基于Myeclipse6.5的SSH框架整合

Mx25184

2012-08-14

## 1、文章目的

最近整合SSH框架,期间遇到很多问题,中途几次熬夜查找,精神几度崩溃,心想如此好的框架为啥搭建整合的时候确实如此费劲,在网上找了很多解决方案,均不能解决问题。为了以后让想学SSH的人少走点弯路，故此次将整个完整搭建步骤写出来，供大家分享，希望对大家有所帮助。

## 2、运行环境

1、jdk-6u1-windows-i586-p

2、MyEclipse\_6.5.0GA\_E3.3.2\_Installer\_A

3、apache-tomcat-6.0.13.exe

4、mysql-essential-5.1.53-win32

5、数据库脚本程序

create database SSH;

create table user(

id int(11) not null auto\_increment,

username varchar(50) default null,

password varchar(50) default null,

primary key(id)

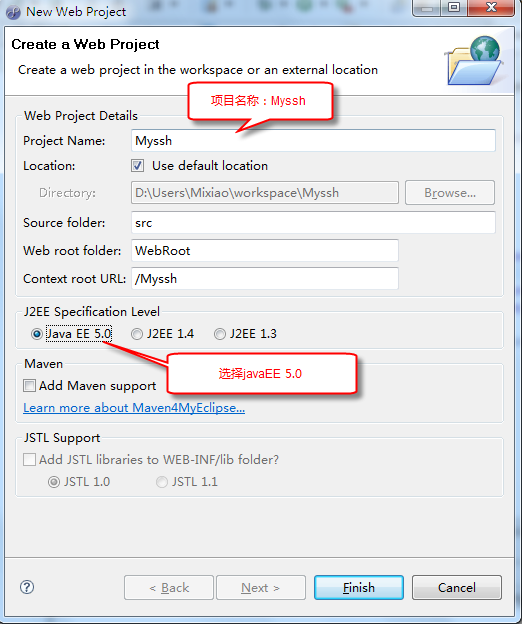
);

insert into user values('1','admin','admin');

具体环境搭建基本上就是下载安装，继续流程就不在详细说明了，一般网上都可以查到。

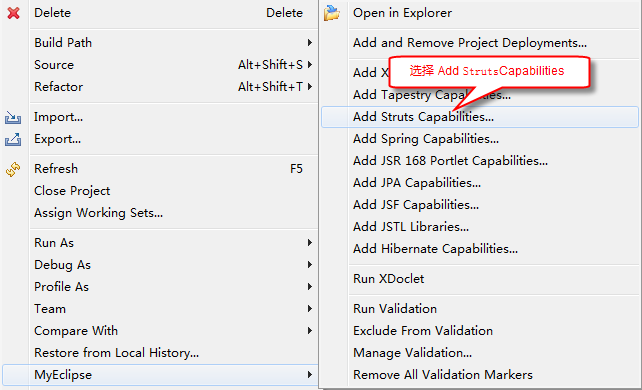
## 3、创建项目工程

再打开的Myeclipse中，新建Web project，项目名字我们起名 Myssh ，J2EE Sqecification Level选择Java EE5.0，然后点击Finish即可。

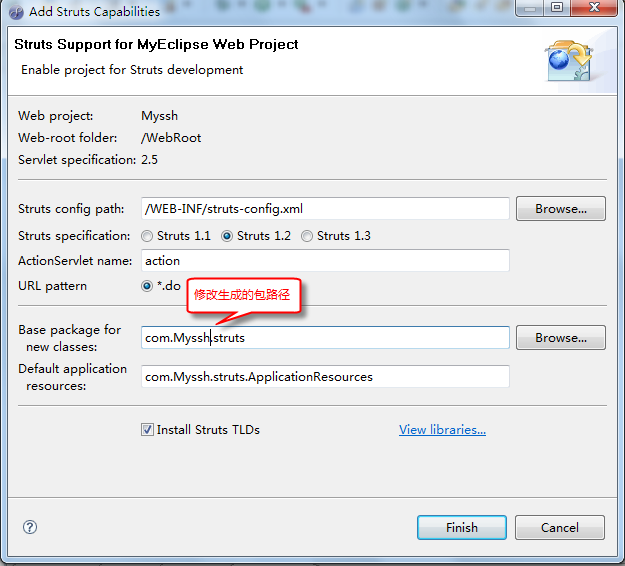


## 4、添加Struts1.2框架支持

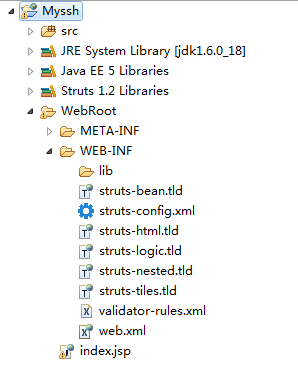
接下来我们将添加Struts框架支持，在项目Myssh上面右键，选择Myeclipse >> Add Struts Capabilities…



在弹出的Add Struts Capabilities 对话框中修改 Base package for new classes中的路径为com.Myss.struts，其他不变，点击finish即可

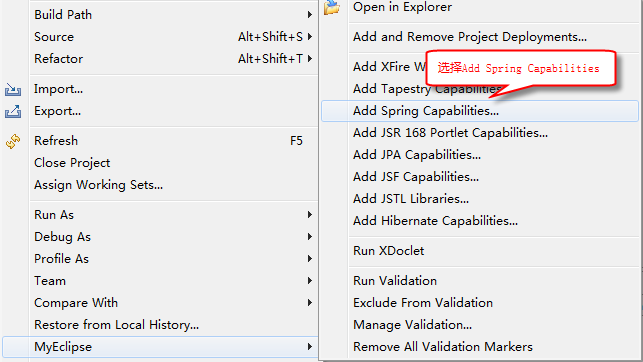


完成添加后的项目路径结构为：



## 5、添加spring2.5框架支持

和添加struts框架一样，在项目Myssh上面右键，选择Myeclipse >> Add Spring Capabilities…

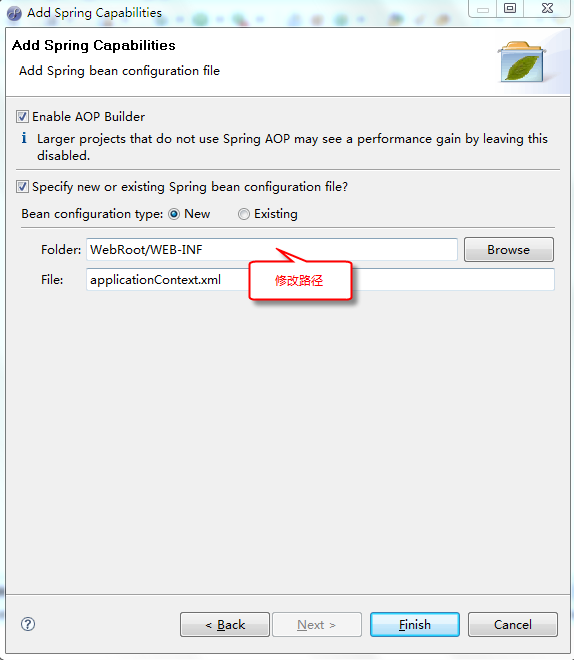


在弹出的Add Spring Capabilities 对话框中选择Spring version ：Spring2.5

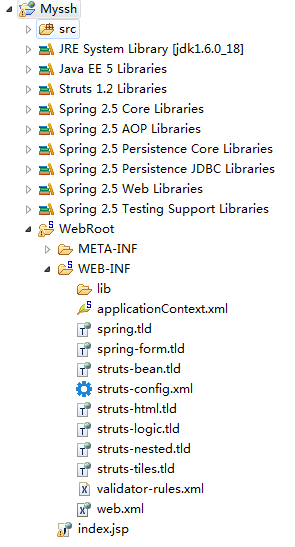
由于Spring采用的最小化发布方式，使得Spring的库文件是分离的，所以我们要选择自己需要添加的库：



选择好 点击next，然后修改folder路径为：WebRoot/WEB-INF 点击finish 即可

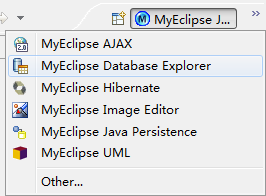


此时项目结构为：

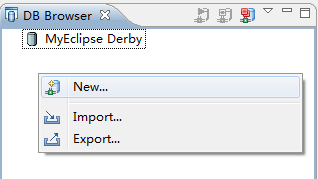


## 6、配置数据源

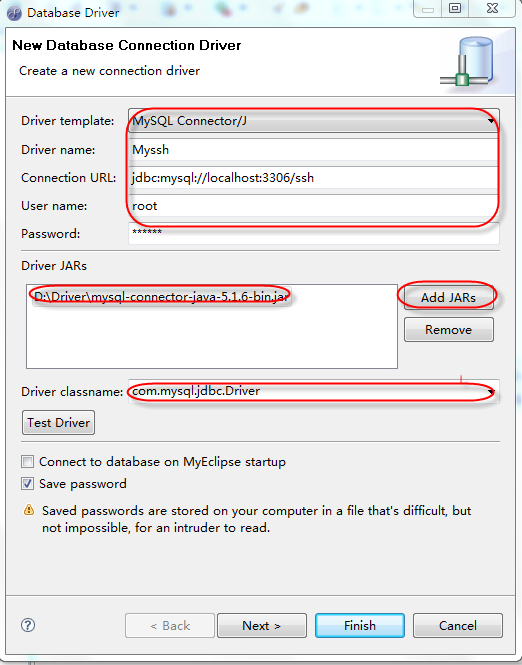
在Myeclipse右上角 打开 Myeclipse Database Explorer 数据库管理视图。



在左侧的 DB Browser 中 右击 New 打开新建对话框

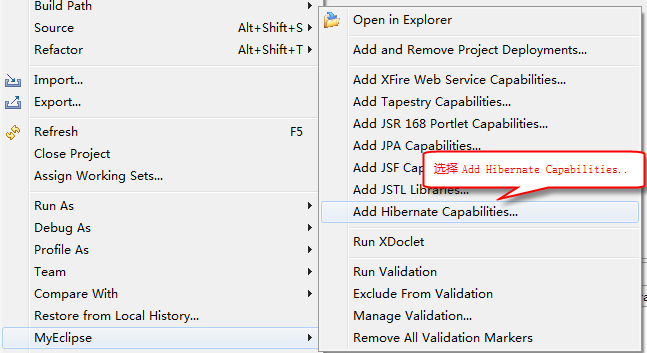


在Database Driver中，配置内容如下:选择Driver Template 为 MySQL，然后根据事情情况写URL，这里我们是本机操作，所以写为localhost下的 ssh ,User name 和 password 根据数据库实际填写。然后添加数据库驱动程序Add JARs，为了方便我这里选择了保存密码Save password 。然后点击finish即可。

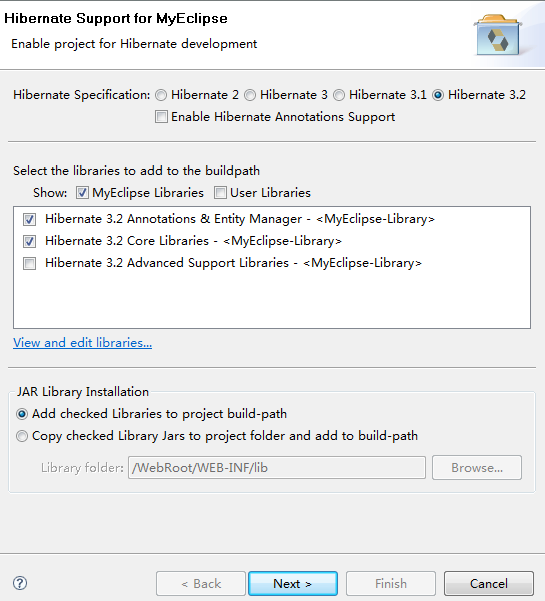


## 7、Spring和hibernate的整合

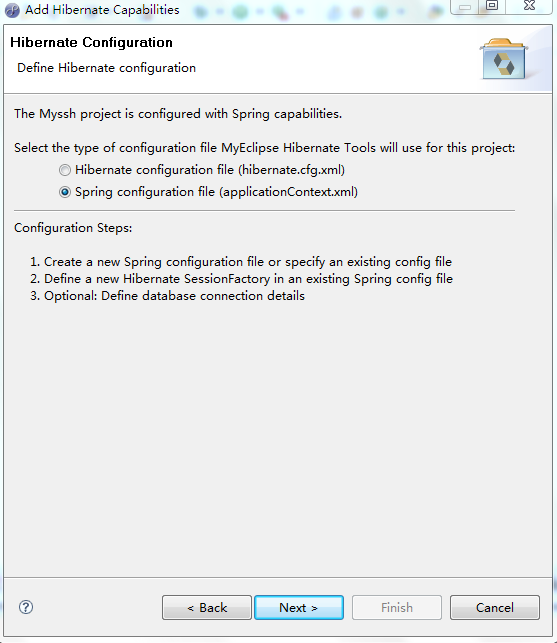
配置完数据源就可以添加 hibernate支持了，切换视图到 Myeclipse java enterprise下，在Myssh工程上右击，选择 Myeclipse>>Add Hibernate Capabilities..



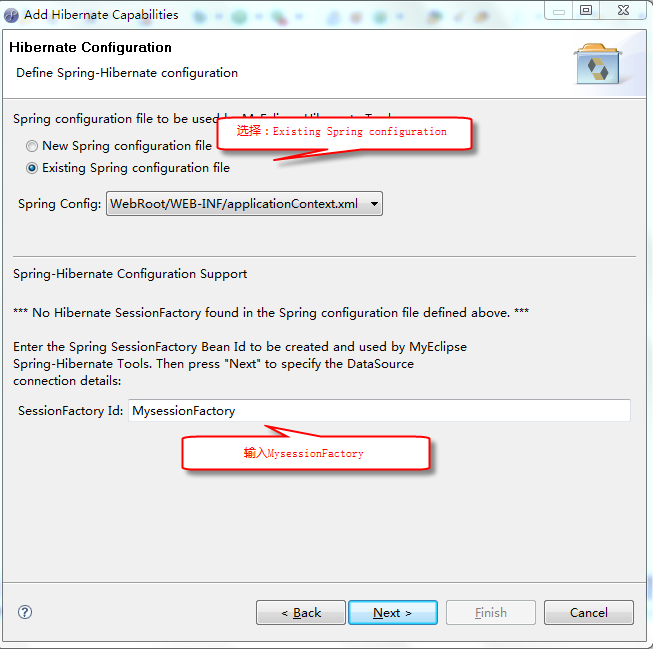
这一步我们选择默认即可 点击 Next



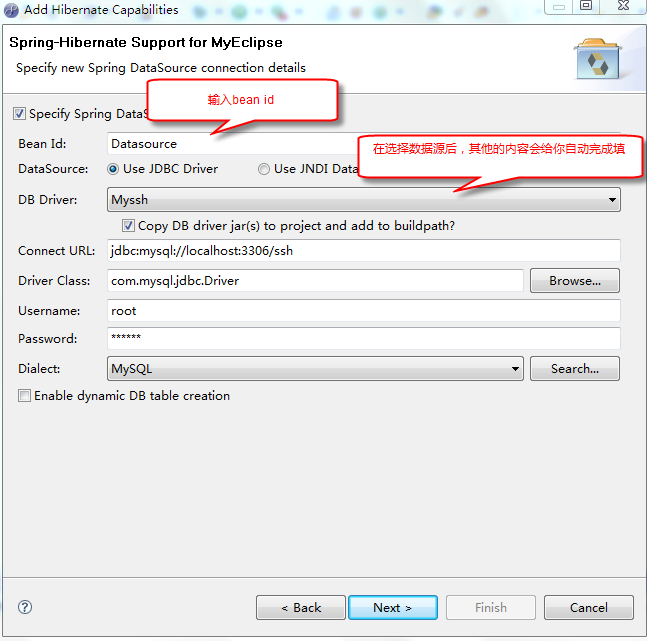
对话框中选择Spring configuration file，表示我们希望将Hibernate 托管给Spring进行管理，然后点击 Next；



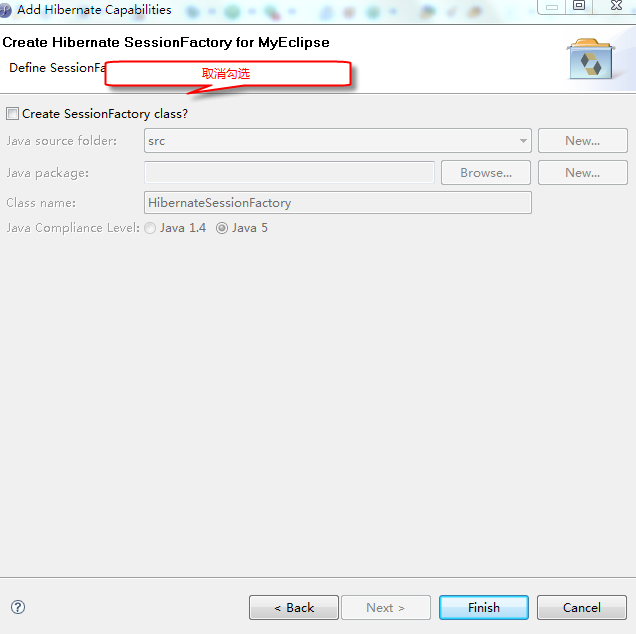
在出现的对话框中询问我们是否新建一个Spring配置文件，我们选择：Existing Spring configuration file。因为我们已经添加了Spring的配置文件了，所以我们选择已存在的配置文件即可，Myeclipse自动回查找，然后输入MysessionFactory，点击Next



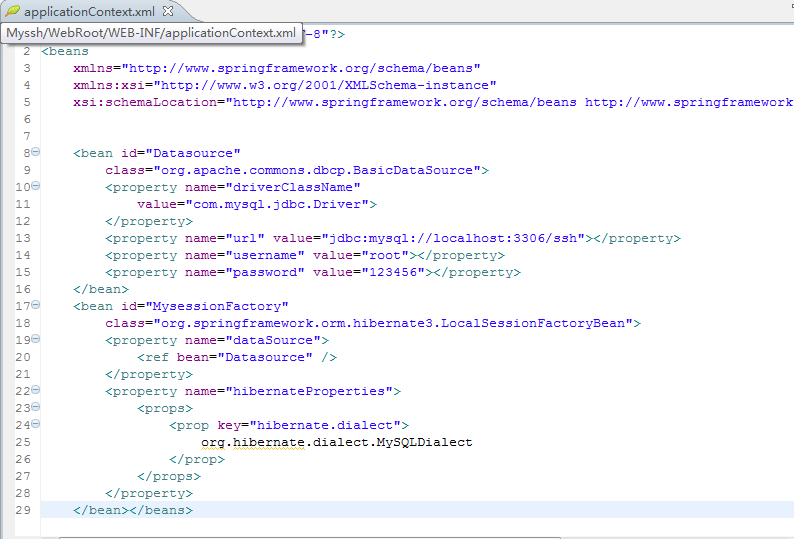
在对话框中输入 Bean id 数据源在Spring中的 Bean ID的名字：datasource 。然后选择 DB Driver 里面我们之前配置的Myssh后， Myeclipse会自动完成其他的填写。点击 Next



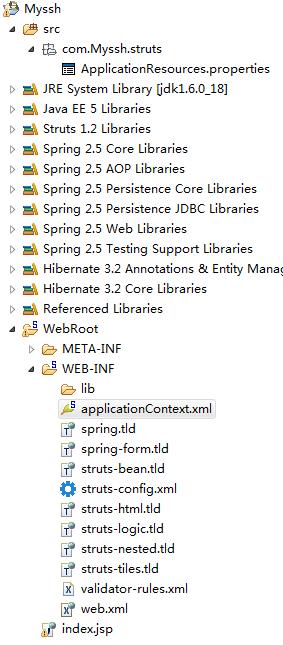
在对话框中取消创新新的hibernate sessionfactory，点击Finish即可。



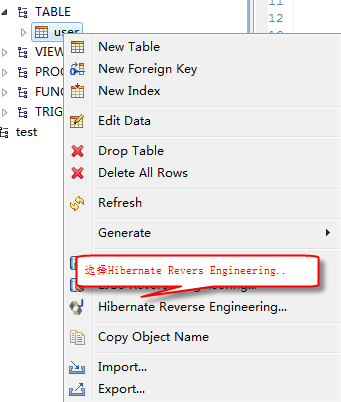
Myeclipse 自动默认打开 Spring配置文件：applicationContext.xml



此时项目结构：

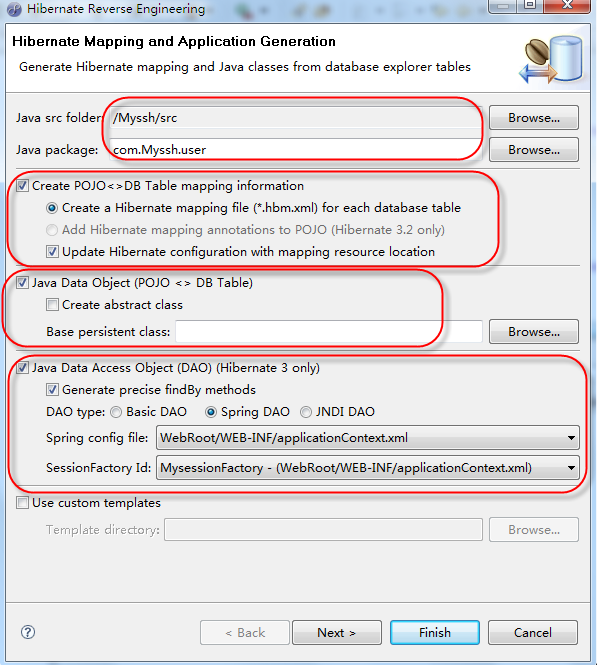


切换到 Myeclipse Database Explorer视图，双击打开数据库连接，找到表 user，然后右击表格，选择 Hibernate Revers Engineering..

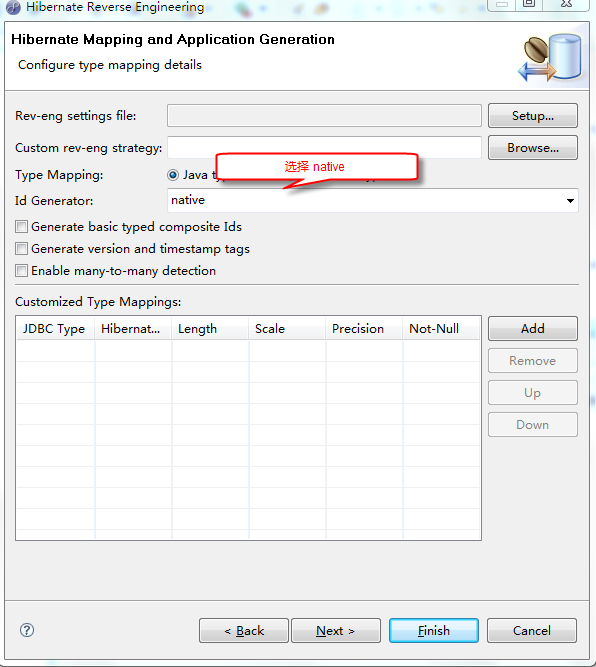


在对话框中修改Java src folder 路径为我们建立的/Myssh/src，然后填写java package 名称：com.Myssh.user

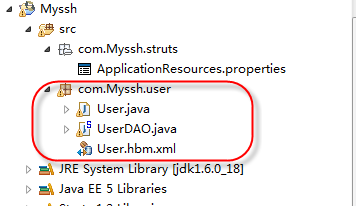
选择Create POJO，选择 java Data Object ,选择 Java Data Access Object 点击 Next



在出现的对话框中选择Id Generator为 native ，然后点击 Finish



此时项目结构为下图所示，其中User.java 是 myeclipse生成的使用面向对象的java语言对数据库表格进行的抽象，User.hbm.xml 是将数据库表格中的字段和POJO类的属性进行映射的定义，UserDAO.java封装了一些 Myeclipse自动生成的对数据库的操作。



这是我们修改一下文件目录，新建一个 dao包，将DAO类与 POJO类分开，然后我们在Struts包下面建立action和form包。为了将实现与接口进行解耦，我们在dao包下面添加接口，然后创建dao.impl包，将实际的DAO类放在这里，DAO类是直接与数据库打交道的类。为了对业务逻辑进行封装，我们将业务逻辑全部写在Service类里面，和dao一样，我们建立service包，里面添加业务接口，具体实现放在service.impl里面。

将 UserDAO.java 移动到dao的Impl包下面，并在dao包下面建立接口 IUserDAO：

**package** com.Myssh.dao;

**import** java.util.List;

**import** javax.xml.registry.infomodel.User;

**public** **interface** IUserDAO {

**public** User findById(Integer id);

**public** List findByUsername(Object username);

**public** **void** save(User user);

}

UserDAO是自动生成的，我们可以简单的修改一下。让UserDAO实现接口IuserDAO,不然在启动Spring的时候会报错。

**public** **class** UserDAO **extends** HibernateDaoSupport **implements** IUserDAO{

**private** **static** **final** Log *log* = LogFactory.*getLog*(UserDAO.**class**);

// property constants

**public** **static** **final** String *USERNAME* = "username";

**public** **static** **final** String *PASSWORD* = "password";

**protected** **void** initDao() {

// do nothing

}

**public** **void** save(User user) {

*log*.debug("saving User instance");

**try** {

getHibernateTemplate().save(user);

*log*.debug("save successful");

} **catch** (RuntimeException re) {

*log*.error("save failed", re);

**throw** re;

}

}

**public** **void** delete(User persistentInstance) {

*log*.debug("deleting User instance");

**try** {

getHibernateTemplate().delete(persistentInstance);

*log*.debug("delete successful");

} **catch** (RuntimeException re) {

*log*.error("delete failed", re);

**throw** re;

}

}

**public** User findById(java.lang.Integer id) {

*log*.debug("getting User instance with id: " + id);

**try** {

User instance = (User) getHibernateTemplate().get(

"com.Myssh.user.User", id);

**return** instance;

} **catch** (RuntimeException re) {

*log*.error("get failed", re);

**throw** re;

}

}

**public** List findByExample(User instance) {

*log*.debug("finding User instance by example");

**try** {

List results = getHibernateTemplate().findByExample(instance);

*log*.debug("find by example successful, result size: "

+ results.size());

**return** results;

} **catch** (RuntimeException re) {

*log*.error("find by example failed", re);

**throw** re;

}

}

**public** List findByProperty(String propertyName, Object value) {

*log*.debug("finding User instance with property: " + propertyName

+ ", value: " + value);

**try** {

String queryString = "from User as model where model."

+ propertyName + "= ?";

**return** getHibernateTemplate().find(queryString, value);

} **catch** (RuntimeException re) {

*log*.error("find by property name failed", re);

**throw** re;

}

}

**public** List findByUsername(Object username) {

**return** findByProperty(*USERNAME*, username);

}

**public** List findByPassword(Object password) {

**return** findByProperty(*PASSWORD*, password);

}

**public** List findAll() {

*log*.debug("finding all User instances");

**try** {

String queryString = "from User";

**return** getHibernateTemplate().find(queryString);

} **catch** (RuntimeException re) {

*log*.error("find all failed", re);

**throw** re;

}

}

**public** User merge(User detachedInstance) {

*log*.debug("merging User instance");

**try** {

User result = (User) getHibernateTemplate().merge(detachedInstance);

*log*.debug("merge successful");

**return** result;

} **catch** (RuntimeException re) {

*log*.error("merge failed", re);

**throw** re;

}

}

**public** **void** attachDirty(User instance) {

*log*.debug("attaching dirty User instance");

**try** {

getHibernateTemplate().saveOrUpdate(instance);

*log*.debug("attach successful");

} **catch** (RuntimeException re) {

*log*.error("attach failed", re);

**throw** re;

}

}

**public** **void** attachClean(User instance) {

*log*.debug("attaching clean User instance");

**try** {

getHibernateTemplate().lock(instance, LockMode.*NONE*);

*log*.debug("attach successful");

} **catch** (RuntimeException re) {

*log*.error("attach failed", re);

**throw** re;

}

}

**public** **static** UserDAO getFromApplicationContext(ApplicationContext ctx) {

**return** (UserDAO) ctx.getBean("UserDAO");

}

}

在service中建立接口 IuserService.java，在service 的Impl中建立UserService.java

**package** com.Myssh.service;

**import** javax.xml.registry.infomodel.User;

**public** **interface** IUserService {

**public** User getuserById(Integer id);

**public** User getUserByUsername(String username);

**public** **void** addUser(User user);

}

**UserService.java**

**package** com.Myssh.service.impl;

**import** java.util.List;

**import** com.Myssh.dao.IUserDAO;

**import** com.Myssh.dao.impl.UserDAO;

**import** com.Myssh.service.IUserService;

**import** com.Myssh.user.User;

**public** **class** UserService **implements** IUserService {

**private** IUserDAO userDAO;

**public** **void** addUser(User user) {

// **TODO** Auto-generated method stub

userDAO.save(user);

}

**public** User getUserByUsername(String username) {

// **TODO** Auto-generated method stub

List list = userDAO.findByUsername(username);

**if**(list.size()==0){

**return** **null**;

}**else** {

**return** (User) list.get(0);

}

}

**public** User getuserById(Integer id) {

// **TODO** Auto-generated method stub

**return** userDAO.findById(id);

}

**public** IUserDAO getUserDAO() {

**return** userDAO;

}

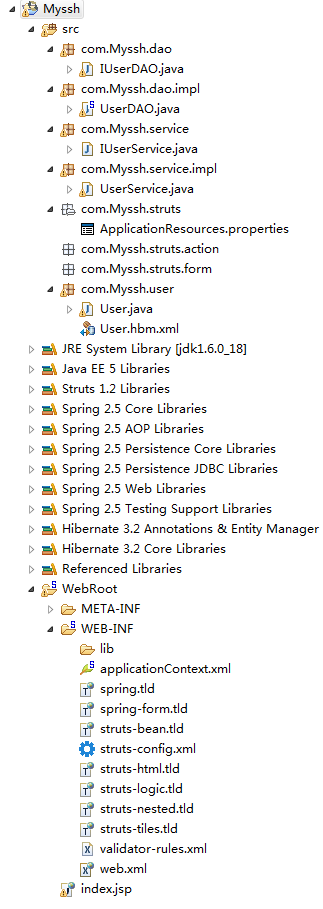
**public** **void** setUserDAO(IUserDAO userDAO) {

**this**.userDAO = userDAO;

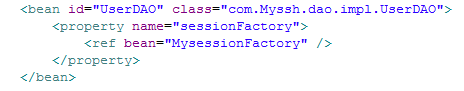
}

}

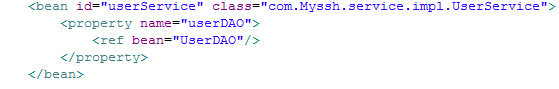
此时的工程目录结构为：

****

这时，Spring配置文件 applicationContext.xml 内容会自动添加如下信息：



然后加入bean的初始化配置。



## 8、整合Spring和Struts

为了能够让Spring托管Struts的类，在网上找了一个方式，只需让Struts从Spring获得需要的类就可以了，为了达到这个目标，我们创建一个类BaseAction

**package** com.Myssh.struts.action;

**import** org.springframework.web.context.WebApplicationContext;

**import** org.springframework.web.context.support.WebApplicationContextUtils;

**import** org.springframework.web.struts.ActionSupport;

**public** **class** BesaAction **extends** ActionSupport {

**protected** Object getBean(String id) {

WebApplicationContext ctxApplicationContext = WebApplicationContextUtils

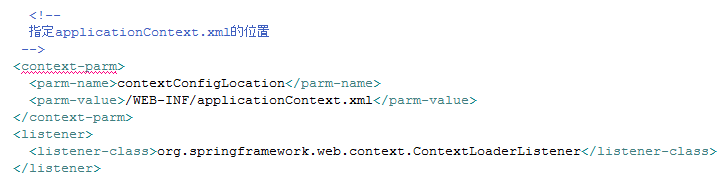
.*getWebApplicationContext*(**this**.servlet.getServletContext());

**return** ctxApplicationContext.getBean(id);

}

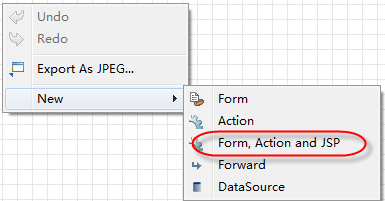
}

然后让容器初始化Spring，我们修改WEB.xml文件。添加如下代码：

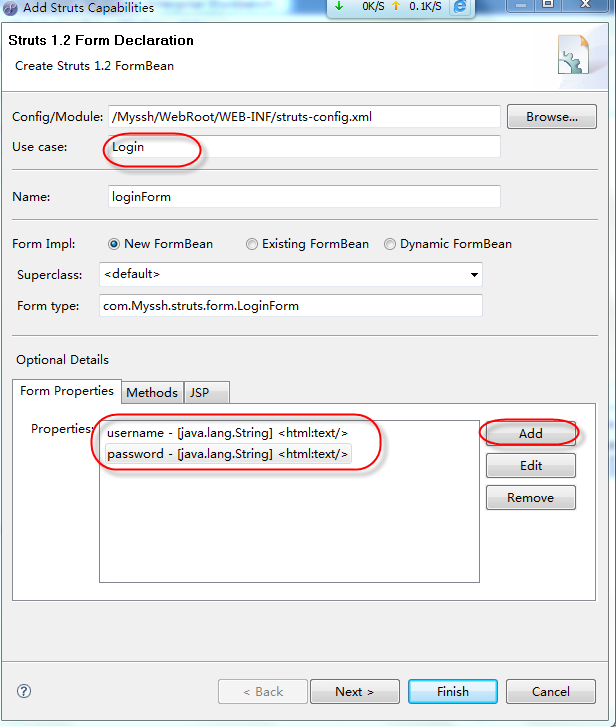
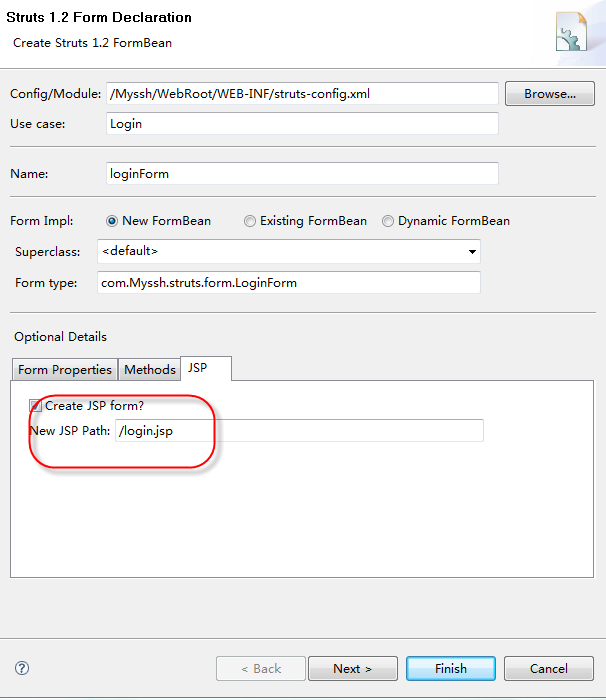
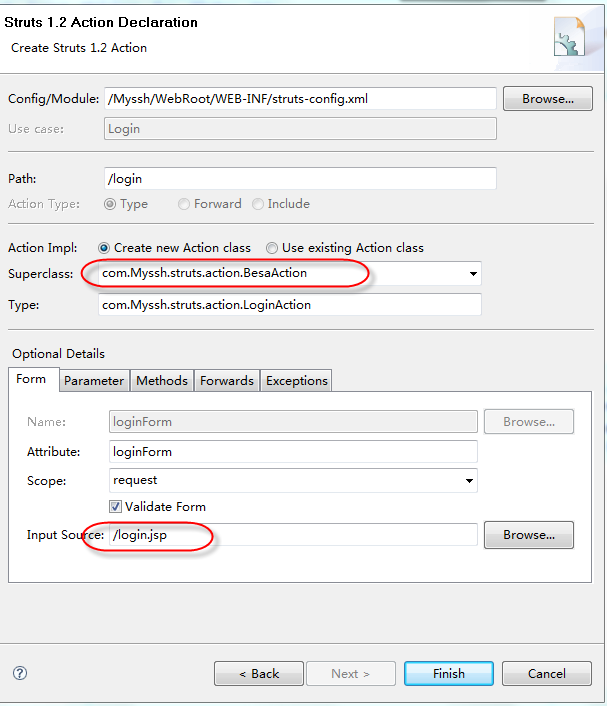


## 9、实例测试

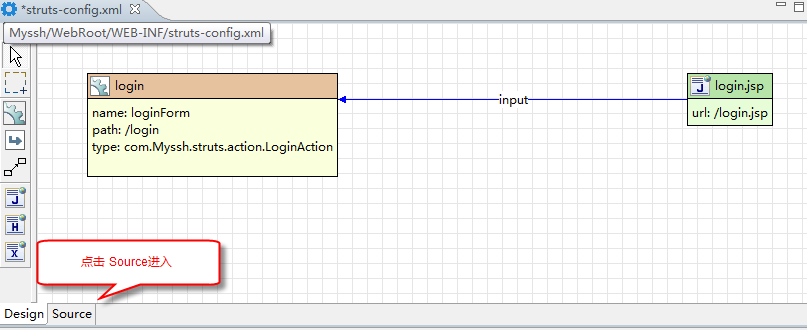
至此配置已经差不多了，我们可以测试一下效果。首先创建一个基于Struts的jsp页面，打开WEB-INF下面的 struts-config.xml，单击右键，选择New>>Form ,Action and jsp.



在弹出的对话框中添加 User case，然后点击 Add 生成 Properties代码。这会由Myeclipse 自动生成相应的Form代码，输入完成后选择JSP选项卡，选上Create JSP form，修改路径 点击Next。



然后我们点击Struts-config.xml 左下角的 Source进入XML编辑页面，添加转发选项。

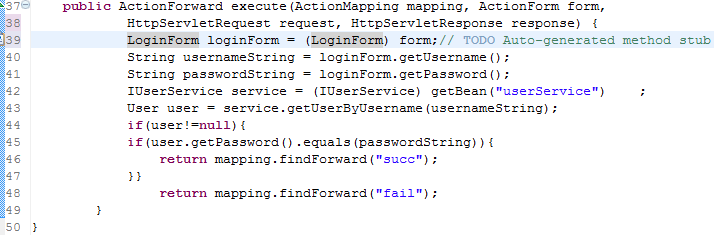




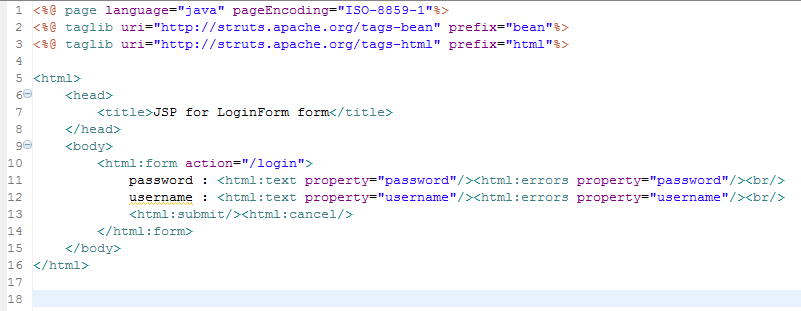


然后我们去index.jsp 一个目录下新建两个jsp页面，一个是succ.jsp, 一个是 fail.jsp。他们的内容分别是简单的一句话登录成功和登录失败。

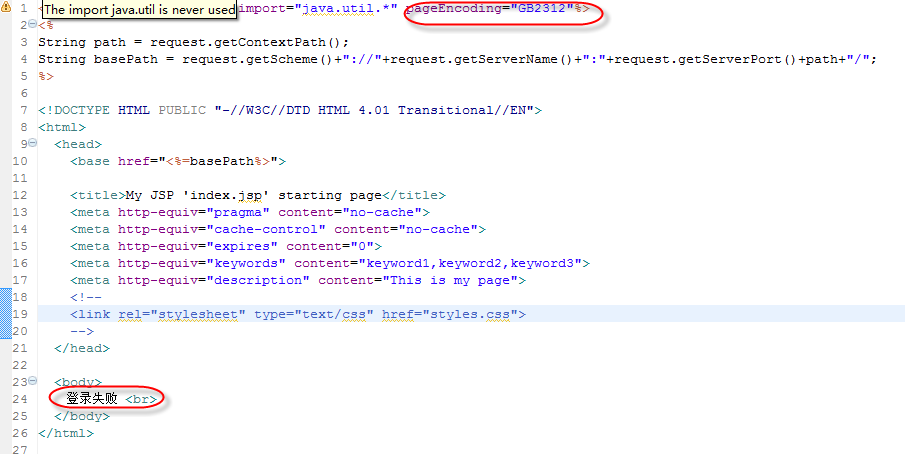
到这里我们就去修改LoginAction类内容如下：



Login.jsp 内容如下：

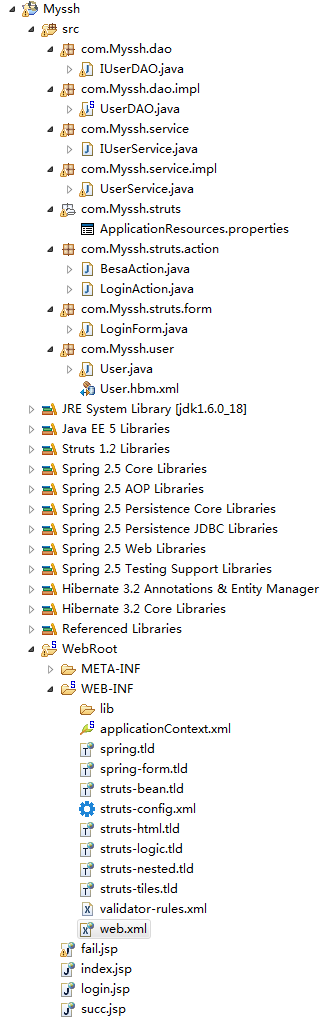


Succ.jsp 这个添加登录失败 汉字是 需要修改字符集：GB2312

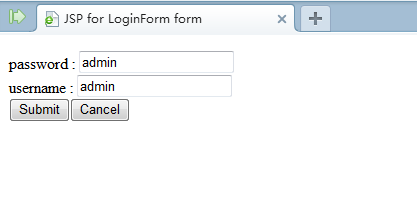


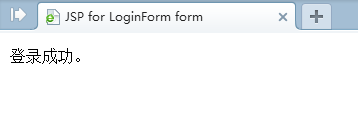
剩下的那个 succ.jsp界面我就不截图了，和上面这个一样。只是内容为：登录成功

到此为止，我们的配置极其逻辑代码都已经写完了，项目文件结构如下图：



启动 tomcat 查看。





## 10、注意事项

1、由于Myeclipse中的 Spring 和 hibernate框架带有的 jar可能重复引用，所以理论上我们建立SSH框架是就应该删除重复jar包，我直接的从Myeclipse下面：D:\Users\administrator\AppData\Roaming\MyEclipse 6.5\myeclipse\eclipse\plugins\com.genuitec.eclipse.springframework\_6.5.0.zmyeclipse650200806\data\2.0\lib\asm\下面的asm-2.2.3.jar 删除了。相应的文件删除根据自己安装目录的路径为主。

2、别忘记UserDAO 去实现 IUserDAO接口。否则一经启动不起来。