## Mybatis 开发环境搭建，

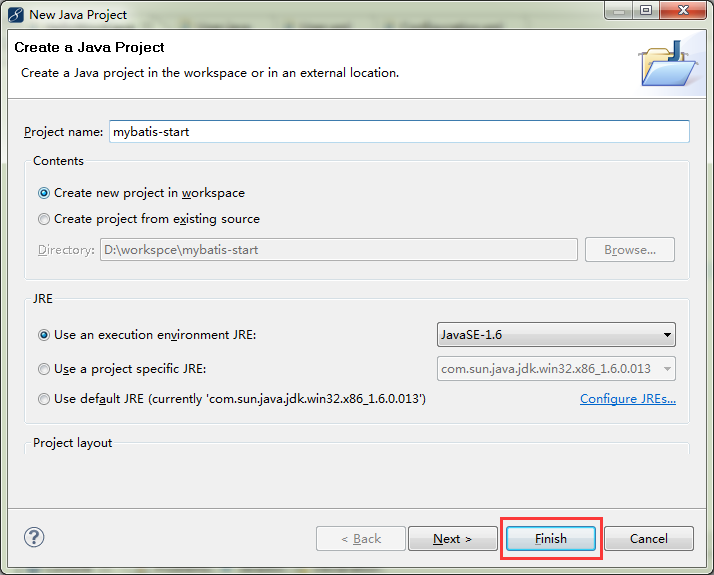
Mybatis 开发环境搭建，选择: MyEclipse8.5 版本，mysql 5.5, jdk 1.8, mybatis3.2.3.jar 包。这些软件工具均可以到各自的官方网站上下载。

整个过程在概如下，

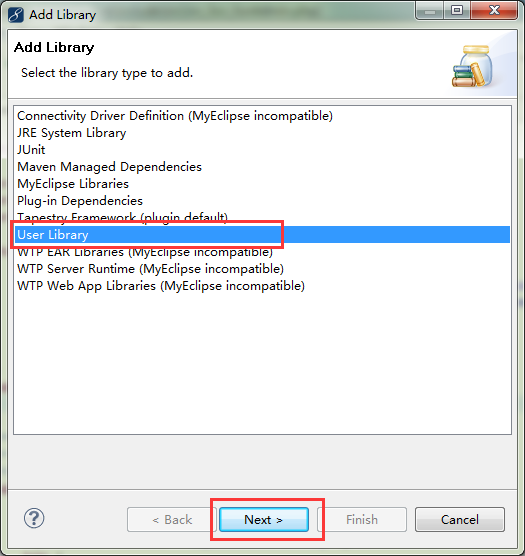
1. 在本教程中，建立 [java](http://www.yiibai.com/java/) 工程，但一般都是开发 web 项目，这个系列教程最后是 web，但这里为了方便学习，本教程前面建立的都是 java 工程。  
2. 将 mybatis-3.2.3.jar，mysql-connector-java-5.1.25-bin.jar 创建两个用户自定库（User Libary）：mysql-connector 和 mybatis；  
3. 创建 [mysql](http://www.yiibai.com/mysql/)测试数据库和用户表, 注意，数据库使用的是 utf-8 编码。以解决不必要的中文乱码问题。

### 一、创建 Java 工程

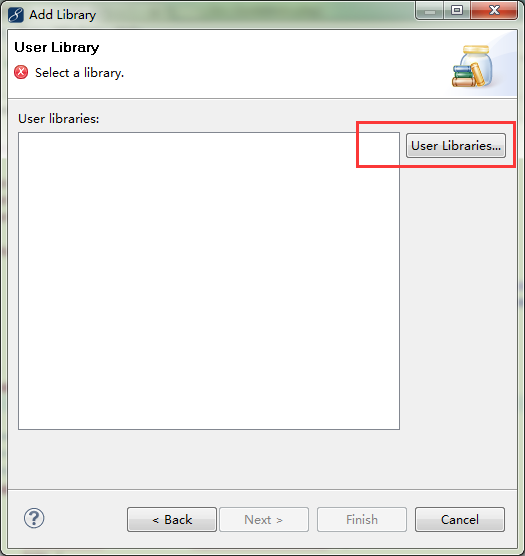
首先建立一个名字为 Helloword 的 java project。选择 "File" -> "New" ->"Java Project"，如下图所示：



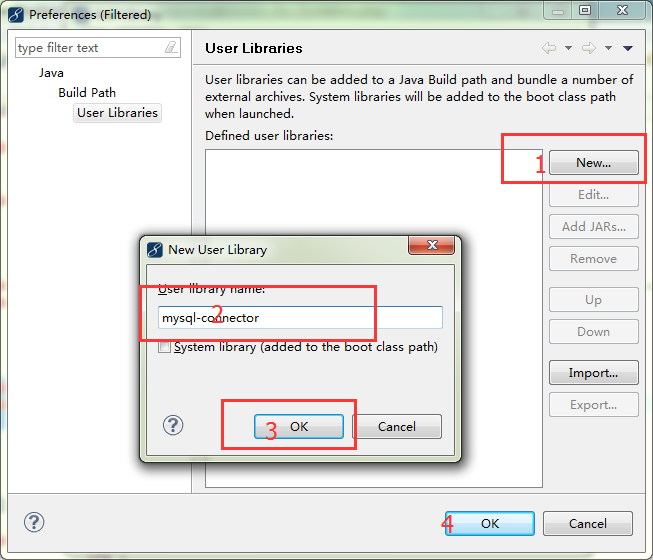
这样就创建了一个 Java 工程了，我们继续下一步。接下来我们在 mybatis-start 项目中加入两个所需的程序库：mysql-connector 和 mybatis，右键点击 "mybatis-start" 项目，从弹出的菜单中选择： "Build Path" -> "Add Libaries..." ，如下图所示：



从中选择“User Library"，点击“Next>"，创建两个类库为：mysql-connector 和 mybatis，如下所示：



点击“User Libararies..."，继续下一步，创建一个新的类库，如下图所示：

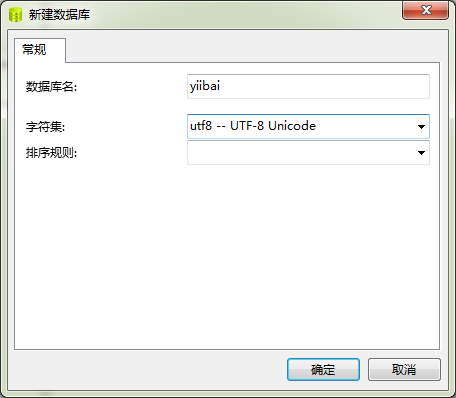


到此用户的一个类库创建完成，以相同的方式来创建另一个类库：

### 二、创建数据库和 User 表

创建所需的数据库和表，要创建的数据库是：yiibai，并在 yiibai 数据库创建一个表：user，如下图所示：

创建数据库：yiibai，使用 utf-8 编码。



接下我们创建一个表：user，并插入一条记录信息，其结构如下所示：

CREATE TABLE `user` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`name` varchar(64) NOT NULL DEFAULT '',

`dept` varchar(254) NOT NULL DEFAULT '',

`website` varchar(254) DEFAULT '',

`phone` varchar(16) NOT NULL DEFAULT '',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=latin1;

-- ----------------------------

-- Records of user

-- ----------------------------

INSERT INTO `user` VALUES ('1', 'yiibai', 'Tech', 'http://www.yiibai.com', '13800009988');

### 三、创建 Mybatis 配置文件

到此为止，前期准备工作就完成了。下面开始真正配置 mybatis-start 项目。设置 mybatis 配置文件: Configure.xml, 在 src/config 目录下建立此文件，内容如下:

<?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>

<typeAliases>

<typeAlias alias="User" type="com.yiibai.mybatis.models.User" />

</typeAliases>

<environments default="development">

<environment id="development">

<transactionManager type="JDBC" />

<dataSource type="POOLED">

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

<property name="url" value="jdbc:mysql://127.0.0.1:3306/yiibai" />

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

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

</dataSource>

</environment>

</environments>

<mappers>

<!-- // power by http://www.yiibai.com -->

<mapper resource="com/yiibai/mybatis/models/User.xml" />

</mappers>

</configuration>

### 四、创建实体类和映射文件

首先创建一个包：com.yiibai.mybatis.models，并在下创建与数据库表对应的 User.java 类及其映射文件：User.xml，详细如下图所示：

package com.yiibai.mybatis.models;

public class User {

private int id;

private String name;

private String dept;

private String phone;

private String website;

public String getWebsite() {

return website;

}

public void setWebsite(String website) {

this.website = website;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDept() {

return dept;

}

public void setDept(String dept) {

this.dept = dept;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

}

同时建立这个 User 类对应的映射文件 User.xml，详细如下代码所示：

<?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.yiibai.mybatis.models.UserMapper">

<select id="GetUserByID" parameterType="int" resultType="User">

select \* from `user` where id = #{id}

</select>

</mapper>

#### 下面是对这几个配置文件一点解释说明： 1、配置文件 Configure.xml 是 mybatis 用来建立 sessionFactory，里面主要包含了数据库连接相关内容，还有 java 类所对应的别名，比如：<typeAlias alias="User" type="com.yiibai.mybatis.models.User"/> 这个别名非常重要，在具体的类的映射中，比如：User.xml 中 resultType 就是对应这个。要保持一致，这里的 resultType 还有另外单独的定义方式，后面学习到我们再详细介绍说明。 2:Configure.xml 里面 的<mapper resource="com/yiibai/mybatis/models/User.xml"/>是包含要映射的类的 xml 配置文件[关联起来]。 3、在User.xml 文件里面主要是定义各种 SQL 语句，以及这些语句的参数，以及要返回的类型等等[注意的是：id=”值”]🡪可以做一个映射接口，里的方法和id的值对应，也可使用sql的注释。

### 五、运行程序测试结果

在 src 源码目录下建立 一个类叫作：HelloWord, 来运行测试配置环境是否成功，具体代码如下示：

import java.io.Reader;

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.yiibai.mybatis.models.\*;

/\*\*

\*

\* @author yiibai

\* @copyright http://www.yiibai.com

\* @date 2015/09/22

\*/

/\*可以抽成一个工具类[有提供SqlSessionFactory，SqlSession，close]\*/

public class HelloWord {

private static SqlSessionFactory sqlSessionFactory;

private static Reader reader;

static {

try {

/\*获取config/Configure.xml配置的连接数据库的要素\*/

reader = Resources.getResourceAsReader("config/Configure.xml");

/\*通过配置的信息，获取到sqlSessionFactory 工厂，下一步就是会话了\*/

sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);

} catch (Exception e) {

e.printStackTrace();

}

}

/\*获取一个会话就可以和数据操作了！记得要管，虽然说一个会话可以多个操作\*/

public static SqlSessionFactory getSession() {

return sqlSessionFactory;

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// TODO Auto-generated method stub

SqlSession session = sqlSessionFactory.openSession();

try {

User user = (User) session.selectOne(

"com.yiibai.mybatis.models.UserMapper.GetUserByID", 1);//1就是参数，相当于？->现在User.xml在配置id = #{id}

if(user!=null){

String userInfo = "名字："+user.getName()+", 所属部门："+user.getDept()+", 主页："+user.getWebsite();

System.out.println(userInfo);

}

} finally {

session.close();

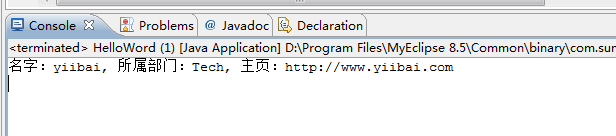
}

}

}

现在运行这个程序，不是得到查询结果了？正确的输出结果应该如下：

名字：yiibai, 所属部门：Tech, 主页：http://www.yiibai.com



恭喜你，环境搭建配置成功，在接下来章节，我们将学习 Mybatis 的操作方式：增删改查。

Jar 包下载：<http://pan.baidu.com/s/1bnyRJ9H>

**易百教程移动端**：请扫描本页面底部(右侧)二维码并关注微信公众号，回复："**教程**" 选择相关教程阅读或直接访问：http://m.yiibai.com 。

## 注释，配置映射文件sql语句，接口映射实体类，

在上一章中，我们已经搭建了 myeclipse,mybatis,mysql 的开发环境，并且实现了 mybatis 的一个简单的查询。要注意的是，这种方式是用 SqlSession 实例来直接执行在User.xml文件中映射的 SQL 语句：  
session.selectOne("com.yiibai.mybatis.models.UserMapper.getUserByID", 1)，但是还有比这更简单的方法，使用合理描述参数和SQL语句返回值的接口(比如：IUser.class)，这样现在就可以不使用类似User.xml配置文件，至此更简单，代码更安全，不容易发生的字符串文字和转换的错误，下面是项目创建的详细过程:

### 1、创建一个接口：IUser，并在其中声明对应的操作方法

在 src 源码目录下创建一个包：com.yiibai.mybatis.dao，并建立接口类 IUser 及一个方法, 在方法上面，我们使用了一个SQL注释，内容如下：

package com.yiibai.mybatis.dao;

/\*sql查询的注释[还有其他的sq的注释，就是为减少对应的实体类的配置的xml文件]\*/

import org.apache.ibatis.annotations.Select;

import com.yiibai.mybatis.models.User;

/\*\*

\* @author yiibai.com

\*/

public interface IUser {

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

public User getUserByID(int id);

}

Java

请注意，这里面代码有一个方法名 getUserByID 必须与 User.xml 里面配置的 select 的 id 对应(<select id="getUserByID">)同名，虽使用注解映射不需要User.xml。

### 2、创建对应映射接口 SQL 语句

首先配置 MyBatis 所需的数据连接文件，这里创建一个文件： src/config/Configure.xml ，其内容如下：

<?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>

<typeAliases>

<typeAlias alias="User" type="com.yiibai.mybatis.models.User" />

</typeAliases>

<environments default="development">

<environment id="development">

<transactionManager type="JDBC" />

<dataSource type="POOLED">

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

<property name="url" value="jdbc:mysql://127.0.0.1:3306/testdb" />

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

<property name="password" value="123456" />

</dataSource>

</environment>

</environments>

<mappers>

<!-- // power by http://www.yiibai.com 注释掉咯...

在使用sql注释：这里可以省略，也可以不省略，但id

<mapper resource="com/yiibai/mybatis/models/User.xml" />-->

</mappers>

</configuration>

XML

在包：com.yiibai.mybatis.models 下创建一个User.java 类文件，与上一节中 User 类代码相同，这里只是拷贝过来，User.java具体的代码内容如下：

package com.yiibai.mybatis.models;

public class User {

private int id;

private String name;

private String dept;

private String phone;

private String website;

public String getWebsite() {

return website;

}

public void setWebsite(String website) {

this.website = website;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDept() {

return dept;

}

public void setDept(String dept) {

this.dept = dept;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

}

Java

与 User.java 对应的 XML 配置文件：User.xml 可以少省略，不用创建，本节讲解的就是如何消灭类似于User.xml的配置文件。

### 3、测试接口映射

我们在 src 这个目录下创建一个类：Main.java，用来测试整个配置和程序运行结果，有关 Main.java 的代码详细内容如下：

import java.io.Reader;

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.yiibai.mybatis.dao.IUser;

import com.yiibai.mybatis.models.User;

public class Main {

private static SqlSessionFactory sqlSessionFactory;

private static Reader reader;

static {

try {

reader = Resources.getResourceAsReader("config/Configure.xml");

sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);

sqlSessionFactory.getConfiguration().addMapper(IUser.class);

} catch (Exception e) {

e.printStackTrace();

}

}

public static SqlSessionFactory getSession() {

return sqlSessionFactory;

}

public static void main(String[] args) {

SqlSession session = sqlSessionFactory.openSession();

try {

IUser iuser = session.getMapper(IUser.class);

User user = iuser.getUserByID(1);

System.out.println("名字："+user.getName());

System.out.println("所属部门："+user.getDept());

System.out.println("主页："+user.getWebsite());

} finally {

session.close();

}

}

}

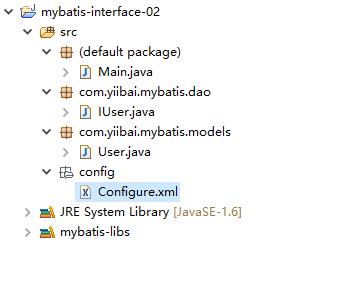
Java

运行上面示例代码，得到以下结果 -

名字：New name, 所属部门：Tech, 主页：http://www.yiibai.com

Shell

最后补充，整个项目 *mybatis-interface-02* 的结构如下图所示：



## Mybatis增删改查（CURD）

前面的小节我们已经讲到用接口的方式编程。使用这种方式，需要注意的一个地方就是，在User.xml 配置文件中，mapper namespace="com.yiibai.mybatis.inter.IUser" ，命名空间对应非常重要，名称不能有错，必须与我们定义的 package 和 接口一致。如果不一致就会出错，这一章主要在上一讲基于接口编程的基础上完成如下操作:

1. 使用 mybatis 查询用户数据(读取用户列表)
2. 使用 mybatis 增加用户数据
3. 使用 mybatis 更新用户数据
4. 使用 mybatis 删除用户数据

查询数据，前面已经讲过简单的查询单个用户数据，在这里将查询出用户列表，  
要查询出列表，也就是返回 List, 在我们这个例子中也就是List<User> , 要以这种方式返回数据，需要在User.xml里面配置返回的类型 resultMap, 注意不是 resultType, 而这个resultMap 所对应的应该是我们自己配置。

在此示例中，我们需要使用到以下表：

接下我们创建一个表：user，并插入一条记录信息，其结构如下所示：

CREATE TABLE `user` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`name` varchar(64) NOT NULL DEFAULT '',

`dept` varchar(254) NOT NULL DEFAULT '',

`website` varchar(254) DEFAULT '',

`phone` varchar(16) NOT NULL DEFAULT '',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=latin1;

-- ----------------------------

-- Records of user

-- ----------------------------

INSERT INTO `user` VALUES ('1', 'yiibai', 'Tech', 'http://www.yiibai.com', '13800009988');

SQL

### 1、创建工程并配置所需环境

我们首先来创建一个工程：*mybatis-curd-03*，与第一节中介绍的环境配置一样，加入所需的 jar 包：mysql-connector 和 mybatis3.jar。配置 src/config/Configure.xml，其文件内容如下：

<?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>

<typeAliases>

<typeAlias alias="User" type="com.yiibai.mybatis.models.User" />

</typeAliases>

<environments default="development">

<environment id="development">

<transactionManager type="JDBC" />

<dataSource type="POOLED">

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

<property name="url" value="jdbc:mysql://127.0.0.1:3306/testdb" />

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

<property name="password" value="123456" />

</dataSource>

</environment>

</environments>

<mappers>

<!-- // power by http://www.yiibai.com -->

<mapper resource="com/yiibai/mybatis/models/User.xml" />

</mappers>

</configuration>

XML

### 2、创建 Java 类和接口

在这里需要创建一个类和一个接口：User.java类和IUser.java接口，User.java类位于包 com.yiibai.mybatis.models 下，User.java类代码内容如下：

package com.yiibai.mybatis.models;

public class User {

private int id;

private String name;

private String dept;

private String phone;

private String website;

public String getWebsite() {

return website;

}

public void setWebsite(String website) {

this.website = website;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDept() {

return dept;

}

public void setDept(String dept) {

this.dept = dept;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

}

Java

IUser.java接口位于包com.yiibai.mybatis.dao 下，IUser.java接口代码内容如下：

package com.yiibai.mybatis.dao;

import java.util.List;

import org.apache.ibatis.annotations.Select;

import com.yiibai.mybatis.models.User;

/\*\*

\* @author yiibai sql和配置映射的xml建立联系！

\*/

public interface IUser {

//sql注解[可以不用配置User的映射xml?]

//@Select("select \* from user where id= #{id}")

//public User getUserByID(int id);

/\*获取用户列表\*/

public List<User> getUserList();

/\*插入新的用户\*/

public void insertUser(User user);

/\*修改用户\*/

public void updateUser(User user);

/\*删除用户\*/

public void deleteUser(int userId);

/\*获取用户[比如登录的时候]\*/

public User getUser(int id);

}

Java

这里还需要一个XML文件，与前一小节中一样，使用的是 User.xml，在这我们分别对应了增删改查的操作(每一个操作的 ID 对应于IUser接口的方法)，其内容如下：

*User.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.yiibai.mybatis.dao.IUser">

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

resultType="com.yiibai.mybatis.models.User">

SELECT \*

FROM USER

WHERE id = #{userId}

</select>

<insert id="insertUser" parameterType="User">

INSERT INTO USER(name,

dept, website,phone)

VALUES(#{name}, #{dept}, #{website}, #{phone})

</insert>

<select id="getUserList" resultType="com.yiibai.mybatis.models.User">

SELECT \* FROM USER

</select>

<update id="updateUser" parameterType="User">

UPDATE USER

SET

name=

#{name},

dept = #{dept},

website = #{website},

phone = #{phone}

WHERE

id =

#{id}

</update>

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

DELETE FROM USER WHERE id = #{id}

</delete>

</mapper>

XML

测试程序主类：

import java.io.Reader;

//这个是格式转换的类 MessageFormat.format(“{0}”,3);//把3替换{0}

import java.text.MessageFormat;

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.yiibai.mybatis.dao.IUser;

import com.yiibai.mybatis.models.User;

public class Main {

private static SqlSessionFactory sqlSessionFactory;

private static Reader reader;

static {

try {

//读取配置文件，连接数据库的要素！

reader = Resources.getResourceAsReader("config/Configure.xml");

//读取配置信息，获取sqlSessionFactory [sql会话工厂]

sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);

} catch (Exception e) {

e.printStackTrace();

}

}

public static SqlSessionFactory getSession() {

return sqlSessionFactory;

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// TODO Auto-generated method stub

//创建一次sql会话，记得用完了要关闭，SqlSession

//这个也可以提到工具类、可以有参数boolean 开启、关闭事务

SqlSession session = sqlSessionFactory.openSession();

try {

//sqlSessionFactory.getConfiguration().addMapper(IUser.class);

//User user = (User) session.selectOne( "com.yiibai.mybatis.models.UserMapper.getUserByID", 1);

// 用户数据列表

getUserList();

// 插入数据

// testInsert();

// 更新用户

//testUpdate();

// 删除数据

//testDelete();

} finally {

session.close();

}

}

//插入数据的方法，通过IUser 的public void insertUser(user);

public static void testInsert()

{

try

{

// 获取Session连接

SqlSession session = sqlSessionFactory.openSession();

// 获取Mapper [映射:就是User实体类对应的接口，User.xml有具体的sql]

IUser userMapper = session.getMapper(IUser.class);

System.out.println("Test insert start...");

// 执行插入

User user = new User();

user.setId(0);

user.setName("Google");

user.setDept("Tech");

user.setWebsite("http://www.google.com");

user.setPhone("120");

userMapper.insertUser(user);

// 提交事务[完成写入数据库]

session.commit();

// 显示插入之后User信息

System.out.println("After insert");

getUserList();

System.out.println("Test insert finished...");

}

catch (Exception e)

{

e.printStackTrace();

}

}

// 获取用户列表

public static void getUserList() {

try {

SqlSession session = sqlSessionFactory.openSession();

IUser iuser = session.getMapper(IUser.class);

// 显示User信息

System.out.println("Test Get start...");

printUsers(iuser.getUserList());

System.out.println("Test Get finished...");

} catch (Exception e) {

e.printStackTrace();

}

}

public static void testUpdate()

{

try

{

SqlSession session = sqlSessionFactory.openSession();

IUser iuser = session.getMapper(IUser.class);

System.out.println("Test update start...");

printUsers(iuser.getUserList());

// 执行更新

User user = iuser.getUser(1);

user.setName("New name");

iuser.updateUser(user);

// 提交事务

session.commit();

// 显示更新之后User信息

System.out.println("After update");

printUsers(iuser.getUserList());

System.out.println("Test update finished...");

}catch (Exception e)

{

e.printStackTrace();

}

}

// 删除用户信息

public static void testDelete()

{

try

{

SqlSession session = sqlSessionFactory.openSession();

IUser iuser = session.getMapper(IUser.class);

System.out.println("Test delete start...");

// 显示删除之前User信息

System.out.println("Before delete");

printUsers(iuser.getUserList());

// 执行删除

iuser.deleteUser(2);

// 提交事务

session.commit();

// 显示删除之后User信息

System.out.println("After delete");

printUsers(iuser.getUserList());

System.out.println("Test delete finished...");

}catch (Exception e)

{

e.printStackTrace();

}

}

/\*\*

\*

\* 打印用户信息到控制台

\*

\* @param users

\*/

private static void printUsers(final List<User> users) {

int count = 0;

for (User user : users) {

System.out.println(MessageFormat.format(

"============= User[{0}]=================", ++count));

System.out.println("User Id: " + user.getId());

System.out.println("User Name: " + user.getName());

System.out.println("User Dept: " + user.getDept());

System.out.println("User Website: " + user.getWebsite());

}

}

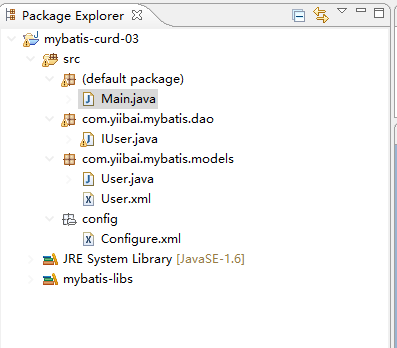
}

Java

### 测试运行

执行以上程序，如果没有问题，应该能正确输出。那么这里所有增删改查都完成了，需要注意的是在增加，更改，删除的时候需要调用 session.commit() 来提交事务，这样才会真正对数据库进行操作提交保存，否则操作没有提交到数据中。  
到此为止，简单的单表操作已经完成了，接下来在下一节中将会讲解多表联合查询，以及结果集的选取。 如遇到不明白的问题，请留言评论。

最后，附上项目结构图，如下：



## Mybatis表关联一对多

有了前面几章的基础，对一些简单的应用是可以处理的，但在实际项目中，经常是关联表的查询，比如：最常见到的多对一，一对多等。这些查询是如何处理的呢，这一讲就讲这个问题。前面几篇教程中介绍的都是单表映射的一些操作，然而在我们的实际项目中往往是用到多表映射。在Java实体对象对中，一对多可以根据List和Set来实现，两者在mybitis中都是通过collection标签来配合来加以实现。这篇介绍的是多表中的多对一表关联查询。

**应用场景：**首先根据用户 ID 读取一个用户信息，然后再读取这个用户所发布贴子(post)。

### 1、先做一些准备工作

我们首先在创建一个 java 工程，工程名称为：mybatis04-one2many([下载](http://pan.baidu.com/s/1pJIeU0b))，还需要创建两张表，它们分别是用户表 user，和帖子表 post，一个户用户可以有多个帖子。  
**user表的结构和数据：**

-- ----------------------------

-- Table structure for `user`

-- ----------------------------

CREATE TABLE `user` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`username` varchar(64) NOT NULL DEFAULT '',

`mobile` int(10) unsigned NOT NULL DEFAULT '0',

`created` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of user

-- ----------------------------

INSERT INTO `user` VALUES ('1', 'yiibai', '100', '2015-09-23 20:11:23');

**帖子表 post 的结构和数据：**

-- ----------------------------

-- Table structure for `post`

-- ----------------------------

CREATE TABLE `post` (

`post\_id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`userid` int(10) unsigned NOT NULL,

`title` varchar(254) NOT NULL DEFAULT '',

`content` text,

`created` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',

PRIMARY KEY (`post\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of post

-- ----------------------------

INSERT INTO `post` VALUES ('1', '1', 'MyBatis关联数据查询', '在实际项目中，经常使用关联表的查询，比如：多对一，一对多等。这些查询是如何处理的呢，这一讲就讲这个问题。我们首先创建一个 post 表，并初始化数据.', '2015-09-23 21:40:17');

INSERT INTO `post` VALUES ('2', '1', 'MyBatis开发环境搭建', '为了方便学习，这里直接建立java 工程，但一般都是开发 Web 项目。', '2015-09-23 21:42:14');

INSERT INTO `post` VALUES ('3', '2', '这个是别人发的', 'content,内容...', '0000-00-00 00:00:00');

从上面应该看出，这几个帖子对应的 userid 都是1，所以需要用户表 user 里面有 id=1 的数据。可以修改成满足自己条件的数据，按照orm的规则，表肯定需要一个对象与之对应，所以我们增加一个 Post 类。

### 2、创建表对应的 JavaBean 对象

这个例子中，我们需要在包 com.yiibai.pojo 下创建两个类，它们分别是： User.java 和 Post.java，我们一个一个地来看它们的代码，User.java 类的代码如下：

package com.yiibai.pojo;

import java.io.Serializable;

import java.util.Date;

import java.util.List;

public class User implements Serializable{

private int id;

private String username;

private String mobile;

private List<Post> posts;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getMobile() {

return mobile;

}

public void setMobile(String mobile) {

this.mobile = mobile;

}

public List<Post> getPosts() {

return posts;

}

public void setPosts(List<Post> posts) {

this.posts = posts;

}

@Override

public String toString() {

return "User [id=" + id + ", name=" + username + "]";

}

}

Post.java 类的代码如下：

package com.yiibai.pojo;

import java.io.Serializable;

public class Post implements Serializable{

private int id;

private User user;

private String title;

private String content;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getContent() {

return content;

}

public void setContent(String content) {

this.content = content;

}

}

### 3、配置文件

在这一章节中，要用到的配置文件有两个，一个是 mybatis 的主配置文件：src/config/Configure.xml 和 User.java对应的配置文件 User.xml，我们先来看看 src/config/Configure.xml，其详细配置信息如下：

<?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>

<typeAliases>

<typeAlias alias="User" type="com.yiibai.pojo.User" />

<typeAlias alias="Post" type="com.yiibai.pojo.Post" />

</typeAliases>

<environments default="development">

<environment id="development">

<transactionManager type="JDBC" />

<dataSource type="POOLED">

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

<property name="url" value="jdbc:mysql://127.0.0.1:3306/yiibai" />

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

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

</dataSource>

</environment>

</environments>

<mappers>

<!-- // power by http://www.yiibai.com -->

<mapper resource="com/yiibai/pojo/User.xml" />

</mappers>

</configuration>

这里需要注意的是 <typeAliases> 这个标签内容，它就是用于定义一个 JavaBean 类的别名，如将 com.yiibai.pojo.User 简写为 User，可以认为  com.yiibai.pojo.User 就是 User，User 就是  com.yiibai.pojo.User 。

另外一个配置文件 User.xml 的内容如下：

<?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.yiibai.userMaper">

<!-- User 级联文章查询 方法配置 (一个用户对多个文章) -->

<!--<typeAliases>标签为 javaBean实体类指定的别名： User -->

<resultMap type="User" id="resultUserMap">

<result property="id" column="user\_id" />

<result property="username" column="username" />

<result property="mobile" column="mobile" />

<collection property="posts" ofType="com.yiibai.pojo.Post" column="userid">

<id property="id" column="post\_id" javaType="int" jdbcType="INTEGER"/>

<result property="title" column="title" javaType="string" jdbcType="VARCHAR"/>

<result property="content" column="content" javaType="string" jdbcType="VARCHAR"/>

</collection>

</resultMap>

<select id="getUser" resultMap="resultUserMap" parameterType="int">

SELECT u.\*,p.\*

FROM user u, post p

WHERE u.id=p.userid AND id=#{user\_id}

</select>

</mapper>

### 4、测试程序运行

到这里，整个工作准备得已经差不多了，我们创建一个主类来测试上面程序，在 src 下创建一个 Main.java，代码如下：

import java.io.Reader;

import java.text.MessageFormat;

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.yiibai.pojo.Post;

import com.yiibai.pojo.User;

public class Main {

private static SqlSessionFactory sqlSessionFactory;

private static Reader reader;

static {

try {

reader = Resources.getResourceAsReader("config/Configure.xml");

sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);

} catch (Exception e) {

e.printStackTrace();

}

}

public static SqlSessionFactory getSession() {

return sqlSessionFactory;

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// TODO Auto-generated method stub

SqlSession session = sqlSessionFactory.openSession();

try {

int userid = 1;

User user = session.selectOne("com.yiibai.userMaper.getUser", 1);

System.out.println("username: "+user.getUsername()+",");

List<Post> posts = user.getPosts();

for(Post p : posts) {

System.out.println("Title:" + p.getTitle());

System.out.println("Content:" + p.getContent());

}

} finally {

session.close();

}

}

}

输出结果如下：

username: yiibai,

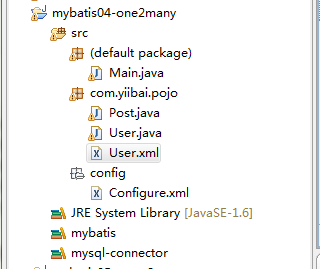
Title:MyBatis关联数据查询

Content:在实际项目中，经常使用关联表的查询，比如：多对一，一对多等。这些查询是如何处理的呢，这一讲就讲这个问题。我们首先创建一个 post 表，并初始化数据.

Title:MyBatis开发环境搭建

Content:为了方便学习，这里直接建立java 工程，但一般都是开发 Web 项目。

**附工程目录结构图如下：**



**易百教程移动端**：请扫描本页面底部(右侧)二维码并关注微信公众号，回复："**教程**" 选择相关教程阅读或直接访问：http://m.yiibai.com 。

## Mybatis表关联多对一

在上章的 一对多 中，我们已经学习如何在 Mybatis 中关联多表，但在实际项目中也是经常使用 多对一 的情况，这些查询是如何处理的呢，在这一节中我们来学习它。多表映射的多对一关系要用到 mybitis 的 association 来加以实现。这篇介绍的是多表中的多对一表关联查询。

**应用场景：**首先根据帖子 ID 读取一个帖子信息，然后再读取这个帖子所属的用户信息。

### 1、先做一些准备工作

我们首先在创建一个 java 工程，工程名称为：mybatis05-many2one(下载)，还需要创建两张表，它们分别是用户表 user，和帖子表 post，一个户用户可以有多个帖子。

**user表的结构和数据：**

-- ----------------------------

-- Table structure for `user`

-- ----------------------------

CREATE TABLE `user` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`username` varchar(64) NOT NULL DEFAULT '',

`mobile` int(10) unsigned NOT NULL DEFAULT '0',

`created` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of user

-- ----------------------------

INSERT INTO `user` VALUES ('1', 'yiibai', '100', '2015-09-23 20:11:23');

**帖子表 post 的结构和数据：**

-- ----------------------------

-- Table structure for `post`

-- ----------------------------

CREATE TABLE `post` (

`post\_id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`userid` int(10) unsigned NOT NULL,

`title` varchar(254) NOT NULL DEFAULT '',

`content` text,

`created` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',

PRIMARY KEY (`post\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of post

-- ----------------------------

INSERT INTO `post` VALUES ('1', '1', 'MyBatis关联数据查询', '在实际项目中，经常使用关联表的查询，比如：多对一，一对多等。这些查询是如何处理的呢，这一讲就讲这个问题。我们首先创建一个 post 表，并初始化数据.', '2015-09-23 21:40:17');

INSERT INTO `post` VALUES ('2', '1', 'MyBatis开发环境搭建', '为了方便学习，这里直接建立java 工程，但一般都是开发 Web 项目。', '2015-09-23 21:42:14');

INSERT INTO `post` VALUES ('3', '2', '这个是别人发的', 'content,内容...', '0000-00-00 00:00:00');

从上面应该看出，这几个帖子对应的 userid 都是1，所以需要用户表 user 里面有 id=1 的数据。可以修改成满足自己条件的数据，按照 orm 的规则，表肯定需要一个对象与之对应，所以我们增加一个 Post 类。

### 2、创建表对应的 JavaBean 对象

这个例子中，我们需要在包 com.yiibai.pojo 下创建两个类，它们分别是： User.java 和 Post.java，我们一个一个地来看它们的代码，User.java 类的代码如下：

package com.yiibai.pojo;

import java.io.Serializable;

import java.util.Date;

import java.util.List;

public class User implements Serializable{

private int id;

private String username;

private String mobile;

private List<Post> posts;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getMobile() {

return mobile;

}

public void setMobile(String mobile) {

this.mobile = mobile;

}

public List<Post> getPosts() {

return posts;

}

public void setPosts(List<Post> posts) {

this.posts = posts;

}

@Override

public String toString() {

return "User [id=" + id + ", name=" + username + "]";

}

}

Post.java 类的代码如下：

package com.yiibai.pojo;

import java.io.Serializable;

public class Post implements Serializable{

private int id;

private User user;

private String title;

private String content;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getContent() {

return content;

}

public void setContent(String content) {

this.content = content;

}

}

### 3、配置文件

在这一章节中，要用到的配置文件有两个，一个是 mybatis 的主配置文件：src/config/Configure.xml 和 User.java对应的配置文件 User.xml，我们先来看看 src/config/Configure.xml，其详细配置信息如下：

<?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>

<typeAliases>

<typeAlias alias="User" type="com.yiibai.pojo.User" />

<typeAlias alias="Post" type="com.yiibai.pojo.Post" />

</typeAliases>

<environments default="development">

<environment id="development">

<transactionManager type="JDBC" />

<dataSource type="POOLED">

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

<property name="url" value="jdbc:mysql://127.0.0.1:3306/yiibai" />

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

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

</dataSource>

</environment>

</environments>

<mappers>

<!-- // power by http://www.yiibai.com -->

<mapper resource="com/yiibai/pojo/User.xml" />

</mappers>

</configuration>

这里需要注意的是 <typeAliases> 这个标签内容，它就是用于定义一个 JavaBean 类的别名，如将 com.yiibai.pojo.User 简写为 User，可以认为  com.yiibai.pojo.User 就是 User，User 就是  com.yiibai.pojo.User 。

另外一个配置文件 User.xml 的内容如下：

<?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.yiibai.userMaper">

<!-- User 级联文章查询 方法配置 (多个文章对一个用户) -->

<resultMap type="Post" id="resultPostsMap">

<result property="id" column="post\_id" />

<result property="title" column="title" />

<result property="content" column="content" />

<association property="user" javaType="User">

<id property="id" column="userid"/>

<result property="username" column="username"/>

<result property="mobile" column="mobile"/>

</association>

</resultMap>

<select id="getPosts" resultMap="resultPostsMap" parameterType="int">

SELECT u.\*,p.\*

FROM user u, post p

WHERE u.id=p.userid AND p.post\_id=#{post\_id}

</select>

</mapper>

**注：**在上面的配置文件中，使用到了一个 <association>标签，关联对应的 User 类。

### 4、测试程序运行

到这里，整个工作准备得已经差不多了，我们创建一个主类来测试上面程序，在 src 下创建一个 Main.java，代码如下：

import java.io.Reader;

import java.text.MessageFormat;

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.yiibai.pojo.Post;

import com.yiibai.pojo.User;

public class Main {

private static SqlSessionFactory sqlSessionFactory;

private static Reader reader;

static {

try {

reader = Resources.getResourceAsReader("config/Configure.xml");

sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);

} catch (Exception e) {

e.printStackTrace();

}

}

public static SqlSessionFactory getSession() {

return sqlSessionFactory;

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// TODO Auto-generated method stub

SqlSession session = sqlSessionFactory.openSession();

try {

int postId = 1;

Post post = session.selectOne("com.yiibai.userMaper.getPosts", postId);

System.out.println("title: "+post.getTitle());

System.out.println("userName: "+post.getUser().getUsername());

} finally {

session.close();

}

}

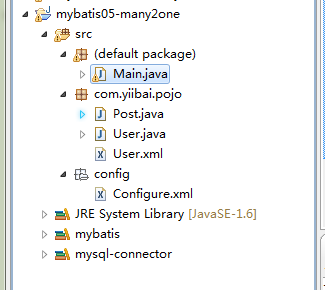
}

输出结果如下：

title: MyBatis关联数据查询

userName: yiibai

**附工程目录结构图如下：**

****

## Mybatis表关联多对多

在前面的章节中，我们学习了一对多，多对一的关系，现在我们来看看 Mybatis 中的多对多应用。

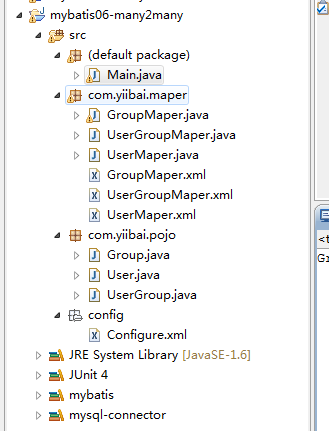
mybatis3.0 添加了association和collection标签专门用于对多个相关实体类数据进行级联查询，但仍不支持多个相关实体类数据的级联保存和级联删除操作。因此在进行实体类多对多映射表设计时，需要专门建立一个关联对象类对相关实体类的关联关系进行描述。下文将以“User”和“Group"两个实体类之间的多对多关联映射为例进行CRUD操作。

### 1、应用场景

假设项目中存在用户和用户组，从一个用户读取出它所在的用户组，从一个用户组也知道这个组内的所有用户信息。

### 2、先做一些准备工作

我们首先在创建一个 java 工程，工程名称为：mybatis06-many2many(下载)，还需要创建三张表，它们分别是用户表 user，用户组表 group 和 用户组映射表 user\_group ，一个户用户可以在多个用户组中，一个用户组中有多个用户。项目工程结构如下：



**user表的结构和数据：**

CREATE TABLE `user` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`username` varchar(64) NOT NULL DEFAULT '',

`mobile` varchar(16) NOT NULL DEFAULT '',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of user

-- ----------------------------

INSERT INTO `user` VALUES ('1', 'yiibai', '13838009988');

INSERT INTO `user` VALUES ('2', 'User-name-1', '13838009988');

**用户组 group 表的结构和数据：**

CREATE TABLE `group` (

`group\_id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`group\_name` varchar(254) NOT NULL DEFAULT '',

PRIMARY KEY (`group\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of group

-- ----------------------------

INSERT INTO `group` VALUES ('1', 'Group-1');

INSERT INTO `group` VALUES ('2', 'Group-2');

**用户组映射表 user\_group 的结构和数据：**

CREATE TABLE `user\_group` (

`user\_id` int(10) unsigned NOT NULL DEFAULT '0',

`group\_id` int(10) unsigned NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of user\_group

-- ----------------------------

INSERT INTO `user\_group` VALUES ('1', '1');

INSERT INTO `user\_group` VALUES ('2', '1');

INSERT INTO `user\_group` VALUES ('1', '2');

从上面应该看出，用户ID为1同时在用户组ID为 1 和 2 中，而用户ID为 2 仅在一个用户组ID为1中。

### 2、创建表对应的 JavaBean 对象

这个例子中，我们需要在包 com.yiibai.pojo 下创建三个类，它们分别是： User.java 、Group.java 和 UserGroup.java，让我们一个一个地来看它们的代码，User.java 类的代码如下：

package com.yiibai.pojo;

import java.util.List;

/\*\*

\* @describe: User

\* @author: Yiibai

\* @version: V1.0

\* @copyright http://www.yiibai.com

\*/

public class User {

private int id;

private String username;

private String mobile;

private List<Group> groups;

public List<Group> getGroups() {

return groups;

}

public void setGroups(List<Group> groups) {

this.groups = groups;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getMobile() {

return mobile;

}

public void setMobile(String mobile) {

this.mobile = mobile;

}

}

Group.java 类的代码如下：

package com.yiibai.pojo;

import java.util.List;

/\*\*

\* @describe: Group

\* @author: Yiibai

\* @version: V1.0

\* @copyright http://www.yiibai.com

\*/

public class Group {

private int groupId;

private String groupName;

private List<User> users;

public List<User> getUsers() {

return users;

}

public void setUsers(List<User> users) {

this.users = users;

}

public int getGroupId() {

return groupId;

}

public void setGroupId(int groupId) {

this.groupId = groupId;

}

public String getGroupName() {

return groupName;

}

public void setGroupName(String groupName) {

this.groupName = groupName;

}

}

UserGroup.java 类(用户和用户组的关系映射)的代码如下：

package com.yiibai.pojo;

public class UserGroup {

private int userId;

private int groupId;

public int getUserId() {

return userId;

}

public void setUserId(int userId) {

this.userId = userId;

}

public int getGroupId() {

return groupId;

}

public void setGroupId(int groupId) {

this.groupId = groupId;

}

}

### 3、配置文件

在这一章节中，要用到的配置文件有四个，一个是 mybatis 的主配置文件：src/config/Configure.xml ，另外就是上面三个Bean类对应的配置文件，如，User.java 对应的配置文件 User.xml，等，我们先来看看 src/config/Configure.xml，其详细配置信息如下：

<?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>

<typeAliases>

<typeAlias alias="User" type="com.yiibai.pojo.User" />

<typeAlias alias="UserGroup" type="com.yiibai.pojo.UserGroup" />

<typeAlias alias="Group" type="com.yiibai.pojo.Group" />

</typeAliases>

<environments default="development">

<environment id="development">

<transactionManager type="JDBC" />

<dataSource type="POOLED">

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

<property name="url" value="jdbc:mysql://127.0.0.1:3306/yiibai" />

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

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

</dataSource>

</environment>

</environments>

<mappers>

<!-- // power by http://www.yiibai.com -->

<mapper resource="com/yiibai/maper/UserMaper.xml" />

<mapper resource="com/yiibai/maper/GroupMaper.xml" />

<mapper resource="com/yiibai/maper/UserGroupMaper.xml" />

</mappers>

</configuration>

Group.java 对应的配置文件 src/com/yiibai/maper/Group.xml 的内容如下：

<?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.yiibai.maper.GroupMaper">

<parameterMap type="Group" id="parameterGroupMap">

<parameter property="groupId"/>

<parameter property="groupName"/>

</parameterMap>

<insert id="insertGroup" parameterMap="parameterGroupMap">

INSERT INTO `group` (group\_name)

VALUES(#{groupName});

</insert>

<resultMap type="Group" id="resultGroupMap\_1">

<result property="id" column="id" />

<result property="groupName" column="group\_name" />

<collection property="users" column="group\_id"

select="com.yiibai.maper.UserGroupMaper.getUsersByGroupId" />

</resultMap>

<select id="getGroup" resultMap="resultGroupMap\_1"

parameterType="int">

SELECT \*

FROM `group`

WHERE group\_id=#{id}

</select>

</mapper>

User.java 对应的配置文件 src/com/yiibai/maper/User.xml 的内容如下：

<?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.yiibai.maper.UserMaper">

<parameterMap type="User" id="parameterUserMap">

<parameter property="id"/>

<parameter property="username"/>

<parameter property="mobile"/>

</parameterMap>

<insert id="insertUser" parameterMap="parameterUserMap">

INSERT INTO user(username,mobile)

VALUES(#{username},#{mobile});

</insert>

<resultMap type="User" id="resultUser">

<result property="id" column="group\_id"/>

<result property="name" column="name"/>

<collection property="groups" column="id" select="com.yiibai.maper.UserGroupMaper.getGroupsByUserId"/>

</resultMap>

<select id="getUser" resultMap="resultUser" parameterType="int">

SELECT \*

FROM user

WHERE id=#{id}

</select>

</mapper>

UserGroup.java 对应的配置文件 src/com/yiibai/maper/UserGroup.xml 的内容如下：

<?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.yiibai.maper.UserGroupMaper">

<parameterMap type="UserGroup" id="parameterUserGroupMap">

<parameter property="userId"/>

<parameter property="groupId"/>

</parameterMap>

<insert id="insertUserGroup" parameterMap="parameterUserGroupMap">

INSERT INTO user\_group(user\_id, group\_id)

VALUES(#{userId},#{groupId})

</insert>

<!-- 根据一个用户组ID,查看这个用户组下的所有用户 -->

<resultMap type="User" id="resultUserMap\_2">

<result property="id" column="id"/>

<result property="username" column="username"/>

<result property="mobile" column="mobile"/>

</resultMap>

<select id="getUsersByGroupId" resultMap="resultUserMap\_2" parameterType="int">

SELECT u.\*, ug.group\_id

FROM user u, user\_group ug

WHERE u.id=ug.user\_id AND ug.group\_id=#{group\_id}

</select>

<!-- 根据一个用户ID,查看这个用户所对应的组-->

<resultMap type="Group" id="resultGroupMap\_2">

<result property="groupId" column="group\_id"/>

<result property="groupName" column="group\_name"/>

</resultMap>

<select id="getGroupsByUserId" resultMap="resultGroupMap\_2" parameterType="int">

SELECT g.\*, u.user\_id

FROM group g, user\_group u

WHERE g.group\_id=u.group\_id AND u.user\_id=#{user\_id}

</select>

</mapper>

**注：**在上面的配置文件中，使用到了 <association>和  <clollection>标签，关联对应的 User 类和 Group类。

### 4、测试程序运行

到这里，整个工作准备得已经差不多了，我们创建一个主类来测试上面程序，在 src 下创建一个 Main.java，代码如下：

import java.io.Reader;

import java.text.MessageFormat;

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.yiibai.maper.GroupMaper;

import com.yiibai.maper.UserGroupMaper;

import com.yiibai.maper.UserMaper;

import com.yiibai.pojo.Group;

import com.yiibai.pojo.User;

import com.yiibai.pojo.UserGroup;

public class Main {

private static SqlSessionFactory sqlSessionFactory;

private static Reader reader;

static {

try {

reader = Resources.getResourceAsReader("config/Configure.xml");

sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);

} catch (Exception e) {

e.printStackTrace();

}

}

public static SqlSessionFactory getSession() {

return sqlSessionFactory;

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// TODO Auto-generated method stub

// testAddGroup();

// testAddUser();

// testAddUserGroup();

testGetGroupAndUsers();

}

public static void testGetGroupAndUsers() {

UserGroup userGroup = new UserGroup();

SqlSession session = sqlSessionFactory.openSession();

try {

GroupMaper groupMaper = session.getMapper(GroupMaper.class);

Group group = groupMaper.getGroup(1);

System.out.println("Group => " + group.getGroupName());

List<User> users = group.getUsers();

for (User user : users) {

System.out.println("\t:" + user.getId() + "\t"

+ user.getUsername());

}

} finally {

session.close();

}

}

public static void testAddUserGroup() {

UserGroup userGroup = new UserGroup();

userGroup.setGroupId(1);

userGroup.setUserId(2);

SqlSession session = sqlSessionFactory.openSession();

try {

UserGroupMaper userGroupMaper = session

.getMapper(UserGroupMaper.class);

userGroupMaper.insertUserGroup(userGroup);

session.commit();

} finally {

session.close();

}

}

public static void testAddUser() {

// TODO Auto-generated method stub

SqlSession session = sqlSessionFactory.openSession();

try {

User user = new User();

user.setUsername("User-name-1");

user.setMobile("13838009988");

UserMaper userMaper = session.getMapper(UserMaper.class);

userMaper.insertUser(user);

session.commit();

// System.out.println(user.getGroupId());

} finally {

session.close();

}

}

public static void testAddGroup() {

// TODO Auto-generated method stub

SqlSession session = sqlSessionFactory.openSession();

try {

Group group = new Group();

group.setGroupName("用户组-1");

GroupMaper groupMapper = session.getMapper(GroupMaper.class);

groupMapper.insertGroup(group);

session.commit();

System.out.println(group.getGroupId());

} finally {

session.close();

}

}

}

运行上述程序，得出结果：

Group => Group-1

:1 yiibai

:2 User-name-1

## [Mybatis与Spring集成](http://pan.baidu.com/s/1wRpOq)

在前面的教程文章中，前面讲到有关 mybatis 连接数据库，然后进行进行数据增删改查，以及多表联合查询的例子，但很多的项目中，通常会用 spring 这个粘合剂来管理 datasource 等。充分利用 spring 基于接口的编程，以及aop ,ioc 带来的方便。用 spring 来管理 [mybatis](http://www.yiibai.com/mybatis/) 与管理 hibernate 有很多类似的地方。在这一节中，我们重点介绍数据源管理以及 bean 的配置。

整个Mybatis与Spring集成示例要完成的步骤如下：

1、示例功能描述

2、创建工程

3、数据库表结构及数据记录

4、实例对象

5、配置文件

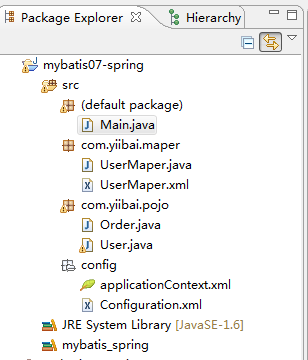
6、测试执行，输出结果

### 1、示例功能描述

在本示例中，需要完成这样的一个简单功能，即，指定一个用户（ID=1），查询出这个用户的基本信息，并关联查询这个用户的所有订单。

### 2、创建工程

 首先创建一个工程的名称为：mybatis07-spring，在 src 源代码目录下建立文件夹 config，并将原来的 mybatis 配置文件 Configuration.xml 移动到这个文件夹中, 并在 config 文家夹中建立 Spring 配置文件：applicationContext.xml。工程结构目录如下：



### 3、数据库表结构及数据记录

在本示例中，用到两个表：用户表和订单表，其结构和数据记录如下：

CREATE TABLE `user` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`username` varchar(64) NOT NULL DEFAULT '',

`mobile` varchar(16) NOT NULL DEFAULT '',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of user

-- ----------------------------

INSERT INTO `user` VALUES ('1', 'yiibai', '13838009988');

INSERT INTO `user` VALUES ('2', 'saya', '13838009988');

订单表结构和数据如下：

CREATE TABLE `order` (

`order\_id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`user\_id` int(10) unsigned NOT NULL DEFAULT '0',

`order\_no` varchar(16) NOT NULL DEFAULT '',

`money` float(10,2) unsigned DEFAULT '0.00',

PRIMARY KEY (`order\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=17 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of order

-- ----------------------------

INSERT INTO `order` VALUES ('1', '1', '1509289090', '99.90');

INSERT INTO `order` VALUES ('2', '1', '1519289091', '290.80');

INSERT INTO `order` VALUES ('3', '1', '1509294321', '919.90');

INSERT INTO `order` VALUES ('4', '1', '1601232190', '329.90');

INSERT INTO `order` VALUES ('5', '1', '1503457384', '321.00');

INSERT INTO `order` VALUES ('6', '1', '1598572382', '342.00');

INSERT INTO `order` VALUES ('7', '1', '1500845727', '458.00');

INSERT INTO `order` VALUES ('8', '1', '1508458923', '1200.00');

INSERT INTO `order` VALUES ('9', '1', '1504538293', '2109.00');

INSERT INTO `order` VALUES ('10', '1', '1932428723', '5888.00');

INSERT INTO `order` VALUES ('11', '1', '2390423712', '3219.00');

INSERT INTO `order` VALUES ('12', '1', '4587923992', '123.00');

INSERT INTO `order` VALUES ('13', '1', '4095378812', '421.00');

INSERT INTO `order` VALUES ('14', '1', '9423890127', '678.00');

INSERT INTO `order` VALUES ('15', '1', '7859213249', '7689.00');

INSERT INTO `order` VALUES ('16', '1', '4598450230', '909.20');

### 4、实例对象

用户表和订单表分别对应两个实例对象，分别是：User.java 和 Order.java，它们都在 com.yiibai.pojo 包中。

User.java代码内容如下：

package com.yiibai.pojo;

import java.util.List;

/\*\*

\* @describe: User

\* @author: Yiibai

\* @version: V1.0

\* @copyright http://www.yiibai.com

\*/

public class User {

private int id;

private String username;

private String mobile;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getMobile() {

return mobile;

}

public void setMobile(String mobile) {

this.mobile = mobile;

}

}

Order.java代码内容如下：

package com.yiibai.pojo;

/\*\*

\* @describe: Order - 订单

\* @author: Yiibai

\* @version: V1.0

\* @copyright http://www.yiibai.com

\*/

public class Order {

private int orderId;

private String orderNo;

private float money;

private int userId;

private User user;

public int getUserId() {

return userId;

}

public void setUserId(int userId) {

this.userId = userId;

}

public int getOrderId() {

return orderId;

}

public void setOrderId(int orderId) {

this.orderId = orderId;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public String getOrderNo() {

return orderNo;

}

public void setOrderNo(String orderNo) {

this.orderNo = orderNo;

}

public float getMoney() {

return money;

}

public void setMoney(float money) {

this.money = money;

}

}

### 5、配置文件

这个实例中有三个重要的配置文件，它们分别是：applicationContext.xml ， Configuration.xml 以及 UserMaper.xml。

applicationContext.xml  配置文件里最主要的配置：

<?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>

<typeAliases>

<typeAlias alias="User" type="com.yiibai.pojo.User" />

<typeAlias alias="Order" type="com.yiibai.pojo.Order" />

</typeAliases>

<!-- Mybatis和Spring 集成之后,这些可以完全删除（注释掉）,数据库连接的管理交给 Spring 来管理 -->

<!--

<environments default="development"> <environment id="development">

<transactionManager type="JDBC"/> <dataSource type="POOLED"> <property

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

value="jdbc:mysql://127.0.0.1:3306/yiibai?characterEncoding=utf8" />

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

value=""/> </dataSource> </environment> </environments>

-->

<mappers>

<mapper resource="com/yiibai/maper/UserMaper.xml" />

</mappers>

</configuration>

配置文件 Configuration.xml 的内容如下：

<?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:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx" 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/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd

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

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

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

default-autowire="byName" default-lazy-init="false">

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

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

<property name="url"

value="jdbc:mysql://127.0.0.1:3306/yiibai?characterEncoding=utf8" />

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

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

</bean>

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

<!--dataSource属性指定要用到的连接池-->

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

<!--configLocation属性指定mybatis的核心配置文件-->

<property name="configLocation" value="config/Configuration.xml" />

</bean>

<bean id="userMaper" class="org.mybatis.spring.mapper.MapperFactoryBean">

<!--sqlSessionFactory属性指定要用到的SqlSessionFactory实例-->

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

<!--mapperInterface属性指定映射器接口，用于实现此接口并生成映射器对象-->

<property name="mapperInterface" value="com.yiibai.maper.UserMaper" />

</bean>

</beans>

UserMaper.xml 用于定义查询和数据对象映射，其内容如下：

<?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.yiibai.maper.UserMaper">

<!-- 为了返回list 类型而定义的returnMap -->

<resultMap type="User" id="resultUser">

<id column="id" property="id" />

<result column="username" property="username" />

<result column="mobile" property="mobile" />

</resultMap>

<!-- User 联合 Order 查询 方法的配置 (多对一的方式) -->

<resultMap id="resultUserOrders" type="Order">

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

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

<result property="money" column="money" />

<result property="userId" column="user\_id" />

<association property="user" javaType="User">

<id property="id" column="id" />

<result property="username" column="username" />

<result property="mobile" column="mobile" />

</association>

</resultMap>

<select id="getUserOrders" parameterType="int" resultMap="resultUserOrders">

SELECT u.\*,o.\* FROM `user` u, `order` o

WHERE u.id=o.user\_id AND u.id=#{id}

</select>

<select id="getUserById" resultMap="resultUser" parameterType="int">

SELECT \*

FROM user

WHERE id=#{id}

</select>

</mapper>

### 6、测试执行，输出结果

我们创建一个测试类为：Main.java , 就在 src 目录中。其代码如下：

import java.util.List;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.yiibai.maper.UserMaper;

import com.yiibai.pojo.Order;

import com.yiibai.pojo.User;

/\*\*

\* Description

\* @author yiibai

\* @date 2015-4-12

\* @copyright http://www.yiibai.com

\* @email yiibai.com@gmai.com

\* @version 1.0

\*/

public class Main {

private static ApplicationContext ctx;

static {

ctx = new ClassPathXmlApplicationContext(

"config/applicationContext.xml");

}

public static void main(String[] args) {

UserMaper userMaper = (UserMaper) ctx.getBean("userMaper");

// 测试id=1的用户查询，可根据数据库中的情况修改.

User user = userMaper.getUserById(1);

System.out.println("获取用户 ID=1 的用户名："+user.getUsername());

// 得到文章列表测试

System.out.println("得到用户id为1的所有订单列表:");

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

List<Order> orders = userMaper.getUserOrders(1);

for (Order order : orders) {

System.out.println("订单号："+order.getOrderNo() + "，订单金额：" + order.getMoney());

}

}

}

运行结果如下：

log4j:WARN No appenders could be found for logger (org.springframework.context.support.ClassPathXmlApplicationContext).

log4j:WARN Please initialize the log4j system properly.

获取用户 ID=1 的用户名：yiibai

得到用户id为1的所有订单列表:

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

订单号：1509289090，订单金额：99.9

订单号：1519289091，订单金额：290.8

订单号：1509294321，订单金额：919.9

订单号：1601232190，订单金额：329.9

订单号：1503457384，订单金额：321.0

订单号：1598572382，订单金额：342.0

订单号：1500845727，订单金额：458.0

订单号：1508458923，订单金额：1200.0

订单号：1504538293，订单金额：2109.0

订单号：1932428723，订单金额：5888.0

订单号：2390423712，订单金额：3219.0

订单号：4587923992，订单金额：123.0

订单号：4095378812，订单金额：421.0

订单号：9423890127，订单金额：678.0

订单号：7859213249，订单金额：7689.0

订单号：4598450230，订单金额：909.2

代码下载：[Mybatis与Spring集成](http://pan.baidu.com/s/1wRpOq) https://pan.baidu.com/s/1wRpOq

## MyBatis整合Spring MVC

前面几篇文章已经讲到了mybatis与spring 的集成。目前主流的Web MVC框架，除了Struts这个主力外，还有Spring MVC，主要是由于 Spring MVC 配置比较简单，使用起来也十分明了，非常灵活，与Spring 集成较好，对 RESTful API 的支持也比 struts 要好。所以Spring MVC在一定程度上有一定的优势。  
MyBatis是ibatis的升级版，作为hibernate的老对手，它是一个可以自定义SQL、存储过程和高级映射的持久层框架。  
与Hibernate 的主要区别就是 Mybatis 是半自动化的，而 Hibernate 是全自动的，所以当应用需求越来越复杂的时候，自动化的 SQL 显得比较笨拙。  
经常搭框架的人应该都清楚，框架搭建的核心就是配置文件。

在这里我们需要创建 web 工程。今天将直接用 mybatis与Spring mvc 的方式集成起来，源码在本文结尾处下载.主要有以下几个方面的配置。

**整个Mybatis与Spring MVC 示例要完成的步骤如下：**

1、示例功能描述

2、创建工程

3、数据库表结构及数据记录

4、实例对象

5、配置文件

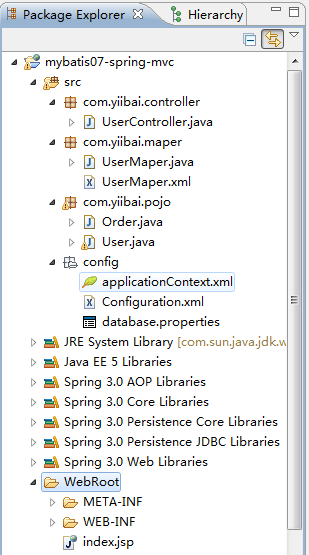
6、测试执行，输出结果

### 1、示例功能描述

在本示例中，需要使用 MyBatis和Spring MVC整合完成这样的一个简单功能，即指定一个用户（ID=1），查询出这个用户关联的所有订单。

### 2、创建工程

首先创建一个工程的名称为：mybatis07-spring-mvc，在 src 源代码目录下建立文件夹 config，并将原来的 mybatis 配置文件 Configuration.xml 移动到这个文件夹中, 并在 config 文家夹中建立 Spring 配置文件：applicationContext.xml。工程结构目录如下：



### 3、数据库表结构及数据记录

在本示例中，用到两个表：用户表和订单表，其结构和数据记录如下：

CREATE TABLE `user` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`username` varchar(64) NOT NULL DEFAULT '',

`mobile` varchar(16) NOT NULL DEFAULT '',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of user

-- ----------------------------

INSERT INTO `user` VALUES ('1', 'yiibai', '13838009988');

INSERT INTO `user` VALUES ('2', 'saya', '13838009988');

订单表结构和数据如下：

CREATE TABLE `order` (

`order\_id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`user\_id` int(10) unsigned NOT NULL DEFAULT '0',

`order\_no` varchar(16) NOT NULL DEFAULT '',

`money` float(10,2) unsigned DEFAULT '0.00',

PRIMARY KEY (`order\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=17 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of order

-- ----------------------------

INSERT INTO `order` VALUES ('1', '1', '1509289090', '99.90');

INSERT INTO `order` VALUES ('2', '1', '1519289091', '290.80');

INSERT INTO `order` VALUES ('3', '1', '1509294321', '919.90');

INSERT INTO `order` VALUES ('4', '1', '1601232190', '329.90');

INSERT INTO `order` VALUES ('5', '1', '1503457384', '321.00');

INSERT INTO `order` VALUES ('6', '1', '1598572382', '342.00');

INSERT INTO `order` VALUES ('7', '1', '1500845727', '458.00');

INSERT INTO `order` VALUES ('8', '1', '1508458923', '1200.00');

INSERT INTO `order` VALUES ('9', '1', '1504538293', '2109.00');

INSERT INTO `order` VALUES ('10', '1', '1932428723', '5888.00');

INSERT INTO `order` VALUES ('11', '1', '2390423712', '3219.00');

INSERT INTO `order` VALUES ('12', '1', '4587923992', '123.00');

INSERT INTO `order` VALUES ('13', '1', '4095378812', '421.00');

INSERT INTO `order` VALUES ('14', '1', '9423890127', '678.00');

INSERT INTO `order` VALUES ('15', '1', '7859213249', '7689.00');

INSERT INTO `order` VALUES ('16', '1', '4598450230', '909.20');

### 4、实例对象

用户表和订单表分别对应两个实例对象，分别是：User.java 和 Order.java，它们都在 com.yiibai.pojo 包中。

User.java代码内容如下：

package com.yiibai.pojo;

import java.util.List;

/\*\*

\* @describe: User

\* @author: Yiibai

\* @version: V1.0

\* @copyright http://www.yiibai.com

\*/

public class User {

private int id;

private String username;

private String mobile;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getMobile() {

return mobile;

}

public void setMobile(String mobile) {

this.mobile = mobile;

}

}

Order.java代码内容如下：

package com.yiibai.pojo;

/\*\*

\* @describe: User - 订单

\* @author: Yiibai

\* @version: V1.0

\* @copyright http://www.yiibai.com

\*/

public class Order {

private int orderId;

private String orderNo;

private float money;

private int userId;

private User user;

public int getUserId() {

return userId;

}

public void setUserId(int userId) {

this.userId = userId;

}

public int getOrderId() {

return orderId;

}

public void setOrderId(int orderId) {

this.orderId = orderId;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public String getOrderNo() {

return orderNo;

}

public void setOrderNo(String orderNo) {

this.orderNo = orderNo;

}

public float getMoney() {

return money;

}

public void setMoney(float money) {

this.money = money;

}

}

### 5、配置文件

这个实例中有三个重要的配置文件，它们分别是：applicationContext.xml ， Configuration.xml 以及 UserMaper.xml。

applicationContext.xml  配置文件里最主要的配置：

<?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:aop="http://www.springframework.org/schema/aop"

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

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

xsi:schemaLocation="

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

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

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

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

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

default-autowire="byName" default-lazy-init="false">

<!--本示例采用DBCP连接池，应预先把DBCP的jar包复制到工程的lib目录下。 -->

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

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

destroy-method="close" p:driverClassName="com.mysql.jdbc.Driver"

p:url="jdbc:mysql://127.0.0.1:3306/yiibai?characterEncoding=utf8"

p:username="root" p:password="" p:maxActive="10" p:maxIdle="10">

</bean>

<bean id="transactionManager"

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

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

</bean>

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

<!--dataSource属性指定要用到的连接池-->

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

<!--configLocation属性指定mybatis的核心配置文件-->

<property name="configLocation" value="classpath:config/Configuration.xml" />

<!-- 所有配置的mapper文件 -->

<property name="mapperLocations" value="classpath\*:com/yiibai/mapepr/\*.xml" />

</bean>

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

<property name="basePackage" value="com.yiibai.maper" />

</bean>

</beans>

配置文件 Configuration.xml 的内容如下：

<?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>

<typeAliases>

<typeAlias alias="User" type="com.yiibai.pojo.User"/>

<typeAlias alias="Order" type="com.yiibai.pojo.Order"/>

</typeAliases>

<!-- 与spring 集成之后,这些可以完全删除,数据库连接的管理交给 spring 去管理 -->

<!--

<environments default="development">

<environment id="development">

<transactionManager type="JDBC"/>

<dataSource type="POOLED">

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

<property name="url" value="jdbc:mysql://127.0.0.1:3306/mybatis?characterEncoding=utf8" />

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

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

</dataSource>

</environment>

</environments>

-->

<!-- 这里交给sqlSessionFactory 的 mapperLocations属性去得到所有配置信息 -->

<!--

<mappers>

<mapper resource="com/yihaomen/mapper/User.xml"/>

</mappers>

-->

</configuration>

UserMaper.xml 用于定义查询和数据对象映射，其内容如下：

<?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.yiibai.maper.UserMaper">

<!-- 为了返回list 类型而定义的returnMap -->

<resultMap type="User" id="resultUser">

<id column="id" property="id" />

<result column="username" property="username" />

<result column="mobile" property="mobile" />

</resultMap>

<!-- User 联合 Order 查询 方法的配置 (多对一的方式) -->

<resultMap id="resultUserOrders" type="Order">

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

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

<result property="money" column="money" />

<result property="userId" column="user\_id" />

<association property="user" javaType="User">

<id property="id" column="id" />

<result property="username" column="username" />

<result property="mobile" column="mobile" />

</association>

</resultMap>

<select id="getUserOrders" parameterType="int" resultMap="resultUserOrders">

SELECT u.\*,o.\* FROM `user` u, `order` o

WHERE u.id=o.user\_id AND u.id=#{id}

</select>

<select id="getUserById" resultMap="resultUser" parameterType="int">

SELECT \*

FROM user

WHERE id=#{id}

</select>

</mapper>

### 6、测试执行，输出结果

我们创建一个控制器类在包 com.yiibai.controller 下，类的名称为：UserController.java，其代码如下

package com.yiibai.controller;

import java.util.List;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

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

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

import com.yiibai.maper.UserMaper;

import com.yiibai.pojo.Order;

/\*\*

\* @describe: 读取一个用户下的所有订单

\* @author: Yiibai

\* @version: V1.0

\* @copyright http://www.yiibai.com

\*/

// http://localhost:8080/mybatis07-spring-mvc/user/orders

@Controller

@RequestMapping("/user")

public class UserController {

@Autowired

UserMaper userMaper;

@RequestMapping("/orders")

public ModelAndView listall(HttpServletRequest request,HttpServletResponse response){

List<Order> orders=userMaper.getUserOrders(1);

System.out.println("orders");

ModelAndView mav=new ModelAndView("user\_orders");

mav.addObject("orders",orders);

return mav;

}

}

接下来还需要创建一个 web 页面作为结果输出，在目录 WebRoot/WEB-INF/pages 下创建一个名为 user\_orders.jsp 文件，其代码如下：

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

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

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<title>用户订单列表</title>

</head>

<body>

<c:forEach items="${orders}" var="order">

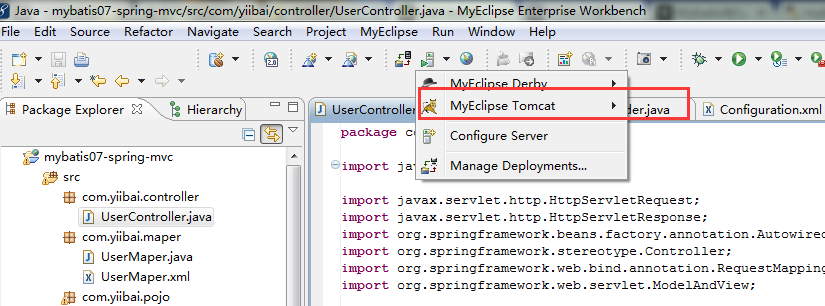
订单号：${order.orderNo }，订单总额：${order.money }<br />

</c:forEach>

</body>

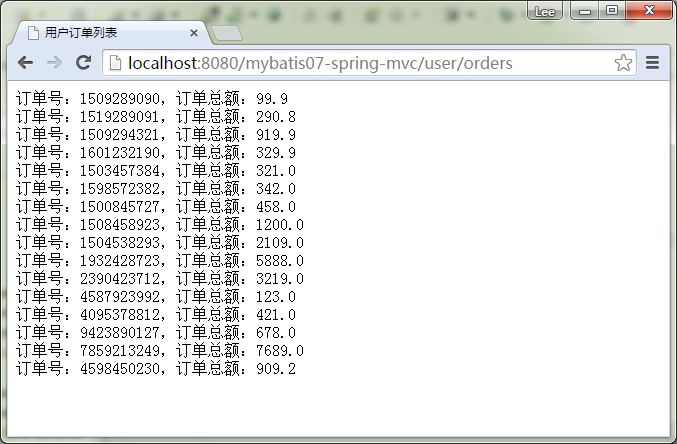
</html>

然后启动 MyEclipse 中的 tomcat，具体如下图所示：



注：web.xml 还需要配置一下，详细见下载代码。

在浏览器中打开网址：http://localhost:8080/mybatis07-spring-mvc/user/orders，输出结果如下图所示：



工程代码下载：<http://pan.baidu.com/s/1bnkE8cZ>

Jar 包下载：<http://pan.baidu.com/s/1bnyRJ9H>

## MyBatis分页

搞清楚什么是分页(pagination)

例如，在数据库的某个表里有1000条数据，我们每次只显示100条数据，在第1页显示第0到第99条，在第2页显示第100到199条，依次类推，这就是分页。

分页可以分为逻辑分页和物理分页。逻辑分页是我们的程序在显示每页的数据时，首先查询得到表中的1000条数据，然后成熟根据当前页的“页码”选出其中的100条数据来显示。

物理分页是程序先判断出该选出这1000条的第几条到第几条，然后数据库根据程序给出的信息查询出程序需要的100条返回给我们的程序。

### MyBatis 物理分页

MyBatis使用RowBounds实现的分页是逻辑分页,也就是先把数据记录全部查询出来,然在再根据 offset 和 limit 截断记录返回。

为了在数据库层面上实现物理分页，又不改变原来 MyBatis 的函数逻辑,可以编写 plugin 截获 MyBatis Executor 的 statementhandler，重写SQL来执行查询。

经常搭框架的人应该都清楚，框架搭建的核心就是配置文件。

在这里我们需要创建 web 工程。也需要用 mybatis与Spring mvc 集成起来，源码在本文结尾处下载，主要有以下几个方面的配置。

**整个Mybatis分页示例要完成的步骤如下：**

1、示例功能描述

2、创建工程

3、数据库表结构及数据记录

4、实例对象

5、配置文件

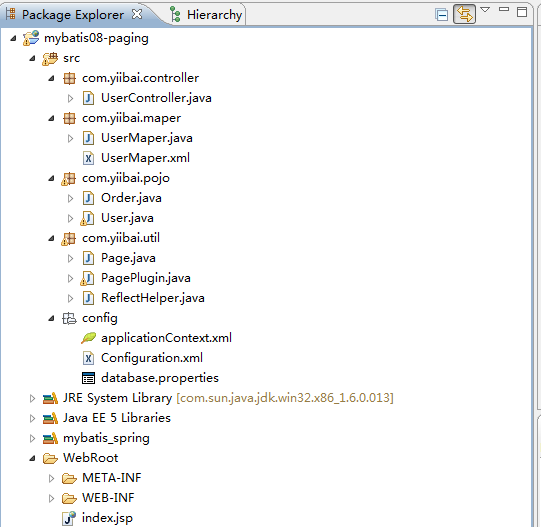
6、测试执行，输出结果

### 1、示例功能描述

在本示例中，需要使用 MyBatis和Spring MVC整合完成分页，完成这样的一个简单功能，即指定一个用户（ID=1），查询出这个用户关联的所有订单分页显示出来（使用的数据库是：MySQL）。

### 2、创建工程

首先创建一个工程的名称为：mybatis08-paging，在 src 源代码目录下建立文件夹 config，并将原来的 mybatis 配置文件 Configuration.xml 移动到这个文件夹中, 并在 config 文家夹中建立 Spring 配置文件：applicationContext.xml。工程结构目录如下：



### 3、数据库表结构及数据记录

在本示例中，用到两个表：用户表和订单表，其结构和数据记录如下：

CREATE TABLE `user` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`username` varchar(64) NOT NULL DEFAULT '',

`mobile` varchar(16) NOT NULL DEFAULT '',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of user

-- ----------------------------

INSERT INTO `user` VALUES ('1', 'yiibai', '13838009988');

INSERT INTO `user` VALUES ('2', 'saya', '13838009988');

订单表结构和数据如下：

CREATE TABLE `order` (

`order\_id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`user\_id` int(10) unsigned NOT NULL DEFAULT '0',

`order\_no` varchar(16) NOT NULL DEFAULT '',

`money` float(10,2) unsigned DEFAULT '0.00',

PRIMARY KEY (`order\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=17 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of order

-- ----------------------------

INSERT INTO `order` VALUES ('1', '1', '1509289090', '99.90');

INSERT INTO `order` VALUES ('2', '1', '1519289091', '290.80');

INSERT INTO `order` VALUES ('3', '1', '1509294321', '919.90');

INSERT INTO `order` VALUES ('4', '1', '1601232190', '329.90');

INSERT INTO `order` VALUES ('5', '1', '1503457384', '321.00');

INSERT INTO `order` VALUES ('6', '1', '1598572382', '342.00');

INSERT INTO `order` VALUES ('7', '1', '1500845727', '458.00');

INSERT INTO `order` VALUES ('8', '1', '1508458923', '1200.00');

INSERT INTO `order` VALUES ('9', '1', '1504538293', '2109.00');

INSERT INTO `order` VALUES ('10', '1', '1932428723', '5888.00');

INSERT INTO `order` VALUES ('11', '1', '2390423712', '3219.00');

INSERT INTO `order` VALUES ('12', '1', '4587923992', '123.00');

INSERT INTO `order` VALUES ('13', '1', '4095378812', '421.00');

INSERT INTO `order` VALUES ('14', '1', '9423890127', '678.00');

INSERT INTO `order` VALUES ('15', '1', '7859213249', '7689.00');

INSERT INTO `order` VALUES ('16', '1', '4598450230', '909.20');

### 4、实例对象

用户表和订单表分别对应两个实例对象，分别是：User.java 和 Order.java，它们都在 com.yiibai.pojo 包中。

User.java代码内容如下：

package com.yiibai.pojo;

import java.util.List;

/\*\*

\* @describe: User

\* @author: Yiibai

\* @version: V1.0

\* @copyright http://www.yiibai.com

\*/

public class User {

private int id;

private String username;

private String mobile;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getMobile() {

return mobile;

}

public void setMobile(String mobile) {

this.mobile = mobile;

}

}

Order.java代码内容如下：

package com.yiibai.pojo;

/\*\*

\* @describe: User - 订单

\* @author: Yiibai

\* @version: V1.0

\* @copyright http://www.yiibai.com

\*/

public class Order {

private int orderId;

private String orderNo;

private float money;

private int userId;

private User user;

public int getUserId() {

return userId;

}

public void setUserId(int userId) {

this.userId = userId;

}

public int getOrderId() {

return orderId;

}

public void setOrderId(int orderId) {

this.orderId = orderId;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public String getOrderNo() {

return orderNo;

}

public void setOrderNo(String orderNo) {

this.orderNo = orderNo;

}

public float getMoney() {

return money;

}

public void setMoney(float money) {

this.money = money;

}

}

### 5、配置文件

这个实例中有三个重要的配置文件，它们分别是：applicationContext.xml ， Configuration.xml 以及 UserMaper.xml。

applicationContext.xml  配置文件里最主要的配置：

<?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:aop="http://www.springframework.org/schema/aop"

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

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

xsi:schemaLocation="

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

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

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

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

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

default-autowire="byName" default-lazy-init="false">

<!--本示例采用DBCP连接池，应预先把DBCP的jar包复制到工程的lib目录下。 -->

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

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

destroy-method="close" p:driverClassName="com.mysql.jdbc.Driver"

p:url="jdbc:mysql://127.0.0.1:3306/yiibai?characterEncoding=utf8"

p:username="root" p:password="" p:maxActive="10" p:maxIdle="10">

</bean>

<bean id="transactionManager"

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

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

</bean>

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

<!--dataSource属性指定要用到的连接池-->

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

<!--configLocation属性指定mybatis的核心配置文件-->

<property name="configLocation" value="classpath:config/Configuration.xml" />

<!-- 所有配置的mapper文件 -->

<property name="mapperLocations" value="classpath\*:com/yiibai/mapepr/\*.xml" />

</bean>

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

<property name="basePackage" value="com.yiibai.maper" />

</bean>

</beans>

配置文件 Configuration.xml 的内容如下：

<?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>

<typeAliases>

<typeAlias alias="User" type="com.yiibai.pojo.User"/>

<typeAlias alias="Order" type="com.yiibai.pojo.Order"/>

</typeAliases>

<!-- 与spring 集成之后,这些可以完全删除,数据库连接的管理交给 spring 去管理 -->

<!--

<environments default="development">

<environment id="development">

<transactionManager type="JDBC"/>

<dataSource type="POOLED">

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

<property name="url" value="jdbc:mysql://127.0.0.1:3306/mybatis?characterEncoding=utf8" />

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

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

</dataSource>

</environment>

</environments>

-->

<!-- 这里交给sqlSessionFactory 的 mapperLocations属性去得到所有配置信息 -->

<!--

<mappers>

<mapper resource="com/yihaomen/mapper/User.xml"/>

</mappers>

-->

</configuration>

UserMaper.xml 用于定义查询和数据对象映射，其内容如下：

<?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.yiibai.maper.UserMaper">

<!-- 为了返回list 类型而定义的returnMap -->

<resultMap type="User" id="resultUser">

<id column="id" property="id" />

<result column="username" property="username" />

<result column="mobile" property="mobile" />

</resultMap>

<!-- User 联合 Order 查询 方法的配置 (多对一的方式) -->

<resultMap id="resultUserOrders" type="Order">

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

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

<result property="money" column="money" />

<result property="userId" column="user\_id" />

<association property="user" javaType="User">

<id property="id" column="id" />

<result property="username" column="username" />

<result property="mobile" column="mobile" />

</association>

</resultMap>

<select id="getUserOrders" parameterType="int" resultMap="resultUserOrders">

SELECT u.\*,o.\* FROM `user` u, `order` o

WHERE u.id=o.user\_id AND u.id=#{id}

</select>

<select id="getUserById" resultMap="resultUser" parameterType="int">

SELECT \*

FROM user

WHERE id=#{id}

</select>

</mapper>

### 6、测试执行，输出结果

我们创建一个控制器类在包 com.yiibai.controller 下，类的名称为：UserController.java，这里新增了一个方法：pageList， 对应请求URL是 /orderpages，其代码如下：

package com.yiibai.controller;

import java.util.List;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

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

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

import com.yiibai.maper.UserMaper;

import com.yiibai.pojo.Order;

import com.yiibai.util.Page;

// http://localhost:8080/mybatis08-paging/user/orders

@Controller

@RequestMapping("/user")

public class UserController {

@Autowired

UserMaper userMaper;

/\*\*

\* 某一个用户下的所有订单

\*

\* @param request

\* @param response

\* @return

\*/

@RequestMapping("/orders")

public ModelAndView listall(HttpServletRequest request,

HttpServletResponse response) {

List<Order> orders = userMaper.getUserOrders(1);

System.out.println("orders");

ModelAndView mav = new ModelAndView("user\_orders");

mav.addObject("orders", orders);

return mav;

}

/\*\*

\* 订单分页

\*

\* @param request

\* @param response

\* @return

\*/

@RequestMapping("/orderpages")

public ModelAndView pageList(HttpServletRequest request,

HttpServletResponse response) {

int currentPage = request.getParameter("page") == null ? 1 : Integer

.parseInt(request.getParameter("page"));

int pageSize = 3;

if (currentPage <= 0) {

currentPage = 1;

}

int currentResult = (currentPage - 1) \* pageSize;

System.out.println(request.getRequestURI());

System.out.println(request.getQueryString());

Page page = new Page();

page.setShowCount(pageSize);

page.setCurrentResult(currentResult);

List<Order> orders = userMaper.getOrderListPage(page, 1);

System.out.println("Current page =>" + page);

int totalCount = page.getTotalResult();

int lastPage = 0;

if (totalCount % pageSize == 0) {

lastPage = totalCount % pageSize;

} else {

lastPage = 1 + totalCount / pageSize;

}

if (currentPage >= lastPage) {

currentPage = lastPage;

}

String pageStr = "";

pageStr = String.format(

"<a href=\"%s\">上一页</a> <a href=\"%s\">下一页</a>", request

.getRequestURI()

+ "?page=" + (currentPage - 1), request.getRequestURI()

+ "?page=" + (currentPage + 1));

// 制定视图，也就是list.jsp

ModelAndView mav = new ModelAndView("pagelist");

mav.addObject("orders", orders);

mav.addObject("pagelist", pageStr);

return mav;

}

}

注意，在这个分页工程中，在 com.yiibai.util 包下新增了几个类，它们分别是：PagePlugin.java，Page.java, PageHelper.java，其中 PagePlugin 是针对 MyBatis 分页的插件。由于代码太多，这里列出 PagePlugin.java 的代码，其它两个类的代码有兴趣的读者可以在下载代码后阅读取研究。 PagePlugin.java 的代码如下所示：

package com.yiibai.util;

import java.lang.reflect.Field;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.List;

import java.util.Map;

import java.util.Properties;

import javax.xml.bind.PropertyException;

import org.apache.ibatis.executor.ErrorContext;

import org.apache.ibatis.executor.Executor;

import org.apache.ibatis.executor.ExecutorException;

import org.apache.ibatis.executor.statement.BaseStatementHandler;

import org.apache.ibatis.executor.statement.RoutingStatementHandler;

import org.apache.ibatis.executor.statement.StatementHandler;

import org.apache.ibatis.mapping.BoundSql;

import org.apache.ibatis.mapping.MappedStatement;

import org.apache.ibatis.mapping.ParameterMapping;

import org.apache.ibatis.mapping.ParameterMode;

import org.apache.ibatis.plugin.Interceptor;

import org.apache.ibatis.plugin.Intercepts;

import org.apache.ibatis.plugin.Invocation;

import org.apache.ibatis.plugin.Plugin;

import org.apache.ibatis.plugin.Signature;

import org.apache.ibatis.reflection.MetaObject;

import org.apache.ibatis.reflection.property.PropertyTokenizer;

import org.apache.ibatis.scripting.xmltags.ForEachSqlNode;

import org.apache.ibatis.session.Configuration;

import org.apache.ibatis.session.ResultHandler;

import org.apache.ibatis.session.RowBounds;

import org.apache.ibatis.type.TypeHandler;

import org.apache.ibatis.type.TypeHandlerRegistry;

@Intercepts( { @Signature(type = StatementHandler.class, method = "prepare", args = { Connection.class }) })

public class PagePlugin implements Interceptor {

private static String dialect = "";

private static String pageSqlId = "";

@SuppressWarnings("unchecked")

public Object intercept(Invocation ivk) throws Throwable {

if (ivk.getTarget() instanceof RoutingStatementHandler) {

RoutingStatementHandler statementHandler = (RoutingStatementHandler) ivk

.getTarget();

BaseStatementHandler delegate = (BaseStatementHandler) ReflectHelper

.getValueByFieldName(statementHandler, "delegate");

MappedStatement mappedStatement = (MappedStatement) ReflectHelper

.getValueByFieldName(delegate, "mappedStatement");

if (mappedStatement.getId().matches(pageSqlId)) {

BoundSql boundSql = delegate.getBoundSql();

Object parameterObject = boundSql.getParameterObject();

if (parameterObject == null) {

throw new NullYiibaierException("parameterObject error");

} else {

Connection connection = (Connection) ivk.getArgs()[0];

String sql = boundSql.getSql();

String countSql = "select count(0) from (" + sql

+ ") myCount";

System.out.println("总数sql 语句:" + countSql);

PreparedStatement countStmt = connection

.prepareStatement(countSql);

BoundSql countBS = new BoundSql(mappedStatement

.getConfiguration(), countSql, boundSql

.getParameterMappings(), parameterObject);

setParameters(countStmt, mappedStatement, countBS,

parameterObject);

ResultSet rs = countStmt.executeQuery();

int count = 0;

if (rs.next()) {

count = rs.getInt(1);

}

rs.close();

countStmt.close();

Page page = null;

if (parameterObject instanceof Page) {

page = (Page) parameterObject;

page.setTotalResult(count);

} else if (parameterObject instanceof Map) {

Map<String, Object> map = (Map<String, Object>) parameterObject;

page = (Page) map.get("page");

if (page == null)

page = new Page();

page.setTotalResult(count);

} else {

Field pageField = ReflectHelper.getFieldByFieldName(

parameterObject, "page");

if (pageField != null) {

page = (Page) ReflectHelper.getValueByFieldName(

parameterObject, "page");

if (page == null)

page = new Page();

page.setTotalResult(count);

ReflectHelper.setValueByFieldName(parameterObject,

"page", page);

} else {

throw new NoSuchFieldException(parameterObject

.getClass().getName());

}

}

String pageSql = generatePageSql(sql, page);

System.out.println("page sql:" + pageSql);

ReflectHelper.setValueByFieldName(boundSql, "sql", pageSql);

}

}

}

return ivk.proceed();

}

private void setParameters(PreparedStatement ps,

MappedStatement mappedStatement, BoundSql boundSql,

Object parameterObject) throws SQLException {

ErrorContext.instance().activity("setting parameters").object(

mappedStatement.getParameterMap().getId());

List<ParameterMapping> parameterMappings = boundSql

.getParameterMappings();

if (parameterMappings != null) {

Configuration configuration = mappedStatement.getConfiguration();

TypeHandlerRegistry typeHandlerRegistry = configuration

.getTypeHandlerRegistry();

MetaObject metaObject = parameterObject == null ? null

: configuration.newMetaObject(parameterObject);

for (int i = 0; i < parameterMappings.size(); i++) {

ParameterMapping parameterMapping = parameterMappings.get(i);

if (parameterMapping.getMode() != ParameterMode.OUT) {

Object value;

String propertyName = parameterMapping.getProperty();

PropertyTokenizer prop = new PropertyTokenizer(propertyName);

if (parameterObject == null) {

value = null;

} else if (typeHandlerRegistry

.hasTypeHandler(parameterObject.getClass())) {

value = parameterObject;

} else if (boundSql.hasAdditionalParameter(propertyName)) {

value = boundSql.getAdditionalParameter(propertyName);

} else if (propertyName

.startsWith(ForEachSqlNode.ITEM\_PREFIX)

&& boundSql.hasAdditionalParameter(prop.getName())) {

value = boundSql.getAdditionalParameter(prop.getName());

if (value != null) {

value = configuration.newMetaObject(value)

.getValue(

propertyName.substring(prop

.getName().length()));

}

} else {

value = metaObject == null ? null : metaObject

.getValue(propertyName);

}

TypeHandler typeHandler = parameterMapping.getTypeHandler();

if (typeHandler == null) {

throw new ExecutorException(

"There was no TypeHandler found for parameter "

+ propertyName + " of statement "

+ mappedStatement.getId());

}

typeHandler.setParameter(ps, i + 1, value, parameterMapping

.getJdbcType());

}

}

}

}

private String generatePageSql(String sql, Page page) {

if (page != null && (dialect != null || !dialect.equals(""))) {

StringBuffer pageSql = new StringBuffer();

if ("mysql".equals(dialect)) {

pageSql.append(sql);

pageSql.append(" limit " + page.getCurrentResult() + ","

+ page.getShowCount());

} else if ("oracle".equals(dialect)) {

pageSql

.append("select \* from (select tmp\_tb.\*,ROWNUM row\_id from (");

pageSql.append(sql);

pageSql.append(") tmp\_tb where ROWNUM<=");

pageSql.append(page.getCurrentResult() + page.getShowCount());

pageSql.append(") where row\_id>");

pageSql.append(page.getCurrentResult());

}

return pageSql.toString();

} else {

return sql;

}

}

public Object plugin(Object arg0) {

// TODO Auto-generated method stub

return Plugin.wrap(arg0, this);

}

public void setProperties(Properties p) {

dialect = p.getProperty("dialect");

if (dialect == null || dialect.equals("")) {

try {

throw new PropertyException("dialect property is not found!");

} catch (PropertyException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

pageSqlId = p.getProperty("pageSqlId");

if (dialect == null || dialect.equals("")) {

try {

throw new PropertyException("pageSqlId property is not found!");

} catch (PropertyException e) {

// TODO Auto-generated catch block

e.printStackTrace();

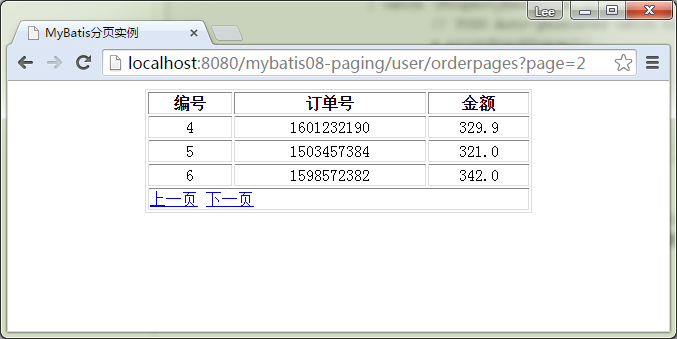
}

}

}

}

接下来部署 mybatis08-paging 这个工程，启动 tomcat ，打开浏览器输入网址：<http://localhost:8080/mybatis08-paging/user/orderpages> ，显示结果如下：



工程 mybatis08-paging 的代码下载：<http://pan.baidu.com/s/1pJxhvNt>

Jar 包下载：<http://pan.baidu.com/s/1bnyRJ9H>

## MyBatis动态SQL语句

MyBatis 的强大特性之一便是它的动态 SQL。如果你有使用 JDBC 或其他类似框架的经验，你就能体会到根据不同条件拼接 SQL 语句有多么痛苦。拼接的时候要确保不能忘了必要的空格，还要注意省掉列名列表最后的逗号。利用动态 SQL 这一特性可以彻底摆脱这种痛苦。

通常使用动态 SQL 不可能是独立的一部分,MyBatis 当然使用一种强大的动态 SQL 语言来改进这种情形,这种语言可以被用在任意的 SQL 映射语句中。

动态 SQL 元素和使用 JSTL 或其他类似基于 XML 的文本处理器相似。在 MyBatis 之前的版本中,有很多的元素需要来了解。MyBatis 3 大大提升了它们,现在用不到原先一半的元素就可以了。MyBatis 采用功能强大的基于 OGNL 的表达式来消除其他元素。

mybatis 的动态sql语句是基于OGNL表达式的。可以方便的在 sql 语句中实现某些逻辑. 总体说来mybatis 动态SQL 语句主要有以下几类:  
1. if 语句 (简单的条件判断)  
2. choose (when,otherwize) ,相当于java 语言中的 switch ,与 jstl 中的choose 很类似.  
3. trim (对包含的内容加上 prefix,或者 suffix 等，前缀，后缀)  
4. where (主要是用来简化sql语句中where条件判断的，能智能的处理 and or ,不必担心多余导致语法错误)  
5. set (主要用于更新时)  
6. foreach (在实现 mybatis in 语句查询时特别有用)

### if

动态 SQL 通常要做的事情是有条件地包含 where 子句的一部分。比如:

<select id="findActiveBlogWithTitleLike"

resultType="Blog">

SELECT \* FROM BLOG

WHERE state = ‘ACTIVE’

<if test="title != null">

AND title like #{title}

</if>

</select>

这条语句提供了一个可选的文本查找类型的功能。如果没有传入“title”，那么所有处于“ACTIVE”状态的BLOG都会返回；反之若传入了“title”，那么就会把模糊查找“title”内容的BLOG结果返回(就这个例子而言，细心的读者会发现其中的参数值是可以包含一些掩码或通配符的)。

如果想可选地通过“title”和“author”两个条件搜索该怎么办呢？首先，改变语句的名称让它更具实际意义；然后只要加入另一个条件即可。

<select id="findActiveBlogLike"

resultType="Blog">

SELECT \* FROM BLOG WHERE state = ‘ACTIVE’

<if test="title != null">

AND title like #{title}

</if>

<if test="author != null and author.name != null">

AND author\_name like #{author.name}

</if>

</select>

### choose, when, otherwise

有些时候，我们不想用到所有的条件语句，而只想从中择其一二。针对这种情况，MyBatis 提供了 choose 元素，它有点像 Java 中的 switch 语句。

还是上面的例子，但是这次变为提供了“title”就按“title”查找，提供了“author”就按“author”查找，若两者都没有提供，就返回所有符合条件的BLOG(实际情况可能是由管理员按一定策略选出BLOG列表，而不是返回大量无意义的随机结果)。

<select id="findActiveBlogLike"

resultType="Blog">

SELECT \* FROM BLOG WHERE state = ‘ACTIVE’

<choose>

<when test="title != null">

AND title like #{title}

</when>

<when test="author != null and author.name != null">

AND author\_name like #{author.name}

</when>

<otherwise>

AND featured = 1

</otherwise>

</choose>

</select>

### trim, where, set

前面几个例子已经合宜地解决了一个臭名昭著的动态 SQL 问题。现在考虑回到“if”示例，这次我们将“ACTIVE = 1”也设置成动态的条件，看看会发生什么。

<select id="findActiveBlogLike"

resultType="Blog">

SELECT \* FROM BLOG

WHERE

<if test="state != null">

state = #{state}

</if>

<if test="title != null">

AND title like #{title}

</if>

<if test="author != null and author.name != null">

AND author\_name like #{author.name}

</if>

</select>

如果这些条件没有一个能匹配上将会怎样？最终这条 SQL 会变成这样：

SELECT \* FROM BLOG

WHERE

这会导致查询失败。如果仅仅第二个条件匹配又会怎样？这条 SQL 最终会是这样:

SELECT \* FROM BLOG

WHERE

AND title like ‘yiibai.com’

这个查询也会失败。这个问题不能简单的用条件句式来解决，如果你也曾经被迫这样写过，那么你很可能从此以后都不想再这样去写了。

MyBatis 有一个简单的处理，这在90%的情况下都会有用。而在不能使用的地方，你可以自定义处理方式来令其正常工作。一处简单的修改就能得到想要的效果：

<select id="findActiveBlogLike"

resultType="Blog">

SELECT \* FROM BLOG

<where>

<if test="state != null">

state = #{state}

</if>

<if test="title != null">

AND title like #{title}

</if>

<if test="author != null and author.name != null">

AND author\_name like #{author.name}

</if>

</where>

</select>

where 元素知道只有在一个以上的if条件有值的情况下才去插入“WHERE”子句。而且，若最后的内容是“AND”或“OR”开头的，where 元素也知道如何将他们去除。

如果 where 元素没有按正常套路出牌，我们还是可以通过自定义 trim 元素来定制我们想要的功能。比如，和 where 元素等价的自定义 trim 元素为：

<trim prefix="WHERE" prefixOverrides="AND |OR "> ... </trim>

prefixOverrides 属性会忽略通过管道分隔的文本序列(注意此例中的空格也是必要的)。它带来的结果就是所有在 prefixOverrides 属性中指定的内容将被移除，并且插入 prefix 属性中指定的内容。

类似的用于动态更新语句的解决方案叫做 set。set 元素可以被用于动态包含需要更新的列，而舍去其他的。比如：

<update id="updateAuthorIfNecessary">

update Author

<set>

<if test="username != null">username=#{username},</if>

<if test="password != null">password=#{password},</if>

<if test="email != null">email=#{email},</if>

<if test="bio != null">bio=#{bio}</if>

</set>

where id=#{id}

</update>

这里，set 元素会动态前置 SET 关键字，同时也会消除无关的逗号，因为用了条件语句之后很可能就会在生成的赋值语句的后面留下这些逗号。

若你对等价的自定义 trim 元素的样子感兴趣，那这就应该是它的真面目：

<trim prefix="SET" suffixOverrides=","> ... </trim>

注意这里我们忽略的是后缀中的值，而又一次附加了前缀中的值。

### foreach

动态 SQL 的另外一个常用的必要操作是需要对一个集合进行遍历，通常是在构建 IN 条件语句的时候。比如：

<select id="selectPostIn" resultType="domain.blog.Post">

SELECT \*

FROM POST P

WHERE ID in

<foreach item="item" index="index" collection="list"

open="(" separator="," close=")">

#{item}

</foreach>

</select>

foreach 元素的功能是非常强大的，它允许你指定一个集合，声明可以用在元素体内的集合项和索引变量。它也允许你指定开闭匹配的字符串以及在迭代中间放置分隔符。这个元素是很智能的，因此它不会偶然地附加多余的分隔符。

注意 你可以将一个 List 实例或者数组作为参数对象传给 MyBatis，当你这么做的时候，MyBatis 会自动将它包装在一个 Map 中并以名称为键。List 实例将会以“list”作为键，而数组实例的键将是“array”。

到此我们已经完成了涉及 XML 配置文件和 XML 映射文件的讨论。下一部分将详细探讨 Java API，这样才能从已创建的映射中获取最大利益。

### bind

bind 元素可以从 OGNL 表达式中创建一个变量并将其绑定到上下文。比如：

<select id="selectBlogsLike" resultType="Blog">

<bind name="pattern" value="'%' + \_parameter.getTitle() + '%'" />

SELECT \* FROM BLOG

WHERE title LIKE #{pattern}

</select>

### Multi-db vendor support

一个配置了“\_databaseId”变量的 databaseIdProvider 对于动态代码来说是可用的，这样就可以根据不同的数据库厂商构建特定的语句。比如下面的例子：

<insert id="insert">

<selectKey keyProperty="id" resultType="int" order="BEFORE">

<if test="\_databaseId == 'oracle'">

select seq\_users.nextval from dual

</if>

<if test="\_databaseId == 'db2'">

select nextval for seq\_users from sysibm.sysdummy1"

</if>

</selectKey>

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

</insert>

### 动态 SQL 中可插拔的脚本语言

MyBatis 从 3.2 开始支持可插拔的脚本语言，因此你可以在插入一种语言的驱动(language driver)之后来写基于这种语言的动态 SQL 查询。

可以通过实现下面接口的方式来插入一种语言：

public interface LanguageDriver {

ParameterHandler createParameterHandler(MappedStatement mappedStatement, Object parameterObject, BoundSql boundSql);

SqlSource createSqlSource(Configuration configuration, XNode script, Class<?> parameterType);

SqlSource createSqlSource(Configuration configuration, String script, Class<?> parameterType);

}

一旦有了自定义的语言驱动，你就可以在 mybatis-config.xml 文件中将它设置为默认语言：

<typeAliases>

<typeAlias type="org.sample.MyLanguageDriver" alias="myLanguage"/>

</typeAliases>

<settings>

<setting name="defaultScriptingLanguage" value="myLanguage"/>

</settings>

除了设置默认语言，你也可以针对特殊的语句指定特定语言，这可以通过如下的 lang 属性来完成：

<select id="selectBlog" lang="myLanguage"> SELECT \* FROM BLOG </select>

或者在你正在使用的映射中加上注解 @Lang 来完成：

public interface Mapper {

@Lang(MyLanguageDriver.class)

@Select("SELECT \* FROM BLOG")

List<Blog> selectBlog();

}

## mybaits if标签语句

### Mybatis if 标签可用在许多类型的 [SQL](http://www.yiibai.com/sql/)语句中，我们以查询为例。首先看一个很普通的查询：

<!-- 查询用户列表，like用户名称 -->

<select id="getUserListLikeName" parameterType="User" resultMap="userResultMap">

SELECT \* from user u

WHERE u.username LIKE CONCAT(CONCAT('%', #{username}),'%')

</select>

但是当 username 或 sex 为 null 时，此语句很可能报错或查询结果为空。此时我们使用 if 动态 sql 语句先进行判断，如果值为 null 或等于空字符串，我们就不进行此条件的判断，增加灵活性。

参数为实体类：User。将实体类中所有的属性均进行判断，如果不为空则执行判断条件。

<!-- 添加 if(判断参数) - 将实体类 User 不为空的属性作为 where 条件 -->

<select id="getUserList" resultMap="resultMap\_User" parameterType="com.yiibai.pojo.User">

SELECT u.username,

u.password,

u.sex,

u.birthday,

u.photo,

u.score,

u.sign

FROM user u

WHERE

<if test="username !=null ">

u.username LIKE CONCAT(CONCAT('%', #{username, jdbcType=VARCHAR}),'%')

</if>

<if test="sex!= null and sex != '' ">

AND u.sex = #{Sex, jdbcType=INTEGER}

</if>

<if test="birthday != null ">

AND u.birthday = #{birthday, jdbcType=DATE}

</if>

<if test="userId != null and userId != '' ">

AND id.user\_id = #{userId, jdbcType=VARCHAR}

</if>

</select>

使用时比较灵活，创建新的一个这样的实体类，我们需要限制那个条件，只需要附上相应的值就会 where 这个条件，相反不去赋值就可以不在 where 中判断。

public void select\_by\_if() {

User user = new User();

user.setUsername("");

user.setSex(1);

user.setBirthday(DateUtil.parse("1990-08-18"));

List<User> userList = this.dynamicSqlMapper.getUserList\_if(user);

for (user u : userList) {

System.out.println(u.toString());

}

}

我们再看看一下另一个示例，先来看看下面的代码：

<select id="dynamicIfTest" parameterType="Blog" resultType="Blog">

select \* from t\_blog where 1 = 1

<if test="title != null">

and title = #{title}

</if>

<if test="content != null">

and content = #{content}

</if>

<if test="owner != null">

and owner = #{owner}

</if>

</select>这条语句的意思非常简单，如果提供了 title 参数，那么就要满足 title=#{title}，同样如果提供了 Content 和 Owner 的时候，它们也需要满足相应的条件，之后就是返回满足这些条件的所有 Blog，这是非常有用的一个功能，以往我们使用其他类型框架或者直接使用 [JDBC](http://www.yiibai.com/jdbc/) 的时候， 如果我们要达到同样的选择效果的时候，我们就需要拼 SQL 语句，这是极其麻烦的，比起来，上述的动态SQL就比较简单了。

## MyBatis where标签语句

### 当 where 中的条件使用的 if 标签较多时，这样的组合可能会导致错误。当 java 代码按如下方法调用时：

@Test

public void select\_test\_where() {

User user = new User();

user.setUsername(null);

user.setSex(1);

List<User> userList = this.dynamicSqlMapper.getUsertList\_where(user);

for (User u : userList ) {

System.out.println(u.toString());

}

}

如果上面例子，参数 username 为 null，将不会进行列 username 的判断，则会直接导“WHERE AND”关键字多余的错误 SQL。

这时可以使用 where 动态语句来解决。“where”标签会知道如果它包含的标签中有返回值的话，它就插入一个‘where’。此外，如果标签返回的内容是以 AND 或OR 开头的，则它会剔除掉。

上面例子修改为：

<select id="getUserList\_whereIf" resultMap="resultMap\_User" parameterType="com.yiibai.pojo.User">

SELECT u.user\_id,

u.username,

u.sex,

u.birthday

FROM User u

<where>

<if test="username !=null ">

u.username LIKE CONCAT(CONCAT('%', #{username, jdbcType=VARCHAR}),'%')

</if>

<if test="sex != null and sex != '' ">

AND u.sex = #{sex, jdbcType=INTEGER}

</if>

<if test="birthday != null ">

AND u.birthday = #{birthday, jdbcType=DATE}

</if>

</where>

</select>

where 主要是用来简化 sql 语句中 where 条件判断，自动地处理 AND/OR 条件。

<select id="dynamicWhereTest" parameterType="Blog" resultType="Blog">

select \* from t\_blog

<where>

<if test="title != null">

title = #{title}

</if>

<if test="content != null">

and content = #{content}

</if>

<if test="owner != null">

and owner = #{owner}

</if>

</where>

</select>

where 元素的作用是会在写入 where 元素的地方输出一个 where，另外一个好处是你不需要考虑 where 元素里面的条件输出是什么样子的，MyBatis 会智能的帮处理，如果所有的条件都不满足那么 MyBatis 就会查出所有的记录，如果输出后是 and 开头的，MyBatis 会把第一个and忽略，当然如果是 or 开头的，MyBatis 也会把它忽略；此外，在 where 元素中你不需要考虑空格的问题，MyBatis 会智能的帮你加上。像上述例子中，如果 title=null， 而 content != null，那么输出的整个语句会是 select \* from t\_blog where content = #{content}，而不是 select \* from t\_blog where and content = #{content}，因为 MyBatis 会自动地把首个 and / or 给忽略。

## Mybatis set标签

### set - 更新语句

当 update 语句中没有使用 if 标签时，如果有一个参数为 null，都会导致错误。

当在 update 语句中使用if标签时，如果前面的if没有执行，则或导致逗号多余错误。使用set标签可以将动态的配置 SET 关键字，并剔除追加到条件末尾的任何不相关的逗号。使用 if+set 标签修改后，如果某项为 null 则不进行更新，而是保持数据库原值。如下示例：

<!-- if/set(判断参数) - 将实体 User类不为空的属性更新 -->

<update id="updateUser\_if\_set" parameterType="com.pojo.User">

UPDATE user

<set>

<if test="username!= null and username != '' ">

username = #{username},

</if>

<if test="sex!= null and sex!= '' ">

sex = #{sex},

</if>

<if test="birthday != null ">

birthday = #{birthday},

</if>

</set>

WHERE user\_id = #{userid};

</update>

再看看下面的一个示例：

<update id="dynamicSetTest" parameterType="Blog">

update t\_blog

<set>

<if test="title != null">

title = #{title},

</if>

<if test="content != null">

content = #{content},

</if>

<if test="owner != null">

owner = #{owner}

</if>

</set>

where id = #{id}

</update>

### set 标签元素主要是用在更新操作的时候，它的主要功能和 where 标签元素其实是差不多的，主要是在包含的语句前输出一个 set，然后如果包含的语句是以逗号结束的话将会把该逗号忽略，如果 set 包含的内容为空的话则会出错。有了 set 元素就可以动态的更新那些修改了的字段。

## Mybatis trim标签

### trim代替where/set标签

trim 是更灵活用来去处多余关键字的标签，它可以用来实现 where 和 set 的效果。

<!-- 使用 if/trim 代替 where(判断参数) - 将 User 类不为空的属性作为 where 条件 -->

<select id="getUsertList\_if\_trim" resultMap="resultMap\_User">

SELECT \*

FROM user u

<trim prefix="WHERE" prefixOverrides="AND|OR">

<if test="username !=null ">

u.username LIKE CONCAT(CONCAT('%', #{username, jdbcType=VARCHAR}),'%')

</if>

<if test="sex != null and sex != '' ">

AND u.sex = #{sex, jdbcType=INTEGER}

</if>

<if test="birthday != null ">

AND u.birthday = #{birthday, jdbcType=DATE}

</if>

</trim>

</select>

### trim 代替 set

<!-- if/trim代替set(判断参数) - 将 User 类不为空的属性更新 -->

<update id="updateUser\_if\_trim" parameterType="com.yiibai.pojo.User">

UPDATE user

<trim prefix="SET" suffixOverrides=",">

<if test="username != null and username != '' ">

username = #{username},

</if>

<if test="sex != null and sex != '' ">

sex = #{sex},

</if>

<if test="birthday != null ">

birthday = #{birthday},

</if>

</trim>

WHERE user\_id = #{user\_id}

</update>

**trim (对包含的内容加上 prefix,或者 suffix 等，前缀，后缀)**

<select id="dynamicTrimTest" parameterType="Blog" resultType="Blog">

select \* from t\_blog

<trim prefix="where" prefixOverrides="and |or">

<if test="title != null">

title = #{title}

</if>

<if test="content != null">

and content = #{content}

</if>

<if test="owner != null">

or owner = #{owner}

</if>

</trim>

</select>

trim 元素的主要功能是可以在自己包含的内容前加上某些前缀，也可以在其后加上某些后缀，与之对应的属性是 prefix 和 suffix；可以把包含内容的首部某些内容覆盖，即忽略，也可以把尾部的某些内容覆盖，对应的属性是 prefixOverrides 和 suffixOverrides；正因为 trim 有这样的功能，所以我们也可以非常简单的利用 trim 来代替 where 元素的功能。

## MyBatis choose(when, otherwise)标签

### choose (when, otherwise)标签

有时候我们并不想应用所有的条件，而只是想从多个选项中选择一个。而使用if标签时，只要test中的表达式为 true，就会执行 if 标签中的条件。MyBatis 提供了 choose 元素。if标签是与(and)的关系，而 choose 是或(or)的关系。

choose标签是按顺序判断其内部when标签中的test条件出否成立，如果有一个成立，则 choose 结束。当 choose 中所有 when 的条件都不满则时，则执行 otherwise 中的sql。类似于Java 的 switch 语句，choose 为 switch，when 为 case，otherwise 则为 default。

例如下面例子，同样把所有可以限制的条件都写上，方面使用。choose会从上到下选择一个when标签的test为true的sql执行。安全考虑，我们使用where将choose包起来，放置关键字多于错误。

<!-- choose(判断参数) - 按顺序将实体类 User 第一个不为空的属性作为：where条件 -->

<select id="getUserList\_choose" resultMap="resultMap\_user" parameterType="com.yiibai.pojo.User">

SELECT \*

FROM User u

<where>

<choose>

<when test="username !=null ">

u.username LIKE CONCAT(CONCAT('%', #{username, jdbcType=VARCHAR}),'%')

</when >

<when test="sex != null and sex != '' ">

AND u.sex = #{sex, jdbcType=INTEGER}

</when >

<when test="birthday != null ">

AND u.birthday = #{birthday, jdbcType=DATE}

</when >

<otherwise>

</otherwise>

</choose>

</where>

</select>

choose (when,otherwize) ,相当于java 语言中的 switch ,与 jstl 中 的 choose 很类似。

<select id="dynamicChooseTest" parameterType="Blog" resultType="Blog">

select \* from t\_blog where 1 = 1

<choose>

<when test="title != null">

and title = #{title}

</when>

<when test="content != null">

and content = #{content}

</when>

<otherwise>

and owner = "owner1"

</otherwise>

</choose>

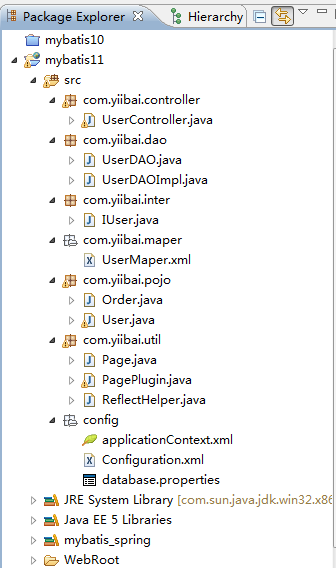
</select>

when元素表示当 when 中的条件满足的时候就输出其中的内容，跟 JAVA 中的 switch 效果差不多的是按照条件的顺序，当 when 中有条件满足的时候，就会跳出 choose，即所有的 when 和 otherwise 条件中，只有一个会输出，当所有的我很条件都不满足的时候就输出 otherwise 中的内容。所以上述语句的意思非常简单， 当 title!=null 的时候就输出 and titlte = #{title}，不再往下判断条件，当title为空且 content!=null 的时候就输出 and content = #{content}，当所有条件都不满足的时候就输出 otherwise 中的内容。

## MyBatis SqlSessionDaoSupport实例

在前面的章节中，我们已经讲到了基本的 mybatis 操作，但都是基于 mapper 隐射操作的，在 mybatis3 中这个 mapper 接口貌似充当了以前在ibatis 2中的 DAO 层的作用。但事实上，如果有这个mapper接口不能完成的工作，或者需要更复杂的扩展的时候，我们就需要自己写 DAO 层。 mybatis 3 也是支持 DAO 层设计的，类似于 ibatis2。下面我们结合一个实例来介绍。

### 首页我们创建一个工程为：mybatis11，再创建一个 com.yihaomen.dao 包，然后在里面分别创建接口UserDAO，并实现该接口 UserDAOImpl 。整个工程的目录结构如下：



UserDao接口的代码如下：

package com.yiibai.dao;

import java.util.List;

import com.yiibai.pojo.Order;

public interface UserDAO {

public List<Order> getUserOrders(int userId);

}

UserDaoImpl 实现UserDao接口的代码如下：

package com.yiibai.dao;

import java.util.List;

import org.mybatis.spring.support.SqlSessionDaoSupport;

import org.springframework.stereotype.Repository;

import com.yiibai.pojo.Order;

@Repository

public class UserDAOImpl extends SqlSessionDaoSupport implements UserDAO {

public List<Order> getUserOrders(int userId) {

// TODO Auto-generated method stub

return this.getSqlSession().selectList("com.yiibai.inter.IUser.getUserOrders",userId);

}

}

控制类 UserController.java 的代码如下：

package com.yiibai.controller;

import java.util.List;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

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

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

import com.yiibai.dao.UserDAO;

import com.yiibai.pojo.Order;

import com.yiibai.util.Page;

// http://localhost:8080/mybatis08-paging/user/orders

@Controller

@RequestMapping("/user")

public class UserController {

//UserMaper userMaper;

@Autowired

UserDAO userDao;

/\*\*

\* 某一个用户下的所有订单（Dao方式）

\* URL => http://localhost:8080/mybatis11/user/lists

\*

\* @param request

\* @param response

\* @return

\*/

@RequestMapping("/lists")

public ModelAndView listalldao(HttpServletRequest request,

HttpServletResponse response) {

List<Order> orders = this.userDao.getUserOrders(1);

// 制定视图 =>list.jsp

ModelAndView mav = new ModelAndView("lists");

mav.addObject("orders", orders);

return mav;

}

}

下一步，我们运行检验结果（根据用户ID，读取这个用户的所有订单），打开浏览器，输入网址：http://localhost:8080/mybatis11/user/lists ，结果如下图所示：



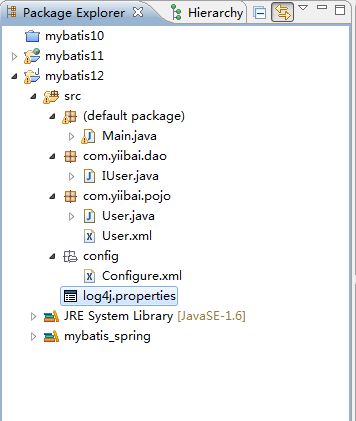
工程 mybatis11 的代码下载：<http://pan.baidu.com/s/1sjrEL9V>

Jar 包下载：<http://pan.baidu.com/s/1bnyRJ9H>

## MyBatis打印输出SQL语句

Hibernate是可以配置 show\_sql 显示 自动生成的SQL 语句，用 format\_sql 可以格式化SQL 语句，但如果用 mybatis 怎么实现这个功能呢？如果你搜索看一下，基本都是通过配置日志来实现的，比如配置我们最常用的 log4j.properties 来实现。

首页我们创建一个 java 工程叫作：mybatis12，内容与之前 Mybatis+Spring 差不多，实现一个通过指定用户ID并读取其订单列表，来观察SQL的执行情况。其工程目录结构如下：



### log4j.properties 内容如下：

# by yiibai.com

log4j.rootLogger=debug,stdout,logfile

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

#log4j.appender.stdout.Target=System.err

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

log4j.appender.logfile=org.apache.log4j.FileAppender

log4j.appender.logfile.File=C:/mybatis\_show\_sql.log

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

log4j.appender.logfile.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %F %p %m%n

log4j.logger.com.ibatis=DEBUG

log4j.logger.com.ibatis.common.jdbc.SimpleDataSource=DEBUG

log4j.logger.com.ibatis.common.jdbc.ScriptRunner=DEBUG

log4j.logger.com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate=DEBUG

log4j.logger.java.sql.Connection=DEBUG

log4j.logger.java.sql.Statement=DEBUG

log4j.logger.java.sql.PreparedStatement=DEBUG

### 主类测试代码如下：

import java.io.Reader;

import java.text.MessageFormat;

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.yiibai.dao.IUser;

import com.yiibai.pojo.User;

public class Main {

private static SqlSessionFactory sqlSessionFactory;

private static Reader reader;

static {

try {

reader = Resources.getResourceAsReader("config/Configure.xml");

sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);

} catch (Exception e) {

e.printStackTrace();

}

}

public static SqlSessionFactory getSession() {

return sqlSessionFactory;

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// TODO Auto-generated method stub

SqlSession session = sqlSessionFactory.openSession();

try {

//User user = (User) session.selectOne(

// "com.yiibai.mybatis.models.UserMapper.getUserByID", 1);

IUser iuser = session.getMapper(IUser.class);

getUserList();

//testInsert();

testUpdate();

//testDelete();

} finally {

session.close();

}

}

//

public static void testInsert()

{

try

{

SqlSession session = sqlSessionFactory.openSession();

IUser userMapper = session.getMapper(IUser.class);

System.out.println("Test insert start...");

User user = new User();

user.setId(0);

user.setName("Google");

user.setDept("Tech");

user.setWebsite("http://www.google.com");

user.setPhone("120");

userMapper.insertUser(user);

session.commit();

System.out.println("\r\nAfter insert");

getUserList();

System.out.println("Test insert finished...");

}

catch (Exception e)

{

e.printStackTrace();

}

}

// 用户列表

public static void getUserList(){

try

{

SqlSession session = sqlSessionFactory.openSession();

IUser iuser = session.getMapper(IUser.class);

System.out.println("Test Get start...");

printUsers(iuser.getUserList());

System.out.println("Test Get finished...");

}catch (Exception e)

{

e.printStackTrace();

}

}

public static void testUpdate()

{

try

{

SqlSession session = sqlSessionFactory.openSession();

IUser iuser = session.getMapper(IUser.class);

System.out.println("Test update start...");

printUsers(iuser.getUserList());

User user = iuser.getUser(1);

user.setName("New name");

iuser.updateUser(user);

session.commit();

System.out.println("\r\nAfter update");

printUsers(iuser.getUserList());

System.out.println("Test update finished...");

}catch (Exception e)

{

e.printStackTrace();

}

}

public static void testDelete()

{

try

{

SqlSession session = sqlSessionFactory.openSession();

IUser iuser = session.getMapper(IUser.class);

System.out.println("Test delete start...");

System.out.println("Before delete");

printUsers(iuser.getUserList());

iuser.deleteUser(3);

session.commit();

System.out.println("\r\nAfter delete");

printUsers(iuser.getUserList());

System.out.println("Test delete finished...");

}catch (Exception e)

{

e.printStackTrace();

}

}

/\*\*

\*

\*

\* @param users

\*/

private static void printUsers(final List<User> users)

{

int count = 0;

for (User user : users)

{

System.out.println(MessageFormat.format("============= User[{0}]=================", ++count));

System.out.println("User Id: " + user.getId());

System.out.println("User Name: " + user.getName());

System.out.println("User Dept: " + user.getDept());

System.out.println("User Website: " + user.getWebsite());

}

}

}

### 执行后，在MyEclise终端输出结果如下：

DEBUG - Logging initialized using 'class org.apache.ibatis.logging.commons.JakartaCommonsLoggingImpl' adapter.

DEBUG - PooledDataSource forcefully closed/removed all connections.

DEBUG - PooledDataSource forcefully closed/removed all connections.

DEBUG - PooledDataSource forcefully closed/removed all connections.

DEBUG - PooledDataSource forcefully closed/removed all connections.

Test Get start...

DEBUG - Opening JDBC Connection

DEBUG - Created connection 22927632.

DEBUG - Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection@15dd910]

DEBUG - ooo Using Connection [com.mysql.jdbc.JDBC4Connection@15dd910]

DEBUG - ==> Preparing: select \* from user

DEBUG - ==> Parameters:

DEBUG - <== Total: 2

============= User[1]=================

User Id: 1

User Name: New name

User Dept: Tech

User Website: http://www.yiibai.com

============= User[2]=================

User Id: 2

User Name: Hevi

User Dept: Tech

User Website: http://www.baidu.com

Test Get finished...

Test update start...

DEBUG - Opening JDBC Connection

DEBUG - Created connection 33189144.

DEBUG - Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection@1fa6d18]

DEBUG - ooo Using Connection [com.mysql.jdbc.JDBC4Connection@1fa6d18]

DEBUG - ==> Preparing: select \* from user

DEBUG - ==> Parameters:

DEBUG - <== Total: 2

============= User[1]=================

User Id: 1

User Name: New name

User Dept: Tech

User Website: http://www.yiibai.com

============= User[2]=================

User Id: 2

User Name: Hevi

User Dept: Tech

User Website: http://www.baidu.com

DEBUG - ooo Using Connection [com.mysql.jdbc.JDBC4Connection@1fa6d18]

DEBUG - ==> Preparing: select \* from user where id=?

DEBUG - ==> Parameters: 1(Integer)

DEBUG - <== Total: 1

DEBUG - ooo Using Connection [com.mysql.jdbc.JDBC4Connection@1fa6d18]

DEBUG - ==> Preparing: UPDATE user SET name = ?, dept = ?, website = ?, phone = ? where id = ?

DEBUG - ==> Parameters: New name(String), Tech(String), http://www.yiibai.com(String), 13800009988(String), 1(Integer)

DEBUG - <== Updates: 1

DEBUG - Committing JDBC Connection [com.mysql.jdbc.JDBC4Connection@1fa6d18]

After update

DEBUG - ooo Using Connection [com.mysql.jdbc.JDBC4Connection@1fa6d18]

DEBUG - ==> Preparing: select \* from user

DEBUG - ==> Parameters:

DEBUG - <== Total: 2

============= User[1]=================

User Id: 1

User Name: New name

User Dept: Tech

User Website: http://www.yiibai.com

============= User[2]=================

User Id: 2

User Name: Hevi

User Dept: Tech

User Website: http://www.baidu.com

Test update finished...

代码下载：<http://pan.baidu.com/s/1jGk165o>

Jar 包下载：<http://pan.baidu.com/s/1bnyRJ9H>