SSH综合案例\_day03总结

今日内容:

* SSH综合案例

## 网上商城需求分析:

### 前台:用户模块.

注册:

\* JS非空校验.(不是必须)

\* Struts2数据校验

\* AJAX校验用户名是否存在.

\* 发送激活邮件.

\* 验证码程序.

激活:

\* 点击邮件中连接地址,对用户进行激活.

登录:

\* 数据校验.

\* 激活后的用户才可以进行登录.

\* 验证码程序.

退出:

\* 销毁session.

### 前台:分类模块(首页):

查询一级分类:显示到菜单位置.

查询一级分类:关联二级分类.

### 前台:二级分类

根据一级分类:查询到二级分类.

### 前台:商品模块(首页):

查询热门商品:

\* 检索10个热门商品.

查询最新商品:

\* 检索10个最新商品.(按时间排序)

根据分类查询商品:

\* 根据分类的ID检索商品.

\* 对商品显示的时候,分页显示.

根据商品ID检索商品:

\* 显示商品详情.

### 购物模块:

添加到购物车:

\* 将购物项添加到购物车.

\* 如果购物项已经存在在购物车中,那么数量和小计需要增加.

\* 如果购物项没有在购物车中,添加一个新的购物项.

清空购物车:

\* 将购物车中所有的购物项,清空.

从购物车中移除:

\* 将某一个购物项从购物车中移除.

### 订单模块:

保存订单:

\* 数据从购物车中获取.

显示某个订单:

\* 根据订单ID检索订单.

查询我的订单:

\* 根据用户ID检索订单.

在线支付:

\* 选择第三方支付平台,为订单进行付款.

### 用户模块:(后台)

查询所有用户:

修改用户信息:

删除用户信息:

### 一级分类:(后台)

查询所有一级分类:

\* 分页.

添加一级分类:

修改一级分类:

删除一级分类:

\* 删除二级分类.(级联)

### 二级分类:(后台)

查询所有二级分类:

添加二级分类:

\* 所属一级分类:

修改二级分类:

\* 所属一级分类:

删除二级分类:

### 商品模块(后台)

查询所有商品:

\* 分页.

\* 条件查询.

添加商品:

\* 上传商品图片.

修改商品:

\* 修改上传图片.

删除商品:

### 订单模块(后台)

查询所有订单:

\* 分页

按状态查询订单:

\* 分页

是否可以删除订单:

## 设计页面:

静态页面:

## 数据库分析与设计:

用户表:

\* 用户ID

\* 用户名

\* 密码

\* 邮箱

\* 姓名

\* 性别

\* 电话

\* 地址

\* 用户状态:

\* 激活码:

一级分类表

\* 一级分类ID

\* 一级分类名称

二级分类表:

\* 二级分类ID

\* 二级分类名称

\* 一级分类ID

商品表:

\* 商品ID

\* 商品名称

\* 商品商城价格

\* 商品指导价格

\* 商品描述

\* 商品图片

\* 二级分类ID

订单表:

\* 订单ID

\* 订单时间

\* 订单地址

\* 订单总价

\* 订单状态

\* 用户ID

订单项表:

\* 订单项ID

\* 商品ID

\* 订单ID

\* 数量

\* 小计

## SSH环境搭建:

### 第一步:创建一个web项目.

### 第二步:导入相应jar包.

Struts2的jar包:

\* struts2框架解压路径/apps/struts2-blank.war/WEB-INF/lib/\*.jar

\* struts2框架解压路径/lib/struts2-spring-plugin-2.3.15.3.jar

\* struts2框架解压路径/lib/struts2-json-plugin-2.3.15.3.jar

Spring的jar包:

\* Spring开发基本jar包

\* spring框架解压路径/lib/spring-beans-3.2.0.RELEASE.jar

\* spring框架解压路径/lib/spring-context-3.2.0.RELEASE.jar

\* spring框架解压路径/lib/spring-core-3.2.0.RELEASE.jar

\* spring框架解压路径/lib/spring-expression-3.2.0.RELEASE.jar

\* spring框架依赖包解压路径/com.springsource.org.apache.commons.logging-1.1.1.jar

\* spring框架依赖包解压路径/com.springsource.org.apache.log4j-1.2.15.jar

\* Spring的AOP开发(Aspectj)

\* spring框架解压路径/lib/spring-aop-3.2.0.RELEASE.jar

\* spring框架解压路径/lib/spring-aspects-3.2.0.RELEASE.jar

\* spring框架依赖包解压路径/com.springsource.org.aopalliance-1.0.0.jar

\* spring框架依赖包解压路径/com.springsource.org.aspectj.weaver-1.6.8.RELEASE.jar

\* Spring的JDBC支持、事务管理、整合Hibernate

\* spring框架解压路径/lib/spring-jdbc-3.2.0.RELEASE.jar

\* spring框架解压路径/lib/spring-tx-3.2.0.RELEASE.jar

\* spring框架解压路径/lib/spring-orm-3.2.0.RELEASE.jar

\* Spring整合web项目:

\* spring框架解压路径/lib/spring-web-3.2.0.RELEASE.jar

\* Spring整合Junit单元测试:

\* spring框架解压路径/lib/spring-test-3.2.0.RELEASE.jar

Hibernate框架jar包:

\* hibernate框架解压路径/hibernate3.jar

\* hibernate框架解压路径/lib/required/\*.jar

\* hibernate框架解压路径/lib/jpr/\*.jar

\* hibernate框架整合log4j

\* slf4j-log4j12-1.7.2.jar

\* 数据库驱动包

\* c3p0连接池jar包.

### 第三步导入相应配置文件:

Struts框架中:

\* web.xml

\* 核心过滤器:

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

\* struts.xml

Spring框架中:

\* web.xml

<!-- Spring框架使用监听器,服务器启动的时候加载Spring的配置文件 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- 监听器默认加载WEB-INF/application.xml -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

\* applicationCotnext.xml

<!-- 引入外部jdbc.properties文件 -->

<context:property-placeholder location="classpath:jdbc.properties"/>

<!-- 配置连接池的信息 -->

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">

<!-- 数据库连接的四个基本参数 -->

<property name="driverClass" value="${jdbc.driver}"/>

<property name="jdbcUrl" value="${jdbc.url}"/>

<property name="user" value="${jdbc.user}"/>

<property name="password" value="${jdbc.password}"/>

</bean>

<!-- 配置Hibernate的相关属性 -->

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<!-- 注入连接池 -->

<property name="dataSource" ref="dataSource"/>

<!-- 配置Hibernate的其他的属性 -->

<property name="hibernateProperties">

<props>

<!-- Hibernate的方言 -->

<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>

<prop key="hibernate.show\_sql">true</prop>

<prop key="hibernate.format\_sql">true</prop>

<prop key="hibernate.hbm2ddl.auto">update</prop>

<prop key="hibernate.connection.autocommit">false</prop>

</props>

</property>

</bean>

<!-- 声明式事务管理 -->

<!-- 配置事务管理器 -->

<bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<!-- 注入 sessionFactory-->

<property name="sessionFactory" ref="sessionFactory"/>

</bean>

<tx:annotation-driven transaction-manager="transactionManager"/>

\* 不使用hibernate配置文件:

\* 将hibernate的信息配置到spring框架中.

log4j的配置文件:

连接数据库基本参数配置文件:

## 编码实现:

### 首页显示:

复制所有文件到工程下:

\* css

\* js

\* image

复制页面到工程WEB-INF/jsp/

\* 将后缀.htm改为jsp

访问一个Action通过Action转向到JSP.

\* 创建包:

\* 编写Action:

\* 直接返回一个字符串.

\* 配置struts.xml

<!-- 配置首页访问的Action -->

<action name="index" class="indexAction">

<result name="indexSuccess">/WEB-INF/jsp/index.jsp</result>

</action>

\* 配置applicationContext.xml

<bean id="indexAction" class="cn.itcast.shop.index.IndexAction" scope="prototype"></bean>

### 用户模块:

#### 注册功能:

在index.jsp页面上点击注册:

\* 跳转到regist.jsp页面上(通过action跳转)

1. 创建表:

\* 建表语句

create table user(

uid int primary key auto\_increment,

username varchar(20) ,

password varchar(20) ,

name varchar(20),

email varchar(30),

phone varchar(20),

addr varchar(50),

sex varchar(10),

state int,

code varchar(64)

);

1. 创建包:

\* cn.itcast.shop.user;

3.在包下创建类和文件:

\* UserAction; --- Action类

\* UserService; --- 业务层类

\* UserDao --- 持久层类

\* User --- 实体类

public class User {

private Integer uid;

private String username;

private String password;

private String name;

private String email;

private String phone;

private String addr;

private String sex;

private Integer state;

private String code;

public Integer getUid() {

return uid;

}

public void setUid(Integer uid) {

this.uid = uid;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

public String getAddr() {

return addr;

}

public void setAddr(String addr) {

this.addr = addr;

}

public String getSex() {

return sex;

}

public void setSex(String sex) {

this.sex = sex;

}

public Integer getState() {

return state;

}

public void setState(Integer state) {

this.state = state;

}

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

}

\* User.hbm.xml --- 创建一个映射

<hibernate-mapping>

<class name="cn.itcast.shop.user.User" table="user">

<!-- 配置唯一标识 -->

<id name="uid" column="uid">

<generator class="native"/>

</id>

<!-- 配置普通属性 -->

<property name="username" column="username"/>

<property name="password" column="password"/>

<property name="name" column="name"/>

<property name="email" column="email"/>

<property name="phone" column="phone"/>

<property name="addr" column="addr"/>

<property name="sex" column="sex"/>

<property name="state" column="state"/>

<property name="code" column="code"/>

</class>

</hibernate-mapping>

\* 修改struts.xml

<!-- 配置用户的Action -->

<action name="user\_\*" class="userAction" method="{1}">

<result name="registPageSuccess">/WEB-INF/jsp/regist1.jsp</result>

</action>

\* 修改applicationContext.xml

<bean id="userAction" class="cn.itcast.shop.user.UserAction" scope="prototype">

</bean>

\* 在注册页面上添加前台JS校验:

function checkForm(){

// 校验用户名:

var username = document.getElementById("username").value;

if(username == ''){

alert("用户名不能为空!");

return false;

}

// 校验密码:

var password = document.getElementById("password").value;

if(password == ''){

alert("密码不能为空!");

return false;

}

// 校验确认密码

var repassword = document.getElementById("repassword").value;

if(password != repassword){

alert("两次密码不一致!");

return false;

}

}

#### 注册:

Regist.jsp------->UserAction(regist方法)------>UserService------------UserDao

\* 校验:(后台)

\* 在Action类所在包下创建一个XML.(针对某个方法校验)

\* Action类名-方法对应路径名称-validation.xml

\* UserAction-user\_regist-validation.xml

\* 引入DTD.

\* 编写校验:

\* 校验后的INPUT视图配置.

\* 发送邮件:

\* 搭建邮箱服务器:

1. 安装邮箱服务器:
2. 设置邮箱服务器:

\* 设置服务器域名.

\* 工具--->服务器设置--->设置域名

\* shop.com

1. 创建几个账号:

\* 账号--->新建账号

1. **配置邮箱客户端:**

\* foxmail:

\* 安装foxmail:

\* 配置一个账号:

\* [bbb@shop.com](mailto:bbb@shop.com)

\* [aaa@shop.com](mailto:aaa@shop.com)

\* 发送邮件:

public class MailUtils {

public static void sendMail(String to,String code) throws Exception{

Properties props = new Properties();

props.setProperty("mail.smtp", "localhost");

// 1.Session对象.连接(与邮箱服务器连接)

Session session = Session.getInstance(props, new Authenticator() {

@Override

protected PasswordAuthentication getPasswordAuthentication() {

return new PasswordAuthentication("service@shop.com", "111");

}

});

// 2.构建邮件信息:

Message message = new MimeMessage(session);

// 发件人:

message.setFrom(new InternetAddress("service@shop.com"));

// 收件人:

message.setRecipient(RecipientType.TO, new InternetAddress(to));

// 设置标题

message.setSubject("来自SHOP激活邮件");

// 设置正文

message.setContent("<h1>来自SHOP的官网激活邮件</h1><h3><a href='http://192.168.40.99:8080/shop/user\_active.action?code="+code+"'>http://192.168.40.99:8080/shop/user\_active.action?code="+code+"</a></h3>", "text/html;charset=UTF-8");

// 3.发送对象

Transport.send(message);

}

}

验证码程序:

1. 创建一个画布
2. 设置一个画笔
3. 生成四个随机的字母或数字
4. 将四个字母或数字写到图片上.
5. 将图片输出到浏览器.

\* 编写一个Action类

\* 生成验证码

\* 在struts.xml中配置

\* 在applicationContext.xml中配置

\* 在regist1.jsp中验证码图片的位置:

\* <img src=”${pageContext.request.contextPath}/checkImg.action”>

\* 点击图片切面生成字母或数字

\* 在UserAction中regist方法上进行校验.

#### 激活:

在邮箱地址上点击链接---------->UserAction(active方法)------------>UserService--------------->UserDao

\* 1.根据激活码完成查找

\* 2.找到该用户

\* 修改用户的状态. 0 修改为1.

#### 登录:

首页-------------->UserAction(loginPage方法)

登录页面----------->UserAction(login方法)---------------UserService--------------UserDao

\* 1.接收提交的用户名和密码.

\* 2.调用Service根据用户名和密码及用户的状态查询.

\* 3.登录成功:

\* 将用户存到Session作用范围.

#### 退出:

获得用户的session.

将session销毁.

重定向到首页的Action.

### 一级分类模块:

1. 创建表:

create table category(

cid int primary key auto\_increment,

cname varchar(20)

);

INSERT INTO `category` VALUES ('1', '女装男装');

INSERT INTO `category` VALUES ('2', '鞋靴箱包');

INSERT INTO `category` VALUES ('3', '运动户外');

INSERT INTO `category` VALUES ('4', '珠宝配饰');

INSERT INTO `category` VALUES ('5', '手机数码');

INSERT INTO `category` VALUES ('6', '家电办公');

INSERT INTO `category` VALUES ('7', '护肤彩妆');

1. 创建包:

\* cn.itcast.shop.category;

1. 创建类:

\* CategoryAction

\* CategoryService

\* CategoryDao

\* Category

\* Category.hbm.xml

1. 添加映射文件到spring的配置文件

\* <value>cn/itcast/shop/category/Category.hbm.xml</value>

1. 配置Service和DAO到Spring的配置文件.

\* 在Service中注入DAO.

\* 在Service层开启事务.

1. 在IndexAction类中注入Service

// 注入一级分类的Service

private CategoryService categoryService;

public void setCategoryService(CategoryService categoryService) {

this.categoryService = categoryService;

}

1. 查询后将集合存入到Session范围.

\* 以后查询一级分类不用去数据库查询.

### 首页商品显示:

1. 创建表:

二级分类表

CREATE TABLE `categorysecond` (

`csid` int(11) NOT NULL AUTO\_INCREMENT,

`csname` varchar(255) DEFAULT NULL,

`cid` int(11) DEFAULT NULL,

PRIMARY KEY (`csid`),

KEY `FK936FCAF21DB1FD15` (`cid`),

CONSTRAINT `FK936FCAF21DB1FD15` FOREIGN KEY (`cid`) REFERENCES `category` (`cid`)

) ENGINE=InnoDB AUTO\_INCREMENT=37 DEFAULT CHARSET=utf8;

商品表

CREATE TABLE `product` (

`pid` int(11) NOT NULL AUTO\_INCREMENT,

`pname` varchar(255) DEFAULT NULL,

`market\_price` double DEFAULT NULL,

`shop\_price` double DEFAULT NULL,

`image` varchar(255) DEFAULT NULL,

`pdesc` varchar(255) DEFAULT NULL,

`is\_hot` int(11) DEFAULT NULL,

`pdate` datetime DEFAULT NULL,

`csid` int(11) DEFAULT NULL,

PRIMARY KEY (`pid`),

KEY `FKED8DCCEFB9B74E02` (`csid`),

CONSTRAINT `FKED8DCCEFB9B74E02` FOREIGN KEY (`csid`) REFERENCES `categorysecond` (`csid`)

) ENGINE=InnoDB AUTO\_INCREMENT=74 DEFAULT CHARSET=utf8;

1. 创建包结构:

\* cn.itcast.shop.product

1. 创建类:

\* ProductAction

\* ProductService

\* ProductDao

\* Product

\* Product.hbm.xml

1. 把映射配置到Spring中.
2. 将商品的Service和DAO在Spring中配置
3. 将商品Service注入到IndexAction.
4. 查询热门商品:

\* 分页查询:

\* 使用离线条件查询:

\* 手动封装一个HibernateCallBack的实现类.

1. 查询最新商品:

\* 分页查询:

\* 使用离线条件查询:

\* 手动封装一个HibernateCallBack的实现类.

### 二级分类:

分类页面:

\* 查询一级分类 及 一级分类关联二级分类 还有 相应商品信息.

1. 建表已经完成
2. 创建包:

\* cn.itcast.shop.categorysecond

1. 创建类:

\* CategorySecondAction

\* CategorySecondService

\* CategorySecondDao

\* CategorySecond

\* CategorySecond.hbm.xml

1. 配置一级分类与二级分类的关联关系 :

\* 需要改造实体类:

\* 需要改映射文件:

\* 二级分类的映射文件中:

<!-- 配置关联关系 -->

<many-to-one name="category" class="cn.itcast.shop.category.Category" column="cid"/>

\* 一级分类的映射文件中:

<!-- 配置关联关系 -->

<set name="categorySeconds">

<key column="cid"/>

<one-to-many class="cn.itcast.shop.categorysecond.CategorySecond"/>

</set>

1. 点击查询 :

\* ProductAction中findByCid的方法.

1. 配置ProductAction.

\* struts.xml

\* applicationContext.xml

1. 在Action中调用Service---DAO.

\* Action调用CategoryService.

\* Category.hbm.xml中:

\* <set lazy=”false” order-by=”csid”>

1. 显示商品:

\* 传递page=1;

\* 在ProductAction中接收page.

\* 向页面返回数据:

\* 显示商品集合:

\* 当前页数:

\* 总页数:

\* 第 1/12 页

\* 通过总记录数.

\* 参数每页显示记录数.

\*\*\*\*\* 将数据封装到一个Bean中.

\* 配置商品和二级分类关联:

\* 查询某个分类的商品数量.

\* 在业务层将需要的数据封装到PageBean中.

\* 在Action中获得到PageBean数据.

\* 将数据带到页面上进行显示.

### 显示某个商品的详情:

在商品列表页面上对图片提供一个链接:

根据商品ID进行查询商品:

在页面显示商品的数据 :

\* 通过模型驱动将数据回显到页面.

在页面上显示分类信息 :

### 根据二级分类的ID显示商品:

\* 在分类栏上点击二级分类:

\* 查询二级分类的商品:

\* 传递二级分类ID

\* 传递页数page=1

\* 在ProductAction中编写方法findByCsid();

\* 按分页查询:

\* 查询所有分类信息进行显示:

### 购物模块:

封装实体对象:

\* 购物项:CartItem

\* 商品对象.

\* 数量

\* 小计

\* 购物车:Cart

\* 购物项集合:

\* Map<String , CartItem>:方便移除.

\* 总计:

\* 购物车中:

\* 属性:

\* 购物项集合:

\* 总计

\* 方法:

\* 添加到购物车

\* 从购物车中移除购物项

\* 清空购物车

\* 添加到购物车:

\* 商品详情页面:---->CartAction

\* 调用Cart中的addCart方法.

\* 编写了一个Action:

\* 接收pid和count

\* 执行addCart方法的时候.封装一个购物项.

\* 将购物项添加到购物车.

\* 购物车需要从session中获得.

\* 在struts.xml中配置:

\* 在applicationContext.xml中配置:

\* 在Cart中,提供了获得所有map的value的方法.

\* getCartItems();

### 订单模块;

创建表:

CREATE TABLE `orders` (

`oid` int(11) NOT NULL AUTO\_INCREMENT,

`total` double DEFAULT NULL,

`ordertime` datetime DEFAULT NULL,

`state` int(11) DEFAULT NULL,

`addr` varchar(50) DEFAULT NULL,

`phone` varchar(20) DEFAULT NULL,

`uid` int(11) DEFAULT NULL,

PRIMARY KEY (`oid`),

KEY `FKC3DF62E5AA3D9C7` (`uid`),

CONSTRAINT `FKC3DF62E5AA3D9C7` FOREIGN KEY (`uid`) REFERENCES `user` (`uid`)

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8;

创建中间表:

CREATE TABLE `orderitem` (

`itemid` int(11) NOT NULL AUTO\_INCREMENT,

`count` int(11) DEFAULT NULL,

`subtotal` double DEFAULT NULL,

`pid` int(11) DEFAULT NULL,

`oid` int(11) DEFAULT NULL,

PRIMARY KEY (`itemid`),

KEY `FKE8B2AB6166C01961` (`oid`),

KEY `FKE8B2AB6171DB7AE4` (`pid`),

CONSTRAINT `FKE8B2AB6166C01961` FOREIGN KEY (`oid`) REFERENCES `orders` (`oid`),

CONSTRAINT `FKE8B2AB6171DB7AE4` FOREIGN KEY (`pid`) REFERENCES `product` (`pid`)

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8;

创建包:

\* cn.itcast.shop.order;

创建类;

\* Order

\* OrderItem

\* OrderAction

\* OrderService

\* OrderDao

创建映射;

\* 关联关系配置:

生成订单:

\* 购物车页面:提交订单-----OrderAction----->OrderService----->OrderDao

\* 封装订单对象和订单项对象.

\* Aciont---Service----Dao:

配置:

Struts.xml

applicationContext.xml

## 后台管理:

页面:

### 登录模块:

建表:

建包;

建类: