[配置文件 1](#_Toc250045081)

[web.xml 1](#_Toc250045082)

[struts.xml 3](#_Toc250045083)

[applicationContext-framework.xml 3](#_Toc250045084)

[框架代码 6](#_Toc250045085)

[BaseDao 6](#_Toc250045086)

[DaoImp的写法: 12](#_Toc250045087)

# 配置文件

## web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app version=*"2.5"* xmlns=*"http://java.sun.com/xml/ns/javaee"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*  *http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*> |
| <!-- 配置spring -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>/WEB-INF/applicationContext.xml</param-value>  <!-- 或者<param-value>classpath:/context/applicationContext-\*.xml</param-value> -->  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener> |
| <!-- 延长session的生命周期, 直到页面显示完了才关闭session ; 要放在Struts2配置之前-->  <filter>  <filter-name>openSessionInView</filter-name>  <filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>openSessionInView</filter-name>  <url-pattern>\*.action</url-pattern>  </filter-mapping> |
| <!-- 配置struts2 -->  <filter>  <filter-name>struts2</filter-name>  <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>  </filter>  <filter-mapping>  <filter-name>struts2</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |
| <!-- 解决乱码方案 -->  <filter>  <filter-name>encoding</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>encoding</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |
| <!-- 欢迎文件 -->  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>  </web-app> |

## struts.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"  "http://struts.apache.org/dtds/struts-2.0.dtd">  <struts> |
| <!-- 使用Spring的IOC管理Struts的Action -->  <constant name=*"struts.objectFactory"* value=*"spring"*/> |
| <package name=*"common"* extends=*"struts-default"* >  <!-- 这里到class值为Spring配置文件中到一个Bean的ID -->  <action name=*"login"* class=*"loginAction"* method=*"login"*>  <result name=*"success"*>/common/main.jsp</result>  <result name=*"error"*>/login.jsp</result>  </action>  </package>  </struts> |

## applicationContext-framework.xml

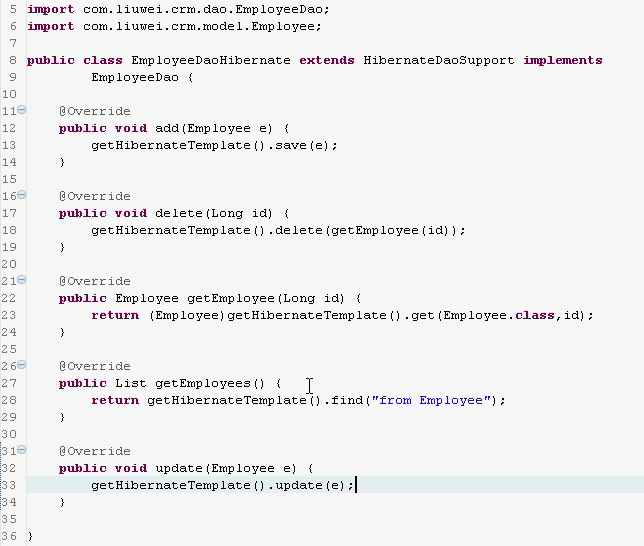
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd"*> |
| <!-- apache.dbcp连接池的配置 需要增加两个jar文件: commons-dbcp 和 commons-pool -->  <bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"* destroy-method=*"close"*>  <property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver" /*>  <property name=*"url"* value=*"jdbc:mysql://localhost:3306/s2sh?useUnicode=true&amp;characterEncoding=UTF-8" /*>  <property name=*"username"* value=*"root" /*>  <property name=*"password"* value=*"root" /*>  <!-- 最大活动连接数 -->  <property name=*"maxActive"* value=*"100" /*>  <!-- 最大可空闲连接数 -->  <property name=*"maxIdle"* value=*"30" /*>  <!-- 最大可等待连接数 -->  <property name=*"maxWait"* value=*"500" /*>  <!-- 默认的提交方式（如果不需要事务可以设置成true，在实际应用中一般设置为false,默认为 false） -->  <property name=*"defaultAutoCommit"* value=*"true" /*>  </bean> |
| <!-- sessionFactory -->  <bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate3.LocalSessionFactoryBean"*>  <property name=*"dataSource"* ref=*"dataSource" /*>  <!-- hibernate方言等相关配置 -->  <property name=*"hibernateProperties"*>  <props>  <prop key=*"connection.useUnicode"*>true</prop>  <prop key=*"connection.characterEncoding"*>utf-8</prop>  <prop key=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</prop>  <prop key=*"hibernate.show\_sql"*>true</prop>  </props>  </property>  <!-- hbm.xml的映射文件 -->  <property name=*"mappingResources"*>  <list>  <value>cn/edu/swu/thesis/framework/entity/Agenda.hbm.xml</value>  <value>cn/edu/swu/thesis/framework/entity/Classes.hbm.xml</value>  <value>cn/edu/swu/thesis/framework/entity/Department.hbm.xml</value>  <value>cn/edu/swu/thesis/framework/entity/Message.hbm.xml</value>  <value>cn/edu/swu/thesis/framework/entity/Module.hbm.xml</value>  <value>cn/edu/swu/thesis/framework/entity/News.hbm.xml</value>  </list>  </property>  </bean> |
| <!-- 事务管理 -->  <bean id=*"transactionManger"*  class=*"org.springframework.orm.hibernate3.HibernateTransactionManager"*>  <property name=*"sessionFactory"* ref=*"sessionFactory"* />  </bean>  <!-- 声明式处理事务 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"transactionManger"*>  <tx:attributes>  <tx:method name=*"add\*"* propagation=*"REQUIRED"* />  <tx:method name=*"del\*"* propagation=*"REQUIRED"* />  <tx:method name=*"upd\*"* propagation=*"REQUIRED"* />  <tx:method name=*"\*"* read-only=*"true"* />  </tx:attributes>  </tx:advice>  <!-- 设定切面, 配置哪些类的方法进行事务管理 -->  <aop:config>  <aop:pointcut expression=*"execution(\* com.\*.\*.biz.impl.\*.\*(..))"*  id=*"managerMethod"* />  <aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"managerMethod"* />  </aop:config> |
| <!-- 配置DAO, BIZ, Action -->  <bean id=*"iUserDao"* class=*"cn.edu.swu.thesis.common.dao.imp.UserDaoImp"* scope=*"singleton"*>  <!-- 由于该类是继承于HibernateDaoSupport，所以需要将 sessionFactory注入 -->  <property name=*"sessionFactory"* ref=*"sessionFactory"* />  </bean>  <bean id=*"iUserBiz"* class=*"cn.edu.swu.thesis.common.biz.imp.UserBizImp"*>  <property name=*"iUserDao"* ref=*"iUserDao"*></property>  </bean>  <!-- 配置Actin 因为Struts2是每个请求创建一个Servlet, 所以必需设置scope ; 在里面可以配置需要注入到各种东东-->  <bean id=*"loginAction"* class=*"cn.edu.swu.thesis.common.action.LoginAction"* scope=*"prototype"*>  <property name=*"iUserBiz"* ref=*"iUserBiz"*></property>  </bean> |
| </beans> |

# 框架代码

## BaseDao

|  |
| --- |
| **extends** HibernateDaoSupport  **public** **class** BaseDao **extends** HibernateDaoSupport {  /\*\*  \* 通过ID获得对象  \*/  **public** <T> T getEntityByID(Class<T> entityClass, Serializable id) {  **return** (T) getHibernateTemplate().get(entityClass, id);  }    /\*\*  \* 更新数据，更新成功返回true；  \* 失败返回false；  \* \*/  **public** **boolean** bUpdate(Object entity) {  **try**{  **this**.getHibernateTemplate().update(entity);  }  **catch**(Exception e){  e.printStackTrace();  **return** **false**;  }  **return** **true**;    }    /\*\*  \* 获得所有数据集  \*/  **public** <T> List<T> getAll(Class<T> entityClass) {  **return** getHibernateTemplate().loadAll(entityClass);  }  /\*\*  \* 获得所有数据集，通过升序或降序 isAsc trure为升序 isAsc false为降序  \*/  **public** <T> List<T> getAll(Class<T> entityClass, String orderBy,  **boolean** isAsc) {  Assert.*hasText*(orderBy);  **if** (isAsc)  **return** getHibernateTemplate().findByCriteria(  DetachedCriteria.*forClass*(entityClass).addOrder(  Order.*asc*(orderBy)));  **else**  **return** getHibernateTemplate().findByCriteria(  DetachedCriteria.*forClass*(entityClass).addOrder(  Order.*desc*(orderBy)));  }    /\*\*  \* 更新数据  \* \*/  **public** **void** update(Object obj){  getHibernateTemplate().update(obj);  }    /\*\*  \* 保存一条数据  \*/  **public** **void** save(Object o) {  getHibernateTemplate().saveOrUpdate(o);  }  /\*\*  \* 保存多条记录  \*/  **public** <T> **void** saveDataList(List<T> dataList) {  // 记录存储的条数,每存放20条，清除一次内存  **int** count = 0;  **for** (Iterator itr = dataList.iterator(); itr.hasNext();) {  Class<T> obj = (Class<T>) itr.next();  **this**.save(obj);  count++;  **if** (count == 20) {  count = 0;  **this**.flush();  }  }  }  /\*\*  \* 删除一条数据  \*/  **public** **void** remove(Object o) {  getHibernateTemplate().delete(o);  }  /\*\*  \* 删除一批数据  \*/  **public** <T> **void** removeDataList(List<T> dataList) {  // 记录删除的条数,每删除20条，清除一次内存  **int** count = 0;  **for** (Iterator itr = dataList.iterator(); itr.hasNext();) {  Class<T> obj = (Class<T>) itr.next();  **this**.remove(obj);  count++;  **if** (count == 20) {  count = 0;  **this**.flush();  }  }  }  /\*\*  \* 通过ID删除一条数据  \*/  **public** <T> **void** removeById(Class<T> entityClass, Serializable id) {  remove(**this**.getEntityByID(entityClass, id));  }  **public** **void** flush() {  getHibernateTemplate().flush();  }  **public** **void** clear() {  getHibernateTemplate().clear();  }  /\*\*  \* 临时构造条件查询 Query查询  \*/  **public** Query createQuery(String hql, Object... values) {  Assert.*hasText*(hql);  Query query = getSession().createQuery(hql);  **for** (**int** i = 0; i < values.length; i++) {  query.setParameter(i, values[i]);  }  **return** query;  }  /\*\*  \* Query查询  \*/  **public** Query createQuery(String hql) {  Assert.*hasText*(hql);  **return** getSession().createQuery(hql);  }  /\*\*  \* Criteria查询  \*/  **public** <T> Criteria createCriteria(Class<T> entityClass,  Criterion... criterions) {  Criteria criteria = getSession().createCriteria(entityClass);  **for** (Criterion c : criterions) {  criteria.add(c);  }  **return** criteria;  }  /\*\*  \* 临时构造条件查询 直接查询HQL  \*/  **public** <T> List<T> find(String hql, Object... values) {  Assert.*hasText*(hql);  **return** (List<T>) getHibernateTemplate().find(hql, values);  }  /\*\*  \* 直接查询HQL  \*/  **public** <T> List<T> find(String hql) {  Assert.*hasText*(hql);  **return** (List<T>) getHibernateTemplate().find(hql);  }  /\*\*  \* 通过实体的某个成员进行简单查询  \*/  **public** <T> List<T> findBy(Class<T> entityClass, String propertyName,  Object value) {  Assert.*hasText*(propertyName);  **return** createCriteria(entityClass, Restrictions.*eq*(propertyName, value))  .list();  }  /\*\*  \* 删除HQL语句中select词语  \*/  **private** **static** String removeSelect(String hql) {  Assert.*hasText*(hql);  **int** beginPos = hql.toLowerCase().indexOf("from");  Assert.*isTrue*(beginPos != -1, " hql : " + hql  + " must has a keyword 'from'");  **return** hql.substring(beginPos);  }  /\*\*  \*删除HQL语句中order词语  \*  \*\*/  **private** **static** String removeOrders(String hql) {  Assert.*hasText*(hql);  Pattern p = Pattern.*compile*("order\\s\*by[\\w|\\W|\\s|\\S]\*",  Pattern.*CASE\_INSENSITIVE*);  Matcher m = p.matcher(hql);  StringBuffer sb = **new** StringBuffer();  **while** (m.find()) {  m.appendReplacement(sb, "");  }  m.appendTail(sb);  **return** sb.toString();  }  } |

一个简单的dao实现



## DaoImp的写法:

|  |
| --- |
| **public** **class** AdvisoryDaoImpl **extends** BaseDao **implements** IAdvisoryDao {  /\* |