**一、Mybatis介绍**

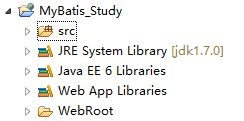


　　MyBatis是一个支持***普通SQL查询***，***存储过程***和***高级映射***的优秀**持久层框架**。MyBatis消除了几乎所有的JDBC代码和参数的手工设置以及对结果集的检索封装。MyBatis可以使用简单的**XML或注解**用于配置和原始映射，将接口和Java的**POJO**（Plain Old Java Objects，普通的Java对象）映射成数据库中的记录。

**二、mybatis快速入门**

**2.1、准备开发环境**

 1、创建测试项目，普通java项目或者是JavaWeb项目均可，如下图所示：

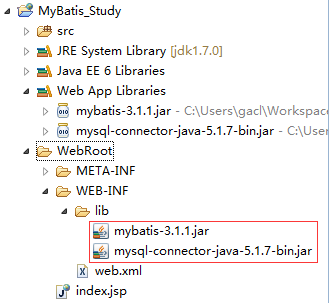


2、添加相应的jar包

**【mybatis】**

       　　mybatis-3.1.1.jar

**【MYSQL驱动包】**  
　　　　mysql-connector-java-5.1.7-bin.jar



3、创建数据库和表，针对MySQL数据库

　　SQL脚本如下：

1createdatabase mybatis;

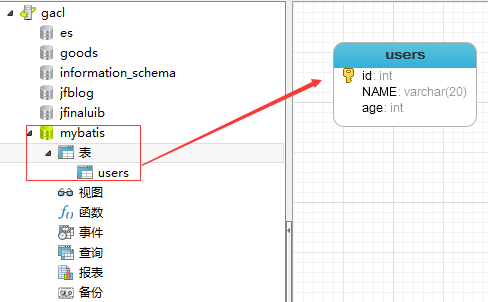
2use mybatis;

3CREATETABLE users(id INTPRIMARYKEY AUTO\_INCREMENT, NAME VARCHAR(**20**), age INT);

4INSERTINTO users(NAME, age) VALUES('孤傲苍狼', **27**);

5INSERTINTO users(NAME, age) VALUES('白虎神皇', **27**);

　　将SQL脚本在MySQL数据库中执行，完成创建数据库和表的操作，如下：

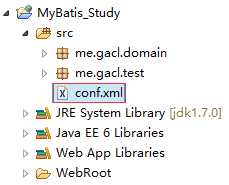


　　到此，前期的开发环境准备工作全部完成。

**2.2、使用MyBatis查询表中的数据**

**1、添加Mybatis的配置文件conf.xml**

　　在src目录下创建一个conf.xml文件，如下图所示：



　　conf.xml文件中的内容如下：

[复制代码](javascript:void(0);)

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

2<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">

3<configuration>

4<environments default="development">

5<environment id="development">

6<transactionManager type="JDBC"/>

7<!-- 配置数据库连接信息 -->

8<dataSource type="POOLED">

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

10<property name="url" value="jdbc:mysql://localhost:3306/mybatis"/>

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

12<property name="password" value="XDP"/>

13</dataSource>

14</environment>

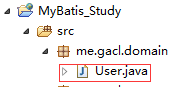
15</environments>

16

17</configuration>

[复制代码](javascript:void(0);)

**2、定义表所对应的实体类，如下图所示：**

****

　　User类的代码如下：

[复制代码](javascript:void(0);)

1package me.gacl.domain;

2

3/\*\*

4 \* @author gacl

5 \* users表所对应的实体类

6\*/

7publicclass User {

8

9//实体类的属性和表的字段名称一一对应

10privateint id;

11private String name;

12privateint age;

13

14publicint getId() {

15return id;

16 }

17

18publicvoid setId(int id) {

19this.id = id;

20 }

21

22public String getName() {

23return name;

24 }

25

26publicvoid setName(String name) {

27this.name = name;

28 }

29

30publicint getAge() {

31return age;

32 }

33

34publicvoid setAge(int age) {

35this.age = age;

36 }

37

38 @Override

39public String toString() {

40return "User [id=" + id + ", name=" + name + ", age=" + age + "]";

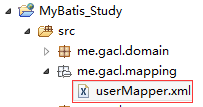
41 }

42 }

[复制代码](javascript:void(0);)

　　3、定义操作users表的sql映射文件userMapper.xml

　　创建一个me.gacl.mapping包，专门用于存放sql映射文件，在包中创建一个userMapper.xml文件，如下图所示：



　　userMapper.xml文件的内容如下：

[复制代码](javascript:void(0);)

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

2<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

3<!-- 为这个mapper指定一个唯一的namespace，namespace的值习惯上设置成包名+sql映射文件名，这样就能够保证namespace的值是唯一的

4例如namespace="me.gacl.mapping.userMapper"就是me.gacl.mapping(包名)+userMapper(userMapper.xml文件去除后缀)

5-->

6<mapper namespace="me.gacl.mapping.userMapper">

7<!-- 在select标签中编写查询的SQL语句， 设置select标签的id属性为getUser，id属性值必须是唯一的，不能够重复

8 使用parameterType属性指明查询时使用的参数类型，resultType属性指明查询返回的结果集类型

9 resultType="me.gacl.domain.User"就表示将查询结果封装成一个User类的对象返回

10 User类就是users表所对应的实体类

11-->

12<!--

13 根据id查询得到一个user对象

14-->

15<select id="getUser" parameterType="int"

16 resultType="me.gacl.domain.User">

17 select \* from users where id=#{id}

18</select>

19</mapper>

[复制代码](javascript:void(0);)

　　4、在conf.xml文件中注册userMapper.xml文件

[复制代码](javascript:void(0);)

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

2<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">

3<configuration>

4<environments default="development">

5<environment id="development">

6<transactionManager type="JDBC"/>

7<!-- 配置数据库连接信息 -->

8<dataSource type="POOLED">

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

10<property name="url" value="jdbc:mysql://localhost:3306/mybatis"/>

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

12<property name="password" value="XDP"/>

13</dataSource>

14</environment>

15</environments>

16

17<mappers>

18<!-- 注册userMapper.xml文件，

19 userMapper.xml位于me.gacl.mapping这个包下，所以resource写成me/gacl/mapping/userMapper.xml-->

20<mapper resource="me/gacl/mapping/userMapper.xml"/>

21</mappers>

22

23</configuration>

[复制代码](javascript:void(0);)

　　5、编写测试代码：执行定义的select语句

　　创建一个Test1类，编写如下的测试代码：

[复制代码](javascript:void(0);)

package me.gacl.test;

import java.io.IOException;

import java.io.InputStream;

import java.io.Reader;

import me.gacl.domain.User;

import org.apache.ibatis.io.Resources;

import org.apache.ibatis.session.SqlSession;

import org.apache.ibatis.session.SqlSessionFactory;

import org.apache.ibatis.session.SqlSessionFactoryBuilder;

publicclass Test1 {

publicstaticvoid main(String[] args) throws IOException {

//mybatis的配置文件

String resource = "conf.xml";

//使用类加载器加载mybatis的配置文件（它也加载关联的映射文件）

InputStream is = Test1.class.getClassLoader().getResourceAsStream(resource);

//构建sqlSession的工厂

SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder().build(is);

//使用MyBatis提供的Resources类加载mybatis的配置文件（它也加载关联的映射文件）

//Reader reader = Resources.getResourceAsReader(resource);

//构建sqlSession的工厂

//SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder().build(reader);

//创建能执行映射文件中sql的sqlSession

SqlSession session = sessionFactory.openSession();

/\*\*

\* 映射sql的标识字符串，

\* me.gacl.mapping.userMapper是userMapper.xml文件中mapper标签的namespace属性的值，

\* getUser是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL

\*/

String statement = "me.gacl.mapping.userMapper.getUser";//映射sql的标识字符串

//执行查询返回一个唯一user对象的sql

User user = session.selectOne(statement, 1);

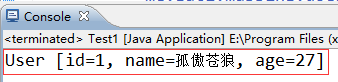
System.out.println(user);

}

}

[复制代码](javascript:void(0);)

　　执行结果如下：



　　可以看到，数据库中的记录已经成功查询出来了。

**一、使用MyBatis对表执行CRUD操作——基于XML的实现**

1、定义sql映射xml文件

　　userMapper.xml文件的内容如下：

[复制代码](javascript:void(0);)

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

2<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

3<!-- 为这个mapper指定一个唯一的namespace，namespace的值习惯上设置成包名+sql映射文件名，这样就能够保证namespace的值是唯一的

4例如namespace="me.gacl.mapping.userMapper"就是me.gacl.mapping(包名)+userMapper(userMapper.xml文件去除后缀)

5-->

6<mapper namespace="me.gacl.mapping.userMapper">

7<!-- 在select标签中编写查询的SQL语句， 设置select标签的id属性为getUser，id属性值必须是唯一的，不能够重复

8 使用parameterType属性指明查询时使用的参数类型，resultType属性指明查询返回的结果集类型

9 resultType="me.gacl.domain.User"就表示将查询结果封装成一个User类的对象返回

10 User类就是users表所对应的实体类

11-->

12<!--

13 根据id查询得到一个user对象

14-->

15<select id="getUser" parameterType="int"

16 resultType="me.gacl.domain.User">

17 select \* from users where id=#{id}

18</select>

19

20<!-- 创建用户(Create) -->

21<insert id="addUser" parameterType="me.gacl.domain.User">

22 insert into users(name,age) values(#{name},#{age})

23</insert>

24

25<!-- 删除用户(Remove) -->

26<delete id="deleteUser" parameterType="int">

27 delete from users where id=#{id}

28</delete>

29

30<!-- 修改用户(Update) -->

31<update id="updateUser" parameterType="me.gacl.domain.User">

32 update users set name=#{name},age=#{age} where id=#{id}

33</update>

34

35<!-- 查询全部用户-->

36<select id="getAllUsers" resultType="me.gacl.domain.User">

37 select \* from users

38</select>

39

40</mapper>

[复制代码](javascript:void(0);)

　　单元测试类代码如下：

[复制代码](javascript:void(0);)

1package me.gacl.test;

2

3import java.util.List;

4import me.gacl.domain.User;

5import me.gacl.util.MyBatisUtil;

6import org.apache.ibatis.session.SqlSession;

7import org.junit.Test;

8

9publicclass TestCRUDByXmlMapper {

10

11 @Test

12publicvoid testAdd(){

13//SqlSession sqlSession = MyBatisUtil.getSqlSession(false);

14 SqlSession sqlSession = MyBatisUtil.getSqlSession(true);

15/\*\*

16 \* 映射sql的标识字符串，

17 \* me.gacl.mapping.userMapper是userMapper.xml文件中mapper标签的namespace属性的值，

18 \* addUser是insert标签的id属性值，通过insert标签的id属性值就可以找到要执行的SQL

19\*/

20 String statement = "me.gacl.mapping.userMapper.addUser";//映射sql的标识字符串

21 User user = new User();

22 user.setName("用户孤傲苍狼");

23 user.setAge(20);

24//执行插入操作

25int retResult = sqlSession.insert(statement,user);

26//手动提交事务

27//sqlSession.commit();

28//使用SqlSession执行完SQL之后需要关闭SqlSession

29 sqlSession.close();

30 System.out.println(retResult);

31 }

32

33 @Test

34publicvoid testUpdate(){

35 SqlSession sqlSession = MyBatisUtil.getSqlSession(true);

36/\*\*

37 \* 映射sql的标识字符串，

38 \* me.gacl.mapping.userMapper是userMapper.xml文件中mapper标签的namespace属性的值，

39 \* updateUser是update标签的id属性值，通过update标签的id属性值就可以找到要执行的SQL

40\*/

41 String statement = "me.gacl.mapping.userMapper.updateUser";//映射sql的标识字符串

42 User user = new User();

43 user.setId(3);

44 user.setName("孤傲苍狼");

45 user.setAge(25);

46//执行修改操作

47int retResult = sqlSession.update(statement,user);

48//使用SqlSession执行完SQL之后需要关闭SqlSession

49 sqlSession.close();

50 System.out.println(retResult);

51 }

52

53 @Test

54publicvoid testDelete(){

55 SqlSession sqlSession = MyBatisUtil.getSqlSession(true);

56/\*\*

57 \* 映射sql的标识字符串，

58 \* me.gacl.mapping.userMapper是userMapper.xml文件中mapper标签的namespace属性的值，

59 \* deleteUser是delete标签的id属性值，通过delete标签的id属性值就可以找到要执行的SQL

60\*/

61 String statement = "me.gacl.mapping.userMapper.deleteUser";//映射sql的标识字符串

62//执行删除操作

63int retResult = sqlSession.delete(statement,5);

64//使用SqlSession执行完SQL之后需要关闭SqlSession

65 sqlSession.close();

66 System.out.println(retResult);

67 }

68

69 @Test

70publicvoid testGetAll(){

71 SqlSession sqlSession = MyBatisUtil.getSqlSession();

72/\*\*

73 \* 映射sql的标识字符串，

74 \* me.gacl.mapping.userMapper是userMapper.xml文件中mapper标签的namespace属性的值，

75 \* getAllUsers是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL

76\*/

77 String statement = "me.gacl.mapping.userMapper.getAllUsers";//映射sql的标识字符串

78//执行查询操作，将查询结果自动封装成List<User>返回

79 List<User> lstUsers = sqlSession.selectList(statement);

80//使用SqlSession执行完SQL之后需要关闭SqlSession

81 sqlSession.close();

82 System.out.println(lstUsers);

83 }

84 }

[复制代码](javascript:void(0);)

**二、使用MyBatis对表执行CRUD操作——基于注解的实现**

　　1、定义sql映射的接口

　　UserMapperI接口的代码如下：

[复制代码](javascript:void(0);)

1package me.gacl.mapping;

2

3import java.util.List;

4import me.gacl.domain.User;

5import org.apache.ibatis.annotations.Delete;

6import org.apache.ibatis.annotations.Insert;

7import org.apache.ibatis.annotations.Select;

8import org.apache.ibatis.annotations.Update;

9

10/\*\*

11 \* @author gacl

12 \* 定义sql映射的接口，使用注解指明方法要执行的SQL

13\*/

14publicinterface UserMapperI {

15

16//使用@Insert注解指明add方法要执行的SQL

17 @Insert("insert into users(name, age) values(#{name}, #{age})")

18publicint add(User user);

19

20//使用@Delete注解指明deleteById方法要执行的SQL

21 @Delete("delete from users where id=#{id}")

22publicint deleteById(int id);

23

24//使用@Update注解指明update方法要执行的SQL

25 @Update("update users set name=#{name},age=#{age} where id=#{id}")

26publicint update(User user);

27

28//使用@Select注解指明getById方法要执行的SQL

29 @Select("select \* from users where id=#{id}")

30public User getById(int id);

31

32//使用@Select注解指明getAll方法要执行的SQL

33 @Select("select \* from users")

34public List<User> getAll();

35 }

[复制代码](javascript:void(0);)

　　需要说明的是，我们不需要针对UserMapperI接口去编写具体的实现类代码，这个具体的实现类由MyBatis帮我们动态构建出来，我们只需要直接拿来使用即可。

　　2、在conf.xml文件中注册这个映射接口

[复制代码](javascript:void(0);)

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

2<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">

3<configuration>

4<environments default="development">

5<environment id="development">

6<transactionManager type="JDBC"/>

7<!-- 配置数据库连接信息 -->

8<dataSource type="POOLED">

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

10<property name="url" value="jdbc:mysql://localhost:3306/mybatis"/>

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

12<property name="password" value="XDP"/>

13</dataSource>

14</environment>

15</environments>

16

17<mappers>

18<!-- 注册userMapper.xml文件，

19 userMapper.xml位于me.gacl.mapping这个包下，所以resource写成me/gacl/mapping/userMapper.xml-->

20<mapper resource="me/gacl/mapping/userMapper.xml"/>

21<!-- 注册UserMapper映射接口-->

22<mapper class="me.gacl.mapping.UserMapperI"/>

23</mappers>

24

25</configuration>

[复制代码](javascript:void(0);)

　　单元测试类的代码如下：

[复制代码](javascript:void(0);)

1package me.gacl.test;

2

3import java.util.List;

4import me.gacl.domain.User;

5import me.gacl.mapping.UserMapperI;

6import me.gacl.util.MyBatisUtil;

7import org.apache.ibatis.session.SqlSession;

8import org.junit.Test;

9

10publicclass TestCRUDByAnnotationMapper {

11

12 @Test

13publicvoid testAdd(){

14 SqlSession sqlSession = MyBatisUtil.getSqlSession(true);

15//得到UserMapperI接口的实现类对象，UserMapperI接口的实现类对象由sqlSession.getMapper(UserMapperI.class)动态构建出来

16  UserMapperI mapper = sqlSession.getMapper(UserMapperI.class);

17 User user = new User();

18 user.setName("用户xdp");

19 user.setAge(20);

20int add = mapper.add(user);

21//使用SqlSession执行完SQL之后需要关闭SqlSession

22 sqlSession.close();

23 System.out.println(add);

24 }

25

26 @Test

27publicvoid testUpdate(){

28 SqlSession sqlSession = MyBatisUtil.getSqlSession(true);

29//得到UserMapperI接口的实现类对象，UserMapperI接口的实现类对象由sqlSession.getMapper(UserMapperI.class)动态构建出来

30  UserMapperI mapper = sqlSession.getMapper(UserMapperI.class);

31 User user = new User();

32 user.setId(3);

33 user.setName("孤傲苍狼\_xdp");

34 user.setAge(26);

35//执行修改操作

36int retResult = mapper.update(user);

37//使用SqlSession执行完SQL之后需要关闭SqlSession

38 sqlSession.close();

39 System.out.println(retResult);

40 }

41

42 @Test

43publicvoid testDelete(){

44 SqlSession sqlSession = MyBatisUtil.getSqlSession(true);

45//得到UserMapperI接口的实现类对象，UserMapperI接口的实现类对象由sqlSession.getMapper(UserMapperI.class)动态构建出来

46 UserMapperI mapper = sqlSession.getMapper(UserMapperI.class);

47//执行删除操作

48int retResult = mapper.deleteById(7);

49//使用SqlSession执行完SQL之后需要关闭SqlSession

50 sqlSession.close();

51 System.out.println(retResult);

52 }

53

54 @Test

55publicvoid testGetUser(){

56 SqlSession sqlSession = MyBatisUtil.getSqlSession();

57//得到UserMapperI接口的实现类对象，UserMapperI接口的实现类对象由sqlSession.getMapper(UserMapperI.class)动态构建出来

58 UserMapperI mapper = sqlSession.getMapper(UserMapperI.class);

59//执行查询操作，将查询结果自动封装成User返回

60 User user = mapper.getById(8);

61//使用SqlSession执行完SQL之后需要关闭SqlSession

62 sqlSession.close();

63 System.out.println(user);

64 }

65

66 @Test

67publicvoid testGetAll(){

68 SqlSession sqlSession = MyBatisUtil.getSqlSession();

69//得到UserMapperI接口的实现类对象，UserMapperI接口的实现类对象由sqlSession.getMapper(UserMapperI.class)动态构建出来

70 UserMapperI mapper = sqlSession.getMapper(UserMapperI.class);

71//执行查询操作，将查询结果自动封装成List<User>返回

72 List<User> lstUsers = mapper.getAll();

73//使用SqlSession执行完SQL之后需要关闭SqlSession

74 sqlSession.close();

75 System.out.println(lstUsers);

76 }

77 }

[复制代码](javascript:void(0);)

　　用到的MyBatisUtil工具类代码如下：

[复制代码](javascript:void(0);)

1package me.gacl.util;

2

3import java.io.InputStream;

4

5import org.apache.ibatis.session.SqlSession;

6import org.apache.ibatis.session.SqlSessionFactory;

7import org.apache.ibatis.session.SqlSessionFactoryBuilder;

8

9publicclass MyBatisUtil {

10

11/\*\*

12 \* 获取SqlSessionFactory

13 \* @return SqlSessionFactory

14\*/

15publicstatic SqlSessionFactory getSqlSessionFactory() {

16 String resource = "conf.xml";

17 InputStream is = MyBatisUtil.class.getClassLoader().getResourceAsStream(resource);

18 SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(is);

19return factory;

20 }

21

22/\*\*

23 \* 获取SqlSession

24 \* @return SqlSession

25\*/

26publicstatic SqlSession getSqlSession() {

27return getSqlSessionFactory().openSession();

28 }

29

30/\*\*

31 \* 获取SqlSession

32 \* @param isAutoCommit

33 \* true 表示创建的SqlSession对象在执行完SQL之后会自动提交事务

34 \* false 表示创建的SqlSession对象在执行完SQL之后不会自动提交事务，这时就需要我们手动调用sqlSession.commit()提交事务

35 \* @return SqlSession

36\*/

37publicstatic SqlSession getSqlSession(boolean isAutoCommit) {

38return getSqlSessionFactory().openSession(isAutoCommit);

39 }

40 }

[复制代码](javascript:void(0);)

　　 以上的相关代码是全部测试通过的，关于使用MyBatis对表执行CRUD操作的内容就这么多。

[**MyBatis学习总结(三)——优化MyBatis配置文件中的配置**](http://www.cnblogs.com/xdp-gacl/p/4264301.html)

**一、连接数据库的配置单独放在一个properties文件中**

　　之前，我们是直接将数据库的连接配置信息写在了MyBatis的conf.xml文件中，如下：

[复制代码](javascript:void(0);)

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

2<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">

3<configuration>

4<environments default="development">

5<environment id="development">

6<transactionManager type="JDBC"/>

7<!-- 配置数据库连接信息 -->

8<dataSource type="POOLED">

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

10<property name="url" value="jdbc:mysql://localhost:3306/mybatis"/>

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

12<property name="password" value="XDP"/>

13</dataSource>

14</environment>

15</environments>

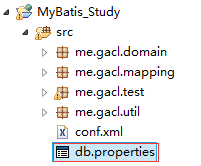
16

17</configuration>

[复制代码](javascript:void(0);)

　　其实我们完全可以将数据库的连接配置信息写在一个properties文件中，然后在conf.xml文件中引用properties文件，具体做法如下：

　　1、在src目录下新建一个db.properties文件，如下图所示：



　　在db.properties文件编写连接数据库需要使用到的数据库驱动，连接URL地址，用户名，密码，如下：

1 driver=com.mysql.jdbc.Driver

2 url=jdbc:mysql://localhost:3306/mybatis

3 name=root

4 password=XDP

　　2、在MyBatis的conf.xml文件中引用db.properties文件，如下：

[复制代码](javascript:void(0);)

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

2<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">

3<configuration>

4

5<!-- 引用db.properties配置文件 -->

6<properties resource="db.properties"/>

7<!--

8 development : 开发模式

9 work : 工作模式

10-->

11<environments default="development">

12<environment id="development">

13<transactionManager type="JDBC"/>

14<!-- 配置数据库连接信息 -->

15<dataSource type="POOLED">

16<!-- value属性值引用db.properties配置文件中配置的值 -->

17<property name="driver" value="${driver}"/>

18<property name="url" value="${url}"/>

19<property name="username" value="${name}"/>

20<property name="password" value="${password}"/>

21</dataSource>

22</environment>

23</environments>

24

25</configuration>

[复制代码](javascript:void(0);)

**二、为实体类定义别名，简化sql映射xml文件中的引用**

　　之前，我们在sql映射xml文件中的引用实体类时，需要写上实体类的全类名(包名+类名)，如下：

<!-- 创建用户(Create) -->

<insert id="addUser" parameterType="**me.gacl.domain.User**">

insert into users(name,age) values(#{name},#{age})

</insert>

　　parameterType="**me.gacl.domain.User**"这里写的实体类User的全类名**me.gacl.domain.User，**每次都写这么一长串内容挺麻烦的，而我们希望能够简写成下面的形式

<insert id="addUser2" parameterType="**\_User**">

insert into users(name,age) values(#{name},#{age})

</insert>

　　parameterType="**\_User**"这样写就简单多了，为了达到这种效果，我们需要在conf.xml文件中为实体类="**me.gacl.domain.User**"定义一个别名为"**\_User**"，具体做法如下：  
　　在conf.xml文件中<configuration></configuration>标签中添加如下配置：

<typeAliases>

<typeAlias type="me.gacl.domain.User" alias="\_User"/>

</typeAliases>

　　这样就可以为**me.gacl.domain.User**类定义了一个别名为**\_User，**以后**\_User**就代表了**me.gacl.domain.User**类，这样sql映射xml文件中的凡是需要引用**me.gacl.domain.User**类的地方都可以使用**\_User**来代替，这就达到了一个简化实体类引用的目的。

　　除了可以使用<typeAlias type="me.gacl.domain.User" alias="\_User"/>这种方式单独为某一个实体类设置别名之外，我们还可以使用如下的方式批量为某个包下的所有实体类设置别名，如下：

[复制代码](javascript:void(0);)

<!-- 配置实体类的别名，配置实体类别名的目的是为了在引用实体类时可以使用实体类的别名来代替实体类，达到简写的目的 -->

<typeAliases>

<!-- 为实体类me.gacl.domain.User配置一个别名\_User -->

<!-- <typeAlias type="me.gacl.domain.User" alias="\_User"/> -->

<!-- 为me.gacl.domain包下的所有实体类配置别名，MyBatis默认的设置别名的方式就是去除类所在的包后的简单的类名

比如me.gacl.domain.User这个实体类的别名就会被设置成User

-->

<package name="me.gacl.domain"/>

</typeAliases>

[复制代码](javascript:void(0);)

　　<package name="me.gacl.domain"/>就表示为这个包下面的所有实体类设置别名。MyBatis默认的设置别名的方式就是去除类所在的包后的简单的类名，比如me.gacl.domain.User这个实体类的别名就会被设置成User。

**只为成功找方法，不为失败找借口！**

[**MyBatis学习总结(四)——解决字段名与实体类属性名不相同的冲突**](http://www.cnblogs.com/xdp-gacl/p/4264425.html)

　　在平时的开发中，我们表中的字段名和表对应实体类的属性名称不一定都是完全相同的，下面来演示一下这种情况下的如何解决字段名与实体类属性名不相同的冲突。

**一、准备演示需要使用的表和数据**

[复制代码](javascript:void(0);)

CREATETABLE orders(

order\_id INTPRIMARYKEY AUTO\_INCREMENT,

order\_no VARCHAR(**20**),

order\_price FLOAT

);

INSERTINTO orders(order\_no, order\_price) VALUES('aaaa', **23**);

INSERTINTO orders(order\_no, order\_price) VALUES('bbbb', **33**);

INSERTINTO orders(order\_no, order\_price) VALUES('cccc', **22**);

[复制代码](javascript:void(0);)

**二、定义实体类**

[复制代码](javascript:void(0);)

1package me.gacl.domain;

2

3/\*\*

4 \* @author gacl

5 \* 定义orders表对应的实体类

6\*/

7publicclass Order {

8/\*\*

9 \*

10 CREATE TABLE orders(

11 order\_id INT PRIMARY KEY AUTO\_INCREMENT,

12 order\_no VARCHAR(20),

13 order\_price FLOAT

14 );

15\*/

16

17//Order实体类中属性名和orders表中的字段名是不一样的

18privateint id; //id===>order\_id

19private String orderNo; //orderNo===>order\_no

20privatefloat price; //price===>order\_price

21

22publicint getId() {

23return id;

24 }

25

26publicvoid setId(int id) {

27this.id = id;

28 }

29

30public String getOrderNo() {

31return orderNo;

32 }

33

34publicvoid setOrderNo(String orderNo) {

35this.orderNo = orderNo;

36 }

37

38publicfloat getPrice() {

39return price;

40 }

41

42publicvoid setPrice(float price) {

43this.price = price;

44 }

45

46 @Override

47public String toString() {

48return "Order [id=" + id + ", orderNo=" + orderNo + ", price=" + price+ "]";

49 }

50 }

[复制代码](javascript:void(0);)

**三、编写测试代码**

**3.1、编写SQL的xml映射文件**

　　1、创建一个orderMapper.xml文件，orderMapper.xml的内容如下：

[复制代码](javascript:void(0);)

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

2<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

3<!-- 为这个mapper指定一个唯一的namespace，namespace的值习惯上设置成包名+sql映射文件名，这样就能够保证namespace的值是唯一的

4例如namespace="me.gacl.mapping.orderMapper"就是me.gacl.mapping(包名)+orderMapper(orderMapper.xml文件去除后缀)

5-->

6<mapper namespace="me.gacl.mapping.orderMapper">

7

8<!--

9 根据id查询得到一个order对象，使用这个查询是查询不到我们想要的结果的，

10 这主要是因为实体类的属性名和数据库的字段名对应不上的原因，因此无法查询出对应的记录

11-->

12<select id="getOrderById" parameterType="int"

13 resultType="me.gacl.domain.Order">

14 select \* from orders where order\_id=#{id}

15</select>

16

17<!--

18 根据id查询得到一个order对象，使用这个查询是可以正常查询到我们想要的结果的，

19 这是因为我们将查询的字段名都起一个和实体类属性名相同的别名，这样实体类的属性名和查询结果中的字段名就可以一一对应上

20-->

21<select id="selectOrder" parameterType="int"

22 resultType="me.gacl.domain.Order">

23 select order\_id id, order\_no orderNo,order\_price price from orders where order\_id=#{id}

24</select>

25

26<!--

27 根据id查询得到一个order对象，使用这个查询是可以正常查询到我们想要的结果的，

28 这是因为我们通过<resultMap>映射实体类属性名和表的字段名一一对应关系 -->

29<select id="selectOrderResultMap" parameterType="int" resultMap="orderResultMap">

30 select \* from orders where order\_id=#{id}

31</select>

32<!--通过<resultMap>映射实体类属性名和表的字段名对应关系 -->

33<resultMap type="me.gacl.domain.Order" id="orderResultMap">

34<!-- 用id属性来映射主键字段 -->

35<id property="id" column="order\_id"/>

36<!-- 用result属性来映射非主键字段 -->

37<result property="orderNo" column="order\_no"/>

38<result property="price" column="order\_price"/>

39</resultMap>

40

41</mapper>

[复制代码](javascript:void(0);)

　　2、在conf.xml文件中注册orderMapper.xml映射文件

<mappers>

<!-- 注册orderMapper.xml文件，

orderMapper.xml位于me.gacl.mapping这个包下，所以resource写成me/gacl/mapping/orderMapper.xml-->

<mapper resource="me/gacl/mapping/orderMapper.xml"/>

</mappers>

**3.2、编写单元测试代码**

[复制代码](javascript:void(0);)

1package me.gacl.test;

2

3import me.gacl.domain.Order;

4import me.gacl.util.MyBatisUtil;

5import org.apache.ibatis.session.SqlSession;

6import org.junit.Test;

7

8publicclass Test2 {

9

10 @Test

11publicvoid testGetOrderById(){

12 SqlSession sqlSession = MyBatisUtil.getSqlSession();

13/\*\*

14 \* 映射sql的标识字符串，

15 \* me.gacl.mapping.orderMapper是orderMapper.xml文件中mapper标签的namespace属性的值，

16 \* getOrderById是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL

17\*/

18 String statement = "me.gacl.mapping.orderMapper.getOrderById";//映射sql的标识字符串

19//执行查询操作，将查询结果自动封装成Order对象返回

20 Order order = sqlSession.selectOne(statement,1);//查询orders表中id为1的记录

21//使用SqlSession执行完SQL之后需要关闭SqlSession

22 sqlSession.close();

23 System.out.println(order);//打印结果：null，也就是没有查询出相应的记录

24 }

25

26 @Test

27publicvoid testGetOrderById2(){

28 SqlSession sqlSession = MyBatisUtil.getSqlSession();

29/\*\*

30 \* 映射sql的标识字符串，

31 \* me.gacl.mapping.orderMapper是orderMapper.xml文件中mapper标签的namespace属性的值，

32 \* selectOrder是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL

33\*/

34 String statement = "me.gacl.mapping.orderMapper.selectOrder";//映射sql的标识字符串

35//执行查询操作，将查询结果自动封装成Order对象返回

36 Order order = sqlSession.selectOne(statement,1);//查询orders表中id为1的记录

37//使用SqlSession执行完SQL之后需要关闭SqlSession

38 sqlSession.close();

39 System.out.println(order);//打印结果：Order [id=1, orderNo=aaaa, price=23.0]

40 }

41

42 @Test

43publicvoid testGetOrderById3(){

44 SqlSession sqlSession = MyBatisUtil.getSqlSession();

45/\*\*

46 \* 映射sql的标识字符串，

47 \* me.gacl.mapping.orderMapper是orderMapper.xml文件中mapper标签的namespace属性的值，

48 \* selectOrderResultMap是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL

49\*/

50 String statement = "me.gacl.mapping.orderMapper.selectOrderResultMap";//映射sql的标识字符串

51//执行查询操作，将查询结果自动封装成Order对象返回

52 Order order = sqlSession.selectOne(statement,1);//查询orders表中id为1的记录

53//使用SqlSession执行完SQL之后需要关闭SqlSession

54 sqlSession.close();

55 System.out.println(order);//打印结果：Order [id=1, orderNo=aaaa, price=23.0]

56 }

57 }

[复制代码](javascript:void(0);)

　　执行单元测试的结果：

　　1、testGetOrderById方法执行查询后返回一个null。

　　2、testGetOrderById2方法和testGetOrderById3方法执行查询后可以正常得到想要的结果。

**四、总结**

　　上面的测试代码演示当实体类中的属性名和表中的字段名不一致时，使用MyBatis进行查询操作时无法查询出相应的结果的问题以及针对问题采用的两种办法：

**解决办法一: 通过在查询的sql语句中定义字段名的别名，让字段名的别名和实体类的属性名一致，这样就可以表的字段名和实体类的属性名一一对应上了，这种方式是通过在sql语句中定义别名来解决字段名和属性名的映射关系的。**

**解决办法二: 通过<resultMap>来映射字段名和实体类属性名的一一对应关系。这种方式是使用MyBatis提供的解决方式来解决字段名和属性名的映射关系的。**

[MyBatis学习总结(五)——实现关联表查询](http://www.cnblogs.com/xdp-gacl/p/4264440.html)

## 一、一对一关联

### 1.1、提出需求

　　根据班级id查询班级信息(带老师的信息)

### 1.2、创建表和数据

　　创建一张教师表和班级表，这里我们假设一个老师只负责教一个班，那么老师和班级之间的关系就是一种一对一的关系。

[复制代码](javascript:void(0);)

1CREATETABLE teacher(

2 t\_id INTPRIMARYKEY AUTO\_INCREMENT,

3 t\_name VARCHAR(**20**)

4 );

5CREATETABLE class(

6 c\_id INTPRIMARYKEY AUTO\_INCREMENT,

7 c\_name VARCHAR(**20**),

8 teacher\_id INT

9 );

10ALTERTABLE class ADDCONSTRAINT fk\_teacher\_id FOREIGNKEY (teacher\_id) REFERENCES teacher(t\_id);

11

12INSERTINTO teacher(t\_name) VALUES('teacher1');

13INSERTINTO teacher(t\_name) VALUES('teacher2');

14

15INSERTINTO class(c\_name, teacher\_id) VALUES('class\_a', **1**);

16INSERTINTO class(c\_name, teacher\_id) VALUES('class\_b', **2**);

[复制代码](javascript:void(0);)

　　表之间的关系如下：



### 1.3、定义实体类

　　1、Teacher类，Teacher类是teacher表对应的实体类。

[复制代码](javascript:void(0);)

1package me.gacl.domain;

2

3/\*\*

4 \* @author gacl

5 \* 定义teacher表对应的实体类

6\*/

7publicclass Teacher {

8

9//定义实体类的属性，与teacher表中的字段对应

10privateint id; //id===>t\_id

11private String name; //name===>t\_name

12

13publicint getId() {

14return id;

15 }

16

17publicvoid setId(int id) {

18this.id = id;

19 }

20

21public String getName() {

22return name;

23 }

24

25publicvoid setName(String name) {

26this.name = name;

27 }

28

29 @Override

30public String toString() {

31return "Teacher [id=" + id + ", name=" + name + "]";

32 }

33 }

[复制代码](javascript:void(0);)

　　2、Classes类，Classes类是class表对应的实体类

[复制代码](javascript:void(0);)

1package me.gacl.domain;

2

3/\*\*

4 \* @author gacl

5 \* 定义class表对应的实体类

6\*/

7publicclass Classes {

8

9//定义实体类的属性，与class表中的字段对应

10privateint id; //id===>c\_id

11private String name; //name===>c\_name

12

13/\*\*

14 \* class表中有一个teacher\_id字段，所以在Classes类中定义一个teacher属性，

15 \* 用于维护teacher和class之间的一对一关系，通过这个teacher属性就可以知道这个班级是由哪个老师负责的

16\*/

17private Teacher teacher;

18

19publicint getId() {

20return id;

21 }

22

23publicvoid setId(int id) {

24this.id = id;

25 }

26

27public String getName() {

28return name;

29 }

30

31publicvoid setName(String name) {

32this.name = name;

33 }

34

35public Teacher getTeacher() {

36return teacher;

37 }

38

39publicvoid setTeacher(Teacher teacher) {

40this.teacher = teacher;

41 }

42

43 @Override

44public String toString() {

45return "Classes [id=" + id + ", name=" + name + ", teacher=" + teacher+ "]";

46 }

47 }

[复制代码](javascript:void(0);)

### 1.4、定义sql映射文件classMapper.xml

[复制代码](javascript:void(0);)

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

2<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

3<!-- 为这个mapper指定一个唯一的namespace，namespace的值习惯上设置成包名+sql映射文件名，这样就能够保证namespace的值是唯一的

4例如namespace="me.gacl.mapping.classMapper"就是me.gacl.mapping(包名)+classMapper(classMapper.xml文件去除后缀)

5-->

6<mapper namespace="me.gacl.mapping.classMapper">

7

8<!--

9 根据班级id查询班级信息(带老师的信息)

10 ##1. 联表查询

11 SELECT \* FROM class c,teacher t WHERE c.teacher\_id=t.t\_id AND c.c\_id=1;

12

13 ##2. 执行两次查询

14 SELECT \* FROM class WHERE c\_id=1; //teacher\_id=1

15 SELECT \* FROM teacher WHERE t\_id=1;//使用上面得到的teacher\_id

16-->

17

18<!--

19 方式一：嵌套结果：使用嵌套结果映射来处理重复的联合结果的子集

20 封装联表查询的数据(去除重复的数据)

21 select \* from class c, teacher t where c.teacher\_id=t.t\_id and c.c\_id=1

22-->

23<select id="getClass" parameterType="int" resultMap="ClassResultMap">

24 select \* from class c, teacher t where c.teacher\_id=t.t\_id and c.c\_id=#{id}

25</select>

26<!-- 使用resultMap映射实体类和字段之间的一一对应关系 -->

27<resultMap type="me.gacl.domain.Classes" id="ClassResultMap">

28<id property="id" column="c\_id"/>

29<result property="name" column="c\_name"/>

30<association property="teacher" javaType="me.gacl.domain.Teacher">

31<id property="id" column="t\_id"/>

32<result property="name" column="t\_name"/>

33</association>

34</resultMap>

35

36<!--

37 方式二：嵌套查询：通过执行另外一个SQL映射语句来返回预期的复杂类型

38 SELECT \* FROM class WHERE c\_id=1;

39 SELECT \* FROM teacher WHERE t\_id=1 //1 是上一个查询得到的teacher\_id的值

40-->

41<select id="getClass2" parameterType="int" resultMap="ClassResultMap2">

42 select \* from class where c\_id=#{id}

43</select>

44<!-- 使用resultMap映射实体类和字段之间的一一对应关系 -->

45<resultMap type="me.gacl.domain.Classes" id="ClassResultMap2">

46<id property="id" column="c\_id"/>

47<result property="name" column="c\_name"/>

48<association property="teacher" column="teacher\_id" select="getTeacher"/>

49</resultMap>

50

51<select id="getTeacher" parameterType="int" resultType="me.gacl.domain.Teacher">

52 SELECT t\_id id, t\_name name FROM teacher WHERE t\_id=#{id}

53</select>

54

55</mapper>

[复制代码](javascript:void(0);)

　　在conf.xml文件中注册classMapper.xml

<mappers>

<!-- 注册classMapper.xml文件，

classMapper.xml位于me.gacl.mapping这个包下，所以resource写成me/gacl/mapping/classMapper.xml-->

<mapper resource="me/gacl/mapping/classMapper.xml"/>

</mappers>

### 1.5、编写单元测试代码

[复制代码](javascript:void(0);)

1package me.gacl.test;

2

3import me.gacl.domain.Classes;

4import me.gacl.util.MyBatisUtil;

5import org.apache.ibatis.session.SqlSession;

6import org.junit.Test;

7

8publicclass Test3 {

9

10 @Test

11publicvoid testGetClass(){

12 SqlSession sqlSession = MyBatisUtil.getSqlSession();

13/\*\*

14 \* 映射sql的标识字符串，

15 \* me.gacl.mapping.classMapper是classMapper.xml文件中mapper标签的namespace属性的值，

16 \* getClass是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL

17\*/

18 String statement = "me.gacl.mapping.classMapper.getClass";//映射sql的标识字符串

19//执行查询操作，将查询结果自动封装成Classes对象返回

20 Classes clazz = sqlSession.selectOne(statement,1);//查询class表中id为1的记录

21//使用SqlSession执行完SQL之后需要关闭SqlSession

22 sqlSession.close();

23 System.out.println(clazz);//打印结果：Classes [id=1, name=class\_a, teacher=Teacher [id=1, name=teacher1]]

24 }

25

26 @Test

27publicvoid testGetClass2(){

28 SqlSession sqlSession = MyBatisUtil.getSqlSession();

29/\*\*

30 \* 映射sql的标识字符串，

31 \* me.gacl.mapping.classMapper是classMapper.xml文件中mapper标签的namespace属性的值，

32 \* getClass2是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL

33\*/

34 String statement = "me.gacl.mapping.classMapper.getClass2";//映射sql的标识字符串

35//执行查询操作，将查询结果自动封装成Classes对象返回

36 Classes clazz = sqlSession.selectOne(statement,1);//查询class表中id为1的记录

37//使用SqlSession执行完SQL之后需要关闭SqlSession

38 sqlSession.close();

39 System.out.println(clazz);//打印结果：Classes [id=1, name=class\_a, teacher=Teacher [id=1, name=teacher1]]

40 }

41 }

[复制代码](javascript:void(0);)

### 1.6、MyBatis一对一关联查询总结

　　MyBatis中使用association标签来解决一对一的关联查询，association标签可用的属性如下：

* property:对象属性的名称
* javaType:对象属性的类型
* column:所对应的外键字段名称
* select:使用另一个查询封装的结果

## 二、一对多关联

### 2.1、提出需求

　　根据classId查询对应的班级信息,包括学生,老师

### 2.2、创建表和数据

　　在上面的一对一关联查询演示中，我们已经创建了班级表和教师表，因此这里再创建一张学生表

[复制代码](javascript:void(0);)

CREATETABLE student(

s\_id INTPRIMARYKEY AUTO\_INCREMENT,

s\_name VARCHAR(**20**),

class\_id INT

);

INSERTINTO student(s\_name, class\_id) VALUES('student\_A', **1**);

INSERTINTO student(s\_name, class\_id) VALUES('student\_B', **1**);

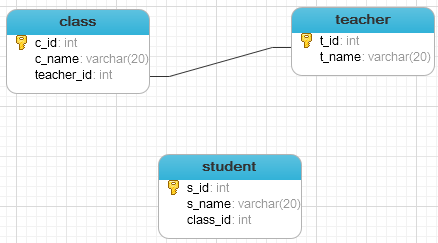
INSERTINTO student(s\_name, class\_id) VALUES('student\_C', **1**);

INSERTINTO student(s\_name, class\_id) VALUES('student\_D', **2**);

INSERTINTO student(s\_name, class\_id) VALUES('student\_E', **2**);

INSERTINTO student(s\_name, class\_id) VALUES('student\_F', **2**);

[复制代码](javascript:void(0);)



### 2.3、定义实体类

　　1、Student类

[复制代码](javascript:void(0);)

1package me.gacl.domain;

2

3/\*\*

4 \* @author gacl

5 \* 定义student表所对应的实体类

6\*/

7publicclass Student {

8

9//定义属性，和student表中的字段对应

10privateint id; //id===>s\_id

11private String name; //name===>s\_name

12

13publicint getId() {

14return id;

15 }

16

17publicvoid setId(int id) {

18this.id = id;

19 }

20

21public String getName() {

22return name;

23 }

24

25publicvoid setName(String name) {

26this.name = name;

27 }

28

29 @Override

30public String toString() {

31return "Student [id=" + id + ", name=" + name + "]";

32 }

33 }

[复制代码](javascript:void(0);)

　　2、修改Classes类，添加一个List<Student> students属性，使用一个List<Student>集合属性表示班级拥有的学生，如下：

[复制代码](javascript:void(0);)

1package me.gacl.domain;

2

3import java.util.List;

4

5/\*\*

6 \* @author gacl

7 \* 定义class表对应的实体类

8\*/

9publicclass Classes {

10

11//定义实体类的属性，与class表中的字段对应

12privateint id; //id===>c\_id

13private String name; //name===>c\_name

14

15/\*\*

16 \* class表中有一个teacher\_id字段，所以在Classes类中定义一个teacher属性，

17 \* 用于维护teacher和class之间的一对一关系，通过这个teacher属性就可以知道这个班级是由哪个老师负责的

18\*/

19private Teacher teacher;

20//使用一个List<Student>集合属性表示班级拥有的学生

21private List<Student> students;

22

23publicint getId() {

24return id;

25 }

26

27publicvoid setId(int id) {

28this.id = id;

29 }

30

31public String getName() {

32return name;

33 }

34

35publicvoid setName(String name) {

36this.name = name;

37 }

38

39public Teacher getTeacher() {

40return teacher;

41 }

42

43publicvoid setTeacher(Teacher teacher) {

44this.teacher = teacher;

45 }

46

47public List<Student> getStudents() {

48return students;

49 }

50

51publicvoid setStudents(List<Student> students) {

52this.students = students;

53 }

54

55 @Override

56public String toString() {

57return "Classes [id=" + id + ", name=" + name + ", teacher=" + teacher

58 + ", students=" + students + "]";

59 }

60 }

[复制代码](javascript:void(0);)

### 2.4、修改sql映射文件classMapper.xml

　　添加如下的SQL映射信息

[复制代码](javascript:void(0);)

1<!--

2 根据classId查询对应的班级信息,包括学生,老师

3-->

4<!--

5 方式一: 嵌套结果: 使用嵌套结果映射来处理重复的联合结果的子集

6 SELECT \* FROM class c, teacher t,student s WHERE c.teacher\_id=t.t\_id AND c.C\_id=s.class\_id AND c.c\_id=1

7-->

8<select id="getClass3" parameterType="int" resultMap="ClassResultMap3">

9 select \* from class c, teacher t,student s where c.teacher\_id=t.t\_id and c.C\_id=s.class\_id and c.c\_id=#{id}

10</select>

11<resultMap type="me.gacl.domain.Classes" id="ClassResultMap3">

12<id property="id" column="c\_id"/>

13<result property="name" column="c\_name"/>

14<association property="teacher" column="teacher\_id" javaType="me.gacl.domain.Teacher">

15<id property="id" column="t\_id"/>

16<result property="name" column="t\_name"/>

17</association>

18<!-- ofType指定students集合中的对象类型 -->

19<collection property="students" ofType="me.gacl.domain.Student">

20<id property="id" column="s\_id"/>

21<result property="name" column="s\_name"/>

22</collection>

23</resultMap>

24

25<!--

26 方式二：嵌套查询：通过执行另外一个SQL映射语句来返回预期的复杂类型

27 SELECT \* FROM class WHERE c\_id=1;

28 SELECT \* FROM teacher WHERE t\_id=1 //1 是上一个查询得到的teacher\_id的值

29 SELECT \* FROM student WHERE class\_id=1 //1是第一个查询得到的c\_id字段的值

30-->

31<select id="getClass4" parameterType="int" resultMap="ClassResultMap4">

32 select \* from class where c\_id=#{id}

33</select>

34<resultMap type="me.gacl.domain.Classes" id="ClassResultMap4">

35<id property="id" column="c\_id"/>

36<result property="name" column="c\_name"/>

37<association property="teacher" column="teacher\_id" javaType="me.gacl.domain.Teacher" select="getTeacher2"></association>

38<collection property="students" ofType="me.gacl.domain.Student" column="c\_id" select="getStudent"></collection>

39</resultMap>

40

41<select id="getTeacher2" parameterType="int" resultType="me.gacl.domain.Teacher">

42 SELECT t\_id id, t\_name name FROM teacher WHERE t\_id=#{id}

43</select>

44

45<select id="getStudent" parameterType="int" resultType="me.gacl.domain.Student">

46 SELECT s\_id id, s\_name name FROM student WHERE class\_id=#{id}

47</select>

[复制代码](javascript:void(0);)

### 2.5、编写单元测试代码

[复制代码](javascript:void(0);)

1package me.gacl.test;

2

3import me.gacl.domain.Classes;

4import me.gacl.util.MyBatisUtil;

5import org.apache.ibatis.session.SqlSession;

6import org.junit.Test;

7

8publicclass Test4 {

9

10 @Test

11publicvoid testGetClass3(){

12 SqlSession sqlSession = MyBatisUtil.getSqlSession();

13/\*\*

14 \* 映射sql的标识字符串，

15 \* me.gacl.mapping.classMapper是classMapper.xml文件中mapper标签的namespace属性的值，

16 \* getClass3是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL

17\*/

18 String statement = "me.gacl.mapping.classMapper.getClass3";//映射sql的标识字符串

19//执行查询操作，将查询结果自动封装成Classes对象返回

20 Classes clazz = sqlSession.selectOne(statement,1);//查询class表中id为1的记录

21//使用SqlSession执行完SQL之后需要关闭SqlSession

22 sqlSession.close();

23//打印结果：Classes [id=1, name=class\_a, teacher=Teacher [id=1, name=teacher1], students=[Student [id=1, name=student\_A], Student [id=2, name=student\_B], Student [id=3, name=student\_C]]]

24 System.out.println(clazz);

25 }

26

27 @Test

28publicvoid testGetClass4(){

29 SqlSession sqlSession = MyBatisUtil.getSqlSession();

30/\*\*

31 \* 映射sql的标识字符串，

32 \* me.gacl.mapping.classMapper是classMapper.xml文件中mapper标签的namespace属性的值，

33 \* getClass4是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL

34\*/

35 String statement = "me.gacl.mapping.classMapper.getClass4";//映射sql的标识字符串

36//执行查询操作，将查询结果自动封装成Classes对象返回

37 Classes clazz = sqlSession.selectOne(statement,1);//查询class表中id为1的记录

38//使用SqlSession执行完SQL之后需要关闭SqlSession

39 sqlSession.close();

40//打印结果：Classes [id=1, name=class\_a, teacher=Teacher [id=1, name=teacher1], students=[Student [id=1, name=student\_A], Student [id=2, name=student\_B], Student [id=3, name=student\_C]]]

41 System.out.println(clazz);

42 }

43 }

[复制代码](javascript:void(0);)

### 2.6、MyBatis一对多关联查询总结

　　MyBatis中使用collection标签来解决一对多的关联查询，ofType属性指定集合中元素的对象类型。

[**MyBatis学习总结(六)——调用存储过程**](http://www.cnblogs.com/xdp-gacl/p/4270352.html)

**一、提出需求**

查询得到男性或女性的数量, 如果传入的是0就女性否则是男性

**二、准备数据库表和存储过程**

[复制代码](javascript:void(0);)

1createtable p\_user(

2 id int primary key auto\_increment,

3 name varchar(**10**),

4 sex char(**2**)

5 );

6

7insertinto p\_user(name,sex) values('A',"男");

8insertinto p\_user(name,sex) values('B',"女");

9insertinto p\_user(name,sex) values('C',"男");

10

11-- 创建存储过程(查询得到男性或女性的数量, 如果传入的是0就女性否则是男性)

12 DELIMITER $

13CREATE PROCEDURE mybatis.ges\_user\_count(IN sex\_id INT, OUT user\_count INT)

14BEGIN

15IF sex\_id=**0**THEN

16SELECT COUNT(\*) FROM mybatis.p\_user WHERE p\_user.sex='女'INTO user\_count;

17ELSE

18SELECT COUNT(\*) FROM mybatis.p\_user WHERE p\_user.sex='男'INTO user\_count;

19ENDIF;

20END

21 $

22

23-- 调用存储过程

24 DELIMITER ;

25SET@user\_count=**0**;

26 CALL mybatis.ges\_user\_count(**1**, @user\_count);

27SELECT@user\_count;

[复制代码](javascript:void(0);)

**三、编辑userMapper.xml**

　　编辑userMapper.xml文件，添加如下的配置项

[复制代码](javascript:void(0);)

1<!--

2 查询得到男性或女性的数量, 如果传入的是0就女性否则是男性

3-->

4<select id="getUserCount" parameterMap="getUserCountMap" statementType="CALLABLE">

5 CALL mybatis.ges\_user\_count(?,?)

6</select>

7

8<!--

9 parameterMap.put("sexid", 0);

10 parameterMap.put("usercount", -1);

11-->

12<parameterMap type="java.util.Map" id="getUserCountMap">

13<parameter property="sexid" mode="IN" jdbcType="INTEGER"/>

14<parameter property="usercount" mode="OUT" jdbcType="INTEGER"/>

15</parameterMap>

[复制代码](javascript:void(0);)

**四、编写单元测试代码**

[复制代码](javascript:void(0);)

1package me.gacl.test;

2

3import java.util.HashMap;

4import java.util.List;

5import java.util.Map;

6

7import me.gacl.custom.model.ConditionUser;

8import me.gacl.domain.User;

9import me.gacl.util.MyBatisUtil;

10import org.apache.ibatis.session.SqlSession;

11import org.junit.Test;

12

13/\*\*

14 \* @author gacl

15 \* 测试调用存储过程

16\*/

17publicclass Test6 {

18

19 @Test

20publicvoid testGetUserCount(){

21 SqlSession sqlSession = MyBatisUtil.getSqlSession();

22/\*\*

23 \* 映射sql的标识字符串，

24 \* me.gacl.mapping.userMapper是userMapper.xml文件中mapper标签的namespace属性的值，

25 \* getUserCount是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL

26\*/

27 String statement = "me.gacl.mapping.userMapper.getUserCount";//映射sql的标识字符串

28 Map<String, Integer> parameterMap = new HashMap<String, Integer>();

29 parameterMap.put("sexid", 1);

30 parameterMap.put("usercount", -1);

31 sqlSession.selectOne(statement, parameterMap);

32 Integer result = parameterMap.get("usercount");

33 System.out.println(result);

34 sqlSession.close();

35 }

36 }

[复制代码](javascript:void(0);)

**只为成功找方法，不为失败找借口！**

## [MyBatis学习总结(七)——Mybatis缓存](http://www.cnblogs.com/xdp-gacl/p/4270403.html)

## 一、MyBatis缓存介绍

　　正如大多数持久层框架一样，MyBatis 同样提供了**一级缓存**和**二级缓存**的支持

1. **一级缓存**: 基于PerpetualCache 的 HashMap本地缓存，其**存储作用域为** **Session**，当 **Session flush 或 close** 之后，该**Session中的所有 Cache 就将清空**。

　　2. **二级缓存**与一级缓存其机制相同，默认也是采用 PerpetualCache，HashMap存储，不同在于其**存储作用域为 Mapper(Namespace)**，并且**可自定义存储源**，如 Ehcache。

　　3. 对于缓存数据更新机制，当某一个作用域(一级缓存Session/二级缓存Namespaces)的进行了 C/U/D 操作后，默认该作用域下所有 select 中的缓存将被clear。

### 1.1、Mybatis一级缓存测试

[复制代码](javascript:void(0);)

1package me.gacl.test;

2

3import me.gacl.domain.User;

4import me.gacl.util.MyBatisUtil;

5import org.apache.ibatis.session.SqlSession;

6import org.junit.Test;

7

8/\*\*

9 \* @author gacl

10 \* 测试一级缓存

11\*/

12publicclass TestOneLevelCache {

13

14/\*

15 \* 一级缓存: 也就Session级的缓存(默认开启)

16\*/

17 @Test

18publicvoid testCache1() {

19 SqlSession session = MyBatisUtil.getSqlSession();

20 String statement = "me.gacl.mapping.userMapper.getUser";

21 User user = session.selectOne(statement, 1);

22 System.out.println(user);

23

24/\*

25 \* 一级缓存默认就会被使用

26\*/

27 user = session.selectOne(statement, 1);

28 System.out.println(user);

29 session.close();

30/\*

31 1. 必须是同一个Session,如果session对象已经close()过了就不可能用了

32\*/

33 session = MyBatisUtil.getSqlSession();

34 user = session.selectOne(statement, 1);

35 System.out.println(user);

36

37/\*

38 2. 查询条件是一样的

39\*/

40 user = session.selectOne(statement, 2);

41 System.out.println(user);

42

43/\*

44 3. 没有执行过session.clearCache()清理缓存

45\*/

46//session.clearCache();

47 user = session.selectOne(statement, 2);

48 System.out.println(user);

49

50/\*

51 4. 没有执行过增删改的操作(这些操作都会清理缓存)

52\*/

53 session.update("me.gacl.mapping.userMapper.updateUser",

54new User(2, "user", 23));

55 user = session.selectOne(statement, 2);

56 System.out.println(user);

57

58 }

59 }

[复制代码](javascript:void(0);)

### 1.2、Mybatis二级缓存测试

　　1、开启二级缓存，在userMapper.xml文件中添加如下配置

<mapper namespace="me.gacl.mapping.userMapper">

<!-- 开启二级缓存 -->

<cache/>

　　2、测试二级缓存

[复制代码](javascript:void(0);)

1package me.gacl.test;

2

3import me.gacl.domain.User;

4import me.gacl.util.MyBatisUtil;

5import org.apache.ibatis.session.SqlSession;

6import org.apache.ibatis.session.SqlSessionFactory;

7import org.junit.Test;

8

9/\*\*

10 \* @author gacl

11 \* 测试二级缓存

12\*/

13publicclass TestTwoLevelCache {

14

15/\*

16 \* 测试二级缓存

17 \* 使用两个不同的SqlSession对象去执行相同查询条件的查询，第二次查询时不会再发送SQL语句，而是直接从缓存中取出数据

18\*/

19 @Test

20publicvoid testCache2() {

21 String statement = "me.gacl.mapping.userMapper.getUser";

22 SqlSessionFactory factory = MyBatisUtil.getSqlSessionFactory();

23//开启两个不同的SqlSession

24 SqlSession session1 = factory.openSession();

25 SqlSession session2 = factory.openSession();

26//使用二级缓存时，User类必须实现一个Serializable接口===> User implements Serializable

27 User user = session1.selectOne(statement, 1);

28 session1.commit();//不懂为啥，这个地方一定要提交事务之后二级缓存才会起作用

29 System.out.println("user="+user);

30

31//由于使用的是两个不同的SqlSession对象，所以即使查询条件相同，一级缓存也不会开启使用

32 user = session2.selectOne(statement, 1);

33//session2.commit();

34 System.out.println("user2="+user);

35 }

36 }

[复制代码](javascript:void(0);)

### 1.3、二级缓存补充说明

　　1. 映射语句文件中的所有select语句将会被缓存。

　　2. 映射语句文件中的所有insert，update和delete语句会刷新缓存。

　　3. 缓存会使用Least Recently Used（LRU，最近最少使用的）算法来收回。

　　4. 缓存会根据指定的时间间隔来刷新。

　　5. 缓存会存储1024个对象

cache标签常用属性：

<cache

eviction="FIFO"<!--回收策略为先进先出-->

flushInterval="60000" <!--自动刷新时间60s-->

size="512" <!--最多缓存512个引用对象-->

readOnly="true"/><!--只读-->

**只为成功找方法，不为失败找借口！**

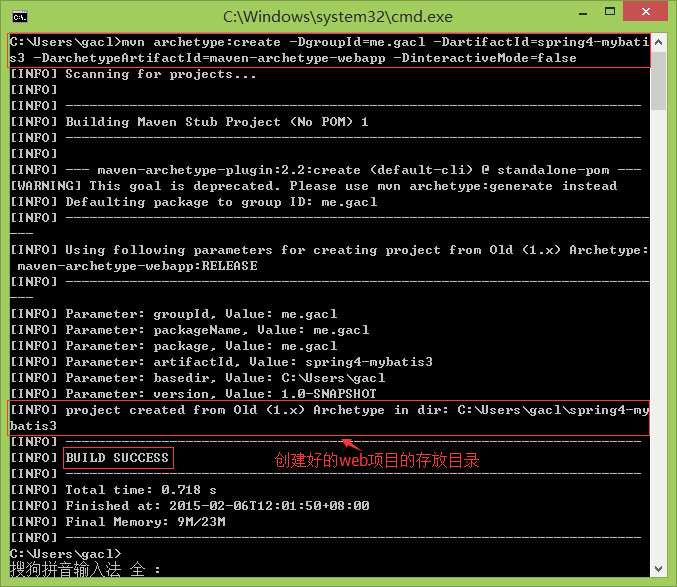
[**MyBatis学习总结(八)——Mybatis3.x与Spring4.x整合**](http://www.cnblogs.com/xdp-gacl/p/4271627.html)

**一、搭建开发环境**

**1.1、使用Maven创建Web项目**

　　执行如下命令：

mvn archetype:create -DgroupId=me.gacl -DartifactId=spring4-mybatis3 -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false

　　如下图所示：  
　　

　　创建好的项目如下：



　　编辑pom.xml文件

[复制代码](javascript:void(0);)

1<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

2 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

3<modelVersion>4.0.0</modelVersion>

4<groupId>me.gacl</groupId>

5<artifactId>spring4-mybatis3</artifactId>

6<packaging>war</packaging>

7<version>1.0-SNAPSHOT</version>

8<name>spring4-mybatis3 Maven Webapp</name>

9<url>http://maven.apache.org</url>

10<dependencies>

11<dependency>

12<groupId>junit</groupId>

13<artifactId>junit</artifactId>

14<version>3.8.1</version>

15<scope>test</scope>

16</dependency>

17</dependencies>

18<build>

19<finalName>spring4-mybatis3</finalName>

20</build>

21</project>

[复制代码](javascript:void(0);)

　　修改 <name>spring4-mybatis3 Maven Webapp</name> 部分，把" Maven Webapp"这部分包含空格的内容去掉，否则Maven在编译项目时会因为空格的原因导致一些莫名其妙的错误出现，修改成： <name>spring4-mybatis3</name> 。

　　另外，把以下内容删掉：

1<dependency>

2<groupId>junit</groupId>

3<artifactId>junit</artifactId>

4<version>3.8.1</version>

5<scope>test</scope>

6</dependency>

　　这部分是junit的jar包依赖信息，这个版本太低了，我们不使用这个Junit测试版本，修改过后的pom.xml内容如下：

[复制代码](javascript:void(0);)

1<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

2 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

3<modelVersion>4.0.0</modelVersion>

4<groupId>me.gacl</groupId>

5<artifactId>spring4-mybatis3</artifactId>

6<packaging>war</packaging>

7<version>1.0-SNAPSHOT</version>

8<name>spring4-mybatis3</name>

9<url>http://maven.apache.org</url>

10<dependencies>

11

12</dependencies>

13<build>

14<finalName>spring4-mybatis3</finalName>

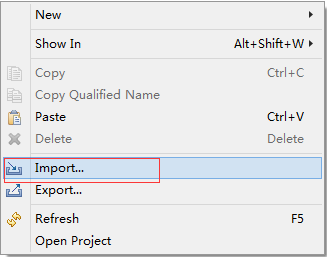
15</build>

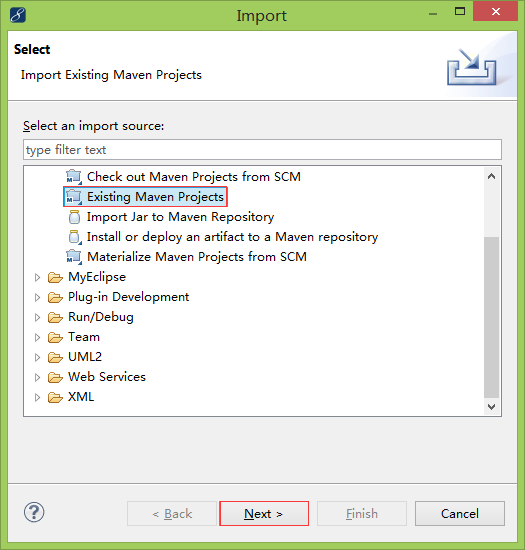
16</project>

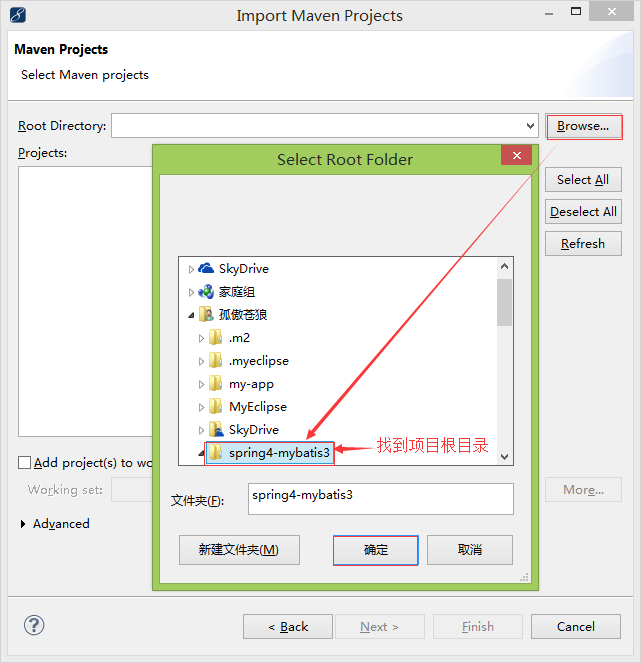
[复制代码](javascript:void(0);)

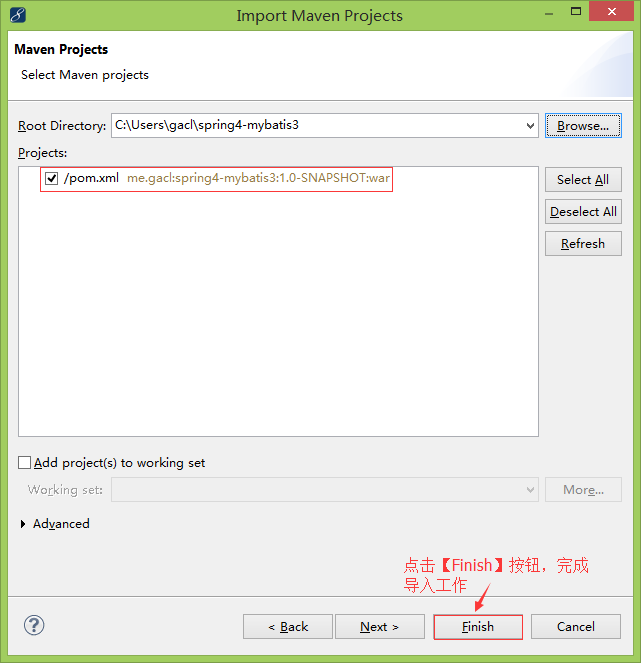
**1.2、将创建好的项目导入MyEclipse中**

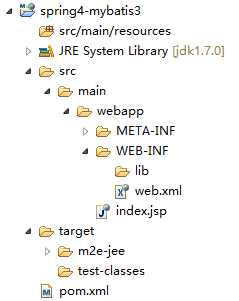
　　具体操作步骤如下图所示：



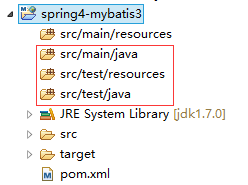








　　手动创建【src/main/java】、【src/test/resources】、【src/test/java】这三个source folder，如下图所示：



　　到此，项目搭建的工作就算是全部完成了。

**二、创建数据库和表(针对MySQL)**

SQL脚本如下：

[复制代码](javascript:void(0);)

CreateDATABASE spring4\_mybatis3;

USE spring4\_mybatis3;

DROPTABLEIFEXISTS t\_user;

CREATETABLE t\_user (

user\_idchar(**32**) NOTNULL,

user\_namevarchar(**30**) DEFAULTNULL,

user\_birthday date DEFAULTNULL,

user\_salary doubleDEFAULTNULL,

PRIMARYKEY (user\_id)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

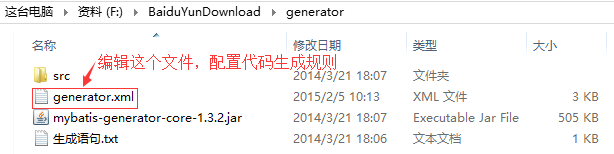
[复制代码](javascript:void(0);)

　　创建好的数据库和表如下：



**三、使用generator工具生成代码**

　　在网上找到了一个generator工具可以根据创建好的数据库表生成MyBatis的表对应的实体类，SQL映射文件和dao，找到generator工具根目录下的generator.xml文件，这个文件是用来配置代码生成规则的，如下图所示：



　　编辑generator.xml文件，内容如下：

[复制代码](javascript:void(0);)

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

2<!DOCTYPE generatorConfiguration PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN" "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">

3<generatorConfiguration>

4<!-- 数据库驱动包位置 -->

5<classPathEntry location="E:\repository\mysql\mysql-connector-java\5.1.34\mysql-connector-java-5.1.34.jar"/>

6<!-- <classPathEntry location="C:\oracle\product\10.2.0\db\_1\jdbc\lib\ojdbc14.jar" />-->

7<context id="DB2Tables" targetRuntime="MyBatis3">

8<commentGenerator>

9<property name="suppressAllComments" value="true"/>

10</commentGenerator>

11<!-- 数据库链接URL、用户名、密码 -->

12<jdbcConnection driverClass="com.mysql.jdbc.Driver" connectionURL="jdbc:mysql://localhost:3306/spring4\_mybatis3" userId="root" password="XDP">

13<!--<jdbcConnection driverClass="oracle.jdbc.driver.OracleDriver" connectionURL="jdbc:oracle:thin:@localhost:1521:orcl" userId="msa" password="msa">-->

14</jdbcConnection>

15<javaTypeResolver>

16<property name="forceBigDecimals" value="false"/>

17</javaTypeResolver>

18<!-- 生成实体类的包名和位置，这里配置将生成的实体类放在me.gacl.domain这个包下 -->

19<javaModelGenerator targetPackage="me.gacl.domain" targetProject="C:\Users\gacl\spring4-mybatis3\src\main\java">

20<property name="enableSubPackages" value="true"/>

21<property name="trimStrings" value="true"/>

22</javaModelGenerator>

23<!-- 生成的SQL映射文件包名和位置，这里配置将生成的SQL映射文件放在me.gacl.mapping这个包下 -->

24<sqlMapGenerator targetPackage="me.gacl.mapping" targetProject="C:\Users\gacl\spring4-mybatis3\src\main\java">

25<property name="enableSubPackages" value="true"/>

26</sqlMapGenerator>

27<!-- 生成DAO的包名和位置，这里配置将生成的dao类放在me.gacl.dao这个包下 -->

28<javaClientGenerator type="XMLMAPPER" targetPackage="me.gacl.dao" targetProject="C:\Users\gacl\spring4-mybatis3\src\main\java">

29<property name="enableSubPackages" value="true"/>

30</javaClientGenerator>

31<!-- 要生成那些表(更改tableName和domainObjectName就可以) -->

32<table tableName="t\_user" domainObjectName="User" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false"/>

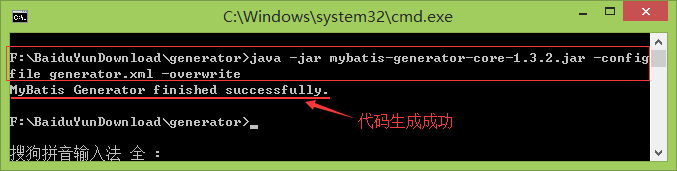
33</context>

34</generatorConfiguration>

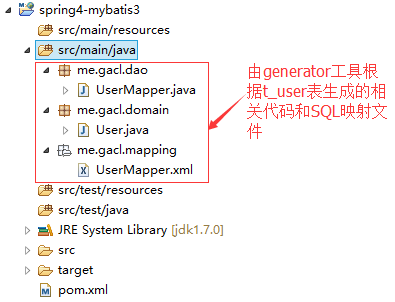
[复制代码](javascript:void(0);)

　　打开命令行窗口，切换到生成工具的根目录下，执行如下命令：

java -jar mybatis-generator-core-1.3.2.jar -configfile generator.xml -overwrite

　　如下图所示：  
　　

　　刚才我们在generator.xml文件中配置将生成的代码和SQL映射文件放到"C:\Users\gacl\spring4-mybatis3\src\main\java"这个目录下，这个目录就是我们的spring4-mybatis3项目所在目录，我们刷新一下src/main/java目录，就可以看到生成的代码和映射文件了，如下图所示：



　　生成的代码和映射文件一行都不用改，可以直接应用到项目当中。下面我们看一眼由generator工具生成的代码和映射文件：

　　1、生成的dao类

[复制代码](javascript:void(0);)

1package me.gacl.dao;

2

3import me.gacl.domain.User;

4

5publicinterface UserMapper {

6int deleteByPrimaryKey(String userId);

7

8int insert(User record);

9

10int insertSelective(User record);

11

12 User selectByPrimaryKey(String userId);

13

14int updateByPrimaryKeySelective(User record);

15

16int updateByPrimaryKey(User record);

17 }

[复制代码](javascript:void(0);)

　　生成的UserMapper是一个接口，里面定义了一些操作t\_user表的增删改查方法。

2、生成的实体类

[复制代码](javascript:void(0);)

1package me.gacl.domain;

2

3import java.util.Date;

4

5publicclass User {

6private String userId;

7

8private String userName;

9

10private Date userBirthday;

11

12private Double userSalary;

13

14public String getUserId() {

15return userId;

16 }

17

18publicvoid setUserId(String userId) {

19this.userId = userId == null ? null : userId.trim();

20 }

21

22public String getUserName() {

23return userName;

24 }

25

26publicvoid setUserName(String userName) {

27this.userName = userName == null ? null : userName.trim();

28 }

29

30public Date getUserBirthday() {

31return userBirthday;

32 }

33

34publicvoid setUserBirthday(Date userBirthday) {

35this.userBirthday = userBirthday;

36 }

37

38public Double getUserSalary() {

39return userSalary;

40 }

41

42publicvoid setUserSalary(Double userSalary) {

43this.userSalary = userSalary;

44 }

45 }

[复制代码](javascript:void(0);)

　　User类是t\_user表的对应的实体类，User类中定义的属性和t\_user表中的字段一一对应。

　　3、生成的SQL映射文件

[复制代码](javascript:void(0);)

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

2<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >

3<mapper namespace="me.gacl.dao.UserMapper">

4<resultMap id="BaseResultMap" type="me.gacl.domain.User">

5<id column="user\_id" property="userId" jdbcType="CHAR"/>

6<result column="user\_name" property="userName" jdbcType="VARCHAR"/>

7<result column="user\_birthday" property="userBirthday" jdbcType="DATE"/>

8<result column="user\_salary" property="userSalary" jdbcType="DOUBLE"/>

9</resultMap>

10<sql id="Base\_Column\_List">

11 user\_id, user\_name, user\_birthday, user\_salary

12</sql>

13<select id="selectByPrimaryKey" resultMap="BaseResultMap" parameterType="java.lang.String">

14 select

15<include refid="Base\_Column\_List"/>

16 from t\_user

17 where user\_id = #{userId,jdbcType=CHAR}

18</select>

19<delete id="deleteByPrimaryKey" parameterType="java.lang.String">

20 delete from t\_user

21 where user\_id = #{userId,jdbcType=CHAR}

22</delete>

23<insert id="insert" parameterType="me.gacl.domain.User">

24 insert into t\_user (user\_id, user\_name, user\_birthday,

25 user\_salary)

26 values (#{userId,jdbcType=CHAR}, #{userName,jdbcType=VARCHAR}, #{userBirthday,jdbcType=DATE},

27 #{userSalary,jdbcType=DOUBLE})

28</insert>

29<insert id="insertSelective" parameterType="me.gacl.domain.User">

30 insert into t\_user

31<trim prefix="(" suffix=")" suffixOverrides=",">

32<if test="userId != null">

33 user\_id,

34</if>

35<if test="userName != null">

36 user\_name,

37</if>

38<if test="userBirthday != null">

39 user\_birthday,

40</if>

41<if test="userSalary != null">

42 user\_salary,

43</if>

44</trim>

45<trim prefix="values (" suffix=")" suffixOverrides=",">

46<if test="userId != null">

47 #{userId,jdbcType=CHAR},

48</if>

49<if test="userName != null">

50 #{userName,jdbcType=VARCHAR},

51</if>

52<if test="userBirthday != null">

53 #{userBirthday,jdbcType=DATE},

54</if>

55<if test="userSalary != null">

56 #{userSalary,jdbcType=DOUBLE},

57</if>

58</trim>

59</insert>

60<update id="updateByPrimaryKeySelective" parameterType="me.gacl.domain.User">

61 update t\_user

62<set >

63<if test="userName != null">

64 user\_name = #{userName,jdbcType=VARCHAR},

65</if>

66<if test="userBirthday != null">

67 user\_birthday = #{userBirthday,jdbcType=DATE},

68</if>

69<if test="userSalary != null">

70 user\_salary = #{userSalary,jdbcType=DOUBLE},

71</if>

72</set>

73 where user\_id = #{userId,jdbcType=CHAR}

74</update>

75<update id="updateByPrimaryKey" parameterType="me.gacl.domain.User">

76 update t\_user

77 set user\_name = #{userName,jdbcType=VARCHAR},

78 user\_birthday = #{userBirthday,jdbcType=DATE},

79 user\_salary = #{userSalary,jdbcType=DOUBLE}

80 where user\_id = #{userId,jdbcType=CHAR}

81</update>

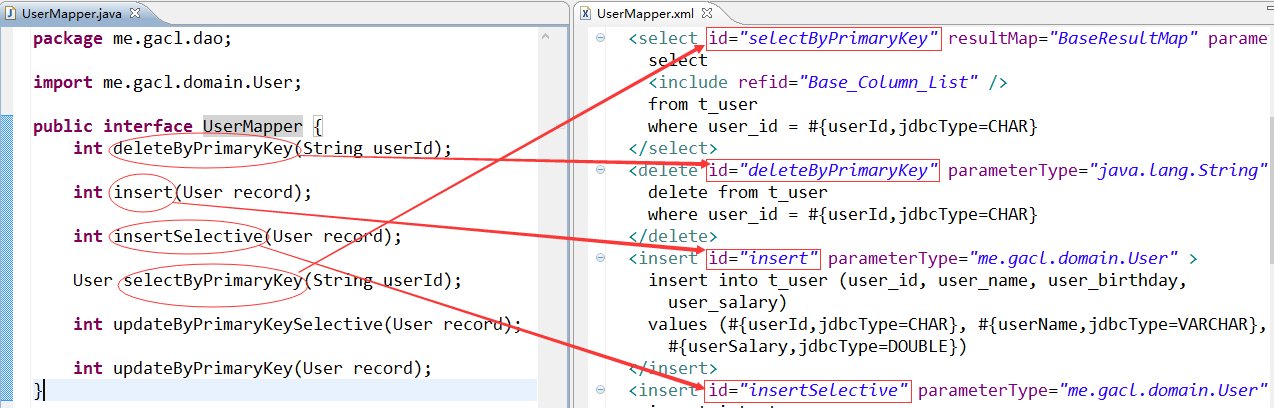
82</mapper>

[复制代码](javascript:void(0);)

　　UserMapper.xml这个文件的内容是编写操作t\_user表的SQL语句，重点说一下UserMapper.xml配置中需要注意的几个小细节问题：

　　1、UserMapper.xml的<mapper>标签的namespace必须是UserMapper接口的全类名，既**<mapper namespace="me.gacl.dao.UserMapper" >**

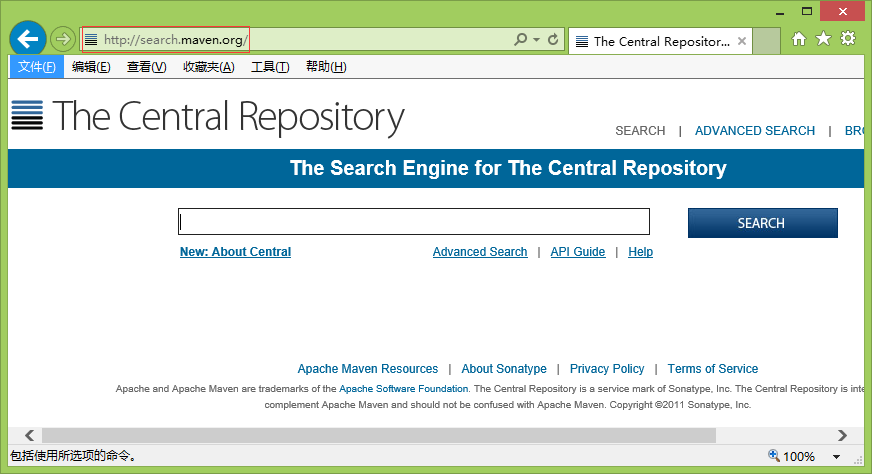
　　2、UserMapper.xml的定义操作数据库的<select><delete><update><insert>这些标签的id属性的值必须和UserMapper接口定义的方法名一致，如下图所示：



　　之所以有上述说的这两点要求，就是为了能够让MyBatis能够根据UserMapper接口和UserMapper.xml文件去自动实现UserMapper接口中定义的相关方法，这样我们就不再需要针对UserMapper接口去编写具体的实现代码了。

**四、Spring与MyBatis整合**

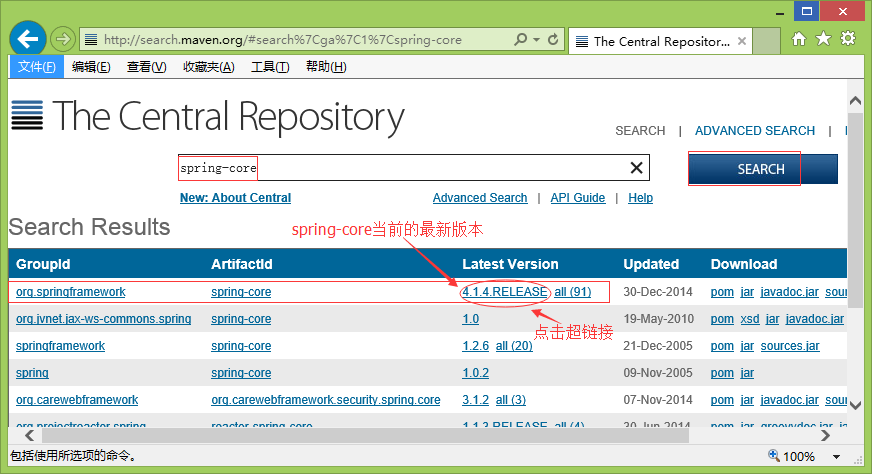
　　首先我们要在项目中加入我们需要的相关jar包，我们可以到Maven的中央仓库：http://search.maven.org/ 找到我们要的相关jar包，如下图所示：



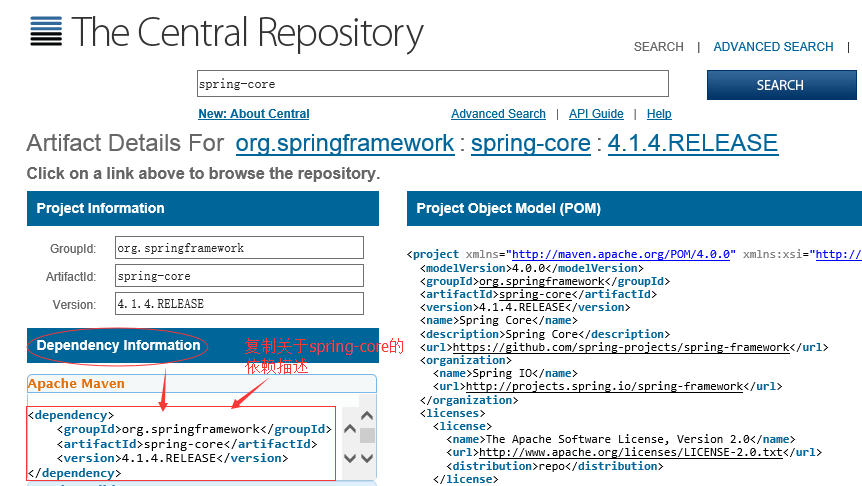
　　我们只需要在搜索框中输入要找的jar包的名称，点击【SEARCH】按钮，就可以找到我们要的jar包了。

**4.1、添加Spring与Mybatis的相关jar包**

　　1、添加spring-core，输入spring-core关键字进行查找，如下图所示：



　　找到关于spring-core的依赖描述信息，如下图所示：



　　将

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>4.1.4.RELEASE</version>

</dependency>

　　复制到项目的pom.xml文件中，如下所示：

[复制代码](javascript:void(0);)

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>me.gacl</groupId>

<artifactId>spring4-mybatis3</artifactId>

<packaging>war</packaging>

<version>1.0-SNAPSHOT</version>

<name>spring4-mybatis3</name>

<url>http://maven.apache.org</url>

<dependencies>

<!-- 添加Spring4.1.4的核心包 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>4.1.4.RELEASE</version>

</dependency>

</dependencies>

<build>

<finalName>spring4-mybatis3</finalName>

</build>

</project>

[复制代码](javascript:void(0);)

　　这样Maven就会自动帮我们从Maven的中央仓库中下载spring-core这个jar包到我们的本地仓库，然后将spring-core这个jar包以及它的相关依赖包加入到我们的项目当中，如下所示：



　　spring4.x与mybatis3.x所需要的相关jar包都可以采用上述所说的方式进行查找，然后添加到项目当中，添加完spring4.x与mybatis3.x相关jar包后，pom.xml文件内容最终如下：

[复制代码](javascript:void(0);)

1<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

2 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

3<modelVersion>4.0.0</modelVersion>

4<groupId>me.gacl</groupId>

5<artifactId>spring4-mybatis3</artifactId>

6<packaging>war</packaging>

7<version>1.0-SNAPSHOT</version>

8<name>spring4-mybatis3</name>

9<url>http://maven.apache.org</url>

10<dependencies>

11<!-- 添加Spring-core包 -->

12<dependency>

13<groupId>org.springframework</groupId>

14<artifactId>spring-core</artifactId>

15<version>4.1.4.RELEASE</version>

16</dependency>

17<!-- 添加spring-context包 -->

18<dependency>

19<groupId>org.springframework</groupId>

20<artifactId>spring-context</artifactId>

21<version>4.1.4.RELEASE</version>

22</dependency>

23<!-- 添加spring-tx包 -->

24<dependency>

25<groupId>org.springframework</groupId>

26<artifactId>spring-tx</artifactId>

27<version>4.1.4.RELEASE</version>

28</dependency>

29<!-- 添加spring-jdbc包 -->

30<dependency>

31<groupId>org.springframework</groupId>

32<artifactId>spring-jdbc</artifactId>

33<version>4.1.4.RELEASE</version>

34</dependency>

35<!-- 为了方便进行单元测试，添加spring-test包 -->

36<dependency>

37<groupId>org.springframework</groupId>

38<artifactId>spring-test</artifactId>

39<version>4.1.4.RELEASE</version>

40</dependency>

41<!--添加spring-web包 -->

42<dependency>

43<groupId>org.springframework</groupId>

44<artifactId>spring-web</artifactId>

45<version>4.1.4.RELEASE</version>

46</dependency>

47<!--添加aspectjweaver包 -->

48<dependency>

49<groupId>org.aspectj</groupId>

50<artifactId>aspectjweaver</artifactId>

51<version>1.8.5</version>

52</dependency>

53<!-- 添加mybatis的核心包 -->

54<dependency>

55<groupId>org.mybatis</groupId>

56<artifactId>mybatis</artifactId>

57<version>3.2.8</version>

58</dependency>

59<!-- 添加mybatis与Spring整合的核心包 -->

60<dependency>

61<groupId>org.mybatis</groupId>

62<artifactId>mybatis-spring</artifactId>

63<version>1.2.2</version>

64</dependency>

65<!-- 添加servlet3.0核心包 -->

66<dependency>

67<groupId>javax.servlet</groupId>

68<artifactId>javax.servlet-api</artifactId>

69<version>3.0.1</version>

70</dependency>

71<dependency>

72<groupId>javax.servlet.jsp</groupId>

73<artifactId>javax.servlet.jsp-api</artifactId>

74<version>2.3.2-b01</version>

75</dependency>

76<!-- jstl -->

77<dependency>

78<groupId>javax.servlet</groupId>

79<artifactId>jstl</artifactId>

80<version>1.2</version>

81</dependency>

82<!-- 添加mysql驱动包 -->

83<dependency>

84<groupId>mysql</groupId>

85<artifactId>mysql-connector-java</artifactId>

86<version>5.1.34</version>

87</dependency>

88<!-- 添加druid连接池包 -->

89<dependency>

90<groupId>com.alibaba</groupId>

91<artifactId>druid</artifactId>

92<version>1.0.12</version>

93</dependency>

94<!-- 添加junit单元测试包 -->

95<dependency>

96<groupId>junit</groupId>

97<artifactId>junit</artifactId>

98<version>4.12</version>

99<scope>test</scope>

100</dependency>

101</dependencies>

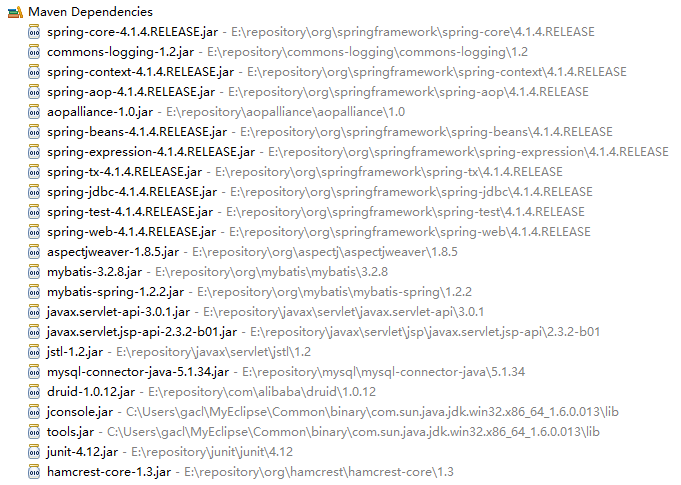
102<build>

103<finalName>spring4-mybatis3</finalName>

104</build>

105</project>

[复制代码](javascript:void(0);)



**4.2、编写相关配置文件**

　　1、**dbconfig.properties**

　　在src/main/resources目录下创建一个dbconfig.properties文件，用于编写连接MySQL数据库的相关信息，dbconfig.properties的内容如下：

driverClassName=com.mysql.jdbc.Driver

validationQuery=SELECT 1

jdbc\_url=jdbc:mysql://localhost:3306/spring4\_mybatis3?useUnicode=true&characterEncoding=UTF-8&zeroDateTimeBehavior=convertToNull

jdbc\_username=root

jdbc\_password=XDP

　　2、**spring.xml(spring框架的配置文件)**

　　在src/main/resources目录下创建一个spring.xml文件，spring.xml文件就是针对Spring框架编写的核心配置文件，spring.xml的内容如下：

[复制代码](javascript:void(0);)

<?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"

xsi:schemaLocation="

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

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

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

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<!-- 引入dbconfig.properties属性文件 -->

<context:property-placeholder location="classpath:dbconfig.properties"/>

<!-- 自动扫描(自动注入)，扫描me.gacl.service这个包以及它的子包的所有使用@Service注解标注的类 -->

<context:component-scan base-package="me.gacl.service"/>

</beans>

[复制代码](javascript:void(0);)

　　我们的spring.xml文件的配置非常简单，就两个配置。  
　　3、**spring-mybatis.xml(spring与mybatis整合的配置文件)**

　　在src/main/resources目录下创建一个spring-mybatis.xml文件，spring-mybatis.xml文件就是针对Spring框架与Mybatis框架整合编写的配置文件，spring-mybatis.xml的内容如下：

[复制代码](javascript:void(0);)

<?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:tx="http://www.springframework.org/schema/tx" xmlns:aop="http://www.springframework.org/schema/aop" xsi:schemaLocation="

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

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

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

http://www.springframework.org/schema/tx/spring-tx-3.0.xsd

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

http://www.springframework.org/schema/aop/spring-aop-3.0.xsd

">

<!-- JNDI方式配置数据源 -->

<!-- <bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean"><property name="jndiName" value="${jndiName}"></property></bean> -->

<!-- ========================================配置数据源========================================= -->

<!-- 配置数据源，使用的是alibaba的Druid(德鲁伊)数据源 -->

<bean name="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close">

<property name="url" value="${jdbc\_url}"/>

<property name="username" value="${jdbc\_username}"/>

<property name="password" value="${jdbc\_password}"/>

<!-- 初始化连接大小 -->

<property name="initialSize" value="0"/>

<!-- 连接池最大使用连接数量 -->

<property name="maxActive" value="20"/>

<!-- 连接池最大空闲 -->

<property name="maxIdle" value="20"/>

<!-- 连接池最小空闲 -->

<property name="minIdle" value="0"/>

<!-- 获取连接最大等待时间 -->

<property name="maxWait" value="60000"/>

<!--

<property name="poolPreparedStatements" value="true" />

<property name="maxPoolPreparedStatementPerConnectionSize" value="33" />

-->

<property name="validationQuery" value="${validationQuery}"/>

<property name="testOnBorrow" value="false"/>

<property name="testOnReturn" value="false"/>

<property name="testWhileIdle" value="true"/>

<!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->

<property name="timeBetweenEvictionRunsMillis" value="60000"/>

<!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->

<property name="minEvictableIdleTimeMillis" value="25200000"/>

<!-- 打开removeAbandoned功能 -->

<property name="removeAbandoned" value="true"/>

<!-- 1800秒，也就是30分钟 -->

<property name="removeAbandonedTimeout" value="1800"/>

<!-- 关闭abanded连接时输出错误日志 -->

<property name="logAbandoned" value="true"/>

<!-- 监控数据库 -->

<!-- <property name="filters" value="stat" /> -->

<property name="filters" value="mergeStat"/>

</bean>

<!-- ========================================分隔线========================================= -->

<!-- ========================================针对myBatis的配置项============================== -->

<!-- 配置sqlSessionFactory -->

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

<!-- 实例化sqlSessionFactory时需要使用上述配置好的数据源以及SQL映射文件 -->

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

<!-- 自动扫描me/gacl/mapping/目录下的所有SQL映射的xml文件, 省掉Configuration.xml里的手工配置

value="classpath:me/gacl/mapping/\*.xml"指的是classpath(类路径)下me.gacl.mapping包中的所有xml文件

UserMapper.xml位于me.gacl.mapping包下，这样UserMapper.xml就可以被自动扫描

-->

<property name="mapperLocations" value="classpath:me/gacl/mapping/\*.xml"/>

</bean>

<!-- 配置扫描器 -->

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

<!-- 扫描me.gacl.dao这个包以及它的子包下的所有映射接口类 -->

<property name="basePackage" value="me.gacl.dao"/>

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

</bean>

<!-- ========================================分隔线========================================= -->

<!-- 配置Spring的事务管理器 -->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

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

</bean>

<!-- 注解方式配置事物 -->

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

<!-- 拦截器方式配置事物 -->

<tx:advice id="transactionAdvice" transaction-manager="transactionManager">

<tx:attributes>

<tx:method name="add\*" propagation="REQUIRED"/>

<tx:method name="append\*" propagation="REQUIRED"/>

<tx:method name="insert\*" propagation="REQUIRED"/>

<tx:method name="save\*" propagation="REQUIRED"/>

<tx:method name="update\*" propagation="REQUIRED"/>

<tx:method name="modify\*" propagation="REQUIRED"/>

<tx:method name="edit\*" propagation="REQUIRED"/>

<tx:method name="delete\*" propagation="REQUIRED"/>

<tx:method name="remove\*" propagation="REQUIRED"/>

<tx:method name="repair" propagation="REQUIRED"/>

<tx:method name="delAndRepair" propagation="REQUIRED"/>

<tx:method name="get\*" propagation="SUPPORTS"/>

<tx:method name="find\*" propagation="SUPPORTS"/>

<tx:method name="load\*" propagation="SUPPORTS"/>

<tx:method name="search\*" propagation="SUPPORTS"/>

<tx:method name="datagrid\*" propagation="SUPPORTS"/>

<tx:method name="\*" propagation="SUPPORTS"/>

</tx:attributes>

</tx:advice>

<aop:config>

<aop:pointcut id="transactionPointcut" expression="execution(\* me.gacl.service..\*Impl.\*(..))"/>

<aop:advisor pointcut-ref="transactionPointcut" advice-ref="transactionAdvice"/>

</aop:config>

<!-- 配置druid监控spring jdbc -->

<bean id="druid-stat-interceptor" class="com.alibaba.druid.support.spring.stat.DruidStatInterceptor">

</bean>

<bean id="druid-stat-pointcut" class="org.springframework.aop.support.JdkRegexpMethodPointcut" scope="prototype">

<property name="patterns">

<list>

<value>me.gacl.service.\*</value>

</list>

</property>

</bean>

<aop:config>

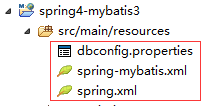
<aop:advisor advice-ref="druid-stat-interceptor" pointcut-ref="druid-stat-pointcut"/>

</aop:config>

</beans>

[复制代码](javascript:void(0);)

　　到此，相关的配置文件算是编写完成了，如下图所示：



**4.3、进行单元测试**

　　经过以上两个步骤，spring4与mybatis3的整合算是全部完成了。接下来我们要做的工作就算进行单元测试，测试一下spring4与mybatis3的整合是否成功。

　　1、在src/main/java目录下创建一个me.gacl.service包，然后在me.gacl.service包创建一个UserServiceI接口，如下所示：

[复制代码](javascript:void(0);)

1package me.gacl.service;

2

3import me.gacl.domain.User;

4

5publicinterface UserServiceI {

6

7/\*\*

8 \* 添加用户

9 \* @param user

10\*/

11void addUser(User user);

12

13/\*\*

14 \* 根据用户id获取用户

15 \* @param userId

16 \* @return

17\*/

18 User getUserById(String userId);

19 }

[复制代码](javascript:void(0);)

　　2、在src/main/java目录下创建一个me.gacl.service.impl包，然后在me.gacl.service.impl包创建一个针对UserServiceI接口的实现类：UserServiceImpl，如下所示：

[复制代码](javascript:void(0);)

1package me.gacl.service.impl;

2

3import org.springframework.beans.factory.annotation.Autowired;

4import org.springframework.stereotype.Service;

5import me.gacl.dao.UserMapper;

6import me.gacl.domain.User;

7import me.gacl.service.UserServiceI;

8

9/\*\*

10 \* @author gacl

11 \* 使用@Service注解将UserServiceImpl类标注为一个service

12 \* service的id是userService

13\*/

14 @Service("userService")

15publicclass UserServiceImpl implements UserServiceI {

16

17/\*\*

18 \* 使用@Autowired注解标注userMapper变量，

19 \* 当需要使用UserMapper时，Spring就会自动注入UserMapper

20\*/

21 @Autowired

22private UserMapper userMapper;//注入dao

23

24 @Override

25publicvoid addUser(User user) {

26 userMapper.insert(user);

27 }

28

29 @Override

30public User getUserById(String userId) {

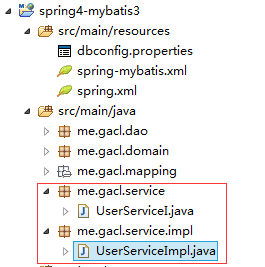
31return userMapper.selectByPrimaryKey(userId);

32 }

33 }

[复制代码](javascript:void(0);)

　　创建好的两个类如下所示：



　　3、在src/test/java目录下编写单元测试类，新建一个me.gacl.test包，然后在这个包下创建一个MyBatisTest类，代码如下：

[复制代码](javascript:void(0);)

1package me.gacl.test;

2

3import java.util.Date;

4import java.util.UUID;

5import me.gacl.domain.User;

6import me.gacl.service.UserServiceI;

7//import me.gacl.service.UserServiceI;

8import org.junit.Before;

9import org.junit.Test;

10import org.springframework.context.ApplicationContext;

11import org.springframework.context.support.ClassPathXmlApplicationContext;

12

13publicclass MyBatisTest {

14

15private UserServiceI userService;

16

17/\*\*

18 \* 这个before方法在所有的测试方法之前执行，并且只执行一次

19 \* 所有做Junit单元测试时一些初始化工作可以在这个方法里面进行

20 \* 比如在before方法里面初始化ApplicationContext和userService

21\*/

22 @Before

23publicvoid before(){

24//使用"spring.xml"和"spring-mybatis.xml"这两个配置文件创建Spring上下文

25 ApplicationContext ac = new ClassPathXmlApplicationContext(new String[]{"spring.xml","spring-mybatis.xml"});

26//从Spring容器中根据bean的id取出我们要使用的userService对象

27 userService = (UserServiceI) ac.getBean("userService");

28 }

29

30 @Test

31publicvoid testAddUser(){

32//ApplicationContext ac = new ClassPathXmlApplicationContext(new String[]{"spring.xml","spring-mybatis.xml"});

33//UserServiceI userService = (UserServiceI) ac.getBean("userService");

34 User user = new User();

35 user.setUserId(UUID.randomUUID().toString().replaceAll("-", ""));

36 user.setUserName("白虎神皇xdp");

37 user.setUserBirthday(new Date());

38 user.setUserSalary(10000D);

39 userService.addUser(user);

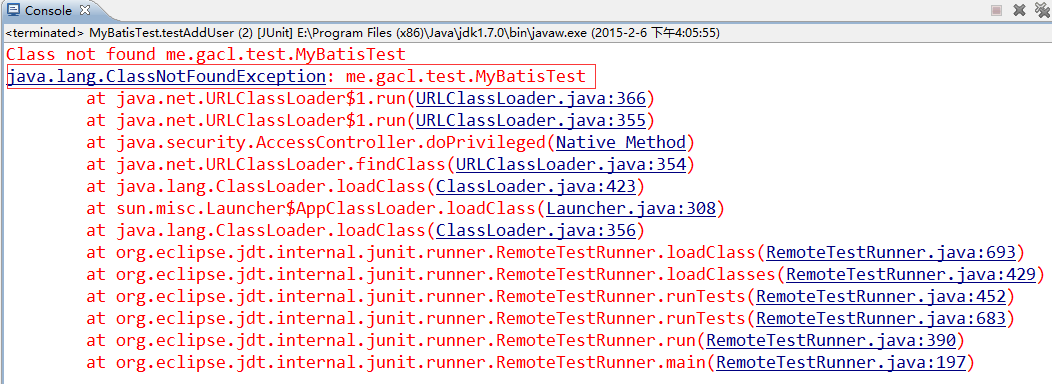
40 }

41

42 }

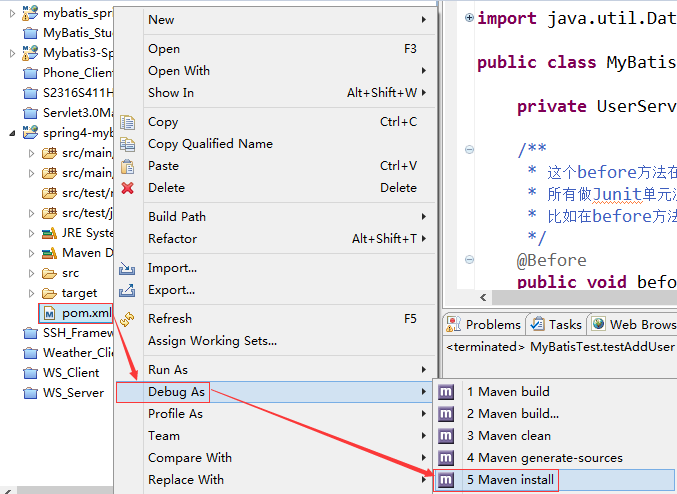
[复制代码](javascript:void(0);)

　　执行单元测试代码，这时候会报如下错误：

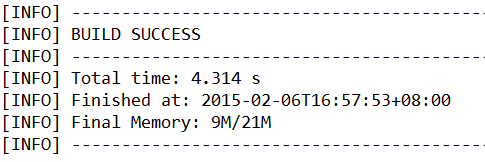


　　错误提示是说没有找到"me.gacl.test.MyBatisTest"这个类，这是因为我们没有使用maven编译项目中的类的缘故。

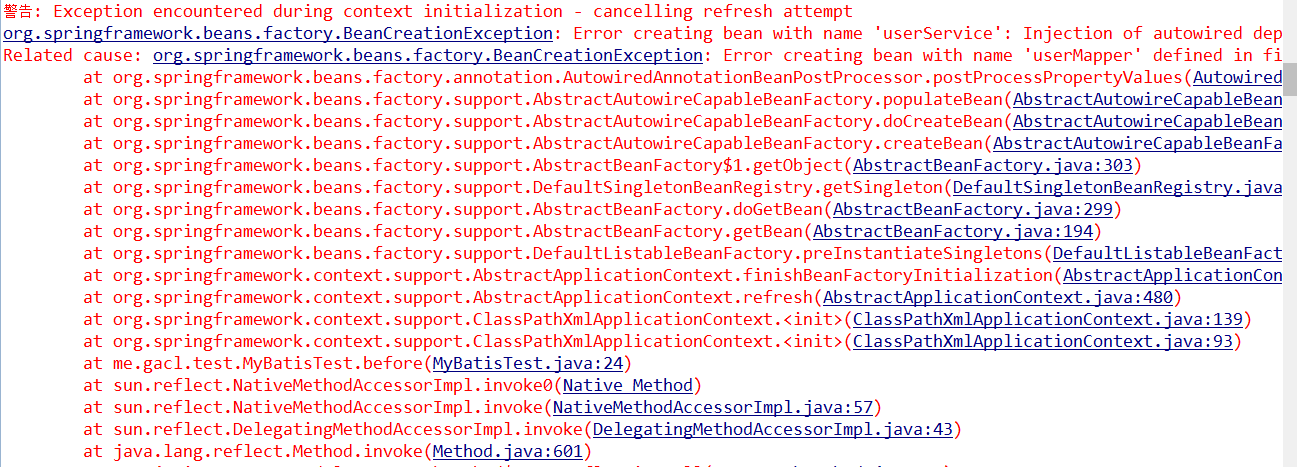
　　下面我们使用Maven编译项目，选中项目的pom.xml文件→【Debug As】→【maven install】，如下所示：



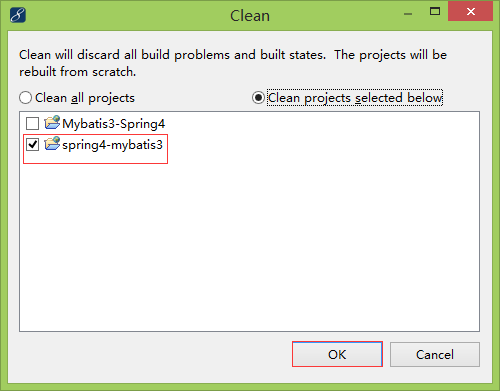
编译结果如下：



　　在这里说一下我执行Maven install之后遇到的问题，第一次执行Maven install命令时，就出现了如下一堆乱七八糟的错误：



　　后来我把项目删掉，再重新导入项目，然后再执行Clean项目操作之后，如下图所示：



　　再执行Maven install操作又可以正常编译通过了，这让我郁闷了好久，这应该不是我项目配置的原因，而是Maven的原因，具体也不知道为啥会这样。反正这算是一种解决办法吧，**如果遇到执行Maven install操作不能正常编译通过的情况：可以尝试采用：Maven clean→Clean项目→Maven install这三个步骤去解决问题**。

　　除了可以用常规的Junit进行单元测试之外，我们还可以使用Spring提供的Junit测试框架进行单元测试，在me.gacl.test下新建一个MyBatisTestBySpringTestFramework类，代码如下：

[复制代码](javascript:void(0);)

1package me.gacl.test;

2

3import java.util.Date;

4import java.util.UUID;

5import me.gacl.domain.User;

6import me.gacl.service.UserServiceI;

7import org.junit.Test;

8import org.junit.runner.RunWith;

9import org.springframework.beans.factory.annotation.Autowired;

10import org.springframework.test.context.ContextConfiguration;

11import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

12

13 @RunWith(SpringJUnit4ClassRunner.class)

14//配置了@ContextConfiguration注解并使用该注解的locations属性指明spring和配置文件之后，

15 @ContextConfiguration(locations = {"classpath:spring.xml", "classpath:spring-mybatis.xml" })

16publicclass MyBatisTestBySpringTestFramework {

17

18//注入userService

19 @Autowired

20private UserServiceI userService;

21

22 @Test

23publicvoid testAddUser(){

24 User user = new User();

25 user.setUserId(UUID.randomUUID().toString().replaceAll("-", ""));

26 user.setUserName("xdp\_gacl\_白虎神皇");

27 user.setUserBirthday(new Date());

28 user.setUserSalary(10000D);

29 userService.addUser(user);

30 }

31

32 @Test

33publicvoid testGetUserById(){

34 String userId = "fb1c5941094e400b975f10d9a9d602a3";

35 User user = userService.getUserById(userId);

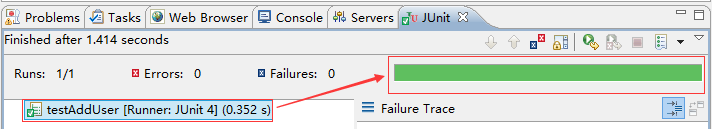
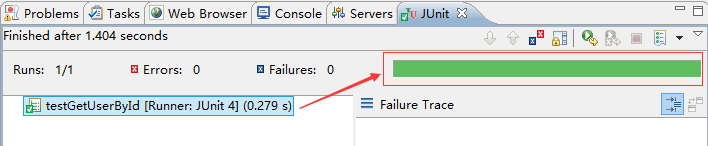
36 System.out.println(user.getUserName());

37 }

38 }

[复制代码](javascript:void(0);)

　　执行这两个测试方法，是可以正常测试通过的，如下所示：

　　到此，我们框架的整合测试工作就算是全部通过了，整合成功。

**4.4、在web服务器中进行测试**

　　1、编辑web.xml文件，添加spring监听器配置项，内容如下：

[复制代码](javascript:void(0);)

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

<web-app xmlns="http://java.sun.com/xml/ns/javaee" 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-app\_3\_0.xsd"

version="3.0">

<welcome-file-list>

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

</welcome-file-list>

<listener>

<description>Spring监听器</description>

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

</listener>

<!-- ContextLoaderListener初始化Spring上下文时需要使用到的contextConfigLocation参数 -->

<context-param>

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

<!-- 配置spring.xml和spring-mybatis.xml这两个配置文件的位置，固定写法 -->

<param-value>classpath:spring.xml,classpath:spring-mybatis.xml</param-value>

</context-param>

</web-app>

[复制代码](javascript:void(0);)

　　2、在UserMapper接口中添加一个获取所有用户信息的getAllUser()方法，如下所示：

[复制代码](javascript:void(0);)

1package me.gacl.dao;

2

3import java.util.List;

4

5import me.gacl.domain.User;

6

7publicinterface UserMapper {

8int deleteByPrimaryKey(String userId);

9

10int insert(User record);

11

12int insertSelective(User record);

13

14 User selectByPrimaryKey(String userId);

15

16int updateByPrimaryKeySelective(User record);

17

18int updateByPrimaryKey(User record);

19

20/\*\*获取所有用户信息

21 \* @return List<User>

22\*/

23 List<User> getAllUser();

24 }

[复制代码](javascript:void(0);)

　　3、在UserMapper.xml文件中编写getAllUser()方法要执行的SQL语句，如下所示：

[复制代码](javascript:void(0);)

<?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="me.gacl.dao.UserMapper">

<resultMap id="BaseResultMap" type="me.gacl.domain.User">

<id column="user\_id" property="userId" jdbcType="CHAR"/>

<result column="user\_name" property="userName" jdbcType="VARCHAR"/>

<result column="user\_birthday" property="userBirthday" jdbcType="DATE"/>

<result column="user\_salary" property="userSalary" jdbcType="DOUBLE"/>

</resultMap>

<sql id="Base\_Column\_List">

user\_id, user\_name, user\_birthday, user\_salary

</sql>

<select id="selectByPrimaryKey" resultMap="BaseResultMap" parameterType="java.lang.String">

select

<include refid="Base\_Column\_List"/>

from t\_user

where user\_id = #{userId,jdbcType=CHAR}

</select>

<delete id="deleteByPrimaryKey" parameterType="java.lang.String">

delete from t\_user

where user\_id = #{userId,jdbcType=CHAR}

</delete>

<insert id="insert" parameterType="me.gacl.domain.User">

insert into t\_user (user\_id, user\_name, user\_birthday,

user\_salary)

values (#{userId,jdbcType=CHAR}, #{userName,jdbcType=VARCHAR}, #{userBirthday,jdbcType=DATE},

#{userSalary,jdbcType=DOUBLE})

</insert>

<insert id="insertSelective" parameterType="me.gacl.domain.User">

insert into t\_user

<trim prefix="(" suffix=")" suffixOverrides=",">

<if test="userId != null">

user\_id,

</if>

<if test="userName != null">

user\_name,

</if>

<if test="userBirthday != null">

user\_birthday,

</if>

<if test="userSalary != null">

user\_salary,

</if>

</trim>

<trim prefix="values (" suffix=")" suffixOverrides=",">

<if test="userId != null">

#{userId,jdbcType=CHAR},

</if>

<if test="userName != null">

#{userName,jdbcType=VARCHAR},

</if>

<if test="userBirthday != null">

#{userBirthday,jdbcType=DATE},

</if>

<if test="userSalary != null">

#{userSalary,jdbcType=DOUBLE},

</if>

</trim>

</insert>

<update id="updateByPrimaryKeySelective" parameterType="me.gacl.domain.User">

update t\_user

<set >

<if test="userName != null">

user\_name = #{userName,jdbcType=VARCHAR},

</if>

<if test="userBirthday != null">

user\_birthday = #{userBirthday,jdbcType=DATE},

</if>

<if test="userSalary != null">

user\_salary = #{userSalary,jdbcType=DOUBLE},

</if>

</set>

where user\_id = #{userId,jdbcType=CHAR}

</update>

<update id="updateByPrimaryKey" parameterType="me.gacl.domain.User">

update t\_user

set user\_name = #{userName,jdbcType=VARCHAR},

user\_birthday = #{userBirthday,jdbcType=DATE},

user\_salary = #{userSalary,jdbcType=DOUBLE}

where user\_id = #{userId,jdbcType=CHAR}

</update>

<!-- ==============以下内容是根据自身业务扩展的内容======================= -->

<!-- select标签的id属性与UserMapper接口中定义的getAllUser方法要一模一样 -->

<select id="getAllUser" resultMap="BaseResultMap">

select user\_id, user\_name, user\_birthday, user\_salary from t\_user

</select>

</mapper>

[复制代码](javascript:void(0);)

　　4、在UserServiceI接口中也添加一个getAllUser()方法，如下：

[复制代码](javascript:void(0);)

1package me.gacl.service;

2

3import java.util.List;

4

5import me.gacl.domain.User;

6

7publicinterface UserServiceI {

8

9/\*\*

10 \* 添加用户

11 \* @param user

12\*/

13void addUser(User user);

14

15/\*\*

16 \* 根据用户id获取用户

17 \* @param userId

18 \* @return

19\*/

20 User getUserById(String userId);

21

22/\*\*获取所有用户信息

23 \* @return List<User>

24\*/

25 List<User> getAllUser();

26 }

[复制代码](javascript:void(0);)

　　5、在UserServiceImpl类中实现getAllUser方法，如下：

[复制代码](javascript:void(0);)

1package me.gacl.service.impl;

2

3import java.util.List;

4

5import org.springframework.beans.factory.annotation.Autowired;

6import org.springframework.stereotype.Service;

7import me.gacl.dao.UserMapper;

8import me.gacl.domain.User;

9import me.gacl.service.UserServiceI;

10

11/\*\*

12 \* @author gacl

13 \* 使用@Service注解将UserServiceImpl类标注为一个service

14 \* service的id是userService

15\*/

16 @Service("userService")

17publicclass UserServiceImpl implements UserServiceI {

18

19/\*\*

20 \* 使用@Autowired注解标注userMapper变量，

21 \* 当需要使用UserMapper时，Spring就会自动注入UserMapper

22\*/

23 @Autowired

24private UserMapper userMapper;//注入dao

25

26 @Override

27publicvoid addUser(User user) {

28 userMapper.insert(user);

29 }

30

31 @Override

32public User getUserById(String userId) {

33return userMapper.selectByPrimaryKey(userId);

34 }

35

36 @Override

37public List<User> getAllUser() {

38return userMapper.getAllUser();

39 }

40 }

[复制代码](javascript:void(0);)

　　6、在src/main/java目录下创建一个me.gacl.web.controller包，然后在me.gacl.web.controller下创建一个UserServlet，如下：

[复制代码](javascript:void(0);)

1package me.gacl.web.controller;

2

3import java.io.IOException;

4import java.util.List;

5

6import javax.servlet.ServletException;

7import javax.servlet.annotation.WebServlet;

8import javax.servlet.http.HttpServlet;

9import javax.servlet.http.HttpServletRequest;

10import javax.servlet.http.HttpServletResponse;

11

12import org.springframework.context.ApplicationContext;

13import org.springframework.web.context.support.WebApplicationContextUtils;

14

15import me.gacl.domain.User;

16import me.gacl.service.UserServiceI;

17

18/\*\*

19 \* @author gacl

20 \* @WebServlet是Servlet3.0提供的注解，目的是将一个继承了HttpServlet类的普通java类标注为一个Servlet

21 \* UserServlet使用了@WebServlet标注之后，就不需要在web.xml中配置了

22\*/

23 @WebServlet("/UserServlet")

24publicclass UserServlet extends HttpServlet {

25

26//处理业务逻辑的userService

27private UserServiceI userService;

28

29publicvoid doGet(HttpServletRequest request, HttpServletResponse response)

30throws ServletException, IOException {

31//获取所有的用户信息

32 List<User> lstUsers = userService.getAllUser();

33 request.setAttribute("lstUsers", lstUsers);

34 request.getRequestDispatcher("/index.jsp").forward(request, response);

35 }

36

37publicvoid doPost(HttpServletRequest request, HttpServletResponse response)

38throws ServletException, IOException {

39this.doGet(request, response);

40 }

41

42publicvoid init() throws ServletException {

43//在Servlet初始化时获取Spring上下文对象(ApplicationContext)

44 ApplicationContext ac = WebApplicationContextUtils.getWebApplicationContext(this.getServletContext());

45//从ApplicationContext中获取userService

46 userService = (UserServiceI) ac.getBean("userService");

47 }

48 }

[复制代码](javascript:void(0);)

　　7、编辑index.jsp页面，用于展示查询到的用户信息，内容如下：

[复制代码](javascript:void(0);)

<%@ page language="java" pageEncoding="UTF-8"%>

<%--引入JSTL核心标签库 --%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE html>

<html>

<head>

<title>显示用户信息</title>

<style type="text/css">

table,td{

border: 1px solid;

border-collapse: collapse;

}

</style>

</head>

<body>

<table>

<tr>

<td>用户ID</td>

<td>用户名</td>

<td>用户生日</td>

<td>工资</td>

</tr>

<%--遍历lstUsers集合中的User对象 --%>

<c:forEach var="user" items="${lstUsers}">

<tr>

<td>${user.userId}</td>

<td>${user.userName}</td>

<td>${user.userBirthday}</td>

<td>${user.userSalary}</td>

</tr>

</c:forEach>

</table>

</body>

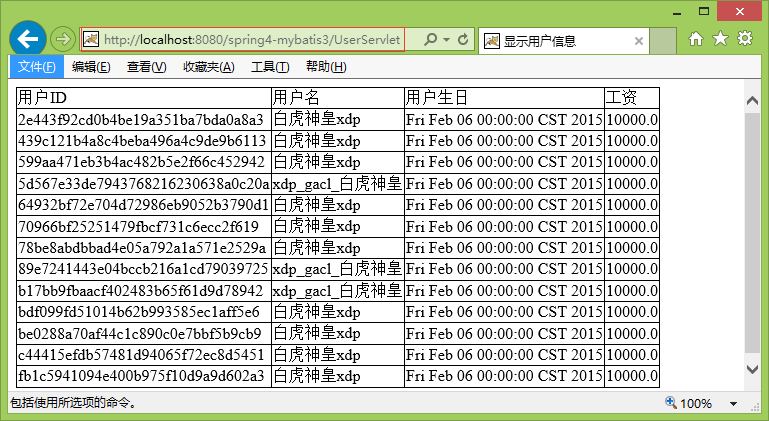
</html>

[复制代码](javascript:void(0);)

　　8、执行maven install命令编译项目，然后将项目部署到tomcat服务器中运行，注意，由于要使用Servlet3.0，所以必须将项目部署到tomcat7.x以上的服务器中去运行，如下所示：

http://images.cnitblog.com/blog/289233/201502/062357104375911.png

　　输入地址：http://localhost:8080/spring4-mybatis3/UserServlet 访问UserServlet，访问结果如下：



　　可以看到，t\_user表中的用户信息全部查询出来显示到页面上了。这样在web服务器中的测试也正常通过了。

　　以上就是Spring4.x与MyBatis3.x整合的全部内容了。编写这个整合例子花了不少时间，使用Maven编译时总是出现莫名其妙的问题，有时候成功，有时候失败，反正很莫名其妙。**如果遇到执行Maven install操作不能正常编译通过的情况：可以尝试采用：Maven clean→Clean项目→Maven install这三个步骤去解决问题**