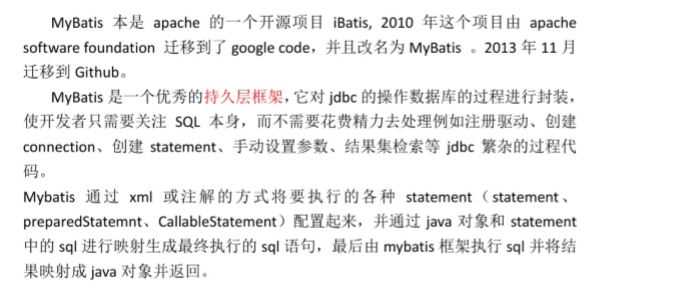
### mybatis的引入

#### mybatis的基本介绍



#### 为什么要学习MyBatis

* 目前最主流的持久层框架为hibernate与mybatis，而且国内目前情况使用Mybatis的公司比hibernate要多。
* Hibernate学习门槛不低，要精通门槛更高。门槛高在怎么设计O/R映射，在性能和对象模型之间如何权衡取得平衡，以及怎样用好Hibernate缓存与数据加载策略方面需要你的经验和能力都很强才行。国内目前前的情况精通hibernate技术大牛非常少。
* sql优化方面，Hibernate的查询会将表中的所有字段查询出来，这一点会有性能消耗。当然了，Hibernate也可以自己写SQL来指定需要查询的字段，但这样就破坏了Hibernate开发的简洁性。说得更深入一些，如果有个查询要关联多张表，比如5张表，10张表时，而且，我们要取的字段只是其中几张表的部分字段。这时用hibernate时就会显得非常力不从心。就算用hibernate的sqlquery，后续的维护工作也会让人发狂。

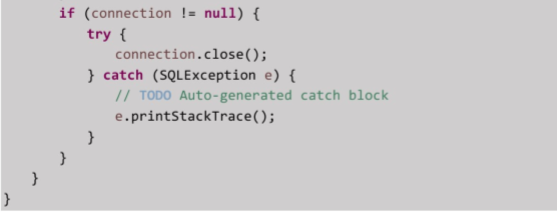
#### 传统JDBC编程回顾与存在的问题分析

Jdbc访问数据库的过程：

* 加载数据库驱动
* 创建数据库连接
* 创建statement
* 设置sql语句
* 设置查询参数
* 执行查询，得到ResultSet
* 解析结果集ResultSet
* 释放资源







#### 1.1.1.4 Jdbc存在的问题：

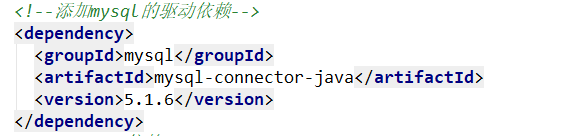
1. 频繁创建和打开、关闭数据连接，太消耗资源
2. Sql语句存在硬编码，不利于维护
3. Sql参数设置硬编码，不利于维护
4. 结果集获取与遍历复杂，存在硬编码，不利于维护，期望能够查询后返回一个java对象

### 1.1.2 第一个mybaitis程序

#### 1.1.2.1 创建一个maven的javaSE项目

#### 1.1.2.2 在项目的pom.xml里面添加对应的依赖

* Mysql驱动包的依赖



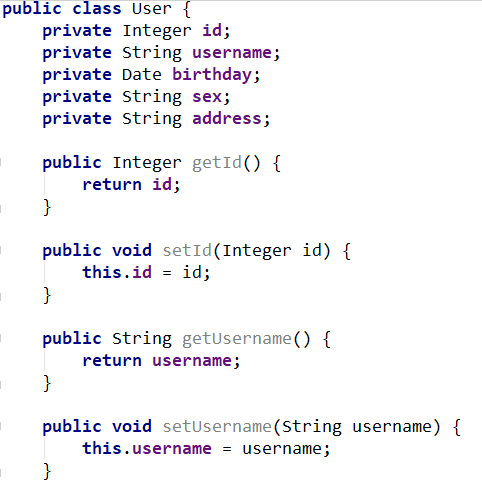
* Mybatis的包的依赖



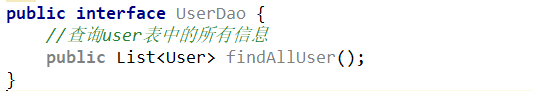
#### 1.1.2.3 提供数据表

|  |
| --- |
| CREATE TABLE `user` (  `id` int(11) NOT NULL auto\_increment,  `username` varchar(32) NOT NULL COMMENT '用户名称',  `birthday` datetime default NULL COMMENT '生日',  `sex` char(1) default NULL COMMENT '性别',  `address` varchar(256) default NULL COMMENT '地址',  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  insert into `user`(`id`,`username`,`birthday`,`sex`,`address`) values (41,'老王','2018-02-27 17:47:08','男','北京'),(42,'小二王','2018-03-02 15:09:37','女','北京金燕龙'),(43,'小二王','2018-03-04 11:34:34','女','北京金燕龙'),(45,'传智播客','2018-03-04 12:04:06','男','北京金燕龙'),(46,'老王','2018-03-07 17:37:26','男','北京'),(48,'小马宝莉','2018-03-08 11:44:00','女','北京修正'); |

#### 1.1.2.4 提供数据表对应的java实体类



#### 1.1.2.5 定义一个dao



#### 1.1.2.6定义mybatis核心的配置文件(xml)

* 在main文件夹下面定义一个资源文件夹 resources

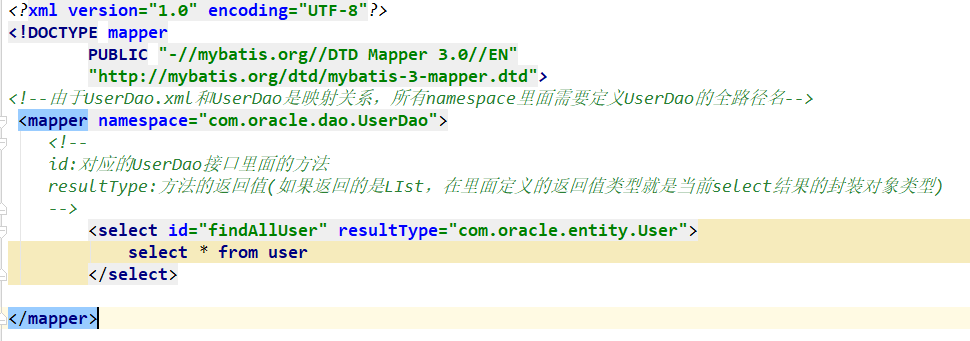


* 在resources文件夹下面，定义sqMapConfig.xml文件

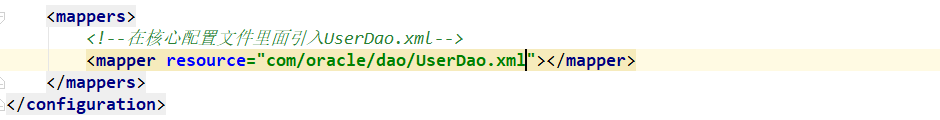


#### 1.1.2.7 配置dao对应的映射文件(xml)

注意：mybatis里面，一个dao对应的就是一个xml的映射文件



注意的是：当前的UserDao.xml文件配置好之后，还需要将其引入到核配置文件里面去。



#### 1.1.2.8 测试类



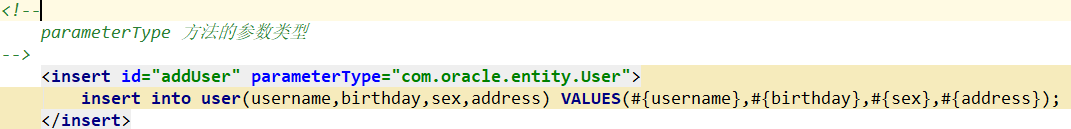
### 1.1.3 基于Mybatis的其他的CRUD操作

#### 1.1.3.1 添加的操作

* Dao



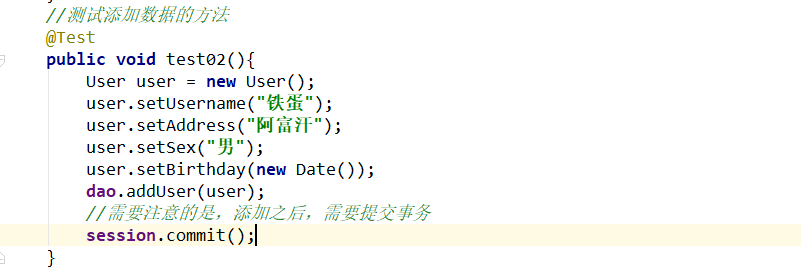
* Mapper映射文件



* 测试
* 共有代码的抽取



* 测试添加的方法

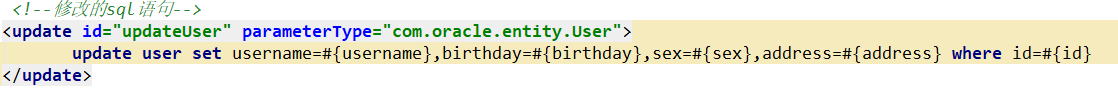


#### 1.1.3.2 修改的操作

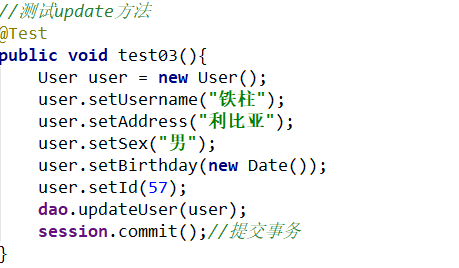
* Dao

|  |
| --- |
| *//修改的方法* **public void** updateUser(User user); |

* Dao的mapper映射文件



* 测试

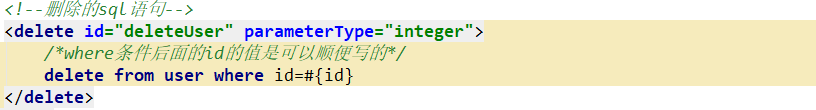


#### 1.1.3.3 删除的操作

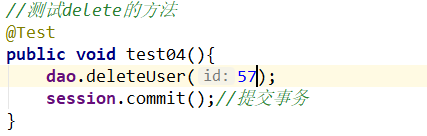
* dao

|  |
| --- |
| *//根据id删除的方法* **public void** deleteUser(Integer id); |

* dao的mapper映射文件



* 测试类

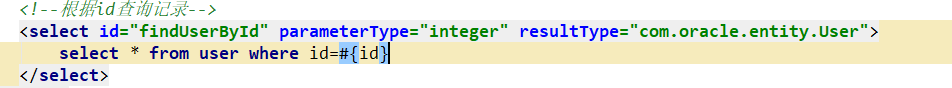


#### 1.1.3.4 基于单个条件的查询

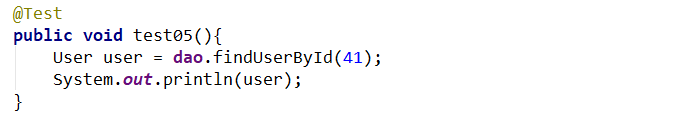
* Dao

|  |
| --- |
| *//根据id查询一条记录* **public** User findUserById(Integer id); |

* dao的mapper映射文件



* 测试类

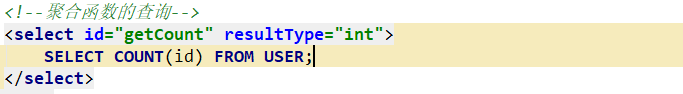


#### 1.1.3.5 基于聚合函数的查询

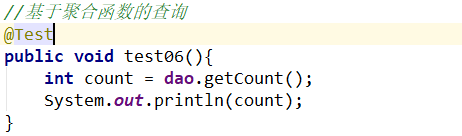
* dao

|  |
| --- |
| *//查询记录的总条数* **public int** getCount(); |

* dao的mapper映射文件



* 测试类

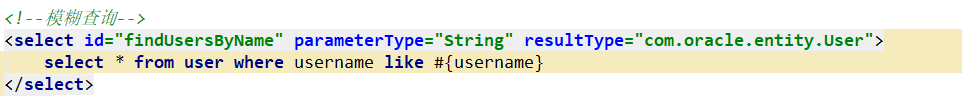


#### 1.1.3.6 基于模糊查询的操作

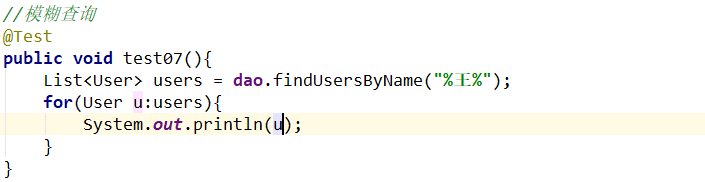
* Dao

|  |
| --- |
| *//模糊查询： 查询姓王的用户信息* **public** List<User> findUsersByName(String name); |

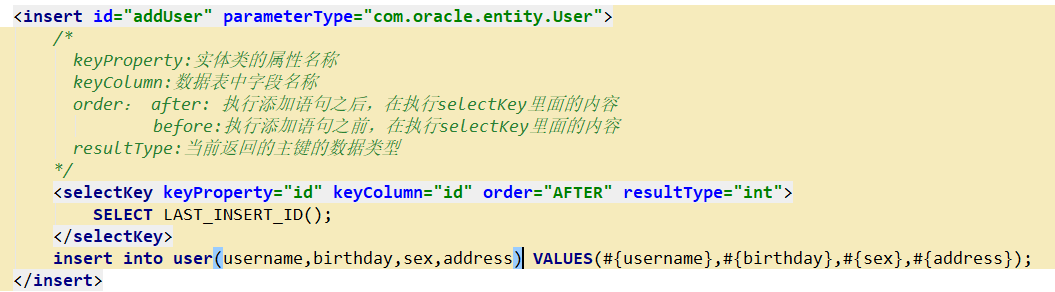
* dao的mapper映射文件



* 测试类



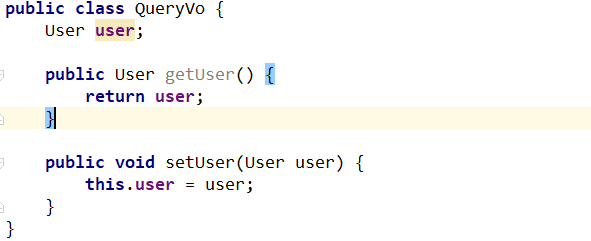
#### 1.1.3.7 补充：如何添加用户信息之后，获取当前用户信息的id



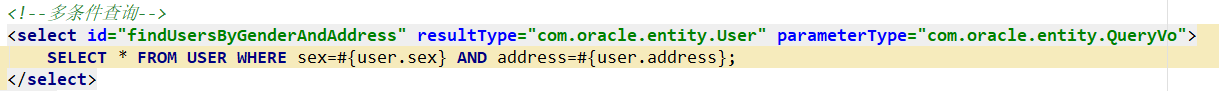
### 1.1.4 将实体类包装作为查询条件

需求：根据用户性别和用户地址查询对应的用户信息

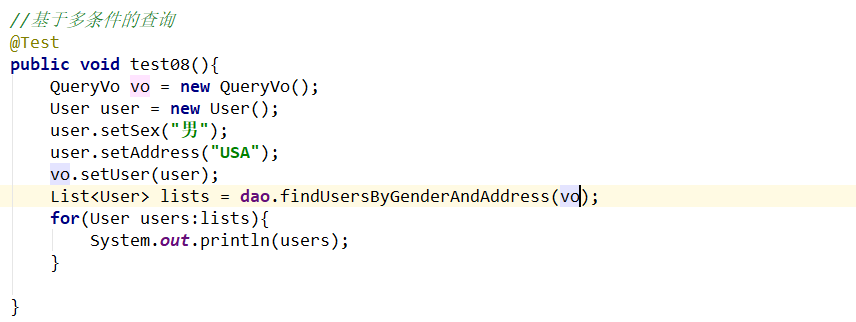
* 将多条件字段对应的实体类包装成另外的一个实体类



* 对应的mapper配置文件中



* 测试文件

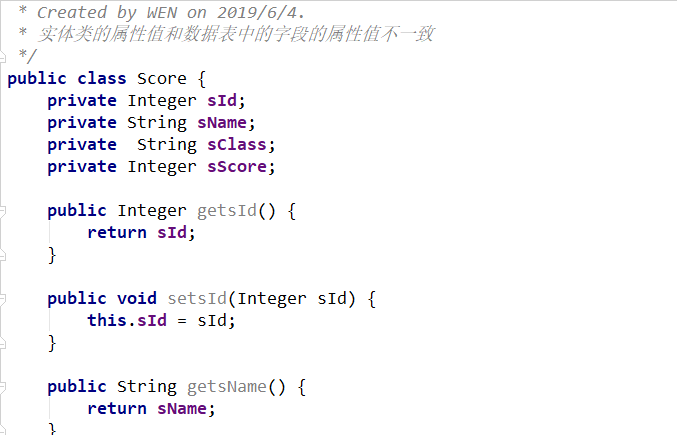


### 1.1.5 数据表的字段和实体类的字段名称不一致

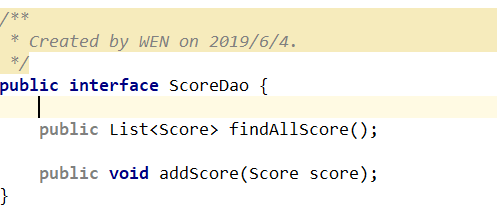
#### 1.1.5.1 数据表

|  |
| --- |
| CREATE TABLE score(  id INT PRIMARY KEY AUTO\_INCREMENT,-- id主键自增长  NAME VARCHAR(30), -- 学生姓名  class VARCHAR(30), -- 学生班级名称  score INT -- 学分数  ) |

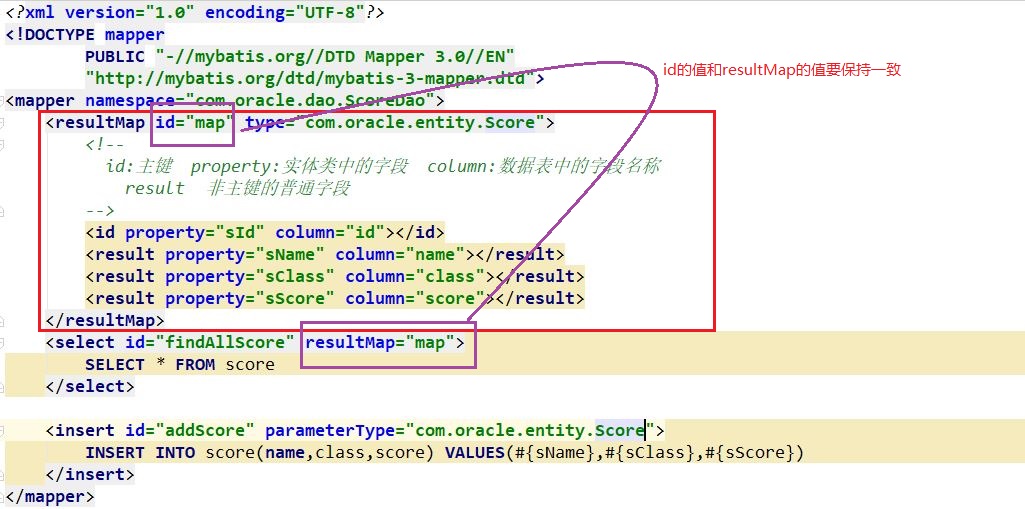
#### 1.1.5.2 实体类



#### 1.1.5.3 Dao



#### 1.1.5.4 Dao的映射文件



#### 1.1.5.5 测试略

### 2.1.1 mybatis配置文件中的一些细节

#### 2.1.1.1 properties文件的引入

我们可以把连接数据库的基本信息定义在propertites文件里面，然后在mybatis的核心配置文件中引入进来即可。我们可以使用一个propertites标签来引入



#### 2.1.1.2 typeAliases 标签的使用

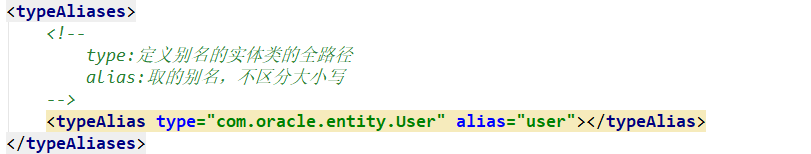
如果在开发中，有一些实体类的全路径会被经常引用，那么我们可以给这个实体类的全路径设置一个别名，那么以后我们就可以使用这个别名即可。不需要去写当前实体类的全路径，这样可以提高开发效率。

<typeAliases>

<typeAlias type="实体类的全路径" alias="别名"></typeAlias>

< /typeAliases>

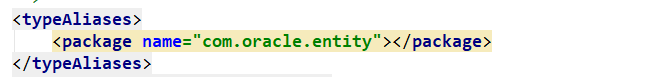
注意当前配置别名的标签是定义在核心配置文件里面



#### 2.1.1.3 package标签的使用

* 取别名

指定当前实体类所在的包名，实体类的别名就是当前实体类的类名并且不区分大小写



* 自动导入dao对应的配置文件

定义package,就可以自动的将指定目录下面的xml配置文件引入进去

**注意: 使用package导入映射配置文件的时候，接口的名称和映射文件的名称必须要保持一致!!!**

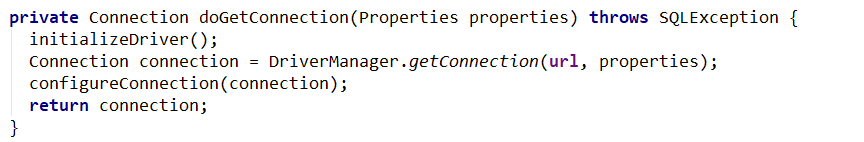


### 2.1.2 Mybatis的连接池(了解)

**dataSource**的type类型有三种方式：

#### 2.1.2.1 UNPOOLED

UNPOOLED:Mybatis会为每一个数据库创建一个新的连接，用完之后直接关闭。这种方式适用于小规模数量并发用户的简单应用程序上(不太常用)。



#### 2.1.2.2 POOLED

POOLED：mybatis会创建一个连接池，连接池中的一个连接将会被用于数据库的操作，如果操作数据库完毕，那么mybatis会将该连接对象返还给连接池，在开发中经常使用这种方式



JNDI:后面讲解

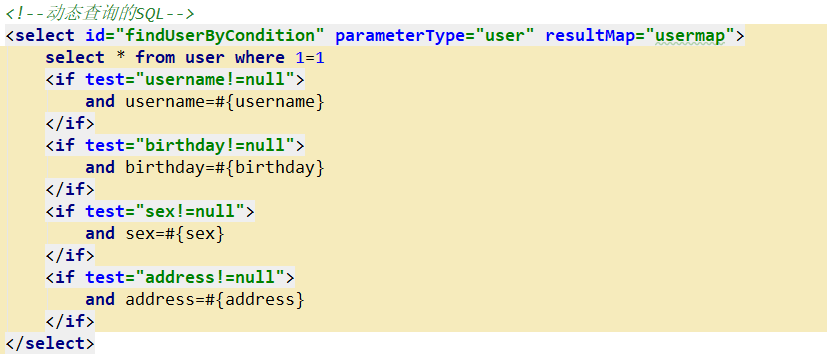
### 2.1.3 动态SQL(重点)

#### 2.1.3.1 动态查询

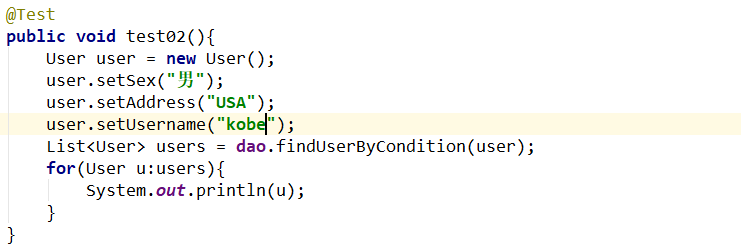
* dao

|  |
| --- |
| *//多条件动态查询* **public** List<User> findUserByCondition(User user); |

* dao对应的映射文件



* 测试

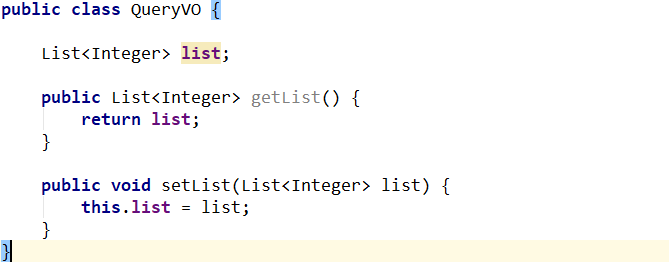


#### 2.1.3.2 批量查询的动态sql

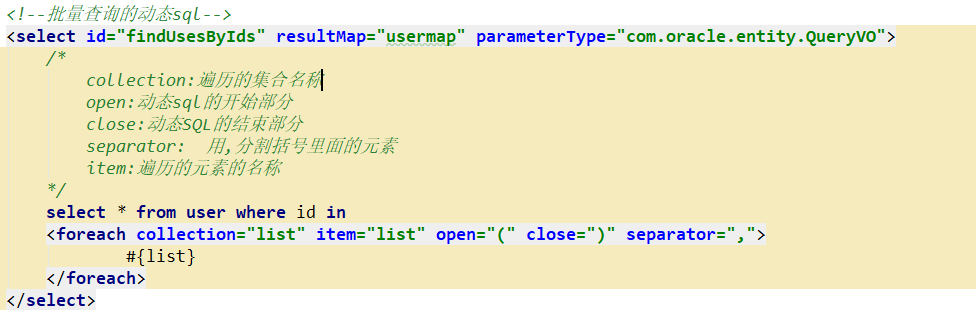
* dao

|  |
| --- |
| *//批量查询* **public** List<User> findUsesByIds(QueryVO vo); |

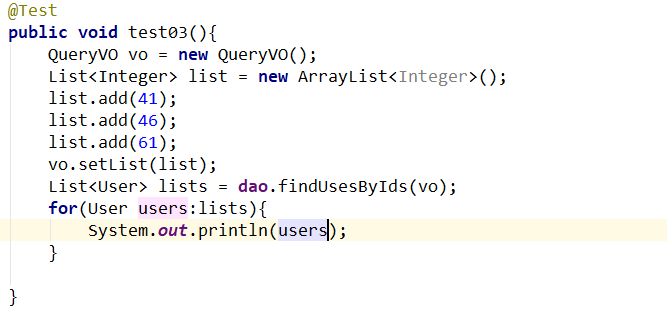
* 参数包装成实体类



* dao对应映射文件



* 测试

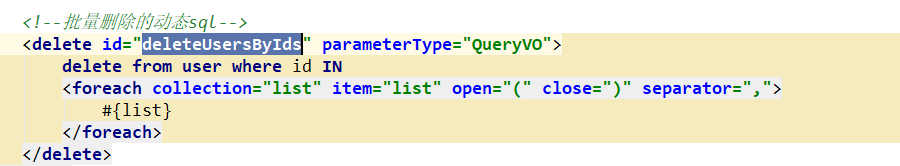


#### 2.1.3.3 批量删除

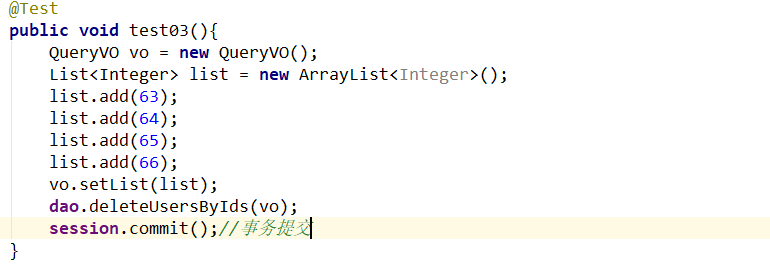
* Dao

|  |
| --- |
| *//批量删除的动态sql* **public void** deleteUsersByIds(QueryVO vo); |

* dao对应映射文件



* 测试



#### 2.1.3.4 动态修改

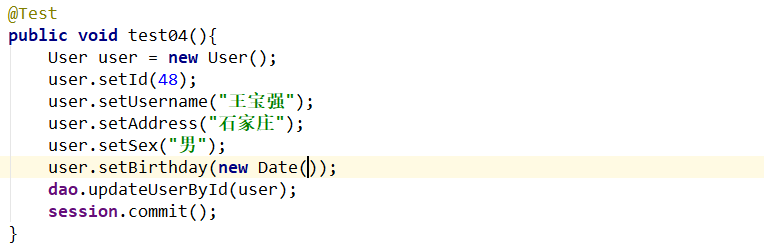
* dao

|  |
| --- |
| *//动态修改* **public void** updateUserById(User user); |

* dao的映射文件



* 测试

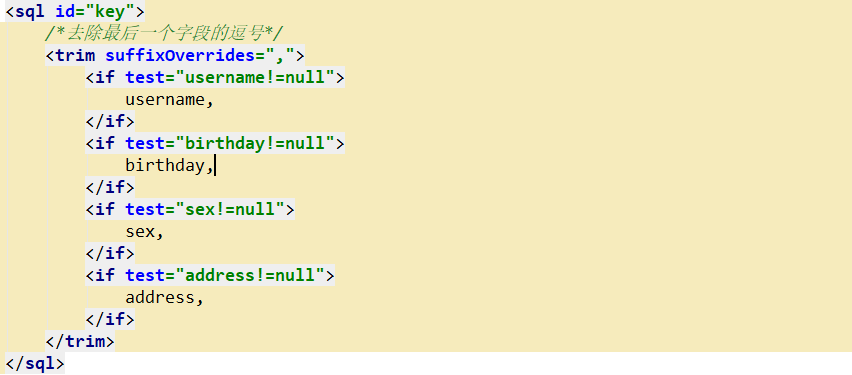


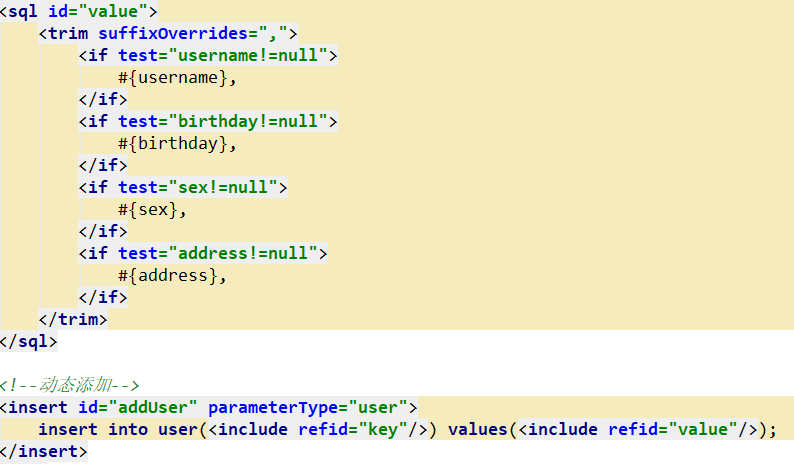
#### 2.1.3.5 动态添加

* dao

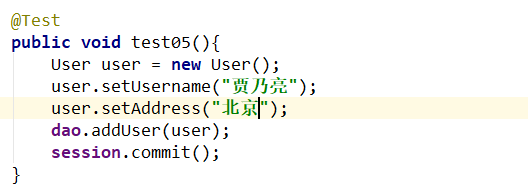
|  |
| --- |
| *//动态添加* **public void** addUser(User user); |

* dao的映射文件配置





* 测试



### 2.1.4 多表查询(重点 难点)

数据表与表之间的关系

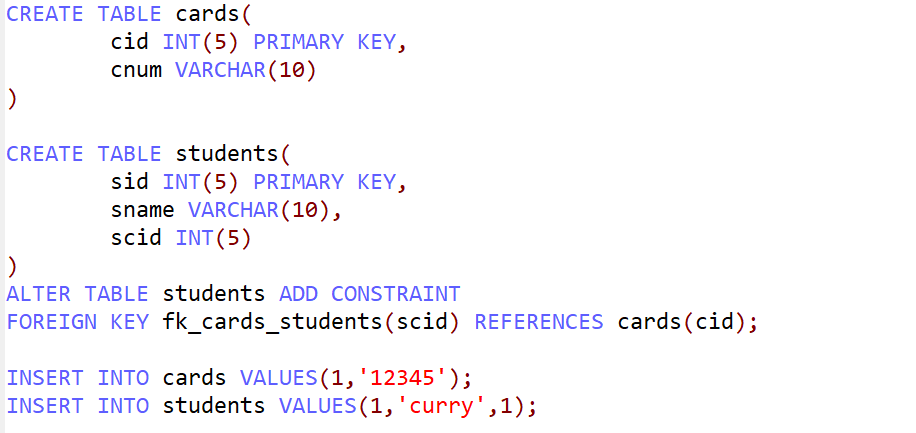
一对一的关系 学生表和 身份证表

一对多的关系 一个用户对应多个订单 通过在多表中建立外键

多对多的关系 学生表和课程表 通过中间表建立关系，中间表里面包含两张表的主键字段

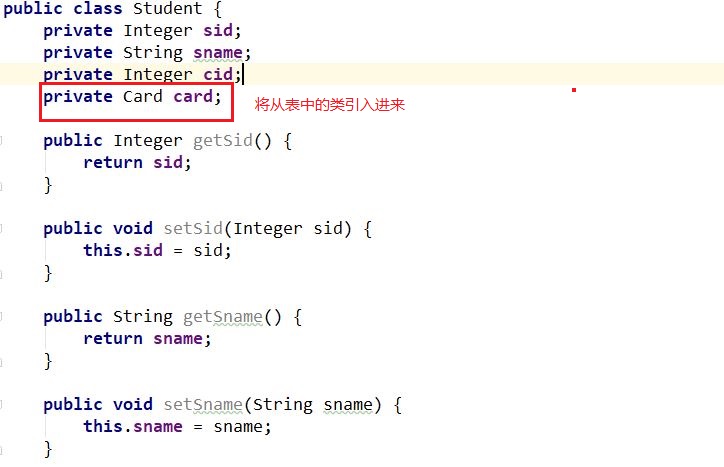
#### 2.1.4.1 一对一关系查询

* 准备数据表

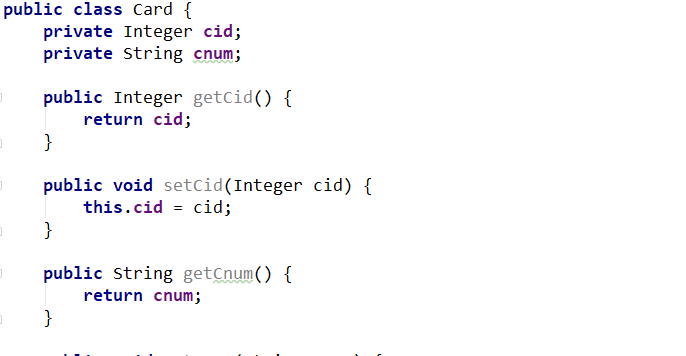


* 实体类的创建

对应的学生表的实体类



对应的cards表的实体类

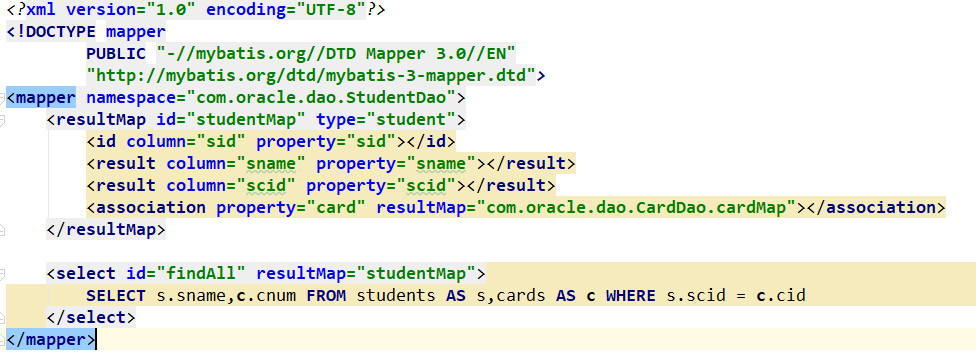


* dao

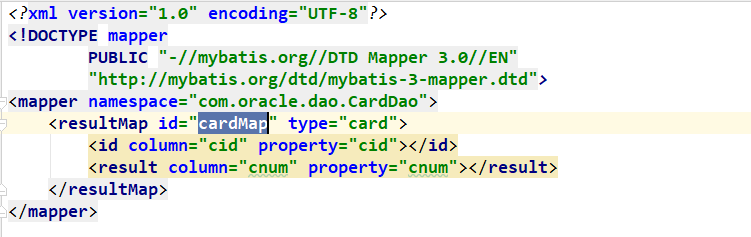


* 配置文件

studentDao.xml



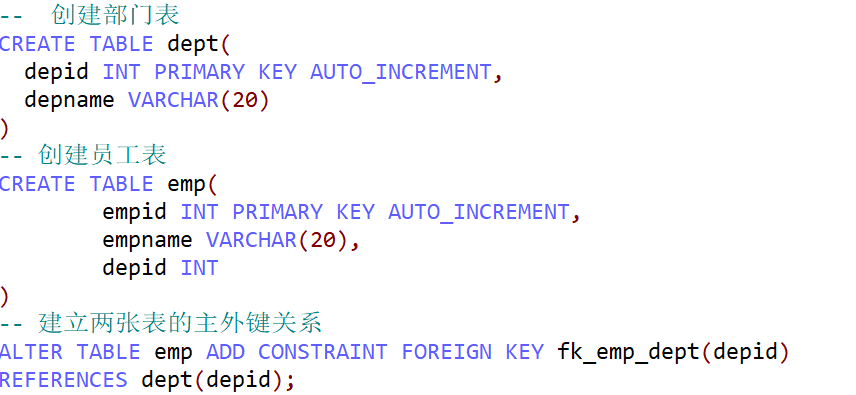
cardDao.xml



#### 2.1.4.2 一对多关系的多表查询

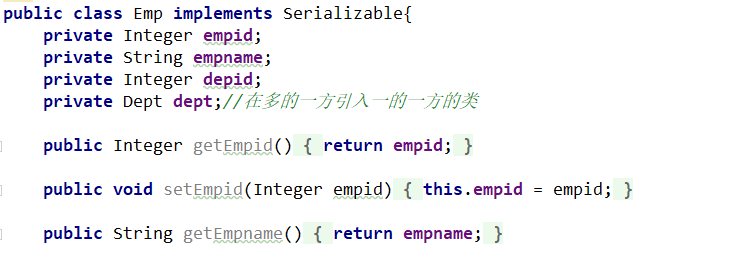
以员工表和部门表为例(一个部门对应有多个员工)

* 数据表的创建

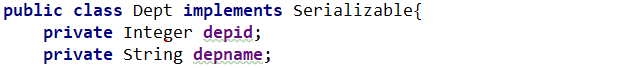


##### 2.1.4.2.1 查询员工及其部门信息(员工表就是主表 部门表就是从表)

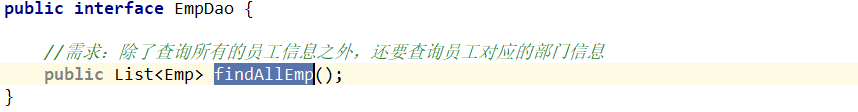
* Emp实体类



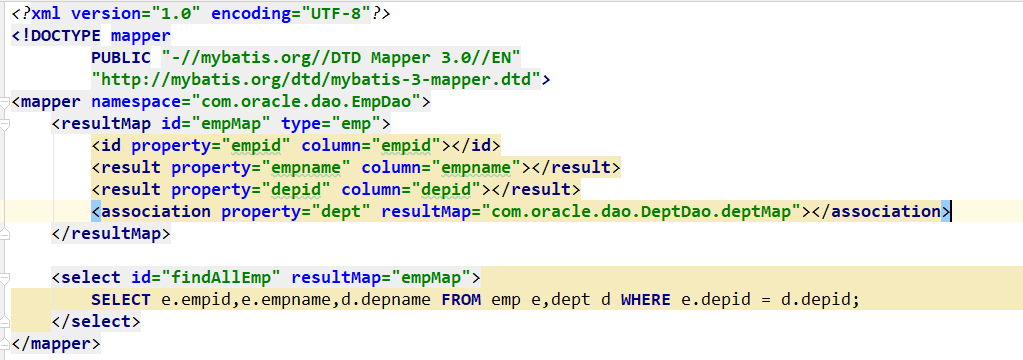
* Dept实体类



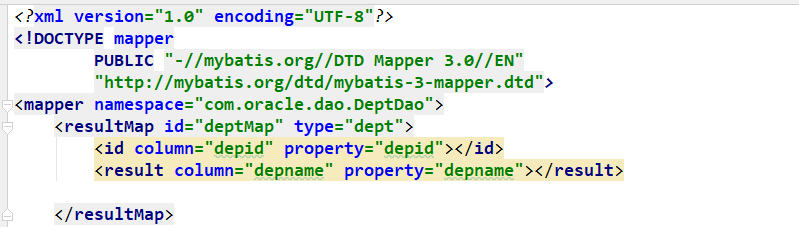
* EmpDao



* empDao的映射文件

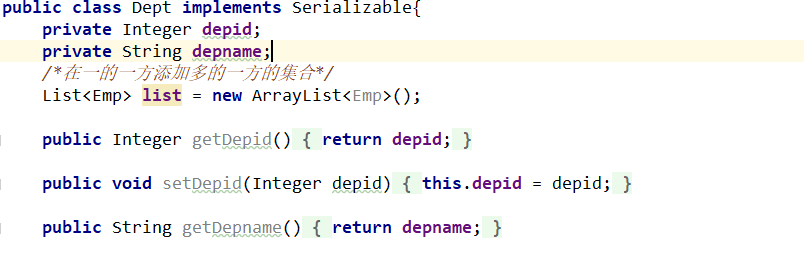


* DeptDao的映射文件

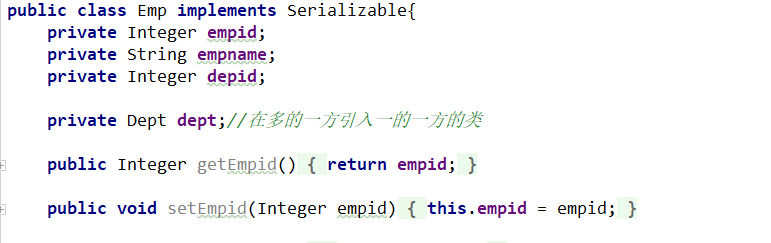


##### 2.1.4.2.2 查询所有的部门信息及其对应的员工信息(部门表是主表 员工表就是从表)

* 提供dept的实体类

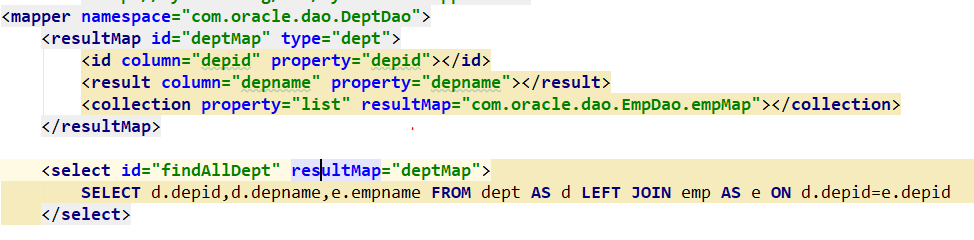


* 提供emp的实体类

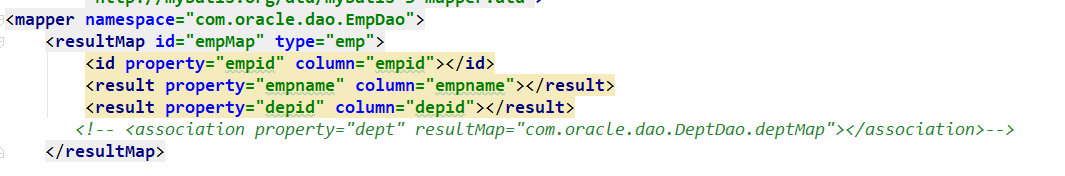


* 映射文件

DeptDao.xml

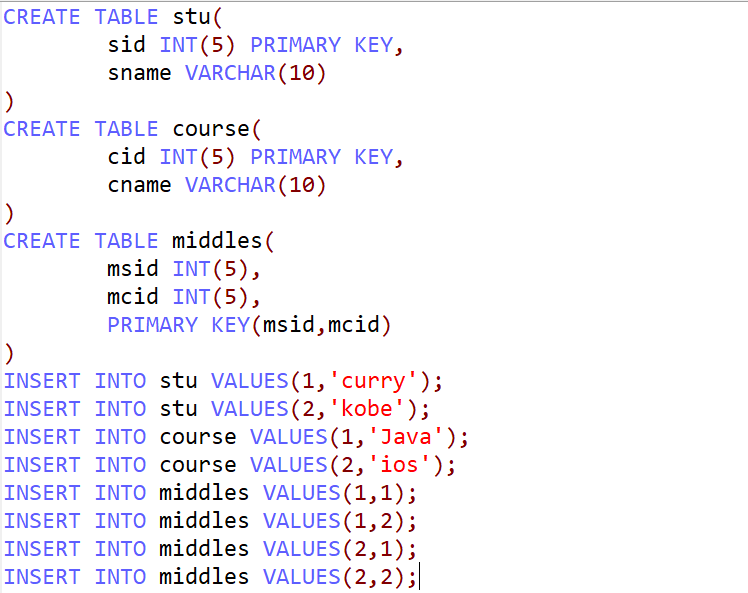


deptDao.xml



#### 2.1.4.3 多对多关系的查询

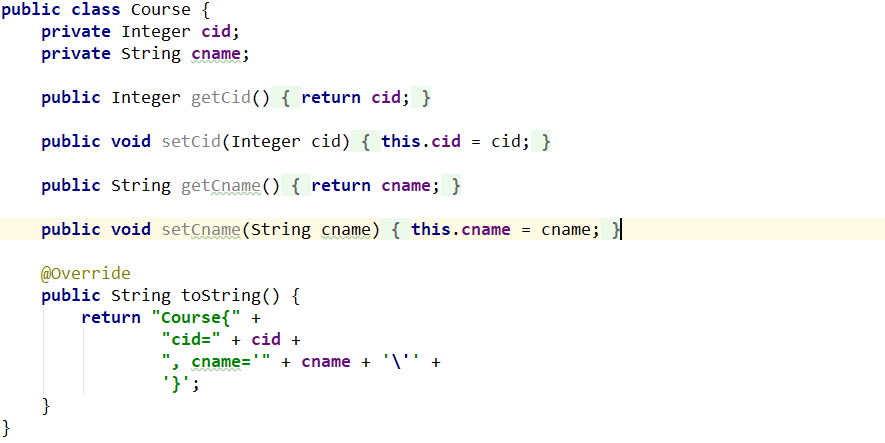
* 数据表的构建



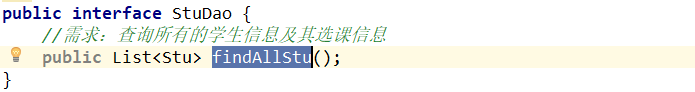
* 构建实体类
* Stu

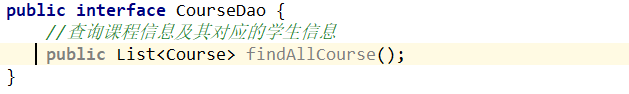


* Course

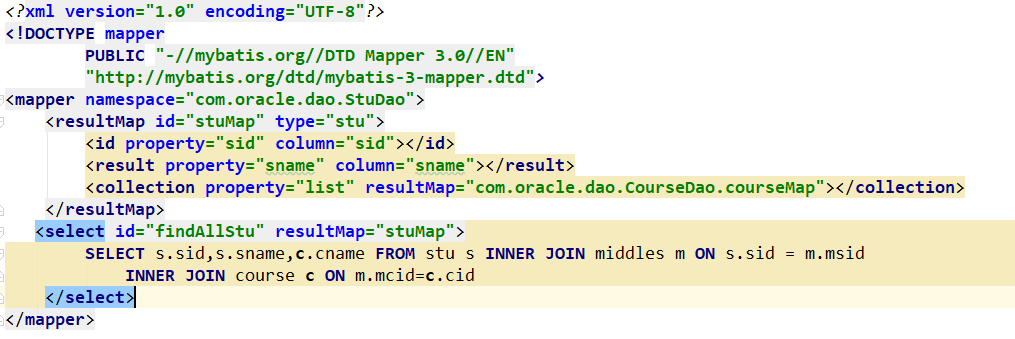


* Dao的构建

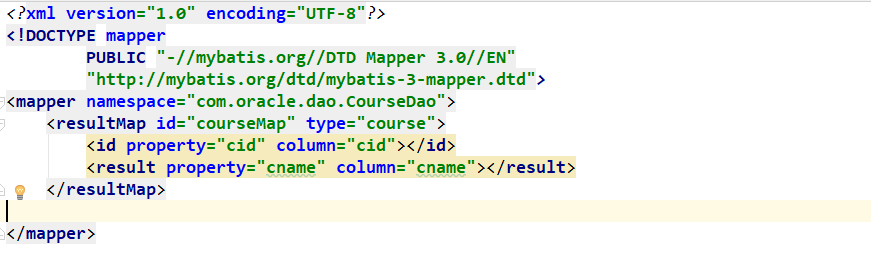




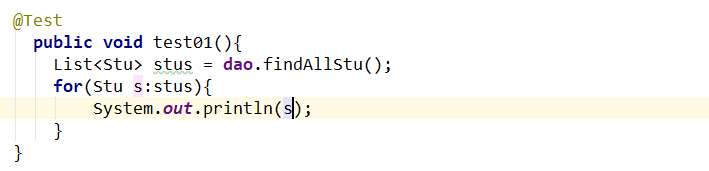
* Dao的xml映射文件
* StuDao.xml



* CourseDao.xml



* 测试类



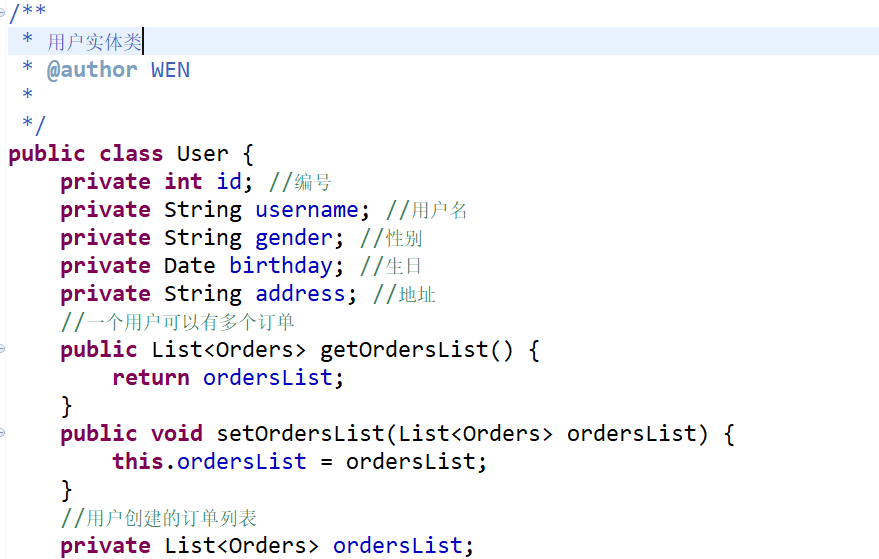
### 2.1.5 多对多查询的综合练习

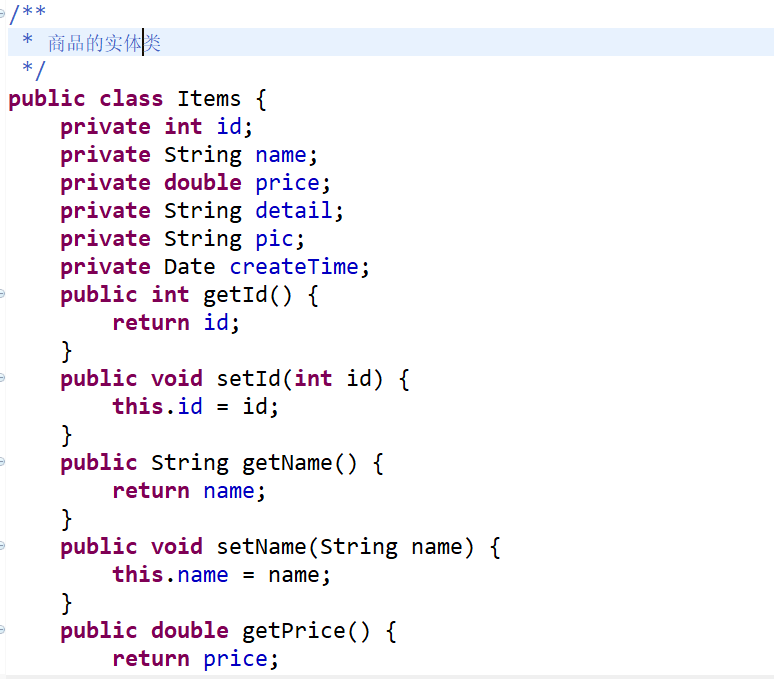
需求查询用户购买的商品信息

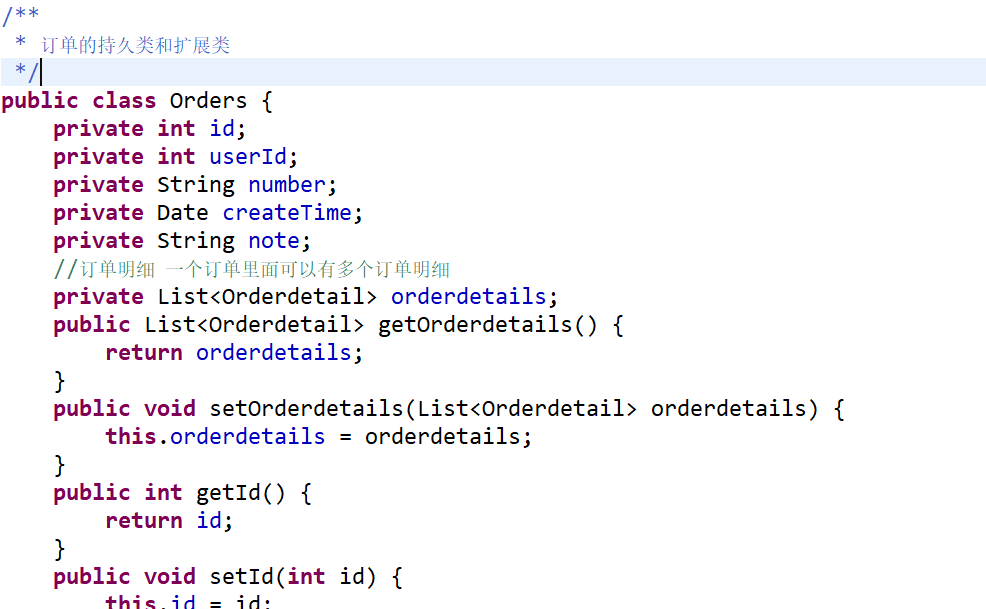
#### 2.1.5.1 表的创建

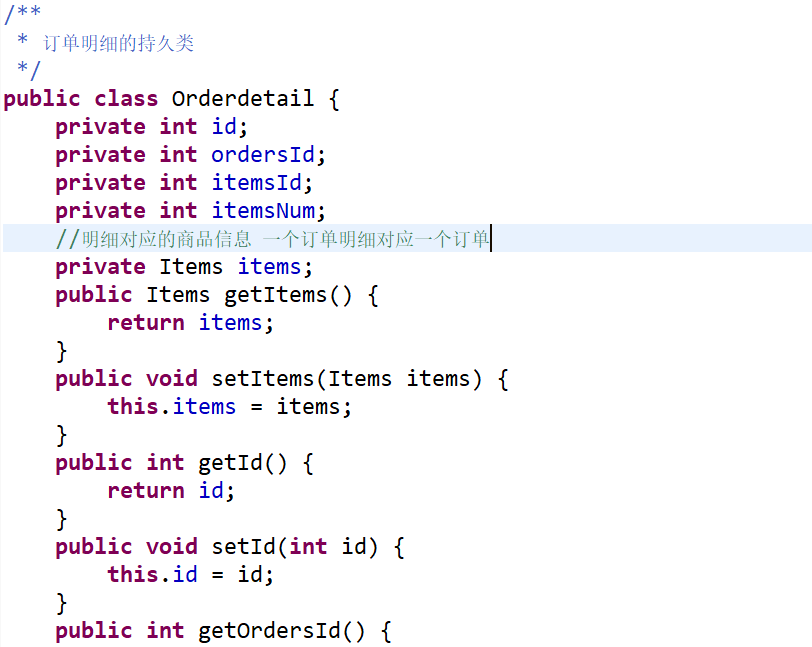
1. /\*items是商品表\*/
2. **CREATE** **TABLE** `items` (
3. `id` **INT**(11) NOT NULL AUTO\_INCREMENT,
4. `**name**` **VARCHAR**(32) NOT NULL COMMENT '商品名称',
5. `price` **FLOAT**(10,1) NOT NULL COMMENT '商品定价',
6. `detail` TEXT COMMENT '商品描述',
7. `pic` **VARCHAR**(64) **DEFAULT** NULL COMMENT '商品图片',
8. `createtime` DATETIME NOT NULL COMMENT '生产日期',
9. **PRIMARY** **KEY** (`id`)
10. ) ENGINE=INNODB AUTO\_INCREMENT=4 **DEFAULT** CHARSET=utf8;
11. /\*user是用户表\*/
12. **CREATE** **TABLE** `user` (
13. `id` **INT**(11) NOT NULL AUTO\_INCREMENT,
14. `username` **VARCHAR**(32) NOT NULL COMMENT '用户名称',
15. `birthday` **DATE** **DEFAULT** NULL COMMENT '生日',
16. `gender` **CHAR**(1) **DEFAULT** NULL COMMENT '性别',
17. `address` **VARCHAR**(256) **DEFAULT** NULL COMMENT '地址',
18. **PRIMARY** **KEY** (`id`)
19. ) ENGINE=INNODB AUTO\_INCREMENT=27 **DEFAULT** CHARSET=utf8;
20. /\*orders是订单表\*/
21. **CREATE** **TABLE** `orders` (
22. `id` **INT**(11) NOT NULL AUTO\_INCREMENT,
23. `user\_id` **INT**(11) NOT NULL COMMENT '下单用户id',
24. `number` **VARCHAR**(32) NOT NULL COMMENT '订单号',
25. `createtime` DATETIME NOT NULL COMMENT '创建订单时间',
26. `note` **VARCHAR**(100) **DEFAULT** NULL COMMENT '备注',
27. **PRIMARY** **KEY** (`id`),
28. **KEY** `FK\_orders\_1` (`user\_id`),
29. **CONSTRAINT** `FK\_orders\_id` **FOREIGN** **KEY** (`user\_id`) **REFERENCES** `user` (`id`) **ON** **DELETE** **NO** **ACTION** **ON** **UPDATE** **NO** **ACTION**
30. ) ENGINE=INNODB AUTO\_INCREMENT=6 **DEFAULT** CHARSET=utf8;
31. /\*orderdetail是订单明细表\*/
32. **DROP** **TABLE** IF EXISTS orderdetail;
33. **CREATE** **TABLE** `orderdetail` (
34. `id` **INT**(11) NOT NULL AUTO\_INCREMENT,
35. `orders\_id` **INT**(11) NOT NULL COMMENT '订单id',
36. `items\_id` **INT**(11) NOT NULL COMMENT '商品id',
37. `items\_num` **INT**(11) **DEFAULT** NULL COMMENT '商品购买数量',
38. **PRIMARY** **KEY** (`id`),
39. **KEY** `FK\_orderdetail\_1` (`orders\_id`),
40. **KEY** `FK\_orderdetail\_2` (`items\_id`),
41. **CONSTRAINT** `FK\_orderdetail\_1` **FOREIGN** **KEY** (`orders\_id`) **REFERENCES** `orders` (`id`) **ON** **DELETE** **NO** **ACTION** **ON** **UPDATE** **NO** **ACTION**,
42. **CONSTRAINT** `FK\_orderdetail\_2` **FOREIGN** **KEY** (`items\_id`) **REFERENCES** `items` (`id`) **ON** **DELETE** **NO** **ACTION** **ON** **UPDATE** **NO** **ACTION**
43. ) ENGINE=INNODB AUTO\_INCREMENT=5 **DEFAULT** CHARSET=utf8;

#### 2.1.5.2 实体类的构建







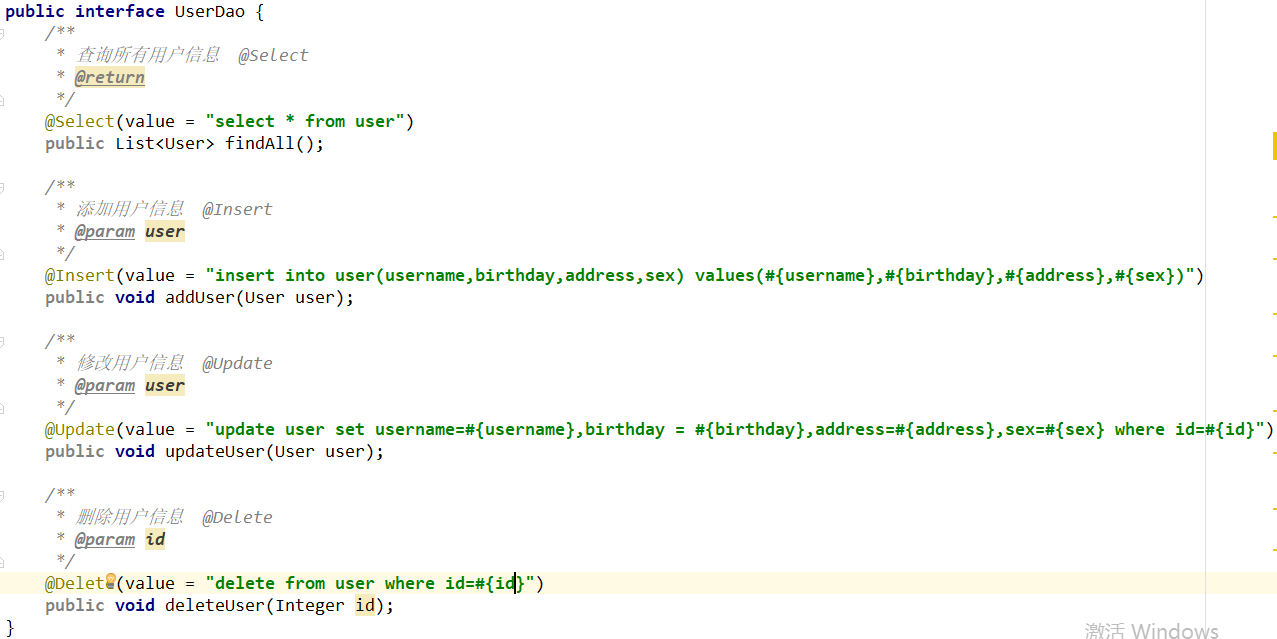


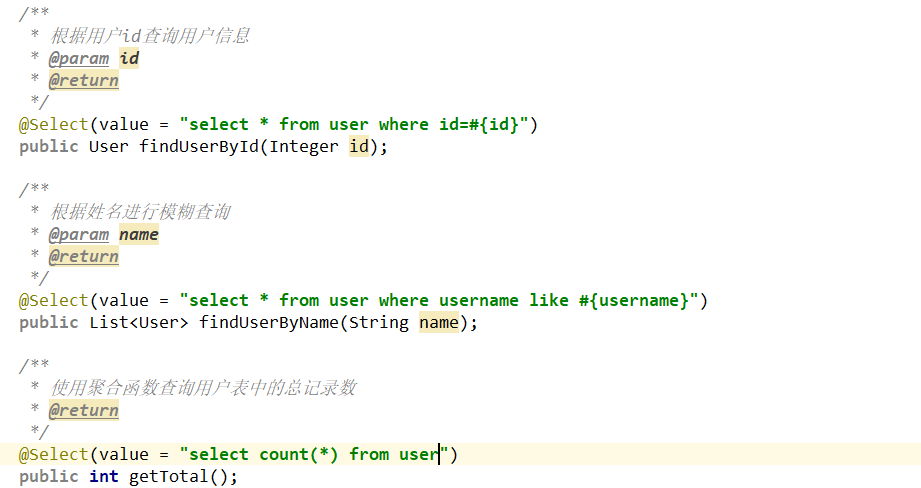
#### 2.1.5.3 配置文件

|  |
| --- |
| <?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.liuyanzhao.mybatis.mapper.OrdersMapper"*>  <!--查询用户购买的商品-->  <resultMap id=*"UserAndItemsResultMap"* type=*"com.mybatis\_day02.orders.User"*>  <!--用户信息-->  <id column=*"user\_id"* property=*"id"*></id>  <result column=*"username"* property=*"username"*></result>  <result column=*"gender"* property=*"gender"*></result>  <result column=*"address"* property=*"address"*></result>  <!--订单信息-->  <!--一个用户可以对应多个订单，故使用collection映射-->  <collection property=*"ordersList"* ofType=*"com.mybatis\_day02.orders.Orders"*>  <id column=*"id"* property=*"id"*></id>  <result column=*"user\_id"* property=*"userId"*></result>  <result column=*"number"* property=*"number"*></result>  <result column=*"createtime"* property=*"createTime"*></result>  <result column=*"node"* property=*"node"*></result>  <!--订单明细-->  <!--一个订单包括多个明细，故使用collection-->  <collection property=*"orderdetails"* ofType=*"com.mybatis\_day02.orders.Orderdetail"*>  <id column=*"orderdetail\_id"* property=*"id"*></id>  <result column=*"items\_id"* property=*"itemsId"*></result>  <result column=*"items\_num"* property=*"itemsNum"*></result>  <result column=*"orders\_id"* property=*"ordersId"*></result>  <!--商品信息-->  <!--一个订单明细对应一个商品-->  <association property=*"items"* javaType=*"com.mybatis\_day02.orders.Items"*>  <id column=*"items\_id"* property=*"id"*></id>  <result column=*"items\_name"* property=*"name"*></result>  <result column=*"items\_price"* property=*"price"*></result>  <result column=*"items\_detail"* property=*"detail"*></result>  </association>  </collection>  </collection>  </resultMap>  <select id=*"findUserAndItemsResultMap"* resultMap=*"UserAndItemsResultMap"*>  SELECT  orders.\*,  user.username,  user.gender,  user.address,  orderdetail.id orderdetail\_id,  orderdetail.items\_id,  orderdetail.items\_num,  orderdetail.orders\_id,  items.name items\_name,  items.detail items\_detail,  items.price items\_price  FROM  orders,  user,  orderdetail,  items  WHERE orders.user\_id=user.id AND orders.id = orderdetail.orders\_id AND orderdetail.items\_id=items.id  </select>  </mapper> |

### 3.1.1 mybatis的注解

#### 3.1.1.1 单表增删改查的注解





使用注解需要注意的事项：

1.千万不要将使用配置文件的方式和注解的方式一起使用，否则会报错。要么只能用配置文件的方式，要么只能使用注解的方式

2.使用注解的方式进行配置，在核心配置文件里面也需要将注解配置的内容引入进去

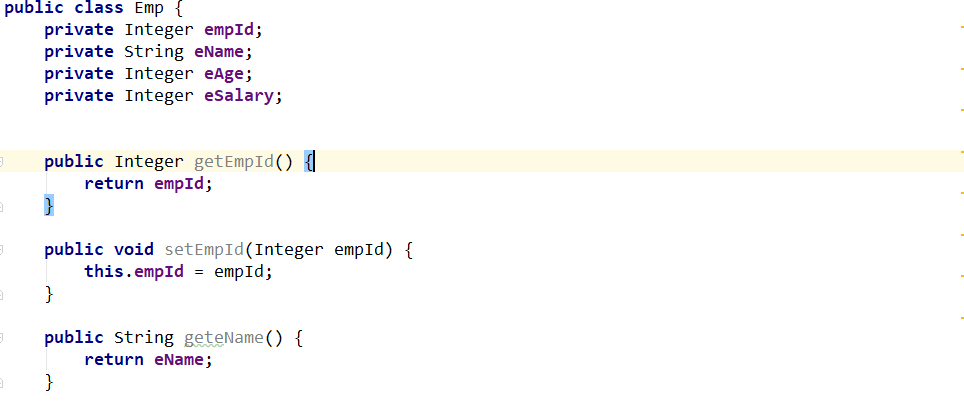


#### 3.1.1.2 实体类和数据表不一致的问题

* 数据表

|  |
| --- |
| CREATE TABLE emp(  id INT PRIMARY KEY AUTO\_INCREMENT,  empname VARCHAR(30),  empage INT,  salary INT  ) |

* 实体类



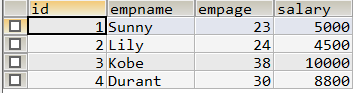
* Dao

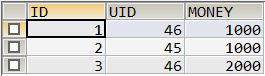


#### 3.1.1.3 Mybatis的多表查询注解

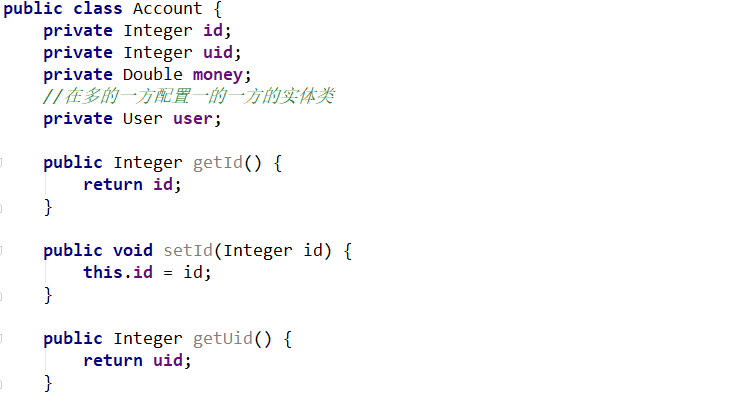
##### 3.1.1.3.1 一对一关系

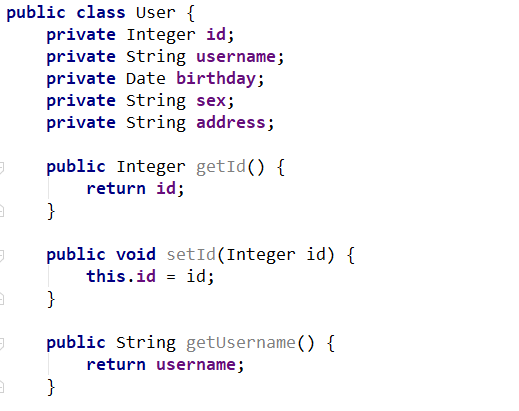
* 数据表(user、account)



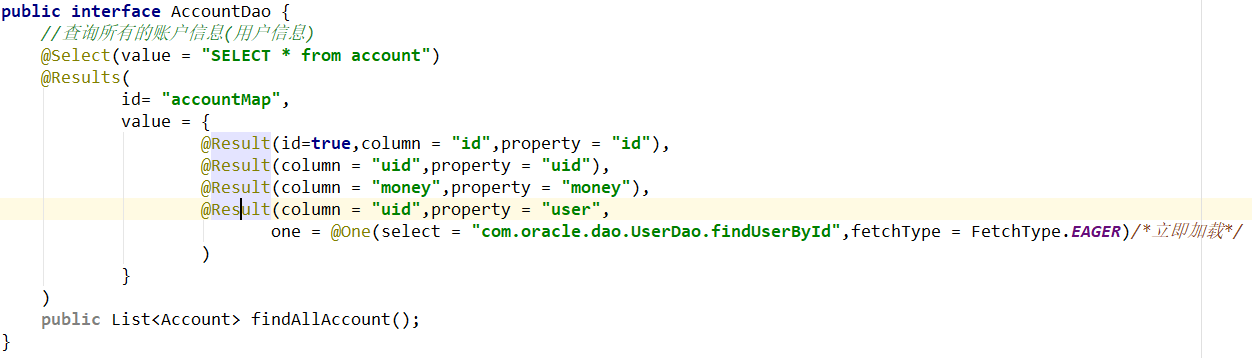


* 实体类

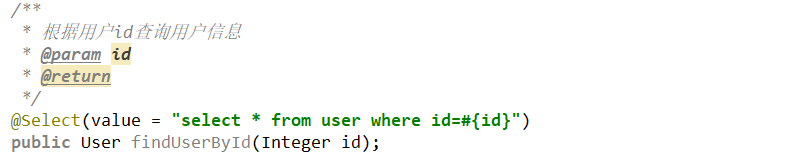




* dao



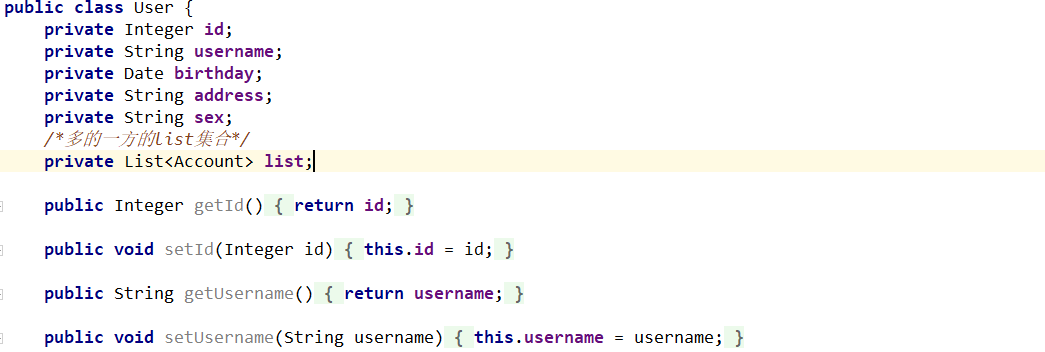
还需要在UserDao里面配置根据用户id查询用户信息的方法



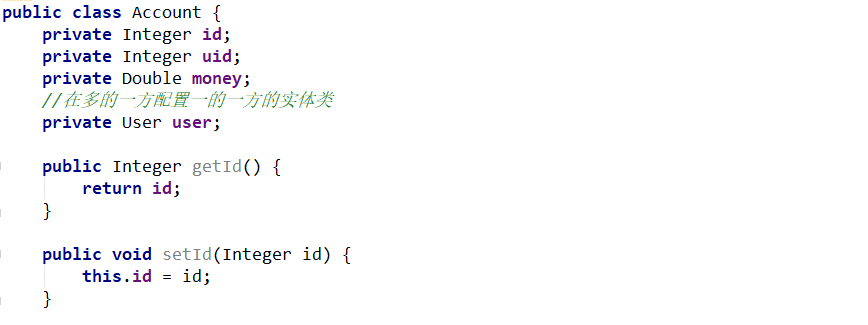
##### 3.1.1.3.2 一对多

* 实体类

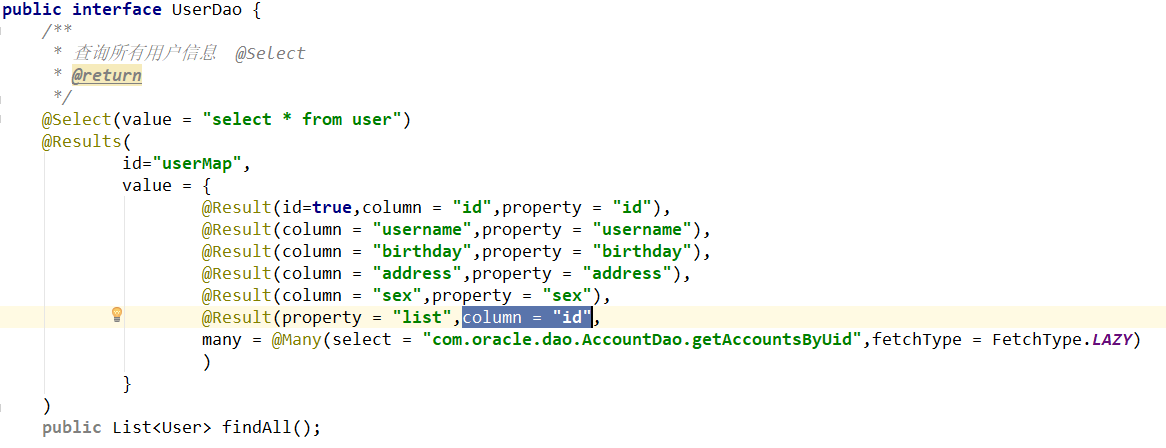
User:



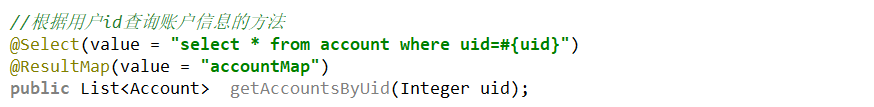
Account:



* dao

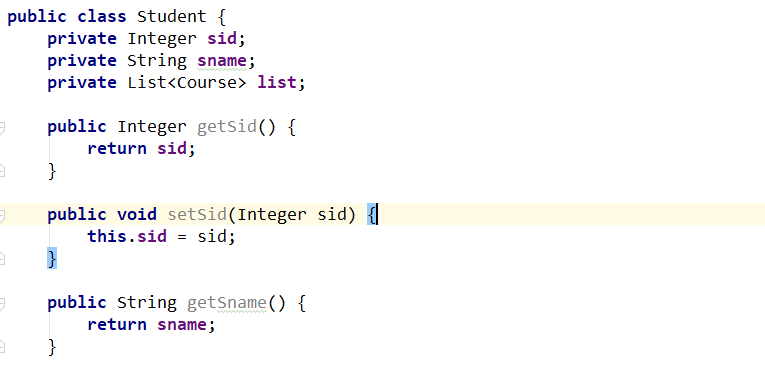


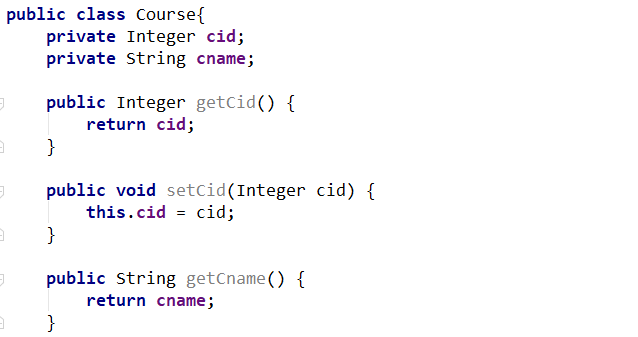
还需要在AccountDao里面配置 根据用户id查询账户信息的方法



##### 3.1.1.3.3 多对多的关系

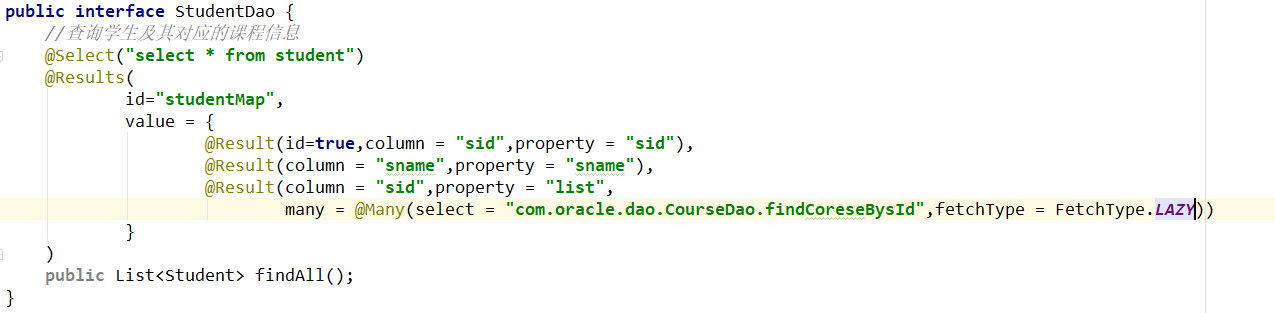
* 实体类



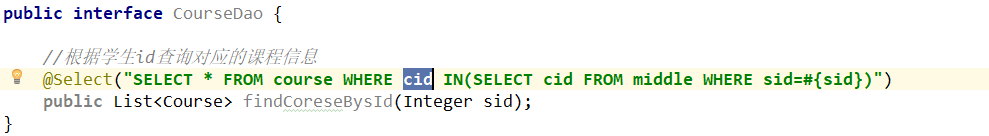


* Dao

StudentDao



CourseDao

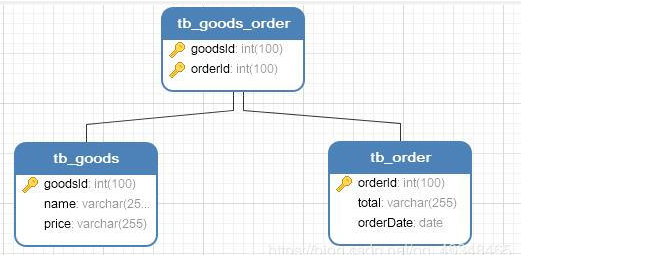


* 测试略

### 3.1.2 练习

#### 3.1.2.1 课程表和学生表，查询课程及其对应的选课的学生信息

#### 3.1.2.2 商品和订单



根据商品Id查找所有相关的订单