# 实验一 JSP+Servlet+Javabean+jdbc-基于MVC模式的数据库的应用

**实验目的：**

**掌握MVC模式的基本原理**

掌握JSP+Servlet+JavaBean架构连接、查询、删除数据库的方法；

**实验要求**：

采用JSP+Servlet+JavaBean架构，设计通过下列JSP页面访问数据库(如表5-1)，具体要求如下：

1．添加用户页面(AddUser.jsp),如图7-1所示；

2. 删除用户页面(Delete.jsp), 如图7-2所示；

3．查找和修改用户页面(SearchAndModify.jsp), 如图7-3所示。



图7-1 添加用户页面

UserVO

DBUtil

AddUser.jsp

DB

UserDAO



图7-2 删除用户页面

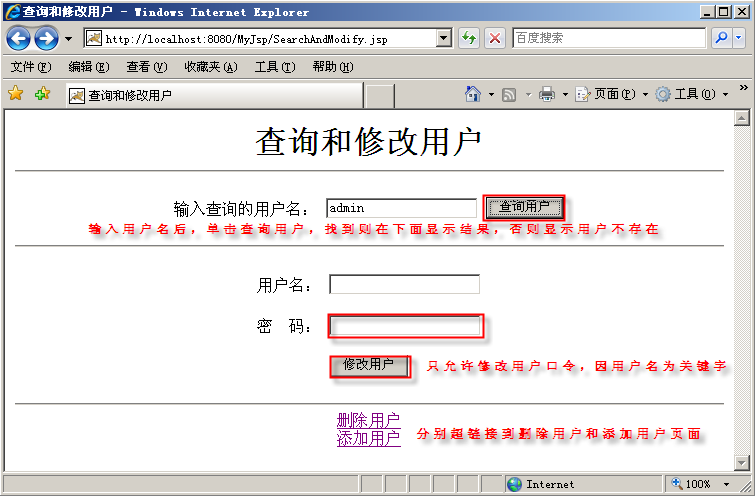


图7-3 查询和修改用户页面

**实验步骤：**

1．准备需要访问的数据库

下载并安装MySQL5.0

Copy MySQL5.0驱动mysql-connector-java-5.0.8-bin.jar到%TOMCAT%\lib文件夹下创建数据库及表：

数据库的设计如下表：

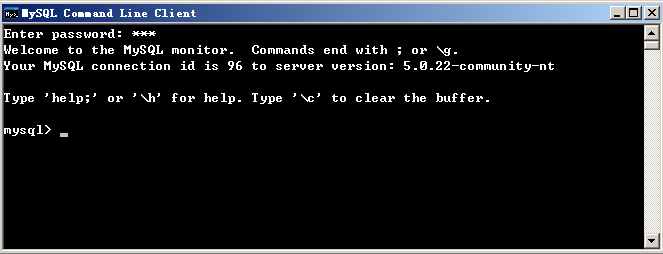
数据库采用MySQL5.0，数据库名：db，表名：user（用户表）

表5-1 数据库的用户表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 长度 | 主键 | 允许空 | 默认值 | 说明 |
| 1 | username | varchar | 40 | 是 | 否 |  | 用户名 |
| 2 | password | varchar | 40 |  | 是 |  | 口令 |

创建过程如下：(可选择使用MySQL的数据库管理和开发工具Navicat 和 MySQL-Front等)

Step1:进入MySQL程序的MySQL Command Line Client界面，如下图所示



Step2:分别执行下列命令：//支持中文

① 创建数据库：

CREATE DATABASE `db`

CHARACTER SET 'utf8'

COLLATE 'utf8\_general\_ci';

USE DB;

② 建表:

CREATE TABLE `user` (

`username` varchar(20) NOT NULL PRIMARY KEY,

`password` varchar(20)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

开发过程如下：

Step1: 创建值JavaBean: User.java,内容如下：

package valueBean;

public class UserVO {

private String username;//用户名

private String password;//口令

public UserVO()

{

this.username="";

this.password="";

}

public void setUsername(String username) {

this.username = username;

}

public String getUsername() {

return username;

}

public void setPassword(String password) {

this.password = password;

}

public String getPassword() {

return password;

}

}

Step2: 创建连接数据库类DBUtil

package toolBean.db;

import java.sql.\*;

import valueBean.User;

public class DBUtil{

private static Connection conn=null;

public static Connection getConnection()

{

String driverClass="com.mysql.jdbc.Driver";

String url=//与创建数据库的代码结合，保证访问数据库时不出现乱码

"jdbc:mysql://localhost:3306/db?useUnicode=true&characterEncoding=UTF-8";

String username = "root";

String password = "123";

try {

Class.forName(driverClass);// 加载数据库驱动

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

if (conn==null)

try {

conn = DriverManager.getConnection(url, username, password);//建立连接

} catch (SQLException e) {

e.printStackTrace();

System.out.print("数据库连接失败！");

}

return conn;

}

Step3: 创建访问数据库的类—UserDAO.java,内容如下：

package toolBean.db;

import java.sql.\*;

import valueBean.User;

public class UserDAO {

private static Connection conn=null;

private Statement stmt=null;

//构造函数中完成对数据库进行初始化

public UserDAO() {

//创建连接

conn=DBUtil.getConnection();

try {

//创建Statement

stmt=conn.createStatement();

} catch (SQLException e) {

e.printStackTrace();

}

}

//将对象user添加到表中

public void Add(VOUser user) throws SQLException

{

String sql="INSERT INTO user(username,password) VALUES('"

+user.getUsername()+"','"+user.getPassword()+"')";

stmt.executeUpdate(sql);

//此处可以使用预定义语句PreparedStatement

}

//根据用户名（关键字）从数据库删除相应的记录

public void Delete(String username) throws SQLException

{

String sql="Delete From user Where username='"+username+"'";

stmt.executeUpdate(sql);

}

//修改数据库中的user

public void Update(User user) throws SQLException

{

String sql="UPDATE user set password='"

+user.getPassword()+"' where username='"+user.getUsername()+"'";

stmt.executeUpdate(sql);

}

//根据用户名称（关键字）从数据库中查找记录，并将找到的记录写入对象user中返回

public User searchByUsername(String username) throws SQLException

{

User user=new User();

String sql="select \* from user where username='"+username+"'";

ResultSet rs=stmt.executeQuery(sql);

while(rs.next())

{

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

if(name.equals(username))

{

//将从数据库查找得到的记录存入对象user中

user.setUsername(name);

user.setPassword(rs.getString("password"));

if(rs!=null)

rs.close();

return user;

}

}

return null;//查找不到返回null

}

//根据用户判断该记录是否存在

public boolean exists(String username)

{

boolean rtn=false;

try {

if(searchByUsername(username)!=null)

rtn=true;

} catch (SQLException e) {

e.printStackTrace();

}

return rtn;

}

//关闭与访问数据库有关连接

public void finalize()

{

try {

if(stmt!=null)

stmt.close();

if(conn!=null)

conn.close();

}

catch (SQLException e) {

e.printStackTrace();

}

}

}

Step4: 创建JSP文件以及相应的Servlet

创建 添加用户页面—AddUser.jsp,代码如下：

<%@ page contentType="text/html; charset=gb2312" language="java" import="java.sql.\*" errorPage="" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />

<title>添加用户</title>

</head>

<body>

<form method="post" action="AddUser">

<div align="center"><font face="宋体" size="6"><strong>

添加用户</strong></font><br/><hr/>

用户名： <input name="username" type="text"/>

<br/>

<br/>

密&nbsp;&nbsp;码：

<input name="password" type="text"/>

<br />

<br />

<input type="submit" value="添加" />

<input type="reset" value="取消" />

</div>

</form>

</body>

</html>

相应的添加用户Servlet(AddUserServlet.java),代码如下：

package servlet.user;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.SQLException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import toolBean.db.AccessUserFromDB;

import valueBean.User;

public class AddUserServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//返回上页

String goBack="<br><a href='javascript:window.history.go(-1);'>返回上页</a>";

response.setContentType("text/html;charset=gb2312");

PrintWriter out = response.getWriter();

String username=request.getParameter("username");

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

{

out.print("用户名不能为空！");

out.print(goBack);

}

String password=request.getParameter("password");

User user=new User();

user.setUsername(username);

user.setPassword(password); //注入属性

UserDAO userDAO=new UserDAO();

if(userDAO.exists(user.getUsername()))

{

out.print("用户名已存在！");

out.print(goBack);

return;

}

try {

userDAO.Add(user);

} catch (SQLException e) {

e.printStackTrace();

out.print("添加失败:"+e.getLocalizedMessage());

out.print(goBack);

return;

}

out.print("添加成功！");

out.print(goBack);

out.close();

}

}

创建删除用户页面--Delete.jsp,代码如下：

<%@ page contentType="text/html; charset=gb2312" language="java" import="java.sql.\*" errorPage="" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />

<title>删除用户</title>

</head>

<body>

<form method="post" action="DeleteUser">

<div align="center"><font face="宋体" size="6"><strong>

删除用户</strong></font><br/>

<hr/>

<p>

</p>

<p>输入删除的用户名 <input name="username" type="text"/><br/><br/>

<input type="submit" name="delete" value="删除用户" />

<br />

<br />

</p>

</div>

</form>

</body>

</html>

相应的删除用户Servlet (DeleteUserServlet.java)的代码，如下：

package servlet.user;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.SQLException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import toolBean.db.AccessUserFromDB;

import valueBean.User;

public class DeleteUserServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//返回上页

String goBack="<br><a href='javascript:window.history.go(-1);'>返回上页</a>";

response.setContentType("text/html;charset=gb2312");

PrintWriter out = response.getWriter();

String username=request.getParameter("username");

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

{

out.print("用户名不能为空！");

out.print(goBack);

}

AccessUserFromDB userDB=new AccessUserFromDB();

if(!userDB.exists(username))

{

out.print("用户名不存在！");

out.print(goBack);

return;

}

try {

userDB.Delete(username);

} catch (SQLException e) {

e.printStackTrace();

out.print("删除失败:"+e.getLocalizedMessage());

out.print(goBack);

return;

}

out.print("删除成功！");

out.print(goBack);

out.close();

}

}

创建查找和修改页面(SearchAndModify.jsp),代码如下：

<%@ page contentType="text/html; charset=gb2312" language="java" import="java.sql.\*" errorPage="" %>

<%@page import="toolBean.db.AccessUserFromDB"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<%

String username=request.getParameter("queryName");

valueBean.User user=new valueBean.User();

if(username!=null&&username!="")

{

AccessUserFromDB dbUser=new AccessUserFromDB();

valueBean.User temp=dbUser.searchByUsername(username);

if(temp==null)

{

out.print("用户名不存在！");

}

else

user=temp;//防止user为null，导致下文的user.getUsername()产生异常

}

%>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />

<title>查询和修改用户</title>

</head>

<body>

<div align="center">

<form name="query" method="post" action="">

<font face="宋体" size="6"><strong>查询和修改用户</strong></font><br/><hr/><br/>

输入查询的用户名： <input name="queryName" type="text"/> <input type="submit" name="query" value="查询用户" /><br/><br/><hr/><br/>

</form>

<form name="modify" method="post" action="ModifyUser">

用户名：

<input name="modfyName" type="text" value="<%=user.getUsername()%>" readonly="readonly" />

<br/><br/>

密&nbsp;&nbsp;码：

<input name="password" type="text" value="<%=user.getPassword()%>"/>

<br /><br />

<input type="submit" name="modify" value="修改用户" />

</form>

<hr/>

<a href="DeleteUser.jsp">删除用户</a><br/>

<a href="AddUser.jsp">添加用户</a><br/>

</div>

</body>

</html>

修改用户的Servlet (ModifyUser.java),代码如下：

package servlet.user;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.SQLException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import valueBean.User;

public class ModifyUserServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//返回上页

String goBack="<br><a href='javascript:window.history.go(-1);'>返回上页</a>";

response.setContentType("text/html;charset=gb2312");

PrintWriter out = response.getWriter();

String name=request.getParameter("modify");

if(name!=null) //单击了‘修改用户’

{

String username=request.getParameter("modfyName");

String password=request.getParameter("password");

User user=new User();

user.setUsername(username);

user.setPassword(password);

toolBean.db.AccessUserFromDB dbUser=new toolBean.db.AccessUserFromDB();

try {

dbUser.Update(user);

} catch (SQLException e) {

e.printStackTrace();

out.print("修改失败："+e.getLocalizedMessage());

out.print(goBack);

return;

}

}

out.print("修改成功！");

out.print(goBack);

out.close();

}

}

# 实验二 JSF+JPA实现用户注册

**实验目的：**

掌握JSF开发界面和控制的方法

掌握JPA连接和更新数据库的方法；

**实验要求**：

采用JSF+JPA架构，设计通过下列JSP页面访问数据库(如表5-1)，具体要求如下：

一、JSF部分实验

1．用户注册页面(AddUser.jsp),如图7-1所示；

如图8-1所示。



图8-1 添加用户页面

VOUser

DB

DAOUser

AddUser.jsp

图8-2 软件架构图

**实验步骤：**

1．准备需要访问的数据库

下载并安装MySQL5.0

Copy MySQL5.0驱动mysql-connector-java-5.0.8-bin.jar到%TOMCAT%\lib文件夹下创建数据库及表：

数据库的设计如下表：

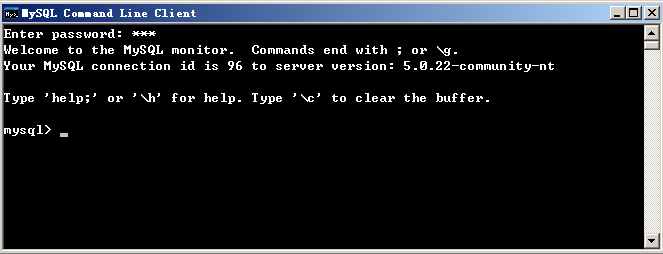
数据库采用MySQL5.0，数据库名：mydb，表名：userinfo（用户表）

表8-1 数据库的用户表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 长度 | 主键 | 允许空 | 默认值 | 说明 |
| 1 | username | varchar | 40 | 是 | 否 |  | 用户名 |
| 2 | password | varchar | 40 |  | 是 |  | 口令 |

创建过程如下：(可选择使用MySQL的数据库管理和开发工具Navicat 和 MySQL-Front等)

Step1:进入MySQL程序的MySQL Command Line Client界面，如下图所示



Step2:分别执行下列命令：//支持中文

① 创建数据库：

CREATE DATABASE `db`

CHARACTER SET 'utf8'

COLLATE 'utf8\_general\_ci';

USE DB;

② 建表:

CREATE TABLE `userinfo` (

`username` varchar(20) NOT NULL PRIMARY KEY,

`password` varchar(20)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

开发过程如下：

Step1: 创建值JavaBean: User.java,内容如下：

package valueBean;

public class UserVO {

private String username;//用户名

private String password;//口令

public UserVO()

{

this.username="";

this.password="";

}

public void setUsername(String username) {

this.username = username;

}

public String getUsername() {

return username;

}

public void setPassword(String password) {

this.password = password;

}

public String getPassword() {

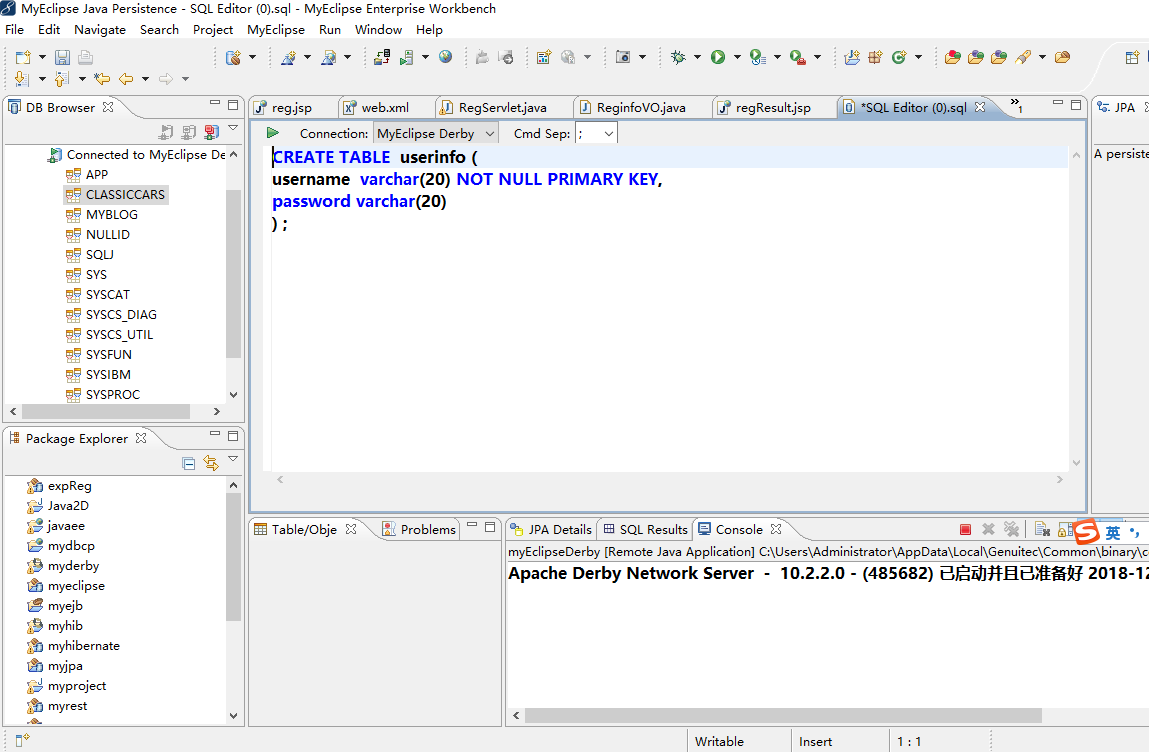
return password;

}

}

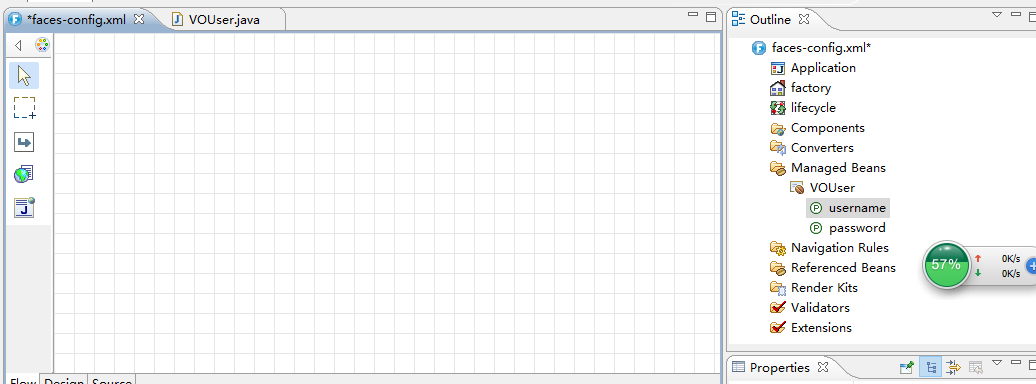
注意：

1、以上数据库也可以使用MyEclipse自带的数据库系统derby，下面的代码以连接derby为例，如果需要连接MySQL，连接字符串和驱动参照实验七。



2、该类也可以通过Myeclipse中Databse Explore透视图或者Java Persistence 透视图中的JPA Reverse Engineering自动导出，点开数据库连接，选择创建的数据库以及表UserInfo，点击右键，选择JPA Reverse Engineering按照向导产生EntityBean和DAO 。

在web-inf下面打开face-config.xml文件，使用右边ManagedBeans，点击右键New Managed Bean，创建UserBean，命名为userBean，关联上述User类，选择该类点击右键New Property，将上述两个属性username和password也添加进去。



配置好的face-config.xml的内容为：

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

<faces-config version=*"1.2"* xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xi=*"http://www.w3.org/2001/XInclude"*

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-facesconfig\_1\_2.xsd"*>

<managed-bean>

<managed-bean-name>userVO</managed-bean-name>

<managed-bean-class>managedBean.userVO</managed-bean-class>

<managed-bean-scope>request</managed-bean-scope>

<managed-property>

<property-name>username</property-name>

<property-class>String</property-class>

<value/>

</managed-property>

<managed-property>

<property-name>password</property-name>

<property-class>String</property-class>

<value/>

</managed-property>

</managed-bean>

</faces-config>

Step2: 创建AddUserBean

在web-inf下面打开face-config.xml文件，使用右边ManagedBeans，点击右键New Managed Bean，创建AddUserBean

类似的将User属性设置为受管的属性。

AddUserBean的配置文件是：

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

<faces-config version=*"1.2"* xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xi=*"http://www.w3.org/2001/XInclude"*

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-facesconfig\_1\_2.xsd"*>

<managed-bean>

<managed-bean-name>UserVO</managed-bean-name>

<managed-bean-class>managedBean.UserVO</managed-bean-class>

<managed-bean-scope>request</managed-bean-scope>

<managed-property>

<property-name>userVO</property-name>

<property-class>managedBean.User</property-class>

<value>#{user}</value>

</managed-property>

<managed-property>

<property-name>userDAO</property-name>

<property-class>managedBean.UserDAO</property-class>

<value>#{userDAO}</value>

</managed-property>

</managed-bean>

</faces-config>

如果要复用实验一中的VOUser,此处创建的AddUserBean为控制类，则代码如下，这里的AddUserBean类相当于实验一的控制类AddUserServlet，也相当于业务控制类BO（本来实验一的体系结构中最好也添加BO,bussiness object）。

**package** control;

**public** **class** AddUserBean {

**private** vo.UserVO userVO;

**private** dao.UserDAO userDAO;

**public** AddUserBean {

}

**public** vo.UserDAO getUserVO() {

**return** userVO;

}

**public** **void** setUserVO(vo.UserVO userVO) {

**this**.userVO = userVO;

}

**public** dao.UserDAO getUserDAO() {

**return** userDAO;

}

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

**this**.userDAO = userDAO;

}

public String addUser(){

if(userDAO.exists(user.getUsername()))

{

return “user Exists”;

}

try {

userDAO.Add(user);

} catch (SQLException e) {

return “sqlerror”;

}

}

}

UserDAO和UserVO参见实验一中的类，但需要在face-config.xml中配置，也就是变成受管bean。

配置文件如下：

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

<faces-config version=*"1.2"* xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xi=*"http://www.w3.org/2001/XInclude"*

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-facesconfig\_1\_2.xsd"*>

<managed-bean>

<managed-bean-name>userDAO</managed-bean-name>

<managed-bean-class>dao.UserDAO</managed-bean-class>

<managed-bean-scope>request</managed-bean-scope>

</managed-bean>

<managed-bean>

</faces-config>

Step3: 创建AddUser.jsp文件。这里要使用JSF定制的标签<f:>

创建 添加用户页面—AddUser.jsp,代码如下：

<%@ page contentType="text/html; charset=gb2312" language="java" import="java.sql.\*" errorPage="" %>

<body>

<form method="post" action="ProcessAddUser.jsp">

<div align="center"><font face="宋体" size="6"><strong>

添加用户</strong></font><br/><hr/>

用户名： <input name="username" type="text"/>

<br/>

<br/>

密&nbsp;&nbsp;码：

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

<br />

<br />

<input type="submit" value="添加" />

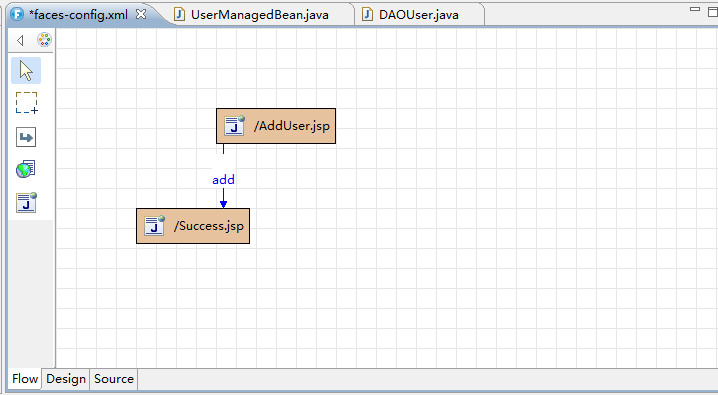
<input type="reset" value="取消" />

</div>

</form>

</body>

</html>

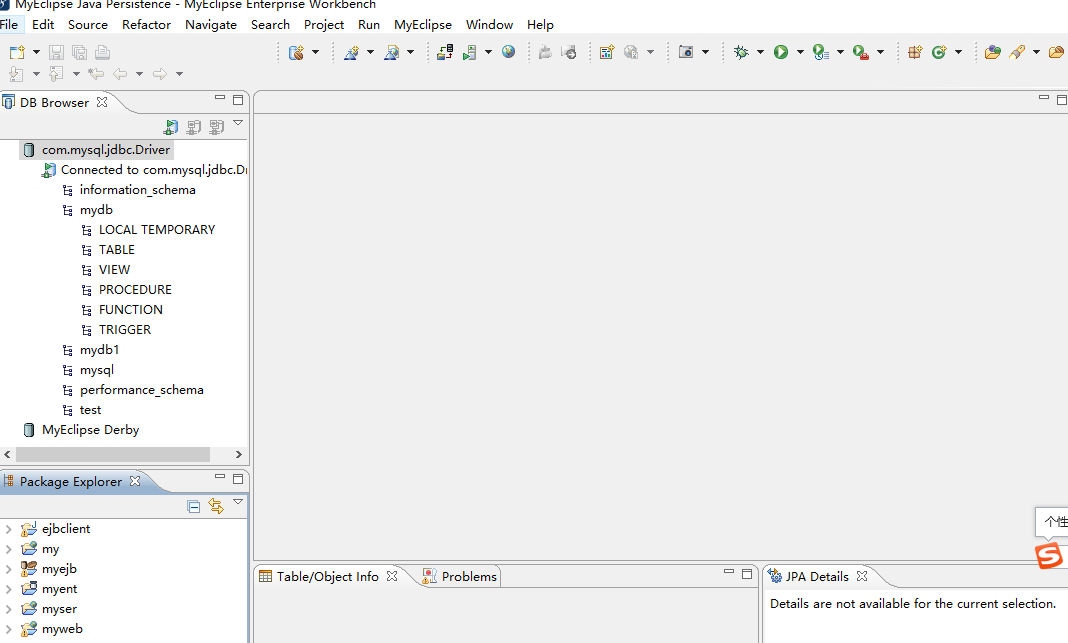


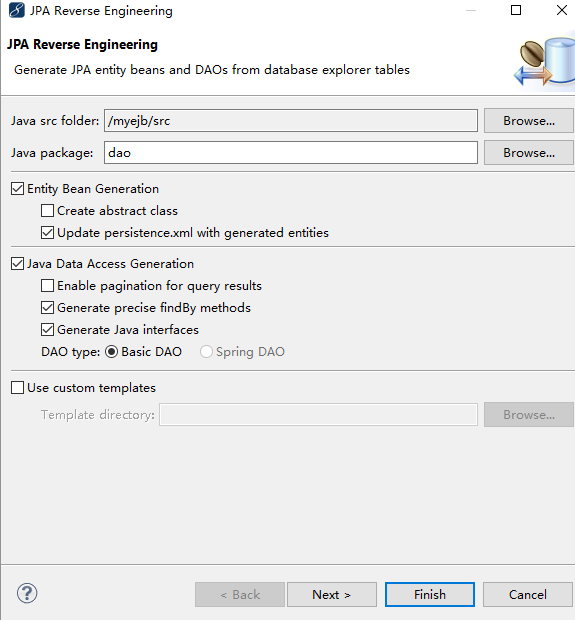
二、JPA实验

上述使用的VO和DAOO都是复用实验一的VO和DAO，没有用到JPA，如果要使用JPA，只要将上述的VO类和DAO类替换为下面的VO和DAO即可。

VO和DAO都可以通过JPA Reverse Engineering来生成。

(注意：在给工程加入jpa capality时，会设置连接数据库的相关信息，并保存在src\meta-inf\下的persistence.xml中。





直接点击finish即可。

产生的UserinfoVO为：

package vo;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

/\*\*

\* Userinfo entity. @author MyEclipse Persistence Tools

\*/

@Entity

@Table(name = "userinfo", catalog = "mydb")

public class UserinfoVO implements java.io.Serializable {

// Fields

private String username;

private String password;

// Constructors

/\*\* default constructor \*/

public UserinfoVO() {

}

/\*\* full constructor \*/

public UserinfoVO(String username, String password) {

this.username = username;

this.password = password;

}

// Property accessors

@Id

@Column(name = "username", unique = true, nullable = false, length = 10)

public String getUsername() {

return this.username;

}

public void setUsername(String username) {

this.username = username;

}

@Column(name = "password", nullable = false, length = 10)

public String getPassword() {

return this.password;

}

public void setPassword(String password) {

this.password = password;

}

}

产生的DAO类为：

package dao;

import java.util.List;

import java.util.logging.Level;

import javax.persistence.EntityManager;

import javax.persistence.Query;

import vo.UserinfoVO;

/\*\*

\* A data access object (DAO) providing persistence and search support for

\* Userinfo entities. Transaction control of the save(), update() and delete()

\* operations must be handled externally by senders of these methods or must be

\* manually added to each of these methods for data to be persisted to the JPA

\* datastore.

\*

\* @see vo.UserinfoVO

\* @author MyEclipse Persistence Tools

\*/

public class UserinfoDAO implements IUserinfoDAO {

// property constants

public static final String PASSWORD = "password";

private EntityManager getEntityManager() {

return EntityManagerHelper.getEntityManager();

}

/\*\*

\* Perform an initial save of a previously unsaved Userinfo entity. All

\* subsequent persist actions of this entity should use the #update()

\* method. This operation must be performed within the a database

\* transaction context for the entity's data to be permanently saved to the

\* persistence store, i.e., database. This method uses the

\* {@link javax.persistence.EntityManager#persist(Object)

\* EntityManager#persist} operation.

\*

\* <pre>

\* EntityManagerHelper.beginTransaction();

\* UserinfoDAO.save(entity);

\* EntityManagerHelper.commit();

\* </pre>

\*

\* @param entity

\* Userinfo entity to persist

\* @throws RuntimeException

\* when the operation fails

\*/

public void save(UserinfoVO entity) {

EntityManagerHelper.log("saving Userinfo instance", Level.INFO, null);

try {

getEntityManager().persist(entity);

EntityManagerHelper.log("save successful", Level.INFO, null);

} catch (RuntimeException re) {

EntityManagerHelper.log("save failed", Level.SEVERE, re);

throw re;

}

}

/\*\*

\* Delete a persistent Userinfo entity. This operation must be performed

\* within the a database transaction context for the entity's data to be

\* permanently deleted from the persistence store, i.e., database. This

\* method uses the {@link javax.persistence.EntityManager#remove(Object)

\* EntityManager#delete} operation.

\*

\* <pre>

\* EntityManagerHelper.beginTransaction();

\* UserinfoDAO.delete(entity);

\* EntityManagerHelper.commit();

\* entity = null;

\* </pre>

\*

\* @param entity

\* Userinfo entity to delete

\* @throws RuntimeException

\* when the operation fails

\*/

public void delete(UserinfoVO entity) {

EntityManagerHelper.log("deleting Userinfo instance", Level.INFO, null);

try {

entity = getEntityManager().getReference(UserinfoVO.class,

entity.getUsername());

getEntityManager().remove(entity);

EntityManagerHelper.log("delete successful", Level.INFO, null);

} catch (RuntimeException re) {

EntityManagerHelper.log("delete failed", Level.SEVERE, re);

throw re;

}

}

/\*\*

\* Persist a previously saved Userinfo entity and return it or a copy of it

\* to the sender. A copy of the Userinfo entity parameter is returned when

\* the JPA persistence mechanism has not previously been tracking the

\* updated entity. This operation must be performed within the a database

\* transaction context for the entity's data to be permanently saved to the

\* persistence store, i.e., database. This method uses the

\* {@link javax.persistence.EntityManager#merge(Object) EntityManager#merge}

\* operation.

\*

\* <pre>

\* EntityManagerHelper.beginTransaction();

\* entity = UserinfoDAO.update(entity);

\* EntityManagerHelper.commit();

\* </pre>

\*

\* @param entity

\* Userinfo entity to update

\* @return Userinfo the persisted Userinfo entity instance, may not be the

\* same

\* @throws RuntimeException

\* if the operation fails

\*/

public UserinfoVO update(UserinfoVO entity) {

EntityManagerHelper.log("updating Userinfo instance", Level.INFO, null);

try {

UserinfoVO result = getEntityManager().merge(entity);

EntityManagerHelper.log("update successful", Level.INFO, null);

return result;

} catch (RuntimeException re) {

EntityManagerHelper.log("update failed", Level.SEVERE, re);

throw re;

}

}

public UserinfoVO findById(String id) {

EntityManagerHelper.log("finding Userinfo instance with id: " + id,

Level.INFO, null);

try {

UserinfoVO instance = getEntityManager().find(UserinfoVO.class, id);

return instance;

} catch (RuntimeException re) {

EntityManagerHelper.log("find failed", Level.SEVERE, re);

throw re;

}

}

/\*\*

\* Find all Userinfo entities with a specific property value.

\*

\* @param propertyName

\* the name of the Userinfo property to query

\* @param value

\* the property value to match

\* @return List<Userinfo> found by query

\*/

@SuppressWarnings("unchecked")

public List<UserinfoVO> findByProperty(String propertyName, final Object value) {

EntityManagerHelper.log("finding Userinfo instance with property: "

+ propertyName + ", value: " + value, Level.INFO, null);

try {

final String queryString = "select model from Userinfo model where model."

+ propertyName + "= :propertyValue";

Query query = getEntityManager().createQuery(queryString);

query.setParameter("propertyValue", value);

return query.getResultList();

} catch (RuntimeException re) {

EntityManagerHelper.log("find by property name failed",

Level.SEVERE, re);

throw re;

}

}

public List<UserinfoVO> findByPassword(Object password) {

return findByProperty(PASSWORD, password);

}

/\*\*

\* Find all Userinfo entities.

\*

\* @return List<Userinfo> all Userinfo entities

\*/

@SuppressWarnings("unchecked")

public List<UserinfoVO> findAll() {

EntityManagerHelper.log("finding all Userinfo instances", Level.INFO,

null);

try {

final String queryString = "select model from Userinfo model";

Query query = getEntityManager().createQuery(queryString);

return query.getResultList();

} catch (RuntimeException re) {

EntityManagerHelper.log("find all failed", Level.SEVERE, re);

throw re;

}

}

}

下面需要在face-config.xml中进行配置成受控ManagedBean即可。

可以用前面的web程序直接进行测试，也可以在DAO类中添加一个main方法，对DAO方法进行调用测试。

public static void main(String[] args){

Userinfo user = new Userinfo();

user.setUsername”admin”);

user.setPassword(“123”);

UserinfoDAO dao = new UserinfoDAO();

dao.save(

}

# 实验三 struts+spring+hibernate框架访问数据库的应用

**实验目的：**

掌握struts框架的配置和简单开发；

掌握spring管理Bean的方法

掌握Hibernate访问数据库的方法

**实验要求**：

1. struts框架实验

采用struts框架的目的在于框架管理页面和业务控制类Action之间联系，从而将程序员的重点关注业务类的开发上。在采用struts框架的前提下，前面的UserVO和UserDAO都可以得到重用。这一点和采用JSF框架的作用是类似的。至于JSF和Struts的比较则是各有优缺点，就像JPA和Hibernate的比较一样。相对于JSF中的ManagedBean，Action则更容易使用些，不需要将每个属性都要配置进行受管，直接就可以在表达式中使用，这样可以减少使用的复杂度。至于页面之间的导航也很类似，可以根据方法的返回字符串决定返回到哪个页面。

采用struts框架，设计通过下列JSP页面访问数据库(如表5-1)，具体要求如下：

1．添加用户页面(AddUser.jsp),如图9-1所示；



图9-1 添加用户页面

VOUser

DB

DAOUser

AddUser.jsp

**实验步骤：**

1．准备需要访问的数据库

下载并安装MySQL5.0

Copy MySQL5.0驱动mysql-connector-java-5.0.8-bin.jar到%TOMCAT%\lib文件夹下创建数据库及表：

数据库的设计如下表：

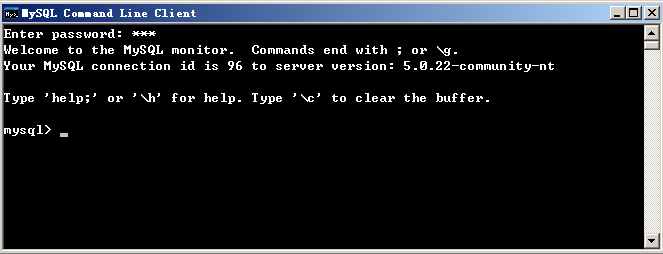
数据库采用MySQL5.0，数据库名：db，表名：user（用户表）

表5-1 数据库的用户表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 长度 | 主键 | 允许空 | 默认值 | 说明 |
| 1 | username | varchar | 40 | 是 | 否 |  | 用户名 |
| 2 | password | varchar | 40 |  | 是 |  | 口令 |

创建过程如下：(可选择使用MySQL的数据库管理和开发工具Navicat 和 MySQL-Front等)

Step1:进入MySQL程序的MySQL Command Line Client界面，如下图所示



Step2:分别执行下列命令：//支持中文

① 创建数据库：

CREATE DATABASE `db`

CHARACTER SET 'utf8'

COLLATE 'utf8\_general\_ci';

USE DB;

② 建表:

CREATE TABLE `user` (

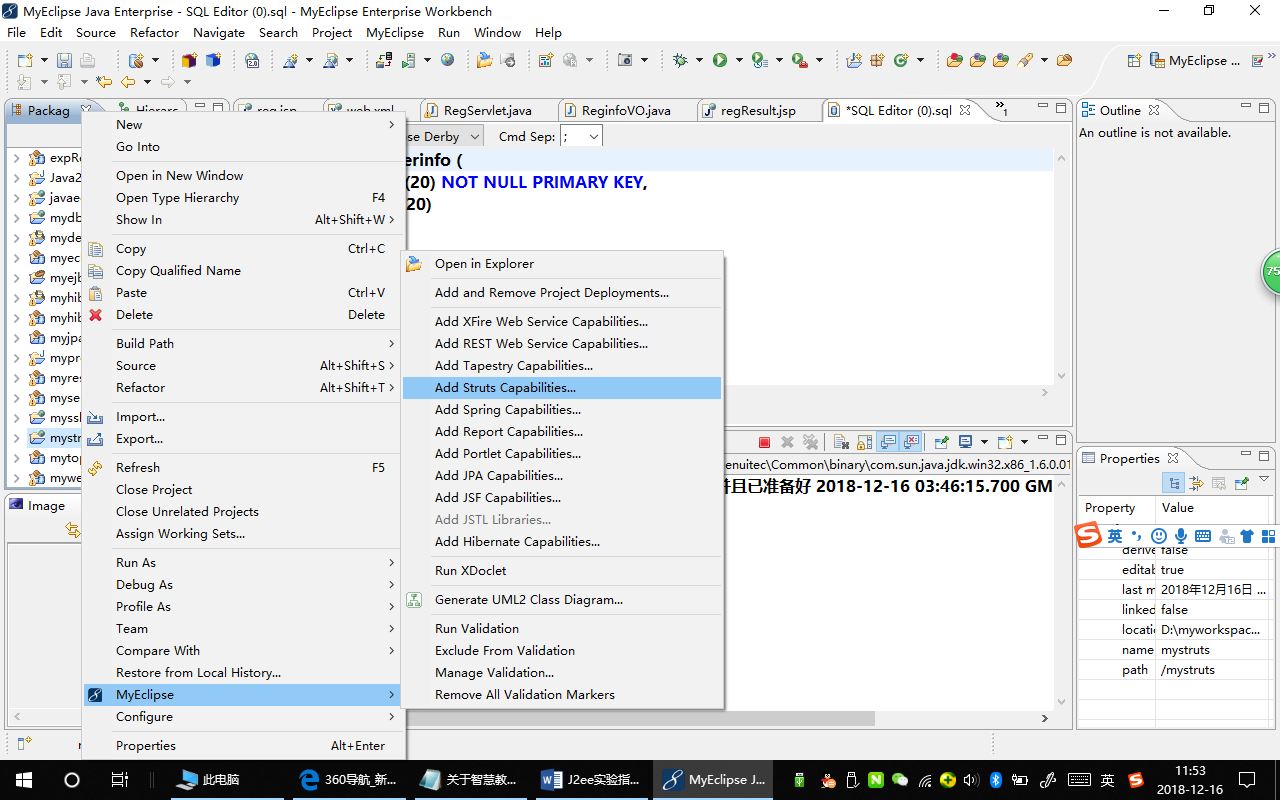
`username` varchar(20) NOT NULL PRIMARY KEY,

`password` varchar(20)

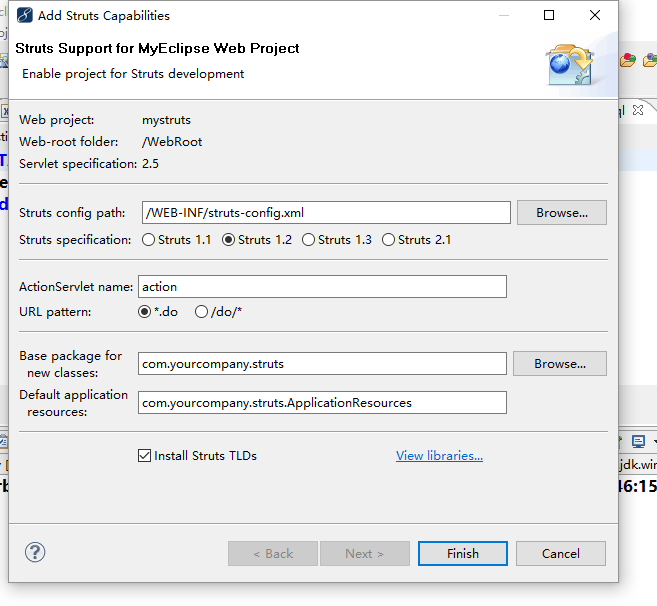
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

开发过程如下：

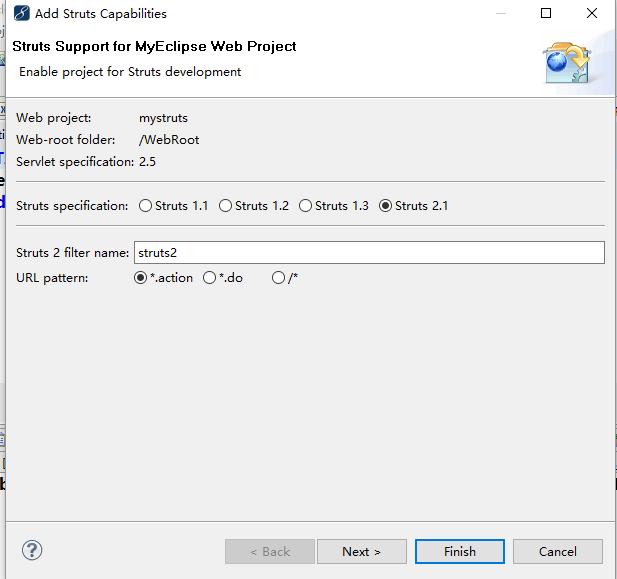
Step0：配置struts框架



出现下面窗口：

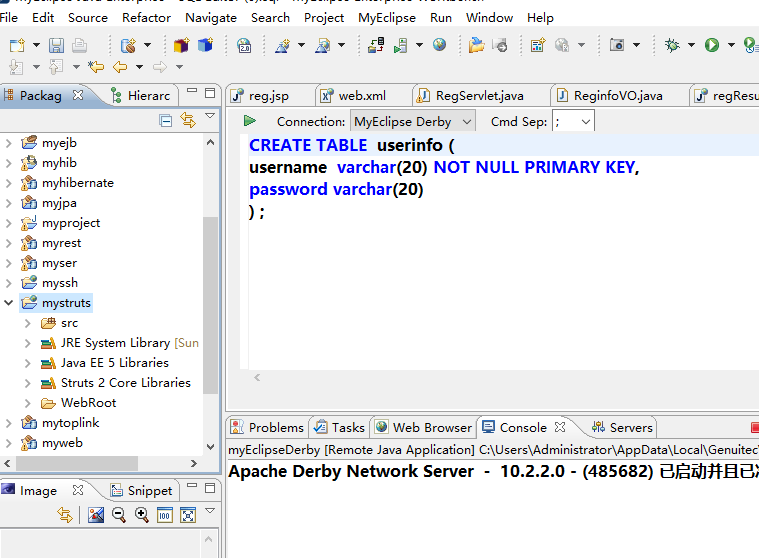


选择 struts2.1



选择finish。

工程下面会出现Struts 2 Core Libraries



此时struts框架已安装好。

***在Struts框架下面，相当于MVC模式中的Servlet，JSF框架下的ManagedBean的角色是Action，在这里起的是业务控制的作用，当然也可以起JavaBean的作用。***

Step1: 创建Action: AddUser.java,内容如下：

package valueBean;

public class UserVO{

private String username;//用户名

private String password;//口令

public UserVO()

{

this.username="";

this.password="";

}

public void setUsername(String username) {

this.username = username;

}

public String getUsername() {

return username;

}

public void setPassword(String password) {

this.password = password;

}

public String getPassword() {

return password;

}

}

Step2: 创建访问数据库的类—UserDAO.java,内容如下：

package toolBean.db;

import java.sql.\*;

import valueBean.User;

public class UserDAO {

private static Connection conn=null;

private static Statement stmt=null;

//构造函数中完成对数据库进行初始化

public UserDAO() {

//创建连接

conn=this.getConnection();

try {

//创建Statement

stmt=conn.createStatement();

} catch (SQLException e) {

e.printStackTrace();

}

}

//将对象user添加到表中

public void Add(VOUser user) throws SQLException

{

String sql="INSERT INTO user(username,password) VALUES('"

+user.getUsername()+"','"+user.getPassword()+"')";

stmt.executeUpdate(sql);

}

//根据用户名（关键字）从数据库删除相应的记录

public void Delete(String username) throws SQLException

{

String sql="Delete From user Where username='"+username+"'";

stmt.executeUpdate(sql);

}

public User searchByUsername(String username) throws SQLException

{

User user=new User();

String sql="select \* from user where username='"+username+"'";

ResultSet rs=stmt.executeQuery(sql);

while(rs.next())

{

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

if(name.equals(username))

{

//将从数据库查找得到的记录存入对象user中

user.setUsername(name);

user.setPassword(rs.getString("password"));

if(rs!=null)

rs.close();

return user;

}

}

return null;//查找不到返回null

}

//根据用户判断该记录是否存在

public boolean exists(String username)

{

boolean rtn=false;

try {

if(searchByUsername(username)!=null)

rtn=true;

} catch (SQLException e) {

e.printStackTrace();

}

return rtn;

}

///建立与数据的连接

private Connection getConnection()

{

String driverClass="com.mysql.jdbc.Driver";

String url=//与创建数据库的代码结合，保证访问数据库时不出现乱码

"jdbc:mysql://localhost:3306/db?useUnicode=true&characterEncoding=UTF-8";

String username = "root";

String password = "123";

try {

Class.forName(driverClass);// 加载数据库驱动

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

try {

conn = DriverManager.getConnection(url, username, password);//建立连接

} catch (SQLException e) {

e.printStackTrace();

System.out.print("数据库连接失败！");

}

return conn;

}

//关闭与访问数据库有关连接

protected void finalize()

{

try {

if(stmt!=null)

stmt.close();

if(conn!=null)

conn.close();

}

catch (SQLException e) {

e.printStackTrace();

}

}

}

Step3: 创建JSP网页

创建 添加用户页面—AddUser.jsp,代码如下：

此处的页面标签需要使用Struts提供的标签库来实现，该标签库为：\struts-tags

<%@ page contentType="text/html; charset=gb2312" language="java" import="java.sql.\*" errorPage="" %>

<%@ taglib uri=”/strut-tags” prefix=”s”/>

**<s:form method="post" action="AddUser.action">**

<div align="center"><font face="宋体" size="6"><strong>

添加用户</strong></font><br/><hr/>

<s:textfield name="vouser.username" label="用户名："/>

<br/>

<br/>

<s:password name="vouser.username" label="密码："/>

<br />

<br />

<input type="submit" value="添加" />

<input type="reset" value="取消" />

</div>

**</s:form>**

利用向导创建success.jsp和error.jsp

其内容分别显示success和register error!

stept 4添加Action,代码如下：

package action;

public class AddUser extends ActionSupport {

UserVO vouser =new UserVO();

UserDAO userDB=new DAOUser();

public void setVouser(UserVO vouser){

this.vouser = vouser;

}

public UserVO getVouser(){

return vouser;

}

public String execute(){

if(userDB.exists(user.getUsername()))

return “INPUT”;

try {

userDB.Add(vouser);

} catch (SQLException e) {

return “ERROR”;

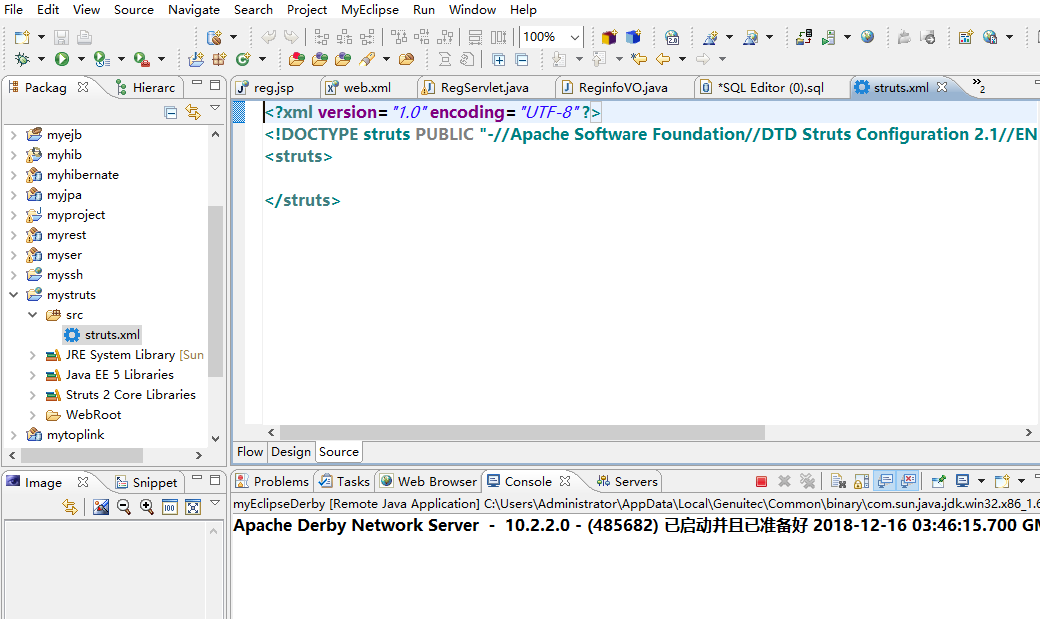
}

}

}

Step5 配置struts.xml

在工程-src下面找到struts.xml，双击打开，在窗口下面点击Source。



在<struts>下面输入：

<package name=”mypack” path=”/” extends=”struts-defaut”>

<action name=”AddUser” class=”action.AddUser”>

<result name=”success”>success.jsp</result>

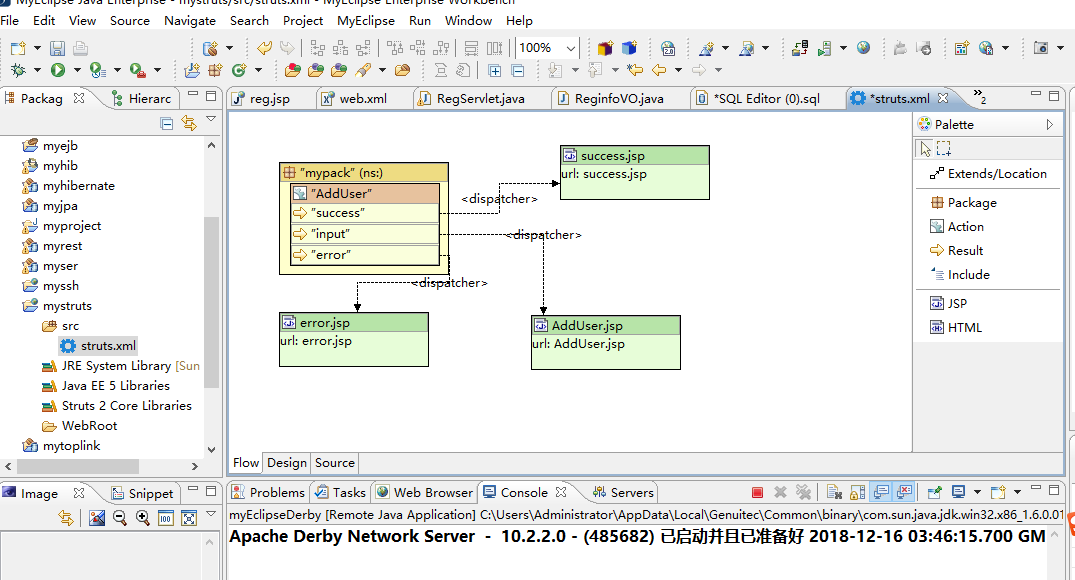
<result name=”input”>AddUser.jsp</result>

<result name=”error”>error.jsp</result>”

</action>

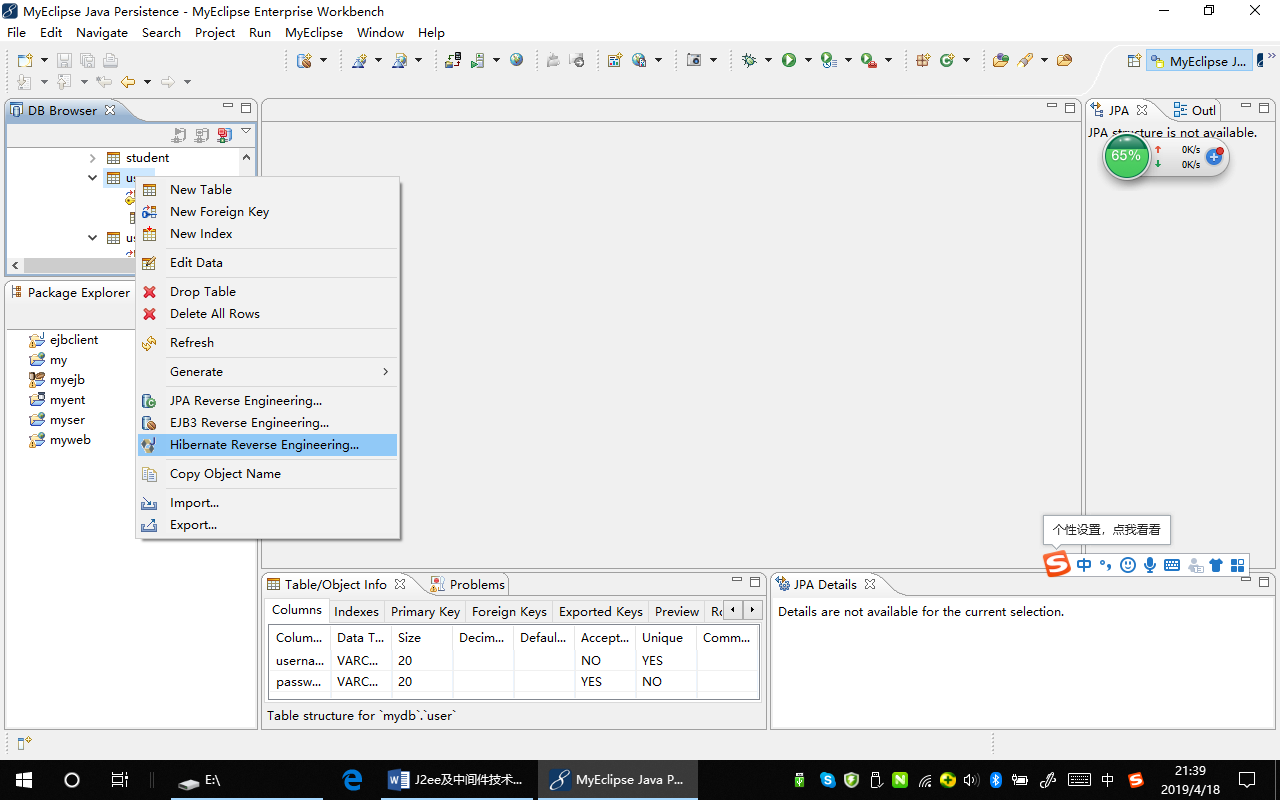
</packge>

如果点击该窗口下方的flow按钮，则系统显示action和页面之间的导航关系。

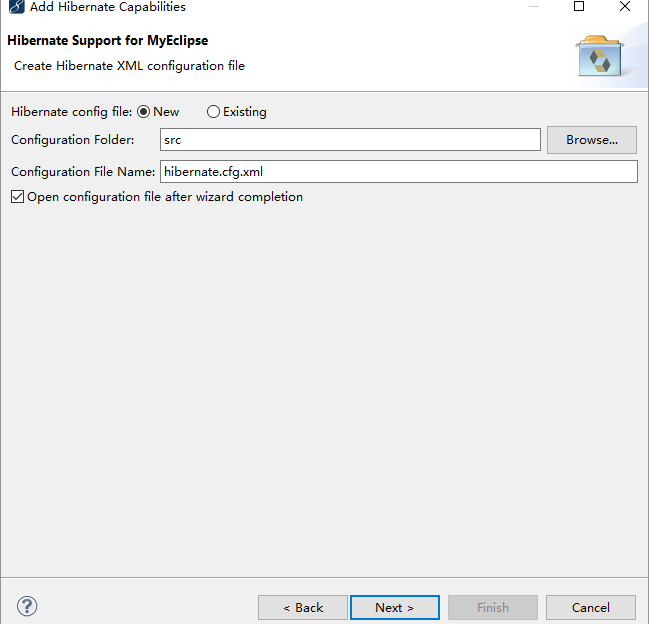


1. Hibernate实验

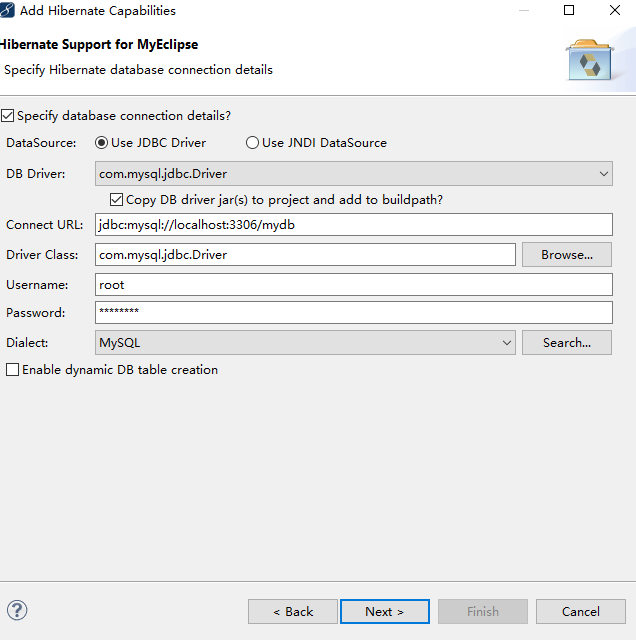
类似于JPA实验，Hibernate是另一种ORM框架，这里可以根据数据库表自动生成VO和DAO，从而替代实验一中的VO和DAO。



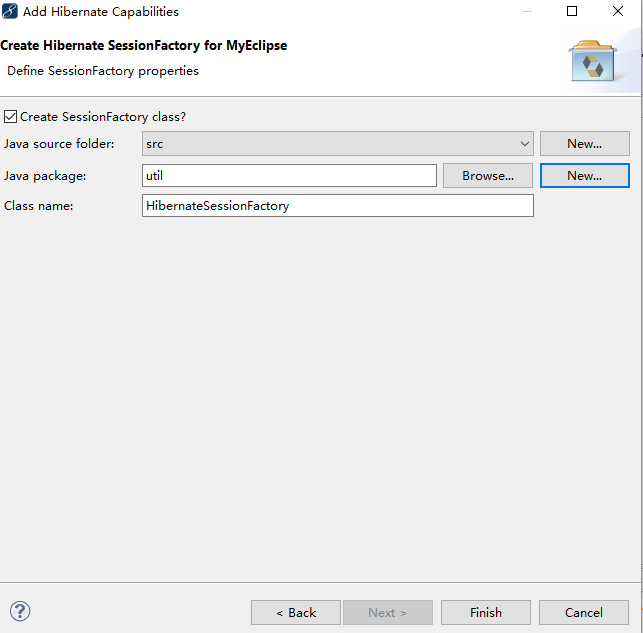
注意在逆向之前首先要在改工程中添加Hibernate Capalities



配置数据源。



选择创建sessionFactory的路径，该类用来创建sessionFactory，实际上就是数据库连接。



点击finish

有关数据库连接的信息都保存在配置文件Hibernate.cfg.xml中。

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

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

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

<!-- Generated by MyEclipse Hibernate Tools. -->

<hibernate-configuration>

<session-factory>

<property name=*"dialect"*>org.hibernate.dialect.MySQLDialect</property>

<property name=*"connection.url"*>jdbc:mysql://localhost:3306/mydb</property>

<property name=*"connection.username"*>root</property>

<property name=*"connection.password"*>12345678</property>

<property name=*"connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<property name=*"myeclipse.connection.profile"*>com.mysql.jdbc.Driver</property>

</session-factory>

</hibernate-configuration>

HibernateSessionFactory的代码如下：

package util;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.cfg.Configuration;

import org.hibernate.cfg.AnnotationConfiguration;

/\*\*

\* Configures and provides access to Hibernate sessions, tied to the

\* current thread of execution. Follows the Thread Local Session

\* pattern, see {@link http://hibernate.org/42.html }.

\*/

public class HibernateSessionFactory {

/\*\*

\* Location of hibernate.cfg.xml file.

\* Location should be on the classpath as Hibernate uses

\* #resourceAsStream style lookup for its configuration file.

\* The default classpath location of the hibernate config file is

\* in the default package. Use #setConfigFile() to update

\* the location of the configuration file for the current session.

\*/

private static String CONFIG\_FILE\_LOCATION = "/hibernate.cfg.xml";

private static final ThreadLocal<Session> threadLocal = new ThreadLocal<Session>();

private static Configuration configuration = new AnnotationConfiguration();

private static org.hibernate.SessionFactory sessionFactory;

private static String configFile = CONFIG\_FILE\_LOCATION;

static {

try {

configuration.configure(configFile);

sessionFactory = configuration.buildSessionFactory();

} catch (Exception e) {

System.err

.println("%%%% Error Creating SessionFactory %%%%");

e.printStackTrace();

}

}

private HibernateSessionFactory() {

}

/\*\*

\* Returns the ThreadLocal Session instance. Lazy initialize

\* the <code>SessionFactory</code> if needed.

\*

\* @return Session

\* @throws HibernateException

\*/

public static Session getSession() throws HibernateException {

Session session = (Session) threadLocal.get();

if (session == null || !session.isOpen()) {

if (sessionFactory == null) {

rebuildSessionFactory();

}

session = (sessionFactory != null) ? sessionFactory.openSession()

: null;

threadLocal.set(session);

}

return session;

}

/\*\*

\* Rebuild hibernate session factory

\*

\*/

public static void rebuildSessionFactory() {

try {

configuration.configure(configFile);

sessionFactory = configuration.buildSessionFactory();

} catch (Exception e) {

System.err

.println("%%%% Error Creating SessionFactory %%%%");

e.printStackTrace();

}

}

/\*\*

\* Close the single hibernate session instance.

\*

\* @throws HibernateException

\*/

public static void closeSession() throws HibernateException {

Session session = (Session) threadLocal.get();

threadLocal.set(null);

if (session != null) {

session.close();

}

}

/\*\*

\* return session factory

\*

\*/

public static org.hibernate.SessionFactory getSessionFactory() {

return sessionFactory;

}

/\*\*

\* return session factory

\*

\* session factory will be rebuilded in the next call

\*/

public static void setConfigFile(String configFile) {

HibernateSessionFactory.configFile = configFile;

sessionFactory = null;

}

/\*\*

\* return hibernate configuration

\*

\*/

public static Configuration getConfiguration() {

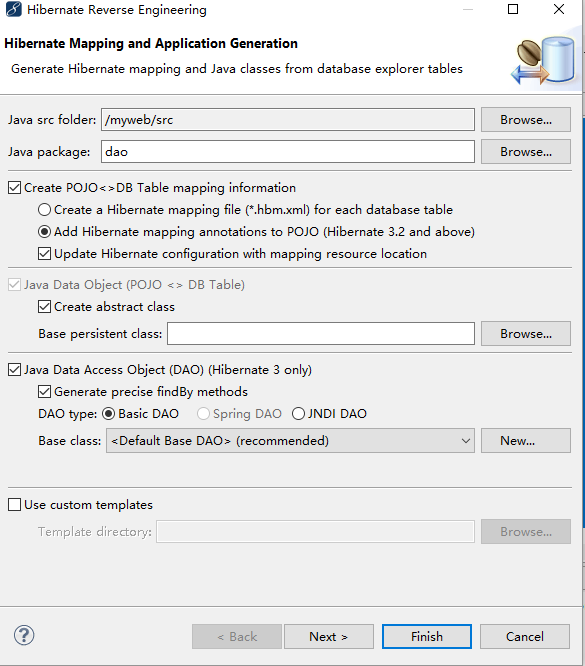
return configuration;

}

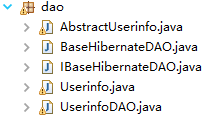
}

该类会在逆向生成的DAO中调用。

逆向工程产生DAO，类似于JPA产生DAO。



产生的类如下：



最核心的就是Userinfo(相当于UserVO)和UserDAO。

**package** dao;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

/\*\*

\* Userinfo entity. **@author** MyEclipse Persistence Tools

\*/

@Entity

@Table(name = "userinfo", catalog = "mydb")

**public** **class** Userinfo **implements** java.io.Serializable {

// Fields

**private** String username;

**private** String password;

// Constructors

/\*\* default constructor \*/

**public** Userinfo() {

}

/\*\* full constructor \*/

**public** Userinfo(String username, String password) {

**this**.username = username;

**this**.password = password;

}

// Property accessors

@Id

@Column(name = "username", unique = **true**, nullable = **false**, length = 10)

**public** String getUsername() {

**return** **this**.username;

}

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

**this**.username = username;

}

@Column(name = "password", nullable = **false**, length = 10)

**public** String getPassword() {

**return** **this**.password;

}

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

**this**.password = password;

}

}

这和JPA中产生的完全相同。

DAO则略有不同。

package dao;

import java.util.List;

import org.hibernate.LockMode;

import org.hibernate.Query;

import static org.hibernate.criterion.Example.create;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

/\*\*

\* A data access object (DAO) providing persistence and search support for

\* Userinfo entities. Transaction control of the save(), update() and delete()

\* operations can directly support Spring container-managed transactions or they

\* can be augmented to handle user-managed Spring transactions. Each of these

\* methods provides additional information for how to configure it for the

\* desired type of transaction control.

\*

\* @see dao.Userinfo

\* @author MyEclipse Persistence Tools

\*/

public class UserinfoDAO extends BaseHibernateDAO {

private static final Logger log = LoggerFactory

.getLogger(UserinfoDAO.class);

// property constants

public static final String PASSWORD = "password";

public void save(Userinfo transientInstance) {

log.debug("saving Userinfo instance");

try {

getSession().save(transientInstance);

log.debug("save successful");

} catch (RuntimeException re) {

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

throw re;

}

}

public void delete(Userinfo persistentInstance) {

log.debug("deleting Userinfo instance");

try {

getSession().delete(persistentInstance);

log.debug("delete successful");

} catch (RuntimeException re) {

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

throw re;

}

}

public Userinfo findById(java.lang.String id) {

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

try {

Userinfo instance = (Userinfo) getSession().get("dao.Userinfo", id);

return instance;

} catch (RuntimeException re) {

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

throw re;

}

}

public List<Userinfo> findByExample(Userinfo instance) {

log.debug("finding Userinfo instance by example");

try {

List<Userinfo> results = (List<Userinfo>) getSession()

.createCriteria("dao.Userinfo").add(create(instance))

.list();

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 Userinfo instance with property: " + propertyName

+ ", value: " + value);

try {

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

+ propertyName + "= ?";

Query queryObject = getSession().createQuery(queryString);

queryObject.setParameter(0, value);

return queryObject.list();

} catch (RuntimeException re) {

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

throw re;

}

}

public List<Userinfo> findByPassword(Object password) {

return findByProperty(PASSWORD, password);

}

public List findAll() {

log.debug("finding all Userinfo instances");

try {

String queryString = "from Userinfo";

Query queryObject = getSession().createQuery(queryString);

return queryObject.list();

} catch (RuntimeException re) {

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

throw re;

}

}

public Userinfo merge(Userinfo detachedInstance) {

log.debug("merging Userinfo instance");

try {

Userinfo result = (Userinfo) getSession().merge(detachedInstance);

log.debug("merge successful");

return result;

} catch (RuntimeException re) {

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

throw re;

}

}

public void attachDirty(Userinfo instance) {

log.debug("attaching dirty Userinfo instance");

try {

getSession().saveOrUpdate(instance);

log.debug("attach successful");

} catch (RuntimeException re) {

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

throw re;

}

}

public void attachClean(Userinfo instance) {

log.debug("attaching clean Userinfo instance");

try {

getSession().lock(instance, LockMode.NONE);

log.debug("attach successful");

} catch (RuntimeException re) {

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

throw re;

}

}

}

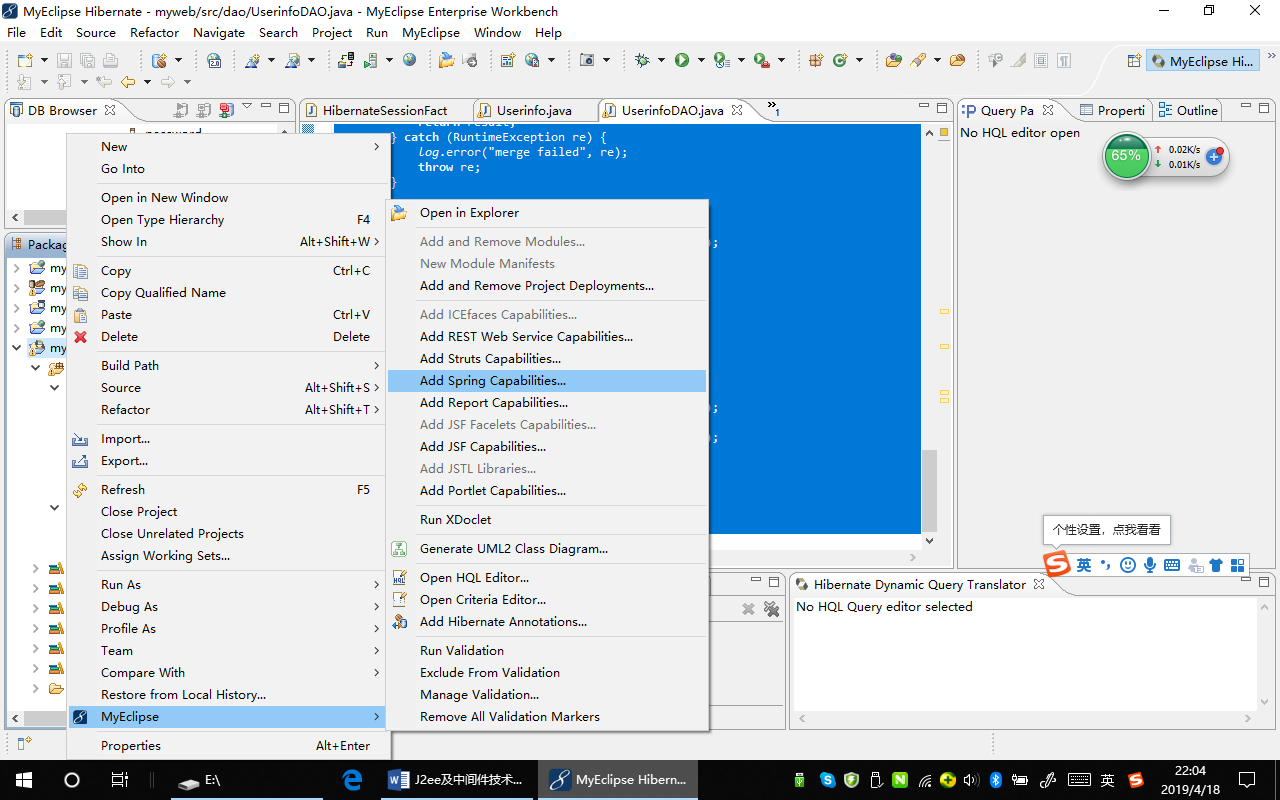
下面只要将该框架下自动产生的VO和DAO去替换前面自己开发的VO和DAO即可。

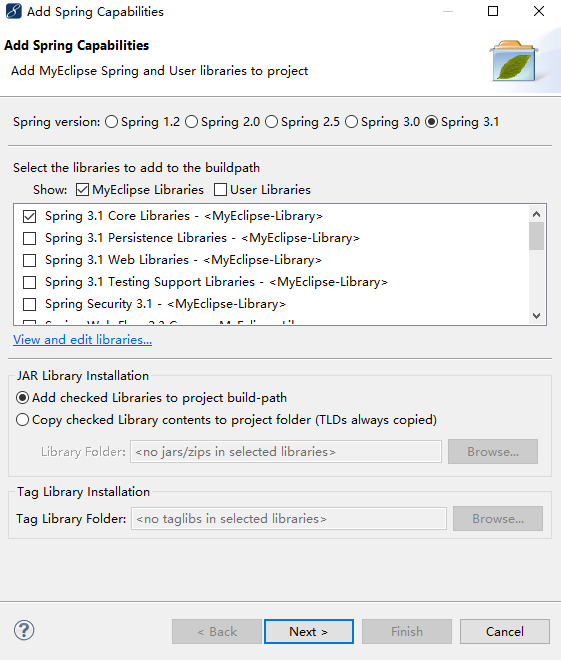
1. Spring实验

Spring的框架在于对象管理以及属性注入(injection)，这是通过类反射reflector和控制反转（Control Inverse）来实现的。

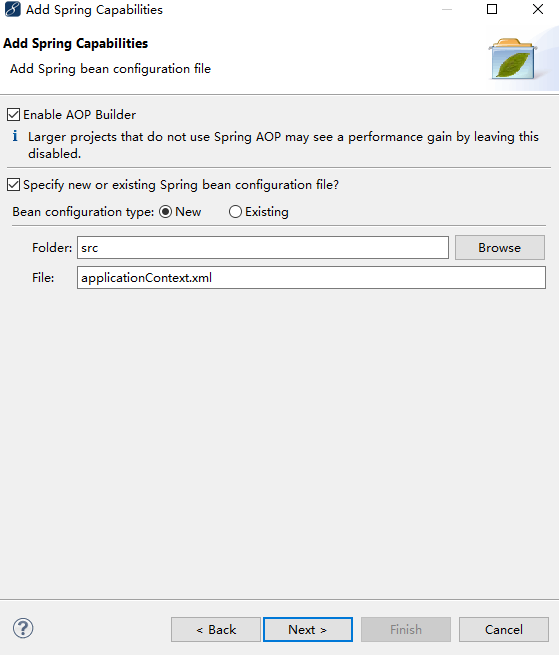
上述例子能实现对象管理的就是Struts中的Action对象，同时对其属性如VO和DAO进行注入。

为了使用Spring框架，首先需要在工程中使能Spring Capalities。





指定配置文件applicatoinContext.xml



将HibernateSessionFactory纳入Spring管理。

applicationContext.xml的内容：

<?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-3.1.xsd"*>

<bean id=*"sessionFactory"*

class=*"org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean"*>

<property name=*"configLocation"*

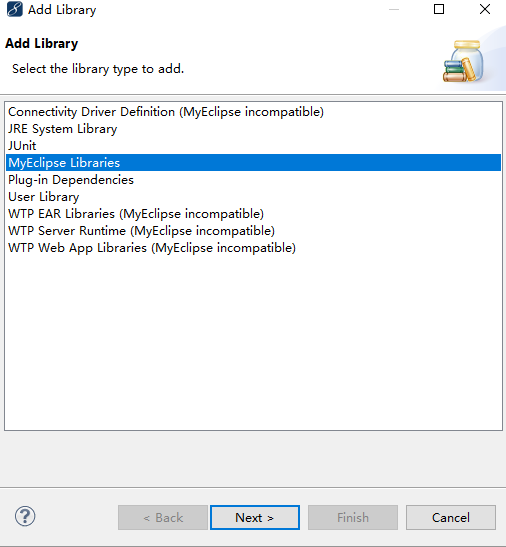
value=*"classpath:hibernate.cfg.xml"*>

</property>

</bean></beans>

此时会出现一个错误，*org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean*找不到。

将spring-persistence core library包导入：



# 实验四 使用WebService或EJBean访问数据库的应用

**实验目的：**

掌握webservice的开发方法；

掌握EJBean的开发方法

**实验要求**：

采用EJBean技术，设计通过下列JSP页面访问数据库(如表5-1)，具体要求如下：

1．添加用户页面(AddUser.jsp),如图4-1所示；



图4-1 添加用户页面

Userinfo

DB

AddUser.jsp

DAOUserinfo

图4-2 程序框架

**实验步骤：**

1．准备需要访问的数据库

下载并安装MySQL5.0

Copy MySQL5.0驱动mysql-connector-java-5.0.8-bin.jar到%TOMCAT%\lib文件夹下创建数据库及表：

数据库的设计如下表：

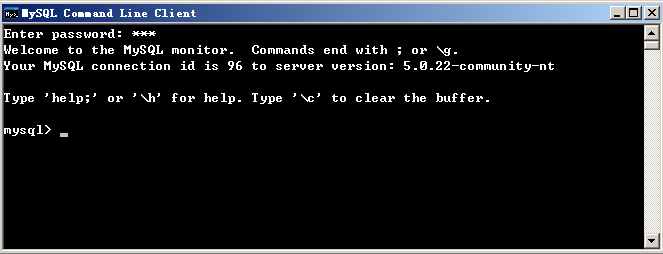
数据库采用MySQL5.0，数据库名：db，表名：user（用户表）

表5-1 数据库的用户表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 长度 | 主键 | 允许空 | 默认值 | 说明 |
| 1 | username | varchar | 40 | 是 | 否 |  | 用户名 |
| 2 | password | varchar | 40 |  | 是 |  | 口令 |

创建过程如下：(可选择使用MySQL的数据库管理和开发工具Navicat 和 MySQL-Front等)

Step1:进入MySQL程序的MySQL Command Line Client界面，如下图所示



Step2:分别执行下列命令：//支持中文

1. 创建数据库：

CREATE DATABASE `mydb`

CHARACTER SET 'utf8'

COLLATE 'utf8\_general\_ci';

USE DB;

1. 建表:

CREATE TABLE `userinfo` (

`username` varchar(20) NOT NULL PRIMARY KEY,

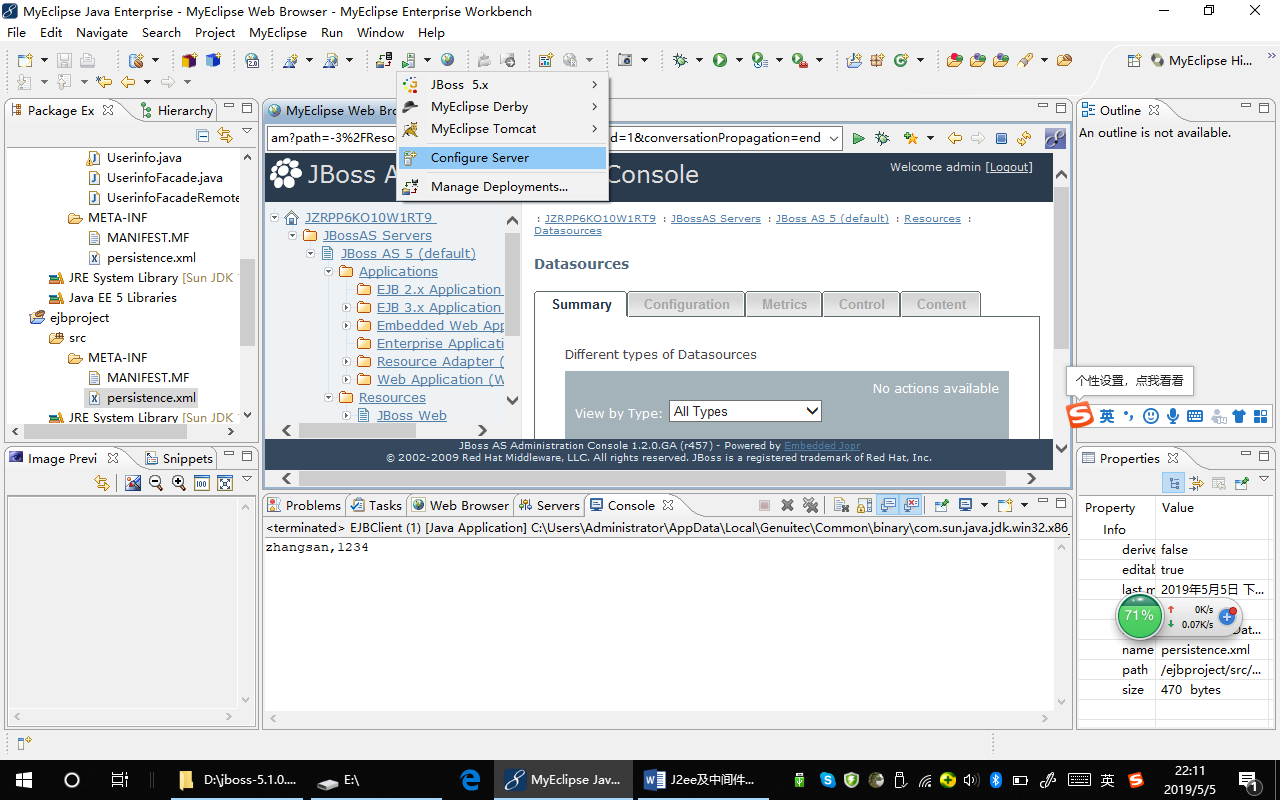
`password` varchar(20)

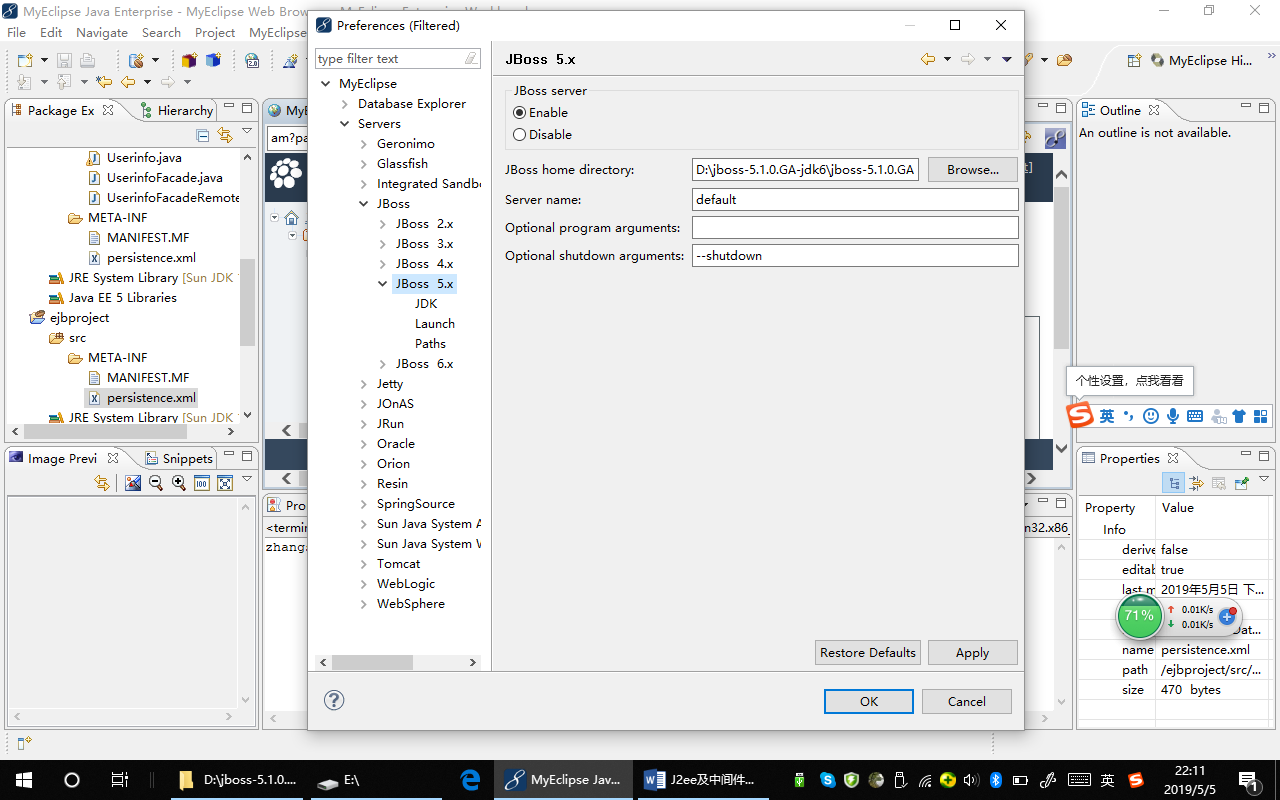
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

2、开发过程如下：

Step1：EJBean实验

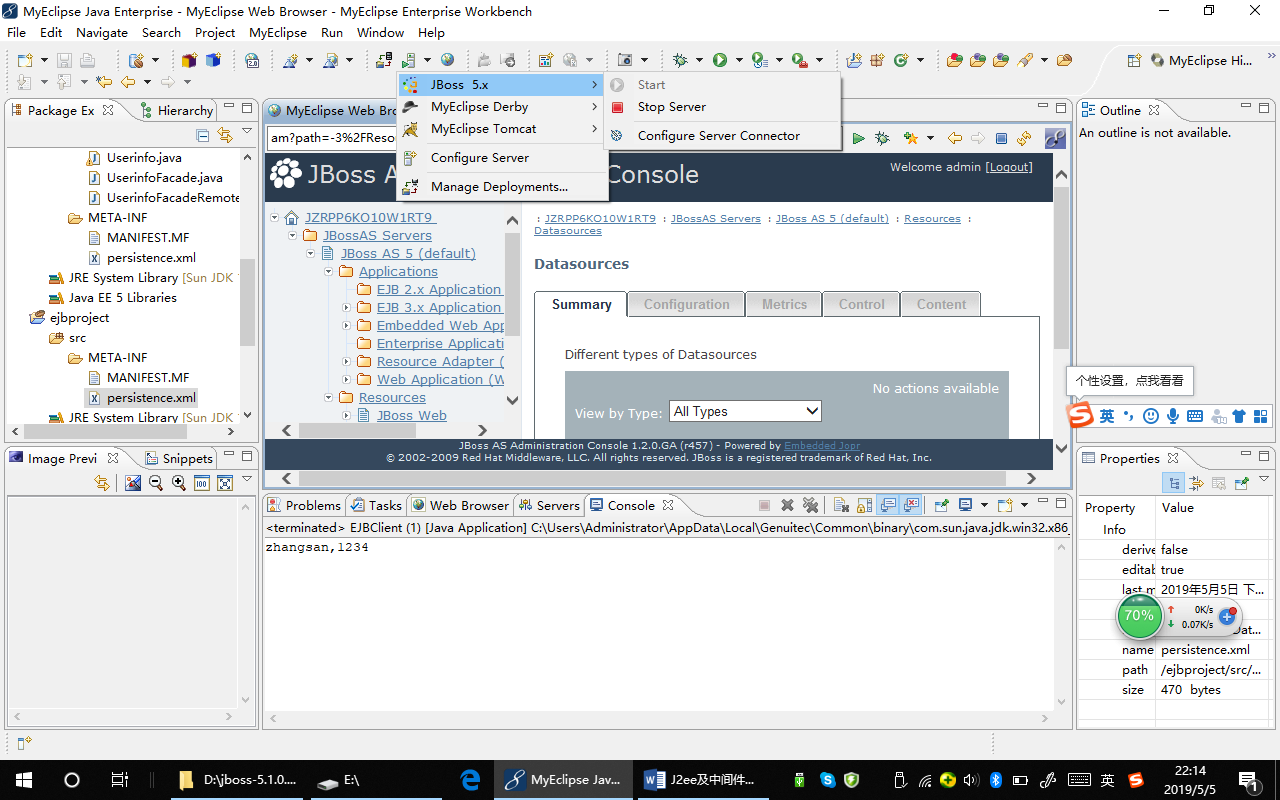
（1）首先进行环境配置，将JBoss5.1.0压缩包拷贝到计算机上，并解压，记住解压的路径，然后在MyEclipse中进行配置。



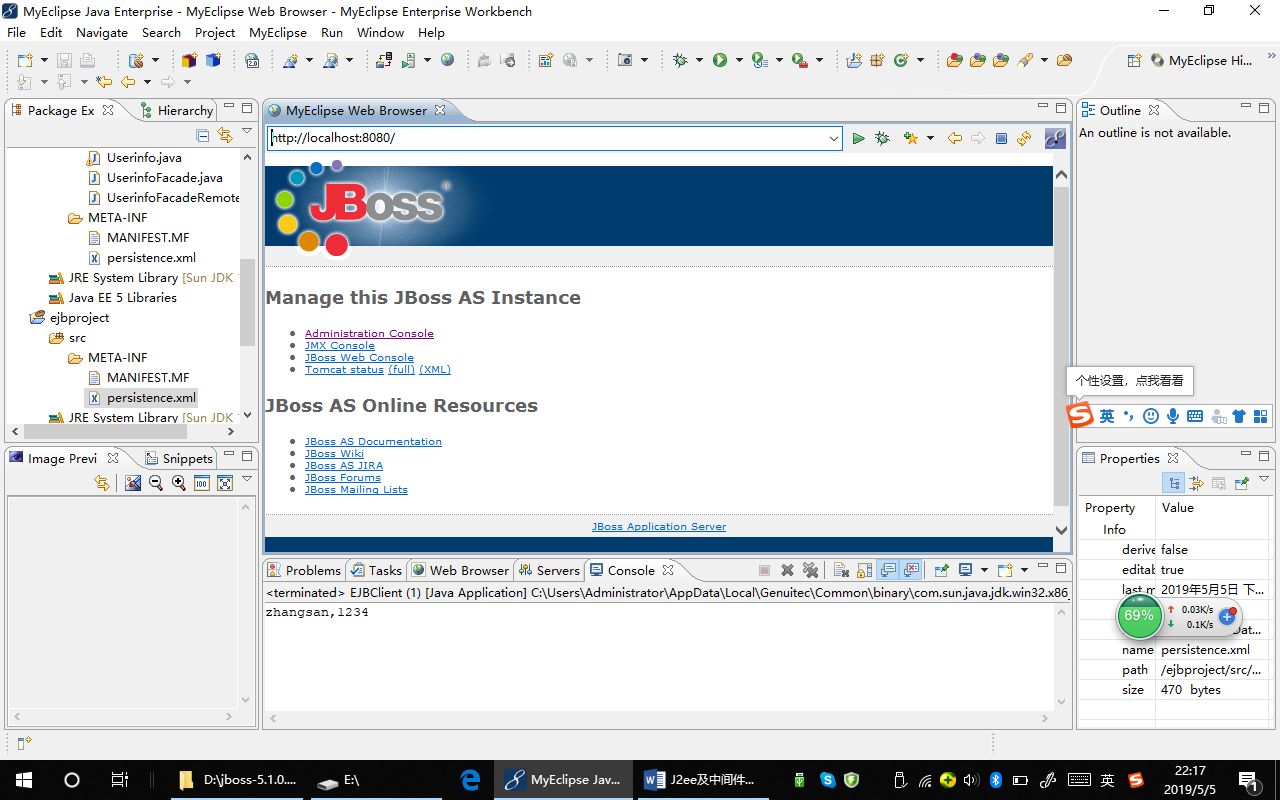


在上述图中，找到JBoss5.x，然后配置JBoss home directory为JBoss解压的目录。如：D:\jboss-5.1.0.GA-jdk6\jboss-5.1.0.GA，另外选择上面的JBoss server为Enable。

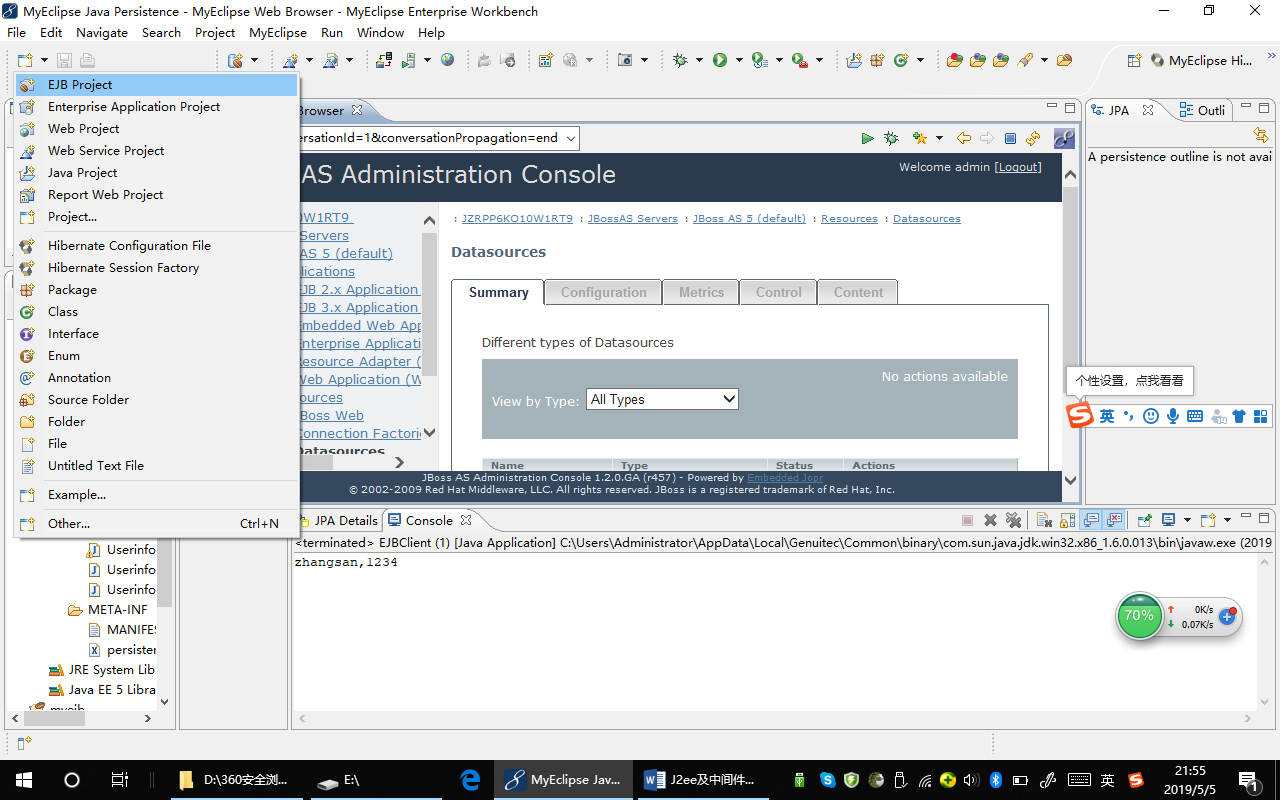
此时在Myeclipse中的Server图标处，会出现JBoss 5.，选择Start进行启动。



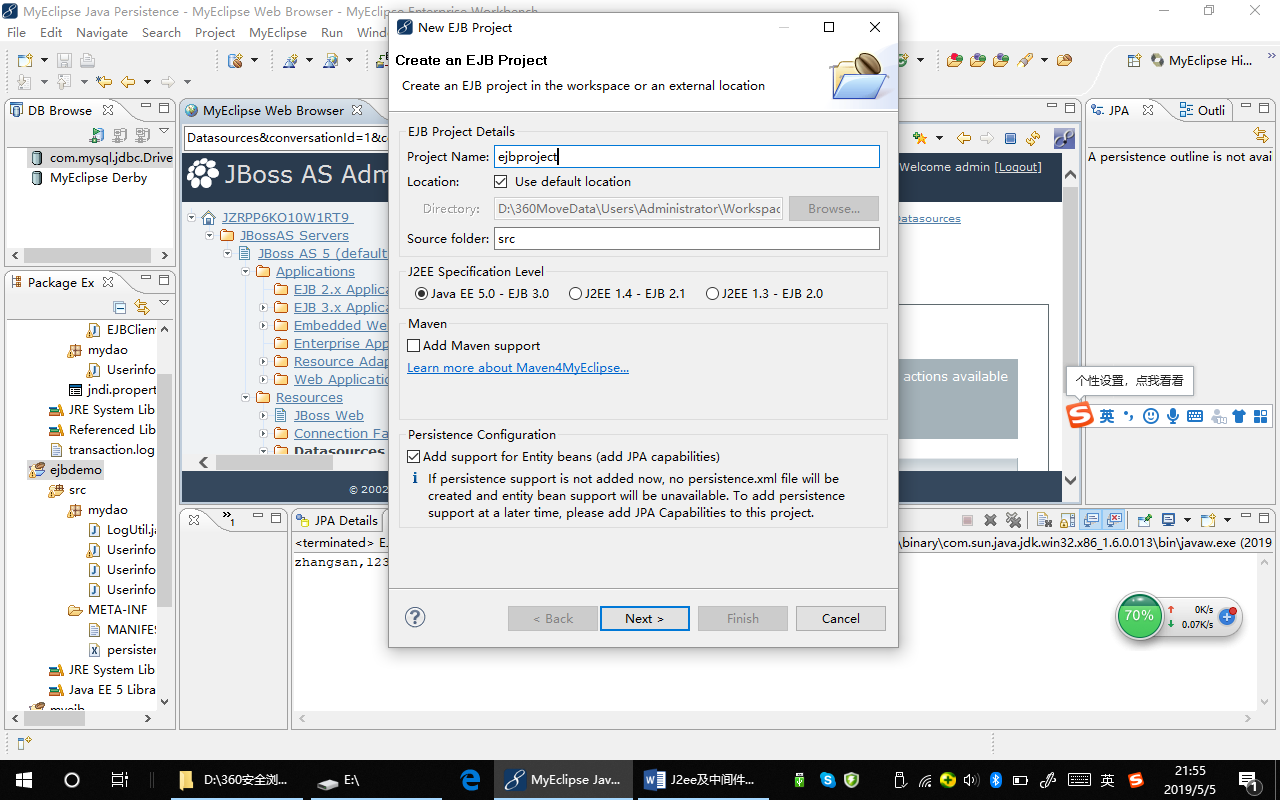
此时可以测试该服务器是否正常工作，新建web工程如myweb，将此工程发布到JBoss服务器下，然后在浏览器上进行测试，在地址栏中输入<http://localhost:8080>，然后出现JBoss的管理页面。



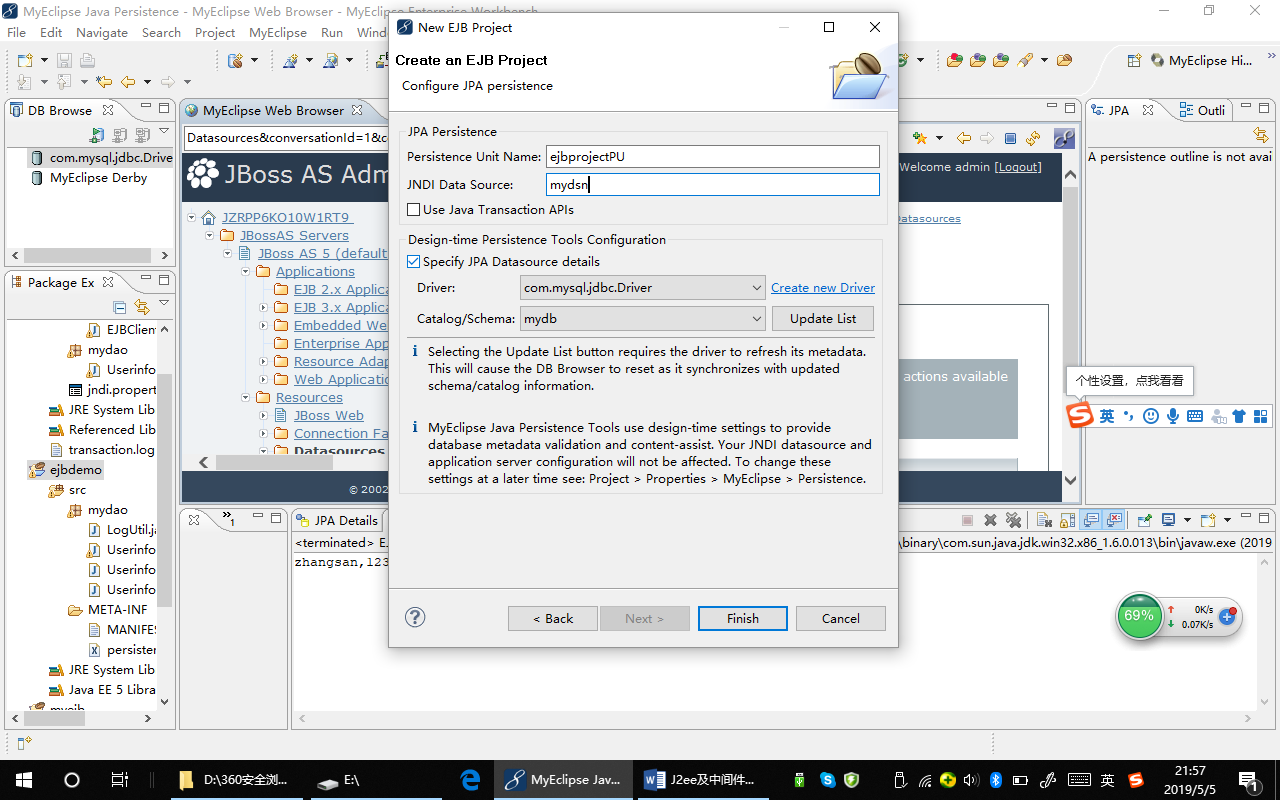
（2）在MyEclipse中，新建EJB Project.



输入EJBean project name:ejbproject



点击下一步，输入数据源JNDI Data Source，该数据源后面介绍如何配置。



点击finish，可以发现在src->META-INF下面出现一个persistence.xml，这一点和引入JPA框架有点类似，因为EJBean中的实体Bean本身用的也是JPA。

Step2：将具有如下内容的mysql-ds.xml文件拷贝到jboss的-server-default-deploy-下面。D:\jboss-5.1.0.GA-jdk6\jboss-5.1.0.GA\server\default\deploy

文件名：mysql-ds.xml

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

<!-- See http://www.jboss.org/community/wiki/Multiple1PC for information about local-tx-datasource -->

<!-- $Id: mysql-ds.xml 97536 2009-12-08 14:05:07Z jesper.pedersen $ -->

<!-- Datasource config for MySQL using 3.0.9 available from:

http://www.mysql.com/downloads/api-jdbc-stable.html

-->

<datasources>

<local-tx-datasource>

<jndi-name>DefaultMySqlDS</jndi-name>

<use-java-context>false</use-java-context>

<connection-url>jdbc:mysql://localhost:3306/mydb</connection-url>

<driver-class>com.mysql.jdbc.Driver</driver-class>

<user-name>root</user-name>

<password>12345678</password>

<exception-sorter-class-name>org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter</exception-sorter-class-name>

<!-- should only be used on drivers after 3.22.1 with "ping" support

<valid-connection-checker-class-name>org.jboss.resource.adapter.jdbc.vendor.MySQLValidConnectionChecker</valid-connection-checker-class-name>

-->

<!-- sql to call when connection is created

<new-connection-sql>some arbitrary sql</new-connection-sql>

-->

<!-- sql to call on an existing pooled connection when it is obtained from pool - MySQLValidConnectionChecker is preferred for newer drivers

<check-valid-connection-sql>some arbitrary sql</check-valid-connection-sql>

-->

<!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml (optional) -->

<metadata>

<type-mapping>mySQL</type-mapping>

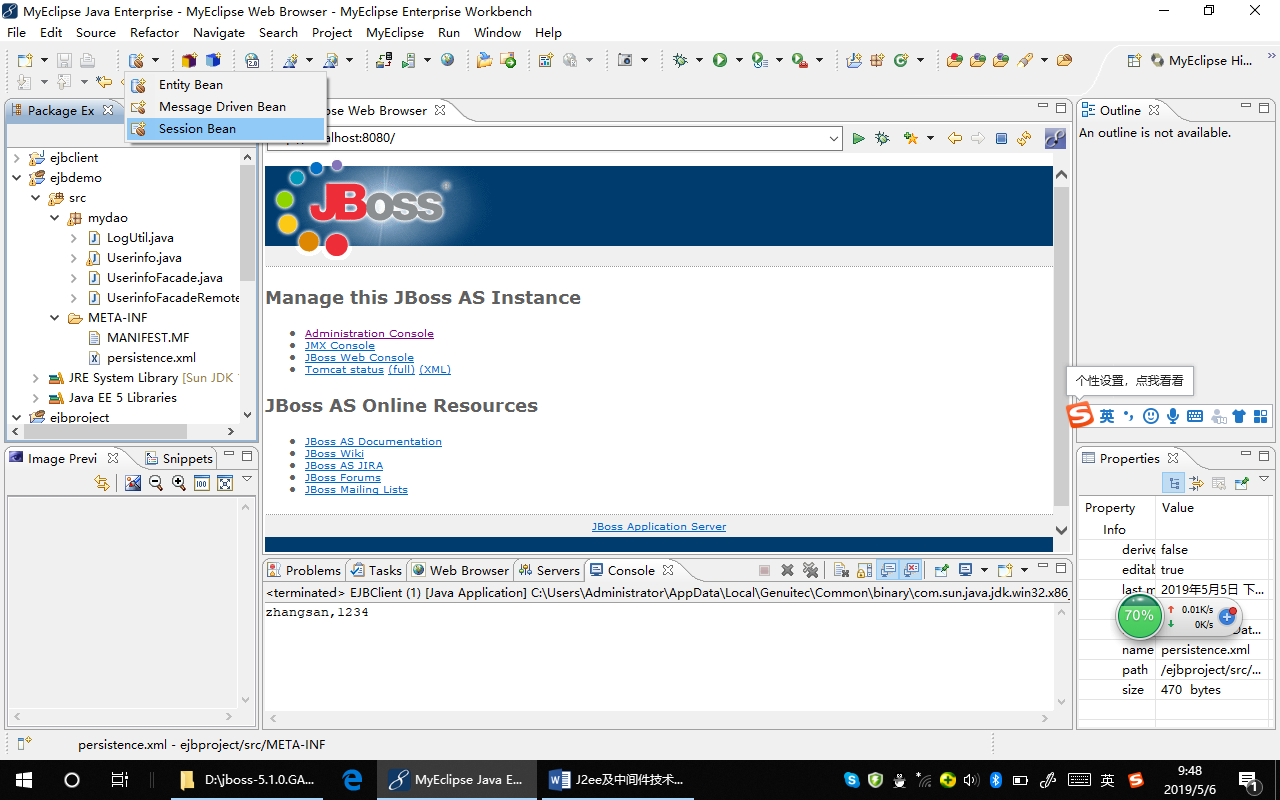
</metadata>

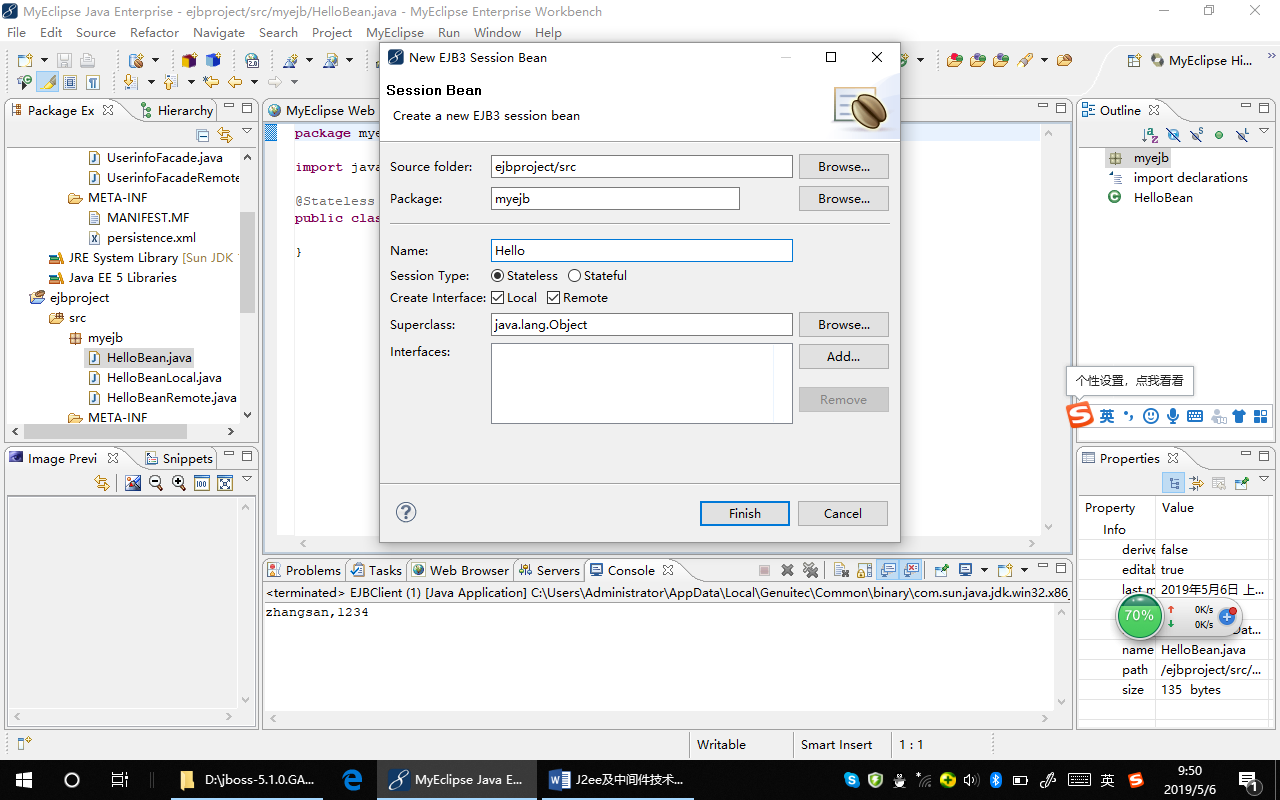
</local-tx-datasource>

</datasources>

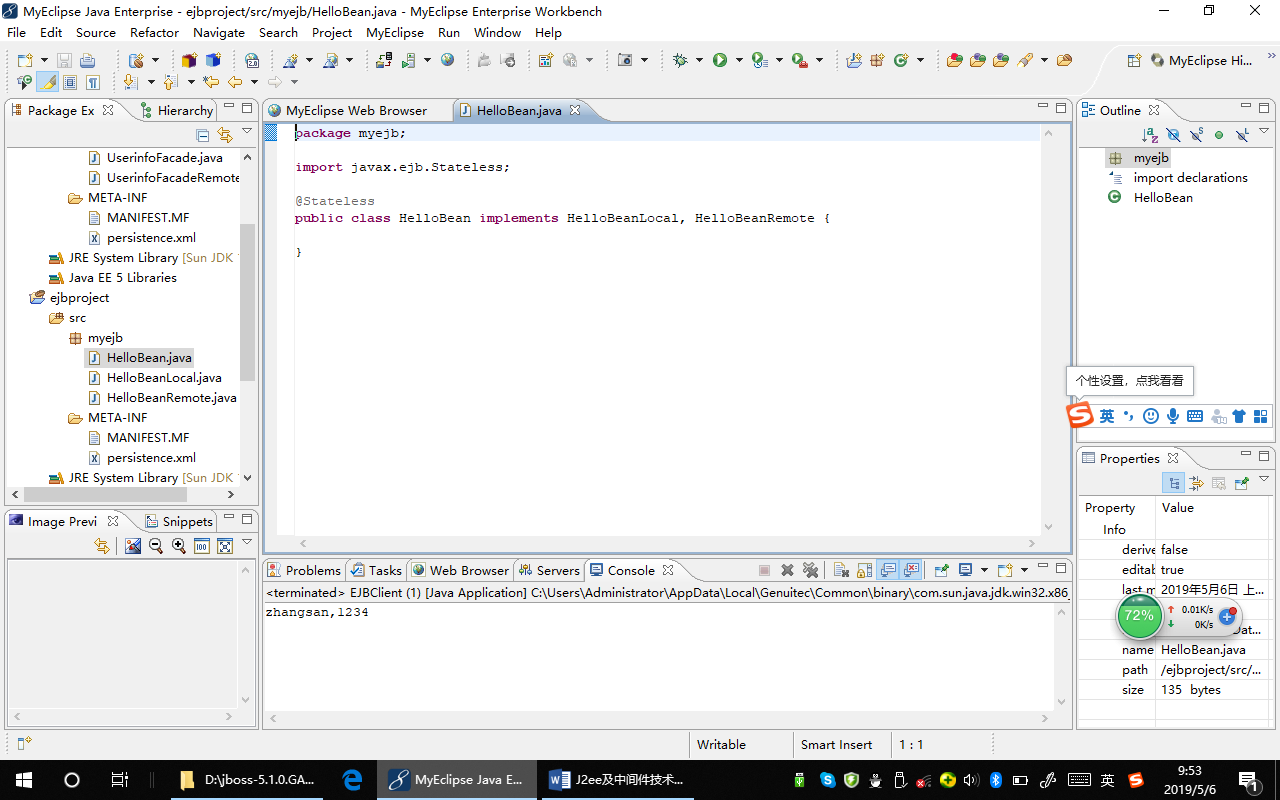
Step2：创建Stateless Session Bean进行测试

在MyEclipse中，找到创建bean的图表，选择session bean





勾选Local,Rmoete选择finish后，在myejb包下，创建了如下接口和类：



HelloBean,HelloBeanRmote,HelloBeanLocal

下面增加一个业务接口，Hello，其中增加一个方法sayHello().

public interface Hello {

public String sayHello(String name)；

}

然后让HelloBeanRmote, HelloBeanLocal都继承自Hello。

**package** myejb;

**import** javax.ejb.Local;

@Local

**public** **interface** HelloBeanLocal **extends** Hello {

}

**package** myejb;

**import** javax.ejb.Remote;

@Remote

**public** **interface** HelloBeanRemote **extends** Hello {

}

在HelloBean下面，实现该方法。

**package** myejb;

**import** javax.ejb.Stateless;

@Stateless

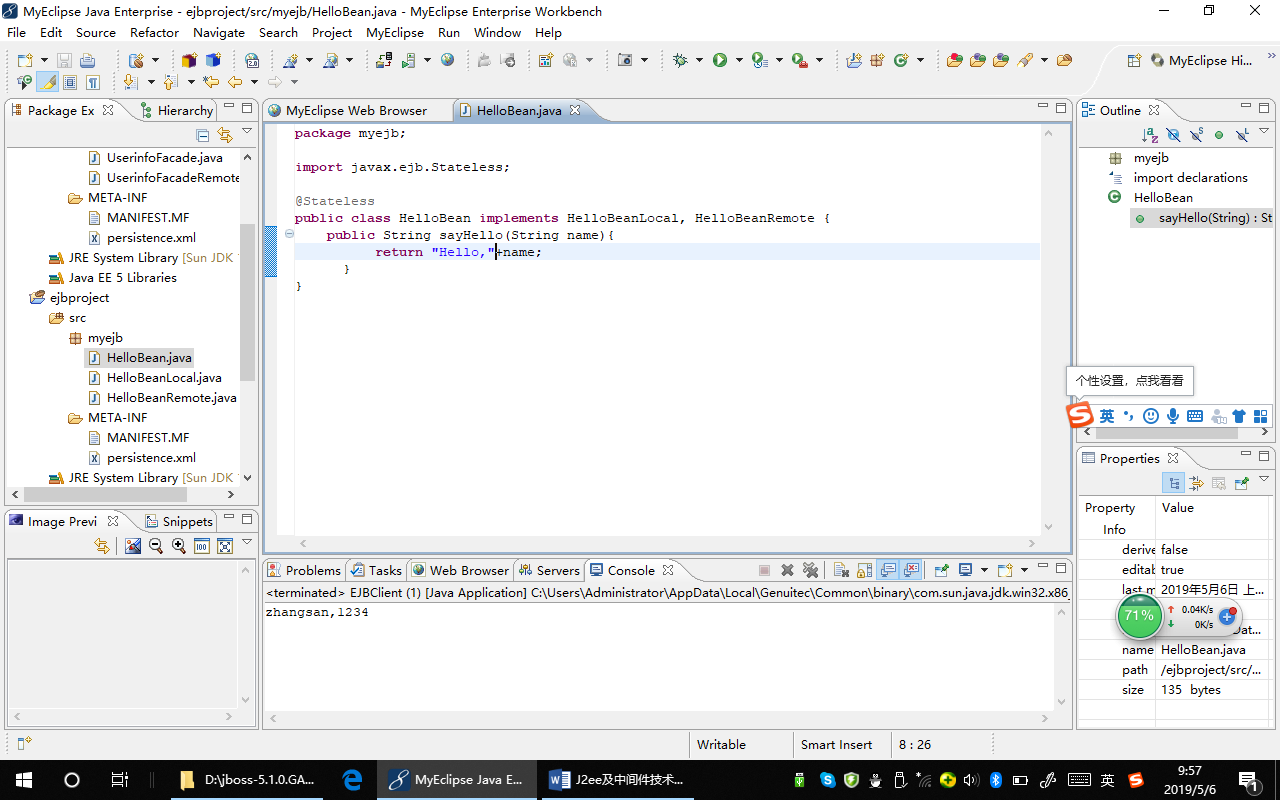
**public** **class** HelloBean **implements** HelloBeanLocal, HelloBeanRemote {

public String sayHello(String name){

return “Hello,”+name;

}

}



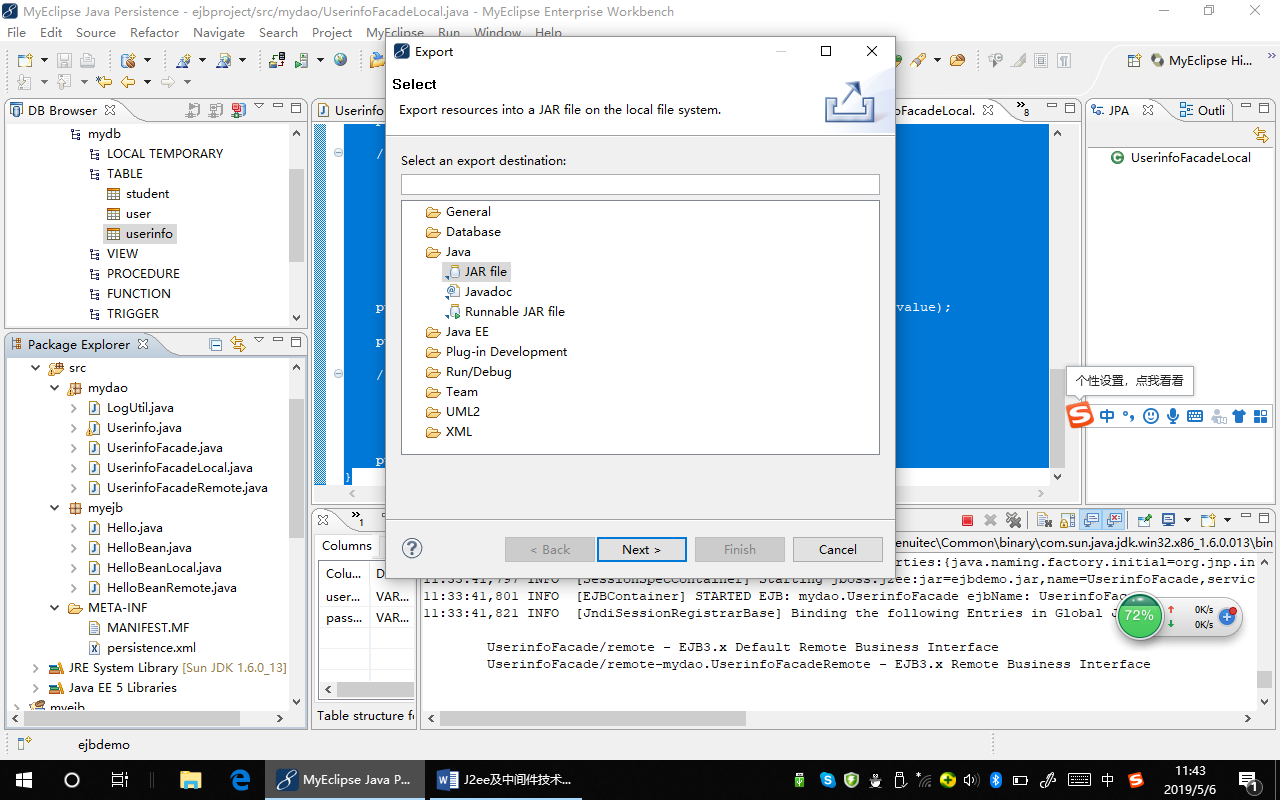
发布该工程。

下面创建客户端进行测试。

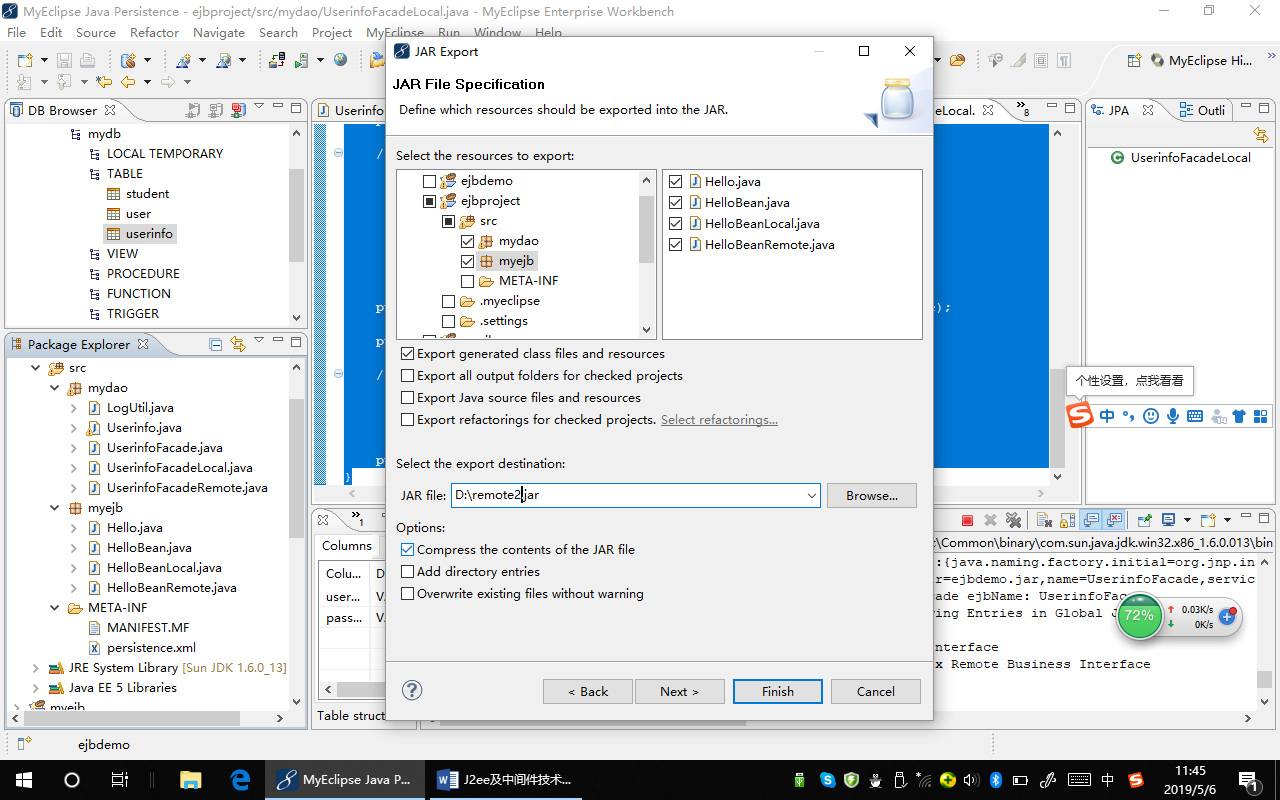
创建两个工程：Java Project和Web Project分别来测试EJBean的远程和本地接口。

在此之前，首先要把HelloBean的远程和本地接口及其相关接口如业务接口导出，从而在其他工程中使用该接口。

在Myeclipse下面，选择File-Export，找到Java-Jar File如下图所示：



点击Next，找到需要导出的类，在下面选择或输入要保存的jar文件的位置及文件名，然后点击Finish。



新建Java工程，然后新建一个Java类，包含main方法，在其中输入如下代码：

**import** java.util.Properties;

**import** javax.naming.Context;

**import** javax.naming.InitialContext;

**import** javax.naming.NamingException;

**import** myejb.HelloBeanRemote;

**public** **class** ejbclient {

/\*\*

\* **@param** args

\*/

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

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

Properties p = **new** Properties();

p.put("java.naming.factory.initial", "org.jnp.interfaces.NamingContextFactory");

p.put("java.naming.factory.url.pkgs", "org.jboss.naming:org.jnp.interfaces");

p.put("java.naming.provider.url", "localhost");

**try** {

Context ctx = **new** InitialContext(p);

HelloBeanRemote h = (HelloBeanRemote)ctx.lookup("HelloBean/remote");

System.*out*.println(h.sayHello("zhangsan"));

} **catch** (NamingException e) {

// **TODO** Auto-generated catch block

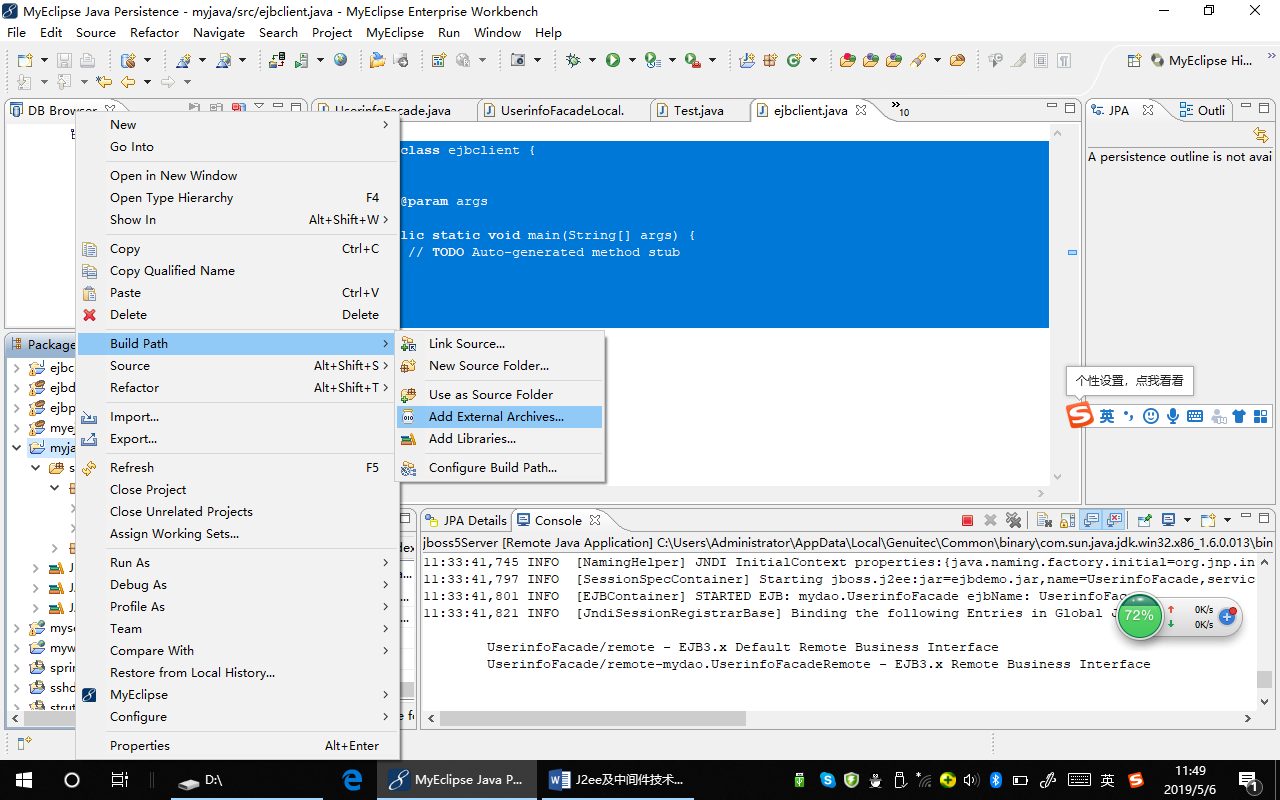
e.printStackTrace();

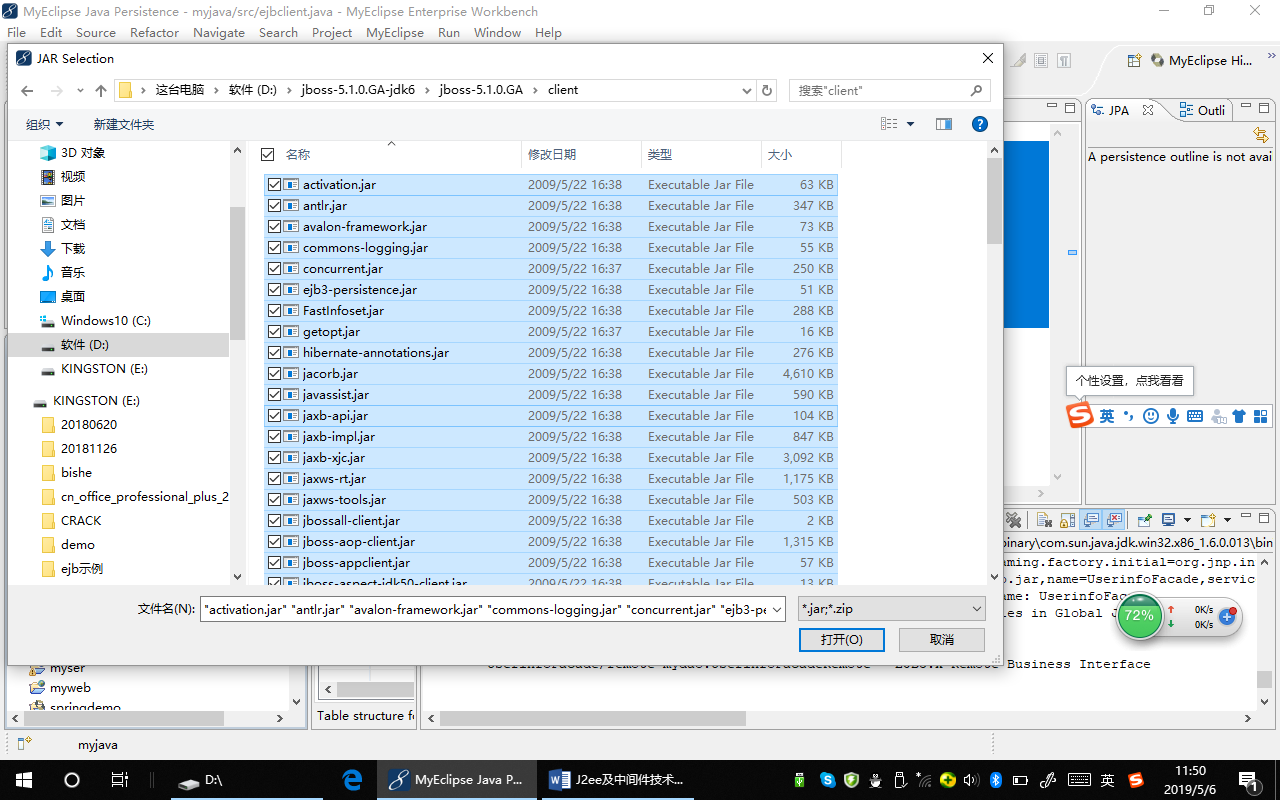
}

}

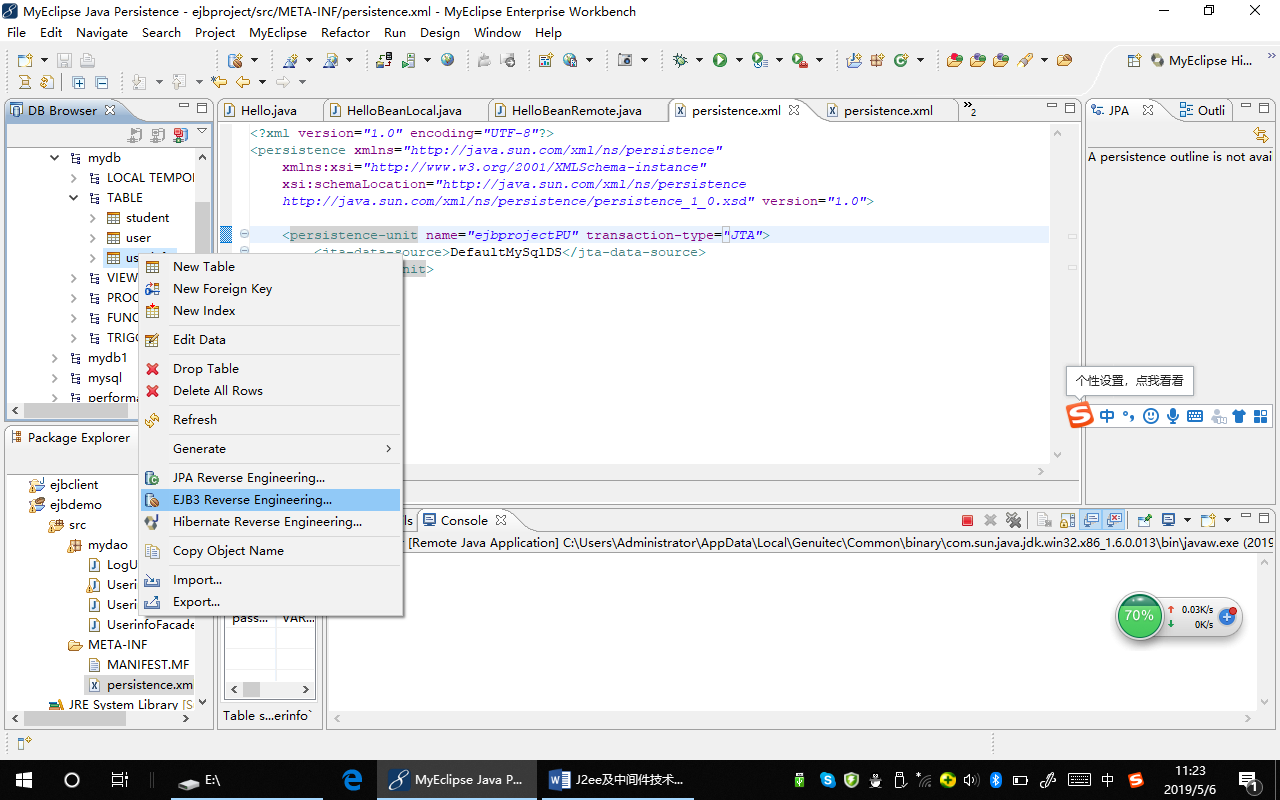
}

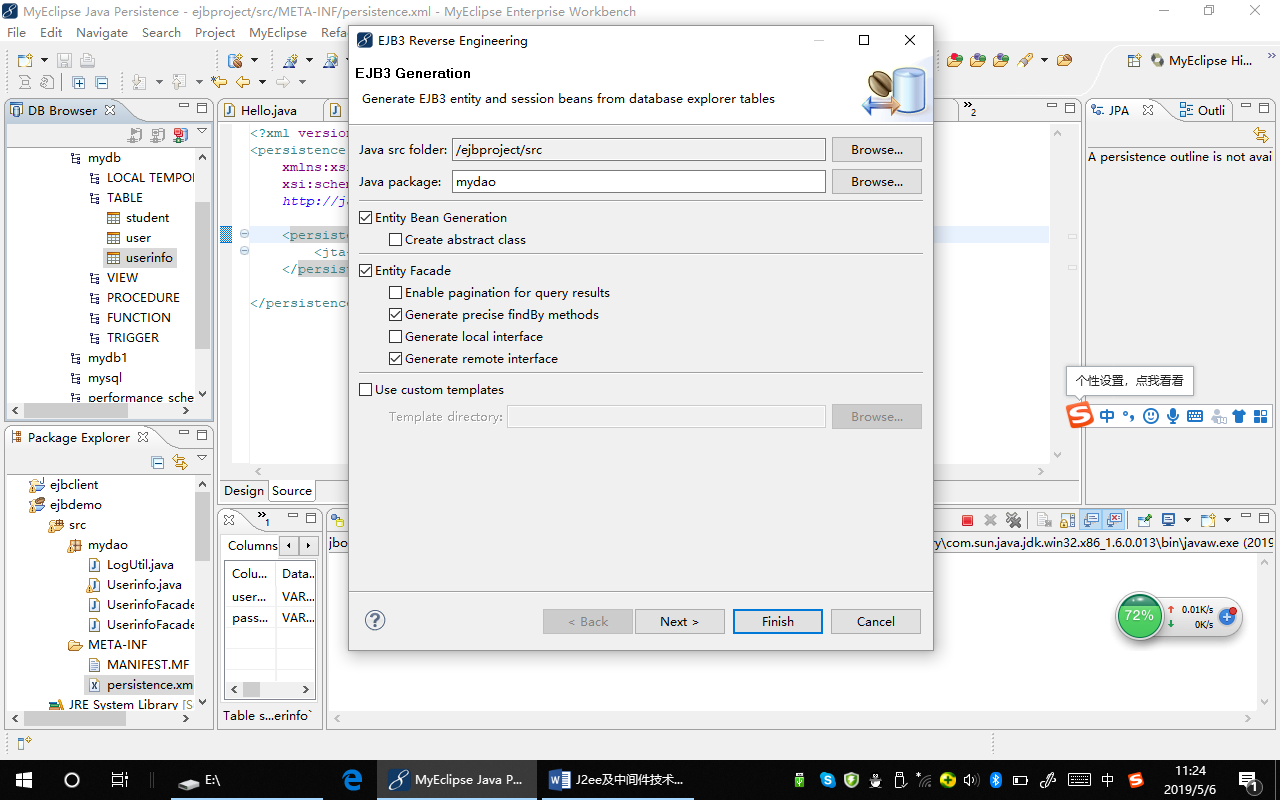
将上面导出的接口类包和JBoss的Client-jar文件导入到工程中。





Step3：使用数据库的逆向工具EJB3 Reverse Engineering导出Façade类，类似于JPA和Hibernate中的实体类和DAO类。从而在Servlet中进行访问。





注意勾选上面Entity Façade下面的Remote和Local接口。

产生的Userinfo类和前面JPA产生的一样，这里就不在展示代码了。

产生的UserinfoFacade相当于JPA中产生的DAO，代码如下：

package mydao;

import java.util.List;

import java.util.logging.Level;

import javax.ejb.Stateless;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import javax.persistence.Query;

/\*\*

\* Facade for entity Userinfo.

\*

\* @see mydao.Userinfo

\* @author MyEclipse Persistence Tools

\*/

@Stateless

public class UserinfoFacade implements UserinfoFacadeLocal,

UserinfoFacadeRemote {

// property constants

public static final String PASSWORD = "password";

@PersistenceContext

private EntityManager entityManager;

/\*\*

\* Perform an initial save of a previously unsaved Userinfo entity. All

\* subsequent persist actions of this entity should use the #update()

\* method.

\*

\* @param entity

\* Userinfo entity to persist

\* @throws RuntimeException

\* when the operation fails

\*/

public void save(Userinfo entity) {

LogUtil.log("saving Userinfo instance", Level.INFO, null);

try {

entityManager.persist(entity);

LogUtil.log("save successful", Level.INFO, null);

} catch (RuntimeException re) {

LogUtil.log("save failed", Level.SEVERE, re);

throw re;

}

}

/\*\*

\* Delete a persistent Userinfo entity.

\*

\* @param entity

\* Userinfo entity to delete

\* @throws RuntimeException

\* when the operation fails

\*/

public void delete(Userinfo entity) {

LogUtil.log("deleting Userinfo instance", Level.INFO, null);

try {

entity = entityManager.getReference(Userinfo.class, entity

.getUsername());

entityManager.remove(entity);

LogUtil.log("delete successful", Level.INFO, null);

} catch (RuntimeException re) {

LogUtil.log("delete failed", Level.SEVERE, re);

throw re;

}

}

/\*\*

\* Persist a previously saved Userinfo entity and return it or a copy of it

\* to the sender. A copy of the Userinfo entity parameter is returned when

\* the JPA persistence mechanism has not previously been tracking the

\* updated entity.

\*

\* @param entity

\* Userinfo entity to update

\* @return Userinfo the persisted Userinfo entity instance, may not be the

\* same

\* @throws RuntimeException

\* if the operation fails

\*/

public Userinfo update(Userinfo entity) {

LogUtil.log("updating Userinfo instance", Level.INFO, null);

try {

Userinfo result = entityManager.merge(entity);

LogUtil.log("update successful", Level.INFO, null);

return result;

} catch (RuntimeException re) {

LogUtil.log("update failed", Level.SEVERE, re);

throw re;

}

}

public Userinfo findById(String id) {

LogUtil.log("finding Userinfo instance with id: " + id, Level.INFO,

null);

try {

Userinfo instance = entityManager.find(Userinfo.class, id);

return instance;

} catch (RuntimeException re) {

LogUtil.log("find failed", Level.SEVERE, re);

throw re;

}

}

/\*\*

\* Find all Userinfo entities with a specific property value.

\*

\* @param propertyName

\* the name of the Userinfo property to query

\* @param value

\* the property value to match

\* @return List<Userinfo> found by query

\*/

@SuppressWarnings("unchecked")

public List<Userinfo> findByProperty(String propertyName, final Object value) {

LogUtil.log("finding Userinfo instance with property: " + propertyName

+ ", value: " + value, Level.INFO, null);

try {

final String queryString = "select model from Userinfo model where model."

+ propertyName + "= :propertyValue";

Query query = entityManager.createQuery(queryString);

query.setParameter("propertyValue", value);

return query.getResultList();

} catch (RuntimeException re) {

LogUtil.log("find by property name failed", Level.SEVERE, re);

throw re;

}

}

public List<Userinfo> findByPassword(Object password) {

return findByProperty(PASSWORD, password);

}

/\*\*

\* Find all Userinfo entities.

\*

\* @return List<Userinfo> all Userinfo entities

\*/

@SuppressWarnings("unchecked")

public List<Userinfo> findAll() {

LogUtil.log("finding all Userinfo instances", Level.INFO, null);

try {

final String queryString = "select model from Userinfo model";

Query query = entityManager.createQuery(queryString);

return query.getResultList();

} catch (RuntimeException re) {

LogUtil.log("find all failed", Level.SEVERE, re);

throw re;

}

}

}

基本和JPA产生的DAO类似，不同的是，这里将Façade类定义成了Stateless Session Bean，另外还产生了DAO的远程和本地接口，供远程和本地访问。

package mydao;

import java.util.List;

import javax.ejb.Local;

/\*\*

\* Local interface for UserinfoFacade.

\*

\* @author MyEclipse Persistence Tools

\*/

@Local

public interface UserinfoFacadeLocal {

/\*\*

\* Perform an initial save of a previously unsaved Userinfo entity. All

\* subsequent persist actions of this entity should use the #update()

\* method.

\*

\* @param entity

\* Userinfo entity to persist

\* @throws RuntimeException

\* when the operation fails

\*/

public void save(Userinfo entity);

/\*\*

\* Delete a persistent Userinfo entity.

\*

\* @param entity

\* Userinfo entity to delete

\* @throws RuntimeException

\* when the operation fails

\*/

public void delete(Userinfo entity);

/\*\*

\* Persist a previously saved Userinfo entity and return it or a copy of it

\* to the sender. A copy of the Userinfo entity parameter is returned when

\* the JPA persistence mechanism has not previously been tracking the

\* updated entity.

\*

\* @param entity

\* Userinfo entity to update

\* @return Userinfo the persisted Userinfo entity instance, may not be the

\* same

\* @throws RuntimeException

\* if the operation fails

\*/

public Userinfo update(Userinfo entity);

public Userinfo findById(String id);

/\*\*

\* Find all Userinfo entities with a specific property value.

\*

\* @param propertyName

\* the name of the Userinfo property to query

\* @param value

\* the property value to match

\* @return List<Userinfo> found by query

\*/

public List<Userinfo> findByProperty(String propertyName, Object value);

public List<Userinfo> findByPassword(Object password);

/\*\*

\* Find all Userinfo entities.

\*

\* @return List<Userinfo> all Userinfo entities

\*/

public List<Userinfo> findAll();

}

远程接口类似，只是把@Local改成了@Remote而已。

我们可以在前面创建的客户端工程和Web工程中访问该Façade，从而访问数据库。

例如：

**import** myejb.HelloBeanRemote;

**public** **class** ejbclient {

/\*\*

\* **@param** args

\*/

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

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

Properties p = **new** Properties();

p.put("java.naming.factory.initial", "org.jnp.interfaces.NamingContextFactory");

p.put("java.naming.factory.url.pkgs", "org.jboss.naming:org.jnp.interfaces");

p.put("java.naming.provider.url", "localhost");

**try** {

Context ctx = **new** InitialContext(p);

UserinfoFacadeRemote h = (UserinfoFacadeRemote)ctx.lookup("HelloBean/remote");

List<Userinfo> users = h.findAll();

for(Userinfo user:users)

System.*out*.println(user.getUsername()+”,”+user.getPassword());

} **catch** (NamingException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

注意：EJBean是由JavaEE服务器中的EJBean的容器进行管理的，所以不能交由Spring来同时管理。

Step4: 创建JSP网页和相应的Servlet

创建 添加用户页面—AddUser.jsp,代码如下：

<%@ page contentType="text/html; charset=gb2312" language="java" import="java.sql.\*" errorPage="" %>

<%@ taglib uri=”/strut-tags” prefix=”s”/>

<form method="post" action="AddUserServlet">

<div align="center"><font face="宋体" size="6"><strong>

添加用户</strong></font><br/><hr/>

<input name="username" label="用户名："/>

<br/>

<br/>

<input type=”password” name="password" label="密码："/>

<br />

<br />

<input type="submit" value="添加" />

<input type="reset" value="取消" />

</div>

</form>

利用向导创建success.jsp和error.jsp

其内容分别显示success和register error!

相应的添加用户Servlet(AddUserServlet.java),代码如下：

package servlet.user;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.SQLException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import toolBean.db.AccessUserFromDB;

import valueBean.User;

public class AddUserServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//返回上页

String goBack="<br><a href='javascript:window.history.go(-1);'>返回上页</a>";

response.setContentType("text/html;charset=gb2312");

PrintWriter out = response.getWriter();

String username=request.getParameter("username");

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

{

out.print("用户名不能为空！");

out.print(goBack);

}

String password=request.getParameter("password");

User user=new User();

user.setUsername(username);

user.setPassword(password); //注入属性

**try** {

Context ctx = **new** InitialContext();

UserinfoFacadeLocal h = (UserinfoFacadeLocal)ctx.lookup("UserinfoFacaed/local");

h.save(user);

response.sendRedirect(“success.jsp”);

} **catch** (NamingException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

response.sendRedirect(“error.jsp”);

}