# 1. SSH整合

## 1.1 目标分析

SSH整合就是在项目中使用Struts、Hibernate和Spring框架，利用Spring进行整合。主要就是将项目中所需对象交由Spring来管理和创建，增强项目的扩展性等。

SSH整合的关键之处在于：

（1）对于Struts，要将各个Action类交由Spring创建；

（2）对于Hibernate，要将数据库配置、sessionFactory和事物交由Spring管理；

（3）三层架构中的其他对象也交由Spring管理。

下面就来搭建一个稍微完整的SSH框架。

## 1.2 新建项目

新建一个Java Web项目，引入下面包：

（1）JDBC驱动包、Hibernate包（包括Hibernate中提供了C3P0相关包）；

（2）Struts包（可包括log4j的包，注意版本匹配），以及Struts对Spring的支持包，在Struts的lib下能找到“struts2-spring-plugin-2.5.10.jar”这个包，必须引入。

（3）Spring中要引入如下包：

必须的5个包和一个commons-logging包；

Spring中数据库相关包：jdbc包、tx包（事物处理）和orm包（因为Hibernate是ORM框架，因此需要引用spring-orm包添加支持），即：“spring-jdbc-4.3.7.RELEASE.jar”、“spring-tx-4.3.7.RELEASE.jar”和“spring-orm-4.3.7.RELEASE.jar”。

Spring中对web支持的包，因为这是web项目，因此要引入，即“spring-web-4.3.7.RELEASE.jar”。

由于我们使用注解声明事物，因此就没有引入Aspectj包（但是Spring的AOP包是必须的）。

## 1.3 编写三层代码及部分配置（使用注解）

我们操作的是student表，字段还是“编号id，姓名name和入学时间entranceTime”，类型分别是int、varchar()和datetime。

首先新建com.ssh.entity包，编写Student实体类和映射文件。

实体类Student：

|  |
| --- |
| **package** com.ssh.entity;  **import** java.util.Date;  **public class** Student {  **private int id**;  **private** String **name**;  **private** Date **entranceTime**;  **public** Student() {}  *// 其他构造器、setter/getter/toString等* } |

映射文件Student.hbm.xml：

|  |
| --- |
| *<?***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 package="com.ssh.entity"**>  <**class name="Student" table="student"**>  <**id name="id" column="id"**>  <**generator class="assigned"**></**generator**>  </**id**>  <**property name="name" column="name"**></**property**>  <**property name="entranceTime" column="entranceTime"**></**property**>  </**class**> </**hibernate-mapping**> |

现在写DAO层（带有接口层）。由于每个DAO类都应该有增删改查的基本方法，因此我们写好一个BaseDao，其他DAO类只要继承此类即有基本的操作方法。而由于基础DAO需要适应不同的实体类型，因此用了泛型。

（1）IBaseDao：基础DAO的接口（在com.ssh.dao包中）

|  |
| --- |
| **package** com.ssh.dao;  **import** java.io.Serializable; **import** java.util.List;  */\*\*  \* 基础DAO接口提供基本操作方法的原型  \*/* **public interface** IBaseDao<T> {  */\*\*  \* 添加一个对象  \** ***@param t*** *对象  \*/* **void** add(T t);   */\*\*  \* 根据主键删除对象数据  \*/* **void** delete(T t);   */\*\*  \* 更新对象数据  \** ***@param t*** *对象  \*/* **void** update(T t);   */\*\*  \* 根据主键查询对象数据  \** ***@param clazz*** *类字节码文件  \** ***@param id*** *主键  \** ***@return*** *对象  \*/* T findById(Class<T> clazz, Serializable id);   */\*\*  \* 查询某实体所有的数据  \** ***@param clazz*** *类字节码文件  \** ***@return*** *对象集合  \*/* List<T> findAll(Class<T> clazz); } |

（2）BaseDaoImpl：基础DAO的实现（在com.ssh.dao.impl包中）。

|  |
| --- |
| **package** com.ssh.dao.impl;  **import** com.ssh.dao.IBaseDao; **import** org.hibernate.Session; **import** org.hibernate.SessionFactory;  **import** javax.annotation.Resource; **import** java.io.Serializable; **import** java.util.List;  **public class** BaseDaoImpl<T> **implements** IBaseDao<T> {   @Resource  **protected** SessionFactory **sessionFactory**;   @Override  **public void** add(T t) {  **sessionFactory**.getCurrentSession().save(t);  }   @Override  **public void** delete(T t) {  **sessionFactory**.getCurrentSession().delete(t);  }   @Override  **public void** update(T t) {  **sessionFactory**.getCurrentSession().update(t);  }   @Override  **public** T findById(Class<T> clazz, Serializable id) {  **return sessionFactory**.getCurrentSession().get(clazz, id);  }   @Override  **public** List<T> findAll(Class<T> clazz) {  Session session = **sessionFactory**.getCurrentSession();  **return** session.createQuery(**"from "** + clazz.getName(), clazz).list();  } } |

以上基础的DAO不用加入到Spring容器中，只要将下面继承他的类加入到容器中即可，因为基础DAO并不是其他DAO依赖的属性，而是父子类关系。但是其中引用的sessionFactory资源是Spring提供的，因此需要用@Resource引用sessionFactory。并且，为了方便其他DAO类使用sessionFactory，我们把这个修饰符写为protected，这样下面的DAO类可直接使用sessionFactory。

可以看到我们上面并没有处理Hibernate的事物，这是因为我们下面将要用声明式事物。

（3）示例IStudentDao：操作学生的DAO接口。这时每个实体的DAO接口都要继承基础DAO接口，这样才会有基础方法。接口中只要声明自己特有的方法：

|  |
| --- |
| **package** com.ssh.dao;  **import** com.ssh.entity.Student;  **import** java.util.List;  *// 这里继承的泛型参数就用Student* **public interface** IStudentDao **extends** IBaseDao<Student> {  */\*\*  \* 按姓名查找学生  \** ***@param name*** *姓名  \** ***@return*** *学生列表  \*/* List<Student> findByName(String name); } |

（4）示例StudentDaoImpl：操作学生的DAO实现。DAO类不仅要继承基础DAO类，还要实现对应的接口，比如这里的IStudentDao。这里要加入容器。

|  |
| --- |
| **package** com.ssh.dao.impl;  **import** com.ssh.dao.IStudentDao; **import** com.ssh.entity.Student; **import** org.hibernate.query.Query; **import** org.springframework.stereotype.Repository;  **import** java.util.List;  *// 该类既要继承，又要实现* @Repository **public class** StudentDaoImpl **extends** BaseDaoImpl<Student> **implements** IStudentDao {  @Override  **public** List<Student> findByName(String name) {  Query<Student> query = **sessionFactory**.getCurrentSession().createQuery(**"from Student where name = ?"**, Student.**class**);  **return** query.list();  } } |

（5）接下来在com.ssh.service包中写Service层。先写IStudentService（所有的接口都不要加入Spring容器）：

|  |
| --- |
| **package** com.ssh.service;  **public interface** IStudentService {  *// 添加学生* **void** add(Student student);  *// 删除学生(根据id)* **void** delete(**int** id);  *// 更新学生* **void** update(Student student);  *// 查找学生* Student findById(**int** id);  *// 查询所有学生* List<Student> findAll();  *// 查询指定姓名学生* List<Student> findByName(String name); } |

（6）在com.ssh.service.impl包中StudentServcieImpl类（注入StudentDao，并且，一定要在Service中添加事物管理（使用@Transactional）！否则Hibernate没有事物处理将会报错。）：

|  |
| --- |
| **package** com.ssh.service.impl;  **import** com.ssh.dao.IStudentDao; **import** com.ssh.entity.Student; **import** com.ssh.service.IStudentService; **import** org.springframework.stereotype.Service; **import** org.springframework.transaction.annotation.Transactional;  **import** javax.annotation.Resource; **import** java.util.List;  @Service *// 加入容器* **public class** StudentServiceImpl **implements** IStudentService {  @Resource  **private** IStudentDao **studentDao**;   @Override  @Transactional(readOnly = **false**)  **public void** add(Student student) {  **studentDao**.add(student);  }   @Override  @Transactional(readOnly = **false**)  **public void** delete(**int** id) {  Student student = **new** Student();  student.setId(id);  **studentDao**.delete(student);  }   @Override  @Transactional(readOnly = **false**)  **public void** update(Student student) {  **studentDao**.update(student);  }   @Override  @Transactional(readOnly = **true**)  **public** Student findById(**int** id) {  **return studentDao**.findById(Student.**class**, id);  }   @Override  @Transactional(readOnly = **true**)  **public** List<Student> findAll() {  **return studentDao**.findAll(Student.**class**);  }   @Override  @Transactional(readOnly = **true**)  **public** List<Student> findByName(String name) {  **return studentDao**.findByName(name);  } } |

（7）编写Action类。

需要注意，Spring默认创建对象是以单例模式创建的，这对于上述的Service和Dao层都没有问题，因为这两个层的类中并没有包含状态，不会产生线程安全问题，使用单例也能节省内存。

但是Struts不同，Struts是基于类接收参数、调用方法的，这样就必须使用多例模式，避免多线程访问时有数据安全问题。因此我们使用@Scope设置Action为多例。

|  |
| --- |
| **package** com.ssh.action;  **import** com.opensymphony.xwork2.ActionSupport; **import** com.ssh.entity.Student; **import** com.ssh.service.IStudentService; **import** org.springframework.context.annotation.Scope; **import** org.springframework.stereotype.Controller;  **import** javax.annotation.Resource;  @Controller *// Action也得加入容器中* @Scope(**"prototype"**) **public class** StudentAction **extends** ActionSupport {  @Resource  **private** IStudentService **studentService**; *// 维护service   // 接收的参数* **private** Student student;  **private int id**;  **private** String **name**;  *// 下面是处理的方法。共用了上述的参数。可以按需编写。* **public** String addStudent() {  **studentService**.add(student);  **return *SUCCESS***;  }  **public** String deleteStudent() {  **studentService**.delete(**id**);  **return *SUCCESS***;  }  **public** String updateStudent() {  **studentService**.update(student);  **return *SUCCESS***;  }  **public** String findById() {  **studentService**.findById(**id**);  **return *SUCCESS***;  }  **public** String findAll() {  **studentService**.findAll();  **return *SUCCESS***;  }  **public** String findByName() {  **studentService**.findByName(**name**);  **return *SUCCESS***;  }   *// getter和setter* **public** Student getStudent() {  **return** student;  }   **public void** setStudent(Student student) {  **this**.student = student;  }   **public int** getId() {  **return id**;  }   **public void** setId(**int** id) {  **this**.**id** = id;  }   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  } } |

（8）Action的配置（struts.xml）。其中的class属性值不再写具体的类名了，因为Action对象由Spring创建了，这里的class值只要写引用即可，所以直接写“studentAction”：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* **<!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"  "http://struts.apache.org/dtds/struts-2.5.dtd"*>*** <**struts**>  <**package name="default" namespace="/" extends="struts-default"**>  <**action name="addStudent" class="studentAction" method="addStudent"**>  <**result name="success"**>/index.jsp</**result**>  </**action**>  <**action name="deleteStudent" class="studentAction" method="deleteStudent"**>  <**result name="success"**>/index.jsp</**result**>  </**action**>  <**action name="updateStudent" class="studentAction" method="updateStudent"**>  <**result name="success"**>/index.jsp</**result**>  </**action**>  <**action name="findById" class="studentAction" method="findById"**>  <**result name="success"**>/index.jsp</**result**>  </**action**>  <**action name="findAll" class="studentAction" method="findAll"**>  <**result name="success"**>/index.jsp</**result**>  </**action**>  <**action name="findByName" class="studentAction" method="findByName"**>  <**result name="success"**>/index.jsp</**result**>  </**action**>  </**package**> </**struts**> |

这里只是简单的例子，具体的页面跳转并没有详细设置，可以自行编写。这里只要访问相关Action能够对数据操作即可。

## 1.4 Spring与Hibernate整合

主要是提供sessionFactory以及数据库相关配置。我们在src下新建bean.xml文件作为Spring的配置文件，内容如下：

注意还要开启注解扫描，这样我们不用再XML中配置DAO、Service和Action层对象。

|  |
| --- |
| *<?***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"  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"**>  *<!-- 开启注解扫描 -->* <**context:component-scan base-package="com.ssh"** />  *<!-- 数据源配置 -->* <**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/dbtest?useSSL=false&amp;useUnicode=true&amp;characterEncoding=utf8"** />  <**property name="user" value="root"** />  <**property name="password" value="123456"** />  <**property name="initialPoolSize" value="3"** />  <**property name="maxPoolSize" value="10"** />  <**property name="maxStatements" value="100"** />  <**property name="acquireIncrement" value="2"** />  </**bean**>  *<!-- 配置sessionFactory。class用的是Hibernate5包下的，对应即可。 -->  <!-- 实际上sessionFactory是一个接口，LocalSessionFactoryBean是Spring提供的一个实现类 -->* <**bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean"**>  *<!-- 引用dataSource -->* <**property name="dataSource" ref="dataSource"** />  *<!-- Hibernate映射文件配置 -->* <**property name="mappingLocations"**>  <**list**>  *<!-- 可用通配符 -->* <**value**>classpath:com/ssh/entity/\*.hbm.xml</**value**>  </**list**>  </**property**>  *<!-- Hibernate的其他配置，这样就无需hibernate主配置文件了，所有主配置都可在此配置 -->* <**property name="hibernateProperties"**>  <**props**>  *<!-- 注意，这里的key必须加上“hibernate...”前缀，否则无效！！！ -->* <**prop key="hibernate.dialect"**>org.hibernate.dialect.MySQLDialect</**prop**>  <**prop key="hibernate.show\_sql"**>true</**prop**>  <**prop key="hibernate.format\_sql"**>true</**prop**>  <**prop key="hibernate.hbm2ddl.auto"**>update</**prop**>  </**props**>  </**property**>  </**bean**>  *<!-- 配置使用注解来管理事物 -->  <!-- 事务管理器配置, Hibernate单数据源事务 -->* <**bean id="hbnTx" class="org.springframework.orm.hibernate5.HibernateTransactionManager"**>  <**property name="sessionFactory" ref="sessionFactory"** />  </**bean**>  *<!-- 使用annotation定义事务 -->* <**tx:annotation-driven transaction-manager="hbnTx" proxy-target-class="true"** /> </**beans**> |

说明：Spring与Hibernate的整合时，Hibernate的session默认就是以线程方式创建，不用再配置。

## 1.5 Spring与Struts的整合

由于我们在Action中使用了注解，因此无需在bean.xml中配置创建Action对象，否则还是要手动写XML的。其余的针对Struts没有什么配置。

## 1.6 web.xml配置

最后就是配置web.xml了，主要是配置Spring的配置文件，使得Web服务器能找到Spring配置文件并进行一系列操作，当然，其中还有一些对Struts和Hibernate的必要配置。

内容如下：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  version="3.1"**>  *<!-- 若想使用Hibernate的懒加载数据，一定要在配置Struts过滤器前，配置spring的OpenSessionInView模式，目的就是让JSP能访问懒加载数据。 -->* <**filter**>  <**filter-name**>OpenSessionInView</**filter-name**>  <**filter-class**>org.springframework.orm.hibernate5.support.OpenSessionInViewFilter</**filter-class**>  <**init-param**>  <**param-name**>singleSession</**param-name**>  <**param-value**>false</**param-value**>  </**init-param**>  </**filter**>  <**filter-mapping**>  <**filter-name**>OpenSessionInView</**filter-name**>  <**url-pattern**>\*.action</**url-pattern**>  </**filter-mapping**>  *<!-- Struts过滤器配置 -->* <**filter**>  <**filter-name**>struts2</**filter-name**>  <**filter-class**>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</**filter-class**>  </**filter**>  <**filter-mapping**>  <**filter-name**>struts2</**filter-name**>  <**url-pattern**>\*.action</**url-pattern**>  </**filter-mapping**>   *<!-- Spring配置。指明配置文件的位置。可使用通配符\* -->* <**context-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:bean.xml</**param-value**>  </**context-param**>  *<!-- Spring的监听器。Web应用中需要配置监听。 -->* <**listener**>  <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**>  </**listener**> </**web-app**> |

总结：从上述过程看出，搭建环境首先要熟悉其原理，一步一步完成，会发现逻辑清晰，环环相扣，并且这样开发者做事具有效率。

实际上也可以将Spring配置分离成多个文件，但是运行时会相当于一个整体。比如在src下存在bean1.xml和bean2.xml文件，那么定位Spring配置文件时，可使用“classpath:bean\*.xml”。其中的classpath就表示在classpath路径下找到文件。

对应于JdbcTemplate，Spring中也提供了HibernateTemplate，但是并不推荐使用。因为HibernateTemplate是Spring提供的，不利于解耦。

此外，Java Web项目可以打包成war包，可以直接将war包复制到Tomcat/webapps目录下，当启动Tomcat服务器时，服务器会自动解压war包并加载此项目。用war包进行部署非常方便。

本章的实例文件保存在sshdemo.zip文件中，项目是用Intellij IDEA继承开发环境搭建的。