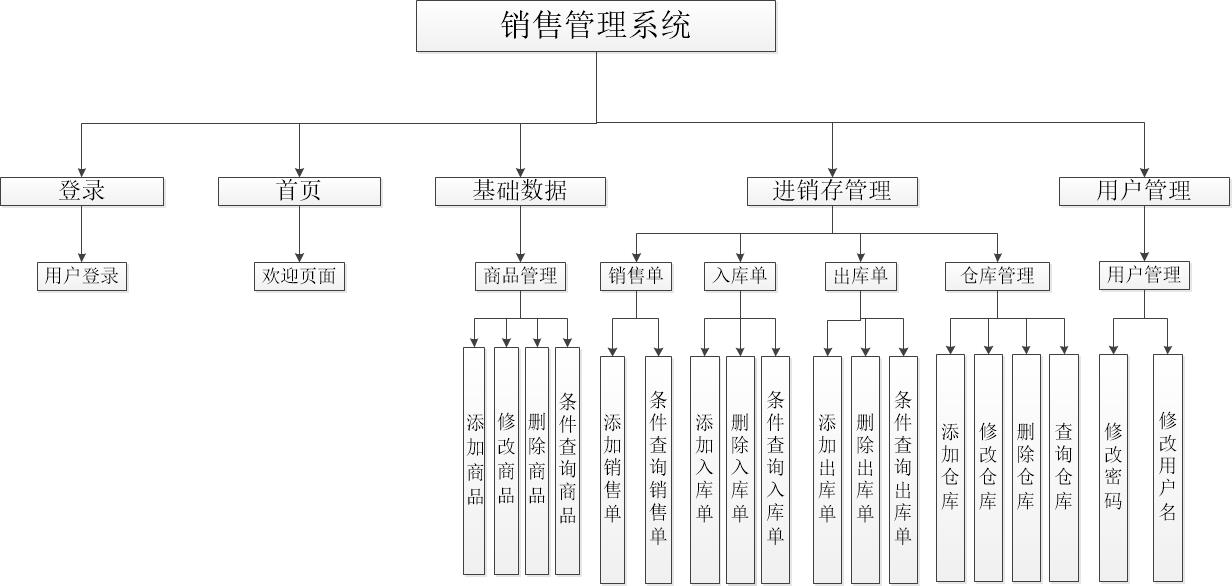
# 销售管理系统设计文档

## 系统功能图

此套系统首先分为五大部分，分别为登录、首页、基础数据、进销存管理、用户管理，首页为欢迎页面，基础数据下设商品管理功能模块，进销存管理下设销售单、入库单、出库单、仓库管理四个功能模块，用户管理为单独功能模块。

功能结构图如下所示：



## E-R图

根据功能需求分析，要有一个用户表，用户具有id、姓名、密码、身份标识四个属性，用于登陆系统；

而整个销售管理系统的核心是商品，那么就产生的商品表，商品表拥有id、姓名、价格、产地、库存、所属仓库、所属分类、删除标识八个属性；

而商品与仓库、分类均为多对一的逻辑关系，那么就产生了仓库表和分类表，仓库表具有id、名称、删除标识、排序规则四个属性，分类表具有id、名称、删除标识三个属性；

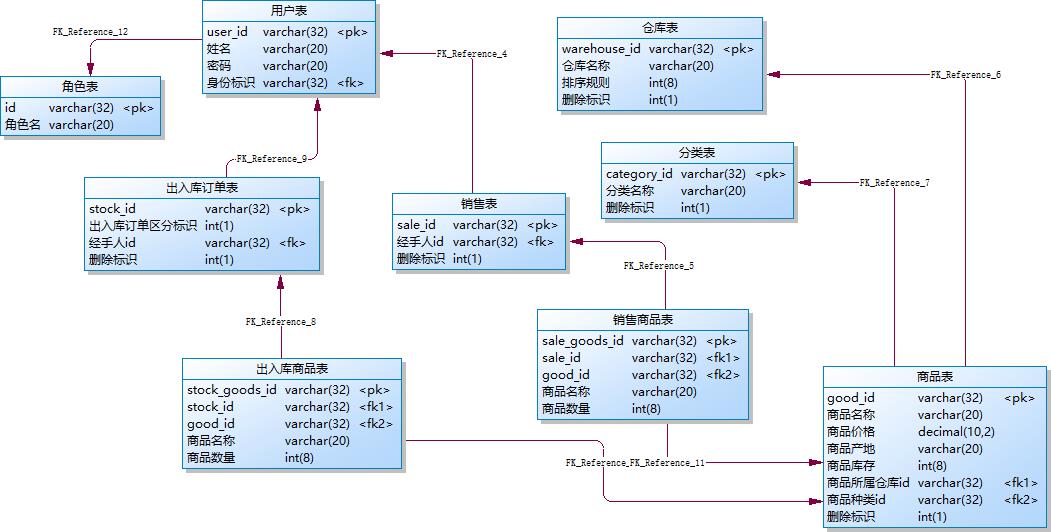
在销售管理过程中，势必会生成订单，根据功能需求分析，首先要将商品入库，这样就生成了入库单，入库单具有id、订单号、经手人id、商品所属仓库id、商品所属分类id、入库数量、商品id、删除标识八个属性；

而在销售中，每有一笔生意，就会生成销售单，需要建立销售表，销售单具有id、订单号、经手人id、商品所属仓库id、商品所属分类id、销售数量、商品id、删除标识八个属性；

在商品销售以及库存调动中又会涉及到出库，这样就生成了出库单，出库单具有id、订单号、经手人id、商品所属仓库id、商品所属分类id、出库数量、商品id、删除标识八个属性，为了数据交互性能的提升，将入库单与出库单合并产生出入库表，他们各自多了一条属性，即出入库标识；

这便是整套系统的数据库需求。

如下为整套系统的E-R设计图：



## 数据库的逻辑结构设计

⑴用户表逻辑结构

表2 用户表逻辑结构表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性 | 数据类型 | 域范围 | 允许为空 | 主外键说明 | 默认值 |
| id | 字符型 | 最大32个字符 | 否 | 主键 | 无 |
| 用户名 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 密码 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 身份标识 | 整型 | 最大1个字符 | 是 | 无 | 0 |

⑵分类表逻辑结构

表3 分类表逻辑结构表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性 | 数据类型 | 域范围 | 允许为空 | 主外键说明 | 默认 |
| id | 字符型 | 最大32个字符 | 否 | 主键 | 无 |
| 分类名 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 删除标识 | 整型 | 最大1个字符 | 是 | 无 | 0 |

⑶仓库表逻辑结构

表4 仓库表逻辑结构表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性 | 数据类型 | 域范围 | 允许为空 | 主外键说明 | 默认 |
| id | 字符型 | 最大32个字符 | 否 | 主键 | 无 |
| 仓库名 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 删除标识 | 整型 | 最大1个字符 | 是 | 无 | 0 |
| 排序规则 | 整型 | 最大10个字符 | 是 | 无 | 0 |

⑷商品表逻辑结构

表5 商品表逻辑结构表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性 | 数据类型 | 域范围 | 允许为空 | 主外键说明 | 默认 |
| id | 字符型 | 最大32个字符 | 否 | 主键 | 无 |
| 商品名 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 价格 | 精确值类型 | 最大10个字符，两位小数 | 否 | 无 | 无 |
| 产地 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 库存 | 实型 | 无 | 否 | 无 | 无 |
| 所属仓库id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 所属分类id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 删除标识 | 整型 | 最大1个字符 | 是 | 无 | 0 |

⑸销售订单表逻辑结构

表6 销售订单表逻辑结构表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性 | 数据类型 | 域范围 | 允许为空 | 主外键说明 | 默认 |
| id | 字符型 | 最大32个字符 | 否 | 主键 | 无 |
| 订单号 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 经手人id | 实型 | 最大32个字符 | 否 | 外键 | 无 |
| 所属仓库id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 所属分类id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 销售数量 | 实型 | 无 | 否 | 无 | 无 |
| 商品id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 删除标识 | 整型 | 最大1个字符 | 是 | 无 | 0 |

⑹出入库订单表逻辑结构

表7 出入库订单表逻辑结构表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性 | 数据类型 | 域范围 | 允许为空 | 主外键说明 | 默认 |
| id | 字符型 | 最大32个字符 | 否 | 主键 | 无 |
| 订单号 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 经手人id | 实型 | 最大32个字符 | 否 | 外键 | 无 |
| 所属仓库id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 所属分类id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 销售数量 | 实型 | 无 | 否 | 无 | 无 |
| 商品id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 出入库标识 | 整形 | 最大1个字符 | 否 | 无 | 无 |
| 删除标识 | 整型 | 最大1个字符 | 是 | 无 | 0 |

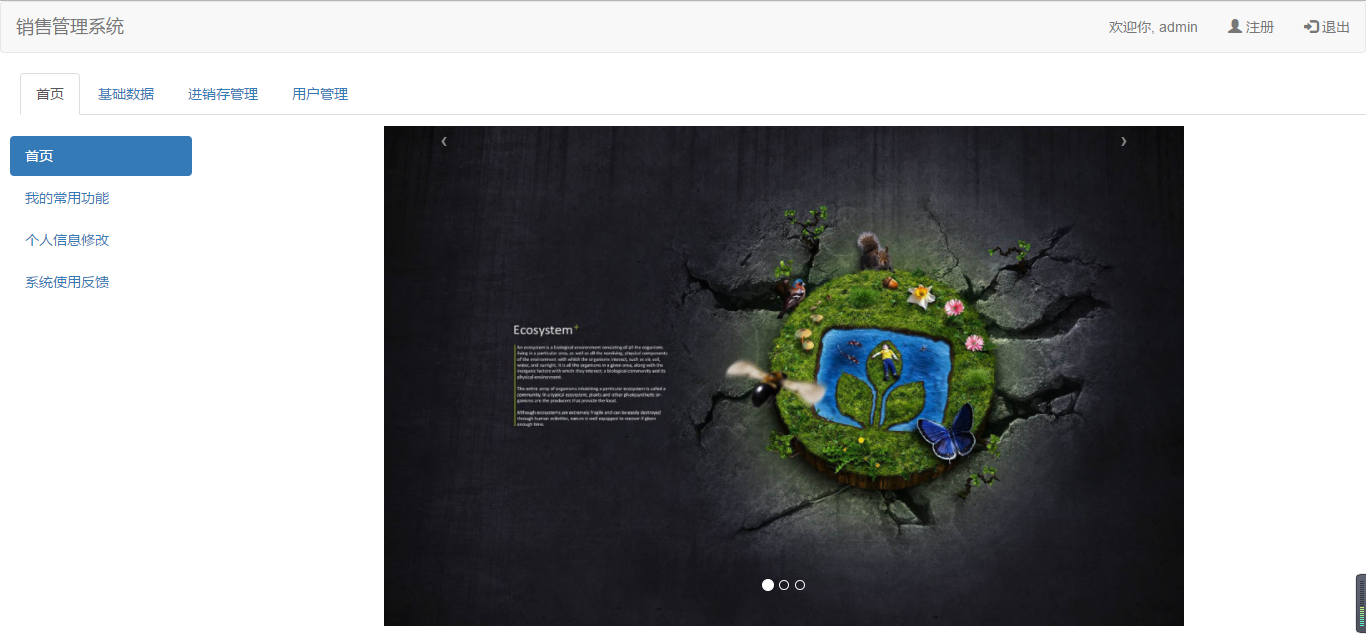
## 系统实现

### 系统界面布局设计

登陆界面：



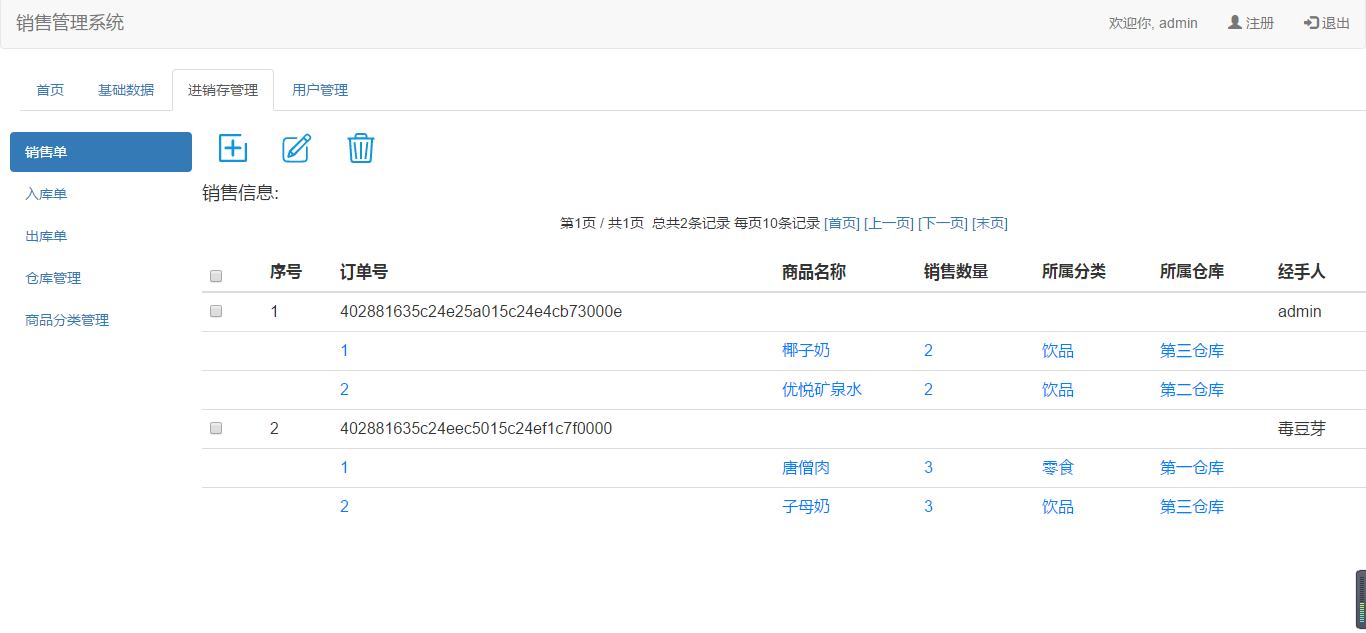
首页：



商品页面：



销售、库存、仓库、分类管理页面：

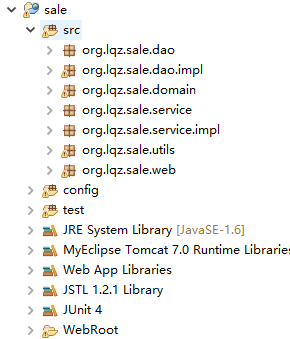


用户、权限管理页面：



### 代码实现：

#### 如下为系统工程结构图：



#### DAO层代码设计：

采用java泛型，基于Hibernate框架，编写了**通用的**基础数据增删改查方法，实现了基础数据库操作的接口。

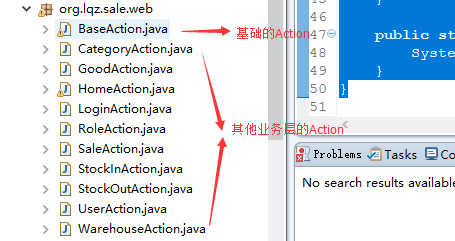
**BaseDao**

|  |
| --- |
| public interface BaseDao {  // 查询所有，带条件查询  public <T> List<T> find(String hql, Class<T> entityClass, Object[] params);  // 获取一条记录  public <T> T get(Class<T> entityClass, Serializable id);  // 分页查询，将数据封装到一个page分页工具类对象  public <T> Page<T> findPage(String hql, Page<T> page, Class<T> entityClass, Object[] params);  // 新增和修改保存  public <T> void saveOrUpdate(T entity);  // 批量新增和修改保存  public <T> void saveOrUpdateAll(Collection<T> entitys);  // 单条删除，按id  public <T> void deleteById(Class<T> entityClass, Serializable id);  // 批量删除  public <T> void delete(Class<T> entityClass, Serializable[] ids);  } |

**BaseDaoImpl**

|  |
| --- |
| public class BaseDaoImpl implements BaseDao {    private SessionFactory sessionFactory;  public void setSessionFactory(SessionFactory sessionFactory) {  this.sessionFactory = sessionFactory;  }  public Session getSession() {  return sessionFactory.getCurrentSession();  }  // 带条件查询  public <T> List<T> find(String hql, Class<T> entityClass, Object[] params) {  Query query = this.getSession().createQuery(hql);  if (params != null) {  for (int i = 0; i < params.length; i++) {  query.setParameter(i, params[i]);  }  }  return (List<T>) query.list();  }  // 获取一条，根据主键id  public <T> T get(Class<T> entityClass, Serializable id) {  return (T) this.getSession().get(entityClass, id);  }  // 分页查询，查询两次，一次查询总数，一次查询分页记录  public <T> Page<T> findPage(String hql, Page<T> page, Class<T> entityClass, Object[] params) {  Query query = this.getSession().createQuery(hql);  if (params != null) {  for (int i = 0; i < params.length; i++) {  query.setParameter(i, params[i]);  }  }  // 查询一次，获取记录总数  int count = query.list().size(); // 可以优化？？？？ "select count(\*) " +hql  page.setTotalRecord(count);  // 设置分页  query.setFirstResult((page.getPageNo() - 1) \* page.getPageSize()); // 设置开始位置  query.setMaxResults(page.getPageSize()); // 设置获取几条  page.setResults((List<T>) query.list());  return page;  }  // 新增和修改，hibernate根据id是否为null自动判断  public <T> void saveOrUpdate(T entity) {  this.getSession().saveOrUpdate(entity);  }  // 集合保存，这时新增还是修改，就自动判断，调用时是否简洁。适合批量新增和修改时。（Mrecord控件）  public <T> void saveOrUpdateAll(Collection<T> entitys) {  for (T entity : entitys) {  this.saveOrUpdate(entity);// 为什么hibernate批量操作时，要用循环一条一条记录去更新？  }  }  // 按主键id删除  public <T> void deleteById(Class<T> entityClass, Serializable id) {  this.getSession().delete(get(entityClass, id));  }  // 批量删除  public <T> void delete(Class<T> entityClass, Serializable[] ids) {  for (Serializable s : ids) {  deleteById(entityClass, s);  }  }  } |

#### Action层代码设计：



**编写BaseAction，实现一些公共的方法，如获取request、application，对值栈的处理，获取当前用户等**

**BaseAction**

|  |
| --- |
| public class BaseAction extends ActionSupport implements ServletRequestAware, ServletContextAware {  private static Logger log = Logger.getLogger(BaseAction.class);  private static final long serialVersionUID = 1L;  protected HttpServletRequest request;  protected HttpSession session;  protected ServletContext application;  @Override  public void setServletContext(ServletContext arg0) {  this.application = arg0;  }  @Override  public void setServletRequest(HttpServletRequest arg0) {  this.request = arg0;  try {  request.setCharacterEncoding("utf-8");  } catch (UnsupportedEncodingException e) {  e.printStackTrace();  }  this.session = request.getSession();  }  /\*\*  \* 将对象放入值栈的栈顶  \*/  public void push(Object obj) {  ActionContext.getContext().getValueStack().push(obj);  }  /\*\*  \* 将key-value对放入值栈的 context中  \*/  public void put(String key, Object value) {  ActionContext.getContext().put(key, value);  }  /\*\*  \* 获取当前登录用户的信息  \*/  public User getCurUser() {  User user = (User) session.getAttribute(SysConstant.CURRENT\_USER\_INFO);  return user;  }  } |

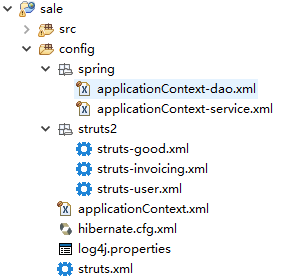
**其他业务层的Action设计结构都差不多，分别包括分页查询，进入新增、修改页面，进行新增、修改、删除操作等。**

**下面就只分析GoodAction的设计：**

**GoodAction**

|  |
| --- |
| public class GoodAction extends BaseAction implements ModelDriven<Good> {  private static final long serialVersionUID = 1202675939258668831L;  private Good model = new Good();  @Override  public Good getModel() {  return model;  }  // 分页查询的类  private Page<Good> page = new Page<Good>();  public Page<Good> getPage() {  return page;  }  public void setPage(Page<Good> page) {  this.page = page;  }  private GoodService goodService = new GoodServiceImpl();  private CategoryService categoryService = new CategoryServiceImpl();  private WarehouseService warehouseService = new WarehouseServiceImpl();  /\*\*  \* 分页查询  \*/  public String list() throws Exception {    String hql = "from Good where delFlag = 0";  goodService.findPage(hql, page, Good.class, null);  // 设置分页的url地址  page.setUrl("goodAction\_list");  // 将page对象压入栈顶  super.push(page);    return "list";  }  /\*\*  \* 查看  \*/  public String toview() throws Exception {  // 1.调用业务方法，根据id,得到对象  Good good = goodService.get(Good.class, model.getId());  // 放入栈顶  super.push(good);  // 3.跳页面  return "toview";  }  /\*\*  \* 进入新增页面  \*/  public String tocreate() throws Exception {  // 查询分类  List<Category> cates = categoryService.find("from Category where delFlag = 0", Category.class, null);    // 查询仓库  List<Warehouse> wares = warehouseService.find("from Warehouse where delFlag = 0", Warehouse.class, null);    super.put("categorys", cates);    super.put("warehouses", wares);    return "tocreate";  }  /\*\*  \* 保存  \*/  public String insert() throws Exception {  goodService.saveOrUpdate(model);  return "alist";  }  /\*\*  \* 进入修改页面  \*/  public String toupdate() throws Exception {  Good good = goodService.get(Good.class, model.getId());  super.push(good);    // 查询分类  List<Category> cates = categoryService.find("from Category where delFlag = 0", Category.class, null);  // 查询仓库  List<Warehouse> wares = warehouseService.find("from Warehouse where delFlag = 0", Warehouse.class, null);  super.put("categorys", cates);  super.put("warehouses", wares);    return "toupdate";  }  /\*\*  \* 修改  \*/  public String update() throws Exception {  Good good = goodService.get(Good.class, model.getId());    good.setName(model.getName());  good.setOrigin(model.getOrigin());  good.setPrice(model.getPrice());  good.setStock(model.getStock());  good.setCategory(model.getCategory());  good.setWarehouse(model.getWarehouse());    goodService.saveOrUpdate(good);  return "alist";  }  /\*\*  \* 删除  \*/  public String delete() throws Exception {  String ids[] = model.getId().split(", ");    //调用业务方法，实现批量删除  goodService.delete(Good.class, ids);    return "alist";  }  } |

#### 各类配置文件



Struts.xml

|  |
| --- |
| <struts>  <constant name="struts.ui.theme" value="simple"/>  <constant name="struts.devMode" value="true" />  <constant name="struts.i18n.encoding" value="UTF-8" />  <package name="default" namespace="/" extends="struts-default">    <action name="login" method="login" class="org.lqz.sale.web.LoginAction">  <result name="login">/WEB-INF/pages/sysadmin/login/login.jsp</result>  <result name="success">/WEB-INF/pages/home/fmain.jsp</result>  </action>  <action name="logout" method="logout" class="org.lqz.sale.web.LoginAction">  <result name="logout">/WEB-INF/pages/sysadmin/login/login.jsp</result>  </action>    <action name="homeAction\_\*" method="{1}" class="org.lqz.sale.web.HomeAction">  <result name="toTitle">/WEB-INF/pages/home/title.jsp</result>  <result name="toLeft">/WEB-INF/pages/${moduleName}/left.jsp</result>  <result name="toMain">/WEB-INF/pages/${moduleName}/main.jsp</result>  </action>    </package>    <include file="struts2/struts-invoicing.xml"></include>  <include file="struts2/struts-user.xml"></include>  <include file="struts2/struts-good.xml"></include>    </struts> |

Struts-**good**.xml

|  |
| --- |
| <struts>  <constant name="struts.ui.theme" value="simple"/>  <constant name="struts.devMode" value="true" />  <constant name="struts.i18n.encoding" value="UTF-8" />  <package name="goods" namespace="/goods" extends="default">  <action name="goodAction\_\*" class="org.lqz.sale.web.GoodAction" method="{1}">  <result name="list">/WEB-INF/pages/goods/jGoodList.jsp</result>  <result name="toview">/WEB-INF/pages/goods/jGoodView.jsp</result>  <result name="tocreate">/WEB-INF/pages/goods/jGoodCreate.jsp</result>  <result name="toupdate">/WEB-INF/pages/goods/jGoodUpdate.jsp</result>  <result name="alist" type="redirectAction">goodAction\_list</result>  </action>  </package>  </struts> |

**Hibernate.cfg.xml**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <session-factory>  <property name="hibernate.dialect">  org.hibernate.dialect.MySQLDialect  </property>  <property name="show\_sql">false</property>  <property name="format\_sql">false</property>  <property name="hbm2ddl.auto">update</property><!-- none、update -->  <!-- 懒加载，配合web.xml中配置的 openSessionInViewFilter -->  <property name="hibernate.enable\_lazy\_load\_no\_trans">true</property>  <!--校验模式 JPA java persistent api-->  <property name="javax.persistence.validation.mode">none</property>  <mapping resource="org/lqz/sale/domain/Category.hbm.xml" />  <mapping resource="org/lqz/sale/domain/Good.hbm.xml" />  <mapping resource="org/lqz/sale/domain/Sale.hbm.xml" />  <mapping resource="org/lqz/sale/domain/SaleGoods.hbm.xml" />  <mapping resource="org/lqz/sale/domain/Stock.hbm.xml" />  <mapping resource="org/lqz/sale/domain/StockGoods.hbm.xml" />  <mapping resource="org/lqz/sale/domain/User.hbm.xml" />  <mapping resource="org/lqz/sale/domain/Warehouse.hbm.xml" />  <mapping resource="org/lqz/sale/domain/Role.hbm.xml"/>  </session-factory>  </hibernate-configuration> |

**applicationContext.xml**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"  xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:aop="http://www.springframework.org/schema/aop"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-4.3.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd">  <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">  <property name="driverClass" value="com.mysql.jdbc.Driver" />  <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/sale?useUnicode=true&amp;characterEncoding=utf-8" />  <property name="user" value="root" />  <property name="password" value="1234" />  </bean>  <!-- 2.sessionFactory -->  <bean id="sessionFactory"  class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">  <property name="dataSource" ref="dataSource"></property>  <property name="configLocation" value="classpath:hibernate.cfg.xml"></property>  </bean>  <!-- 3.事务管理器 -->  <bean id="transactionManager"  class="org.springframework.orm.hibernate3.HibernateTransactionManager">  <property name="sessionFactory" ref="sessionFactory"></property>  </bean>  <!-- 4.txAdvice -->  <tx:advice id="txAdvice" transaction-manager="transactionManager">  <tx:attributes>  <tx:method name="save\*" propagation="REQUIRED" />  <tx:method name="insert\*" propagation="REQUIRED" />  <tx:method name="update\*" propagation="REQUIRED" />  <tx:method name="delete\*" propagation="REQUIRED" />  <tx:method name="get\*" read-only="true" />  <tx:method name="\*" propagation="REQUIRED" />  </tx:attributes>  </tx:advice>  <!-- 5.aop -->  <aop:config>  <aop:pointcut id="pointcut" expression="execution(\* org.lqz.sale.service.\*.\*(..))" />  <aop:advisor advice-ref="txAdvice" pointcut-ref="pointcut" />  </aop:config>  <import resource="classpath:spring/applicationContext-dao.xml"/>  <import resource="classpath:spring/applicationContext-service.xml"/>  </beans> |

其他的请具体看源码，谅解！