MyBatis 第一天基础知识

1. 基础知识（重点 内容比较多）

* 对原生态的jdbc程序 单独使用有哪些问题
* MyBatis框架的原理(掌握)

* MyBatis的入门程序

用户的增删改查

* MyBatis 开发DAO的两种方式：
* 原始dao开发(程序要编写dao接口和实现类)（掌握）
* MyBatis的mapper接口(相当于dao接口) 代理开发的方法 （掌握 重点）
* MyBatis的配置详解 SqlMapConfig.xml
* MyBatis的核心 ：
* MyBatis的输入映射(掌握)
* MyBatis的输出映射(掌握)
* MyBatis的动态SQL(掌握)

1. 高级知识

订单商品的数据模型的分析

高级结果集映射(一对一 、一对多、多对多)

MyBatis的延迟加载

MyBatis的查询缓存(一级缓存、二级缓存)

Mybatis和Spring整合(掌握)

Mybatis的逆向功能

# MyBatis框架基础知识

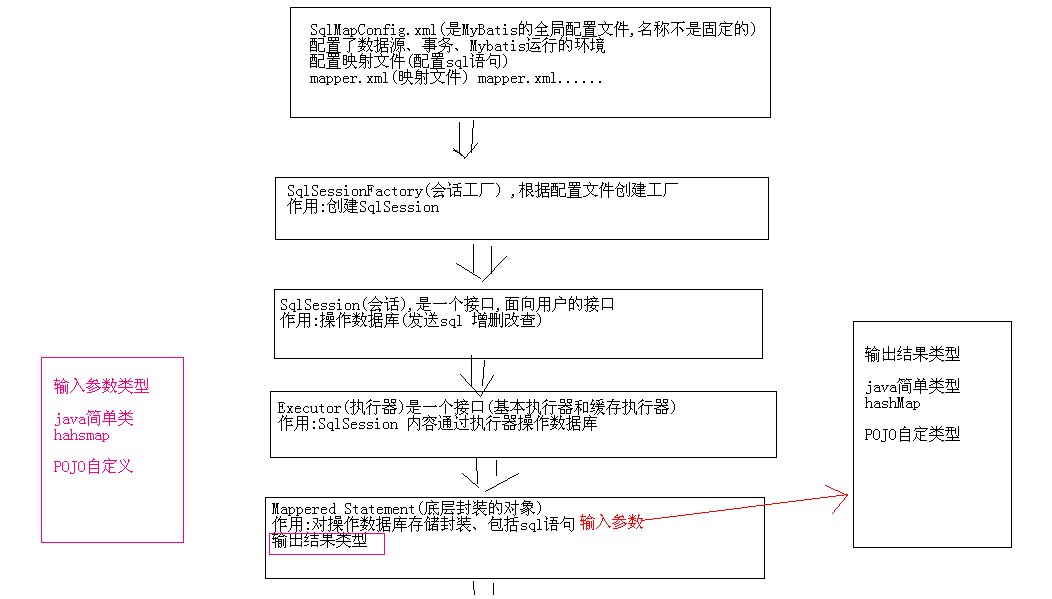
## MyBatis的基础概念

**1)mybatis的什么?**

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

MyBatis让程序将主要精力放在sql上，通过mybatis提供的映射方式，可以自由灵活的生成满足条件的sql（半自动化的orm框架）

## MyBatis框架的流程



# Mybatis入门程序

## 需求

根据用户的id(主键) 查询用户信息

根据用户名模糊查询用户信息

添加用户

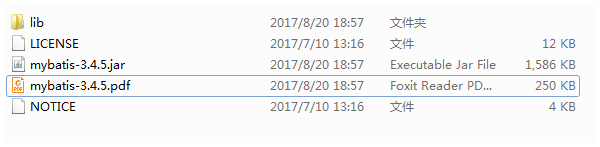
删除用户

更新用户

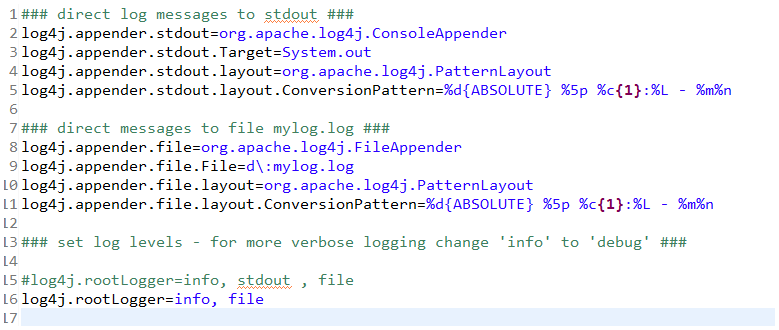
## 搭建开发环境

从mybatis官网下载最新的jar包

https://github.com/mybatis/mybatis-3/releases



## 导入相关的记录日志文件

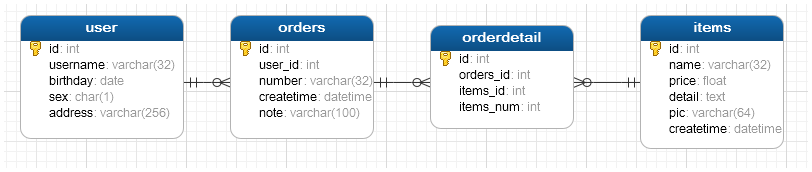


## 编写SqlMapConfig.xml

需要在mybatis的配置文件中配置 运行环境 包括数据源 和事务等相关的内容

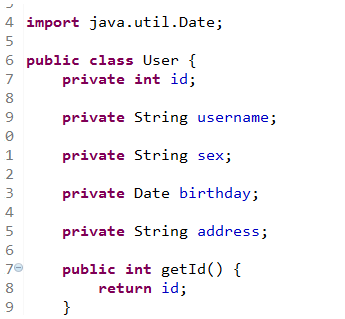


## 创建数据表

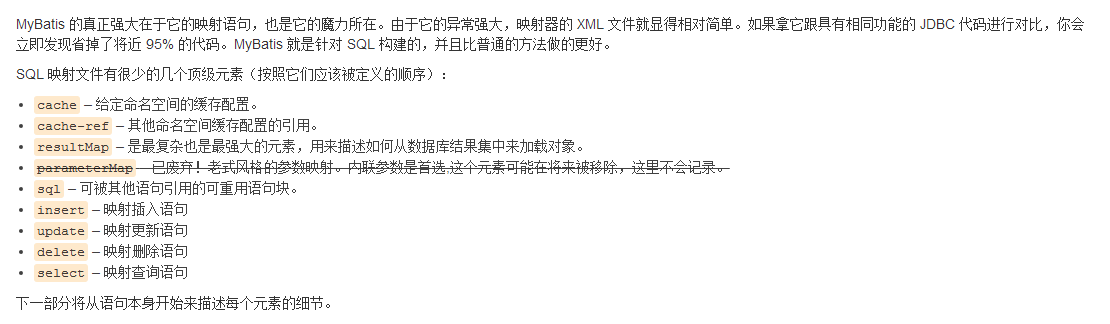


## 根据id(主键)查询用户信息

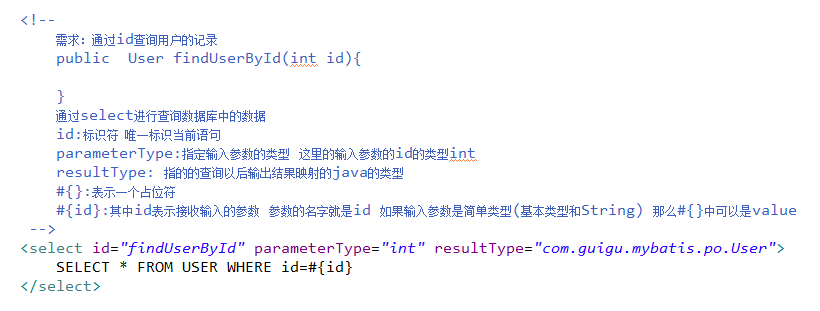
### 创建PO类



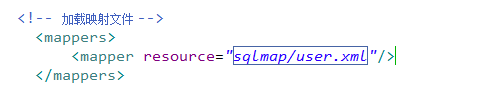
### 映射文件







### 在sqlMapConfig.xml中加载映射文件

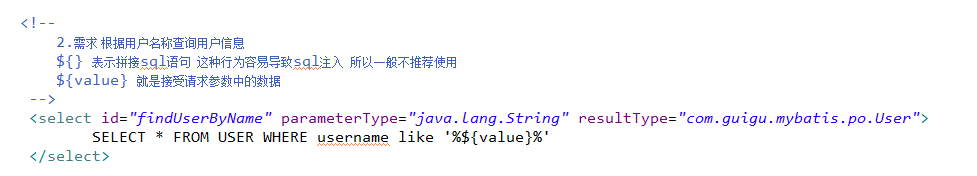


### 编写测试代码

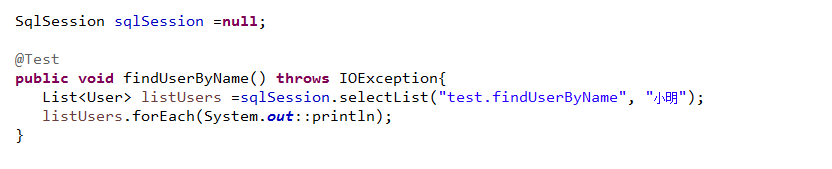


## 根据名字模糊查询用户信息

### 映射文件



### 测试代码



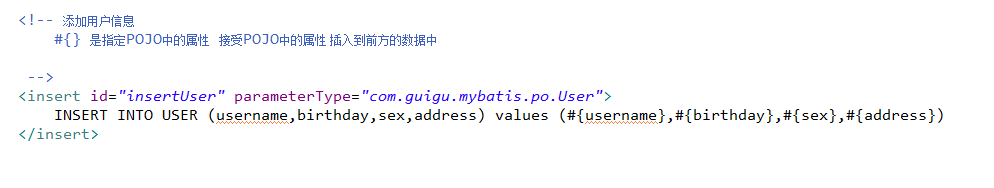


### Select的其他属性

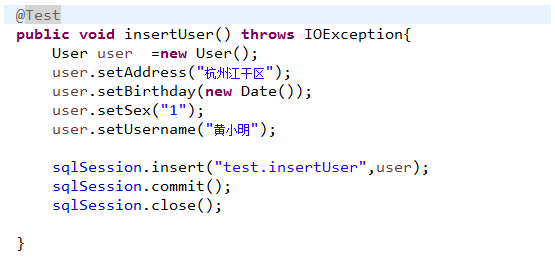


## 添加用户

### 映射文件



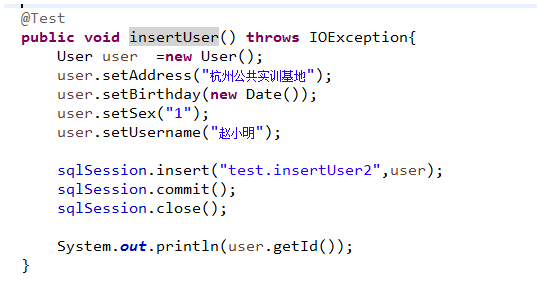
### 测试代码



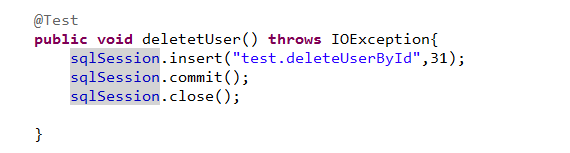
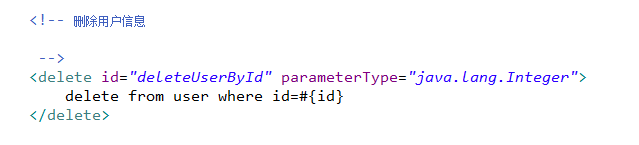
### 自增主键的获取

映射文件

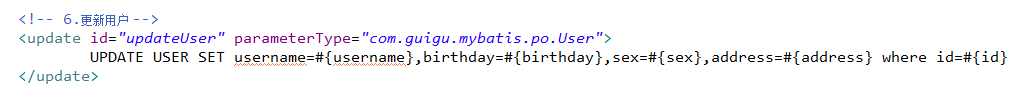


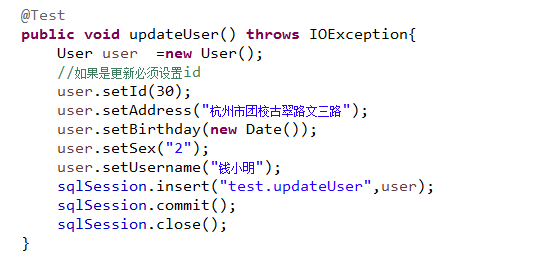


## 删除用户



## 更新用户





## 总结

### parameterType

在映射文件通过parameterType 指定输入参数的类型

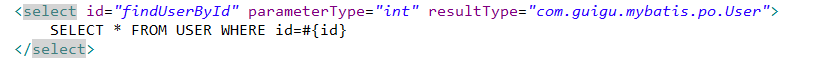
### resultType

在映射文件通过 resultType指定输出参数的类型

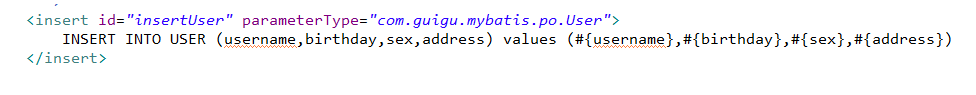
### #{ } 和${ }

#{ } 表示一个占位符 #{ } 接受输入参数 类型可以是简单类型、POJO类型 、HashMap

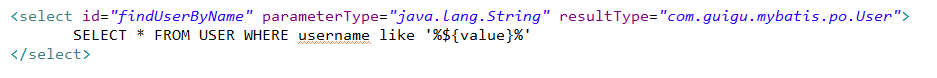
下方的案例#{id} 是获取请求参数的中的id



下方的案例 #{username} 是获取 请求参数中的POJO中的username



${ }是拼接内容



### selectOne 和selectList

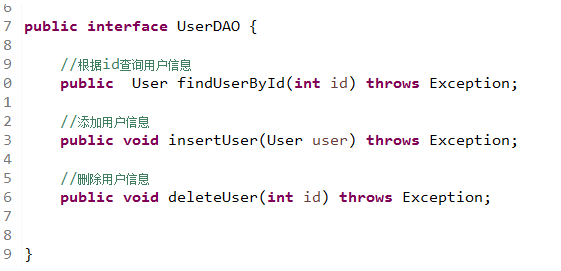
selectOne表示查询一条记录 进行映射，如果使用selectOne可以实现的功能那么可以使用selectList进行替代。 反之报错

报错：TooManyResultsException

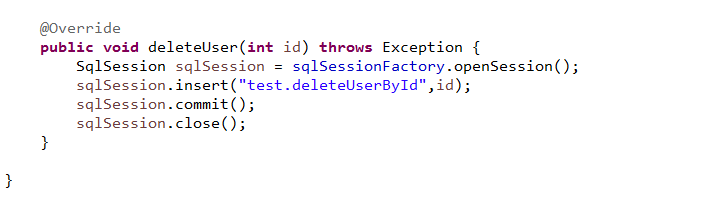
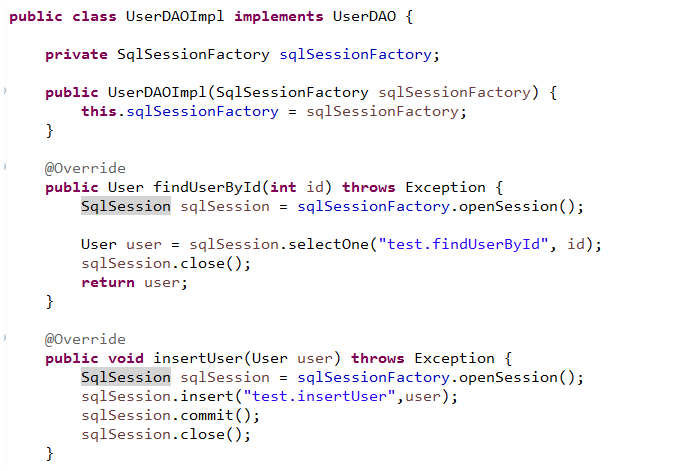
# MyBatis开发DAO的方式

## 原始DAO开发方式(DAO接口和DAO实现类)

### Dao接口



### Dao接口实现类



### 测试代码



### 总结原始DAO开发问题

1.、dao接口实现类中存在大量重复的代码。能否将这些重复的代码提取

2．、调用sqlSession方法将statement的id进行了硬编码

3.、调用的方法传入变量 由于sqlSession方法使用泛型 在传入参数容易导致报错。

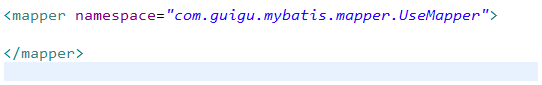
## Mapper代理方法(只需要编写Mapper接口)

程序员只需要编写mapper.xml映射文件和对应的接口即可

在编写的过程中要遵循开发规范。

开发规范有如下内容：

1、在mapper.xml中namespace 等于mapper接口 地址

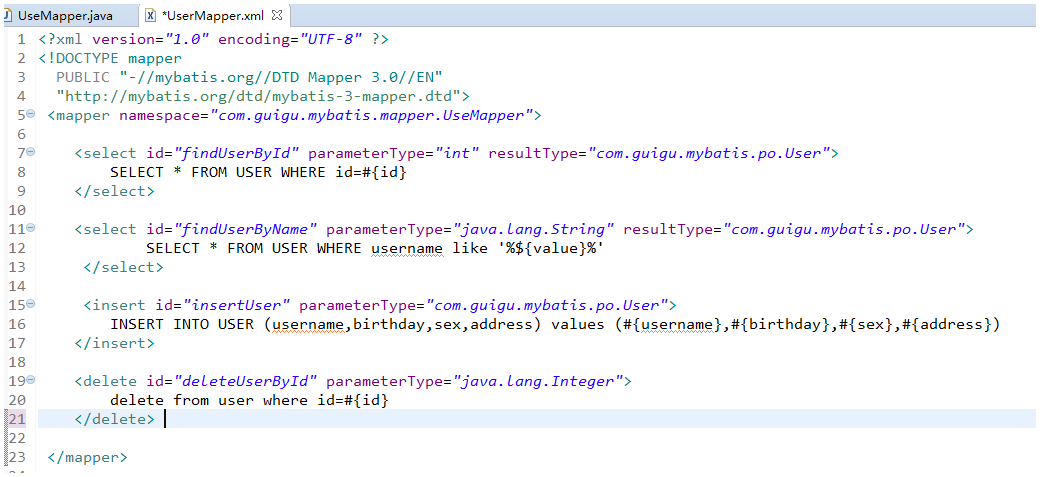


2、mapper.java 接口中方法名和mapper.xml中statement的id是一致的

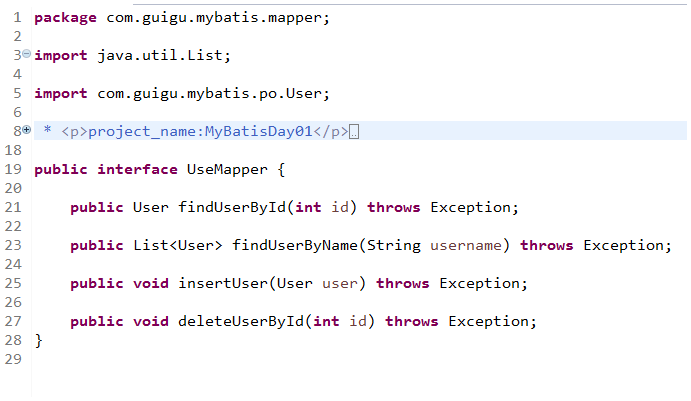
3、mapper.java接口中的输入参数 和mapper.xml中statmenetd的parameterType指定的类型是一致的

4、mapper.java接口中的方法返回值和 和mapper.xml中statmenetd的resultType指定的类型是一致的

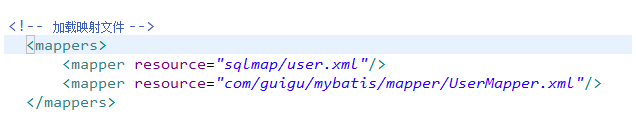
### UserMapper.xml



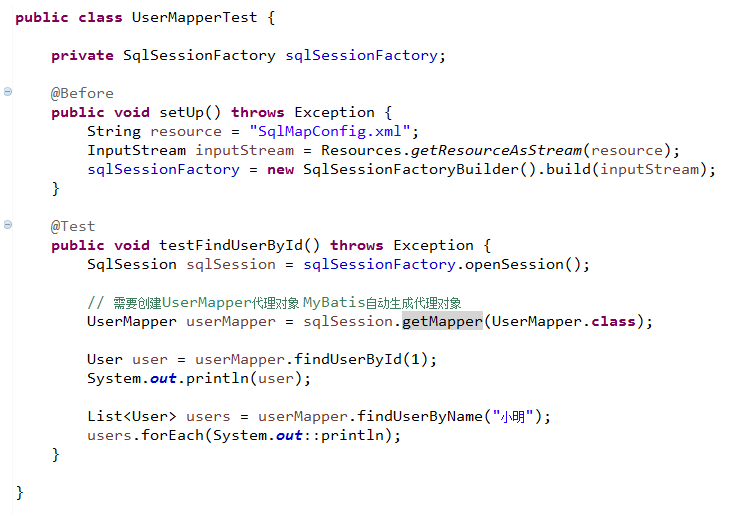
### UserMapper.java



### 在sqlMapConfig.xml中加载UserMapper.xml



### 测试代码



# SqlMapConfig.xml



## properties 属性



Properties特性

注意： MyBatis将按照以下顺序加载属性

* 在Properties元素体内定义的属性首先被加载
* 然后读properties中的resource或url加载属性，它会覆盖掉之前的内容
* 最后读取paramterType传递的属性

## Setting设置

这是 MyBatis 中极为重要的调整设置，它们会改变 MyBatis 的运行时行为。下表描述了设置中各项的意图、默认值等。

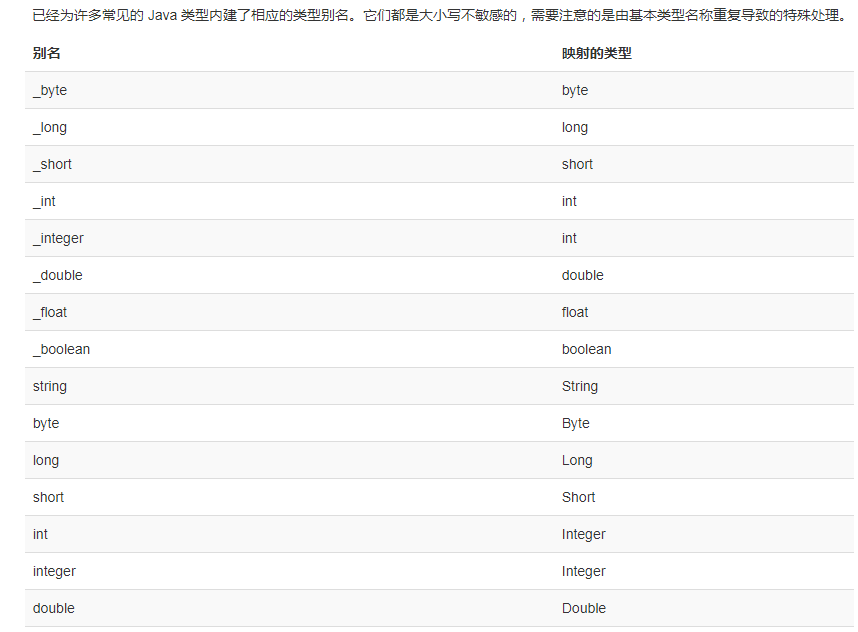




## typeAliases别名

类型别名是为 Java 类型设置一个短的名字。它只和 XML 配置有关，存在的意义仅在于用来减少类完全限定名的冗余。例如:

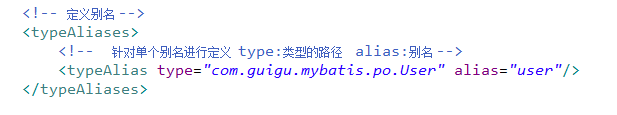
### MyBatis默认支持的别名

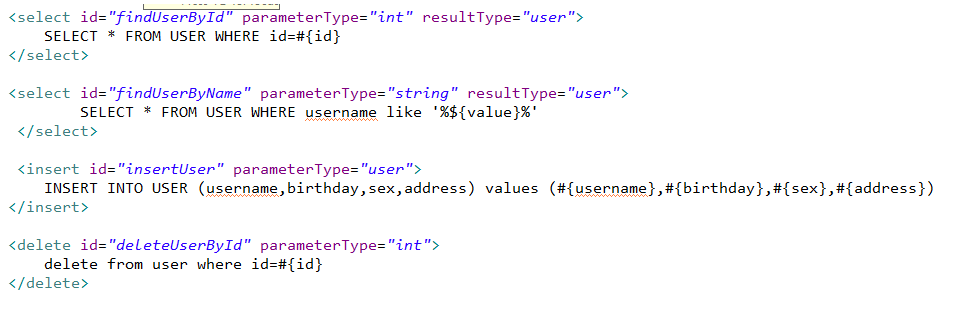




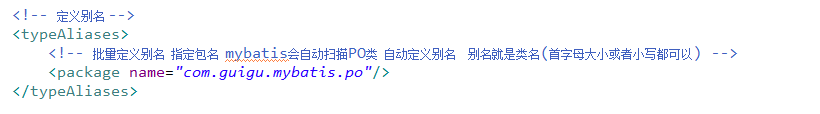
### 自定义别名

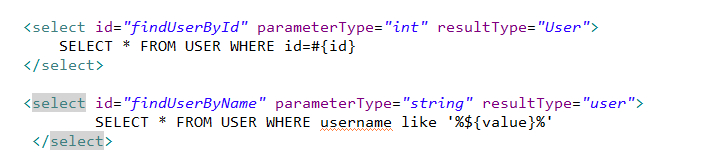
#### 单个别名的定义



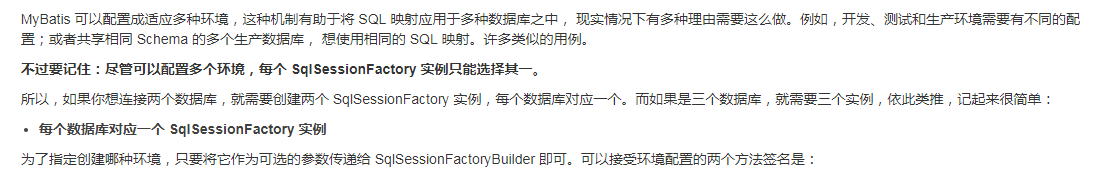


#### 批量定义别名





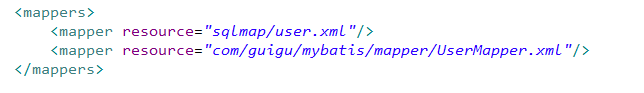
## Environments配置环境



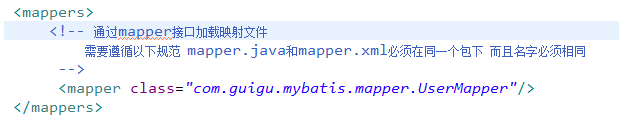


## Mappers(映射器)

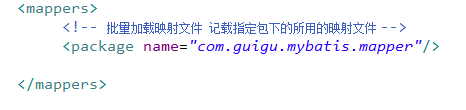
### 通过resource加载单个映射文件



### 通过mapper接口加载单个mapper



### 批量加载mapper(推荐使用)



# 输入映射和输出映射

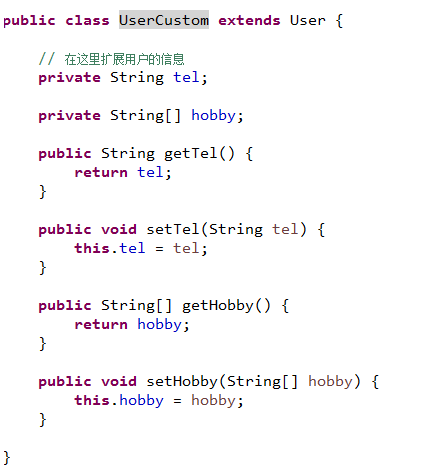
## 输入映射

### 传递POJO的包装对象

需求：完成用户信息的综合查询，需要传入的查询条件很复杂(可能包含用户信息，其他信息，比如商品信息、订单信息)

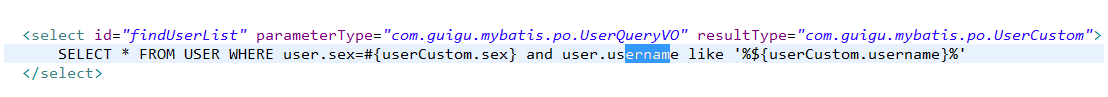
### 自定义包装类型POJO

针对上方的需求 建议使用自定义包装类的POJO





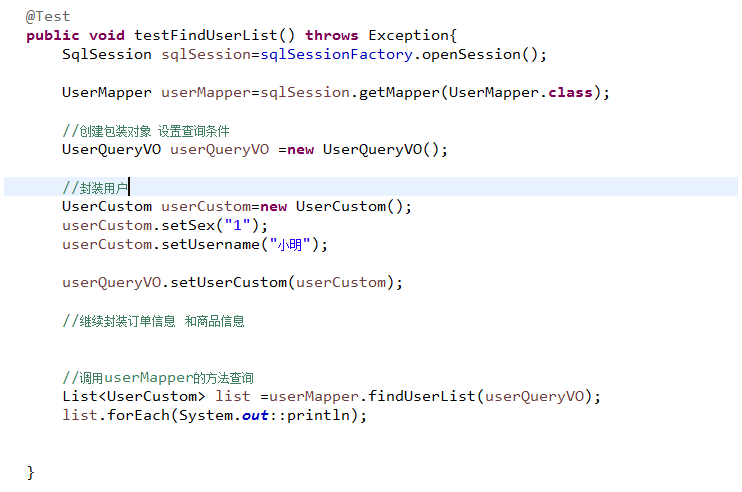
### UserMapper.xml



### UserMapper.java



### 测试代码



## 输出映射

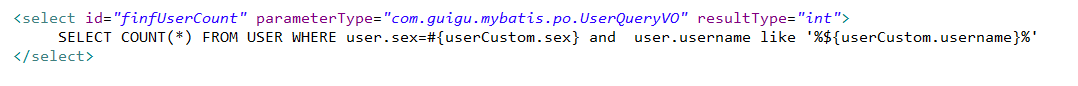
### resultType

只有查询出来的列名和POJO中的属性名完全一致才能完成映射。

#### 需求

用户信息综合列表的查

#### Usermapper.xml



#### Usermapper.java



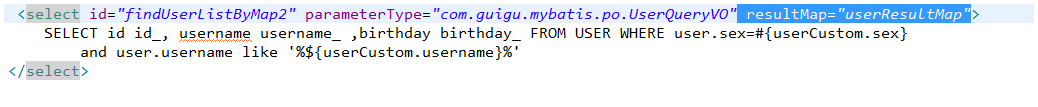
#### 测试代码

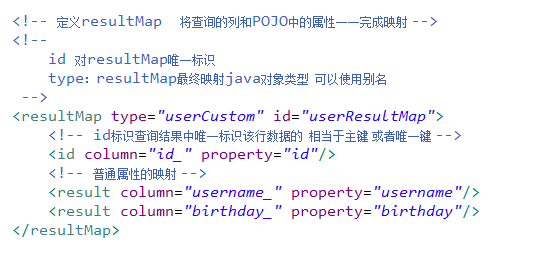


### ResultMap

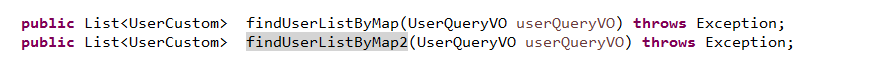
当查询出数据库的列名和POJO中的列名不一致，无法完成映射。 通过ResultMap可以完成一一对应映射。

#### UserMapper.xml





#### userMapper.java



#### 测试代码



# 动态SQL

## 什么是动态SQL

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

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

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

if

choose (when, otherwise)

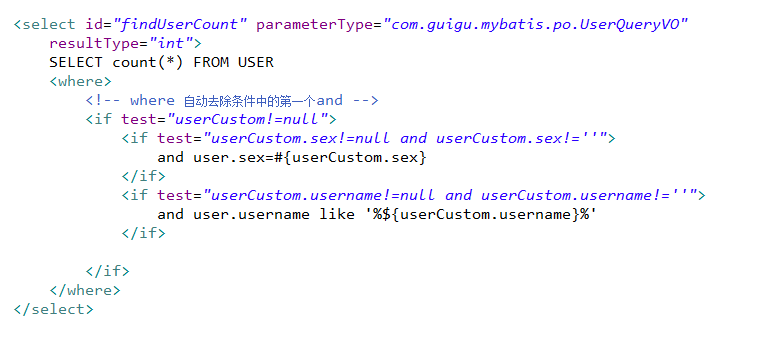
trim (where, set)

foreach

## 需求

用户信息的综合查询 和用户信息查询列表 对输入查询条件进行判断 当条件不为空在执行查询。

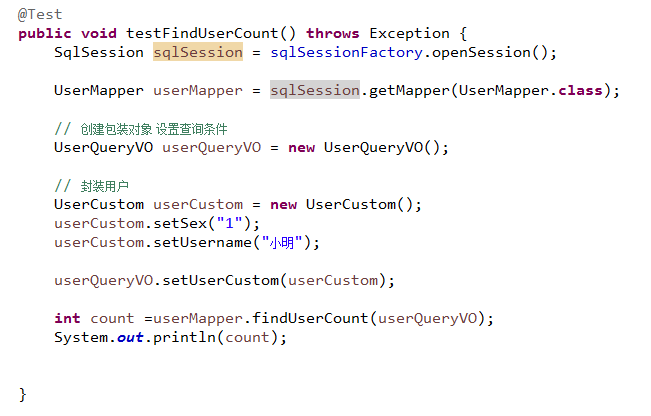
### UserMapp.xml



### UserMapp.java

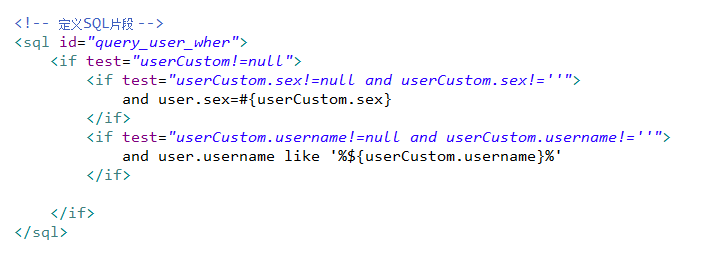


### 测试代码

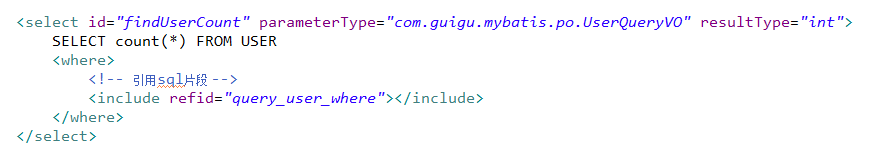


## SQL片段

### UserMapper.xml



### 引用SQL片段

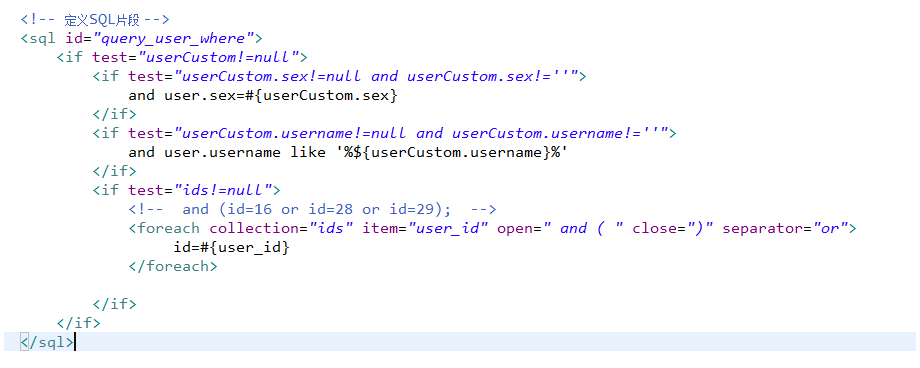


## Foreach SQL

### 需求

在用户查询列表和查询用户总数的时候 需要增加查询条件 根据多个用户的id

### UserMapper.xml



### UserMapper.java



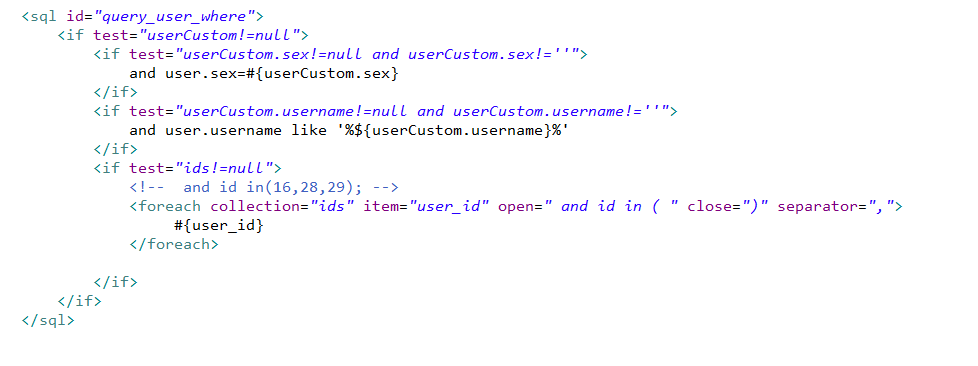
### UserQueryVO



### 测试代码



### 另外一种形式的操作



# MyBatis映射

## 一对一查询

### 需求

查询订单信息 关联查询创建订单的用户信息 两种形式实现resultType和resultMap

### ResultType

#### Sql语句

确定主表 ：订单表 orders

确定关联表 ：用户表 user

select orders.\*,user.username,user.sex,user.address

FROM

orders,

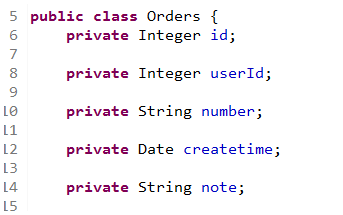
USER

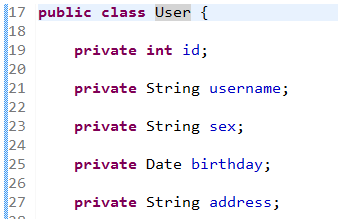
where orders.user\_id=user.id

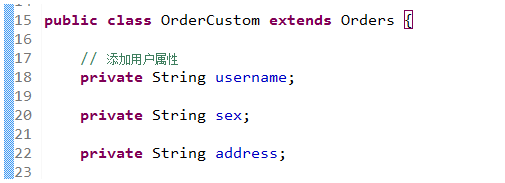
#### sqlMapConfig.xml



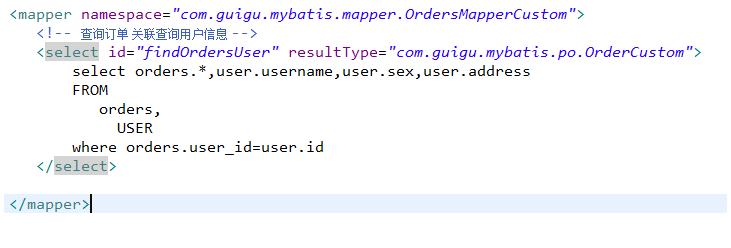
#### 创建POJO



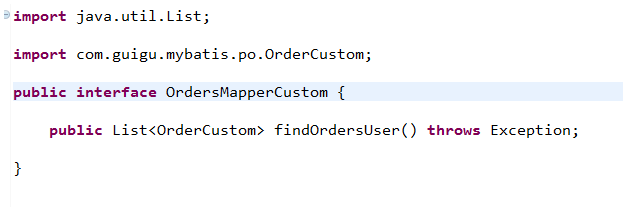




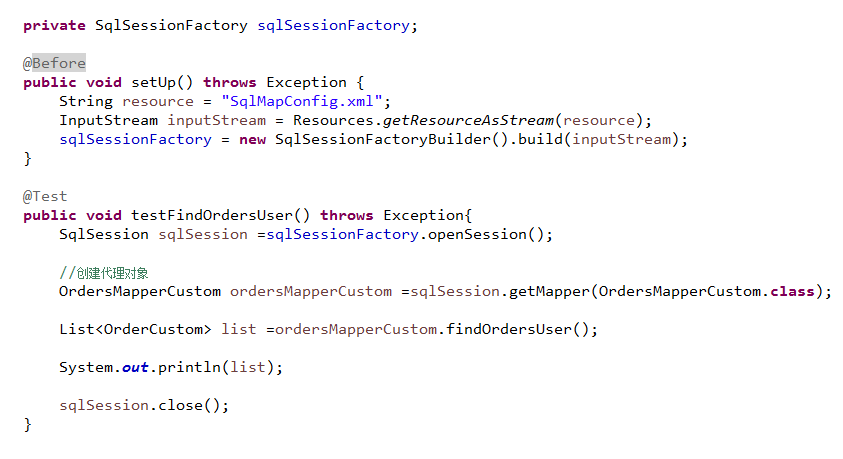
#### Mapper.xml



#### Mapper.java



#### 测试代码



### ResultMap

#### Sql语句

select orders.\*,user.username,user.sex,user.address

FROM

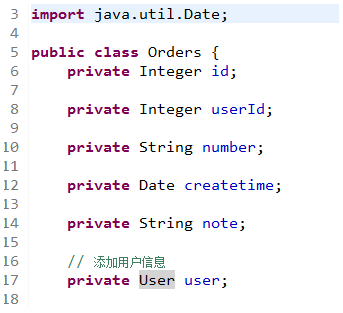
orders,

USER

where orders.user\_id=user.id

#### 使用resultMap映射的思路

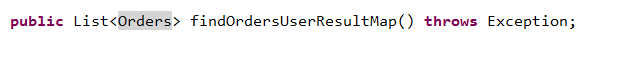
使用resultMap将查询的结果中的订单的新映射到Orders对象中 在orders类中添加用户信息



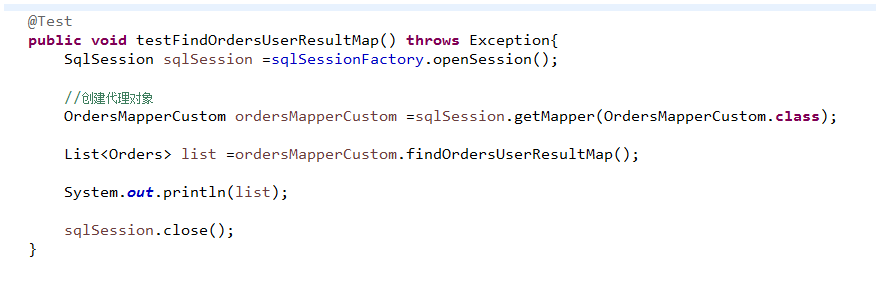
#### Mapper.xml



#### Mapper.java



#### 测试代码



### ResultType和resultMap实现一对一查询总结

* ResultType:使用resultType实现比较简单，如果查询的结果在POJO中不存在对应的属性 无法完成映射，需要增加列名对应的属性名 可以完成映射

如果对查询结果没有特殊的要求建议使用resutlType

* resultMap :需要单独定义resultMap 实现过程有些复杂，如果对查询结果有特殊的要求 可以使用resultMap完成关联映射查询。

ResultMap 可以实现延迟加载，resultType无法实现延迟加载。

## 一对多查询

### 需求

查询订单和用户以及订单的明细

### Sql语句

确定查询主表 ：订单表

确定关联表 ： 订单表明细

select orders.\*,user.username,user.sex,user.address,

orderdetail.id orderdetail\_id,

orderdetail.items\_id,

orderdetail.items\_num,

orderdetail.orders\_id

FROM

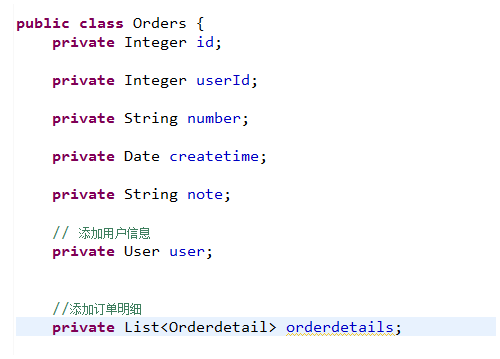
orders,

USER,

orderdetail

where orders.user\_id=user.id

### 在orders类中添加list订单明细的属性



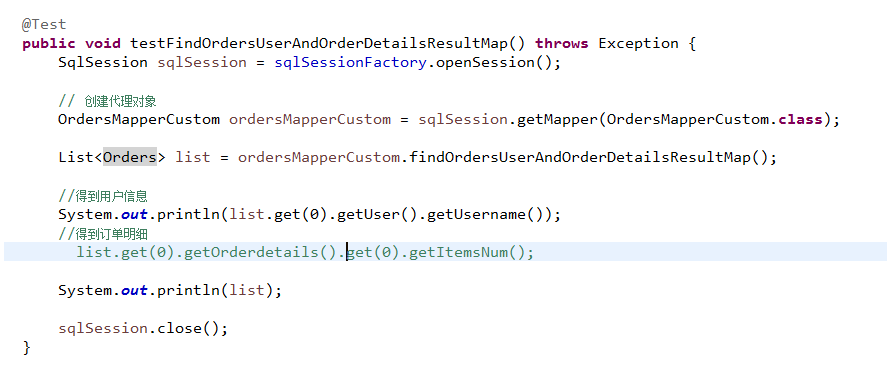
### Mapper.xml



### Mapper.java



### 测试代码



## 多对多查询

### 需求

查询用户以及用户购买的所有商品信息

### Sql语句

查询主表是： 用户表

关联表：由于用户和商品没有直接关联 ，通过订单和订单明细进行关联，所以关联表是orders、orderdetail 、items

select orders.\*,user.username,user.sex,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

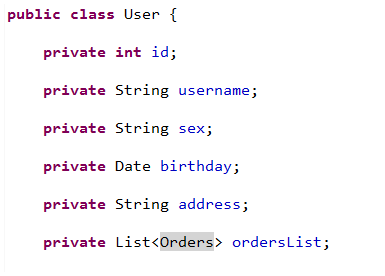
### 映射思路

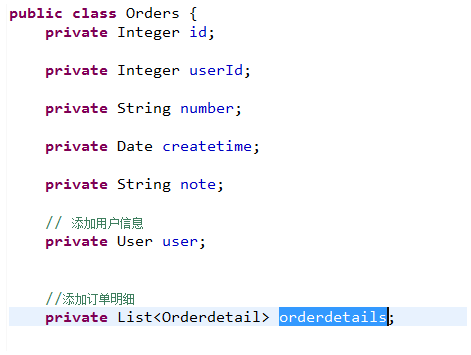
将用户信息映射到user中

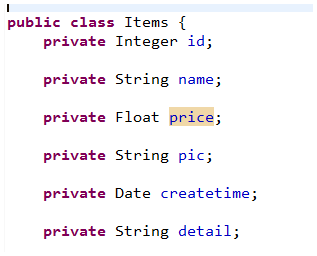
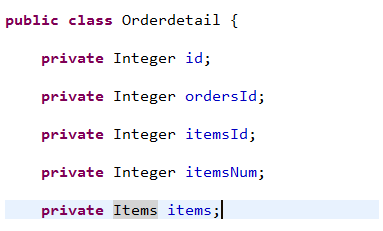
在user类中添加订单列表属性 List<orders> orderslist 将用户的订单映射到orderslist

在orders类中添加订单明细属性List<OrdersDetail> ordersdetails将订单的明细映射到ordersdetails

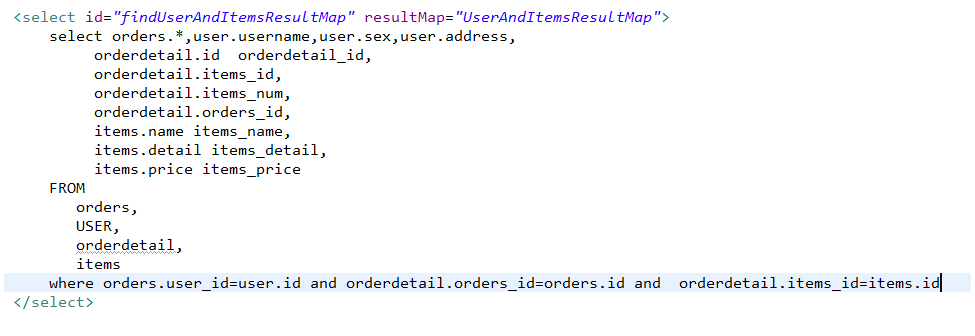
在 orderDetail类中 添加items属性，将订单明细对应的商品映射到Items中







### Mapper.xml

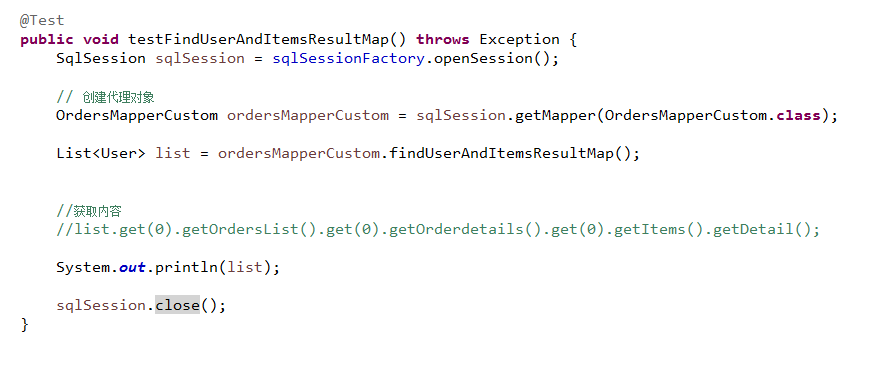




### Mapper.java



### 测试代码



# 延迟加载

## 什么是延迟加载

ResultMap可以实现高级映射(使用 association 和Collection 实现一对一和一对多映射) 、association 和Collection自带延迟加载功能。

如果查询订单 并且关联查询用户信息 。 如果先查询订单 即可满足需求，当我们在查询用户信息时，就说对用户信息按需加载，就是延迟加载。

延迟加载：先从单表查询 需要时在从关联表查询，大大提高数据库性能。 同时加快查询速度。

## 使用association 实现延迟加载

### 需求

查询订单并且关联查询用户信息

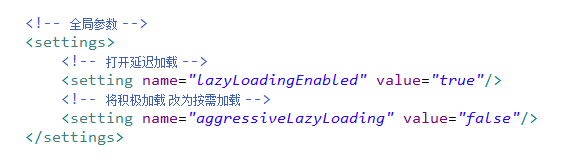
### Mapper.xml



### Mapper.java



### 在sqlMapConfig.xml中开启延迟加载



### 测试代码

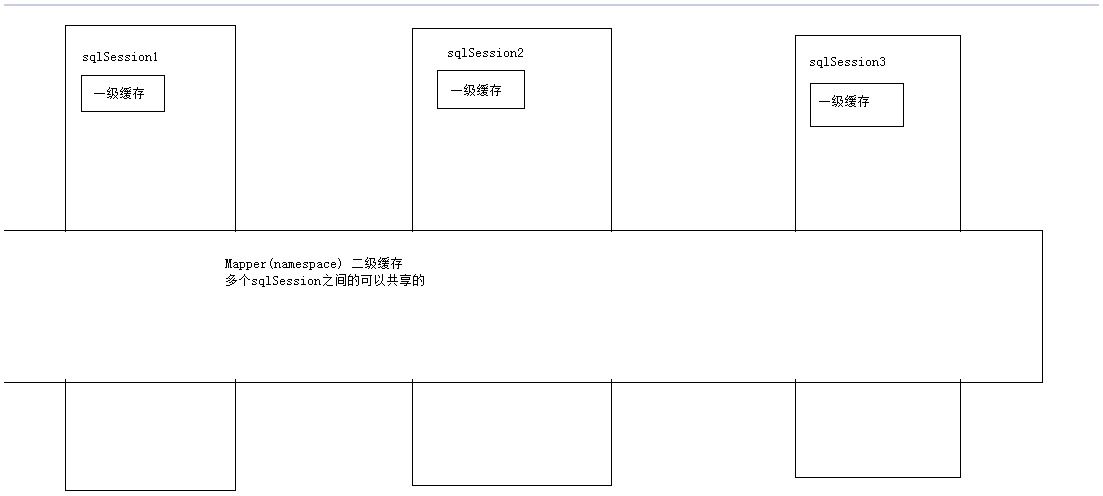


# 查询缓冲

## 什么是查询缓冲

MyBatis提供查询缓存，用于减轻数据压力，提高数据库性能。

MyBatis提供一级缓存和二级缓存。

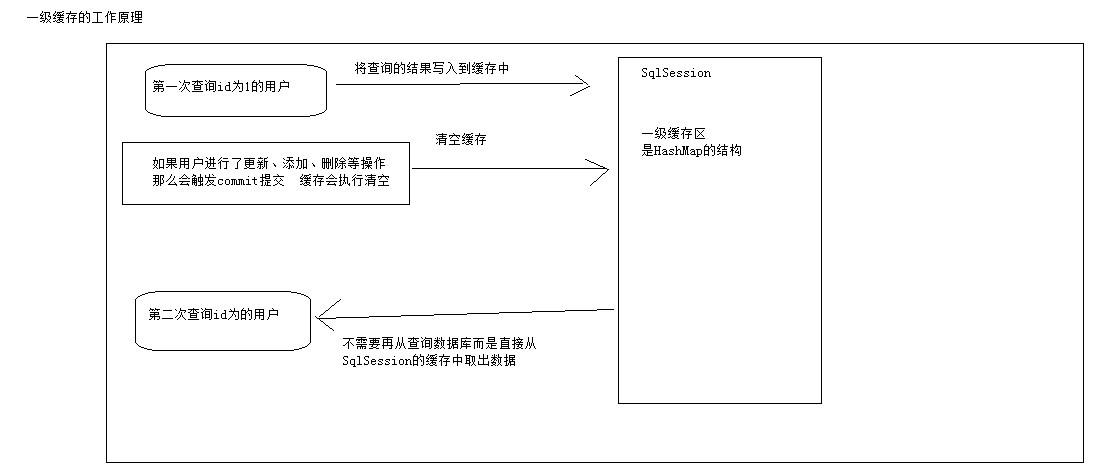


**一级缓存是SqlSession级别的缓存**，在操作数据库时需要构建SqlSession对象，在对象中有一个**数据结构(HashMap)用于存储缓存数据**，不同的SqlSession之间的缓存区域是互相不影响的。一般是系统自动维护

二级缓存是Mapper级别的缓存 ，多个SqlSession之间共同操作一个Mapper的sql语句，多个SqlSession是可以共用一个二级缓存的 也就是二级缓存是跨SqlSession的

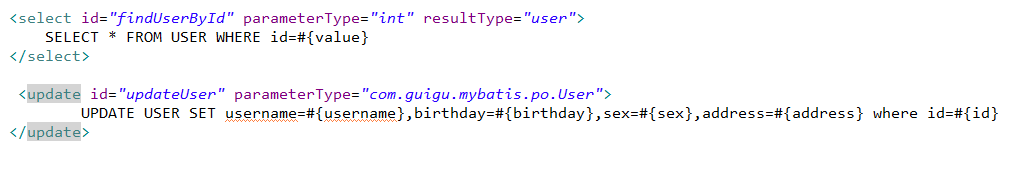
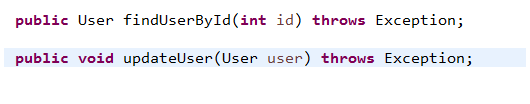
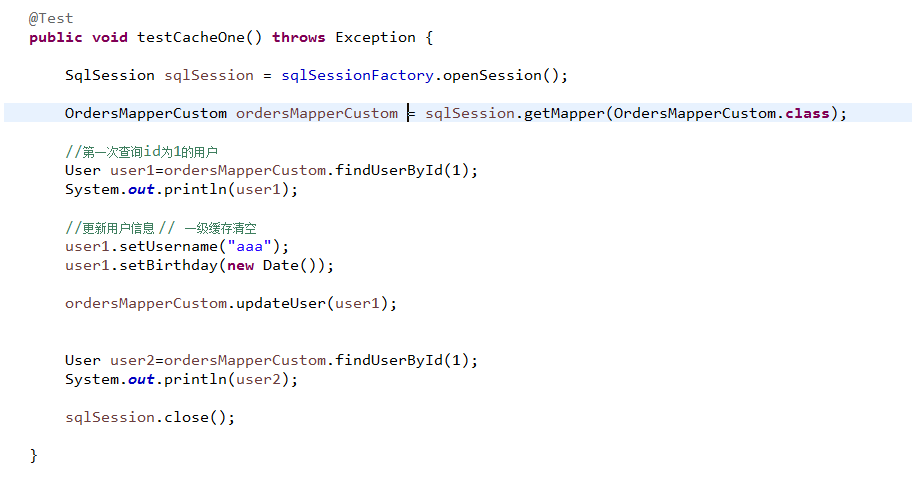
## 一级缓存

### 一级缓存的工作原理

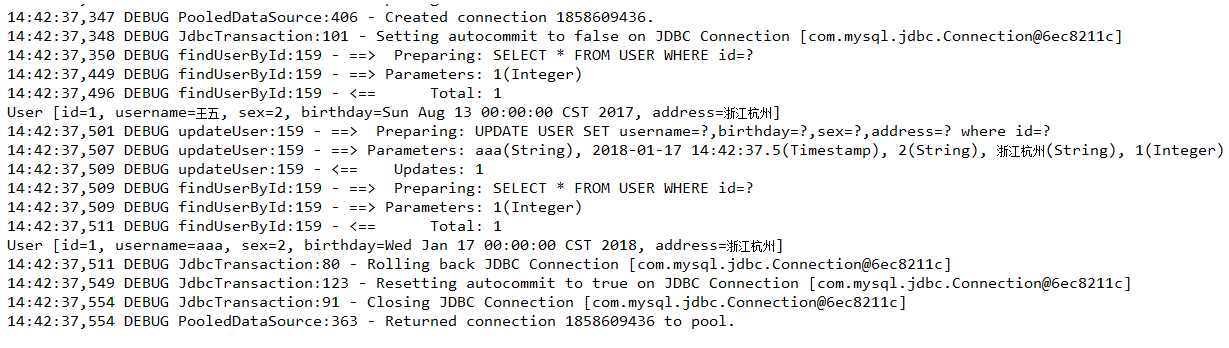


### 测试一级缓存的存在

MyBatis默认支持一级缓存，不需要额外配置

通过日志信息查询结果

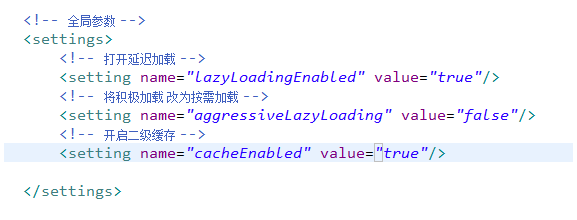


## 二级缓存

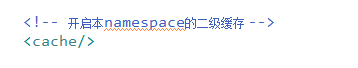
二级缓存是需要配置和开启的

### 开启二级缓存

在SqlMapConfig.xml中配置开启二级缓存



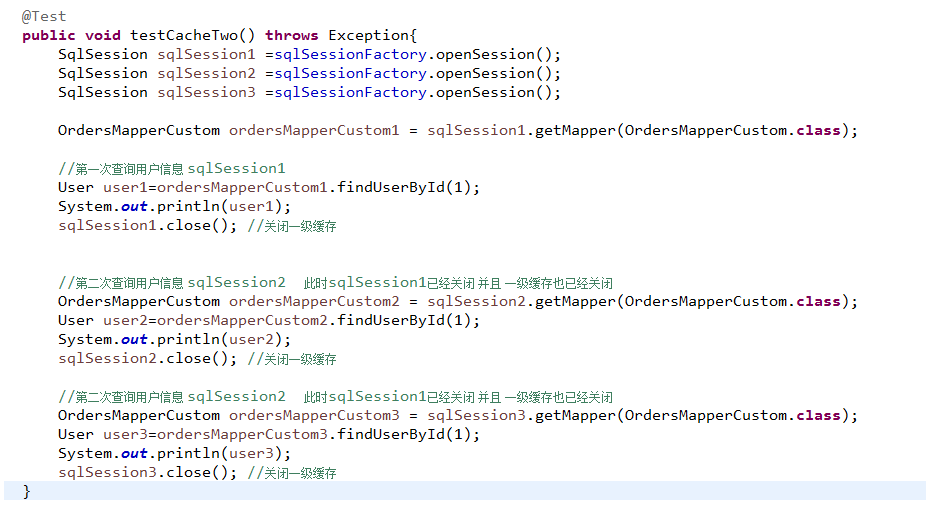
在映射文件中同时设置开启二级缓存

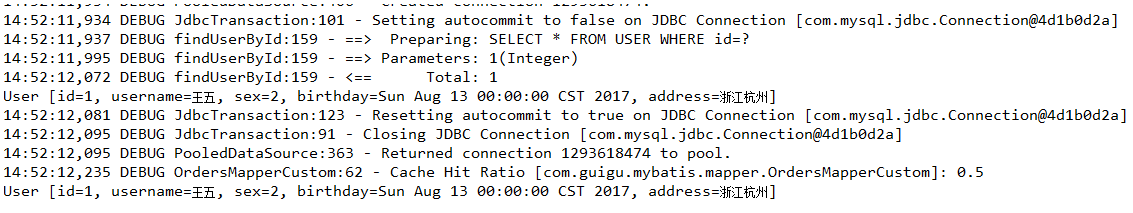


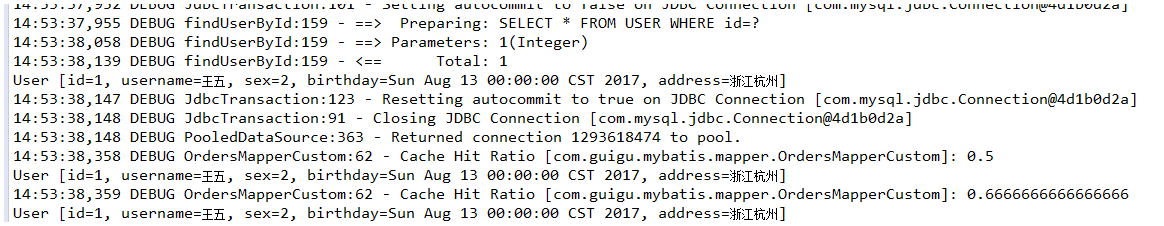
### 调用的POJO类需要实现序列化接口



### 测试代码







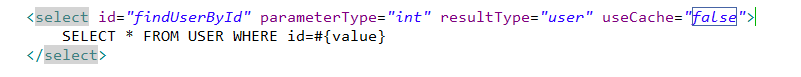
把二级缓存的开关关闭 再次查询数据



### useCahe的配置

在查询多条数据的时候 如果是从缓存中取数据 如果在这个过程中 数据被其他程序所修改，当前缓存中的数据就是过期的数据，会导致出错。

对于某些非重要数据 可以设置到二级缓存中 ，某些数据必须保证实时数据，可以通过设置userCache每次操作都从数据库查询数据。

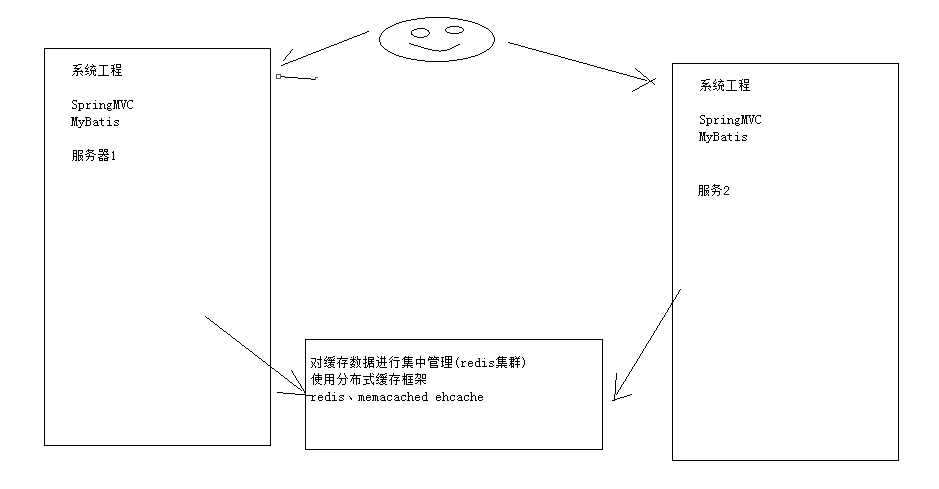


每次操作都是查询数据库

### 配置外置二级缓存 EHCache

EHCache的一个分布式缓存框架

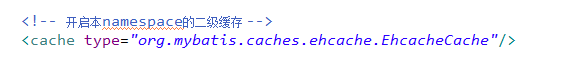
hcache是一种广泛使用的开源Java分布式缓存。主要面向通用缓存,Java EE和轻量级容器。它具有内存和磁盘存储，缓存加载器,缓存扩展,缓存异常处理程序,一个gzip缓存servlet过滤器,支持REST和SOAP api等特点。



#### 导入EHCache的jar包



#### mybatis整合ehcahe



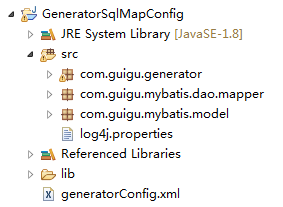
#### 导入EHcache的配置文件



# 逆向工程

## 什么是逆向工程

MyBatis需要程序员自己编写sql语句。MyBatis官方提供了逆向工程，可以自动生成Mapper.xml 和Mapper.java 以及Po



## 配置文件



## 生成配置文件



## 测试生成的代码

**package** com.guigu.mybatis.dao.test;

**import** java.io.InputStream;

**import** java.util.Date;

**import** java.util.List;

**import** org.apache.ibatis.io.Resources;

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

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

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

**import** org.junit.Before;

**import** org.junit.Test;

**import** com.guigu.mybatis.dao.mapper.UserMapper;

**import** com.guigu.mybatis.model.User;

**import** com.guigu.mybatis.model.UserExample;

**import** com.guigu.mybatis.model.UserExample.Criteria;

/\*\*

\* <p>project\_name:MyBatisGeneratorTest</p>

\* <p>package\_name:com.guigu.mybatis.dao.test.MyBatisTest</p>

\* <p>description：</p>

\* <p>@author：刘老师<p>

\* <p> date:2018年1月17日下午3:22:11 </p>

\* <p>comments： </p>

\* <p>@version jdk1.8</p>

\*

\* <p>Copyright (c) 2018, 980991634@qq.com All Rights Reserved. </p>

\*/

**public** **class** MyBatisTest {

**private** SqlSessionFactory sqlSessionFactory;

@Before

**public** **void** setUp() **throws** Exception {

String resource = "SqlMapConfig.xml";

InputStream inputStream = Resources.*getResourceAsStream*(resource);

sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(inputStream);

}

//查询数据 根据主键查询

@Test

**public** **void** testUserSelectByPrimaryKey() **throws** Exception {

SqlSession sqlSession = sqlSessionFactory.openSession();

UserMapper userMapper =sqlSession.getMapper(UserMapper.**class**);

User user=userMapper.selectByPrimaryKey(1);

System.***out***.println(user);

sqlSession.close();

}

//条件查询

@Test

**public** **void** testUserSelectByExample() **throws** Exception {

SqlSession sqlSession = sqlSessionFactory.openSession();

UserMapper userMapper =sqlSession.getMapper(UserMapper.**class**);

UserExample userExample =**new** UserExample();

//通过Criteria构建查询条件

Criteria criteria=userExample.createCriteria();

criteria.andUsernameEqualTo("钱小明");

criteria.andSexEqualTo("2");

// userExample.setDistinct(true);//是否去除重复的元素；

userExample.or(criteria);

//查询结果

List<User> list =userMapper.selectByExample(userExample);

System.***out***.println(list);

sqlSession.close();

}

//插入数据

@Test

**public** **void** testInsertUser() {

SqlSession sqlSession = sqlSessionFactory.openSession();

UserMapper userMapper =sqlSession.getMapper(UserMapper.**class**);

User user =**new** User();

user.setSex("2");

user.setBirthday(**new** Date());

user.setAddress("杭州市公共实训基地");

user.setUsername("张飞");

userMapper.insert(user);

sqlSession.commit();

sqlSession.close();

}

//更新数据

@Test

**public** **void** testUpdateByPrimaryKey() {

SqlSession sqlSession = sqlSessionFactory.openSession();

UserMapper userMapper =sqlSession.getMapper(UserMapper.**class**);

User user =userMapper.selectByPrimaryKey(32);

user.setSex("1");

user.setAddress("杭州西湖");

userMapper.updateByPrimaryKey(user);

sqlSession.commit();

sqlSession.close();

}

}