## 实验七 Hibernate或MyBatis应用基础

### 一、实验目的

1、熟练掌握Hibernate或MyBatis配置、理解配置文件内容；

2、熟悉反向工程的概念，并能完成反向工程的相关ORMapping操作；

3、掌握Hibernate或MyBatis架构中实体操作与事务管理方法；

4、初步了解应用Hibernate或MyBatis架构技术完成数据库应用项目开发。

5、初步掌握Struts2或SpringMVC与Hibernate或MyBatis集成方法；

### 二、实验内容及实施步骤

1、实验重点难点：

重点：实体对象映射，实体关系映，MyBatis（Hibernate）实体对象映射实现方法，MyBatis（Hibernate）与Spring（Struts）集成方法；

难点：MyBatis（Hibernate）实体对象映射实现方法，MyBatis（Hibernate）与Spring（Struts）集成方法\*。

1. MyBatis的工作原理



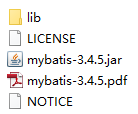
MyBatis本是apache的一个开源项目iBatis，2010年这个项目由apache software foundation迁移到了google code，并且改名为MyBatis。

MyBatis 是一个基于Java的持久层框架。MyBatis提供的持久层框架包括SQL Maps和Data Access Objects（DAO），它消除了几乎所有的JDBC代码和参数的手工设置以及结果集的检索。MyBatis 使用简单的 XML或注解用于配置和原始映射，将接口和Java的 POJOs（Plain Old Java Objects，普通的Java对象）映射成数据库中的记录。

1. 实验步骤

（1）MyBatis环境的构建

MyBatis的3.4.5版本可以通过“https://github.com/mybatis/mybatis-3/releases”网址下载。下载时只需选择mybatis-3.4.5.zip即可，解压后得到如图所示的目录。

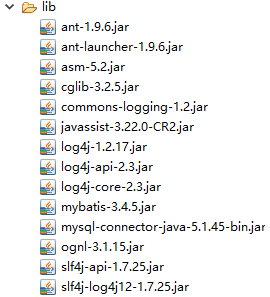


图中mybatis-3.4.5.jar是MyBatis的核心包，mybatis-3.4.5.pdf是MyBatis的使用手册，lib文件夹下的JAR是MyBatis的依赖包。

使用MyBatis框架时，需要将它的核心包和依赖包引入到应用程序中。如果是Web应用，只需将核心包和依赖包复制到/WEB-INF/lib目录中。

1. 创建Web应用，并添加相关JAR包

创建Javaweb应用项目，按下图所示添加相应的jar包



1. 创建日志文件

MyBatis默认使用log4j输出日志信息，如果开发者需要查看控制台输出的SQL语句，那么需要在classpath路径下配置其日志文件。在应用的src目录下创建log4j.properties文件，内容如下：

# Global logging configuration

log4j.rootLogger=ERROR, stdout

# MyBatis logging configuration...

log4j.logger.com.mybatis=DEBUG

# Console output...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

日志文件中配置了全局的日志配置、MyBatis的日志配置和控制台输出，其中MyBatis的日志配置用于将com.mybatis包下所有类的日志记录级别设置为DEBUG。该配置文件内容不需要开发者全部手写，可以从MyBatis使用手册中Logging小节复制，然后进行简单修改。

1. 创建持久化类

在src目录下，创建一个名为com.mybatis.po包，在该包中创建持久化类MyUser。类中声明的属性与数据表user（创建表的代码参见课程提供源代码ch7.sql）字段一致。

package com.mybatis.po;

/\*\*

\*springtest数据库中user表的持久化类

\*/

public class MyUser {

private Integer uid;//主键

private String uname;

private String usex;

public Integer getUid() {

return uid;

}

public void setUid(Integer uid) {

this.uid = uid;

}

public String getUname() {

return uname;

}

public void setUname(String uname) {

this.uname = uname;

}

public String getUsex() {

return usex;

}

public void setUsex(String usex) {

this.usex = usex;

}

@Override

public String toString() {

return "User [uid=" + uid +",uname=" + uname + ",usex=" + usex +"]";

}

}

1. 创建映射文件

在src目录下，创建一个名为com.mybatis.mapper包，在该包中创建映射文件UserMapper.xml。

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

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.mybatis.mapper.UserMapper">

<!-- 根据uid查询一个用户信息 -->

<select id="selectUserById" parameterType="Integer"

resultType="com.mybatis.po.MyUser">

select \* from user where uid = #{uid}

</select>

<!-- 查询所有用户信息 -->

<select id="selectAllUser" resultType="com.mybatis.po.MyUser">

select \* from user

</select>

<!-- 添加一个用户 ，#{uname}为com.mybatis.po.MyUser的属性值-->

<insert id="addUser" parameterType="com.mybatis.po.MyUser">

insert into user (uname,usex) values(#{uname},#{usex})

</insert>

<!-- 修改一个用户 -->

<update id="updateUser" parameterType="com.mybatis.po.MyUser">

update user set uname = #{uname},usex = #{usex} where uid = #{uid}

</update>

<!-- 删除一个用户 -->

<delete id="deleteUser" parameterType="Integer">

delete from user where uid = #{uid}

</delete>

</mapper>

上述映射文件中，<mapper>元素是配置文件的根元素，它包含了一个namespace属性，该属性值通常设置为“包名+SQL映射文件名”，指定了唯一的命名空间。子元素<select>、<insert>、<update>以及<delete>中的信息是用于执行查询、添加、修改以及删除操作的配置。在定义的SQL语句中，“#{}”表示一个占位符，相当于“?”，而“#{uid}”表示该占位符待接收参数的名称为uid。

1. 创建MyBatis的配置文件

在src目录下，创建MyBatis的核心配置文件mybatis-config.xml。在该文件中，配置了数据库环境和映射文件的位置。

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

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!-- 配置环境 -->

<environments default="development">

<environment id="development">

<!-- 使用JDBC的事务管理 -->

<transactionManager type="JDBC"/>

<dataSource type="POOLED">

<!-- MySQL数据库驱动 -->

<property name="driver" value="com.mysql.jdbc.Driver"/>

<!-- 连接数据库的URL -->

<property name="url" value="jdbc:mysql://localhost:3306/springtest?characterEncoding=utf8"/>

<property name="username" value="root"/>

<property name="password" value="1234"/>

</dataSource>

</environment>

</environments>

<mappers>

<!-- 映射文件的位置 -->

<mapper resource="com/mybatis/mapper/UserMapper.xml"/>

</mappers>

</configuration>

1. 创建测试类

package com.mybatis.test;

import java.io.IOException;

import java.io.InputStream;

import java.util.List;

import org.apache.ibatis.io.Resources;

import org.apache.ibatis.session.SqlSession;

import org.apache.ibatis.session.SqlSessionFactory;

import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import com.mybatis.po.MyUser;

public class MyBatisTest {

public static void main(String[] args) {

try {

//读取配置文件mybatis-config.xml

InputStream config = Resources.getResourceAsStream("mybatis-config.xml");

//根据配置文件构建SqlSessionFactory

SqlSessionFactory ssf = new SqlSessionFactoryBuilder().build(config);

//通过SqlSessionFactory创建SqlSession

SqlSession ss = ssf.openSession();

//SqlSession执行映射文件中定义的SQL，并返回映射结果

//com.mybatis.mapper.UserMapper.selectUserById为UserMapper.xml中的命名空间+select的id

//查询一个用户

MyUser mu = ss.selectOne("com.mybatis.mapper.UserMapper.selectUserById", 1);

System.out.println(mu);

//添加一个用户

MyUser addmu = new MyUser();

addmu.setUname("陈恒");

addmu.setUsex("男");

ss.insert("com.mybatis.mapper.UserMapper.addUser",addmu);

//修改一个用户

MyUser updatemu = new MyUser();

updatemu.setUid(1);

updatemu.setUname("张三");

updatemu.setUsex("女");

ss.update("com.mybatis.mapper.UserMapper.updateUser", updatemu);

//删除一个用户

ss.delete("com.mybatis.mapper.UserMapper.deleteUser", 3);

//查询所有用户

List<MyUser> listMu = ss.selectList("com.mybatis.mapper.UserMapper.selectAllUser");

for (MyUser myUser : listMu) {

System.out.println(myUser);

}

//提交事务

ss.commit();

//关闭SqlSession

ss.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

在src目录下，创建一个名为com.mybatis.test包，在该包中创建MyBatisTest测试类。在测试类中，首先使用输入流读取配置文件，然后根据配置信息构建SqlSessionFactory对象。接下来通过SqlSessionFactory对象创建SqlSession对象，并使用SqlSession对象的方法执行数据库操作。

**（7）扩展练习：**

**A、在上述代码基础上，设计相应的JSP、Servlet和JavaBean，实现基于网页的用户信息查询、删除、修改和添加4个页面及对应的功能模块；**

**B、在A步骤基础上，使用Spring架构，实现对JavaBean的控制反转，改写上一步的代码，实现同样的功能。**

1. MyBatis与Spring框架的整合开发
2. 导入相关JAR包

A、MyBatis框架所需的JAR包：包括它的核心包和依赖包

B、Spring框架所需的JAR包：包括它的核心模块JAR、AOP开发使用的JAR、JDBC和事务的JAR包（其中依赖包不需要再导入，因为MyBatis已提供），具体如下：

aopalliance-1.0.jar

aspectjweaver-1.7.3.13.jar

spring-aop-5.0.2.RELEASE.jar

spring-aspects-5.0.2.RELEASE.jar

spring-beans-5.0.2.RELEASE.jar

spring-context-5.0.2.RELEASE.jar

spring-core-5.0.2.RELEASE.jar

spring-expression-5.0.2.RELEASE.jar

spring-jdbc-5.0.2.RELEASE.jar

spring-tx-5.0.2.RELEASE.jar

C、MyBatis与Spring整合的中间JAR包：中间JAR包的版本为mybatis-spring-1.3.1.jar。此版本可从地址“http://mvnrepository.com/artifact/org.mybatis/mybatis-spring/1.3.1”下载。

D、数据库驱动JAR包：MySQL数据库驱动包为mysql-connector-java-5.1.45-bin.jar

E、数据源所需的JAR包：整合时使用的是DBCP数据源，需要准备DBCP和连接池的JAR包。DBCP的JAR包为commons-dbcp2-2.2.0.jar，可从地址“http://commons. apache.org/proper/commons-dbcp/download\_dbcp.cgi”下载；连接池的JAR包为commons -pool2-2.5.0.jar，可从地址“http://commons.apache.org/proper/commons-pool/download\_poo l.cgi”下载。

（2）在Spring中配置MyBatis工厂

通过与Spring的整合，MyBatis的SessionFactory交由Spring来构建。构建时需要在Spring的配置文件中添加如下代码：

<!-- 配置数据源 -->

<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver" />

<property name="url" value="jdbc:mysql://localhost:3306/springtest?characterEncoding=utf8" />

<property name="username" value="root" />

<property name="password" value="root" />

…….

</bean>

<!-- 配置MyBatis工厂，同时指定数据源，并与MyBatis完美整合 -->

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<!-- configLocation的属性值为MyBatis的核心配置文件 -->

<property name="configLocation" value="classpath:com/mybatis/mybatis-config.xml"/>

</bean>

1. 使用Spring管理MyBatis的数据操作接口

使用Spring管理MyBatis的数据操作接口的方式有多种。其中，最常用最简洁的一种是基于MapperScannerConfigurer的整合。该方式需要在Spring的配置文件中加入以下内容：

<!--Mapper代理开发，使用Spring自动扫描MyBatis的接口并装配

（Spring将指定包中所有被@Mapper注解标注的接口自动装配为MyBatis的映射接口） -->

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

<!-- mybatis-spring组件的扫描器。com.dao只需要接口（接口方法与SQL映射文件中相同）-->

<property name="basePackage" value="com.dao"/>

<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>

</bean>

1. 框架整合示例

A、创建应用并导入相关JAR包：创建一个名为JavaEECH06-withSpring的Web应用，并将JAR导入到/ WEB-INF/lib目录下。

B、创建持久化类：在src目录下，创建一个名为com.po的包，将持久化类复制到包中。

MyUser.java

package com.po;

/\*\*

\*springtest数据库中user表的持久化类

\*/

public class MyUser {

private Integer uid;//主键

private String uname;

private String usex;

public Integer getUid() {

return uid;

}

public void setUid(Integer uid) {

this.uid = uid;

}

public String getUname() {

return uname;

}

public void setUname(String uname) {

this.uname = uname;

}

public String getUsex() {

return usex;

}

public void setUsex(String usex) {

this.usex = usex;

}

@Override

public String toString() {

return "User [uid=" + uid +",uname=" + uname + ",usex=" + usex +"]";

}

}

C、创建SQL映射文件和MyBatis核心配置文件：在src目录下，创建一个名为com. mybatis的包，在该包中创建MyBatis核心配置文件mybatis-config.xml和SQL映射文件UserMapper.xml。

mybatis-config.xml

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

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!-- 告诉 MyBatis到哪里去找映射文件 -->

<mappers>

<mapper resource="com/mybatis/UserMapper.xml"/>

</mappers>

</configuration>

UserMapper.xml

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

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.dao.UserDao">

<!-- 根据uid查询一个用户信息 -->

<select id="selectUserById" parameterType="Integer"

resultType="com.po.MyUser">

select \* from user where uid = #{uid}

</select>

<!-- 查询所有用户信息 -->

<select id="selectAllUser" resultType="com.po.MyUser">

select \* from user

</select>

<!-- 添加一个用户 ，#{uname}为com.po.MyUser的属性值-->

<insert id="addUser" parameterType="com.po.MyUser">

insert into user (uname,usex) values(#{uname},#{usex})

</insert>

<!-- 修改一个用户 -->

<update id="updateUser" parameterType="com.po.MyUser">

update user set uname = #{uname},usex = #{usex} where uid = #{uid}

</update>

<!-- 删除一个用户 -->

<delete id="deleteUser" parameterType="Integer">

delete from user where uid = #{uid}

</delete>

</mapper>

1. 创建数据访问接口：在src目录下，创建一个名为com.dao的包，在该包中创建UserDao接口，并将接口使用@Mapper注解为Mapper，接口中的方法与SQL映射文件一致。

UserDao.java：

package com.dao;

import java.util.List;

import org.apache.ibatis.annotations.Mapper;

import org.springframework.stereotype.Repository;

import com.po.MyUser;

@Repository("userDao")

@Mapper

/\*使用Spring自动扫描MyBatis的接口并装配

（Spring将指定包中所有被@Mapper注解标注的接口自动装配为MyBatis的映射接口\*/

public interface UserDao {

/\*\*

\* 接口方法对应SQL映射文件UserMapper.xml中的id

\*/

public MyUser selectUserById(Integer uid);

public List<MyUser> selectAllUser();

public int addUser(MyUser user);

public int updateUser(MyUser user);

public int deleteUser(Integer uid);

}

E、创建日志文件：在src目录下，创建日志文件log4j.properties，文件内容如下：

# Global logging configuration

log4j.rootLogger=ERROR, stdout

# MyBatis logging configuration...

log4j.logger.com.dao=DEBUG

# Console output...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

1. 创建控制层：在src目录下，创建一个名为com.controller的包，在包中创建UserController类，在该类中调用数据访问接口中的方法。

UserController.java

package com.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import com.dao.UserDao;

import com.po.MyUser;

@Controller("userController")

public class UserController {

@Autowired

private UserDao userDao;

public void test() {

//查询一个用户

MyUser auser = userDao.selectUserById(1);

System.out.println(auser);

System.out.println("================");

//添加一个用户

MyUser addmu = new MyUser();

addmu.setUname("陈恒");

addmu.setUsex("男");

int add = userDao.addUser(addmu);

System.out.println("添加了" + add + "条记录");

System.out.println("================");

//修改一个用户

MyUser updatemu = new MyUser();

updatemu.setUid(1);

updatemu.setUname("张三");

updatemu.setUsex("女");

int up = userDao.updateUser(updatemu);

System.out.println("修改了" + up + "条记录");

System.out.println( "================");

//删除一个用户

int dl = userDao.deleteUser(1);

System.out.println("删除了" + dl + "条记录");

System.out.println("================");

//查询所有用户

List<MyUser> list = userDao.selectAllUser();

for (MyUser myUser : list) {

System.out.println(myUser);

}

}

}

G、创建Spring的配置文件：在src目录下，创建配置文件applicationContext.xml。在配置文件中配置数据源、MyBatis工厂以及Mapper代理开发等信息。

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx.xsd">

<!-- 指定需要扫描的包（包括子包），使注解生效 -->

<context:component-scan base-package="com.dao"/>

<context:component-scan base-package="com.controller"/>

<!-- 配置数据源 -->

<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver" />

<property name="url" value="jdbc:mysql://localhost:3306/springtest?characterEncoding=utf8" />

<property name="username" value="root" />

<property name="password" value="1234" />

<!-- 最大连接数 -->

<property name="maxTotal" value="30"/>

<!-- 最大空闲连接数 -->

<property name="maxIdle" value="10"/>

<!-- 初始化连接数 -->

<property name="initialSize" value="5"/>

</bean>

<!-- 添加事务支持 -->

<bean id="txManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource" />

</bean>

<!-- 开启事务注解-->

<tx:annotation-driven transaction-manager="txManager" />

<!-- 配置MyBatis工厂，同时指定数据源，并与MyBatis完美整合 -->

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<!-- configLocation的属性值为MyBatis的核心配置文件 -->

<property name="configLocation" value="classpath:com/mybatis/mybatis-config.xml"/>

</bean>

<!--Mapper代理开发，使用Spring自动扫描MyBatis的接口并装配

（Spring将指定包中所有被@Mapper注解标注的接口自动装配为MyBatis的映射接口） -->

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

<!-- mybatis-spring组件的扫描器 -->

<property name="basePackage" value="com.dao"/>

<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>

</bean>

</beans>

1. 创建测试类

在包com.controller中，创建测试类TestController，代码如下：

package com.controller;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestController {

public static void main(String[] args) {

ApplicationContext appCon = new ClassPathXmlApplicationContext("applicationContext.xml");

UserController uc = (UserController)appCon.getBean("userController");

uc.test();

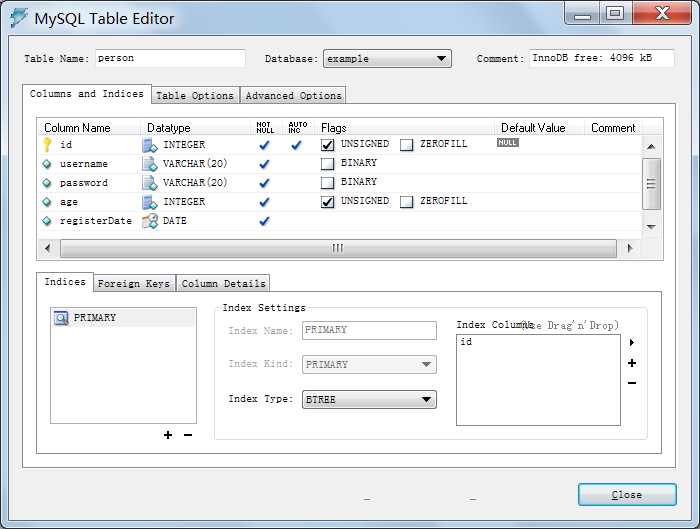
}

}

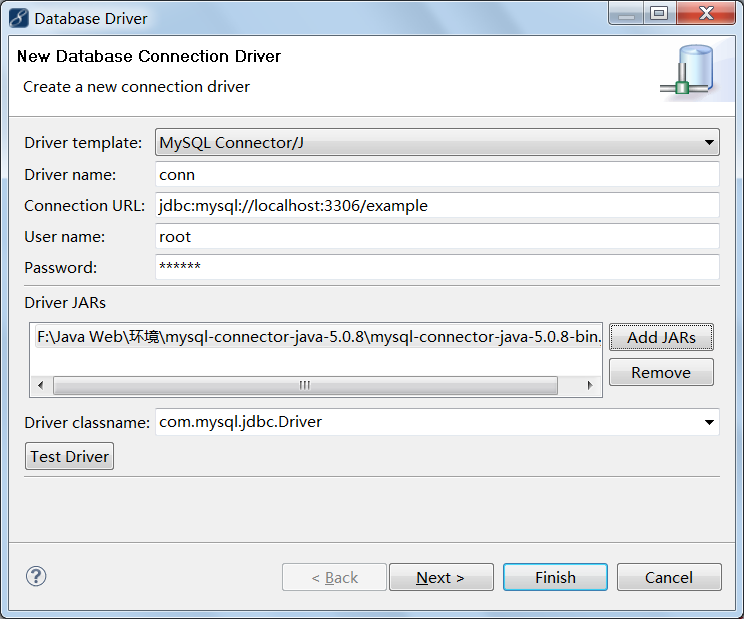
### 三、附录

1、Hibernate编程示例：

（1）建立数据库及表(数据库example,表person)

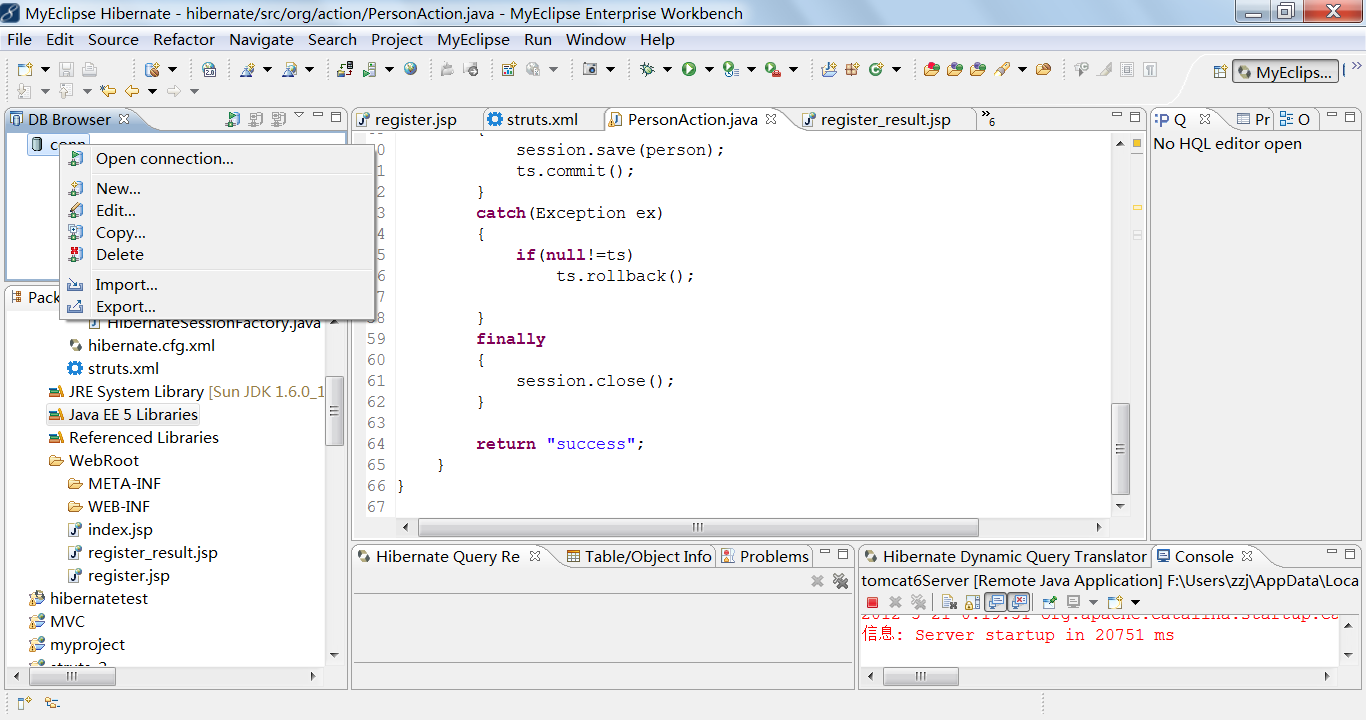
（2）在MyEclipse中创建对myeclipse 的连接

启动MyEclipse，选择【Window】→【Open Perspective】→【MyEclipse Database Explorer】菜单项，打开MyEclipse Database浏览器，右击菜单，选择【New…】菜单项，出现如图所示的对话框，编辑数据库连接驱动。



点击finish.

编辑完成以后，在MyEclipse Database浏览器中，右击刚才创建的conn数据库连接，选择“Open connection…”菜单项，打开名为“conn”的数据连接，



（3） 打开myeclipse.，建立一个web project. 命名为hibernate.

（4）添加struts类库到工程中(参考实验七)

（5）拷贝以下内容到工程中的web.xml中的<web-app>此处</web-app>中间：

<filter>

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

<filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>

</filter>

<filter-mapping>

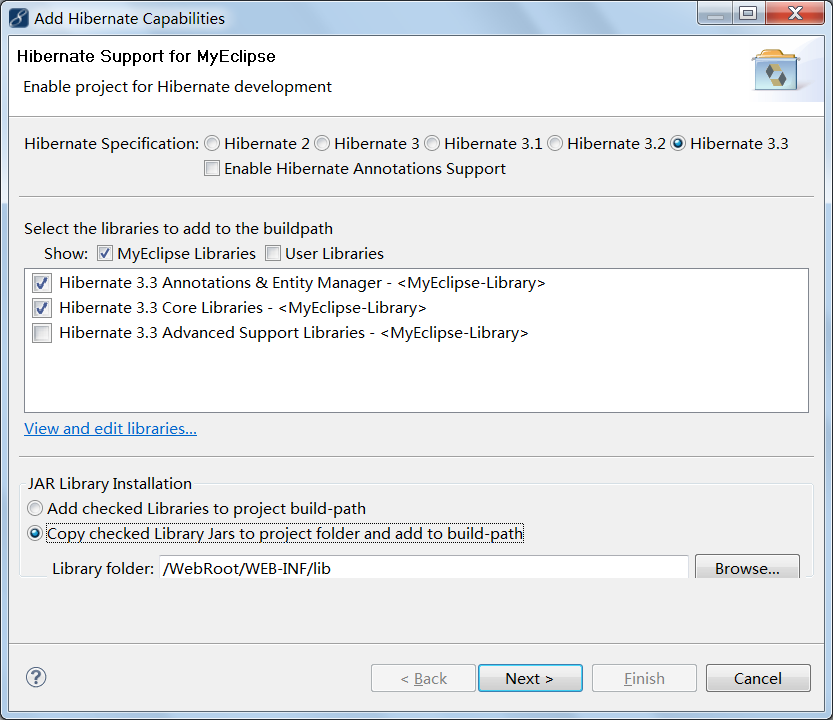
<filter-name>struts2</filter-name>

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

</filter-mapping>

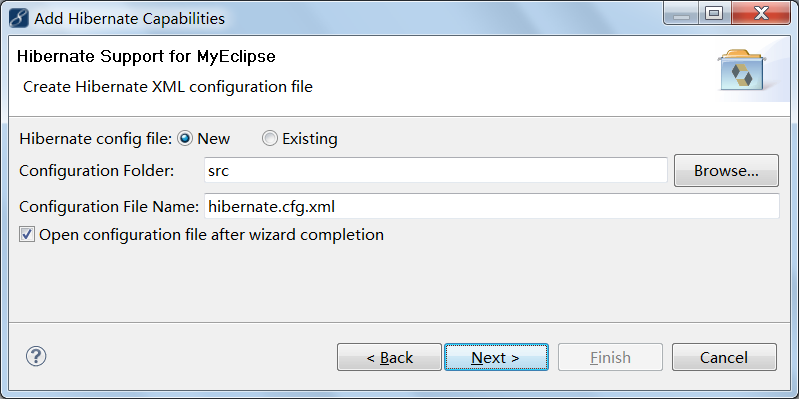
（6）添加Hibernate开发能力

右击项目名hibernate，选择【MyEclipse】→【Add Hibernate Capabilites】菜单项，出现如图所示的对话框，选择Hibernate框架应用版本及所需要的类库。

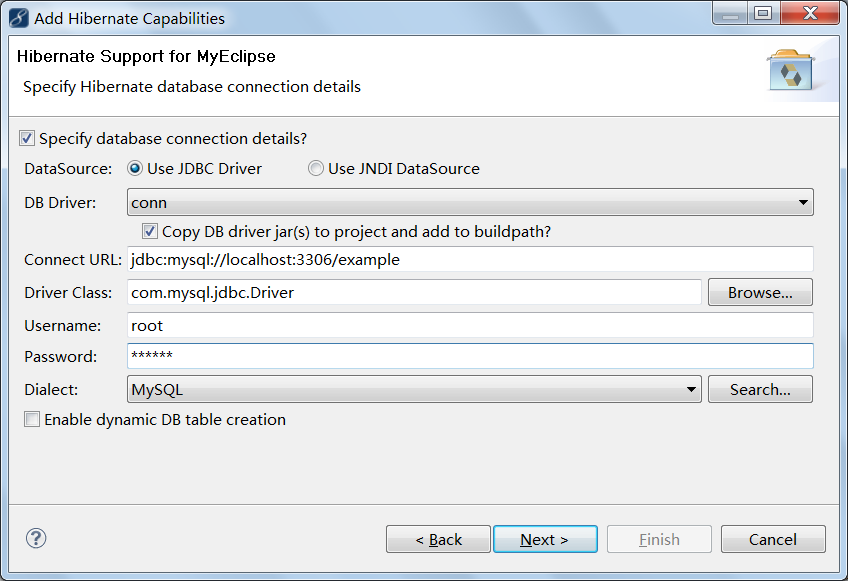


单击【Next】按钮，进入如图

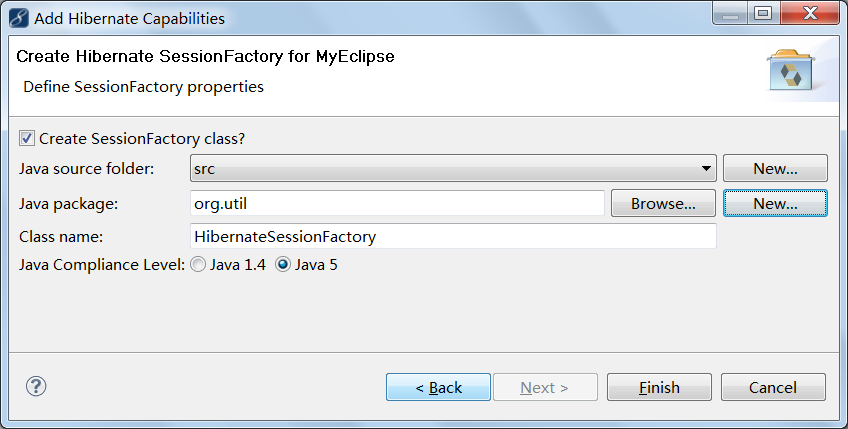
所示界面。创建Hibernate配置文件hibernate.cfg.xml，将该文件放在src文件夹下



单击【Next】按钮，进入如图所示界面，指定Hibernate数据库连接细节。由于在前面已经配置一个名为conn的数据库连接，所以这里只需要选择DB Driver为“conn”即可。

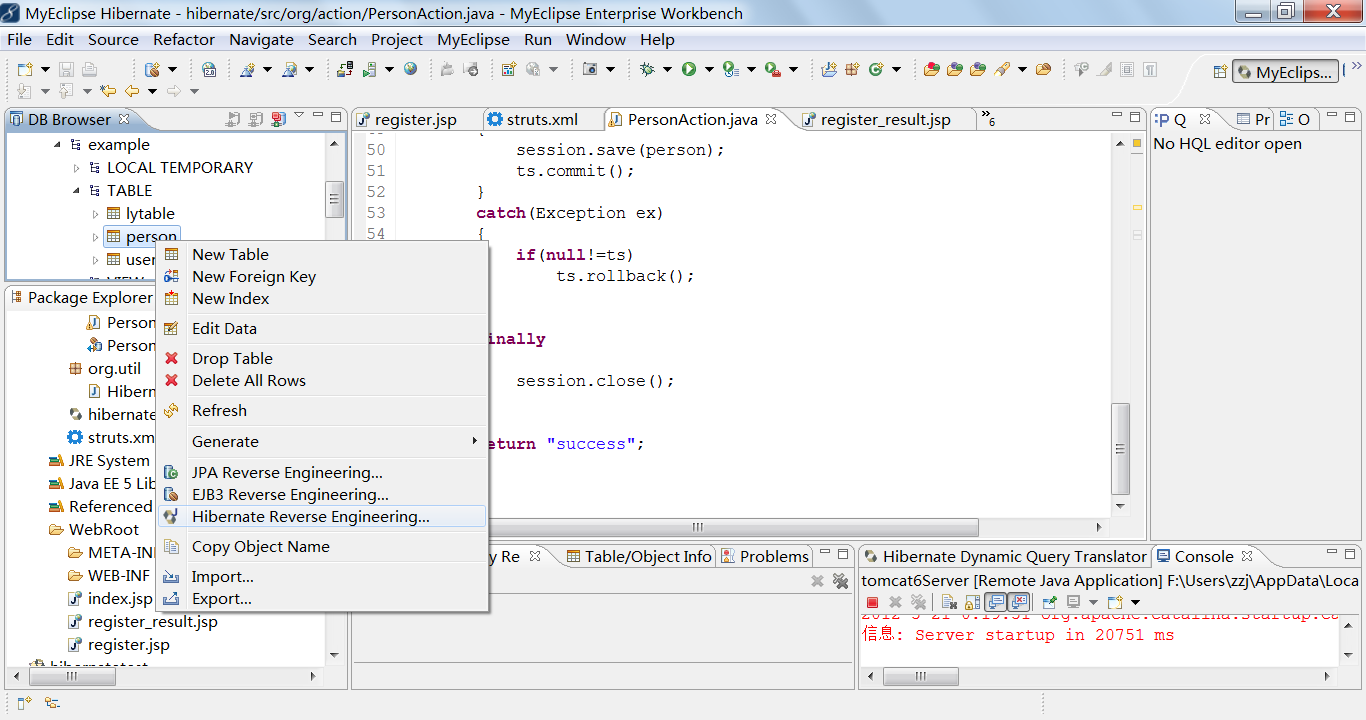


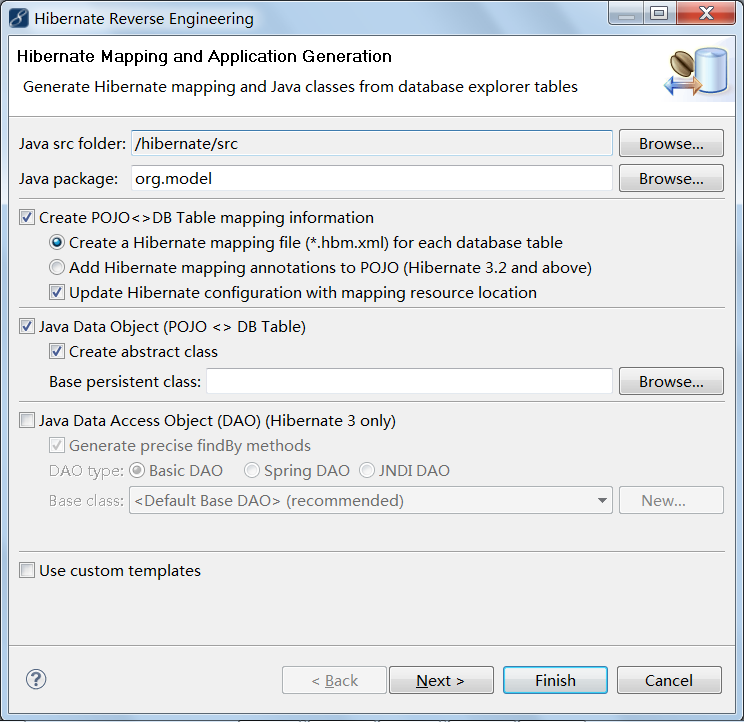
单击【Next】按钮，出现如图所示界面。Hibernate中有一个与数据库打交道重要的类Session。而这个类是由工厂SessionFactory创建的。这个界面询问是否需要创建SessionFactory类。如果需要创建，还需要指定创建的位置和类名。单击【Finish】按钮，完成Hibernate的配置。

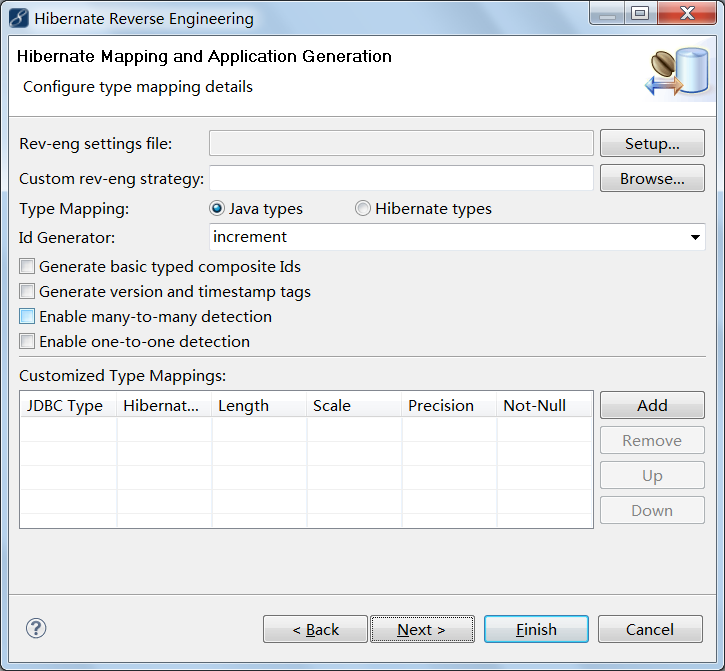


（7）生成数据库表对应的Java类对象和映射文件

首先在src下创建一个名为“org.model”的包，这个包将用来存放与数据库表对应的Java类POJO。在MyEclipse Database Explorer视图中选择表 person, 右击person表，选择【Hibernate Reverse Engineering…】菜单项，将启动Hibernate Reverse Engineering向导，该向导用于完成从已有的数据库表生成对应的Java类和相关映像文件的配置工作。







（8）在webRoot下建立两个文件register.jsp,register\_result.jsp。

可直接将所给代码中的jsp拷贝到自己工程下的webroot下

(1) register.jsp代码：

<%@ page language="java" import="java.util.\*" pageEncoding="GBK"%>

<html>

<body>

<form action="savePerson.action" method="post">

username: <input type="text" name="username"><br>

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

age:<input type="text" name="age"/><br>

<input type="submit" value="submit">&nbsp;&nbsp;&nbsp;&nbsp;

</form> <br>

</body>

</html>

(2) register\_result.jsp代码：

<%@ page language="java" import="java.util.\*" pageEncoding="GBK"%>

<html>

<body>

注册成功<br>

</body>

<html>

（9）在src目录下建立struts.xml文件.

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

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<package name="hibernate" extends="struts-default">

<action name="savePerson" class="org.action.PersonAction" method="savePerson">

<result name="success">register\_result.jsp</result>

</action>

</package>

</struts>

（10） 在src目录下建立包：org.action，在包下建立PersonAction.java

代码如下：

package org.action;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.model.Person;

import org.util.HibernateSessionFactory;

import com.opensymphony.xwork2.ActionSupport;

public class PersonAction extends ActionSupport

{

private String username;

private String password;

private Integer age;

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public Integer getAge() {

return age;

}

public void setAge(Integer age) {

this.age = age;

}

public String savePerson()

{

Person person=new Person();

person.setUsername(username);

person.setPassword(password);

person.setAge(age);

java.sql.Date registerDate=new java.sql.Date(new java.util.Date().getTime());

person.setRegisterDate(registerDate);

Session session=HibernateSessionFactory.getSession();

Transaction ts=session.beginTransaction();

try

{

session.save(person);

ts.commit();

}

catch(Exception ex)

{

if(null!=ts)

ts.rollback();

}

finally

{

session.close();

}

return "success";

}

}

（11）部署程序到tomcat下，运行tomcat.

打开浏览器，输入链接:

http://localhost:8080/hibernate/register.jsp