# 1框架概述

## 什么是框架

### 什么是框架

框架（Framework）是整个或部分系统的可重用设计，表现为一组抽象构件及构件实例间交互的方法;另一种定义认为，框架是可被应用开发者定制的应用骨架。前者是从应用方面而后者是从目的方面给出的定义。

简而言之，框架其实就是某种应用的半成品，就是一组组件，供你选用完成你自己的系统。简单说就是使用别人搭好的舞台，你来做表演。而且，框架一般是成熟的，不断升级的软件。

简单理解:框架是软件开发中的一套解决方案，不同的框架解决不同的问题。

好处：框架封装了很多细节，使开发者可以使用极简的方式实现功能。大大提高了开发效率。

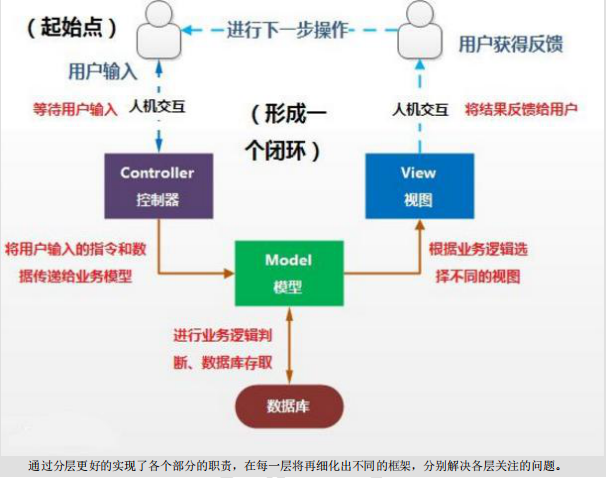
### 框架要解决的问题

框架要解决的最重要的一个问题是技术整合的问题，在 J2EE 的 框架中，有着各种各样的技术，不同的软件企业需要从 J2EE 中选择不同的技术，这就使得软件企业最终的应用依赖于这些技术，技术自身的复杂性和技术的风险性将会直接对应用造成冲击。而应用是软件企业的核心，是竞争力的关键所在，因此应该将应用自身的设计和具体的实现技术解耦。这样，软件企业的研发将集中在应用的设计上，而不是具体的技术实现，技术实现是应用的底层支撑，它不应该直接对应用产生影响。

**框架一般处在低层应用平台（如 J2EE）和高层业务逻辑之间的中间层。**

### 软件开发的分层重要性

框架的重要性在于它实现了部分功能，并且能够很好的将低层应用平台和高层业务逻辑进行了缓和。为了实现软件工程中的“高内聚、低耦合”。把问题划分开来各个解决，易于控制，易于延展，易于分配资源。我们常见的MVC 软件设计思想就是很好的分层思想。



### 分层开发下的常见框架

常见的 JavaEE 开发框架：

**1、解决数据的持久化问题的框架**

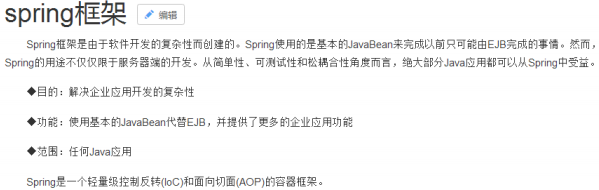


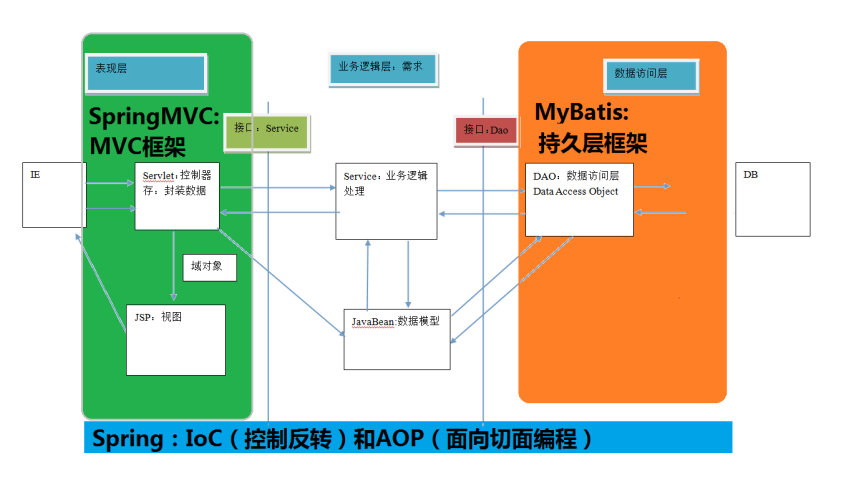
作为持久层的框架，还有一个封装程度更高的框架就是Hibernate，但这个框架因为各种原因目前在国内的流行程度下降太多，现在公司开发也越来越少使用。目前使用 Spring Data 来实现数据持久化也是一种趋势。

**2、解决 WEB 层问题的 MVC 框架**



**3、解决技术整合问题的框架**





### Mybatis框架概述

Mybatis 是一个优秀的基于 java 的持久层框架，它内部封装了 jdbc，使开发者只需要关注 sql 语句本身，而不需要花费精力去处理加载驱动、创建连接、创建 statement 等繁杂的过程。

Mybatis 通过 xml 或注解的方式将要执行的各种 statement 配置起来，并通过 java 对象和 statement 中sql 的动态参数进行映射生成最终执行的 sql 语句，最后由 mybatis 框架执行 sql 并将结果映射为 java 对象并返回。

采用 ORM 思想解决了实体和数据库映射的问题，对 jdbc 进行了封装，屏蔽了 jdbc api 底层访问细节，使我们不用与 jdbc api 打交道，就可以完成对数据库的持久化操作。

为了我们能够更好掌握框架运行的内部过程，并且有更好的体验，下面我们将从自定义 Mybatis 框架开始来学习框架。此时我们将会体验框架从无到有的过程体验，也能够很好的综合前面阶段所学的基础。

Mybatis是一个持久层框架，用java编写的。

它封装了jdbc操作的很多细节，使开发者只需要关注sql语句本身，而无需关注注册驱动，创建连接等繁杂过程

它使用了ORM思想实现了结果集的封装。

ORM：Object Relational Mappging 对象关系映射

简单的说：

就是把数据库表和实体类及实体类的属性对应起来。

让我们可以操作实体类就实现操作数据库表。

user User

id userId

user\_name userName

今天我们需要做到：

实体类中的属性和数据库表的字段名称保持一致。

user User

Id id

user\_name user\_name

## JDBC编程的分析

### JDBC编程的回顾

原始的，未经封装的jdbc方法如下：

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

Connection conn = **null**;

PreparedStatement ps = **null**;

ResultSet rs = **null**;

**try** {

// 加载驱动

Class.*forName*("com.mysql.jdbc.Driver");

// 建立连接

conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/mybatis", "root", "root");

//sql 语句创建，带有占位符

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

// 执行sql，获取PrepareStatement对象

ps = conn.prepareStatement(sql);

// 设置参数

ps.setString(1, ");

//执行sql，获取ResultSet对象

rs = ps.executeQuery();

// 处理结果集

**while**(rs.next()) {

**int** id = rs.getInt("id");

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

String sex = rs.getString("sex");

String address = rs.getString("address");

System.***out***.println(id+","+username+","+sex+","+address);

}

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

} **catch** (SQLException e) {

e.printStackTrace();

} **finally** {

// 释放资源

**if**(rs!=**null**) {

**try** {

rs.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**if**(ps!=**null**) {

**try** {

ps.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**if**(conn!=**null**) {

**try** {

conn.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

}

### JDBC编程的问题

上面的JDBC方法存在以下方面的几个问题：

1、数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库链接池可解决此问题。

2、Sql语句在代码中硬编码，造成代码不易维护，实际应用sql变化的可能较大，sql变动需要改变java代码。

3、使用preparedStatement向占有位符号传参数存在硬编码，因为sql语句的where条件不一定，可能多也可能少，修改sql还要修改代码，系统不易维护。

4、对结果集解析存在硬编码（查询列名），sql变化导致解析代码变化，系统不易维护，如果能将数据库记录封装成pojo对象解析比较方便。

# 2 Mybatis框架入门案例

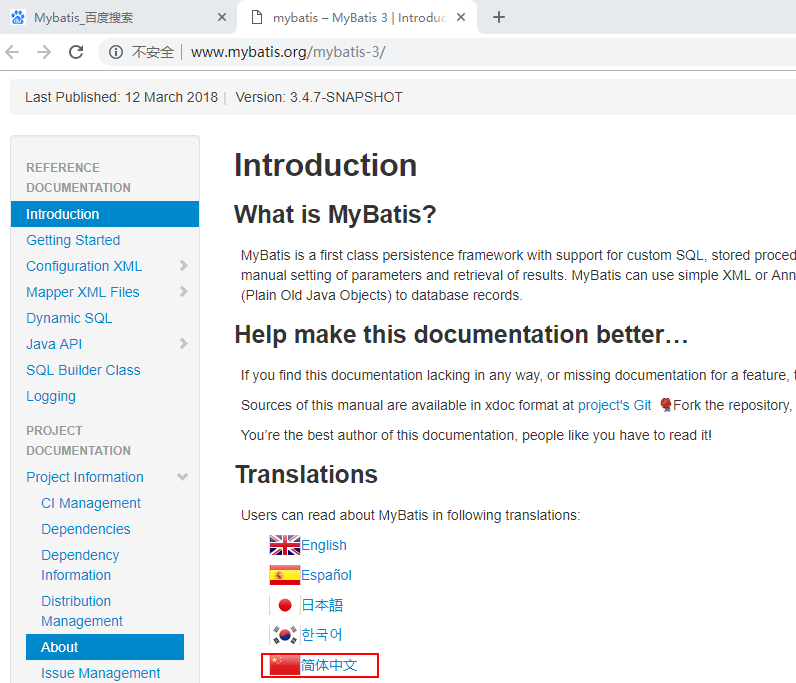
## 2.1 Mybatis框架开发的准备

### 2.1.1 下载Mybatis框架

在百度中搜索 Mybatis，出现如下界面：



点击进入：



点击简体中文，进入中文版本的开发文档：



点击入门，可以下载相关的jar包或者查看Mybatis的Mevan坐标：



下载的资料如下：

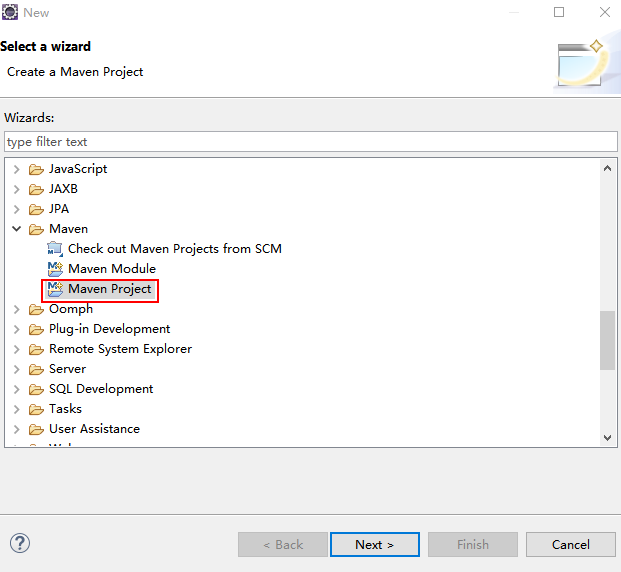


我们下载的版本为3.4.6

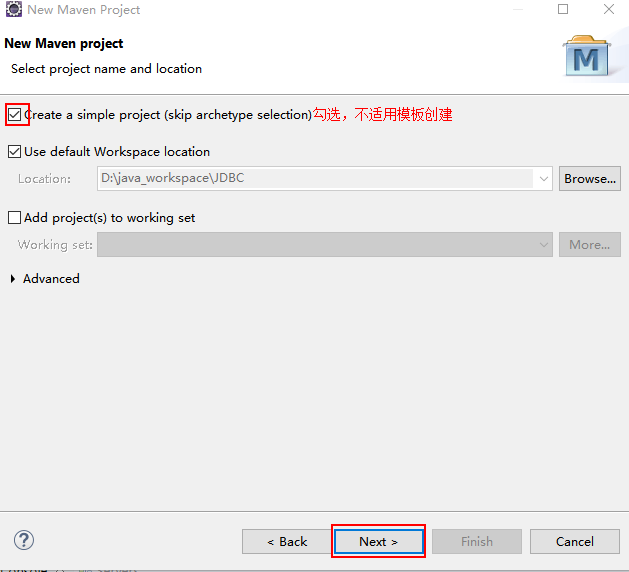
## 2.2 搭建Mybatis开发环境

### 2.2.1 创建Maven工程

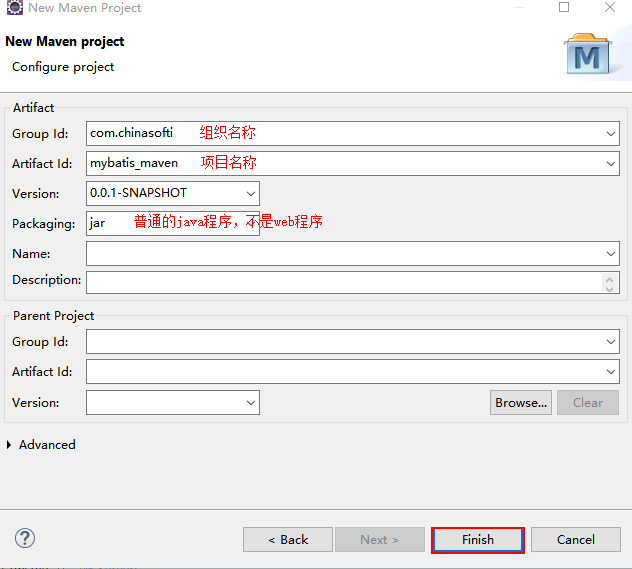
new - other 出现以下选项：



点击Next：



点击next



点击finish，创建完成

### 2.2.2 在pom.xml文件中添加坐标

pom.xml文件中添加坐标如下：其中，前2个mybatis坐标和mysq-connector坐标是必须的

如果想打印日志信息，可以加入log4j坐标

如果想使用单元测试，可以加入junit坐标



### 2.2.3 编写User实体类

在src/main/java下面创建domain包，并在domain包下创建User实体类，如下：

/\*\*

\* 实体类，对应user表

\* 实体类的属性与表中的字段名称相同

\* 实体类实现Serializable接口

\*/

**public** **class** User **implements** Serializable{

**private** Integer id;

**private** String username;

**private** Date birthday;

**private** String sex;

**private** String address;

**public** Integer getId() {

**return** id;

}

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

**this**.id = id;

}

**public** String getUsername() {

**return** username;

}

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

**this**.username = username;

}

**public** Date getBirthday() {

**return** birthday;

}

**public** **void** setBirthday(Date birthday) {

**this**.birthday = birthday;

}

**public** String getSex() {

**return** sex;

}

**public** **void** setSex(String sex) {

**this**.sex = sex;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

@Override

**public** String toString() {

**return** "User [id=" + id + ", username=" + username + ", birthday=" +

           birthday + ", sex=" + sex + ", address=" + address + "]";

}

}

### 2.2.4 编写持久层接口UserDao

在src/main/java下面创建dao包，并在dao包下创建UserDao接口，如下：



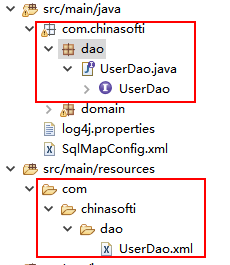
持久层接口可以叫做UserDao，也可以叫UserMapper，这里叫UserDao是为了延续我们以往的编程习惯

### 2.2.5 编写UserDao的映射文件UserDao.xml

在src/main/resources下面创建文件夹com，在com下创建文件夹chinasofti,在chinasofti下创建文件夹dao

注意：是创建三级文件夹，不是包

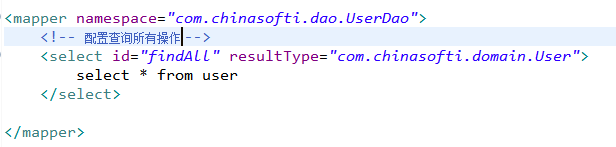
结果如下：



在dao文件夹下创建UserDao的映射文件UserDao.xml,并导入约束



在文件中添加映射:



namespace:UserDao的全限定名，如果是GoodsDao则要写GoodsDao的全限定名

<select>:查询操作

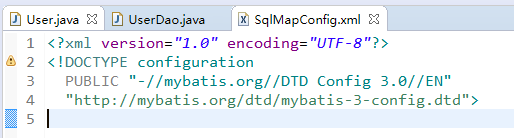
id:对应UserDao接口中的方法名

resultType:结果类型，告诉Mybatis将查询的结果封装在User类中，如果不指定resultType，则执行失败

### 2.2.6 编写SqlMapConfig.xml主配置文件

要想使用Mybatis，必须编写Mybatis主配置文件，一般文件名为SqlMapConfig.xml

在src/main/java下创建SqlMapConfig.xml文件，并导入约束



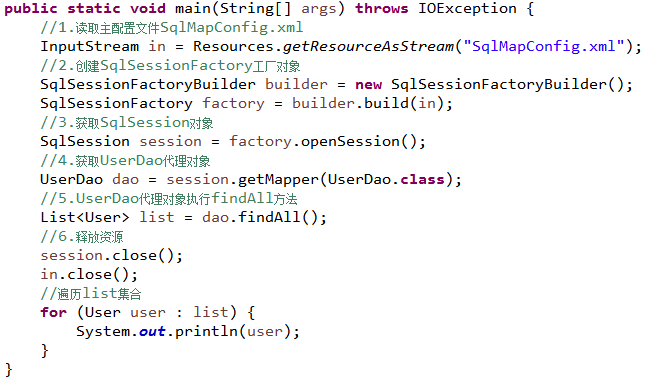
接着需要在文件中添加一些配置：



### 2.2.7 编写测试类

在src/test/java下创建test包，并创建MybatisTest测试类：

在测试类中编写代码运行程序



接下来就可以运行程序了，如果需要查看日志信息，将log4j.properties配置文件放在与SqlMapConfig.xml文件相同的路径下即可

### 2.2.8 环境搭建的注意事项

第一个：创建UserDao.xml 和 UserDao.java时名称是为了和我们之前的知识保持一致。

在Mybatis中它把持久层的操作接口名称和映射文件也叫做：Mapper

所以：UserDao 和 UserMapper是一样的，以后我们会写为UserMapper

第二个：在eclipse中创建目录的时候，它和包是不一样的

包在创建时：com.icss.dao它是三级结构

目录在创建时：com.itheima.dao是一级目录，必须创建三次才可以

第三个：mybatis的映射配置文件位置必须和dao接口的包结构相同

第四个：映射配置文件的mapper标签namespace属性的取值必须是dao接口的全限定类名

第五个：映射配置文件的操作配置（select），id属性的取值必须是dao接口的方法名

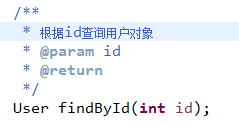
当我们遵从了第三，四，五点之后，我们在开发中就无须再写dao的实现类。

# 3 Mybatis的CRUD操作

## 3.1 根据ID查询

### 3.1.1 在持久层接口中添加方法

在UserDao.java接口中，添加根据id查询的方法，如下：



### 3.1.2 在映射文件中添加配置

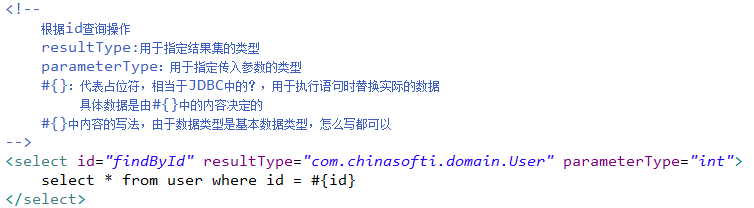
在 UserDao.xml映射配置文件中添加相关配置，如下：

由于在jdbc中，sql语句为 select \* from user where id=?

这里需要占位符，保证安全性。但是在mysql中的sql语句中，是没有？占位符 的，我们用#{}来表示占位符。

同时，在select标签内还需要指定传入的id参数的类型，以及结果类型。

代码如下：



### 3.1.3 在测试类中添加单元测试

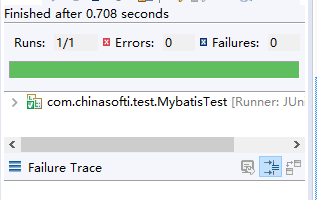
对于之前的使用main方法作为入口来运行程序，我们现在可以改为通过单元测试来测试方法的执行

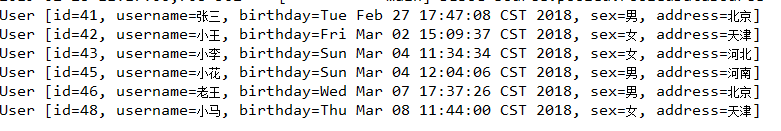
不需要写main方法，只需将main方法改为测试方法即可，如下：



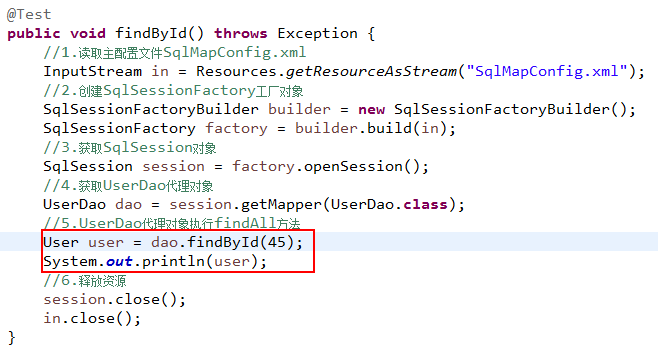
只需将方法声明为上面findAll() 这种形式，并且为方法加上@Test注解即可

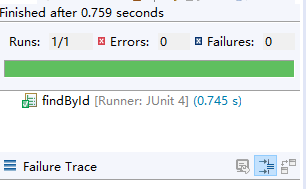
双击方法名--右键--Run As - Junit Test，如果出现如下画面，代表运行成功

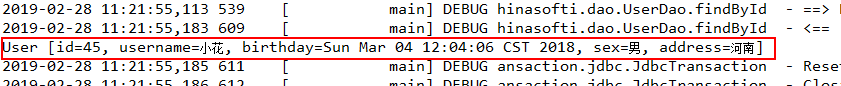




继续添加根据id查询的方法：







我们会发现一个问题，在测试类中，不管是哪一个方法，在执行的过程中，

1.步骤1,2,3,4都是必须要先执行的

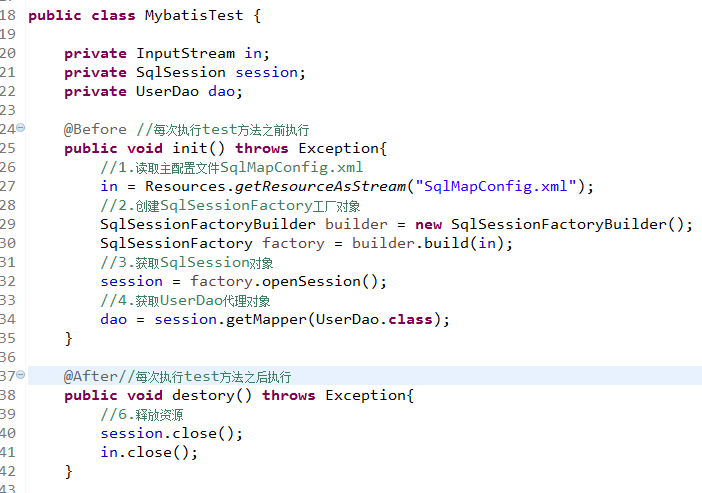
2.而步骤6必须是要后执行的，并且代码都是重复的

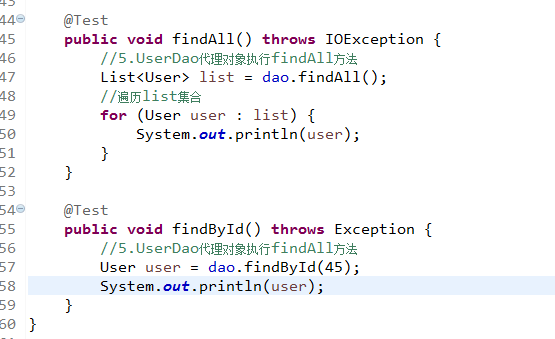
因此，我们把1,2,3,4抽取出来，在测试方法之前执行，步骤6抽取出来，在测试方法之后执行，在测试方法中只需要保留dao调用方法即可





代码修改如下：

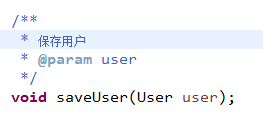




## 3.2 保存用户

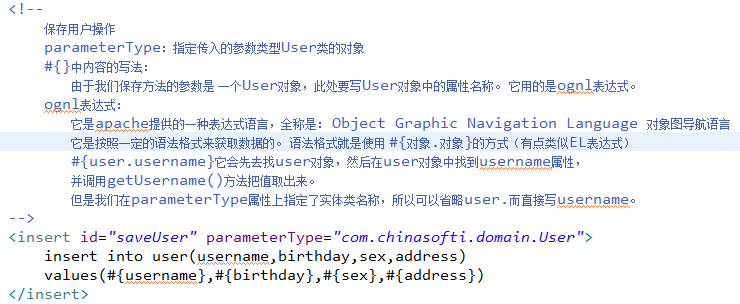
### 3.2.1 在持久层接口中添加方法

在UserDao.java接口中添加保存用户的方法，如下:



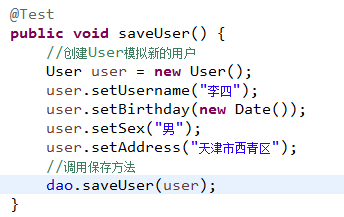
### 3.2.2 在映射文件中添加配置

在UserDao.xml中添加配置：

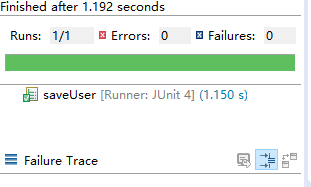


### 3.2.3 在测试类中添加测试

在测试类中添加测试方法：

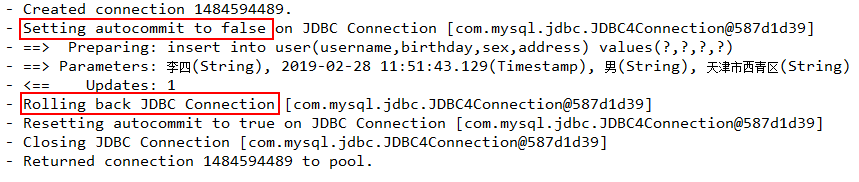


运行程序

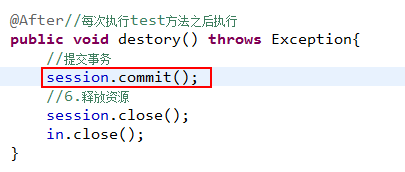


代码没有问题，但是我们检查user表，发现user表中没有插入李四的信息，这是什么原因呢？

通过查看日志信息，发现以下内容：



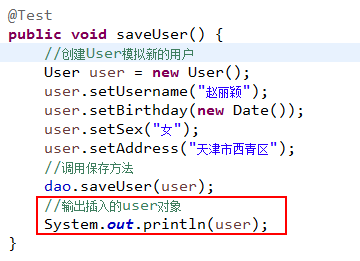
事务默认是不会自动提交的，当执行insert语句后，确实在表中插入了一条记录，但是马上又回滚了事务，所以在表中我们看不到插入的数据，因此在insert语句执行结束以后，我们可以手动提交事务。修改代码如下；



再次执行，插入成功！

### 3.2.4 获取插入用户的id

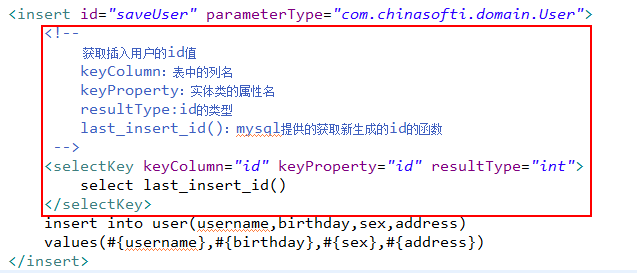
再次执行插入操作，在执行结束以后，输入该用户的信息:



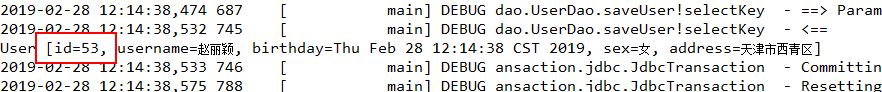
结果如下：



我们发现user对象的id为null，但是表中的id是由数据库的自动增长来实现的，表中是有id的，能不能将自动增长的id值返回呢，存入user对象中？只需加上返回id的代码



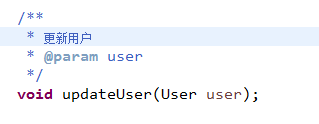
再次运行，id就有值了：



## 3.3 更新用户

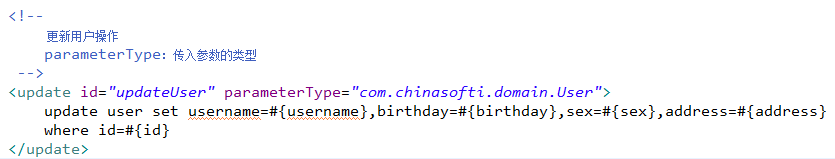
### 3.3.1 在持久层接口中添加方法

在UserDao.java接口中添加方法：



### 3.3.2 在映射文件中添加配置

在UserDao.xml中添加配置信息：



### 3.3.3 在测试类中添加测试

在测试类中添加测试代码：

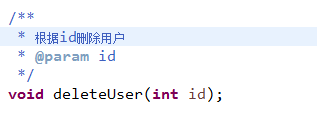


运行程序

## 3.4 删除用户

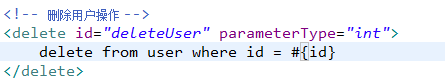
### 3.4.1 在持久层接口中添加方法

在UserDao.java接口中添加方法：



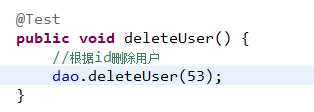
### 3.4.2 在映射文件中添加配置

在UserDao.xml中添加配置：



### 3.4.3 在测试类中添加测试

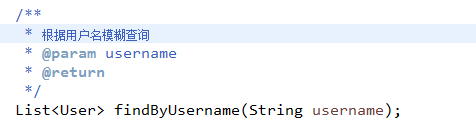
在测试类中添加测试方法：



## 3.5 用户模糊查询

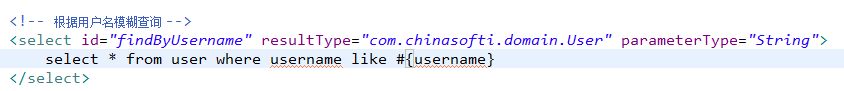
### 3.5.1 在持久层接口中添加方法

在UserDao.java接口中添加方法



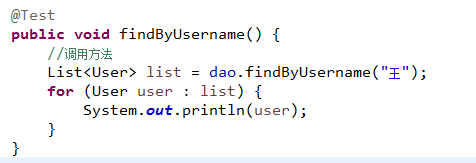
### 3.5.2 在映射文件中添加配置

在UserDao.xml中添加配置：

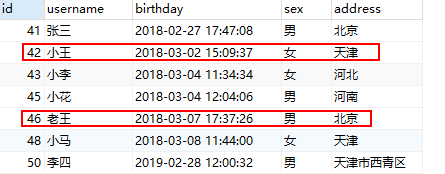


### 3.5.3 在测试类中添加方法

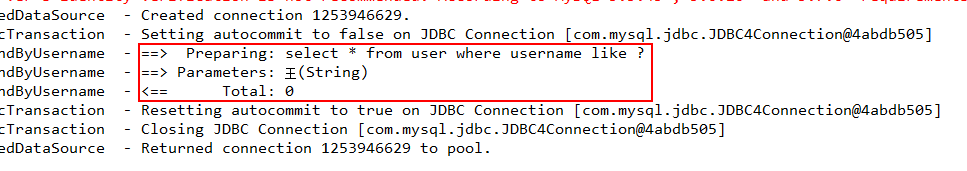
在测试类中添加方法：



运行后发现从表中查询0条记录，检查表发现用户名中带王的有2个：



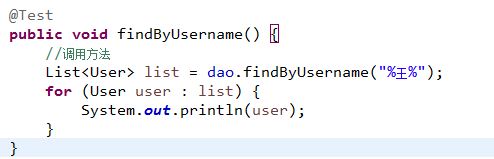
为什么没有查出来呢，查看日志信息：

sql语句为 select \* from user where username like ?

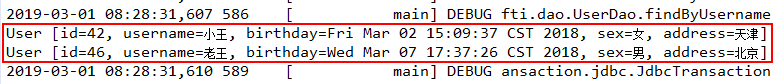
传入参数为"王",那么拼接后的sql语句为：select \* from user where username like '王'

这条语句查找的为username恰好等于王的用户，而不是username中包含王的用户

如果要查找username中包含王的用户，sql语句应该为select \* from user where username like '%王%'，所以传入的参数应该变为 “%王%”



执行结果为：



我们在配置文件中没有加入%来作为模糊查询的条件，所以在传入字符串实参时，就需要给定模糊查询的标识%。配置文件中的#{username}也只是一个占位符，所以SQL语句显示为"？"

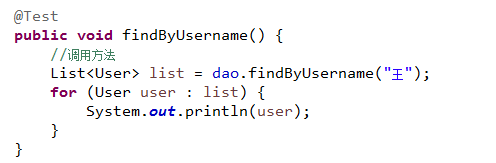
### 3.5.4 模糊查询的另一种配置方式

在UserDao.xml中修改配置信息，如下：



### 3.5.5 修改测试类

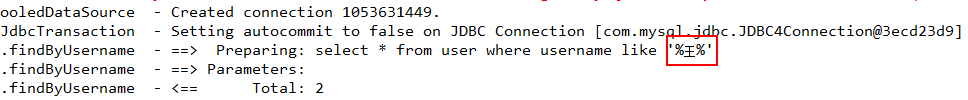
修改测试类代码：



执行结果：



查看sql语句：



可以发现，我们在程序代码中就不需要加入模糊查询的匹配符%了，这两种方式的实现效果是一样的，但执行的语句是不一样的。

### 3.5.6 #{}和${}的区别

**#{}表示一个占位符号**

通过#{}可以实现preparedStatement向占位符中设置值，自动进行java类型和jdbc类型转换，#{}可以有效防止sql注入。

#{}可以接收简单类型值或pojo属性值。 如果parameterType传输单个简单类型值，#{}括号中可以是value或其它名称。

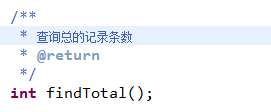
**${}表示拼接sql串**

通过${}可以将parameterType 传入的内容拼接在sql中且不进行jdbc类型转换， ${}可以接收简单类型值或pojo属性值，如果parameterType传输单个简单类型值，${}括号中只能是value。

## 3.6 使用聚合函数查询

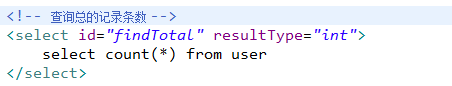
### 3.6.1 在持久层接口中添加方法

在UserDao.java接口中添加方法



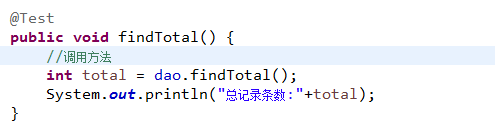
### 3.6.2 在映射文件中添加设置

在UserDao.xml中添加配置信息：

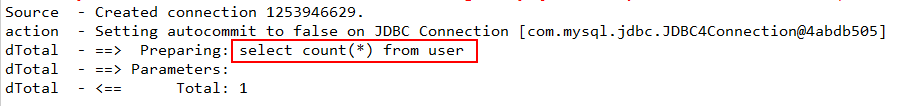


### 3.6.3 在测试类中添加方法

在测试类中添加测试方法：



sql语句：



执行结果：



## 3.7 Mybatis与JDBC编程比较

### 3.7.1 Mybatis与JDBC比较

1.数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库链接池可解决此问题。

解决： 在SqlMapConfig.xml中配置数据链接池，使用连接池管理数据库链接。

2.Sql语句写在代码中造成代码不易维护，实际应用sql变化的可能较大，sql变动需要改变java代码。

解决： 将Sql语句配置在XXXXmapper.xml(XXXXDao.xml)文件中与java代码分离。

3.向sql语句传参数麻烦，因为sql语句的where条件不一定，可能多也可能少，占位符需要和参数对应。

解决： Mybatis自动将java对象映射至sql语句，通过statement中的parameterType定义输入参数的类型。

4.对结果集解析麻烦，sql变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成pojo对象解析比较方便。

解决： Mybatis自动将sql执行结果映射至java对象，通过statement中的resultType定义输出结果的类型。

# 4 Mybatis的参数

## 4.1 parameterType（输入参数）

### 4.1.1 使用说明

前面已经介绍了 SQL 语句传参，使用标签的 parameterType 属性来设定。该属性的取值可以是基本类型，引用类型（例如:String 类型），还可以是实体类类型（POJO 类）。同时也可以使用实体类的包装类，本章节将介绍如何使用实体类的包装类作为参数传递。

### 4.1.2 注意事项

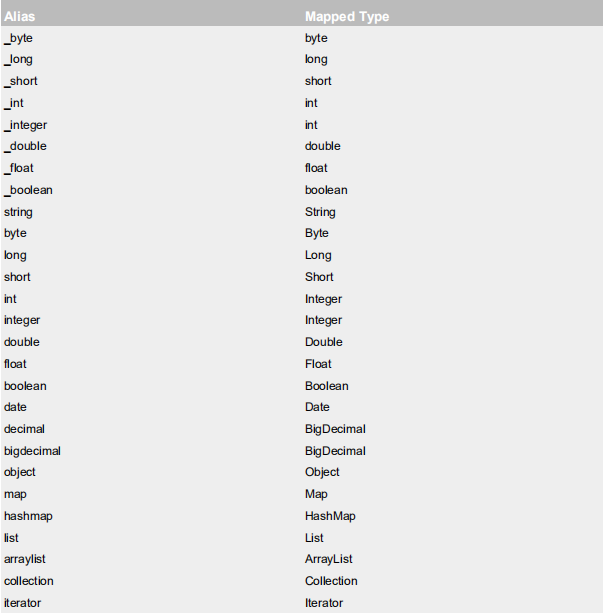
基本类型和String我们可以直接写类型名称，也可以使用包名.类名的方式，例如：java.lang.String。

实体类类型，目前我们只能使用全限定类名。

究其原因，是 mybaits 在加载时已经把常用的数据类型注册了别名，从而我们在使用时可以不写包名，

而我们的是实体类并没有注册别名，所以必须写全限定类名。后面将讲解如何注册实体类的别名。

下面是Mybatis支持的默认别名：



### 4.1.3 输入参数简单数据类型

前面的CRUD操作中，输入参数为int类型，输入参数为String类型，都有例子，这里不再举例。

### 4.1.4 输入参数为pojo对象

pojo对象就是我们封装的实体类对象，前面CRUD的例子中也有输入参数为User对象的情况，这里不再举例。

### 4.1.5 输入参数为pojo包装对象

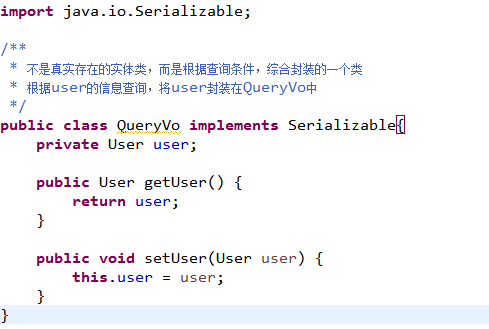
开发中通过 pojo 传递查询条件 ，查询条件是综合的查询条件，不仅包括用户查询条件还包括其它的查

询条件（比如将用户购买商品信息也作为查询条件），这时可以使用包装对象传递输入参数。

Pojo 类中包含 pojo。

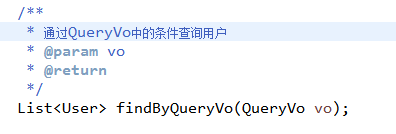
需求：根据用户名查询用户信息，查询条件放到 QueryVo 的 user 属性中。

#### 4.1.5.1 编写QueryVo



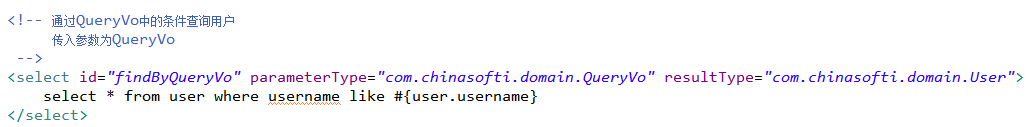
#### 4.1.5.2 在持久层接口添加方法

在持久层接口UserDao.java中添加方法：



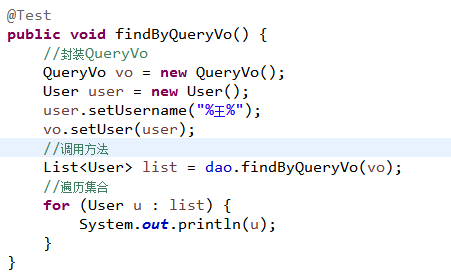
#### 4.1.5.3 在映射文件中添加配置

在UserDao.xml映射文件中添加配置：



#### 4.1.5.4 在测试类中添加测试方法

在测试类中添加测试方法：



运行结果：



# 5 Mybatis的输出结果

## 5.1 resultType（输出类型）

### 5.1.1 输出类型为简单类型

前面的CRUD例子中，输出结果为int类型已经写过，这里不再举例。

### 5.1.2 输出类型为实体类类型

前面的CRUD例子中，已经有输出类型的User类型的例子，这里不再举例。

### 5.1.3 实体类属性名与表列名不相同

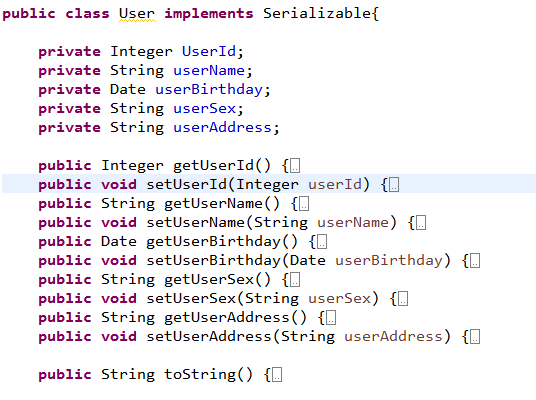
前面我们在进行操作的时候，实体类的属性名和表的列名是相同的，指定resultType以后，mybatis会自动封装。如果实体类的属性名和表列名不相同时，会怎么样呢？

#### 5.1.3.1 修改实体类属性名称

下面是表中的列名：

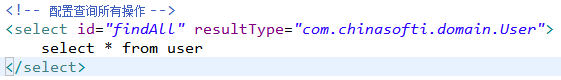


修改User类：

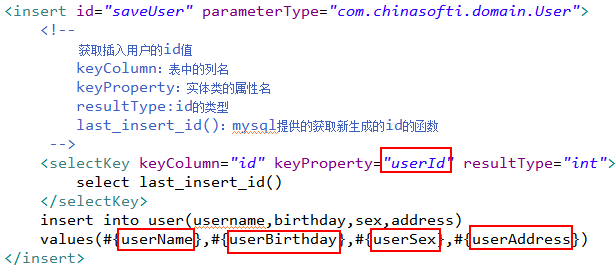


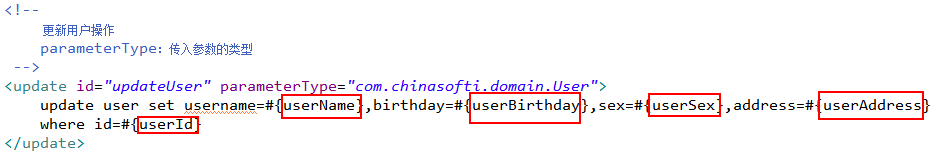
#### 5.1.3.2 修改映射配置文件

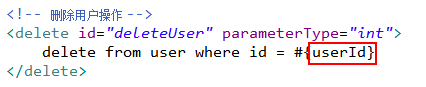
将映射配置文件sql语句中使用user属性名的地方修改，如下：

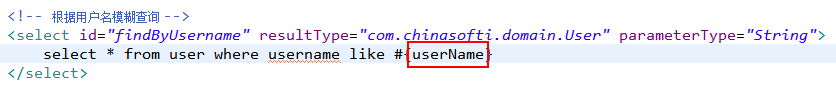


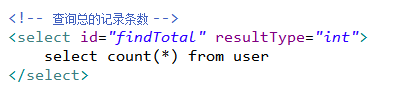


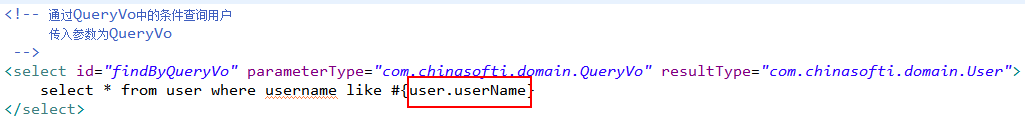








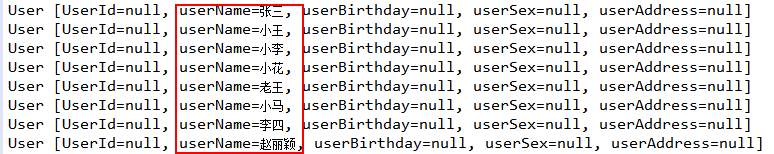




#### 5.1.3.3 重新测试

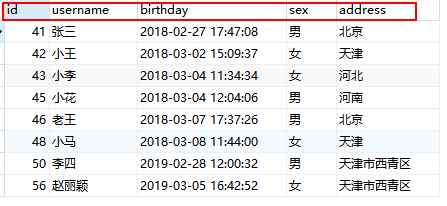
通过测试，我们发现一个问题，insert，update和delete操作都可以正常执行，结果也是正常的。

但是所有的查询操作中，需要封装User实体类的方法中，结果都是有问题的，比如findAll方法，获取所有用户列表，结果如下：



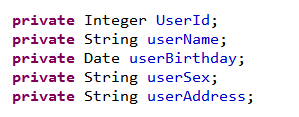
在所有的结果中，只有userName是封装成功的，而userId，userBirthday，userSex，userAddress都是null值，说明并没有将查询出的值封装到user对象中。我们对比一下：

通过执行select \* from user; 结果如下：



这个结果并不是真实存在的表，而是一张存在内存中的虚拟表。

通过该虚拟表的列名与User类型的属性名对应封装User对象，而User对象的属性名如下：



**我们会发现，所有的属性名和列名都是不相同的，为什么userName会有值呢？**

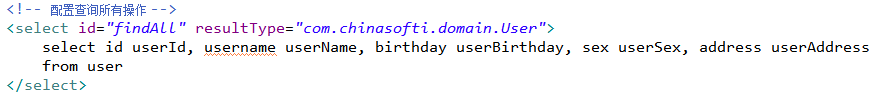
**因为mysql在windows系统中列名是不区分大小写的。**

**其他的值有什么方法能够封装成功呢？刚才我们说了，封装的时候是通过虚拟表的列名与实体类的属性名对象的。**

**所以我们只要在查找的时候，为列名起别名就可以了。**

#### 5.1.3.4 通过别名解决属性名与列名不匹配问题

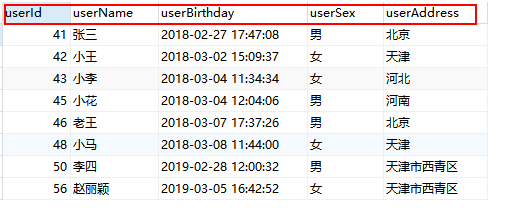
修改映射配置文件中的sql语句，为查询列名起别名：

其他方法一样，这里不再展示。

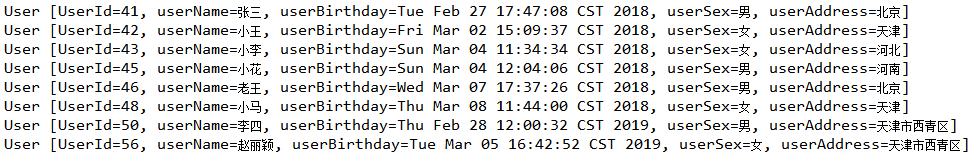
当我们执行查询语句

select id userId, username userName, birthday userBirthday, sex userSex, address userAddress from user

结果为：



测试代码，结果为：



所有内容都正确封装了。

## 5.2 resultMap结果类型

前面我们讲了如果实体类的属性名和表列名不相同时，需要在查询时为列起别名来保证封装结果的准确性。

我们也发现一个问题，每一条查询几乎都要写起别名这段话，重复率是比较高的。

还有一种方式，通过resultMap建立属性名和列表不一致时的对应关系，从而实现封装。

### 5.2.1 定义resultMap

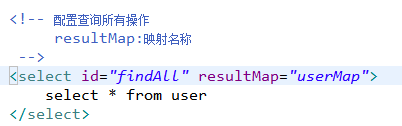
resultMap 标签可以建立查询的列名和实体类的属性名称不一致时建立对应关系。从而实现封装。

在 select 标签中使用 resultMap 属性指定引用即可。同时 resultMap 可以实现将查询结果映射为复杂类型的 pojo，比如在查询结果映射对象中包括 pojo 和 list 实现一对一查询和一对多查询。

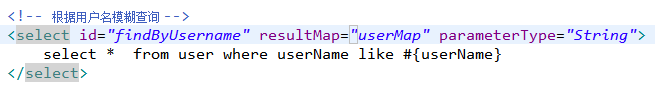


### 5.2.2 配置映射

将resultMap配置到select标签内：





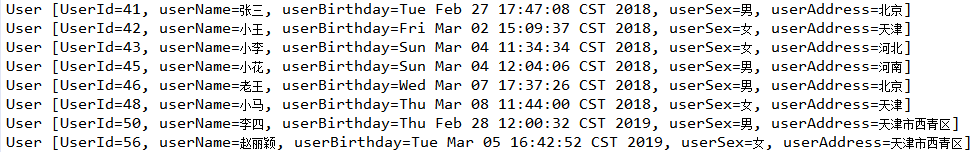




在整个配置文件中，userMap是可以重复使用，这样对开发来说是效率高的。

### 5.2.3 测试结果

测试findAll方法，结果如下：



# 6 Mybatis的传统DAO开发

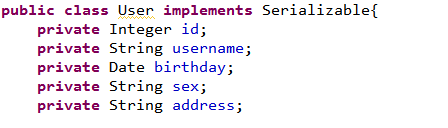
使用 Mybatis 开发 Dao，通常有两个方法，即原始 Dao 开发方式和 Mapper 接口代理开发方式。而现在主流的开发方式是接口代理开发方式，这种方式总体上更加简便。我们主要以接口代理开发方式为主。现在介绍一下基于传统编写 Dao 实现类的开发方式。

## 6.1 复制工程

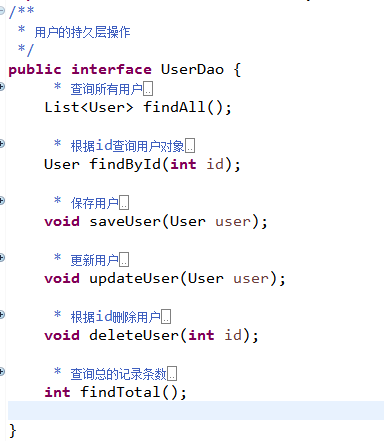
复制工程，保留findAll(查找所有), findById（通过id查找）,saveUser（保存用户）,updateUser（更新用户）,deleteUser（删除用户）,findTotal（查找所有用户的条数）等方法，来演示Mybatis传统DAO开发。

将User实体类的属性改为与表列名相同。

这样比较方便写代码。传统dao开发了解即可，开发中我们还是使用前面讲过的操作。



持久层接口：



## 6.2 创建UserDao的实现类

在src/main/java下创建包com.chinasofti.dao.impl包，并创建UserDaoImpl.java类实现UserDao.java接口

以findAll方法为例，分析改如何实现。

重写findAll方法，在该方法的内部实现查询所有用户功能，并将list集合返回

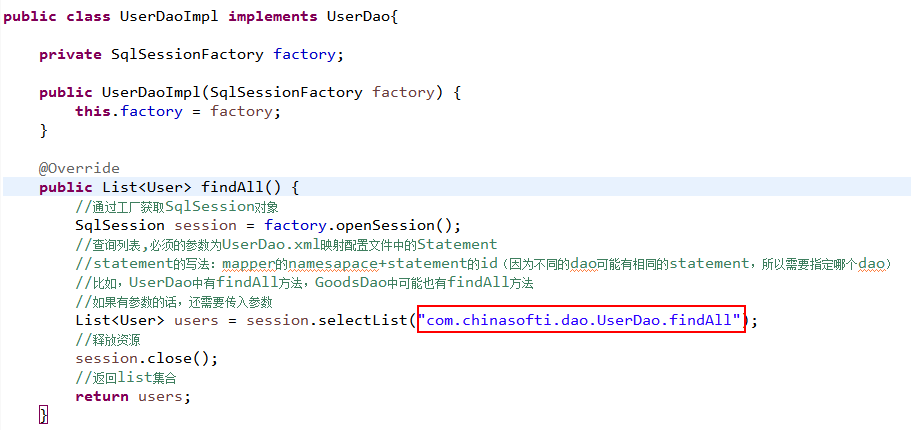
分析：

1.查询操作是通过SqlSession对象来实现的，每一次增删改查的操作应该是一个单独的session，操作结束，接session资源释放

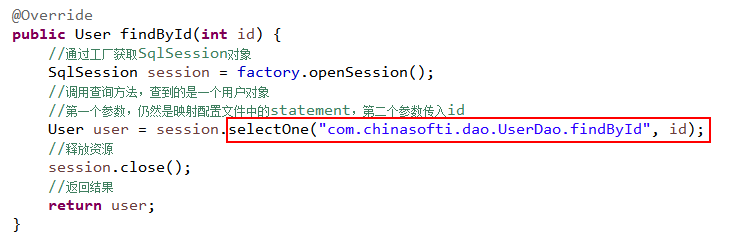
所有在每一个增删改查的方法中都需要一个SqlSession对象，SqlSession是通过SqlSessionFactory工厂对象创建的

2.在每一个增删改查的方法中都需要一个SqlSessionFactory工厂对象吗？不需要，只要保证一个UserDao中有一个工厂对象就可以了。

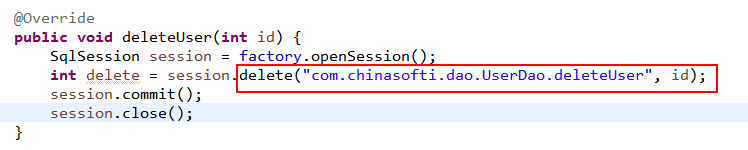
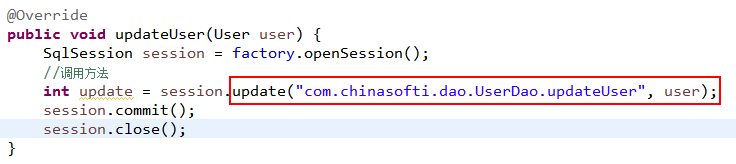
代码如下：



其它方法以此类推：







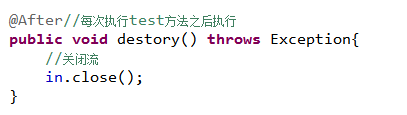
## 6.3 创建测试类

与前面不同的是，获取工厂对象以后，就可以创建UserDao的实现类对象了，由实现类对象去调用方法：

init方法：



修改destroy方法：



创建test方法，测试，结果和之前的一样，这里不再演示。

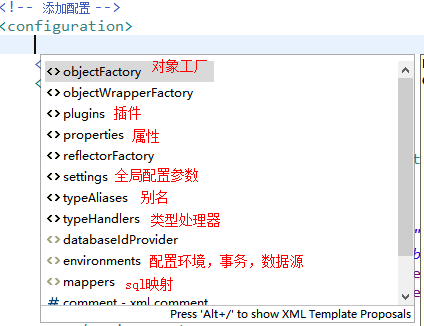
我们发现，这样也是可以实现的。只是比不创建实现类要麻烦。

**我们在实际开发中，都是越简便越好，所以都是采用不写dao实现类的方式。**

# 7 SqlMapConfig.xml配置文件

## 7.1 SqlMapConfig中配置的内容和顺序

SqlMapConfig.xml中可以配置的内容如下（在configuration标签内）：



其中，我们常用的有properties，typeAliases，environments，mappers，接下来一一介绍。

## 7.2 properties(属性)

properties可以对数据源的4个属性进行配置，有2种方式：

**方式一：**

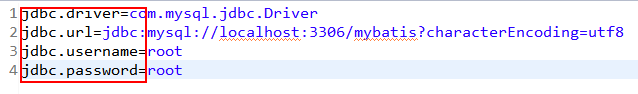
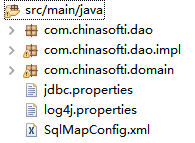


在properties标签内配置属性，在dataSource中获取属性值就可以了。

**方式二：**

数据库的属性发生改变，我们需要修改配置文件。一般我们会把数据库的配置属性写在一个文件中，放在classpath下。

用properties标签读取文件中的属性。写法如下：





properties标签有2个属性，一个是resource，一个是url

这里我们使用的是resource：是一个相对classpath下的路径，所有我们直接写jdbc.properties就可以了

如果使用url的会，必须要满足url的要求才可以。

url：uniform resource locator，统一资源定位符：

http:// localhost: 8080 /day01/servlet1/

协议 主机名 端口号 URI

是web中可以唯一定位一个资源的路径

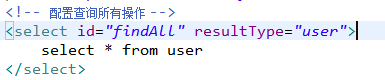


用浏览器打开该文件，然后复制地址栏，粘贴即可。

## 7.3 typeAliases（别名）

在前面我们讲的 Mybatis 支持的默认别名，我们也可以采用自定义别名方式来开发。

这样，在sql映射文件中，就可以修改resultType，获取或者parameterType的值为user（大小写都可以）了





现在有个问题，如果实体类比较多，每一个类都需要设置别名，这样在我们的配置文件中就会写很多的标签



会让配置文件体积变大，还可以通过另一个标签设置别名

只需指定需要设置别名的包即可



## 7.4 mappers(映射器)

mappers标签用于指定sql映射文件：

有3种设置方式：

**方式一 ：**



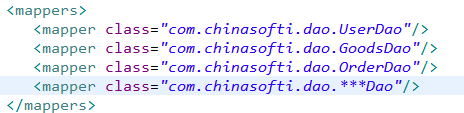
resource： 使用相对于类路径的资源，指定sql映射文件的类路径

**方式二：**

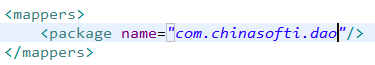


**方式三：**

如果在我们的工程中，持久层接口和映射文件比较多的话，就要写很多的mapper，如下图：



此时，可以通过下面的方式指定映射：



**这种方式要求：持久层接口的名称和sql映射文件的文件名相同，并且路径相同**

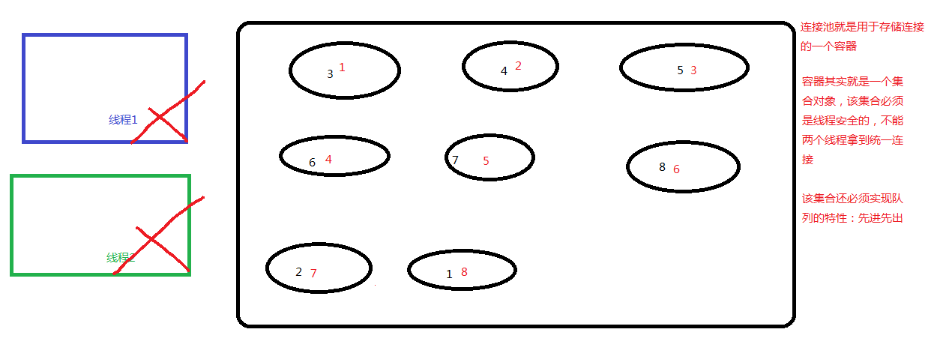
# 8 Mybatis的连接池与事务

## 8.1 Mybatis的连接池技术

我们在前面的 WEB 课程中也学习过类似的连接池技术，而在 Mybatis 中也有连接池技术，但是它采用的是自己的连接池技术。

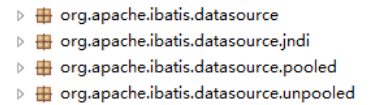
在Mybatis的SqlMapConfig.xml配置文件中，通过<dataSource type=”pooled”>来实现 Mybatis 中连接池的配置。

连接池的原理：



### 8.1.1 连接池的分类

在 Mybatis 中我们将它的数据源 dataSource 分为以下几类：



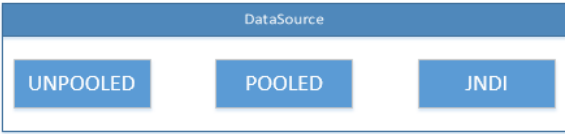
可以看出 Mybatis 将它自己的数据源分为三类：

UNPOOLED 不使用连接池的数据源

POOLED 使用连接池的数据源

JNDI 使用 JNDI 实现的数据源

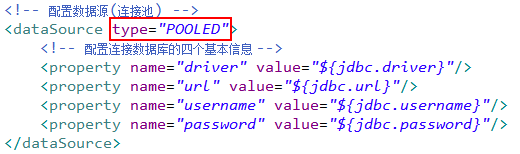
具体结构如下：



相应地，MyBatis内部分别定义了实现了 java.sql.DataSource 接口的 UnpooledDataSource，PooledDataSource 类来表示 UNPOOLED、POOLED 类型的数据源。

### 8.1.2 Mybatis中数据源的配置

我们的数据源配置就是在 SqlMapConfig.xml 文件中，具体配置如下：



POOLED 采用传统的javax.sql.DataSource规范中的连接池，mybatis中有针对规范的实现，创建PooledDataSource实例

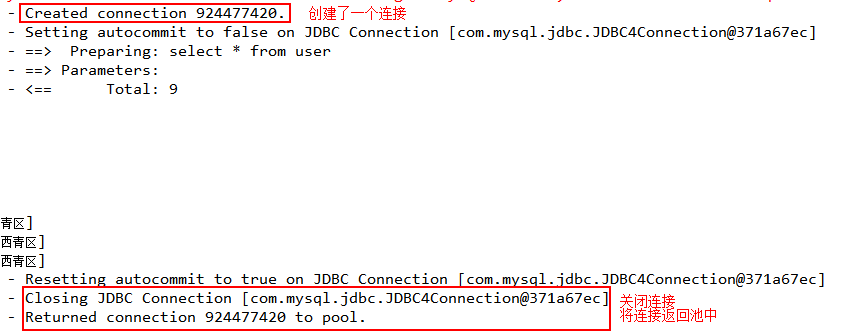
UNPOOLED 采用传统的获取连接的方式，虽然也实现Javax.sql.DataSource接口，但是并没有使用池的思想。

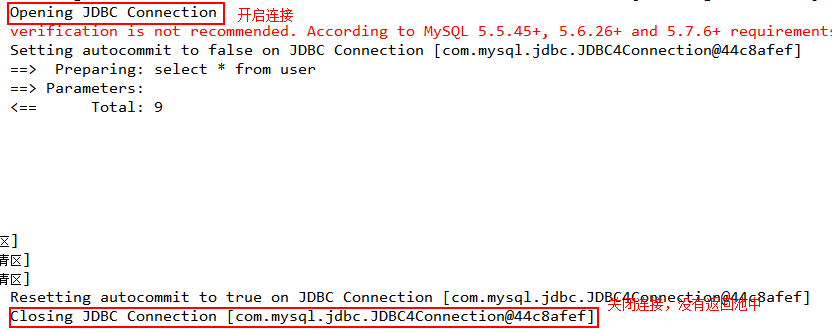
JNDI 采用服务器提供的JNDI技术实现，来获取DataSource对象，不同的服务器所能拿到DataSource是不一样。

注意：如果不是web或者maven的war工程，是不能使用的。

我们使用的是tomcat服务器，采用连接池就是dbcp连接池。

当我们使用POOLED时，测试一个结果：

如果使用UNPOOLED,结果为：



**我们一般采用的是池化的数据源。**

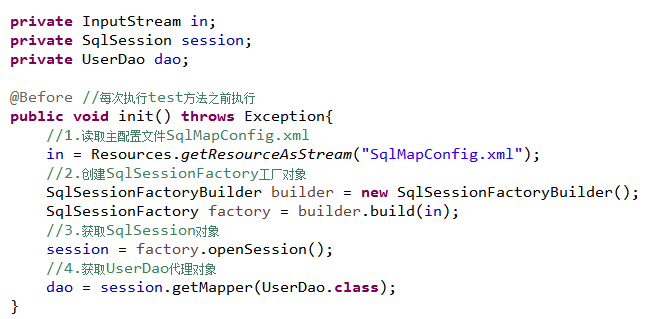
## 8.2 Mybatis的事务控制

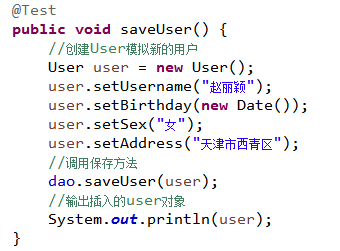
### 8.2.1 Mybatis中事务的提交方式

在 JDBC 中我们可以通过手动方式将事务的提交改为手动方式，通过 setAutoCommit()方法就可以调整。

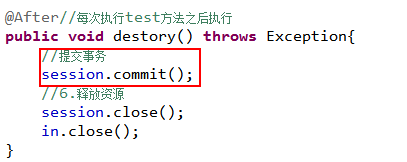
Mybatis 中事务的提交方式，本质上就是调用 JDBC 的 setAutoCommit()来实现事务控制。

看一下我们之前写的增删改的代码，以保存用户为例：

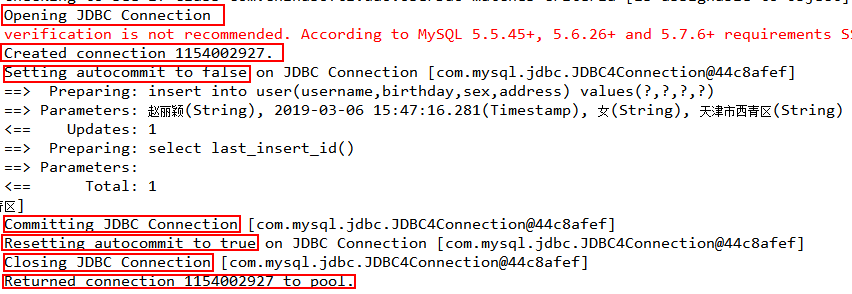




执行完方法之后，需要执行再执行destroy方法：



在该方法中，手动提交事务。看一下执行过程：



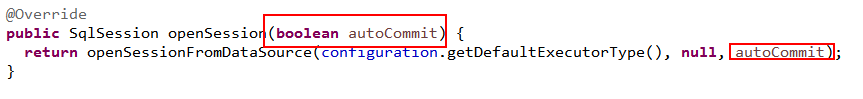
这是Connection的整个变化过程，通过分析我们能够发现之前的 CUD 操作过程中，我们都要手动进行事务的提交，原因是 setAutoCommit()方法，在执行时它的值被设置为 false 了，所以我们在 CUD 操作中，必须通过 sqlSession.commit()方法来执行提交操作。

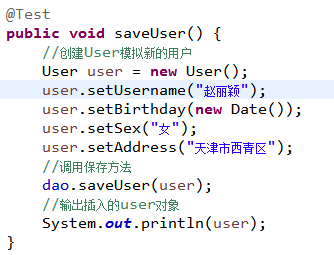
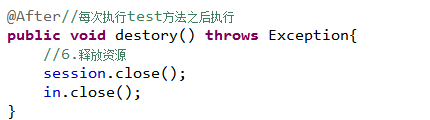
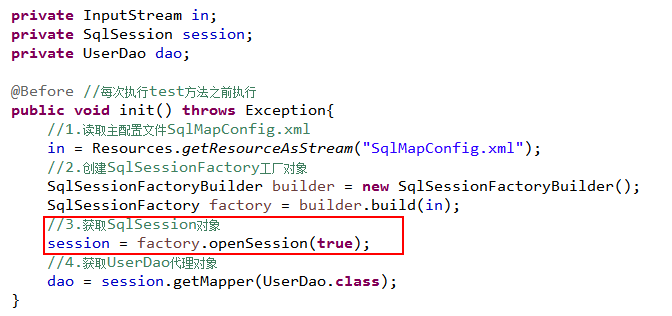
### 8.2.2 Mybatis中自动提交事务

在SqlSession中，openSession方法是重载的，之前我们使用

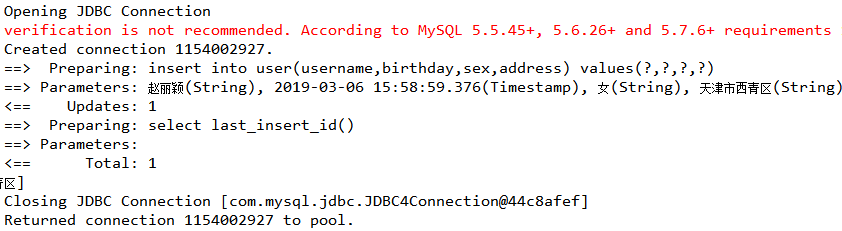
factory.openSession():该方法会自动将autoCommit设置为false，即不会自动提交事务，我们必须自己手动提交事务

factory.openSession(boolean autoCommit):也可以使用该方法，将autoCommit设置为true，这样就会自动提交事务

修改代码如下：



执行结果:



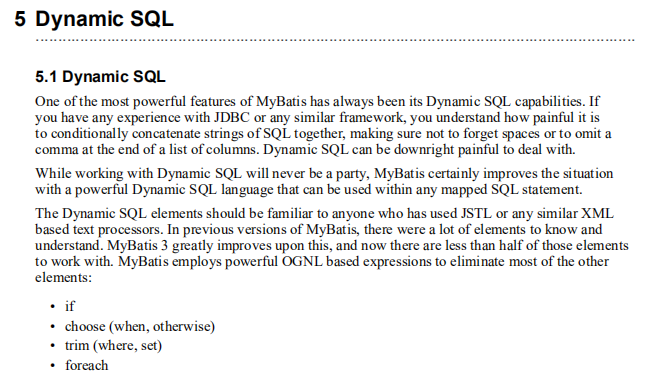


我们发现，此时事务就设置为自动提交了，同样可以实现CUD操作时记录的保存。虽然这也是一种方式，但就编程而言，**设置为自动提交方式为 false 再根据情况决定是否进行提交，这种方式更常用**。因为我们可以根据业务情况来决定提交是否进行提交。

# 9 Mybatis的动态sql

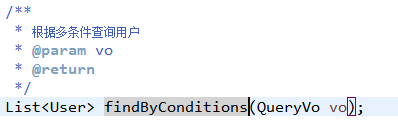
## 9.1 动态sql之if标签

Mybatis 的映射文件中，前面我们的 SQL 都是比较简单的，有些时候业务逻辑复杂时，我们的 SQL 是动态变化的，此时在前面的学习中我们的 SQL 就不能满足要求了。

以多条件查询为例：如果用户名不为空，我们需要根据用户名模糊查询，如果性别不为空，在用户名的基础上，还要加入性别条件，如果地址不为空，在前两个条件的基础上，还要加入地址信息。还要使用QueryVo

### 9.1.1 在持久层接口中添加方法

在持久层接口UserDao.java中添加方法：

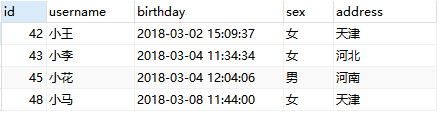


### 9.1.2 在映射文件中添加配置

在UserDao.xml中添加配置：

先测试多条件查询的sql语句：如果不添加任何条件，则查询所有，如果用户名不为null，则根据用户名模糊查询。





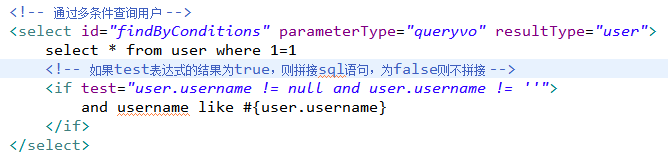
当用户为null时，sql语句为



这时是查询所有用户信息。

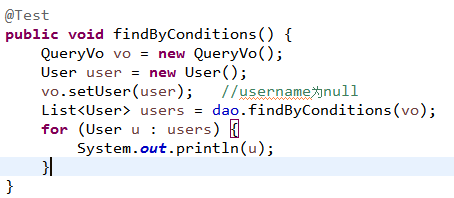
如果用户名不为null，则在sql语句后面拼接 and username like ‘%小%’

根据该思路，我们可以在映射文件中添加配置了：

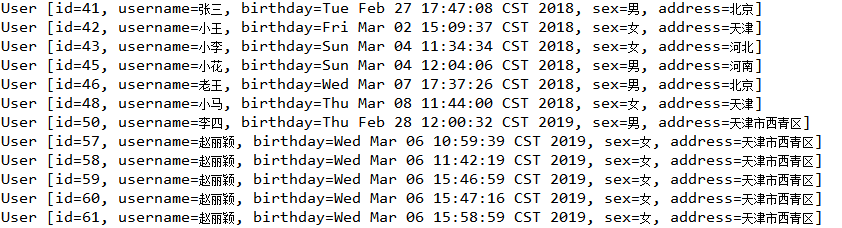


### 9.1.3 在测试类中添加方法

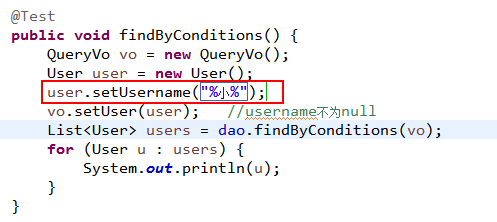
在测试类中添加方法：



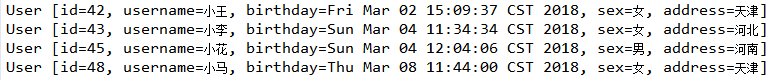
username为null时，查询结果为所有用户：



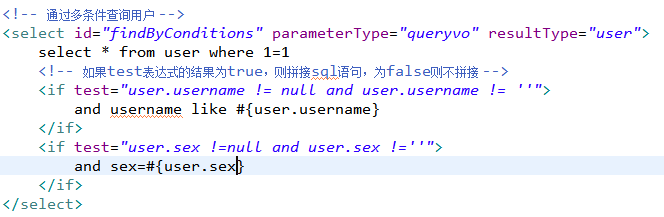
设置username，查询用户名中带小的用户信息：



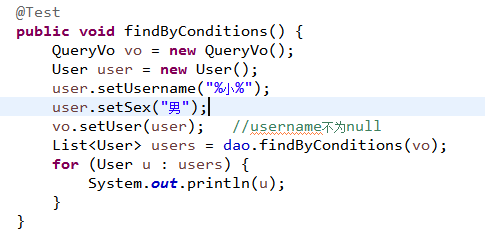
查询结果：



如果再加上性别为女的条件，配置文件修改为：



修改测试方法：



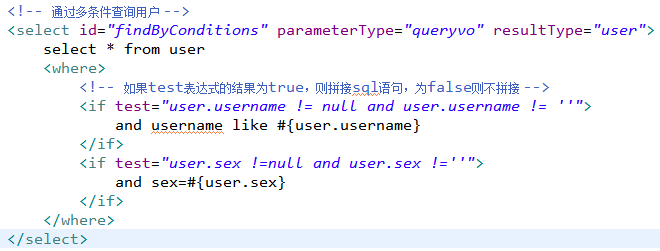
执行结果：



## 9.2 动态sql之where标签

在前面的例子中，我们为sql语句的合法性，必须要加上where 1=1，还有一种方式，可以将1=1 省略，只需加上一个where标签就可以了。

将sql映射文件改为：

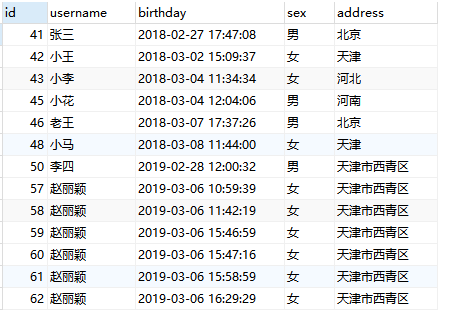


这里where的作用在mybatis帮助文档中，已经说的很清楚。



## 9.3 动态sql之foreach标签

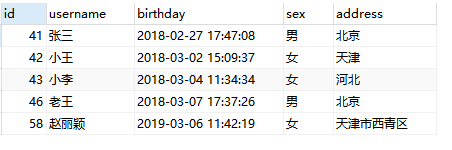
### 9.3.1 需求



查询id为 41,42,43,46,58的用户信息，sql语句如下：



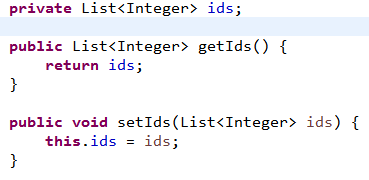
查询结果如下：



如何通过mybatis实现？

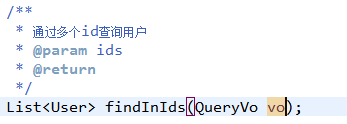
### 9.3.2 修改QueryVo

在QueryVo中添加属性，ids，存储要查询的多个id值



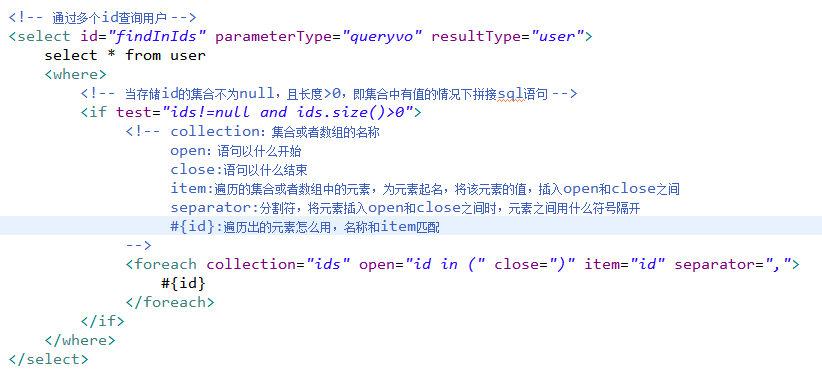
### 9.3.3 在持久层接口中添加方法

在持久层接口UserDao.java中添加方法：



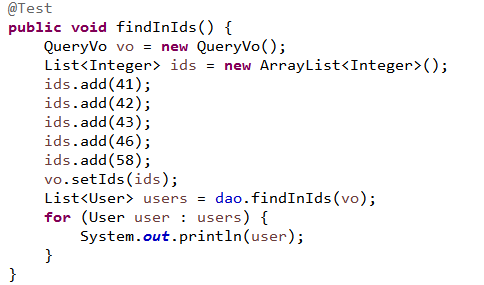
### 9.3.4 在映射文件中添加配置

在映射文件UserDao.xml中添加配置：

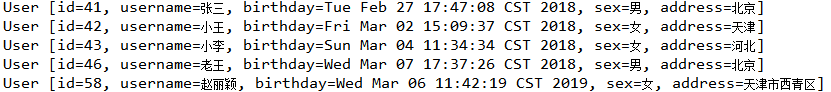


### 9.3.5 在测试类中添加方法

在测试类中添加方法：



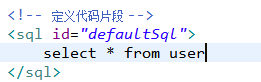
执行结果：



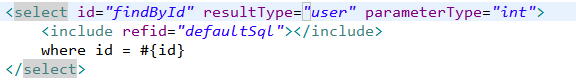
## 9.4 sql片段

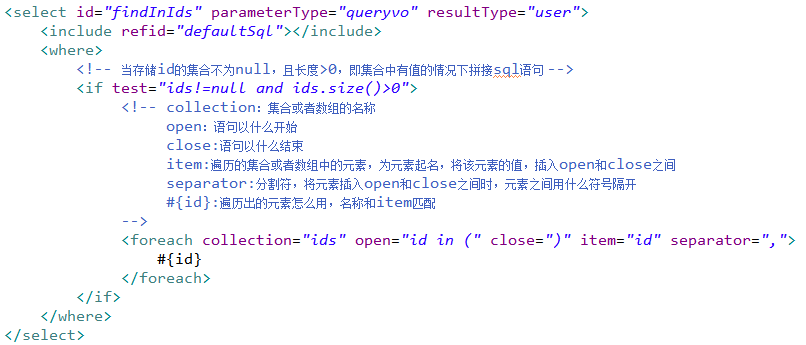
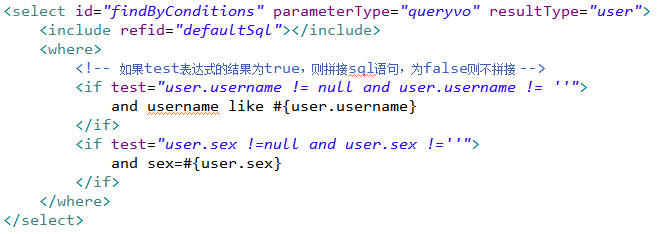
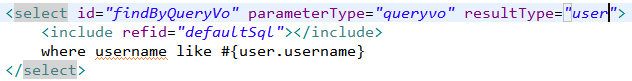
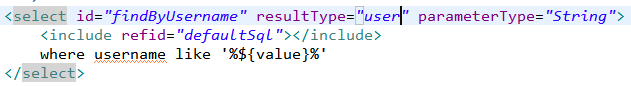
Sql 中可将重复的 sql 提取出来，使用时用 include 引用即可，最终达到 sql 重用的目的。

定义代码片段：



引用代码片段:





# 10 Mybatis多表查询

## 10.1 表之间关系分析

### 10.1.1 表之间关系分析

表之间的关系有几种：

一对多

多对一

一对一

多对多

举例：

用户和订单就是一对多

订单和用户就是多对一

一个用户可以下多个订单

多个订单属于同一个用户

人和身份证号就是一对一

一个人只能有一个身份证号

一个身份证号只能属于一个人

老师和学生之间就是多对多

一个学生可以被多个老师教过

一个老师可以交多个学生

特例：

如果拿出每一个订单，他都只能属于一个用户。

所以Mybatis就把多对一看成了一对一。

### 10.1.2 Mybatis中的多表查询

mybatis中的多表查询：

示例：用户和账户

一个用户可以有多个账户

一个账户只能属于一个用户（多个账户也可以属于同一个用户）

步骤：

1、建立两张表：用户表，账户表

让用户表和账户表之间具备一对多的关系：需要使用外键在账户表中添加

2、建立两个实体类：用户实体类和账户实体类

让用户和账户的实体类能体现出来一对多的关系

3、建立两个配置文件

用户的配置文件

账户的配置文件

4、实现配置：

当我们查询用户时，可以同时得到用户下所包含的账户信息

当我们查询账户时，可以同时得到账户的所属用户信息

示例：用户和角色

一个用户可以有多个角色

一个角色可以赋予多个用户

步骤：

1、建立两张表：用户表，角色表

让用户表和角色表具有多对多的关系。需要使用中间表，中间表中包含各自的主键，在中间表中是外键。

2、建立两个实体类：用户实体类和角色实体类

让用户和角色的实体类能体现出来多对多的关系

各自包含对方一个集合引用

3、建立两个配置文件

用户的配置文件

角色的配置文件

4、实现配置：

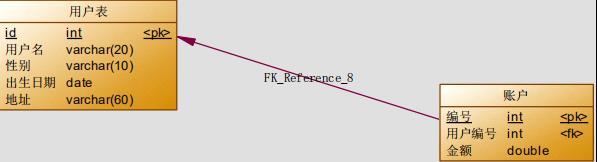
当我们查询用户时，可以同时得到用户所包含的角色信息

当我们查询角色时，可以同时得到角色的所赋予的用户信息

## 10.2 一对一查询（多对一查询）

本次案例主要以最为简单的用户和账户的模型来分析 Mybatis 多表关系。用户为 User 表，账户为Account

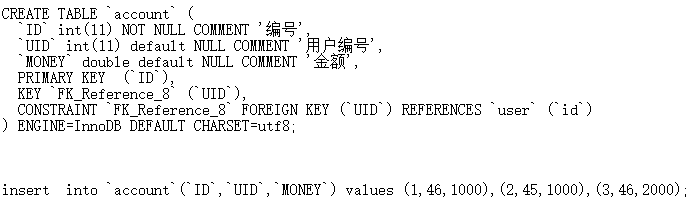
表。一个用户（User）可以有多个账户（Account）。具体关系如下：

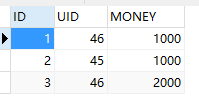


### 10.2.1 环境搭建

**1. 创建用户表（已经创建）和账户表：**

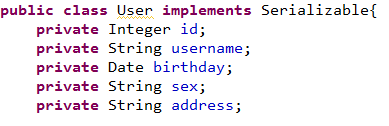
账户表结构和数据如下：

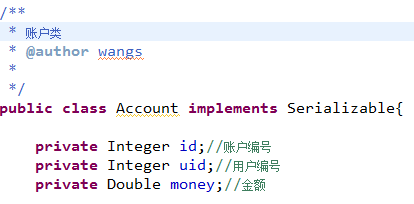




**2.创建工程：mybatis\_one2many**

**3.创建实体类User 和 Account**





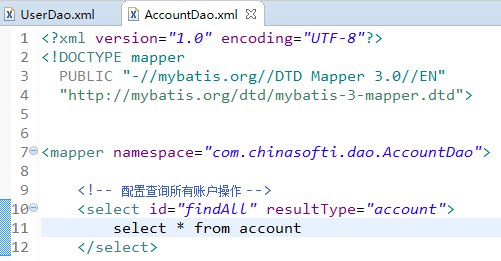
**4. 创建持久层接口**





**5. 创建sql映射文件**





**6.其它配置文件：**

SqlMapConfig.xml主配置文件

log4j.properties配置文件

jdbc.properties配置文件

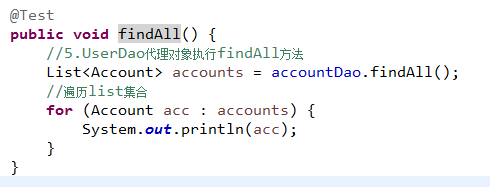
pom.xml配置文件

这里不再一一列举

**7.创建测试类**

UserTest.java 和AccountTest.java

**8. 测试查找所有账户信息方法：**



执行结果：



### 10.2.2 一对一查询方式一

查询所有账户信息，并关联账户对应的用户信息。

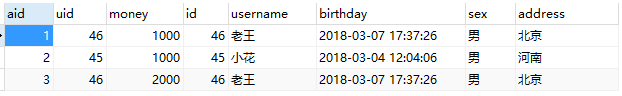
因为一个账户信息只能供某个用户使用，所以从查询账户信息出发关联查询用户信息为一对一查询。

#### 10.2.2.1 编写sql语句

如果我们要查询所有的账户信息，并关联账户所属的用户信息，sql语句如下：



查询结果为：



查询要求得到所有的账户信息，并关联账户所属的用户信息，这里用户信息，我们要了所有列的信息，也可以要部分列的信息，比如用户名和地址



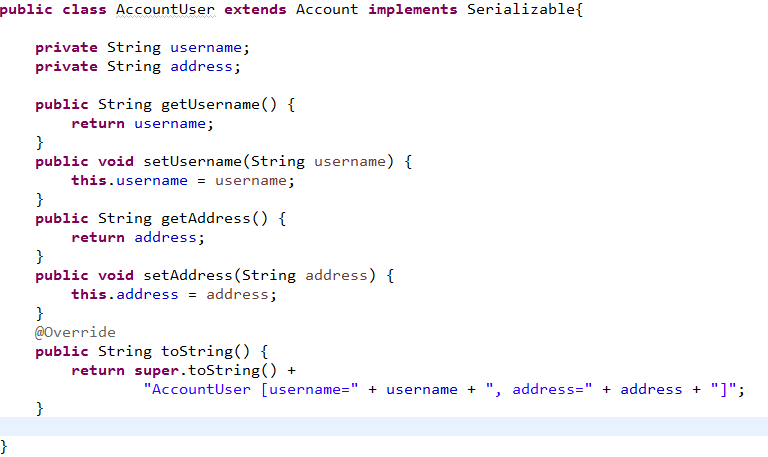
结果为：



以查询账户信息，并关联用户名和地址为例。

#### 10.2.2.2 创建AccountUser类

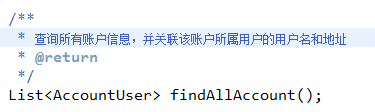
我们要查询的数据，包含账户的所有信息，并加入部分用户信息。可以这样来实现：



AccountUser类继承了Account类，这样AccountUser类中就包含了Account中的所有信息，并包含部分User信息

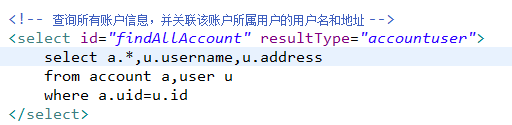
#### 10.2.2.3 在持久层接口创建方法

在AccountDao.java中创建方法：



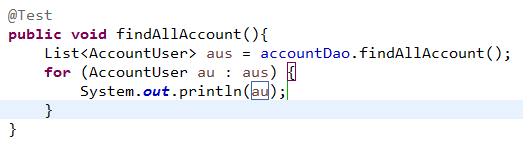
#### 10.2.2.4 在映射文件中添加配置

在AccountDao.xml中添加配置：



#### 10.2.2.5 在AccountTest类中添加方法

在AccountTest中添加测试方法：



执行结果：



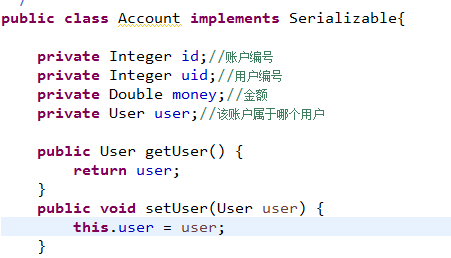
定义专门的AccountUser类作为输出类型，其中定义了 sql 查询结果集所有的字段。此方法较为简单，企业中使用普遍。

### 10.2.3 一对一查询方式二

#### 10.2.3.1 修改Account类

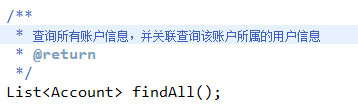
刚才的方法比较简单，我们还可以使用 resultMap，定义专门的 resultMap 用于映射一对一查询结果。

通过面向对象的(has a)关系可以得知，可以在 Account 类中加入一个 User 类的对象来代表这个账户是哪个用户的。



#### 10.2.3.2 修改持久层接口中的方法

在AccountDao.java接口中修改方法：

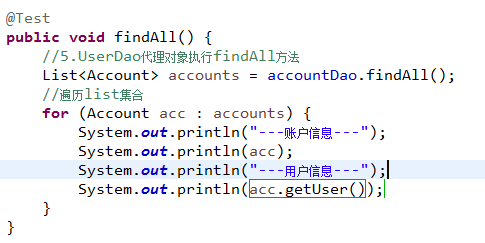


#### 10.2.3.3 在映射文件中修改配置

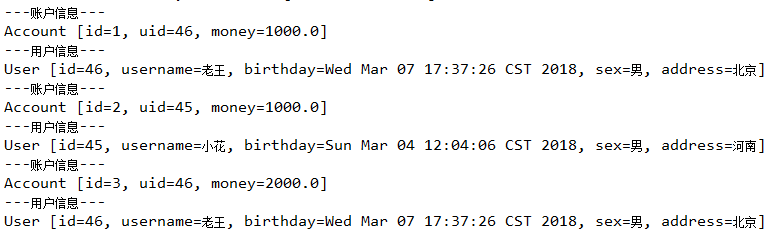
在AccountDao.xml映射文件中修改配置：



#### 10.2.3.4 在AccountTest类中添加测试方法



执行结果：



## 10.3 一对多查询

### 10.3.1 编写sql语句

需求：

查询所有用户信息及用户关联的账户信息。

分析：

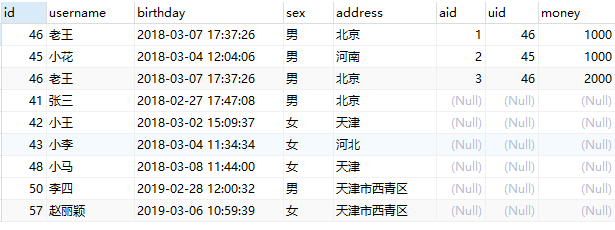
用户信息和他的账户信息为一对多关系，并且查询过程中如果用户没有账户信息，此时也要将用户信息

查询出来，左外连接查询比较合适。

sql语句如下：

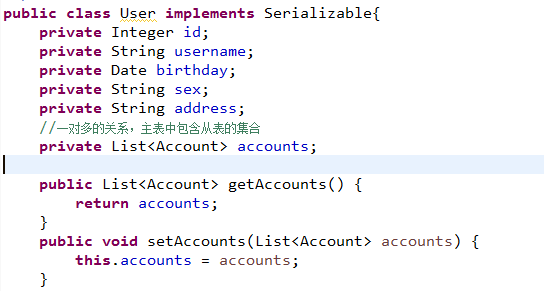


查询结果：



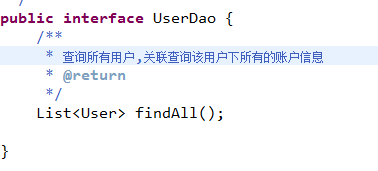
### 10.3.2 改写User类

一个User可能包含多个账户，一对多的关系，表现在代码上为：



### 10.3.3 在持久层接口UserDao中添加方法

在UserDao.java接口中添加方法



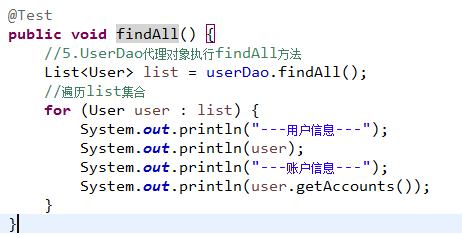
### 10.3.4 在映射文件中添加配置

在UserDao.xml映射文件中添加配置：

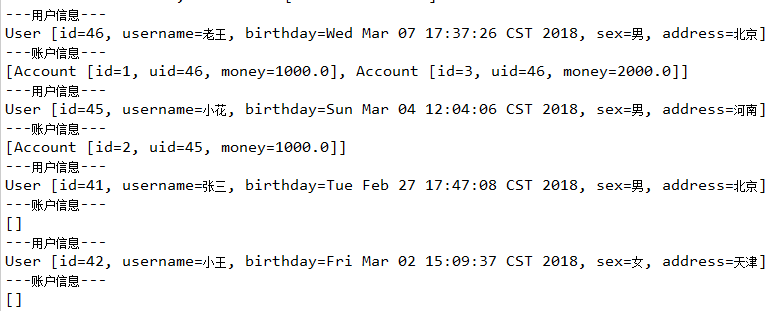


### 10.3.5 在UserTest测试类中添加测试方法

在测试类中添加测试方法：



执行结果为：



除了45,46用户，其它的用户没有账户信息

## 10.4 多对多查询

### 10.4.1 环境准备

#### 10.4.1.1 用户与角色关系

通过前面的学习，我们使用 Mybatis 实现一对多关系的维护。多对多关系其实我们看成是双向的一对多关系。

一个角色可以赋予多个用户，一个用户也可以有多个角色，这种关系就是多对多的关系。



在数据库中添加角色表，用户与角色的中间表。

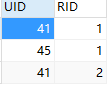
角色表：





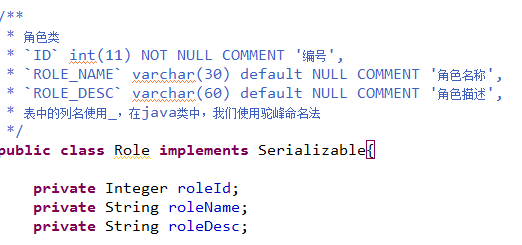
中间表：





#### 10.4.1.2 编写角色实体类

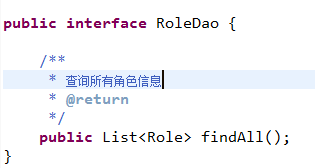
在domain包下创建实体类Role,如下：



set和get，toString这里不再写了

#### 10.4.1.3 编写Role持久层接口

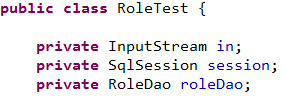
在dao包下，为Role创建持久层接口



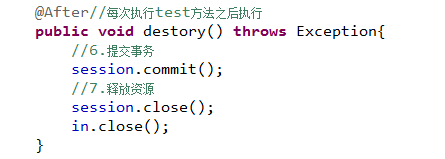
#### 10.4.1.4 编写Role的映射配置文件

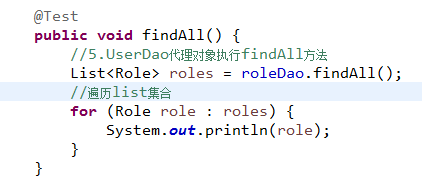


#### 10.4.1.5 编写RoleTest测试类









执行结果为：



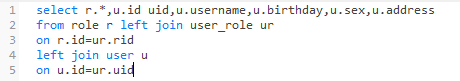
### 10.4.2 实现Role到User多对多

#### 10.4.2.1 编写sql

要查询所有的角色信息，并且关查询该角色下的所有用户信息：

角色信息是都要的，用户信息不一定，所以这里也可以说用左外连接

sql语句如下：



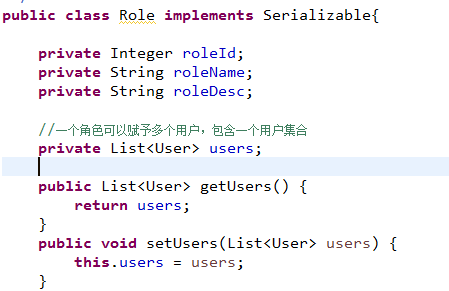
查询结果如下：



所以的角色信息都有，但是用户信息只显示有关联的。

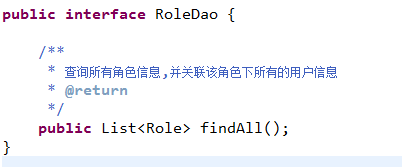
#### 10.4.2.2 修改Role类

一个角色可以对象多个用户，是一对多的关系，在代码上表示如下：



#### 10.4.2.3 在Role的持久层接口中添加方法

在RoleDao.java中添加方法;



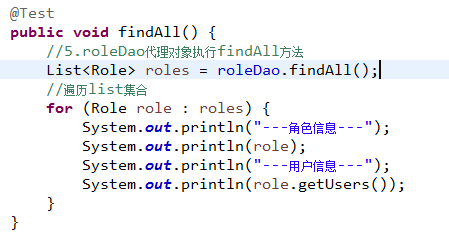
#### 10.4.2.4 在映射文件中添加配置

在RoleDao.xml映射中添加配置：

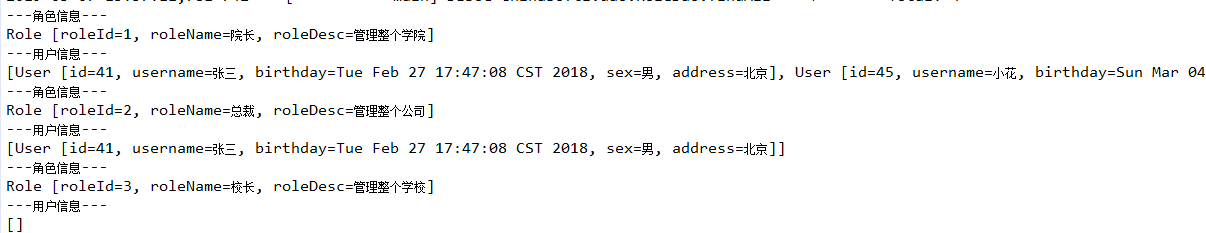


#### 10.4.2.5 在测试类中测试

在测试类中添加方法：



执行结果：



角色1对应2个用户，角色2对应1个用户，角色3没有用户

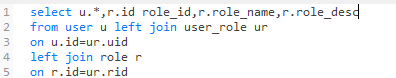
### 10.4.3 实现User到Role多对多

#### 10.4.3.1 编写sql

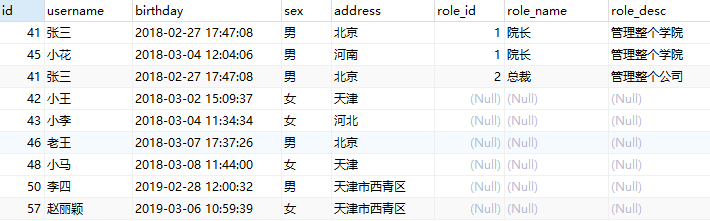
查询所有用户信息，并关联该用户的的所有角色信息。

要求所有的用户信息，角色信息不一定是全的，还是考虑使用左外连接

sql语句如下；



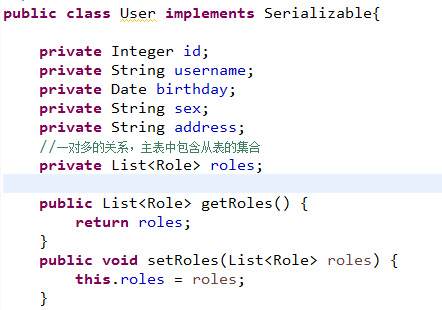
查询结果如下：



所有的用户信息，关联的角色信息

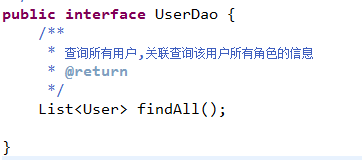
#### 10.4.3.2 修改User类

一个用户包含多个角色信息，这也是一对多的关系，在代码上表示如下：



#### 10.4.3.3 在持久层接口中添加方法

在UserDao.java中添加方法：



#### 10.4.3.4 在映射文件中添加配置

在UserDao.xml文件中添加配置:



#### 10.4.3.5 在测试类中添加测试

在UserTest测试类中添加测试方法：



执行结果：



41用户有2个角色，45有一个角色，其它用户没有角色

# 11 Mybatis延迟加载策略

## 11.1 什么是延迟加载

通过前面的学习，我们已经掌握了 Mybatis 中一对一，一对多，多对多关系的配置及实现，可以实现对象的关联查询。

问题：在一对多中，当我们有一个用户，它有100个账户。

在查询用户的时候，要不要把关联的账户查出来？

在查询账户的时候，要不要把关联的用户查出来？

在查询用户时，用户下的账户信息应该是，什么时候使用，什么时候查询的。

在查询账户时，账户的所属用户信息应该是随着账户查询时一起查询出来。

**延迟加载：**就是在需要用到数据时才进行加载，不需要用到数据时就不加载数据。延迟加载也称懒加载.

**好处：**先从单表查询，需要时再从关联表去关联查询，大大提高数据库性能，因为查询单表要比关联查询多张表速度要快。

**坏处：**因为只有当需要用到数据时，才会进行数据库查询，这样在大批量数据查询时，因为查询工作也要消耗时间，所以可能造成用户等待时间变长，造成用户体验下降。

**立即加载:**不管用不用，只要一调用方法，马上发起查询。

在对应的四种表关系中：一对多，多对一，一对一，多对多

一对多，多对多：通常情况下我们都是采用延迟加载。

多对一，一对一：通常情况下我们都是采用立即加载。

## 11.2 准备环境

### 11.2.1 准备工程

**一 、创建工程**

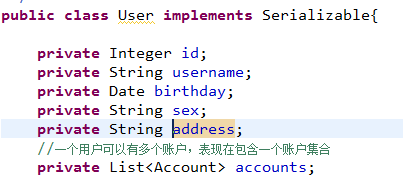
mybatis-lazy

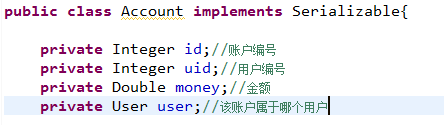
**二 、准备数据库表**

我们用用户表user和账户表account来演示一对一和一对多关系。前面已经创建了表。

**三、创建实体类**

前面我们已经创建了实体类User和Account



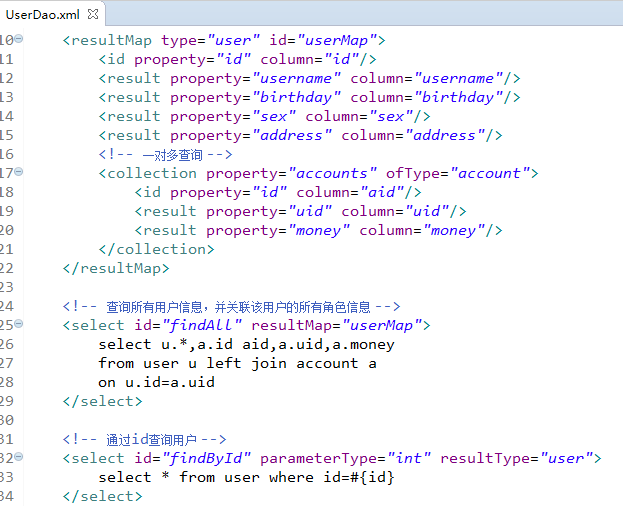


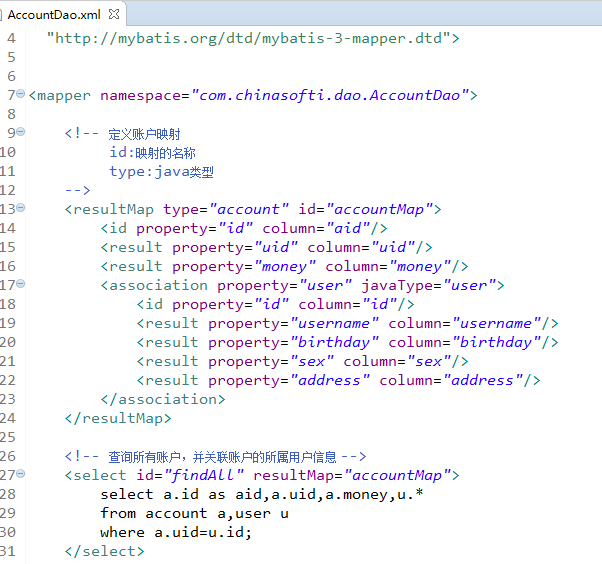
**四、创建持久层接口**





**五、创建sql映射文件**





**六、 其它配置文件**

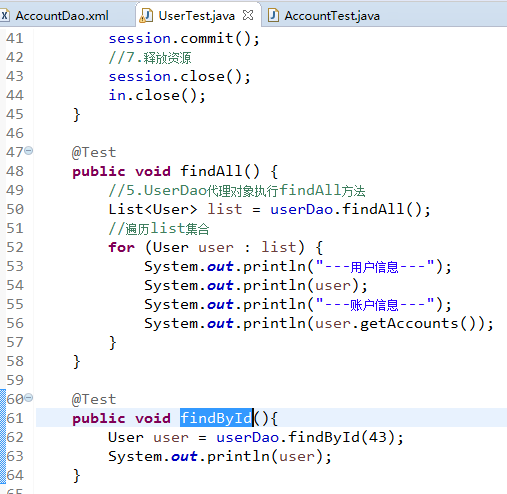
SqlMapConfig.xml主配置文件

jdbc.properties配置文件

log4j.properties配置文件

pom.xml配置文件等

**七、创建测试类**





## 11.3 一对一延迟加载

### 11.3.1 分析一对一查询

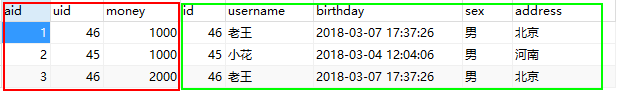
在前面的一对一多表查询时，我们写的sql语句如下：

select a.id as aid,a.uid,a.money,u.\*

from account a,user u

where a.uid=u.id;

执行结果如下：



一旦sql语句执行，account表中的数据会查询，user表也会立即查询，这里并没有体现延迟加载。如果是延迟加载的话，应该只查询account表，什么时候使用user表中的内容，什么时候查询user表。

到底该怎么实现懒加载呢?

### 11.3.2 修改sql映射配置文件

修改AccountDao.xml映射配置文件

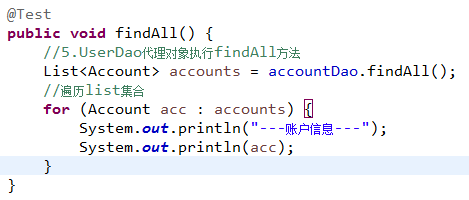


在UserDao.xml映射文件中，有statement：

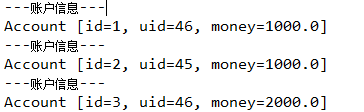


### 11.3.3 在测试类中测试

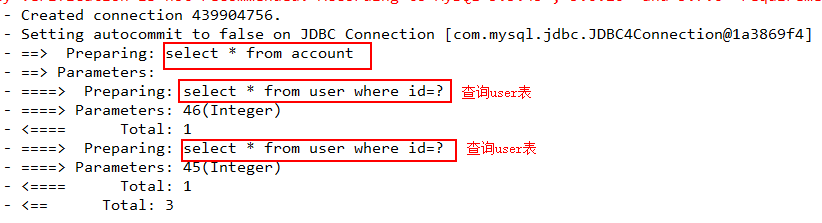
在AccountTest.java中添加测试方法：



执行结果为：



查看日志，可以发现：



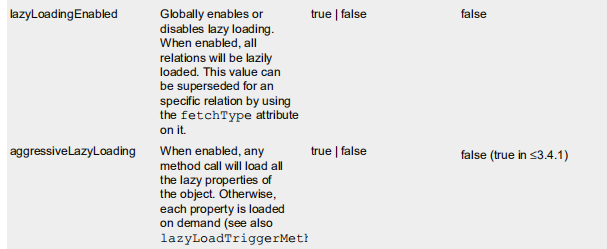
我们发现即使在测试方法中，并没有使用到user，但是却执行了查询user表的操作。这样还是没有实现懒加载。

### 11.3.4 开启mybatis延迟加载策略

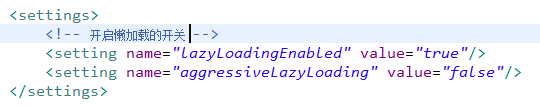
要想使用懒加载，我们需要在配置文件中配置

一.在SqlMapConfig.xml主配置文件中添加配置：

在Mybatis官方文档，找到settings说明信息



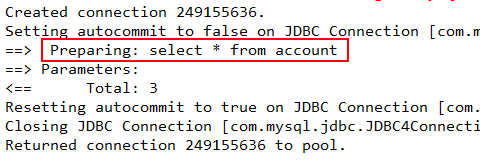
开启延迟加载支持：



测试：如果我们只运行查询方法：

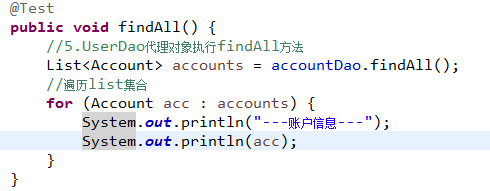


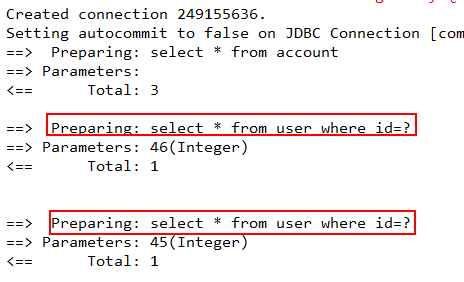
日志信息为：



这时并没有查询user表，真正实现了懒加载。

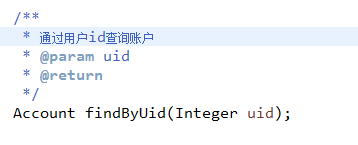
究竟什么时候会查询user表呢，当使用的时候。比如，如果我们不仅查询，还使用acc，使用acc就会使用user，这时就会查询user表



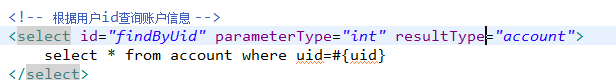


## 11.4 一对多延迟加载

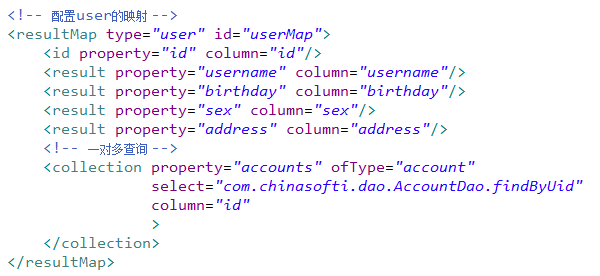
### 11.4.1 在AccountDao接口中添加方法



### 11.4.2 在Account映射文件中添加配置

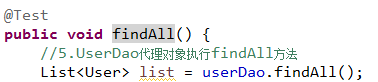


### 11.4.3 修改User的映射文件

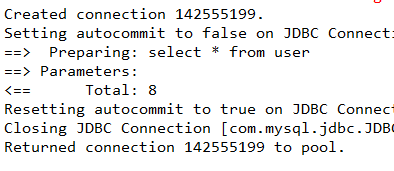


### 11.4.4 测试

测试代码：



执行结果：



这里并没有查询account表，实现了延迟加载。

如果使用user对象，则会查询account表：



# 12 Mybatis缓存

## 12.1 缓存的概念

**什么是缓存**

存在于内存中的临时数据。

**为什么使用缓存**

减少和数据库的交互次数，提高执行效率。

**什么样的数据能使用缓存，什么样的数据不能使用**

适用于缓存：

经常查询并且不经常改变的。

数据的正确与否对最终结果影响不大的。

不适用于缓存：

经常改变的数据

数据的正确与否对最终结果影响很大的。

例如：商品的库存，银行的汇率，股市的牌价。

像大多数的持久化框架一样，Mybatis 也提供了缓存策略，通过缓存策略来减少数据库的查询次数，从而提高性能。

Mybatis 中缓存分为一级缓存，二级缓存



## 12.2 准备环境

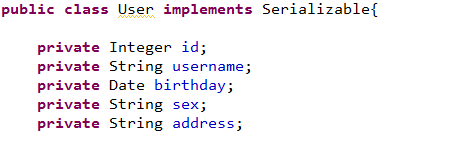
**一、创建工程**

mybatis-cache

**二、创建数据库表**

为了演示的方便，我们只使用user表

**三、创建实体类User**

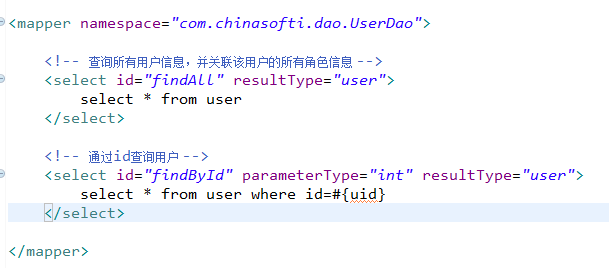


注意这里的User类不提供toString方法

**四、创建User持久层接口**



**五、创建UserDao.xml映射配置文件**



**六、其它配置文件**

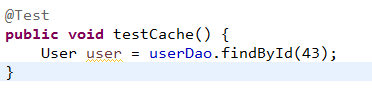
SqlMapConfig.xml主配置文件

jdbc.properties配置文件

log4j.properties配置文件

pom.xml配置文件

**七、创建测试类**



## 12.3 一级缓存

一级缓存：

它指的是Mybatis中SqlSession对象的缓存。

当我们执行查询之后，查询的结果会同时存入到SqlSession为我们提供一块区域中。

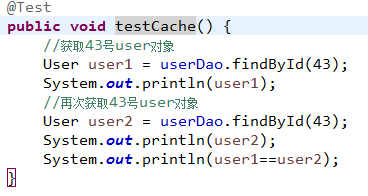
该区域的结构是一个Map。当我们再次查询同样的数据，mybatis会先去sqlsession中查询是否有，有的话直接拿出来用。

当SqlSession对象消失时，mybatis的一级缓存也就消失了。

**一级缓存是 SqlSession 级别的缓存，只要 SqlSession 没有 flush 或 close，它就存在。**

### 12.3.1 证明一级缓存的存在

在测试方法中，两次获取43号用户对象，如下：



执行结果如下：



可以看到，先后获得的两个user对象的地址值相同，说明是同一个对象，查看日志信息：



我们可以发现，虽然在上面的代码中我们查询了两次，但最后只执行了一次数据库操作，这就是Mybatis提供给我们的一级缓存在起作用了。因为一级缓存的存在，导致第二次查询 id 为 43 的记录时，并没有发出 sql 语句从数据库中查询数据，而是从一级缓存中查询。

### 12.3.2 清空一级缓存的情况

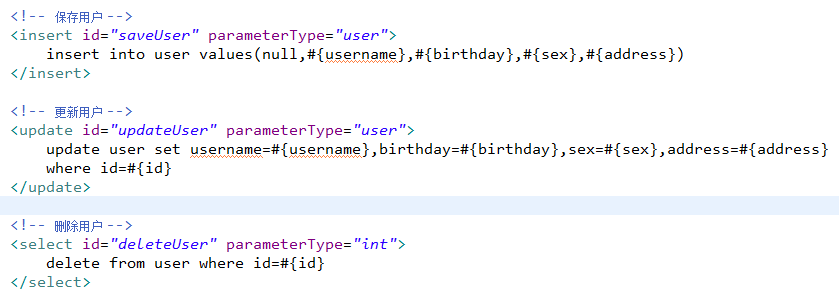
#### 12.3.2.1 在User持久层中添加方法

在UserDao.java中添加保存，更新，删除用户的方法：



#### 12.3.2.2 在UserDao映射文件中添加配置

在UserDao.xml中添加增删改的配置：



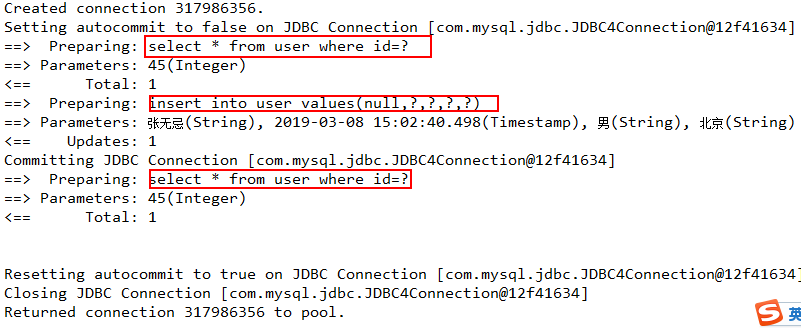
#### 12.3.2.3 在测试类中测试

**情况一：修改测试方法如下：在两次查询中间有插入操作，commit**



执行结果如下：





我们发现此时并没有像之前一样，从SqlSession一次缓存中取数据，而是又操作了一次数据库。

**情况二：在两次查询中执行了更新操作，commit**



**情况三：执行了删除操作，commit**



当执行删除操作时，结果也是一样的，并没有从一级缓存中拿数据，而是重新操作数据库。

**情况四： 调用清空缓存方法**

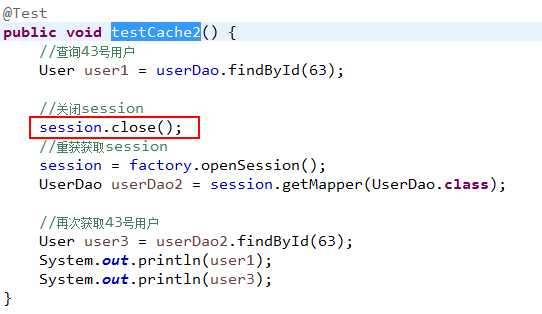
同样的如果我们调用SqlSession的清空缓存的方法，也会清空一级缓存中的内容





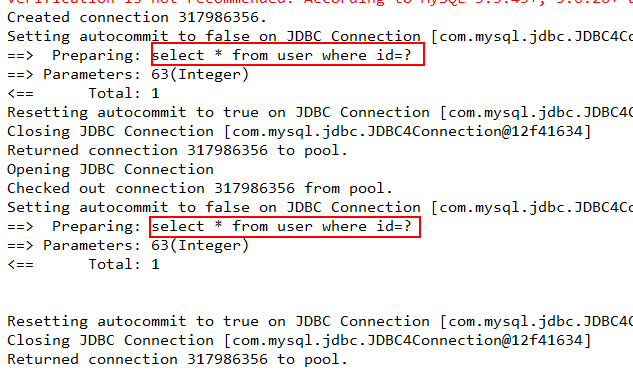


**情况五：session关闭**



执行结果

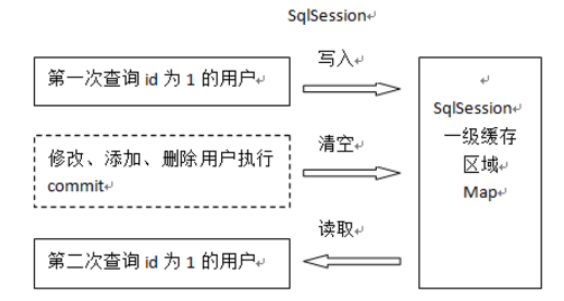




这时，也会清空缓存，再次操作数据库。

#### 12.3.2.4 清空缓存总结

一级缓存是 SqlSession 范围的缓存，当调用 SqlSession 的修改，添加，删除，commit()，close()等方法时，就会清空一级缓存。



第一次发起查询用户 id 为 1 的用户信息，先去找缓存中是否有 id 为 1 的用户信息，如果没有，从数据库查询用户信息。

得到用户信息，将用户信息存储到一级缓存中。

如果 sqlSession 去执行 commit 操作（执行插入、更新、删除），清空 SqlSession 中的一级缓存，这样做的目的为了让缓存中存储的是最新的信息，避免脏读。

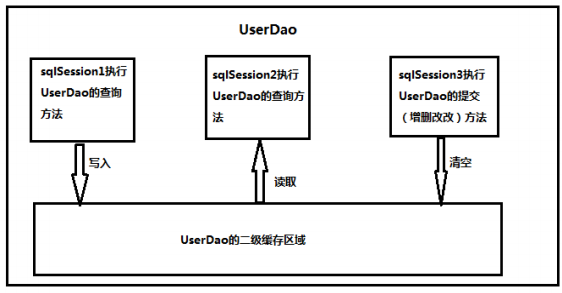
第二次发起查询用户 id 为 1 的用户信息，先去找缓存中是否有 id 为 1 的用户信息，缓存中有，直接从缓存中获取用户信息，如果没有，从数据库查询用户信息。

## 12.4 二级缓存

### 12.4.1 二级缓存分析

二级缓存是 mapper 映射级别的缓存，多个 SqlSession 去操作同一个 Mapper 映射的 sql 语句，多个SqlSession 可以共用二级缓存，二级缓存是跨 SqlSession 的。

可以认为：它指的是Mybatis中SqlSessionFactory对象的缓存。由同一个SqlSessionFactory对象创建的SqlSession共享其缓存。



首先开启 mybatis 的二级缓存。

sqlSession1 去查询用户信息，查询到用户信息会将查询数据存储到二级缓存中。

如果 SqlSession3 去执行相同 mapper 映射下 sql，执行 commit 提交，将会清空该 mapper 映射下的二级缓存区域的数据。

sqlSession2 去查询与 sqlSession1 相同的用户信息，首先会去缓存中找是否存在数据，如果存在直接从缓存中取出数据。

看刚才的例子：



当SqlSession1执行了查询操作以后，清空一级缓存，SqlSession2执行查询操作，结果却是操作了2次数据库，并没有从二级缓存中拿数据。

如何使用二级缓存呢？

### 12.4.2 二级缓存开启

二级缓存的使用步骤：

第一步：让Mybatis框架支持二级缓存（在SqlMapConfig.xml中配置）

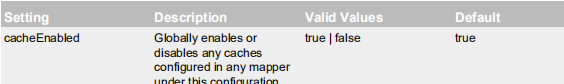
第二步：让当前的映射文件支持二级缓存（在IUserDao.xml中配置）

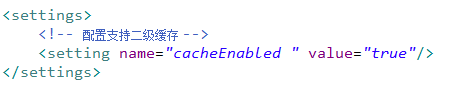
第三步：让当前的操作支持二级缓存（在select标签中配置）

**第一步：让Mybatis框架支持二级缓存**

在全局配置中可以配置，其实也可以不写，因为二级缓存的默认值为true，即支持二级缓存。

为 true 代表开启二级缓存；为false 代表不开启二级缓存

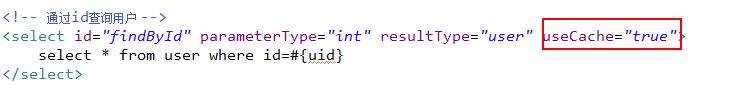




**第二步:让当前映射文件支持二级缓存**



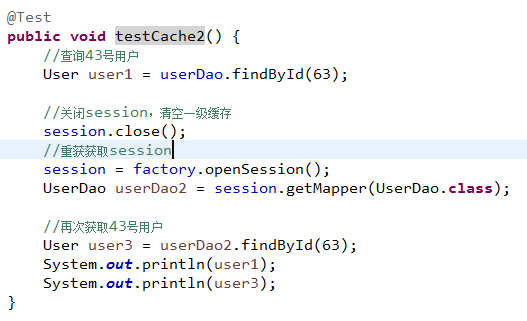
**第三步：让当前操作支持二级缓存**



将 UserDao.xml 映射文件中的<select>标签中设置 useCache=”true”代表当前这个 statement 要使用二级缓存，如果不使用二级缓存可以设置为false。

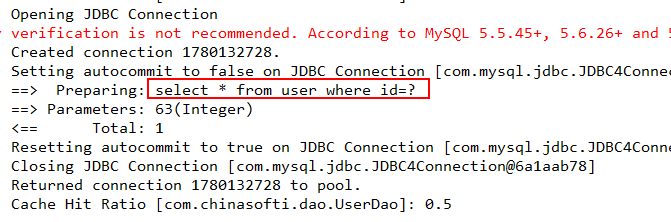
注意：针对每次查询都需要最新的数据 sql，要设置成 useCache=false，禁用二级缓存。

**测试：**



执行结果：





我们发现，虽然两个user对象的地址值是不同的，但是只操作了一次数据库，说明真的从二级缓存中查询了数据。

为什么2个user的不是同一个的。因为二级缓存中存的不是对象，而是以序列化的方式保存对象。User一定要实现序列化接口才可以。否则会出现异常



当我们从二级缓存中获取对象时，实际上是将序列化的数据重新创建为一个user对象。所以2个user的地址值不相同。

# 13 Mybatis注解开发

## 13.1 准备环境

### 13.1.1 准备环境

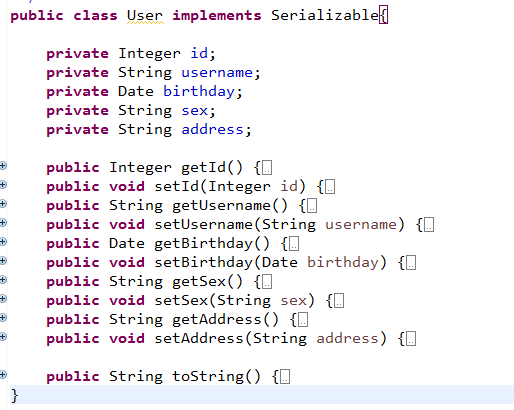
**一、创建工程**

mybatis-annotation1

**二、创建数据库表**

这里我们仍然使用user表

**三、创建实体类User**



**四、创建User持久层接口UserDao**



**注意：在注解开发中，千万不要写UserDao.xml映射配置文件！**

**五、其它配置文件**

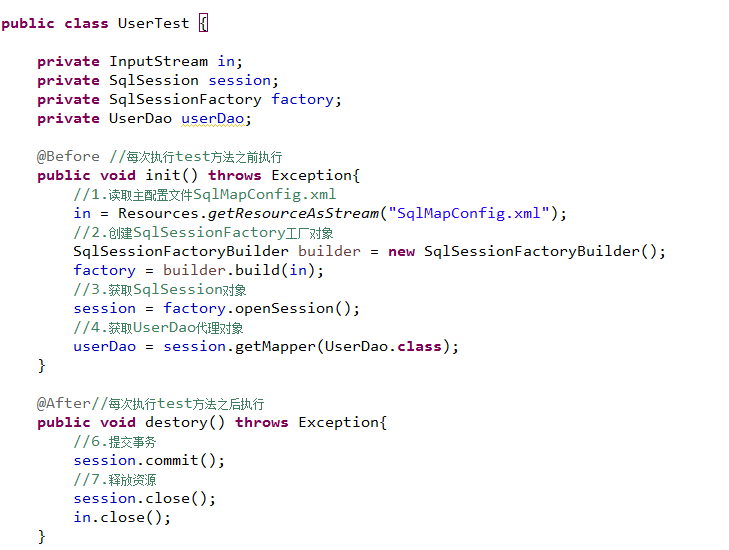
SqlMapConfig.xml主配置文件

jdbc.properties配置文件

log4j.properties配置文件

pom.xml配置文件

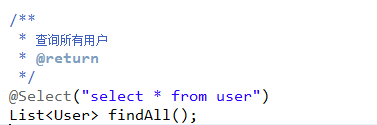
**六、创建测试类UserTest.java**



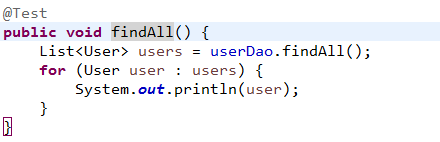
## 13.2 注解实现基本CRUD操作

### 13.2.1 查询所有用户

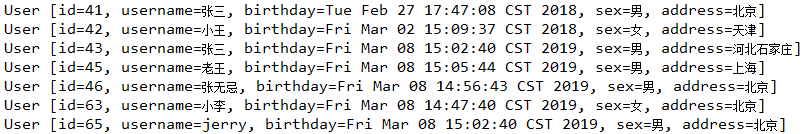
在User持久层接口UserDao.java中为方法添加注解：



测试:

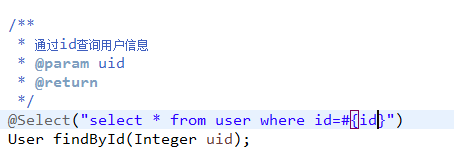


执行结果：

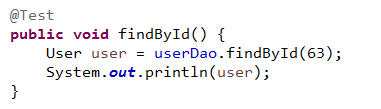


### 13.2.2 根据id查询用户

在User持久层接口UserDao.java中为方法添加注解：



测试:



执行结果：

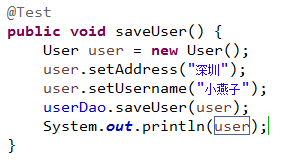


### 13.2.3 插入用户

在User持久层接口UserDao.java中为方法添加注解：

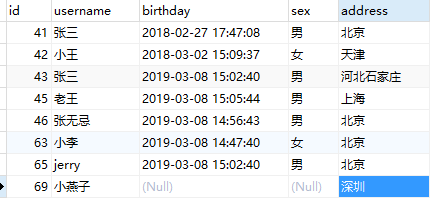


测试:



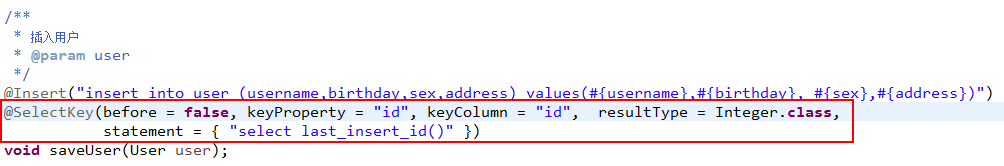
执行结果：



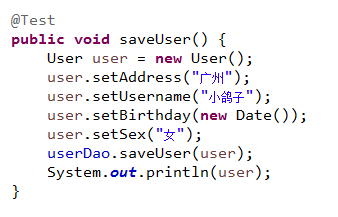


跟前面一样，插入数据的id是根据主键，自动增长实现的，表当中会自动生成一个id，但是生成的id在user对象却没有。

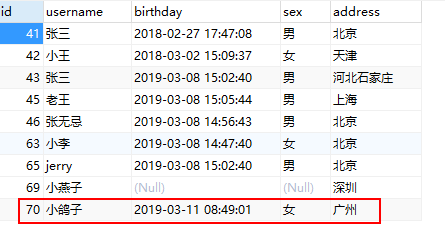
我们可以在生成id之后，将id查询出来，并保存在user对象中，这样需要在持久层接口的方法中添加注解：



测试代码：



结果：

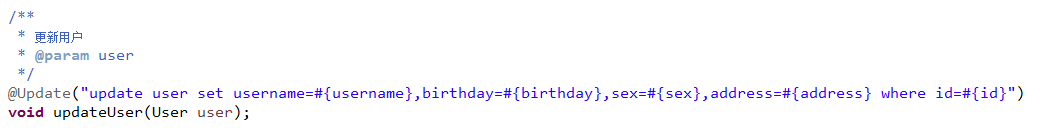




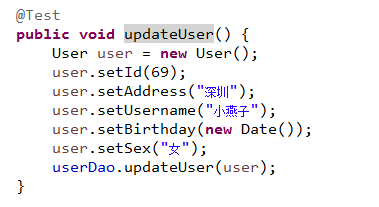
这样就可以了。

### 13.2.4 更新用户

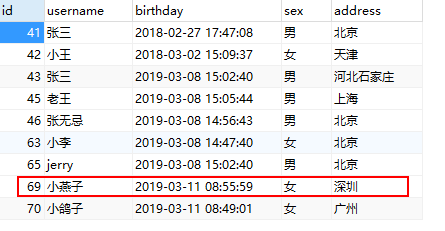
在User持久层接口UserDao.java中为方法添加注解：



测试:

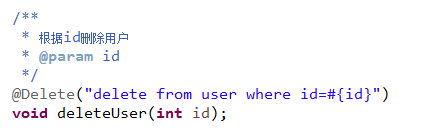


执行结果：

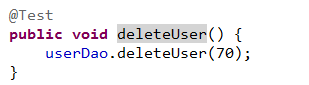


### 13.2.5 删除用户

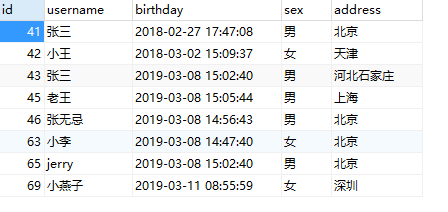
在User持久层接口UserDao.java中为方法添加注解：



测试:

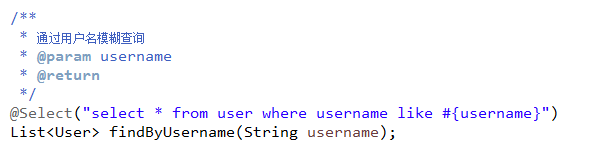


执行结果：

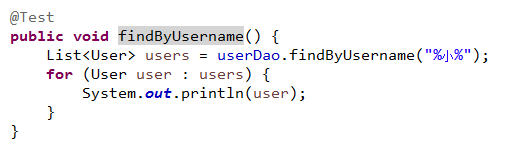


### 13.2.6 根据用户名模糊查询

在User持久层接口UserDao.java中为方法添加注解：



测试:

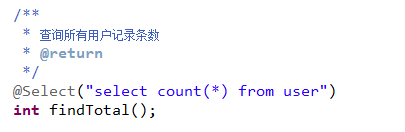


执行结果：

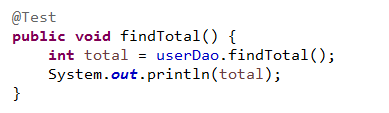


### 13.2.7 查询所有用户条数

在User持久层接口UserDao.java中为方法添加注解：



测试:

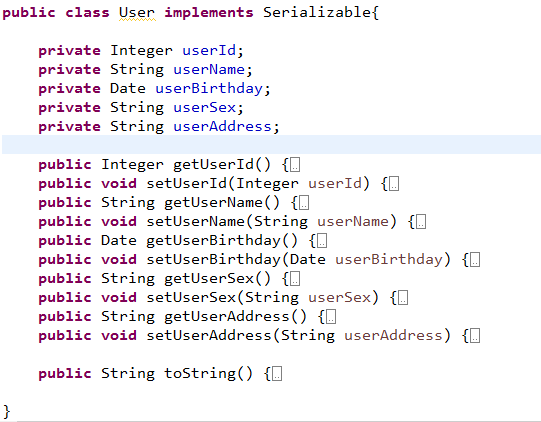


执行结果：

## 13.3 注解实现复杂关系映射

### 13.3.1 准备环境

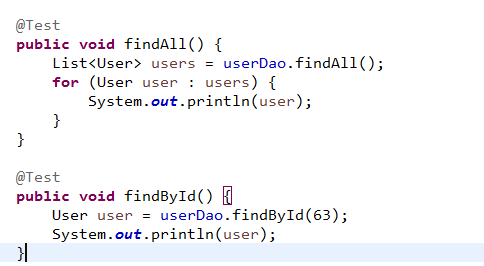
前面user类的属性与表的列名是相同的，下面我们看一下，不相同时该怎么做：



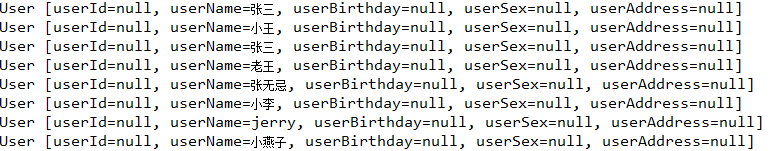
User的持久层接口只保留查询所有用户和根据id查询用户两个方法。



在测试类中也包含2个测试方法:



测试findAll方法，结果如下：



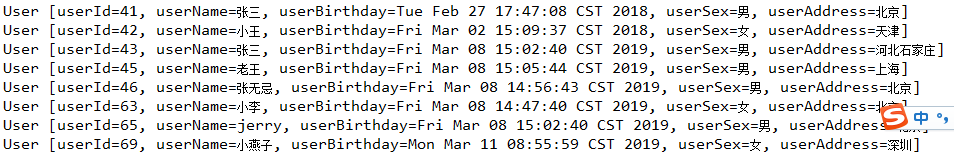
因为属性名称和列名不对应，所以在封装实体类时会出现问题。

### 13.3.2 注解实现属性和列名对应关系

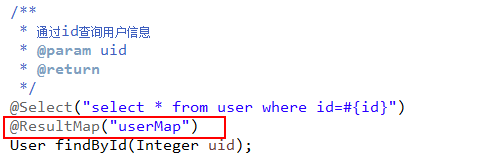
修改UserDao.java上方法注解：



测试结果：



在根据id查询用户的方法上，只需要做如下修改：



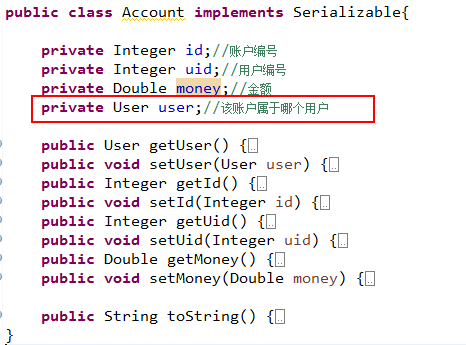
执行结果：



### 13.3.3 注解实现一对一查询

#### 13.3.3.1 添加Account类

一个账户对应一个用户，在类中表示一对一的关系：



#### 13.3.3.2 添加账户的持久层接口

添加Account的持久层接口：



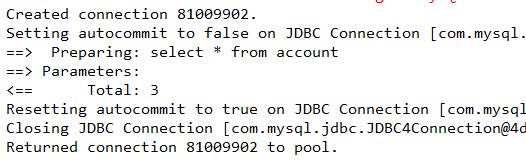
#### 13.3.3.3 查询账户信息并关联用户信息

在AccountDao.java持久层接口中添加注解：

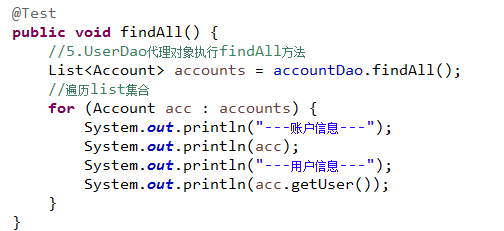


测试：

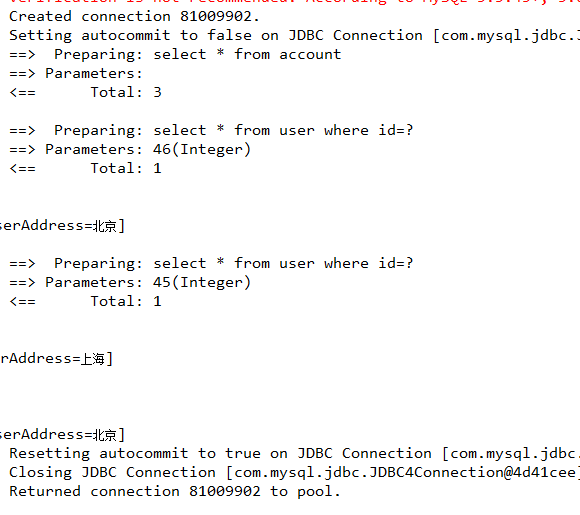




修改测试代码：



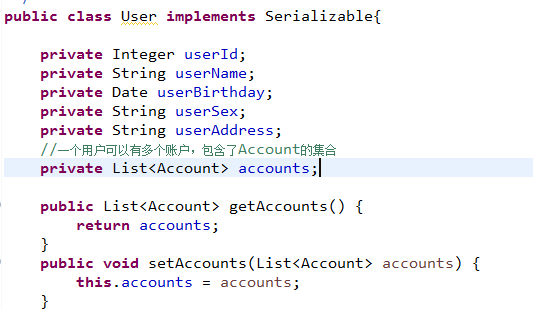
结果：



### 13.3.4 注解实现一对多查询

#### 13.3.4.1 修改User类

在类中添加User与Account的关系：

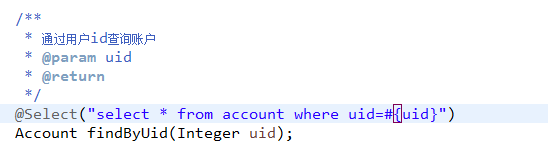


#### 13.3.4.2 修改注解

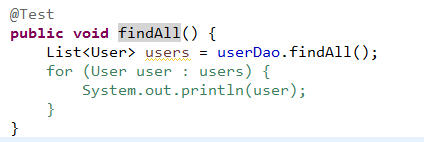
修改UserDao注解:



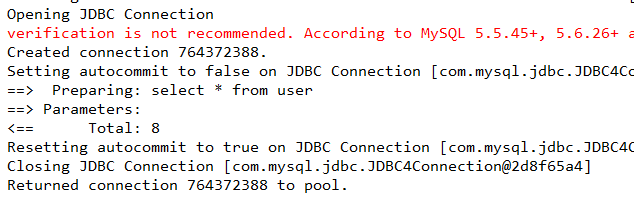
修改AccountDao中注解：



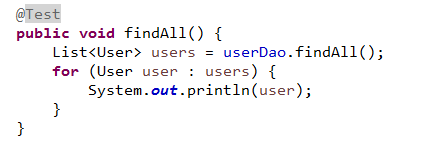
测试：



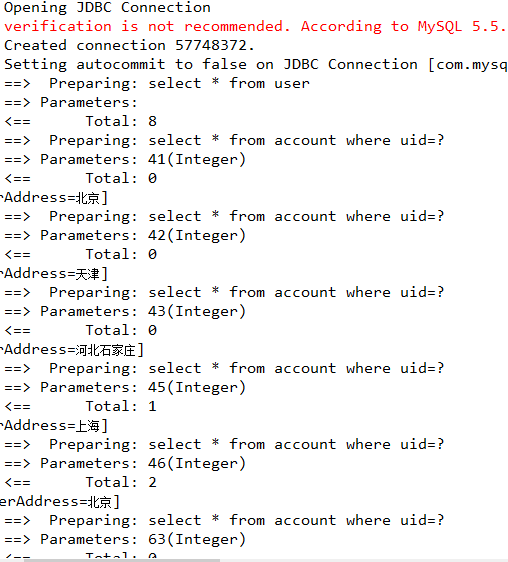
结果：



修改测试代码：

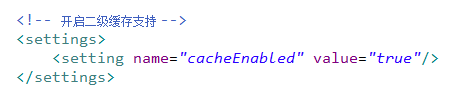


执行结果：



## 13.4 注解实现二级缓存

### 13.4.1 在SqlMapConfig.xml开启二级缓存支持



### 13.4.2 在接口中添加注解

在持久层上添加注解即可：

