# 实训10 SSH集成方法练习

实验目的：了解SSH的集成方法，掌握搭建SSH框架的能力。

作业要求：

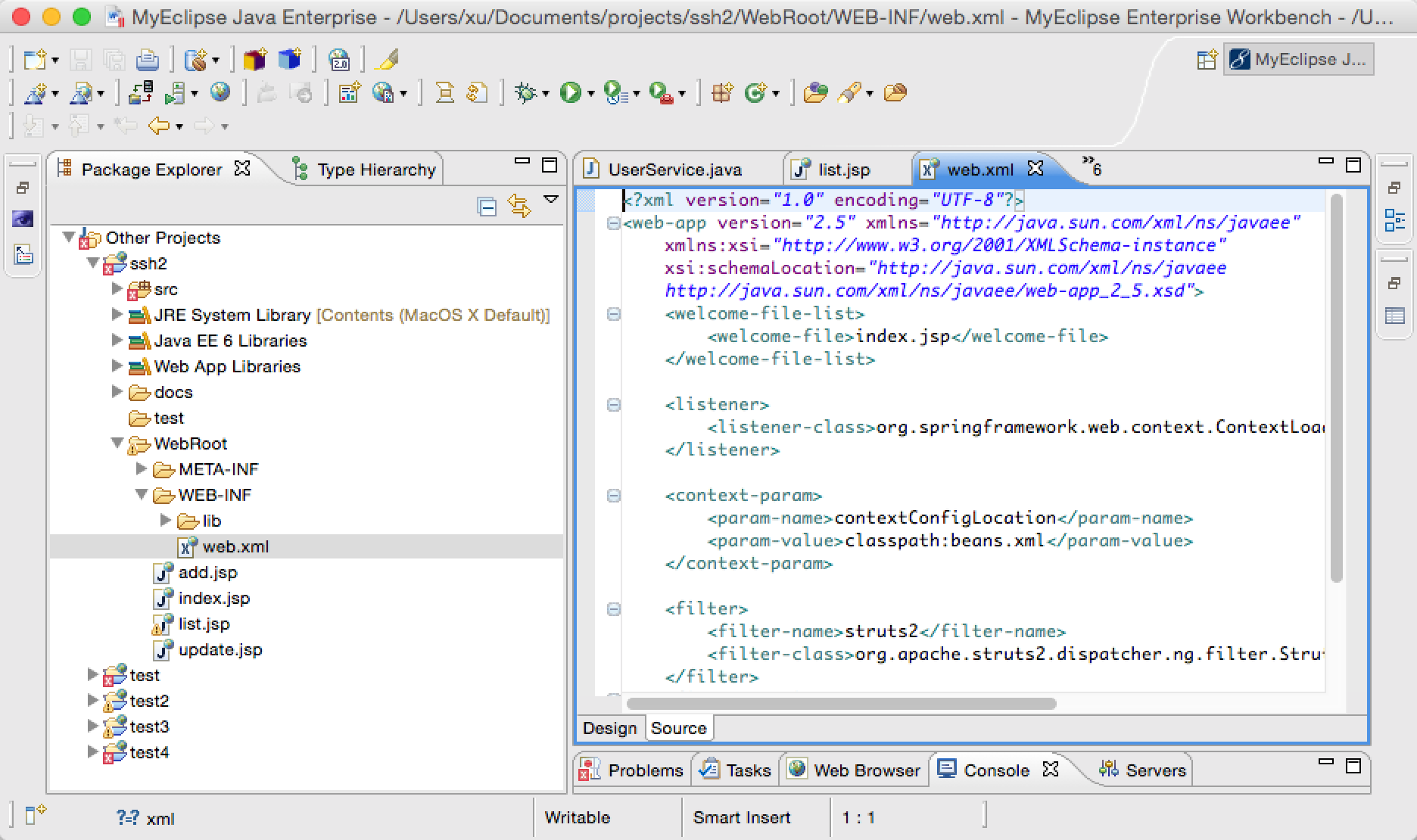
1. 最终内容打包成学号+姓名.rar格式
2. 功能：通过SSH实现登陆界面功能
3. 登陆界面包括用户名，密码。
4. 根据个人情况可以继续实现其他功能（包括增加，删除，修改，生成excel等）（参考P50页面）

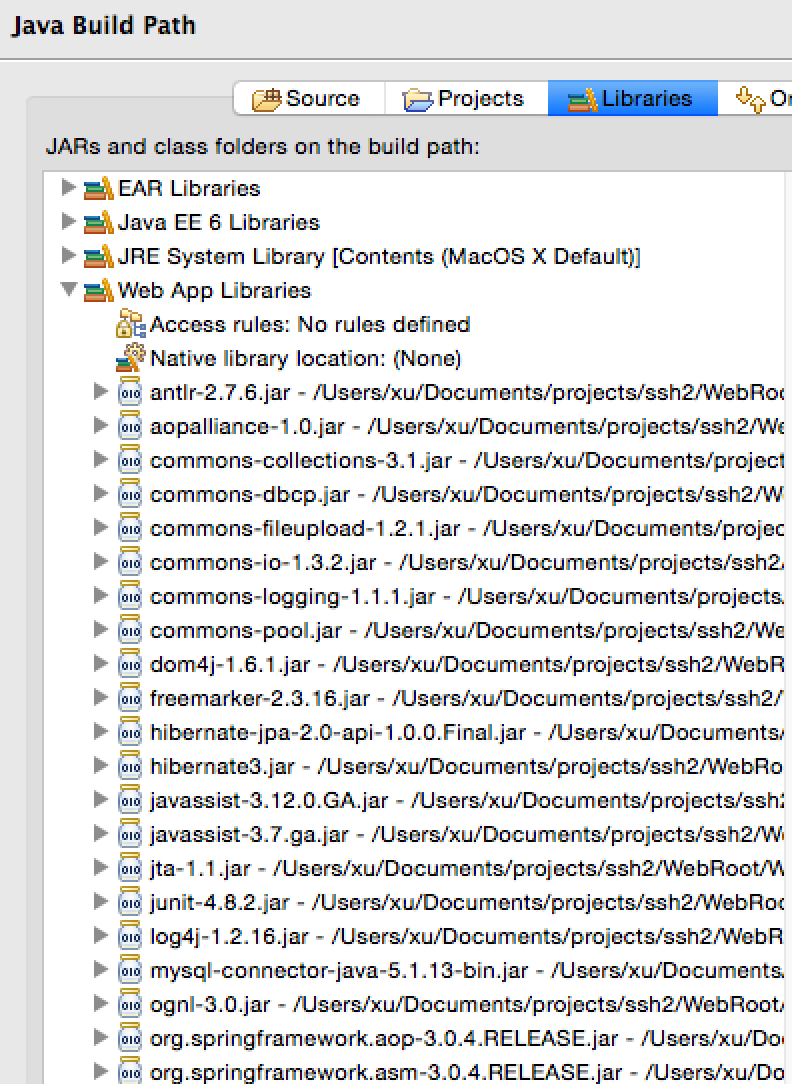
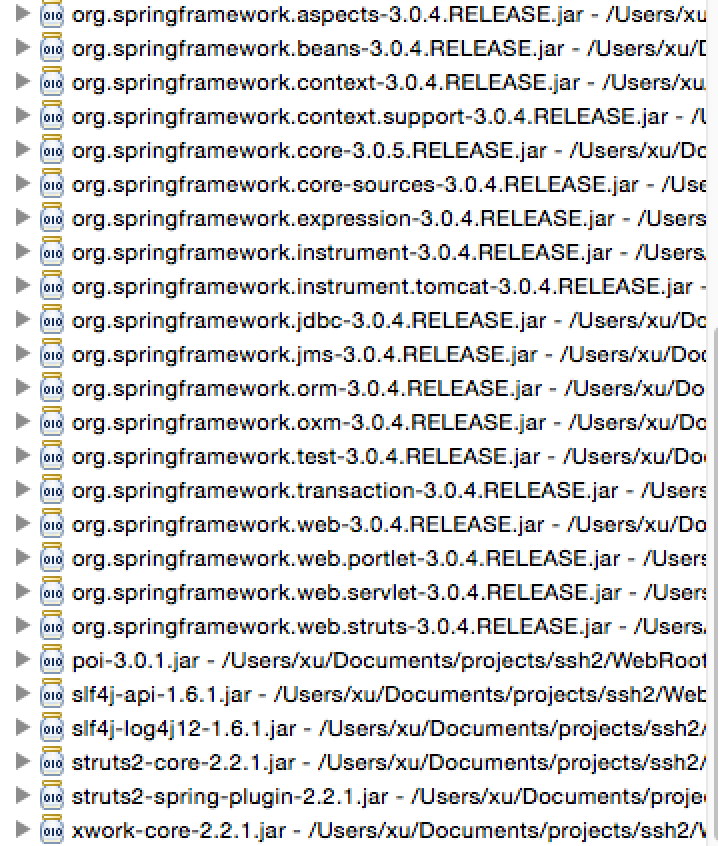
# SSH集成流程。

## 集成环境

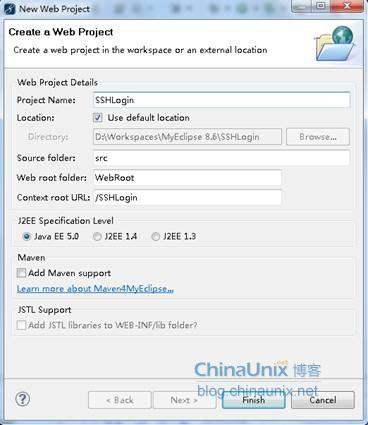
SSH集成使用版本：Struts 2.1 + Spring 3.0 + Hibernate 3.3

IDE：MyEclipse8.6／MyEclipse10



## 新建工程：SSHLogin



Web工程创建完毕后，从开发环境中，我们可以看到MyEclipse为我们创建了

* 一个JSP主页面——index.[jsp](http://biancheng.dnbcw.info/jsp/)
* 一个Web工程配置文件——web.xml

打开配置文件web.xml，可以发现：它只定义了一个欢迎页——index.[jsp](http://biancheng.dnbcw.info/jsp/)。

当我们去启动一个WEB项目时，容器包括（JBoss、Tomcat等）首先会读取项目web.xml配置文件里的配置，当这一步骤没有出错并且完成之后，项目才能正常地被启动起来。

  启动WEB项目的时候，容器首先会去它的配置文件web.xml读取两个节点:

<listener></listener>和<context-param></context-param>。

  紧接着，容器创建一个ServletContext（application），这个WEB项目所有部分都将共享这个上下文。

  容器以<context-param></context-param>的name作为键，value作为值，将其转化为键值对，存入ServletContext。

  容器创建<listener></listener>中的类实例，根据配置的class类路径<listener-class>来创建监听，在监听中会有contextInitialized(ServletContextEvent args)初始化方法，启动Web应用时，系统调用Listener的该方法，在这个方法中获得：

ServletContext application =ServletContextEvent.getServletContext();

context-param的值= application.getInitParameter("context-param的键");

得到这个context-param的值之后，你就可以做一些操作了。

   举例：你可能想在项目启动之前就打开数据库，那么这里就可以在<context-param>中设置数据库的连接方式（驱动、url、user、password），在监听类中初始化数据库的连接。这个监听是自己写的一个类，除了初始化方法，它还有销毁方法，用于关闭应用前释放资源。比如:说数据库连接的关闭，此时，调用contextDestroyed(ServletContextEvent args)，关闭Web应用时，系统调用Listener的该方法。

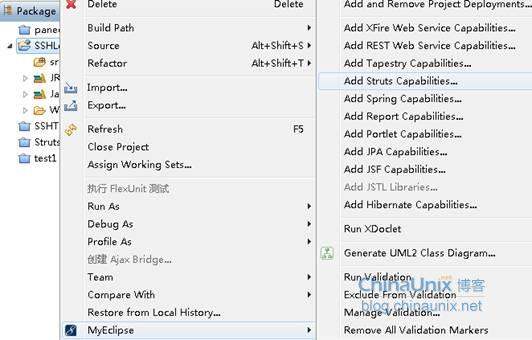
  接着，容器会读取<filter></filter>，根据指定的类路径来实例化过滤器。

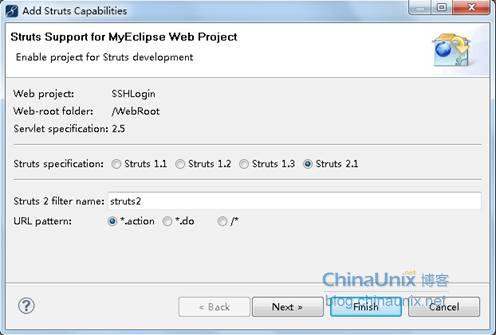
  以上都是在WEB项目还没有完全启动起来的时候就已经完成了的工作。如果系统中有Servlet，则Servlet是在第一次发起请求的时候被实例化的，而且一般不会被容器销毁，它可以服务于多个用户的请求。所以，Servlet的初始化都要比上面提到的那几个要迟。

总的来说，web.xml的加载顺序是: <context-param>-> <listener> -> <filter> -> <servlet>。其中，如果web.xml中出现了相同的元素，则按照在配置文件中出现的先后顺序来加载。

## 配置Struts

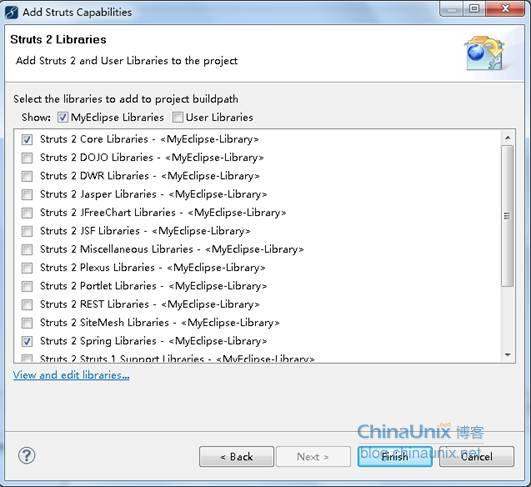
1. 选中工程，右击添加Struts功能支持，如图所示。





1. 随后，在弹出的窗体上选择struts2.1，选择Struts2核心库（注意：Struts2 Spring Libraries是整合Spring所必须的，但是，现阶段如果不使用Spring，就不要添加这个库，否则，测试可能失败）。单击“Finish”，完成Struts功能添加。

注意: 添加Struts功能支持时，不要启动Tomcat[服务器](http://biancheng.dnbcw.info/win2003/)，以免库添加失败。另外，Struts2的默认后缀名为\*.action（一般情况action的后缀是action，do也可以没有后缀），可以根据个人爱好修改。



1. 完成Struts2组件添加后，可以发现：除了新增加了一个Struts2的配置文件struts.xml外，Struts2在web.xml配置文件中还添加了自己的控制器及HTTP请求的过滤器映射，内容如下：

<!——定义控制器（过滤器）名称及实现的类文件——>

  <filter>

    <filter-name>struts2</filter-name>

    <filter-class>

        org.[apache](http://biancheng.dnbcw.info/apache/).struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter

    </filter-class>

  </filter>

  <!——定义过滤器映射——>

  <filter-mapping>

    <filter-name>struts2</filter-name>

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

  </filter-mapping>

完成上述配置修改后，以后发送给该工程的HTTP请求，如果是以\*.action结尾，则请求会转发给控制器StrutsPrepareAndExecuteFilter处理。Struts2的控制器会读取配置文件struts.xml，根据这个文件的定义，选择相应的action，决定页面跳转。

打开配置文件struts.xml，可以发现：初次添加Struts2支持后，该文件是空的，这需要我们根据实际情况自己添加。

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

<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN" "http://struts.[apache](http://biancheng.dnbcw.info/apache/).org/dtds/struts-2.1.dtd">

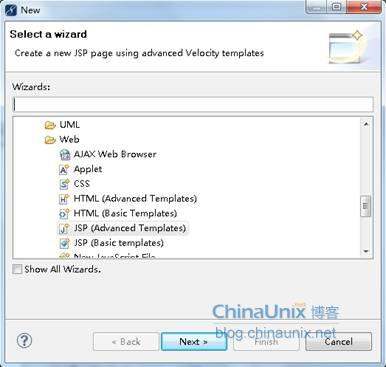
<struts>

</struts>

## Struts2的测试

1. 表示层的设计与实现

本工程表示层的页面有三个。主页面（index.jsp）提示用户输入用户名、密码，进行登录。登录成功，系统显示Success.jsp；登录失败，系统显示Failure.jsp。如图，创建3个JSP页面，注意页面编码采用：UTF-8，页面代码如下：



① index.jsp

<%@ page language=*"*[*java*](http://biancheng.dnbcw.info/java/)*"* import=*"[java](http://biancheng.dnbcw.info/java/).util.\*"* pageEncoding=*"UTF-8"*%>

<%

String path = request.getContextPath();

String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";

%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

  <head>

    <base href=*"*<%=basePath%>*"*>

    <title>My JSP 'index.jsp' starting page</title>

    <meta http-equiv=*"pragma"* content=*"no-cache"*>

    <meta http-equiv=*"cache-control"* content=*"no-cache"*>

    <meta http-equiv=*"expires"* content=*"0"*>

    <meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>

    <meta http-equiv=*"description"* content=*"This is my page"*>

    <!--

    <link rel="stylesheet" type="text/css" href="styles.css">

    -->

  </head>

  <body>

    <form name=*"login"* action=*"login.action"* method=*"post"*>

   用户名:

    <input name=*"username"* type=*"text"* />

    密码:

    <input name=*"password"* type=*"password"*>

    <input type=*"submit"* value=*"登录"*/>

    </form>

  </body>

</html>

注意：<body>标签中的部分是我们新添加的。在MyEclipse开发环境下，点击Preview，看看页面效果，登录页面利用Html标签为我们显示了2个输入框、一个登录按钮。



② Success.jsp

<body>

       登录成功！

</body>

③ Failure.jsp

<body>

       登录失败！

 </body>

完成上述工作后，部署一下工程，启动Tomcat[服务器](http://biancheng.dnbcw.info/win2003/)，在浏览器中输入：

http://localhost:8080/SSHLogin/

测试一个主页效果，正常情况下，会显示登录画面。

打开web.xml文件，可以发现：这个配置文件定义了一个欢迎页：

<welcome-file-list>

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

 </welcome-file-list>

因此，当我们只输入工程名，不输入JSP页面文件名的时候，Tomcat服务器会在webapps目录下寻找SSHLogin文件夹。找到后，按照欢迎页面的配置，打开index.jsp文件，显示登录画面。

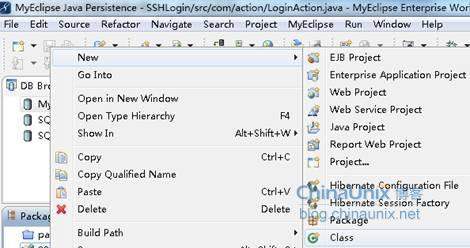
接下来，当用户输入了用户名、密码，单击“登录”按钮后，index.jsp的form表单会向浏览器发送一个如下的Http请求：

http://localhost:8080/SSHLogin/login.action

你会发现，浏览器会显示404错误，这需要我们进一步编写Action类，配置Struts.xml文件。

1. 编写一个Action类：LoginAction，处理超链接请求：login.action

如图，选择工程，创建一个java类。

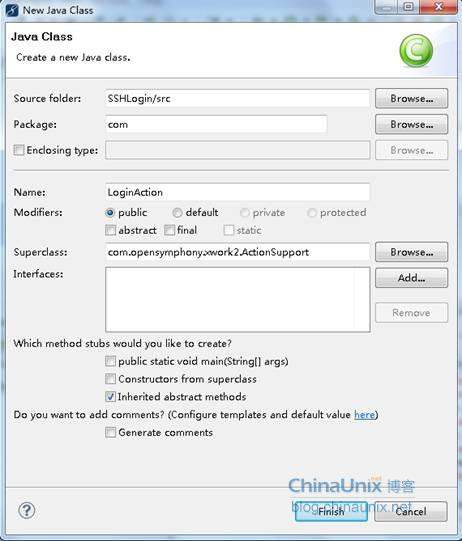


类名：LoginAction

包名：com

基类名：com.opensymphony.xwork2.ActionSupport

注意：很多人喜欢取诸如com.action此类的多级包名，我试过不行，别问为什么。



这里，为了测试Struts2的基本功能，我们先把LoginAction.java的功能写的简单一些，添加如下代码：

//LoginAction.java

**package** com.action;

**import** com.opensymphony.xwork2.ActionSupport;

**public** **class** LoginAction **extends** ActionSupport {

**public** String execute() **throws** Exception {

       System.*out*.println("点击登录执行该方法");

**return** "SUCCESS";

    }

}

3）修改Struts的配置文件

打开struts.xml文件，添加如下配置行：

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

<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN" "http://struts.apache.org/dtds/struts-2.1.dtd">

<struts>

    <package name=*"SSHLogin"* extends=*"struts-default"*>

       <action name=*"login"* class=*"com.action.LoginAction"*>

           <result name=*"SUCCESS"*>/Success.jsp</result>

           <result name=*"FAILURE"*>/Failure.jsp</result>

       </action>

    </package>

</struts>

这里，包名也是我们创建Web工程时的工程名，所有的Action类都存放在这个文件夹下，部署的时候，这个工程被部署在webapps\ SSHLogin目录下。

当用户输入了用户名、密码，单击“登录”按钮后，index.jsp的form表单会向浏览器发送一个如下的Http请求：

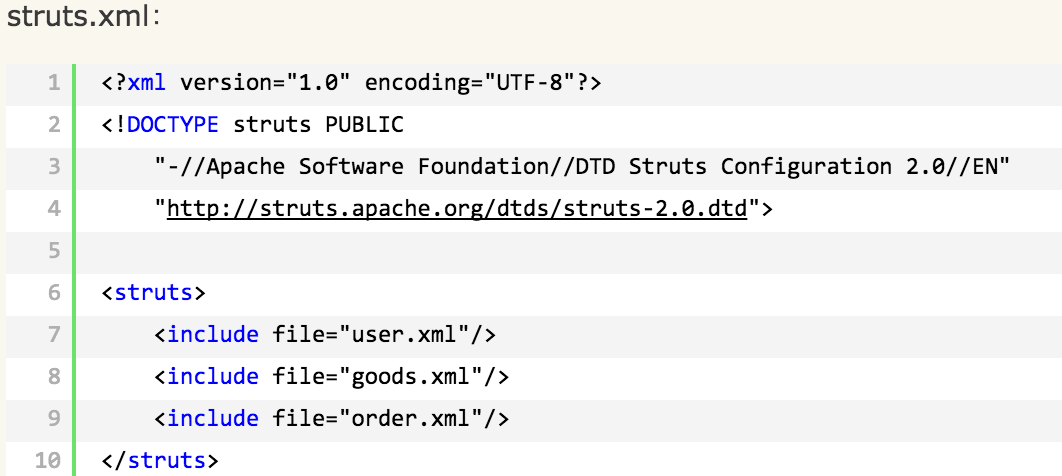
http://localhost:8080/SSHLogin/login.action

由于这个请求是以.action结尾的，因此根据web.xml的配置，这个请求会转发给Struts2控制器处理。根据文件struts.xml的配置，当用户的请求名= login时，控制器会调用com.action.LoginAction.java的execute()方法处理请求。

在LoginAction.java中，execute()方法会打印一个简单的信息，然后返回一个字符串：SUCCESS。这时，根据struts.xml的配置，当execute返回"SUCCESS"时，就会显示：登录成功的页面。因此，正常情况下，当用户点击“登录”，系统会显示登录成功。

重新部署，测试一下看看效果。

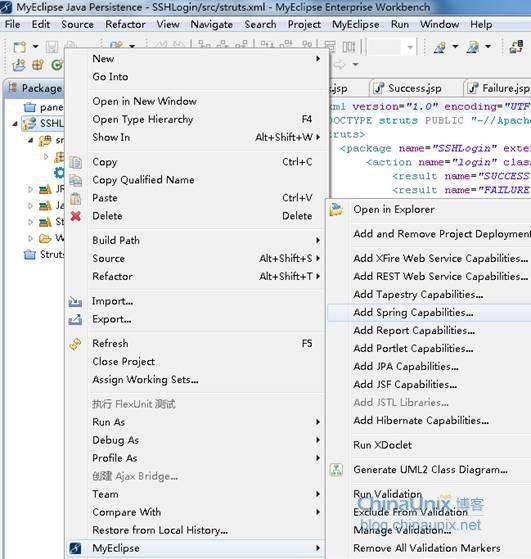
注意：当structs的action较多的时候也可以采用如下的方式，分成多个文件：



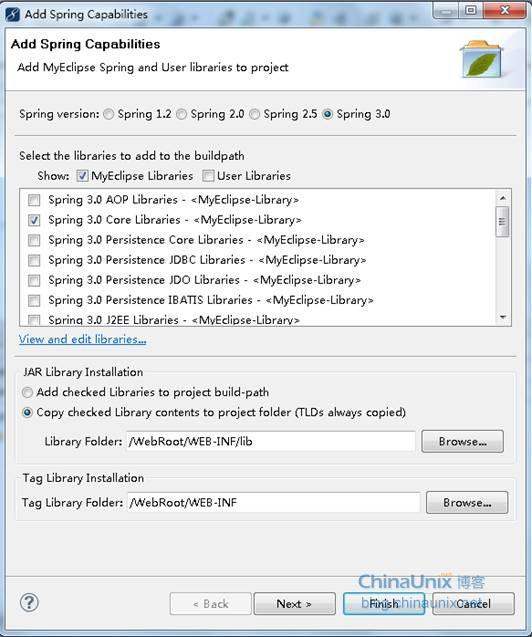


## Struts2整合Spring

1. 选择工程，右击添加Spring支持，如图。



1. 在弹出的对话框，选择：Spring 3.0 Core Lib、Spring 3.0 Web Lib两个库文件。点击“finish”完成Spring库的添加。





1. 打开配置文件Web.xml，添加如下配置行。

<!-- Spring 配置和监听start -->

  <context-param>

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

    <param-value>/WEB-INF/classes/application\*.xml</param-value>

  </context-param>

  <listener>

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

  </listener>

<!-- end -->

**注意**：structs2采用的是过滤器，spring是监听器。

上述“application\*.xml”是Spring配置文件的文件名或者可以采用如下的写法（配置文件为”beans.xml”）：

<context-param>

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

<param-value>classpath:beans.xml</param-value>

</context-param>

<listener>

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

</listener>

1. 修改源程序

在没有添加Spring支持前，用户如果发送

http://localhost:8080/SSHLogin/login.action

根据struts.xml的配置，struts2将调用com包中的LoginAction.java处理用户的请求。添加了Spring框架后，当用户发送请求，struts2框架仍然会调用LoginAction.java的execute()方法处理用户的请求。不同的是，这个时候，不是struts直接调用LoginAction.java的execute()方法，而是由Spring框架以Bean的方式来提供LoginAction.java这个类。为此，需要进行如下修改：

① 打开Spring配置文件applicationContext.xml，添加如下Bean的定义。

<bean id="hwpLoginAction" class="com.LoginAction">

</bean>

② 打开配置文件struts.xml，修改action的class属性。

<package name=*"SSHLogin"* extends=*"struts-default"*>

        <action name=*"login"* class=*"hwpLoginAction"*>

            <result name=*"SUCCESS"*>/Success.jsp</result>

            <result name=*"FAILURE"*>/Failure.jsp</result>

        </action>

</package>

注意：这里struts.xml并没有定义当用户输入：login.action请求时，需要调用的action类。而是定义了一个Bean的ID（Bean ID= hwpLoginAction）。这个Bean的类实现被定义在了Spring的配置文件中。

引入Spring框架后，所有的组件都被定义为Spring框架中的一个Bean，它们在Spring容器中具有唯一的ID。Spring框架定义了每一个Bean的ID及其实现的类（com.LoginAction）。当然，如果这个类有属性，也可以在定义。

1. 部署与测试

将这个工程部署到Tomcat服务器，启动服务器，并输入：

<http://localhost:8080/SSHLogin/>

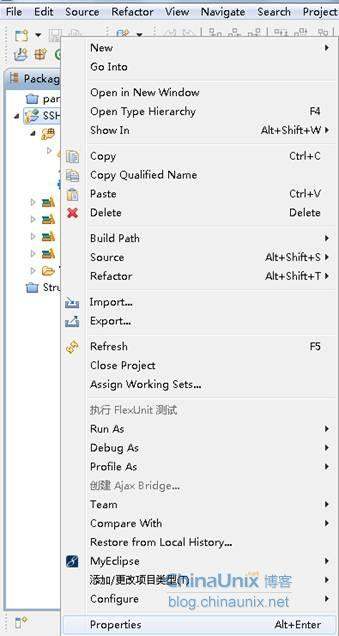


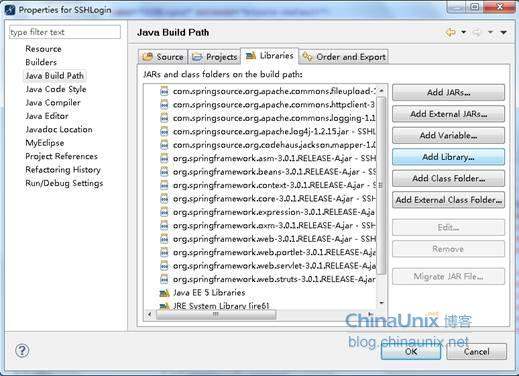
可能会出现404错误，进入MyEclipse，可能看到一些运行错误，例如



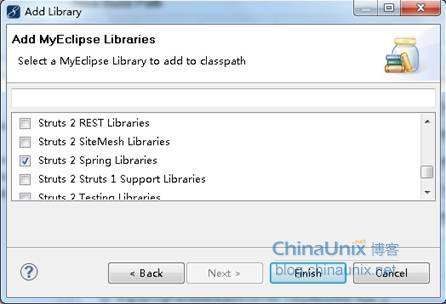
出现这类错误的主要原因是：我们在前面添加Struts支持时没有添加Struts2 Spring Libraries。在Struts2 Spring Libraries库中，有一个很重要的插件struts2-spring-plugin-2.1.8.1.jar，这个插件中提供了spring与struts2整合时所需要的类。

如图，选择工程，右击选择“属性”，为工程添加新库。





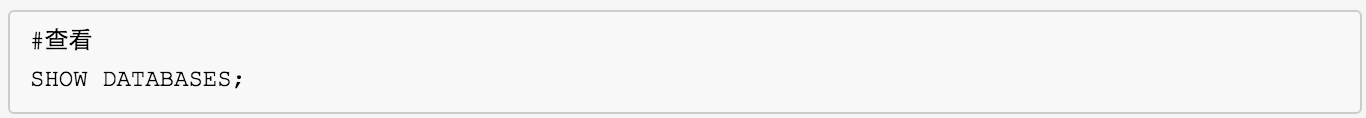
选择Myeclipse Libraries，添加Struts2 Spring Libraries。重新部署、测试工程，即可成功！！！

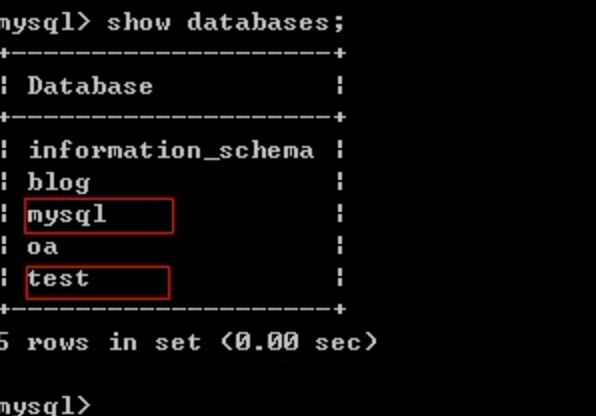


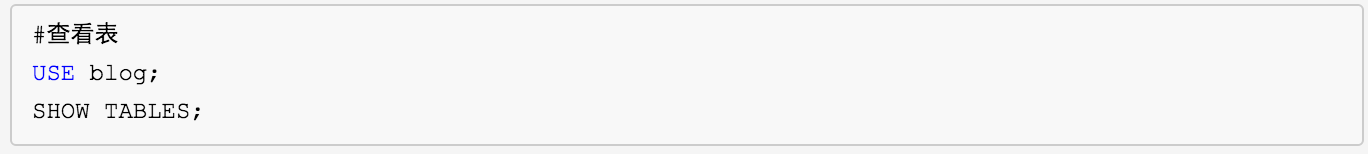
## 继续整合Hibernate

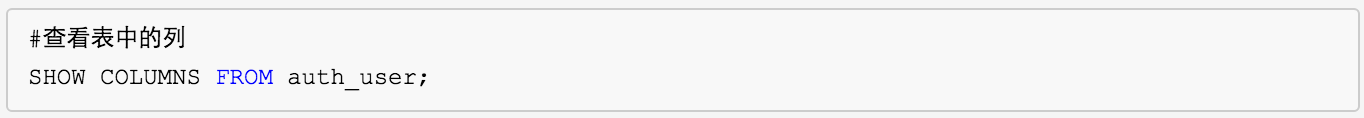
1. 通过Mysql创建数据库

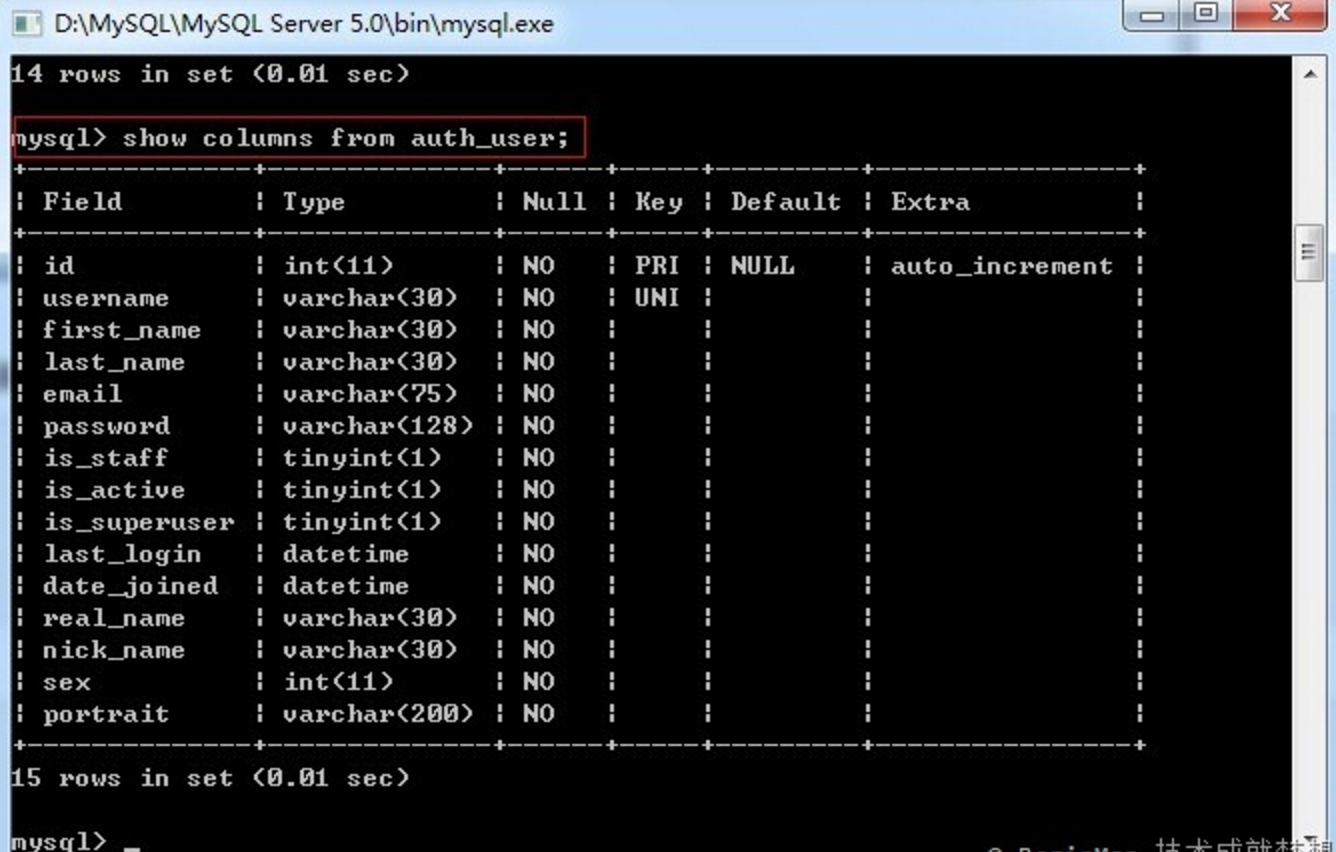
查看数据库



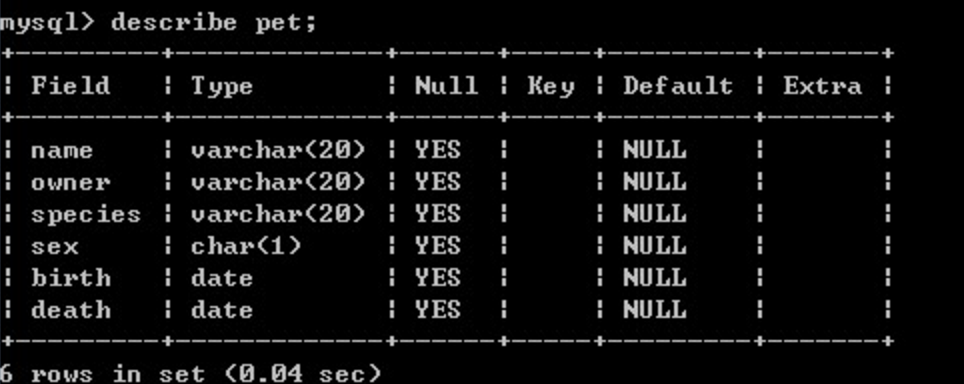




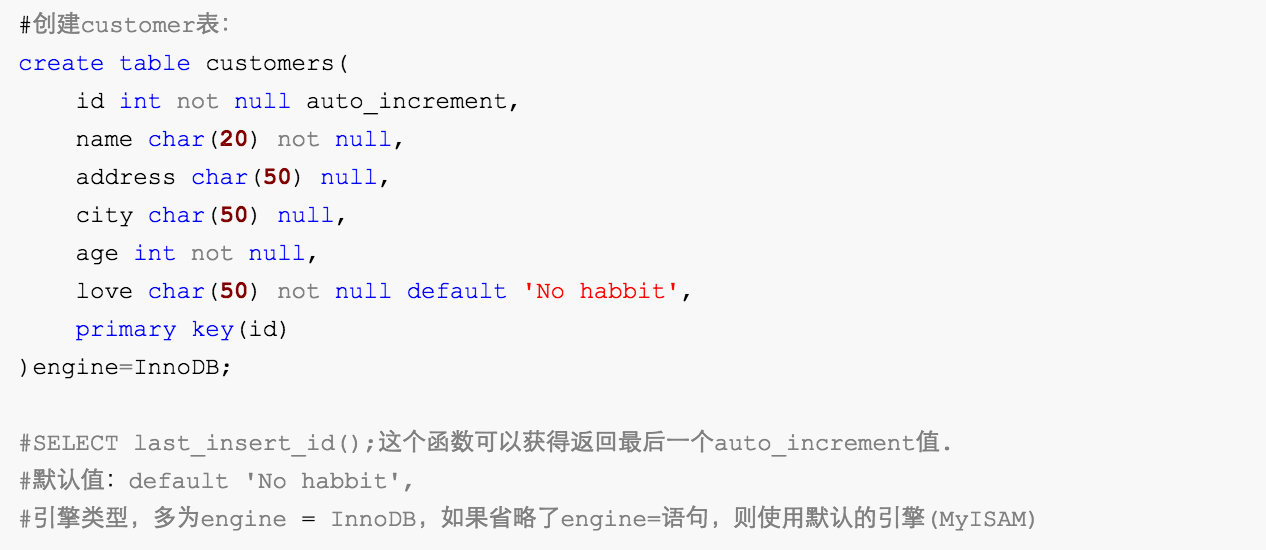








复杂的创建表命令（创建的数据库表的表项根据自己的需要而定）

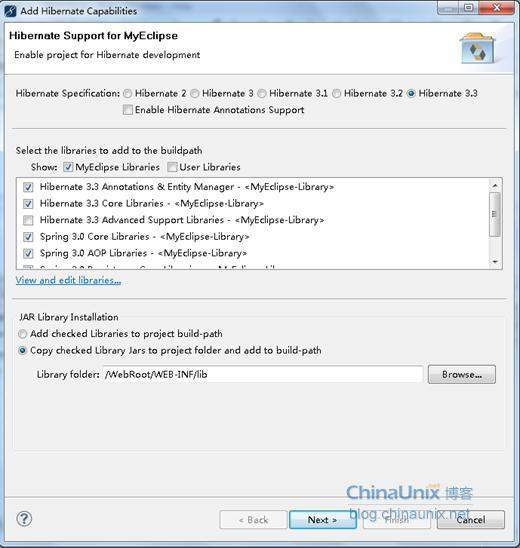


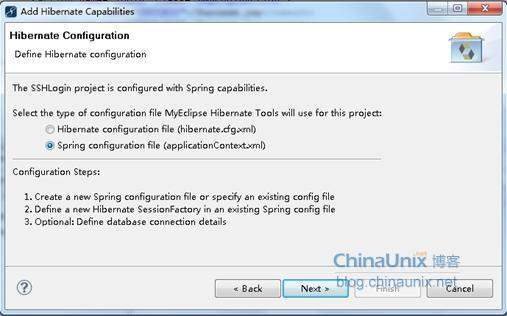
1. 添加Hibernate功能支持

选择工程，右击，添加Hibernate组件。

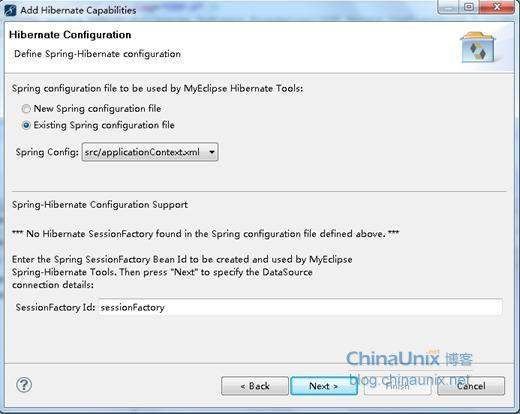


选择所有的库文件（只有一个没选）。

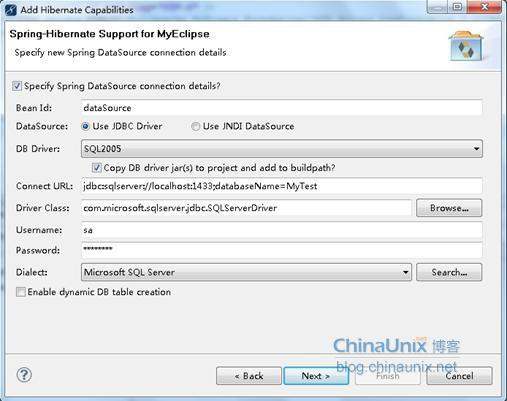


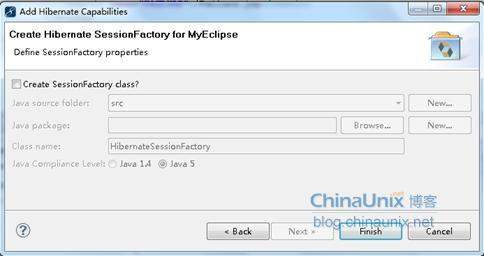


将Hibernate的配置文件合并到Spring的配置文件中。



选择Mysql的JDBC连接（其他数据库类似）

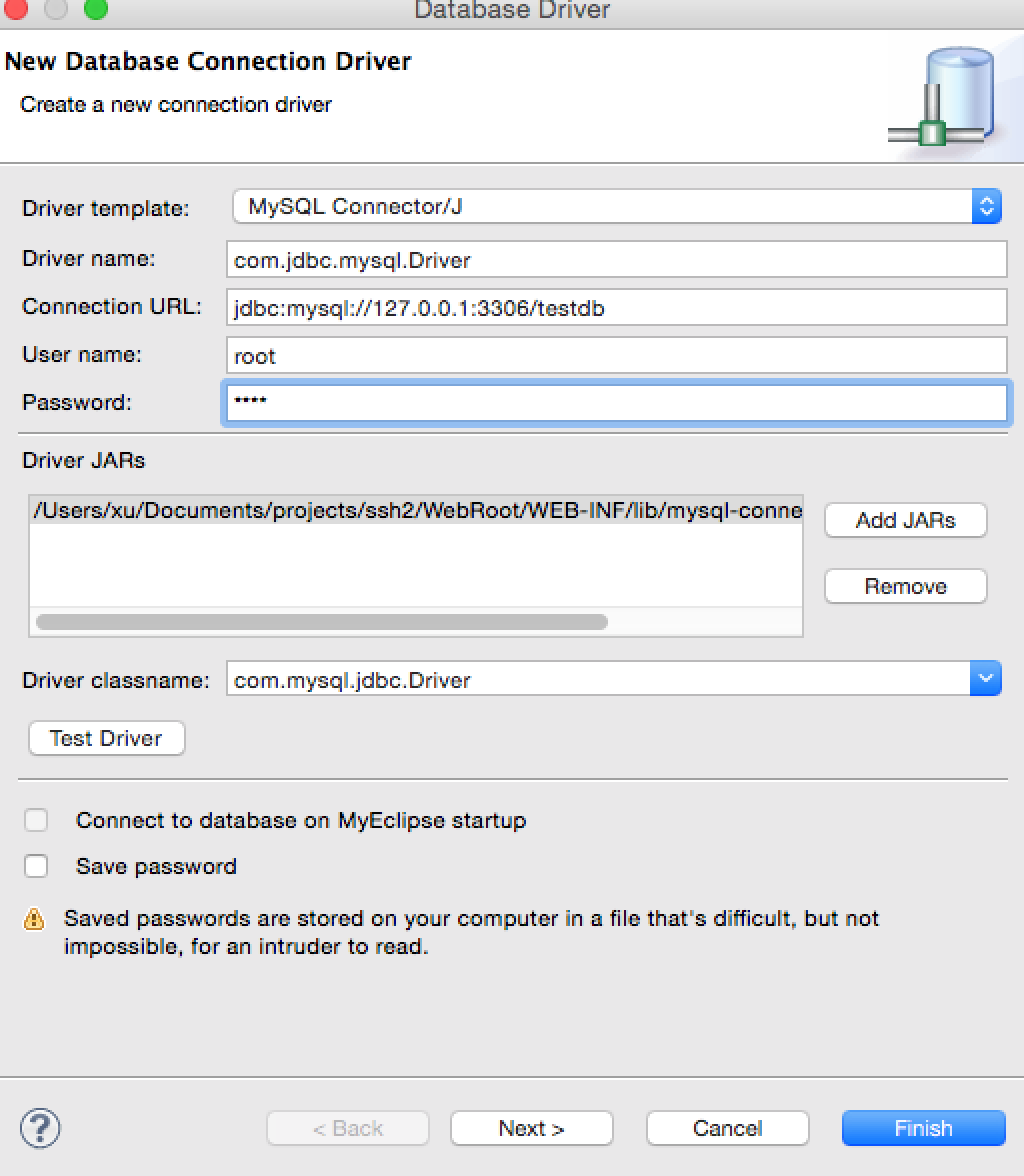


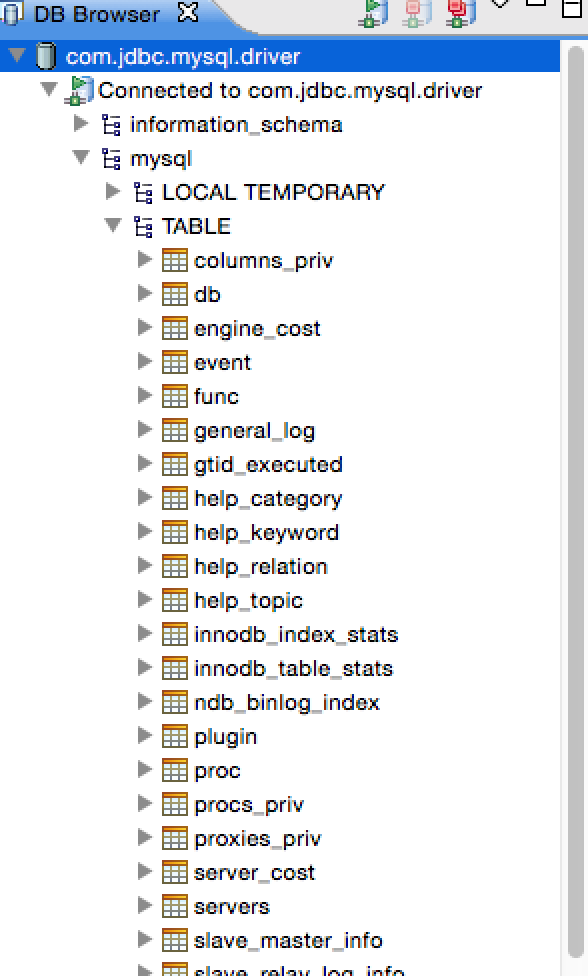


4）完成相关代码的编写与添加

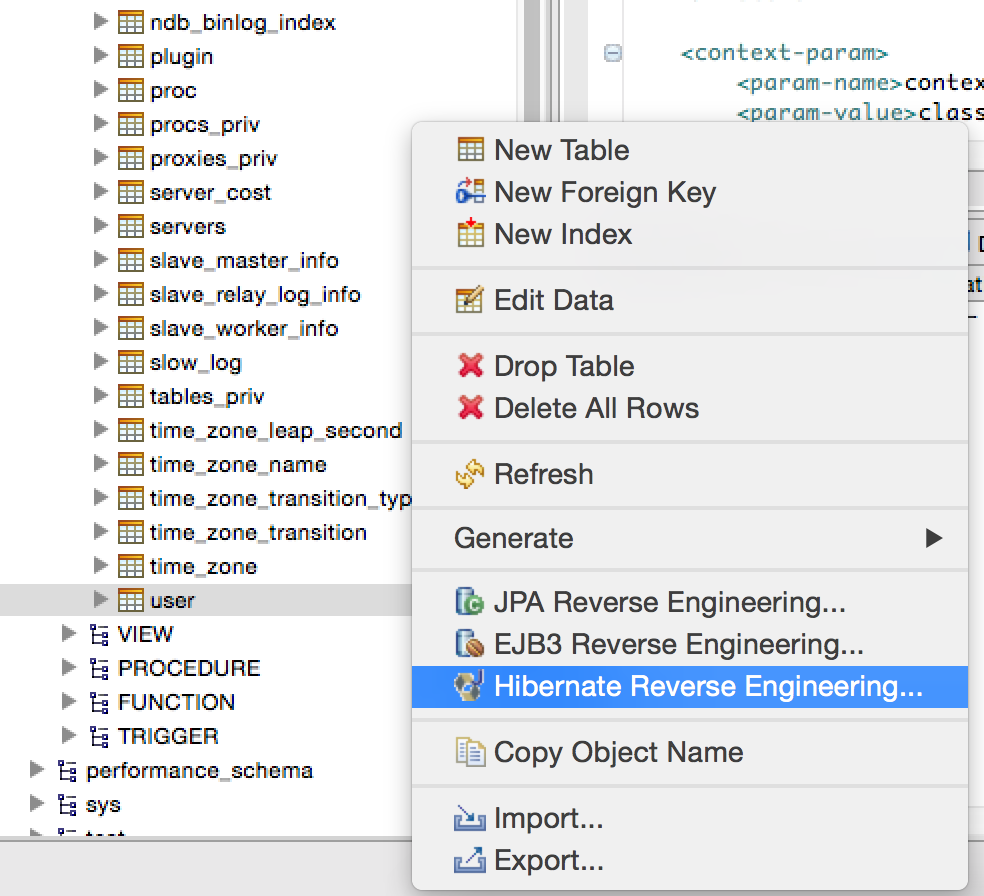
① 逆向工程，生成数据库表的实体类及映射文件

如图，打开MyEclipse Database Explorer，新建数据库链接驱动，打开数据库连接。





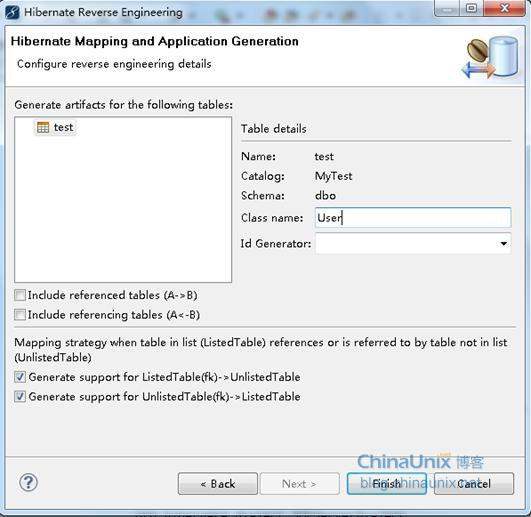
选择数据库test的users表，右击，利用Hibernate逆向工程生成数据库表的实体类及映射文件。



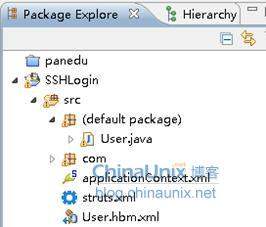
如图，在弹出的窗体下选择源文件夹、Java包名。创建数据库表test的映射文件（.hbm.xml）和数据库表的实体类（POJO）。注意：不要创建DAO类，这个类我们自己生成。



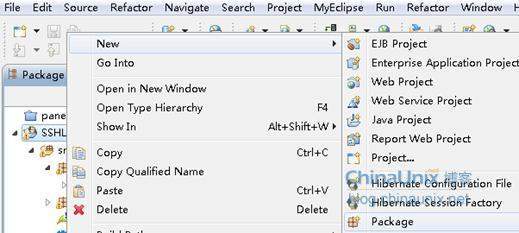
最后，输入数据库表映射的实体类的类名：User.java，点击“finish”即可。

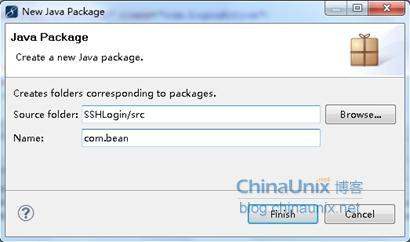


在MyEclipse环境下，展开工程，可以看到Hibernate为我们生成了一个数据库表的映射文件：User.hbm.xml，一个数据库表的实体类：User.java。



如果数据库表的实体类User.java不在我们前面指定的com.bean包内，这时，可以选择工程，创建这个包，并把User.java拖进去，如图。





注意：为什么不能把User类放在缺省的包中？

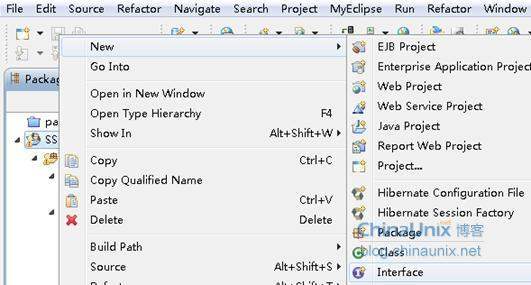
——因为在MyEclipse中，其他的Java类无法导入缺省包中的类。所以，你必须把它放到一个确定的包中。

② 手工创建数据库访问接口DAO及接口的实现类

选择工程，右击创建数据库访问接口UserDAO

接口名：UserDAO

包名：com.dao



采用类似的方法，创建接口UserDAO的实现类。

类名：UserDAOImpl

包名：com.dao.impl



 实现的接口名：UserDAO

③ 业务逻辑层的设计与实现

在com.service包中创建业务逻辑层接口及实现。

接口的创建：

接口名：UserServer

包名：com.service

接口实现类的创建：

类名：UserServerImpl

包名：com.service.impl

实现的接口名：UserServer

参考如下代码，修改数据库接口、接口实现类，业务逻辑层接口及实现类。

//UserDAO.java

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

**package** com.dao;

**import** java.util.List;

**import** com.bean.User;

**public** **interface** UserDAO {

    /\*\*

     \* 根据id查找用户

     \* **@param** id 需要查找的用户id

     \*/

    User get(Integer id);

    /\*\*

     \* 增加用户

     \* **@param** user 需要增加的用户

     \*/

**void** save(User user);

    /\*\*

     \* 修改用户

     \* **@param** user 需要修改的用户

     \*/

**void** update(User user);

    /\*\*

     \* 删除用户

     \* **@param** id 需要删除的用户id

     \*/

**void** delete(Integer id);

    /\*\*

     \* 删除用户

     \* **@param** user 需要删除的用户

     \*/

**void** delete(User user);

    /\*\*

     \* 查询全部用户

     \* **@return** 获得全部用户

     \*/

    List<User> findAll();

    /\*\*

     \* 根据用户名，密码查找用户

     \* **@param** username 查询所需的用户名

     \* **@param** pass 查询所需的密码

     \* **@return** 对应的用户

     \*/

    User findUserByNameAndPass(String username , String password);

}

//UserDAOImpl.java

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

**package** com.dao.impl;

**import** java.util.List;

**import** com.dao.UserDAO;

**import** com.bean.User;

**import** org.springframework.orm.hibernate3.support.HibernateDaoSupport;

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

**public** **void** delete(Integer id) {

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

        getHibernateTemplate().delete(getHibernateTemplate().get(User.**class** , id));

    }

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

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

        getHibernateTemplate().delete(user);

    }

**public** List<User> findAll() {

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

**return** getHibernateTemplate().find("from test");

    }

**public** User findUserByNameAndPass(String username, String password) {

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

        System.*out*.println("验证用户22222222");

        //delete(1);    //从数据库的test表中，成功删除一条记录

        //String[] args = {username , password};

        System.*out*.println("成功123abc");

        List ul = getHibernateTemplate().find("from com.bean.User au where au.username = '"+ username+"' and au.password = '"+ password+"'");

**if** (ul.size() == 1)

        {

            System.*out*.println("OK 123abc");

**return** (User)ul.get(0);

        }

**return** **null**;

    }

**public** User get(Integer id) {

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

**return** (User)getHibernateTemplate().get(User.**class** , id);

    }

**public** **void** save(User user) {

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

        getHibernateTemplate().save(user);

    }

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

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

        getHibernateTemplate().saveOrUpdate(user);

    }

}

注意：findUserByNameAndPass（）方法中的getHibernateTemplate().find，其中的HQL语句中，from后面应该使用实体类名，而不是数据库的表名。因为Hibernate已经实现了数据库表到实体类的映射。所以，对数据库表的访问都转化为对实体类的访问。

//UserServer.java

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

**package** com.service;

**import** com.dao.UserDAO;

**public** **interface** UserServer {

    Integer validLogin(String username , String password);

**public** UserDAO getUserDAO();

**public** **void** setUserDAO(UserDAO userDAO);

}

//UserServerImpl.java

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

**package** com.service.impl;

**import** com.service.UserServer;

**import** com.dao.UserDAO;

**import** com.bean.User;

**public** **class** UserServerImpl **implements** UserServer {

**private** UserDAO userDAO;

**public** UserDAO getUserDAO() {

**return** **this**.userDAO;

    }

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

**this**.userDAO = userDAO;

    }

**public** Integer validLogin(String username, String password){

**try** {

                  User user = userDAO.findUserByNameAndPass(username, password);

**if** (user!= **null**)

                  {

**return** user.getId();

                  }

            }

**catch** (Exception e)

            {

                   System.*out*.println(e.getMessage());

            }

**return** **null**;

    }

}

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

5）编辑、修改LoginAction.java

//LoginAction.java

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

**package** com;

**import** com.opensymphony.xwork2.ActionSupport;

**import** com.service.UserServer;

**public** **class** LoginAction **extends** ActionSupport {

**private** **static** **final** **long** *serialVersionUID* = 1L;

**protected** UserServer mgr;     //该对象采用 Spring 依赖注入

**private** String username;

**private** String password;

**public** String execute() **throws** Exception {

        System.*out*.println("点击登录执行该方法123");

**if**(username.equals("") || username == **null**){

**return** "FAILURE";

        }

        Integer userId = mgr.validLogin(username, password);

**if** (userId != **null**) {

            System.*out*.println("合法用户");

**return** "SUCCESS";

        } **else** {

            addActionError("用户名/密码不匹配");

            System.*out*.println("非法用户");

**return** "FAILURE";

        }

    }

**public** String getPassword() {

**return** password;

    }

**public** String getUsername() {

**return** username;

    }

**public** **void** setPassword(String password) {

**this**.password = password;

    }

**public** **void** setUsername(String username) {

**this**.username = username;

    }

    //Spring依赖注入的对象必须有get、set方法。方法命名规则：get+变量名。为了便于记忆，变量名第一个字母可以大写。

**public** **void** setMgr(UserServer mgr)

    {

**this**.mgr = mgr;

    }

**public** UserServer getmgr()

    {

**return** mgr;

    }

}

6）编辑、修改Spring、Hibernate的配置文件

① 打开Spring的配置文件applicationContext.xml,添加如下配置行

//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:p=*"http://www.springframework.org/schema/p"*

    xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*>

    <bean id=*"dataSource"*

        class=*"org.apache.commons.dbcp.BasicDataSource"*>

        <property name=*"driverClassName"*

            value=*"com.microsoft.sqlserver.jdbc.SQLServerDriver"*>

        </property>

        <property name=*"url"*

            value=*"jdbc:sqlserver://localhost:1433;databaseName=MyTest"*>

        </property>

        <property name=*"username"* value=*"sa"*></property>

        <property name=*"password"* value=*"87601469"*></property>

    </bean>

    <bean id=*"sessionFactory"*

        class=*"org.springframework.orm.hibernate3.LocalSessionFactoryBean"*>

        <property name=*"dataSource"*>

            <ref bean=*"dataSource"*></ref>

        </property>

        <property name=*"hibernateProperties"*>

            <props>

                <prop key=*"hibernate.dialect"*>

                    org.hibernate.dialect.SQLServerDialect

                </prop>

            </props>

        </property>

        <property name=*"mappingResources"*>

            <list>

                <value>./User.hbm.xml</value></list>

        </property></bean>

        <!-- 用户添加的Bean -->

        <bean id=*"hwpLoginAction"* class=*"com.LoginAction"*>

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

        </bean>

        <bean id=*"mgr"* class=*"com.service.impl.UserServerImpl"*>

            <property name=*"userDAO"* ref=*"hwpuserDAO"*></property>

        </bean>

        <bean id=*"hwpuserDAO"* class=*"com.dao.impl.UserDAOImpl"*>

            <property name=*"sessionFactory"*>

                <ref local=*"sessionFactory"* />

            </property>

        </bean>

</beans>

注意：在LoginAction.java中，我们声明了一个UserServer类型的对象mgr，但是没有实例化它（没有new它），实际上，这个对象是通过Spring采用依赖注入完成实例化的。

要想成功地利用Spring依赖注入，需要采取如下4个步骤：

Ⅰ）在Java类中声明这个对象，例如

**public** **class** LoginAction **extends** ActionSupport {

**protected** UserServer mgr;     //该对象采用 Spring 依赖注入

}

Ⅱ）在Java类中为这个对象编写get、set 方法，例如

//Spring依赖注入的对象必须有get、set方法。方法命名规则：get+变量名。为了便于记忆，名第一个字母可以大写。

**public** **void** setMgr(UserServer mgr)

    {

**this**.mgr = mgr;

    }

**public** UserServer getMgr()

    {

**return** mgr;

 }

这里，get、set 方法名原则上是get、set加上变量名，例如：getmgr、stmgr。为了便于记忆，变量名的第一个字母可以大写（其余的不行），例如getMgr、setMgr

Ⅲ）在applicationContext.xml中，声明这个bean，例如

        <bean id=*"mgr"* class=*"com.service.impl.UserServerImpl"*>

            <property name=*"userDAO"* ref=*"hwpuserDAO"*></property>

        </bean>

这里，bean的id可以任意取，但是，在一个applicationContext.xml配置文件中，它应该是唯一的。userDAO是这个bean的一个对象，它也是采用依赖注入方式提供的，就像一个成员变量的声明。

Ⅳ）最后，在applicationContext.xml中，将这个bean注入到其他bean中，例如

<bean id=*"hwpLoginAction"* class=*"com.LoginAction"*>

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

</bean>

本实例中，我们将"mgr注入到LoginAction.java中。

② 打开Hibernate的配置文件User.hbm.xml,填写实体类的完整包名。

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

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

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<!--

    Mapping file autogenerated by MyEclipse Persistence Tools

-->

<hibernate-mapping>

    <class name=*"com.bean.User"* table=*"test"* schema=*"dbo"* catalog=*"MyTest"*>

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

            <column name=*"id"* />

            <generator class=*"assigned"* />

        </id>

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

            <column name=*"username"* length=*"50"* not-null=*"true"* />

        </property>

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

            <column name=*"password"* length=*"50"* />

        </property>

    </class>

</hibernate-mapping>

注意：数据库表的映射文件中，实体类的类名应该包含完整的包名，例如com.bean.User，否则，运行的时候会显示：

nested exception is org.hibernate.MappingException: entity class not found: User

——实体类User没有找到。

所有工作完成后，可以输入：

http://localhost:8080/SSHLogin/

## SSH集成成功案例参考

SSH2（详细源代码在群空间中）

