## BaseAction:

\* 表现层通用实现

\* @author zhaoqx

\*

\* @param <T>

\*/

public class **BaseAction<T>** extends **ActionSupport implements ModelDriven<T>** {

protected PageBean pageBean = new PageBean();

//模型对象

**protected T model;/**

**public T getModel() {**

**return model;**

**}**

/在构造方法中动态获取实体类型，

**1.通过反射创建model对象**

**2.创建离线查询对象，并把离线对象封装到pageBean中**

public BaseAction() {

//获得父类

**ParameterizedType genericSuperclass = (ParameterizedType) this.getClass().getGenericSuperclass();**

**//获得BaseAction上声明的泛型数组**

**Type[] actualTypeArguments = genericSuperclass.getActualTypeArguments();**

**//得到类型的字节码文件**

**Class<T> entityClass = (Class<T>) actualTypeArguments[0];**

**detachedCriteria = DetachedCriteria.forClass(entityClass);**

**pageBean.setDetachedCriteria(detachedCriteria);**

//通过反射创建对象

try {

//通过字节码对象获得对象

**model = entityClass.newInstance();**

} catch (InstantiationException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

}

}

}

//创建离线提交查询对象

DetachedCriteria detachedCriteria = null;//在这里申明是为了其他的方法也可以调用

**//获得前台传入的page 和rows（使用datagrid进行分页查询）**

**public void setPage(int page) {**

**pageBean.setCurrentPage(page);**

**}**

**public void setRows(int rows) {**

**pageBean.setPageSize(rows);**

**}**

/\*\*

\* **将指定对象转为json字符串，并响应到客户端页面(传入对象，对象去除的字段)**

\* @param o

\* @param exclueds

\*/

public void java2Json(**Object o ,String[] exclueds**){

JsonConfig jsonConfig = new JsonConfig();

//指定哪些属性不需要转json

**jsonConfig.setExcludes(exclueds);**

**String json = JSONObject.fromObject(o,jsonConfig).toString();**

**ServletActionContext.getResponse().setContentType("text/json;charset=utf-8");**

try {

**ServletActionContext.getResponse().getWriter().print(json);**

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* 将指定Java对象转为json，**并响应到客户端页面（传入对象的集合,，去除的字段）**

\* @param o

\* @param exclueds

\*/

Exception e) {

e.printStackTrace();

}

}

**public static final String HOME = "home";**

**public static final String LIST = "list";**

**继承类：**

/\*\*

\* 取派员管理

\* @author zhaoqx

\*

\*/

**@Controller**

**@Scope("prototype") 在实例化子类的同时 BaseAction也被实例化了**

public class StaffAction extends BaseAction<Staff>{

@Autowired

private IStaffService staffService;//不用写get、set方法

/\*\*

\* 添加取派员

\*/

public String add(){

staffService.save(**model**); **//子类的成员变量不用声明可以直接使用**

return **LIST;**

}

/\*\*

\* 分页查询方法

\* @throws IOException

\*/

public String pageQuery() throws IOException{

**staffService.pageQuery(pageBean);//因为多次要用到所以把他放到baseAction中**

**this.java2Json**(pageBean, new **String[]{"decidedzones","currentPage","detachedCriteria","pageSize"})**;

return NONE;

}

//属性驱动，接收页面提交的ids参数

**private String ids;**

/\*\*

\* 取派员批量删除

\*/

@RequiresPermissions("staff-delete")//执行这个方法，需要当前用户具有staff-delete这个权限

public String deleteBatch(){

staffService.deleteBatch(ids);

return LIST;

}

/\*\*

\* 修改取派员信息

\*/

@RequiresPermissions("staff-edit")

public String edit(){

//Subject subject = SecurityUtils.getSubject();

//subject.checkPermission("staff-edit");

//显查询数据库，根据id查询原始数据

Staff staff = staffService.findById(model.getId());

//使用页面提交的数据进行覆盖

staff.setName(model.getName());

staff.setTelephone(model.getTelephone());

staff.setHaspda(model.getHaspda());

staff.setStandard(model.getStandard());

staff.setStation(model.getStation());

staffService.update(staff);

return LIST;

}

/\*public static void main(String[] args) {

PageBean pageBean = new PageBean();

pageBean.setCurrentPage(2);

pageBean.setPageSize(30);

String json = JSONObject.fromObject(pageBean ).toString();

System.out.println(json);

}\*/

/\*\*

\* 查询所有未删除的取派员，返回json

\*/

public String listajax(){

List<Staff> list = staffService.findListNotDelete();

this.java2Json(list, new String[]{"decidedzones"});

return NONE;

}

public String getIds() {

return ids;

}

public void setIds(String ids) {

this.ids = ids;

}

}

## BaseDao:

/\*\*

\* 持久层通用实现

\* @author zhaoqx

\*

\* @param <T>

\*/

public class BaseDaoImpl<T> **extends HibernateDaoSupport implements IBaseDao<T>** {

//代表的是某个实体的类型

private Class<T> entityClass;

**@Resource//根据类型注入spring工厂中的会话工厂对象sessionFactory**

**可以在set上，可以在属性上**

**public void setMySessionFactory(SessionFactory sessionFactory){**

**super.setSessionFactory(sessionFactory);**

**}**

//在父类（BaseDaoImpl）的构造方法中动态获得entityClass

public BaseDaoImpl() {

ParameterizedType superclass = (ParameterizedType) this.getClass().getGenericSuperclass();

//获得父类上声明的泛型数组

Type[] actualTypeArguments = superclass.getActualTypeArguments();

entityClass = (Class<T>) actualTypeArguments[0];

}

public void save(T entity) {

**this.getHibernateTemplate().save(entity);**

}

public void delete(T entity) {

**this.getHibernateTemplate().delete(entity);**

}

public void update(T entity) {

**this.getHibernateTemplate().update(entity);**

}

public T findById(Serializable id) {//查询一条记录

**return this.getHibernateTemplate().get(entityClass, id);**

}

public List<T> findAll() {//查询多条记录

String hql = "FROM " + entityClass.getSimpleName();？？

return (List<T>) this.getHibernateTemplate().find(hql);

}

//执行更新（自定义的更新 参数：SQL 参数数组）

public void executeUpdate(String queryName, Object... objects) {

Session session = **this.getSessionFactory().getCurrentSession();**

Query query = **session.getNamedQuery(queryName);**

int i = 0;

for (Object object : objects) {

//为HQL语句中的？赋值

**query.setParameter(i++, object);**

}

//执行更新

**query.executeUpdate();**

}

/\*\*

\* 通用分页查询方法

\*/

public void pageQuery(PageBean pageBean) {

int currentPage = pageBean.getCurrentPage();

int pageSize = pageBean.getPageSize();

**DetachedCriteria** detachedCriteria = pageBean.getDetachedCriteria();

//查询total---总数据量

**detachedCriteria.setProjection(Projections.rowCount());/**/指定hibernate框架发出sql的形式----》select count(\*) from bc\_staff;

**List<Long> countList = (List<Long>) this.getHibernateTemplate().findByCriteria(detachedCriteria);**

**Long count = countList.get(0);**

pageBean.setTotal(count.intValue());

//查询rows---当前页需要展示的数据集合

detachedCriteria.setProjection(null);//指定hibernate框架发出sql的形式----》select \* from bc\_staff;

**//指定hibernate框架封装对象的方式**

**detachedCriteria.setResultTransformer(DetachedCriteria.ROOT\_ENTITY);**

int firstResult = (currentPage - 1) \* pageSize;

int maxResults = pageSize;

List rows = **this.getHibernateTemplate().findByCriteria(detachedCriteria, firstResult, maxResults);**

pageBean.setRows(rows);

}

public void saveOrUpdate(T entity) {

**this.getHibernateTemplate().saveOrUpdate(entity);？？？**

}

public List<T> findByCriteria(DetachedCriteria detachedCriteria) {

**return (List<T>) this.getHibernateTemplate().findByCriteria(detachedCriteria);**

}

}

**继承类：**

**@Repository**

public class StaffDaoImpl extends BaseDaoImpl<Staff> implements IStaffDao{

}

## 配置文件：

### Web.xml

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" version="2.5">

<display-name>bos-web</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

**<!-- 配置spring框架提供的用于整合shiro框架的过滤器 -->**

**<filter>**

**<filter-name>shiroFilter</filter-name>**

**<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>**

**</filter>**

**<filter-mapping>**

**<filter-name>shiroFilter</filter-name>**

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

**</filter-mapping>**

<!-- 配置过滤器，解决hibernate延迟加载问题

**使用懒加载的时候 扩大session的作用范围** -->

**<filter>**

**<filter-name>openSessionInView</filter-name>**

**<filter-class>org.springframework.orm.hibernate5.support.OpenSessionInViewFilter</filter-class>**

**</filter>**

**<filter-mapping>**

**<filter-name>openSessionInView</filter-name>**

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

**</filter-mapping>**

<!-- 通过上下文参数指定spring配置文件位置 -->

**<context-param>**

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

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

**</context-param>**

**<!-- 配置spring框架的监听器 -->**

**<listener>**

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

**</listener>**

<!-- 配置struts2的过滤器  **其他过滤器拦截器应该放在这个之前**

**Struts2的拦截器可以帮他们做一些处理**

**-->**

**<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>**

**<dispatcher>REQUEST</dispatcher>//无论是转发还是重定向都需要经过拦截器**

**<dispatcher>FORWARD</dispatcher>**

**</filter-mapping>**

</web-app>

### appliactionContext.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:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:jaxws="http://cxf.apache.org/jaxws"

xmlns:soap="http://cxf.apache.org/bindings/soap"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.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://cxf.apache.org/bindings/soap

http://cxf.apache.org/schemas/configuration/soap.xsd

http://cxf.apache.org/jaxws

http://cxf.apache.org/schemas/jaxws.xsd

">

<!-- 加载属性文件 -->

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

<!-- 配置数据源 -->

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

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

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

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

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

</bean>

<!-- 配置LocalSessionFactoryBean，spring提供的用于整合hibernate的工厂bean -->

<bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">

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

<!-- 注入hibernate相关的属性配置 -->

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</prop>

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

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

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

</props>

</property>

<!-- 注入hibernate的映射文件 -->

<property name="mappingLocations">

<list>

<value>classpath:com/itheima/bos/domain/\*.xml</value>

</list>

</property>

</bean>

<!-- 配置事务管理器 -->

<bean id="transactionManager" class="org.springframework.orm.hibernate5.HibernateTransactionManager">

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

</bean>

**<!-- 组件扫描 -->**

**<context:component-scan base-package="com.itheima.bos"/>**

<!-- 支持spring注解 -->

**<context:annotation-config/>**

**<tx:annotation-driven/>**

<!-- 注册crm客户端代理对象 -->

**<jaxws:client id="crmClient"**

**serviceClass="com.itheima.crm.ICustomerService"**

**address="http://localhost:8080/crm\_heima32/service/customer"/>**

<!-- 配置shiro框架的过滤器工厂对象 -->

<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

<!-- 注入安全管理器对象 -->

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

<!-- 注入相关页面访问URL -->

<property name="loginUrl" value="/login.jsp"/>

<property name="successUrl" value="/index.jsp"/>

<property name="unauthorizedUrl" value="/unauthorized.jsp"/>

<!--注入URL拦截规则 -->

<property name="filterChainDefinitions">

<value>

/css/\*\* = anon

/js/\*\* = anon

/images/\*\* = anon

/validatecode.jsp\* = anon

/login.jsp = anon

/userAction\_login.action = anon

/page\_base\_staff.action = perms["staff-list"]

/\* = authc

</value>

</property>

</bean>

<!-- 注册安全管理器对象 -->

<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

<property name="realm" ref="bosRealm"/>

<!-- 注入缓存管理器 -->

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

</bean>

<!-- 注册缓存管理器 -->

<bean id="cacheManager" class="org.apache.shiro.cache.ehcache.EhCacheManager">

<!-- 注入ehcache的配置文件 -->

<property name="cacheManagerConfigFile" value="classpath:ehcache.xml"/>

</bean>

<!-- 注册realm -->

<bean id="bosRealm" class="com.itheima.bos.realm.BOSRealm">

</bean>

<!-- 开启shiro框架注解支持 -->

<bean id="defaultAdvisorAutoProxyCreator"

class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator">

<!-- 必须使用cglib方式为Action对象创建代理对象 -->

<property name="proxyTargetClass" value="true"/>

</bean>

<!-- 配置shiro框架提供的切面类，用于创建代理对象 -->

<bean class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor"/>

</beans>

### Struts.xml

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

<constant name="struts.devMode" value="true" />

<!--

<constant name="struts.objectFactory" value="spring"/>

-->

<package name="basicstruts2" extends="struts-default">

<interceptors>

**<!-- 注册自定义拦截器 -->**

<interceptor name="bosLoginInterceptor" class="com.itheima.bos.web.interceptor.BOSLoginInterceptor">

<!-- 指定哪些方法不需要拦截 -->

**<param name="excludeMethods">login</param>**

</interceptor>

**<!-- 定义拦截器栈 -->**

**<interceptor-stack name="myStack">**

**<interceptor-ref name="bosLoginInterceptor"></interceptor-ref>**

**<interceptor-ref name="defaultStack"></interceptor-ref>**

**</interceptor-stack>**

</interceptors>

<!--

<default-interceptor-ref name="myStack"/>

-->

<!-- 全局结果集定义 -->

**<global-results>**

**<result name="login">/login.jsp</result>**

**<result name="unauthorized">/unauthorized.jsp</result>**

**</global-results>**

<**global-exception-mappings>**

**<exception-mapping result="unauthorized"**

**exception="org.apache.shiro.authz.UnauthorizedException"></exception-mapping>**

**</global-exception-mappings>**

<!-- 需要进行权限控制的页面访问 -->

**<action name="page\_\*\_\*">**

**<result type="dispatcher">/WEB-INF/pages/{1}/{2}.jsp</result>**

**</action>**

<!-- 用户管理 -->

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

<result name="home">/index.jsp</result>

<result name="list">/WEB-INF/pages/admin/userlist.jsp</result>

</action>

<!-- 取派员管理 -->

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

<result name="list">/WEB-INF/pages/base/staff.jsp</result>

</action>

<!-- 区域管理 -->

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

</action>

<!-- 分区管理 -->

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

<result name="list">/WEB-INF/pages/base/subarea.jsp</result>

</action>

<!-- 定区管理 -->

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

<result name="list">/WEB-INF/pages/base/decidedzone.jsp</result>

</action>

<!-- 业务通知单管理 -->

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

<result name="noticebill\_add">/WEB-INF/pages/qupai/noticebill\_add.jsp</result>

</action>

<!-- 工作单 管理 -->

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

</action>

<!-- 权限管理 -->

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

<result name="list">/WEB-INF/pages/admin/function.jsp</result>

</action>

<!-- 角色管理 -->

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

<result name="list">/WEB-INF/pages/admin/role.jsp</result>

</action>

</package>

</struts>

### 映射文件

映射文件可以通过工具自动生成

注意：

**1、主键**

**2、那四个参数（**

**lazy（优化是设置，不过需要注意死循环和session问题） 默认使用懒加载 （true）**

**inverse（一的一方设置放弃 多对多设置，必须有一方放弃） 默认维护（false）**

**不设置：级联 查询）**

**3、自定义SQL语句**

**Straff:**

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name="com.itheima.bos.domain.Staff" table="bc\_staff" >

**<id name="id" type="java.lang.String">**

**<column name="id" length="32" />**

**<generator class="uuid" />**

**</id>**

<property name="name" type="java.lang.String">

<column name="name" length="20" />

</property>

<property name="telephone" type="java.lang.String">

<column name="telephone" length="20" />

</property>

<property name="haspda" type="java.lang.String">

<column name="haspda" length="1" />

</property>

<property name="deltag" type="java.lang.String">

<column name="deltag" length="1" />

</property>

<property name="station" type="java.lang.String">

<column name="station" length="40" />

</property>

<property name="standard" type="java.lang.String">

<column name="standard" length="100" />

</property>

**<set name="decidedzones" inverse="true">**

**<key>**

**<column name="staff\_id" length="32" />**

**</key>**

**<one-to-many class="com.itheima.bos.domain.Decidedzone" />**

**</set>**

</class>

<!-- 取派员逻辑删除 -->

**<query name="staff.delete">？？？？？**

**UPDATE Staff SET deltag = '1' WHERE id = ?**

**</query>**

</hibernate-mapping>

**User:**

<set name="roles" table="user\_role" >

<key>

<column name="user\_id" length="32" not-null="true" />

</key>

<many-to-many entity-name="com.itheima.bos.domain.Role">

<column name="role\_id" length="32" not-null="true" />

</many-to-many>

</set>

</class>

<query name="user.editpassword">

UPDATE User SET password = ? WHERE id = ?

</query>

**定区：**

<class name="com.itheima.bos.domain.Decidedzone" table="bc\_decidedzone" >

<id name="id" type="java.lang.String">

<column name="id" length="32" />

<generator class="assigned" />

</id>

<many-to-one lazy="false" name="staff" class="com.itheima.bos.domain.Staff" fetch="select">

<column name="staff\_id" length="32" />

</many-to-one>

<property name="name" type="java.lang.String">

<column name="name" length="30" />

</property>

<set name="subareas" inverse="true">

<key>

<column name="decidedzone\_id" length="32" />

</key>

<one-to-many class="com.itheima.bos.domain.Subarea" />

</set>

</class>

## 主业务（代码加分析）

### 把excel中的数据导入到数据库中

**把Excel中的数据导入到数据库中：（使用技术**OCUpload插件，POI工具**）**

method="post" enctype="multipart/form-data"

前台：

使用OCUpload插件 **1.设置一个button控件 2.给按钮添加事件（链接 文件参数名）**

后台：

获得文件流作为参数传给**HSSFWorkbook对象**

通过文档对象获得sheet对象

通过sheet对象遍历每列

是否是第一列

是

继续循环

不是

通过row获得数据封装到region中

把region集合中的数据插入到数据库中

**使用PinYin4jUtil得到区域编码**

获得省市区三个字符串

把这三个字符串连成一个字符串

使用PinYin4jUtils.getHeadByString(info);获得每个中文的首字母的数组

StringUtils.join(headByString);数组转字符串从而获得城市编码

**中文转拼音：**

PinYin4jUtils.hanziToPinyin(city, "");

前台：

//页面加载完成后，调用OCUpload插件的方法

**$("#button-import").upload({**

**action:'regionAction\_importXls.action',**

**name:'regionFile'**

**});**

**//属性驱动，接收上传的文件**

**private File regionFile;**

public void setRegionFile(File regionFile) {

this.regionFile = regionFile;

}

public String importXls() throws Exception{

List<Region> regionList = new ArrayList<Region>();

**//使用POI解析Excel文件**

**HSSFWorkbook workbook = new HSSFWorkbook(new FileInputStream(regionFile));**

/**/根据名称获得指定Sheet对象**

**HSSFSheet hssfSheet = workbook.getSheet("Sheet1");**

for (Row row : hssfSheet) {

int rowNum = row.getRowNum();

if(rowNum == 0){

continue;

}

String id = row.getCell(0).getStringCellValue();

String province = row.getCell(1).getStringCellValue();

String city = row.getCell(2).getStringCellValue();

String district = row.getCell(3).getStringCellValue();

String postcode = row.getCell(4).getStringCellValue();

//包装一个区域对象

Region region = new Region(id, province, city, district, postcode, null, null, null);

**province = province.substring(0, province.length() - 1);**

**注意：不包含索引为province.length() – 1**

city = city.substring(0, city.length() - 1);

district = district.substring(0, district.length() - 1);

**String info = province + city + district;**

String[] headByString = **PinYin4jUtils.getHeadByString(info);**

**String shortcode =** **StringUtils.join(headByString);**

//城市编码---->>shijiazhuang

**String citycode =** **PinYin4jUtils.hanziToPinyin(city, "");**

region.setShortcode(shortcode);

region.setCitycode(citycode);

regionList.add(region);

}

//批量保存

regionService.saveBatch(regionList);

return NONE;

}

### 把数据库中的数据导出为一个excel

前台

在datagrid中工具栏设置handler事件（发送一个下载的请求）

后台：

获得所有的分区数据

使用POI将数据写到Excel文件中

使用输出流进行文件下载（一个流、两个头）

创建文件名 获得输出流

解决中文乱码问题

String agent = ServletActionContext.getRequest().getHeader("User-Agent");

filename = FileUtils.encodeDownloadFilename(filename, agent);

浏览器下载：

String contentType = ServletActionContext.getServletContext().getMimeType(filename);

ServletActionContext.getResponse().setContentType(contentType);

获得解决中文乱码之后的文件名在执行下一句

ServletActionContext.getResponse().setHeader("content-disposition", "attachment;filename="+filename);

把对象通过输出流写出去

workbook.write(out);

前台：

//导出按钮对应的处理函数

**function doExport(){**

**//发送请求，请求Action，进行文件下载**

**window.location.href = "subareaAction\_exportXls.action";**

**//$.post("subareaAction\_exportXls.action");**

**}**

/\*\*

\* 分区数据导出功能

\* @throws IOException

\*/

public String exportXls() throws IOException{

//第一步：查询所有的分区数据

List<Subarea> list = subareaService.findAll();

//第二步：使用POI将数据写到Excel文件中

//在内存中创建一个Excel文件

HSSFWorkbook workbook = new HSSFWorkbook();

//创建一个标签页

HSSFSheet sheet = workbook.createSheet("分区数据");

//创建标题行

HSSFRow headRow = sheet.createRow(0);

headRow.createCell(0).setCellValue("分区编号");

headRow.createCell(1).setCellValue("开始编号");

headRow.createCell(2).setCellValue("结束编号");

headRow.createCell(3).setCellValue("位置信息");

headRow.createCell(4).setCellValue("省市区");

for (Subarea subarea : list) {

HSSFRow dataRow = sheet.createRow(sheet.getLastRowNum() + 1);

dataRow.createCell(0).setCellValue(subarea.getId());

dataRow.createCell(1).setCellValue(subarea.getStartnum());

dataRow.createCell(2).setCellValue(subarea.getEndnum());

dataRow.createCell(3).setCellValue(subarea.getPosition());

dataRow.createCell(4).setCellValue(subarea.getRegion().getName());

}

//第三步：使用输出流进行文件下载（一个流、两个头）

**String filename = "分区数据.xls";**

**String contentType = ServletActionContext.getServletContext().getMimeType(filename);**

**ServletOutputStream out = ServletActionContext.getResponse().getOutputStream();**

**ServletActionContext.getResponse().setContentType(contentType);**

//获取客户端浏览器类型

**String agent = ServletActionContext.getRequest().getHeader("User-Agent");**

**filename = FileUtils.encodeDownloadFilename(filename, agent);**

**ServletActionContext.getResponse().setHeader("content-disposition", "attachment;filename="+filename);**

**workbook.write(out);**

return NONE;

}

### 添加定区

条件查询区域：（使用easyUI-combobox）

前台：

某个input改为class="easyui-combobox"

写好后台应该传回来的字段对应的value text mode

使用easyUi查询

**前台：**

<input class="easyui-combobox" name="region.id"

data-options="**valueField:'id',textField:'name',mode:'remote',**

**mode:定义当文本改变时如何加载列表数据/**

**将被发送http请求参数名为'q'的服务器，以获取新的数据。**

url…… " />

**后台：**

**private String q; 注意这个参数一定为q不能是别的参数**

/\*\*

\* 查询所有区域，写回json数据

\* @return

\*/

public String listajax(){

List<Region> list = null;

if(**StringUtils.isNotBlank(q)**){

list = regionService.findListByQ(q);

}else{

list = regionService.findAll();

}

this.java2Json(list, new String[]{"subareas"});

return NONE;

}

public String getQ() {

return q;

}

public void setQ(String q) {

this.q = q;

}

### 五、定区关联数据

### 定区关联分区

**查询没有关联的分区**

$('#association\_customer').empty();

//创建展示客户数据的数据表格

$('#association\_customer').datagrid( {

fit : true,

border : true,

rownumbers : true,

striped : true,

url : **"decidedzoneAction\_findListHasAssociation.action?id="+data.id,**

columns : [[{

field : 'id',

title : '客户编号',

width : 120,

align : 'center'

},{

field : 'name',

title : '客户名称',

width : 120,

align : 'center'

}, {

field : 'station',

title : '所属单位',

width : 120,

align : 'center'

}]]

});

**后台：**

/\*\*

\* 查询所有未关联到定区的分区，返回json

\*/

public String listajax(){

List<Subarea> list = subareaService.findListNotAssociation();

this.java2Json(list, new String[]{"decidedzone","region"});

return NONE;

}

### 定区关联客户

获得关联客户和未关联的客户

前台 :datagrid工具栏 handler中设置事件

是否选择了定区

没有

提示

有

打开窗口

设置请求

获得客户对象的json集合

清空select上一次的option

遍历客户把客户的信息组成option拼接到select中

设置关联数据

通过标签对象.append 来把option进行转移

当点击关联的时候触发事件

获得选中定区的ID给一个隐藏域

把关联的用户选中

提交表单

**前台：**

**获得已经关联的客户和未关联的客户：**

function doAssociations(){

//获取当前数据表格所有选中的行，返回数组

var rows = $("#grid").datagrid("getSelections");

if(rows.length != 1){

//弹出提示

$.messager.alert("提示信息","请选择一个定区操作！","warning");

}else{

//选中了一个定区

$('#customerWindow').window('open');

//清理下拉框

$("#noassociationSelect").empty();

$("#associationSelect").empty();

//发送ajax请求，请求定区Action，在定区Action中通过crm代理对象完成对于crm服务远程调用获取客户数据

**var url\_1 = "decidedzoneAction\_findListNotAssociation.action";**

$.post(url\_1,function(data){

//遍历json数组

for(var i=0;i<data.length;i++){

var id = data[i].id;

var name = data[i].name;

var telephone = data[i].telephone;

**name = name + "(" + telephone + ")";**

**$("#noassociationSelect").append("<option value='"+id+"'>"+name+"</option>"**);

}

});

//发送ajax请求，请求定区Action，在定区Action中通过crm代理对象完成对于crm服务远程调用获取客户数据

**var url\_2 = "decidedzoneAction\_findListHasAssociation.action";**

var decidedzoneId = rows[0].id;

$.post(url\_2,{"id":decidedzoneId},function(data){

//遍历json数组

for(var i=0;i<data.length;i++){

var id = data[i].id;

var name = data[i].name;

var telephone = data[i].telephone;

**name = name + "(" + telephone + ")";**

**$("#associationSelect").append("<option value='"+id+"'>"+name+"</option>");**

}

});

}

}

<!-- 关联客户窗口 -->

<div modal=true class="easyui-window" title="关联客户窗口" id="customerWindow" collapsible="false" closed="true" minimizable="false" maximizable="false" style="top:20px;left:200px;width: 400px;height: 300px;">

<div style="overflow:auto;padding:5px;" border="false">

<form id="customerForm" action="**${pageContext.request.contextPath }/decidedzoneAction\_assigncustomerstodecidedzone.action**" method="post">

<table class="table-edit" width="80%" align="center">

<tr class="title">

<td colspan="3">关联客户</td>

</tr>

<tr>

<td>

**<input type="hidden" name="id" id="customerDecidedZoneId" />**

**<select id="noassociationSelect" multiple="multiple" size="10"></select> 注意：size存放最多可以有多少项**

</td>

<td>

<input type="button" value="》》" id="toRight"><br/>

<input type="button" value="《《" id="toLeft">

<script type="text/javascript">

$(function(){

//为左右移动按钮绑定事件

$("#toRight").click(function(){

**$("#associationSelect").append($("#noassociationSelect option:selected"));**

**Option的转移 不是复制**

});

$("#toLeft").click(function(){ $("#noassociationSelect").append($("#associationSelect option:selected"));

});

//为关联客户按钮绑定事件

$("#associationBtn").click(function(){

var rows = $("#grid").datagrid("getSelections");

var id = rows[0].id;

**//为隐藏域（存放定区id）赋值**

**$("input[name=id]").val(id);**

//提交表单之前，需要将右侧下拉框中所有的选项选中,为option添加一个selected属性

**$("#associationSelect option").attr("selected","selected");**

$("#customerForm").submit();

});

});

</script>

</td>

<td>

**<select id="associationSelect" name="customerIds" multiple="multiple" size="10">**

**</select>**

</td>

</tr>

<tr>

<td colspan="3"><a id="associationBtn" href="#" class="easyui-linkbutton" data-options="iconCls:'icon-save'">关联客户</a> </td>

</tr>

</table>

</form>

</div>

</div>

**配置：**

**发布服务**

1.对要发布的接口加上@WebService

2. <!-- 注册服务 -->

<jaxws:server id="myService" address="**/custome**r">

<jaxws:serviceBean>

**<ref bean="customerService"/>//这是具体的实现类**

</jaxws:serviceBean>

</jaxws:server>

**接受的时候**

1. 使用wsimport命令解析wsdl文件生成本地代码，只需要接口文件和实体类

2.接受服务：  **在applicationContext中配置了这个对象就创建了**

<!-- 注册crm客户端代理对象 -->

<jaxws:client id="crmClient"

serviceClass="com.itheima.crm.ICustomerService"

address="h**ttp://localhost:8080/crm\_heima32/service/customer**"/>

**在定区action中**

//注入crm代理对象

**@Autowired**

**private ICustomerService proxy; 获得代理对象**

/\*\*

\* 远程调用crm服务，获取未关联到定区的客户

\*/

public String findListNotAssociation(){

List<Customer> list = proxy.findListNotAssociation();

this.java2Json(list, new String[]{});

return NONE;

}

/\*\*

\* 远程调用crm服务，获取已经关联到指定的定区的客户

\*/

public String findListHasAssociation(){

String id = model.getId();

**List<Customer> list = proxy.findListHasAssociation(id);**

this.java2Json(list, new String[]{});

return NONE;

//属性驱动，接收页面提交的多个客户id

private List<Integer> customerIds;

/\*\*

\* 远程调用crm服务，将客户关联到定区

\*/

public String assigncustomerstodecidedzone(){

**proxy.assigncustomerstodecidedzone(model.getId(), customerIds);**

return LIST;

} }

**Crm中：**

//查询未关联到定区的客户

public List<Customer> findListNotAssociation() {

String sql = "select \* from t\_customer where decidedzone\_id is null";

List<Customer> list = jdbcTemplate.query(sql, new RowMapper<Customer>(){

public Customer mapRow(ResultSet rs, int arg1) throws SQLException {

int id = rs.getInt("id");//根据字段名称从结果集中获取对应的值

String name = rs.getString("name");

String station = rs.getString("station");

String telephone = rs.getString("telephone");

String address = rs.getString("address");

String decidedzone\_id = rs.getString("decidedzone\_id");

return new Customer(id, name, station, telephone, address, decidedzone\_id);

}

});

return list;

}

//查询已经关联到指定定区的客户

public List<Customer> findListHasAssociation(String decidedzoneId) {

String sql = "select \* from t\_customer where decidedzone\_id = ?";

**jdbcTemplate.query获得spring整合基本的jdbc（没有用框架整合）)**

List<Customer> list = jdbcTemplate.query(sql, new RowMapper<Customer>(){

public Customer mapRow(ResultSet rs, int arg1) throws SQLException {

int id = rs.getInt("id");//根据字段名称从结果集中获取对应的值

String name = rs.getString("name");

String station = rs.getString("station");

String telephone = rs.getString("telephone");

String address = rs.getString("address");

String decidedzone\_id = rs.getString("decidedzone\_id");

return new Customer(id, name, station, telephone, address, decidedzone\_id);

}

},decidedzoneId);

return list;

}

//定区关联客户

public void assigncustomerstodecidedzone(String decidedzoneId, Integer[] customerIds) {

String sql = "update t\_customer set decidedzone\_id = null where decidedzone\_id = ?";

jdbcTemplate.update(sql, decidedzoneId);

sql = "update t\_customer set decidedzone\_id = ? where id = ?";

for (Integer id : customerIds) {

jdbcTemplate.update(sql, decidedzoneId,id);

}

}

### 六、订单管理

**业务通知单管理**

**前台：**

$(function(){

//页面加载完成后，为手机号输入框绑定离焦事件

**$("input[name=telephone]").blur(function(){**

//获取页面输入的手机号

var telephone = this.value;

//发送ajax请求，请求Action，在Action中远程掉调用crm服务，获取客户信息，用于页面回显

$.post('**noticebillAction\_findCustomerByTelephone.action**',{"telephone":telephone},

function(data){

if(data != null){

//查询到了客户信息，可以进行页面回显

var customerId = data.id;

var customerName = data.name;

var address = data.address;

$("input[name=customerId]").val(customerId);

$("input[name=customerName]").val(customerName);

$("input[name=delegater]").val(customerName);

$("input[name=pickaddress]").val(address);

}else{

//没有查询到客户信息，不能进行页面回显

$("input[name=customerId]").val("");

$("input[name=customerName]").val("");

$("input[name=delegater]").val("");

$("input[name=pickaddress]").val("");

}

});

});

});

提交：

$(function(){

$("body").css({visibility:"visible"});

// 对save按钮条件 点击事件

$('#save').click(function(){

// 对form 进行校验

**if($('#noticebillForm').form('validate')){**

**$('#noticebillForm').submit();**

**}**

});

});

import com.itheima.crm.Customer;

/\*\*

\* 业务通知单管理

\* @author zhaoqx

\*

\*/

import com.itheima.crm.ICustomerService;

@Controller

@Scope("prototype")

public class NoticebillAction extends BaseAction<Noticebill>{

//注入crm客户端代理对象

**@Autowired**

**private ICustomerService customerService;**

/\*\*

\* 远程调用crm服务，根据手机号查询客户信息

\*/

public String findCustomerByTelephone(){

String telephone = model.getTelephone();

**Customer customer = customerService.findCustomerByTelephone(telephone);**

this.java2Json(customer, new String[]{});

return NONE;

}

@Autowired

private INoticebillService noticebillService;

/\*\*

\* 保存一个业务通知单，并尝试自动分单

\*/

public String add(){

**noticebillService.save(model);**

return "noticebill\_add";

}

}

**工单快速记录（**把execl数据导入到数据库**）**

**编辑datgrid中的数据并提交：**

$(function(){

// 先将body隐藏，再显示，不会出现页面刷新效果

$("body").css({visibility:"visible"});

// 收派标准数据表格

$('#grid').datagrid( {

iconCls : 'icon-forward',

fit : true,

border : true,

rownumbers : true,

striped : true,

pageList: [30,50,100],

pagination : true,

toolbar : toolbar,

url : "",

idField : 'id',

columns : columns,

onDblClickRow : doDblClickRow,

onAfterEdit : function(rowIndex, rowData, changes){

console.info(rowData);

editIndex = undefined;

$.post('workordermanageAction\_add.action',rowData,function(data){

if(data == '0'){

$.messager.alert("提示信息","工作单信息录入失败！","error");

}

});

}

});

});

function doDblClickRow(rowIndex, rowData){

alert("双击表格数据...");

console.info(rowIndex);

$('#grid').datagrid('beginEdit',rowIndex);

editIndex = rowIndex;

}

@Controller

@Scope("prototype")

public class WorkordermanageAction extends BaseAction<Workordermanage> {

@Autowired

private IWorkordermanageService workordermangeService;

public String add() throws IOException{

String f = "1";

try{

workordermangeService.save(model);

}catch(Exception e){

e.printStackTrace();

f = "0";

}

ServletActionContext.getResponse().setContentType("text/html;charset=utf-8");

ServletActionContext.getResponse().getWriter().print(f);

return NONE;

}

}

### 七．权限管理

### 八、多条件查询

**多条件查询分区**

public String pageQuery(){

DetachedCriteria dc = pageBean.getDetachedCriteria();

//动态添加过滤条件

String addresskey = model.getAddresskey();

if(StringUtils.isNotBlank(addresskey)){

//添加过滤条件，根据地址关键字模糊查询

**dc.add(Restrictions.like("addresskey", "%"+addresskey+"%"));**

}

Region region = model.getRegion();

if(region != null){

String province = region.getProvince();

String city = region.getCity();

String district = region.getDistrict();

dc.createAlias("region", "r");

if(StringUtils.isNotBlank(province)){

//添加过滤条件，根据省份模糊查询-----多表关联查询，使用别名方式实现

//参数一：分区对象中关联的区域对象属性名称

//参数二：别名，可以任意

**dc.add(Restrictions.like("r.province", "%"+province+"%"));**

}

if(StringUtils.isNotBlank(city)){

//添加过滤条件，根据市模糊查询-----多表关联查询，使用别名方式实现

//参数一：分区对象中关联的区域对象属性名称

//参数二：别名，可以任意

**dc.add(Restrictions.like("r.city", "%"+city+"%"));**

}

if(StringUtils.isNotBlank(district)){

//添加过滤条件，根据区模糊查询-----多表关联查询，使用别名方式实现

//参数一：分区对象中关联的区域对象属性名称

//参数二：别名，可以任意

**dc.add(Restrictions.like("r.district", "%"+district+"%"));**

}

}

subareaService.pageQuery(pageBean);

this.java2Json(pageBean, new String[]{"currentPage","detachedCriteria","pageSize",

"decidedzone","subareas"});

return NONE;

}

### 九、查询定区关联的分区：

**定区关联的分区：**

//属性驱动，接收定区id

private String decidedzoneId;

/\*\*

\* 根据定区id查询关联的分区

\*/

public String **findListByDecidedzoneId()**{

List<Subarea> list = subareaService.findListByDecidedzoneId(decidedzoneId);

this.java2Json(list, new String[]{"decidedzone","subareas"});

return NONE;

}

public String getDecidedzoneId() {

return decidedzoneId;

}

public void setDecidedzoneId(String decidedzoneId)

this.decidedzoneId = decidedzoneId;

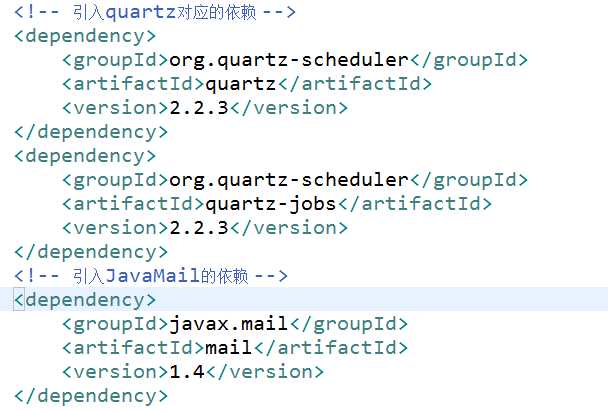
}

### 十、定时任务：

**功能：每隔一段时间执行某个类中的某个方法**

### 举例：在BOS项目中引入quartz

第一步：在pom.xml中引入quartz和JavaMail的依赖



第二步：提供一个作业类，用于为系统管理员发送邮件

**public** **class** MailJob {

@Resource

**private** IWorkbillDao workbillDao;

**private** String username;//发件人的邮箱账号

**private** String password;//密码

**private** String smtpServer;//服务器

**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** **void** execute() {

System.***out***.println("要发邮件了。。。" + **new** Date());

**try** {

//查询工单类型为新单的所有工单

List<Workbill> list = workbillDao.findAll();

**if**(**null** != list && list.size() > 0){

**final** Properties mailProps = **new** Properties();

mailProps.put("mail.smtp.host", **this**.getSmtpServer());

mailProps.put("mail.smtp.auth", "true");

mailProps.put("mail.username", **this**.getUsername());

mailProps.put("mail.password", **this**.getPassword());

// 构建授权信息，用于进行SMTP进行身份验证

Authenticator authenticator = **new** Authenticator() {

**protected** PasswordAuthentication getPasswordAuthentication() {

// 用户名、密码

String userName = mailProps.getProperty("mail.username");

String password = mailProps.getProperty("mail.password");

**return** **new** PasswordAuthentication(userName, password);

}

};

// 使用环境属性和授权信息，创建邮件会话

Session mailSession = Session.*getInstance*(mailProps, authenticator);

**for**(Workbill workbill : list){

// 创建邮件消息

MimeMessage message = **new** MimeMessage(mailSession);

// 设置发件人

InternetAddress from = **new** InternetAddress(mailProps.getProperty("mail.username"));

message.setFrom(from);

// 设置收件人

InternetAddress to = **new** InternetAddress("test@itcast.cn");

message.setRecipient(RecipientType.***TO***, to);

// 设置邮件标题

message.setSubject("系统邮件：新单通知");

// 设置邮件的内容体

message.setContent(workbill.toString(), "text/html;charset=UTF-8");

// 发送邮件

Transport.*send*(message);

}

}

} **catch** (Exception ex) {

ex.printStackTrace();

}

}

**public** String getSmtpServer() {

**return** smtpServer;

}

**public** **void** setSmtpServer(String smtpServer) {

**this**.smtpServer = smtpServer;

}

}

第三步：在spring配置文件中配置

<!-- 注册自定义作业类 -->

<bean id=*"myJob"* class=*"com.itheima.jobs.MailJob"*>

<property name=*"username"* value=*"itcast\_server@126.com"*/>

<property name=*"password"* value=*"147963qP"*/>

<property name=*"smtpServer"* value=*"smtp.126.com"*/>

</bean>

<!-- 配置JobDetail -->

<bean id=*"jobDetail"* class=*"org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean"*>

<!-- 注入目标对象 -->

<property name=*"targetObject"* ref=*"myJob"*/>

<!-- 注入目标方法 -->

<property name=*"targetMethod"* value=*"execute"*/>

</bean>

<!-- 配置触发器 -->

<bean id=*"myTrigger"* class=*"org.springframework.scheduling.quartz.CronTriggerFactoryBean"*>

<!-- 注入任务详情对象 -->

<property name=*"jobDetail"* ref=*"jobDetail"*/>

<!-- 注入cron表达式，通过这个表达式指定触发的时间点 -->

<property name=*"cronExpression"*>

<value>0/5 \* \* \* \* ?</value>

</property>

</bean>

<!-- 配置调度工厂 -->

<bean id=*"schedulerFactoryBean"* class=*"org.springframework.scheduling.quartz.SchedulerFactoryBean"*>

<!-- 注入触发器 -->

<property name=*"triggers"*>

<list>

<ref bean=*"myTrigger"*/>

</list>

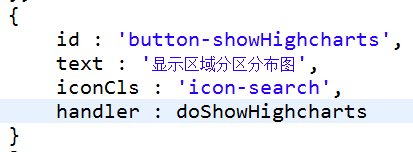
</property>

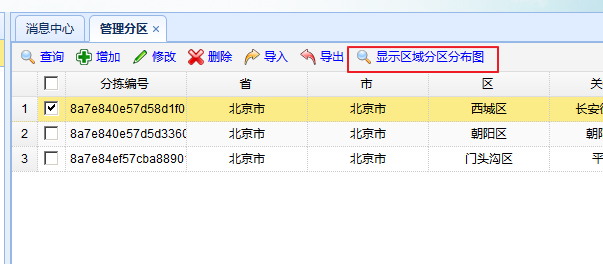
</bean>

### 在BOS项目中展示区域分区分布图

第一步：在subarea.jsp页面中引入Highcharts资源文件

第二步：在jsp页面中提供按钮，并提供div窗口，在这个窗口中展示图表





第三步：定义function

**function** doShowHighcharts(){

$("#showSubareaWindow").window("open");

//页面加载完成后，动态创建图表

$.post("subareaAction\_findSubareasGroupByProvince.action",**function**(data){

$("#test").highcharts({

title: {

text: '区域分区分布图'

},

series: [{

type: 'pie',

name: '区域分区分布图',

data: data

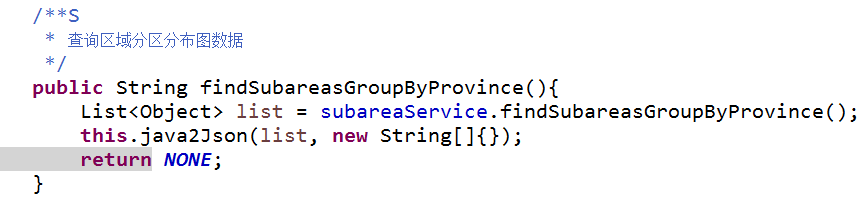
}]

});

});

}

第四步：在服务端Action中提供方法



Dao代码：

@Repository

**public** **class** SubareaDaoImpl **extends** BaseDaoImpl<Subarea> **implements** ISubareaDao {

**public** List<Object> findSubareasGroupByProvince() {

String hql = "SELECT r.province ,count(\*) FROM Subarea s LEFT OUTER JOIN s.region r Group BY r.province";

**return** (List<Object>) **this**.getHibernateTemplate().find(hql);

}

}