OA+Ssh整合

版本信息

|  |  |
| --- | --- |
| Struts2 | 2.37 |
| Hibernate | 3.5.5-final |
| Spring | 3.1.0 |
| Juit | 4 |
| Mysql | 5 |
| Myeclipse | 10.1 |

配置文件描述（注解版ssh）

##### 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: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.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop.xsd*  *"*>  <!-- 自动扫描与装配bean -->  <context:component-scan base-package=*"cn.itcast.oa"*></context:component-scan>  <!-- 配置SessionFactory（整合Hibernate） -->  <context:property-placeholder location=*"classpath:jdbc.properties"*/>  <!--配置数据库连接池 -->  <bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"*>  <!-- 连接信息 -->  <property name=*"url"* value=*"${url}"*></property>  <property name=*"driverClassName"* value=*"${driverClassName}"*></property>  <property name=*"username"* value=*"${username}"*></property>  <property name=*"password"* value=*"${password}"*></property>  <!-- 其他信息 -->  </bean>    <bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate3.LocalSessionFactoryBean"*>  <!-- 指定Hibernate的配置文件 -->  <property name=*"configLocation"* value=*"classpath:hibernate.cfg.xml"*></property>  <!-- 配置数据源 -->  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean>  <!-- 配置声明式事务管理（基于注解的方式） -->  <bean id=*"transactionManager"* class=*"org.springframework.orm.hibernate3.HibernateTransactionManager"*>  <property name=*"sessionFactory"* ref=*"sessionFactory"*></property>  </bean>  <tx:annotation-driven transaction-manager=*"transactionManager"*/>  </beans> |

##### Jdbc.properties

|  |
| --- |
| driverClassName = com.mysql.jdbc.Driver  url = jdbc:mysql://localhost:3306/test  username = root  password =root |

##### Hibernate.cfg.xml

|  |
| --- |
| <!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>  <!-- 数据库连接信息 -->  <property name=*"hibernate.dialect"*>  org.hibernate.dialect.MySQL5Dialect  </property>  <!-- 其他配置 -->  <property name=*"show\_sql"*>true</property>  <property name=*"hbm2ddl.auto"*>update</property>  <!-- 声明映射文件 -->  <mapping resource=*"com/oa/pojo/User.hbm.xml"* />  </session-factory>  </hibernate-configuration> |

##### User.hbm.xml

|  |
| --- |
| <?xml version=*"1.0"*?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name=*"com.oa.pojo.User"* table=*"t\_users"*>  <id name=*"id"* column=*"id"* type=*"integer"*>  <generator class=*"native"*></generator>  </id>  <property name=*"name"* column=*"name"* type=*"string"* length=*"24"*></property>  </class>  </hibernate-mapping> |

Log4j.properties

|  |
| --- |
| log4j.rootLogger=debug,appender1  log4j.appender.appender1=org.apache.log4j.FileAppender  log4j.appender.appender1.layout=org.apache.log4j.HTMLLayout  log4j.appender.appender1.File=D:/MyEclipse/Workspaces/OA/WebRoot/demolog4j.htm |

## S2s3h3整合

### Hibernate 与 spring 整合

版本信息：

Hibernate 3.5.5-Final spring 3.1.0.RC1

#### 测试：

测试1：数据连接池的使用

测试2：事务的使用

##### Juit

|  |
| --- |
| **package** com.oa.test;  **import** org.apache.commons.logging.Log;  **import** org.apache.commons.logging.LogFactory;  **import** org.hibernate.SessionFactory;  **import** org.junit.Test;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **public** **class** TestApplictionContext {  **private** ApplicationContext ac = **new** ClassPathXmlApplicationContext("applicationContext.xml");  @Test  **public** **void** testSessionFactory(){  SessionFactory sf = (SessionFactory) ac.getBean("sessionFactory");  System.*out*.println("---->"+sf);  }  @Test  **public** **void** testTran(){  Log log = LogFactory.*getLog*(getClass());  log.info("------save-------");  TestService test = (TestService) ac.getBean("testService");  test.save();  }  } |

测试事务时，使用User对象写入Id和name，get set下，在用service层

##### TestService

|  |
| --- |
| **@Service("testService")**  **public** **class** TestService {    @Resource//设置SessionFactory数据池拿出session  **private** SessionFactory sessionFactory;    @Transactional//设置事务，说明只有这个方法可以使用事务  **public** **void** save(){  Session session = sessionFactory.getCurrentSession();  session.save(**new** User());  session.save(**new** User());  session.flush();  }  } |

注意：@Component是所有受Spring管理组件的通用形式，Spring还提供了更加细化的注解形式：@Repository、@Service、@Controller，它们分别对应存储层Bean，业务层Bean，和展示层Bean。目前版本（2.5）中，这些注解与@Component的语义是一样的，完全通用，在Spring以后的版本中可能会给它们追加更多的语义。所以，我们推荐使用@Repository、@Service、@Controller来替代@Component。

### Spring与struts2整合

##### 第一步：配置Struts2（TestAction.java）

|  |
| --- |
| **public** **class** TestAction **extends** ActionSupport{  **public** String execute() **throws** Exception {  **return** "success";  }  } |

##### 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>  <!-- 开发模式 -->  <constant name=*"struts.devMode"* value=*"true"*></constant>  <!-- 配置扩展名为.action -->  <constant name=*"struts.action.extension"* value=*"action"*></constant>    <package name=*"com"* namespace=*"/"* extends=*"struts-default"*>  <!-- 测试用的Action，当与Spring整合后，class属性就是配置的bean的名称了 -->  <action name=*"testAction"* class=*"com.oa.test.TestAction"*>  <result name=*"success"*>/index.jsp</result>  </action>  </package>  </struts> |

现在可以测试下，通过<http://localhost:9876/OA/testAction.action>可以访问到index.jsp，由于这样只用于struts2的框架，而没有由于到sprng控制反转的资源，所以由下，可以将Action交付给spring管理，从而ssh整合，而不是各玩各的

##### 第二步，将struts2由spring管理

|  |
| --- |
| @Controller("testAction")//将action交个spring管理  @Scope("prototype")//使用spring管理产生对象是默认为单例，用prototype可以设置为多例  **public** **class** TestAction **extends** ActionSupport{  @Resource//在controller层肯定要调用service层的对象，所以利用spring的依赖注入  **private** TestService testService;    **public** String execute() **throws** Exception {  **return** "success";  }  } |

**注意：那么现在只需要将struts2中action的class=”com.oa.test.TestAction”改为class=”testAction”其中struts2交个spring管理**

**怎么在容器中取到testAction，由于已启动就加载struts对象，所以取到spring管理的对象testAction需要再web.xml中配置spring的监听器，用于创建applicationContext的对象，从而取得applicationContext中testAction**

##### 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的监听器，用于创建ApplicationContext对象 -->  **<listener> <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>**  **</listener>**  **<context-param>**  **<param-name>contextConfigLocation</param-name>**  **<param-value>classpath:applicationContext\*.xml</param-value>**  **</context-param>**  <!-- 配置Struts2的过滤器 -->  <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>  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>  </web-app> |

**现在可以测试下，**<http://localhost:9876/OA/testAction.action，通过在struts.xml>中的修改为class=”testAction”说名spring成功注入，spring监听器取testAction的成功。

**注意：**

**使用时：**

**1，与Spring整合后，配置Action时class属性就是配置的bean的名称了。**

**2，Action上要加注解：**

**@Controller**

**@Scope("prototype")**

## 设计BaseDao接口与BaseDaoImpl类

每个实体都应有一个对应的Dao，他封装了对这个实体的数据库操作。例

实体 Dao接口 实现类

========================================================

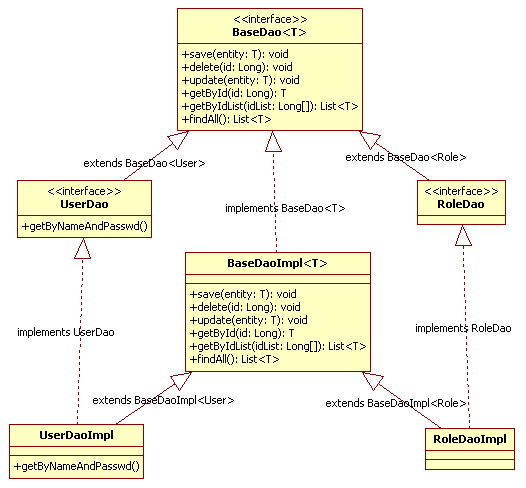
User --> UserDao --> UserDaoImpl

Role --> RoleDao --> RoleDaoImpl

Department --> DepartmentDao --> DepartmentDaoImpl

Article --> ArticleDao --> ArticleDaoImpl

...



说明：

* 1. 实体的Dao接口要继承BaseDao接口。
  2. Dao的实现类要继承DaoImplBase类。
  3. 也可以不继承指定的接口或类，这样就要自己写相应的方法。
  4. T getById(Long id)与List<T> getByIdList(Long[] idList)不要合并为List getById(Long... ids)，因为获取一个对象时也是返回List，不方便。

BaseDao.java

|  |
| --- |
| **package** com.oa.base;  **import** java.util.List;  **public** **interface** BaseDao<T> {  /\*  \* 保存元素  \* \*/  **void** save(T entry);    /\*  \* 更新元素  \* \*/  **void** update(T entry);  /\*  \* 通过id删除元素  \* \*/  **void** dalete(Long id);  /\*  \* 查找全部元素  \* \*/  List<T> getAll();  /\*  \* 通过多个id查找元素  \* \*/  List<T> getByIds(Long[] ids);  /\*  \* 通过id查找单个元素  \* \*/  T getById(Long id);} |

BaseDaoImpl.java

|  |
| --- |
| **package** com.oa.base;  **import** java.lang.reflect.ParameterizedType;  **import** java.util.List;  **import** javax.annotation.Resource;  **import** org.hibernate.Session;  **import** org.hibernate.SessionFactory;  **import** org.springframework.stereotype.Controller;  @SuppressWarnings("unchecked")  **public** **class** BaseDaoImpl<T> **implements** BaseDao<T> {    @Resource  **private** SessionFactory sessionFactory;  **private** Class<T> clazz;    **protected** Session getSession(){  **return** sessionFactory.getCurrentSession();  }  /\*  \* 当用private session时  \* 由于\*DaoImpl实现类都继承了BaseDaoImpl。而当\*DaoImpl都需要写自己独立的类时，又需要  \* 创建Session才可以使用，现在用protected可以使子类使用session，不需要在重新创建session  \* \*/  **public** BaseDaoImpl(){  ParameterizedType pt = (ParameterizedType) **this**.getClass().getGenericSuperclass();  **this**.clazz = (Class<T>) pt.getActualTypeArguments()[0];  }  **public** **void** save(T entry) {  // **TODO** Auto-generated method stub  getSession().save(entry);  }  **public** **void** update(T entry) {  // **TODO** Auto-generated method stub  getSession().update(entry);  }  **public** **void** dalete(Long id) {  // **TODO** Auto-generated method stub  Object o = getSession().get(clazz, id);  getSession().delete(o);  }  **public** List<T> getAll() {  // **TODO** Auto-generated method stub  **return** getSession().createQuery(  "from "+clazz.getSimpleName())  .list();  }  **public** List<T> getByIds(Long[] ids) {  // **TODO** Auto-generated method stub  **return** getSession().createQuery("" +  "from "+clazz.getSimpleName()+" where id in (:ids)")  .setParameterList("ids", ids)  .list();  }  **public** T getById(Long id) {  // **TODO** Auto-generated method stub  **return** (T) getSession().get(clazz, id);  }  } |

**获取 BaseDao的类型参数T的Class**

问题：

1. 有了DaoBase与DaoImplBase，还要用UserDao、RoleDao吗？

答：要用。因为UserDao或RoleDao中的方法可以分为有公有的方法与特有的方法两部分。公有的方法是通过继承BaseDao得到的，特有的方法要写在自己里面（BaseDao中是没有的）。

1. UserDaoImpl已经继承了BaseDaoImpl，就不实现UserDao可以吗？

答：不可以。否则UserDao userDao = new UserDaoImpl(); 就不成立。

使用反射获取类型参数的真实类型的代码如下：

|  |
| --- |
| **public** DaoBaseImpl () {  Type type = **this**.getClass().getGenericSuperclass();  ParameterizedType pt = (ParameterizedType) type;  **this**.clazz = (Class<T>) pt.getActualTypeArguments()[0];  } |

说明：

1. 使用Session时，不要自己创建，也不要管理事务，直接调用getSession()即可。
2. 暂时不实现getSession()方法，在后面的事务管理中实现：  
   **protected** Session getSession(){  
    **throw** **new** UnsupportedOperationException();  
   }

### 步骤

1，充分了解需求（包括所有的细节）：分析页面等。

2，设计实体/表

正向工程：设计实体 --> 建表（推荐）：设计实体 --> JavaBean --> hbm.xml --> 建表

反向工程：设计表 --> 实体

3，分析功能到每个请求的粒度。

得到的结果是我们需要处理多少种请求，每种请求对应一个Action方法。

4，实现功能：

1，创建Action，并定义出其中的方法。

2，实现Action方法，并创建出所用到的新的Service方法。

3，实现Service方法，并创建出所用到的新的Dao方法。

4，实现Dao方法。

5，创建并完成JSP页面。

5，单元测试与运行测试

## 系统管理实现

增删改查 + 树状结构 + 表单验证。

### 设计实体（重要）

1，有几个实体？

一般是一组增删改查对应一个实体。

2，实体之间有什么关系？

一般是页面引用了其他的实体时，就表示与这个实体有关联关系。

3，每个实体中都有什么属性？

**1，主键**

**2，关联关系属性**

**在类图中，关联关系是一条线，有两端，每一端对应一个表达此关联关系的属性。**

**有几个端指定本类，本类中就有几个关联关系属性。**

**3，一般属性：**

**分析所有有关的页面，找出表单中要填写的或是在显示页面中要显示的信息等。**

**4，特殊属性：为解决某问题而设计的属性。**

**比如要显示年龄，但不能设计一个int age字段，而是一个Date birthday字段，年龄是显示时实时计算出来的。**

### 映射实体(重要)

1，写注释：

格式为：？属性，表达的是本对象与？的？关系。

例：<!-- department属性，本对象与Department的多对一 -->

1. 拷模板：

|  |
| --- |
| **多对一：**  <many-to-one name="" class="" column=""></many-to-one>  **一对多（Set）：**  <set name="">  <key column=""></key>  <one-to-many class=""/>  </set>  **多对多（Set）：**  <set name="" table="">  <key column=""></key>  <many-to-many class="" column=""></many-to-many>  </set>  3，填空：  **name属性：属性名（注释中的第1问号）**  **class属性：关联的实体类型（注释中的第2个问号）**  **column属性：**  <many-to-one column="..">：一般可以写成属性名加Id后缀，如属性为department，则column值写成departmentId。  一对多中的<key column="..">：从关联的对方（对方是多对一）映射中把column值拷贝过来。  多对多中的<key column="..">：一般可以写成本对象的名加Id后缀，强本对象名为User，则写为userId。  **多对多中的<many-to-many column="..">：一般可以写为关联对象的名称加Id后缀。** |

实体就不列举了

例举：Role.hbm.xml

|  |
| --- |
| <?xml version=*"1.0"*?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name=*"com.oa.pojo.Role"* table=*"t\_role"*>  <id name=*"id"* column=*"id"* type=*"long"*>  <generator class=*"native"*></generator>  </id>  <property name=*"name"* column=*"name"* type=*"string"* length=*"30"*></property>  <property name=*"* *description"* column=*"* *description"* type=*"string"* length=*"300"*></property>    <!-- Role对象与User对象多对多 -->  **<set name=*"users"* table=*"t\_u\_r"*>**  **<key column=*"roleId"*></key>**  **<many-to-many class=*"com.oa.pojo.User"* column=*"userId"*></many-to-many>**  **</set>**  </class>  </hibernate-mapping> |

**User.hbm.xml**

|  |
| --- |
| <?xml version=*"1.0"*?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name=*"com.oa.pojo.User"* table=*"t\_user"*>  <id name=*"id"* column=*"id"* type=*"long"*>  <generator class=*"native"*></generator>  </id>  <property name=*"name"* column=*"name"* type=*"string"* length=*"30"*></property>  <property name=*"loginName"* column=*"loginName"* type=*"string"* length=*"30"*></property>  <property name=*"password"* column=*"password"* type=*"string"* length=*"30"*></property>  <property name=*"sex"* column=*"sex"* type=*"string"* length=*"10"*></property>  <property name=*"phoneNumber"* column=*"phoneNumber"* type=*"string"* length=*"50"*></property>  <property name=*"email"* column=*"email"* type=*"string"* length=*"50"*></property>  <property name=*"* *description"* column=*"* *description"* type=*"string"* length=*"300"*></property>    <!-- User属性，本对象与department的多对一 -->  <many-to-one name=*"department"* class=*"com.oa.pojo.Department"* column=*"departmentId"*></many-to-one>  <!-- User属性，本对象与Role对象多对多 -->  <set name=*"roles"* table=*"t\_u\_r"*>  <key column=*"userId"*></key>  <many-to-many class=*"com.oa.pojo.Role"* column=*"roleId"*></many-to-many>  </set>  </class>  </hibernate-mapping> |

**Department.hbm.xml**

|  |
| --- |
| <?xml version=*"1.0"*?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping>    <class name=*"com.oa.pojo.Department"* table=*"t\_dept"*>  <id name=*"id"* column=*"id"* type=*"long"*>  <generator class=*"native"*></generator>  </id>  <property name=*"name"* column=*"name"* type=*"string"* length=*"30"*></property>  <property name=*"* *description"* column=*"* *description"* type=*"string"* length=*"300"*></property>  <!--department对象，本对象与User关系一对多 -->  <set name=*"user"*>  <key column=*"departmentId"*></key>  <one-to-many class=*"com.oa.pojo.User"*/>  </set>  <!-- department对象，本对象与parent 多对一 -->  <many-to-one name=*"parent"* class=*"com.oa.pojo.Department"* column=*"parentId"*></many-to-one>  <!-- department对象，本对象与children 一对多 -->  <set name=*"children"*>  <key column=*"parentId"*></key>  <one-to-many class=*"com.oa.pojo.Department"*/>  </set>  </class>  </hibernate-mapping> |

出现问题：我把关键字desc，写入进去了，所以一直创建不了，最后description，就成功了

### 分析增删改查功能

**增删改查共4个功能，6个请求，所以需要在Action中有6个对应的处理方法。**

**作用 方法名 返回值 页面**

**------------------------------------------**

**列表 list() list list.jsp**

**删除 delete() toList actionName="roleAction\_list" type="redirectAction"**

**添加页面 addUI() addUI addUI.jsp**

**添加 add() toList**

**修改页面 editUI() editUI editUI.jsp**

**修改 edit() toList**

**转发 重定向**

**--------------------------------**

**请求数 1 2**

**浏览器地址栏 不变 变**

### 关于spring的注解版在MVC中使用的注意事宜：

对于@Transactional，事务管理，需要在调用其中dao方法是正常使用session时，一般在service中写入事务管理

对于@Service ，@Repository，@Controller中需要使spring管理并可以通过@Resource拿到的类

对于spring默认是单例的，所以在使用action的时候需要注入@Scope("prototype")变成每次获取时都是不同的对象

@Service ，@Repository，@Controller在实现类中使用，而不是在接口中使用，不然就会报错：No matching bean of type [com.oa.pojo.service.RoleService] found for dependency: expected at least 1 bean which qualifies as autowire candidate for this dependency.

Struts.xml部分编码

|  |
| --- |
| <action name=*"roleAction\_\*"* class=*"roleAction"* method=*"{1}"*>  <result name=*"list"*>/WEB-INF/jsp/role/list.jsp</result>  <result name=*"addUI"*>/WEB-INF/jsp/role/add.jsp</result>  <result name=*"editUI"*>/WEB-INF/jsp/role/edit.jsp</result>  <result name=*"tolist"* type=*"redirectAction"*>roleAction\_list</result>  </action> |

其中action中name为地址栏，当显示全部，地址为<http://localhost:9876/OA/roleAction_list.action>，其中\_\*就是方法名，通过method=”{1}”而拿到方法名，从而跳转到Action类中方法去，从而通过返回值list而在xml中result跳转到相应的界面

### Struts2 的流程：

一开始web.xml监听到spring，创建对象，而当浏览器请求时，struts监听到，到struts.xml去找到相应的action，跳转到Action类中去，得到返回值后，再到struts.xml中去在action中找到相应返回值后在result中跳转到页面中，结束

|  |
| --- |
| 1. 客户端浏览器发出Http请求。   🡪2.根据web.xml配置，该请求被FilterDispatch接收  🡪3.根据struts.xml配置，找到需要调用的Action类和方法，并通过IOC方式，将值注入Action  🡪4.Action调用业务逻辑组件处理业务逻辑，这一不包含表单验证  🡪5.Action执行后，根据struts.xml中配置找到对应返回结果result，并跳转到相应界面  🡪6.返回Http相应到客户端浏览器 |

### 使用strurt2标签与传值，：

第一：<%@ taglib prefix=*"s"* uri=*"/struts-tags"* %>

在Action中使用

ActionContext.*getContext*().put("roleList", roleList);

将对象**传入**保存到mapping中，

页面通过标签**获取值**

|  |
| --- |
| <s:iterator value=*"#roleList"*>  <s:property value=*"id"*/>,  ${name }  <s:property value=*"description"*/>  <s:a action=*"roleAction\_delete?id=%{id}"* onClick="return window.confirm('您确定11要将该用户')">删除</s:a><br>  <s:a action=*"roleAction\_edit?id=%{id}"*>修改</s:a><br>  </s:iterator> |

Set或list都可以通过iterator迭代出来，而用#roleList只是同#对象拿到mapping中的值，而其中也可以用EL表达式，因为struts重写了request. getAttribute方法等，所以可以用${name}

故可以更简单写成

|  |
| --- |
| <s:iterator value=*"#roleList"*>  ${id }  ${name }  ${description }  <s:iterator value=*"#roleList"*> |

### 修改时回显 ：

在以前的时候，我是通过链接先跳转到通过id找到对象再用request传入forword页面中，这时候，就复杂并不正规，两次请求

1. 而用struts2继承了ActinSupport就就不需要这么麻烦，直接在方法中写入

|  |
| --- |
| /\*\* 修改页面 \*/  public String editUI() throws Exception {  Role role = roleService.getById(id);  name = role.getName();  description = role.getDescription();  return "editUI";  } |

**因为有栈，所以直接传入name，description就可以了，在表单中就可以直接使用了**

1. **实现了ModelDriver就只需要使用**

用**ActionContext.getContext().getValueStack().push(role);在实现了ModelDriver回显意义**

|  |
| --- |
| <s:form action=*"roleAction\_edit.action"* method=*"post"*>  <s:hidden name=*"id"*></s:hidden>  <s:textfield name=*"name"* label=*"岗位名称"*></s:textfield><br>  <s:textarea name=*"description"* label=*"岗位说明"* ></s:textarea><br>  <s:submit value=*"确定"*></s:submit><s:reset value=*"取消"*></s:reset> </s:form> |

由于Action直继承了ActionSupport，不提供值专递与封装，所以就需要写入实体类，私有属性与相对应的get与set方法，由于我们已经有实体类了，按照java特性就需要封装概念，于是就出来

Action继承ActionSupport并实现ModelDriver<T>

修改Action代码

|  |
| --- |
| @Controller("roleAction")  @Scope("prototype")  public class **RoleAction extends ActionSupport implements ModelDriven<Role>**{    **@Resource**  **private RoleService roleService;**    **Role model = new Role();**    **public Role getModel(){**  **return model;**  **}**  //其中实现了ModelDriver，每一个实体独有一个对应的Action，所以可以只需要写成get方法就可以了  public String list() throws Exception{  List<Role> roleList = roleService.getAll();  ActionContext.getContext().put("roleList", roleList);  return "list";  }    public String delete() throws Exception{  roleService.dalete(model.getId());  return "tolist";  }    public String editUI() throws Exception{  Role role = roleService.getById(model.getId());  **ActionContext.getContext().getValueStack().push(role);**  return "editUI";  }  public String edit() throws Exception{  // Role role = new Role();  // role.setId(model.getId());  // role.setName(model.getName());  // role.setDescription(model.getDescription());  roleService.update(model);  return "tolist";  }    public String addUI() throws Exception{  return "addUI";  }  public String add() throws Exception{  // Role role = new Role();  // role.setName(model.getName());  // role.setDescription(model.getDescription());  roleService.save(model);  return "tolist";  } } |

用**ActionContext.getContext().getValueStack().push(role);在实现了ModelDriver回显意义**

### 整合页面：



其中数据都来自于js中效果，为了使页面更加简便

<script language="javascript" src="../script/DemoData.js" charset="utf-8"></script>

<script language="javascript" src="../script/DataShowManager.js" charset="utf-8"></script> <!--显示数据列表-->

<tbody id="TableData" class="dataContainer" datakey="roleList">

**<tr class="TableDetail1 template">**

<td>${role.name}&nbsp;</td>

<td>${role.description}&nbsp;</td>

<td><a onClick="return delConfirm()" href="list.html">删除</a>

<a href="saveUI.html">修改</a>

<a href="setPrivilegeUI.html">设置权限</a>

</td>

</tr>

</tbody>

1. 将style，script和html都导入相应的位置
2. 将js，css都写入绝对位置，ctrl+f，中查找到../替换成${pageContext.request.contextPath }*/*
3. 将strut2写入**<constant name="struts.ui.theme" value="simple" />** 会使label标签失效

### jQuery的validate插件简单使用

### 部门的实现

要求一：删除部门时，同时删除此部门的所有下级部门

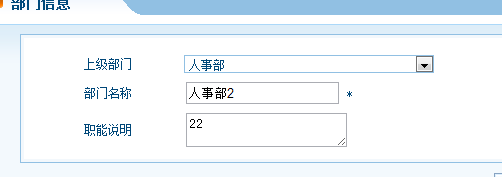
而同时我们创建部门表的自身关联所以在hibernate中department.hbm.xml写入

<set name=*"children"* cascade=*"delete"*>级联操作

经过试验后，可以删除

部门在保存数据同时的时候，选择

部门回显：



回显，难点在于，不仅在上级部门中显示本来的上级部门和能选择的树形（上级部门），并且由于上级部门的回显数据库中没有父类对象存在，而只有id存在，所以必须有parentId显示

当部门对象没有上级部门，同parentId传入值为null，所以通过getById()找到数据，从而得到数据，但是为null，则说明需要判断为null，如果在DepartmentAction中得到每个调用getById（）的，则太过于麻烦，所以在baseDaoImpl写入判断

Select显示

|  |
| --- |
| <s:hidden name=*"id"*></s:hidden>  <s:select name=*"parentId"* list=*"#departmentList"* cssClass=*"SelectStyle"*  listKey=*"id"* listValue=*"name"* headerKey=*""* headerValue=*"请选择部门"*>  </s:select>  由于修改的对象，没有id，所以id需要隐式传入  Name是属性名，list是树形结构需要对象容器，cssClass是样式，listKey是option的value值 也就是传入的值给struts，listValue是option的name显示的值，headerKey开头的id，headerValue树形结构开头的name |

|  |
| --- |
| **public** String editUI() **throws** Exception{  // 准备数据，应显示树状结构的所有部门列表，先使用所有部门列表代替  List<Department> departmentList = departmentService.getAll();  ActionContext.*getContext*().put("departmentList", departmentList);  // 准备回显的数据  Department department = departmentService.getById(model.getId());  ActionContext.*getContext*().getValueStack().push(department);  **if** (department.getParent() != **null**) {  parentId = department.getParent().getId();  }  **return** "editUI";  } |

从上面数据，departmentList可以清楚得到树形结构的值，在select中用list=“#departmentList”，而其中用model.getId()得到对象，从而回显出id和name，description，但是回显不了select中name=“parentId”的值，所以用赋值给parentId

**注意：我不知道为啥可以页面上拿到了parentId能自动转换为对象**

修改操作：

1. 通过id跳转到修改页面，并回显值

准备数据，应显示树状结构的所有部门列表，先使用所有部门列表代替

List<Department> departmentList = departmentService.getAll();

ActionContext.*getContext*().put("departmentList", departmentList);

准备回显的数据

--准备数据，是由于回显是可能需要其他对象的值，所以需要mapping存放，从而不影响取出数据

--回显数据，是回显对象的数据，但需要注意其中特殊数据，比如关联字段的时候，是通过id来联系的，但是由于需要取得相应的对象，所以要赋值给id，通过struts2的，可以通过id来自动得到相应的对象

1. 修改后跳转到列表页面

在保存时，mode可以保存对象的值，但是有关联对象是id，所以需要关联对象实体保存数据

Department parent = departmentService.getById(parentId);

model.setParent(parent);得到model对象

这才同update数据才可以修改得到

ActionContext.getContext().getValueStack().push(department);

（数据保存在值栈中）这个一般用于单个对象的回显作用

ActionContext.*getContext*().put("department", department);

（数据保存在mapping中）一般保存容器对象中