在业务方法上是否应用事务的可配置

2). 利用动态代理实现数据库连接的统一管理和事务处理

# 10. <<文件上传下载>>之后完成

## 10.1. 应用FileUpload: 实现带图片的图书信息添加

# 11. <<I18N>>之后完成

## 11.1. 应用I18N: 实现登陆页面的国际化

# 12. <<Ajax>>之后完成

## 12.1. 检查注册用户名是否可用的Ajax处理

## 12.2. 添加图书到购物车的Ajax处理

## 12.3. 修改购物项图书数量的Ajax处理