Mybatis

框架课程

# 课程计划

1. mybatis的介绍
2. mybatis的框架原理（重点）
3. 入门程序

订单商品案例（用户表）

1. Mybatis开发dao的方式（重点）
   1. 原始dao开发方式（开发dao接口和dao实现类，由ibatis遗留下来的风格）
   2. Mapper代理的开发方式（推荐，开发mapper接口（相当于dao接口））
2. 全局配置文件
3. 映射文件（重点）
   1. 输入映射
   2. 输出映射
   3. 动态sql
4. mybatis和hibernate的区别及应用场景

# mybatis的介绍

mybatis就是一个封装来jdbc的持久层框架，它和hibernate都属于ORM框架，但是具体的说，hibernate是一个完全的orm框架，而mybatis是一个不完全的orm框架。

Mybatis让程序员**只关注sql本身**，而不需要去关注如连接的创建、statement的创建等操作。

Mybatis会将输入参数、输出结果进行映射。

# 分析jdbc的问题

## 原生态的jdbc代码

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

Connection connection = **null**;

PreparedStatement preparedStatement = **null**;

ResultSet resultSet = **null**;

**try** {

//1、加载数据库驱动

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

//2、通过驱动管理类获取数据库链接

connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8", "root", "root");

//3、定义sql语句 ?表示占位符

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

//4、获取预处理statement

preparedStatement = connection.prepareStatement(sql);

//5、设置参数，第一个参数为sql语句中参数的序号（从1开始），第二个参数为设置的参数值

preparedStatement.setString(1, "王五");

//6、向数据库发出sql执行查询，查询出结果集

resultSet = preparedStatement.executeQuery();

//7、遍历查询结果集

**while**(resultSet.next()){

User user

System.*out*.println(resultSet.getString("id")+" "+resultSet.getString("username"));

}

} **catch** (Exception e) {

e.printStackTrace();

}**finally**{

//8、释放资源

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

**try** {

resultSet.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

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

**try** {

preparedStatement.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

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

**try** {

connection.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

## 问题总结

1. 在创建连接时，存在硬编码

配置文件（全局配置文件）

1. 在执行statement时存在硬编码

配置文件（映射文件）

1. 频繁的开启和关闭数据库连接，会造成数据库性能下降。

数据库连接池（全局配置文件）

# Mybatis的框架原理

参考画图.xlsx

# 入门程序

## 需求

对订单商品案例中的用户表进行增删改查操作

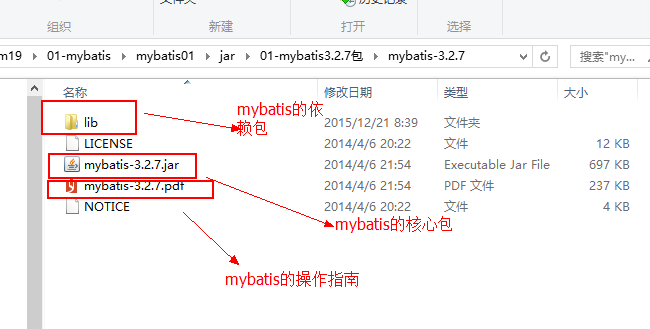
1. 根据用户ID查询用户信息
2. 根据用户名称模糊查询用户列表
3. 添加用户
4. 删除用户（练习）
5. 修改用户（练习）

## 环境准备

* Jdk：1.7
* Ide：eclipse indigo
* Mybatis：3.2.7
* 数据库：MySQL 5X

### 下载mybatis

mybaits的代码由github.com管理，下载地址：<https://github.com/mybatis/mybatis-3/releases>

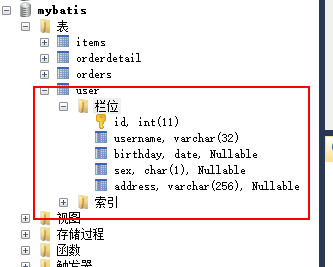


### 数据库脚本初始化

#### 数据库脚本

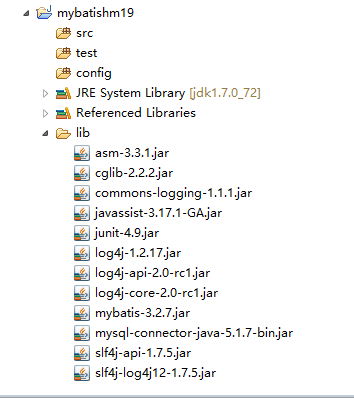


1. 执行sql\_table.sql脚本，创建数据库表；
2. 执行sql\_data.sql初始化测试数据。



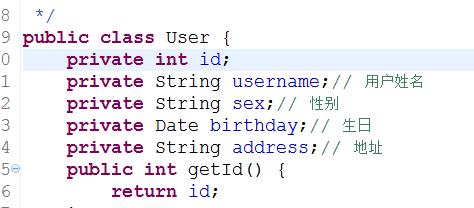
## 工程搭建

* Mybatis的核心包和依赖包
* MySQl的驱动包
* Junit（非必须）



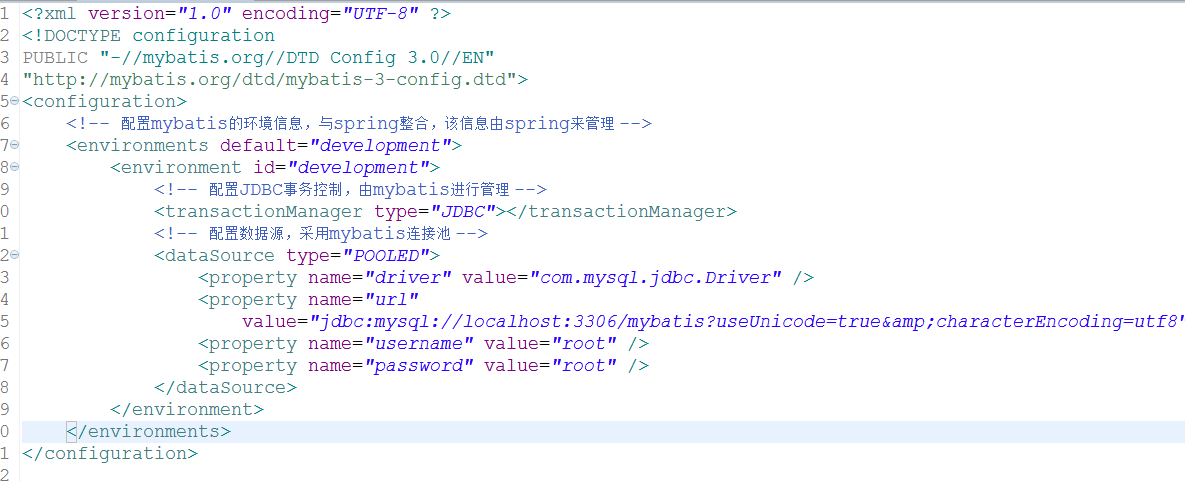
## 代码实现

### 创建po类



### 创建全局配置文件

在config目录下，创建SqlMapConfig.xml文件，该名称不是固定不变的。

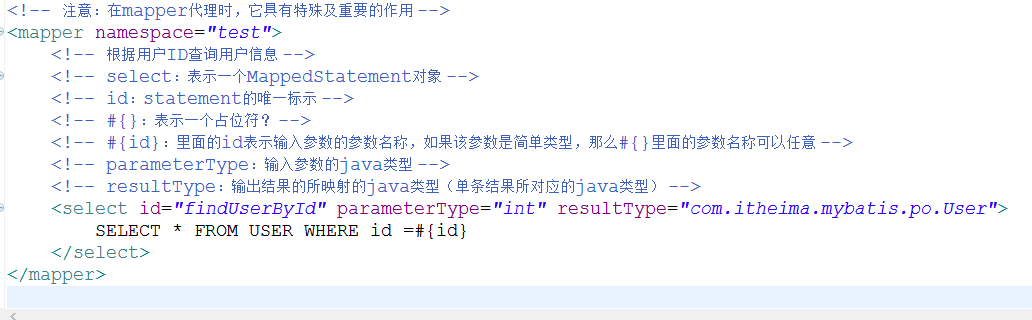


### 需求开发

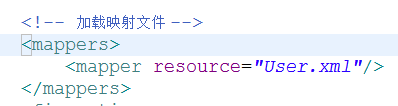
#### 根据用户ID查询用户信息

##### 映射文件

在config目录下，创建User.xml（这种命名规范是由ibatis遗留下来）



##### 在全局配置文件中加载映射文件

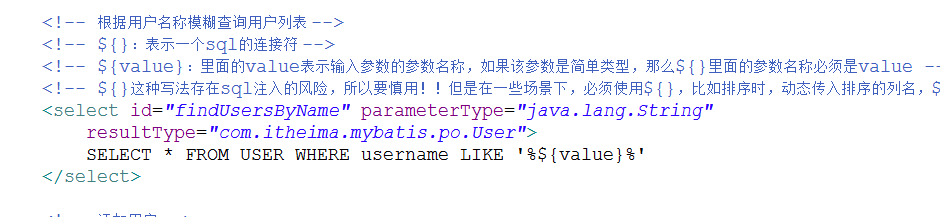


##### 测试代码



#### 根据用户名称模糊查询用户列表

##### 映射文件

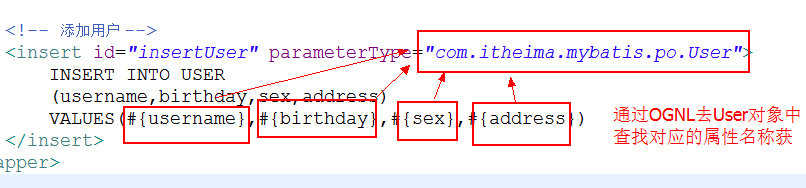


##### 测试代码



#### 添加用户

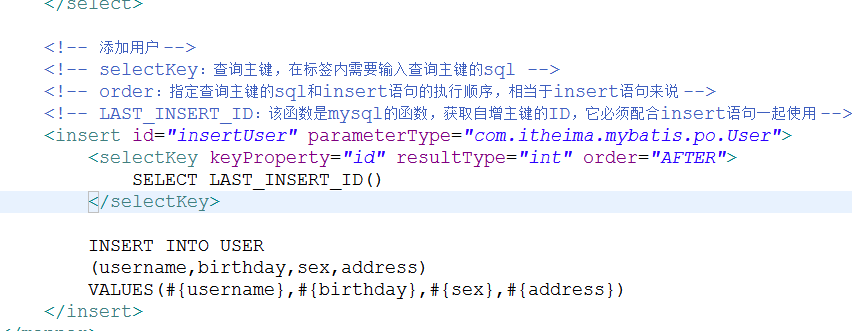
##### 映射文件



##### 测试代码

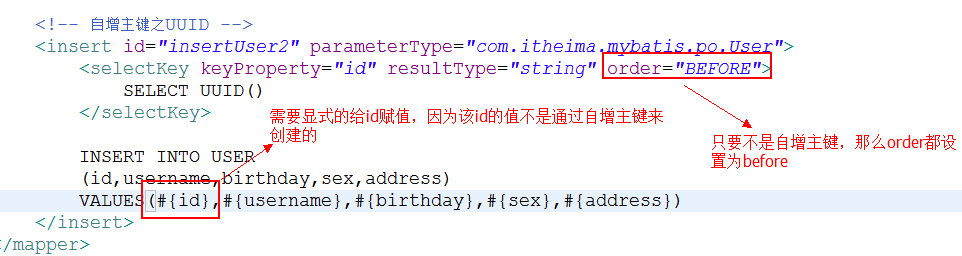


##### 主键返回之自增主键



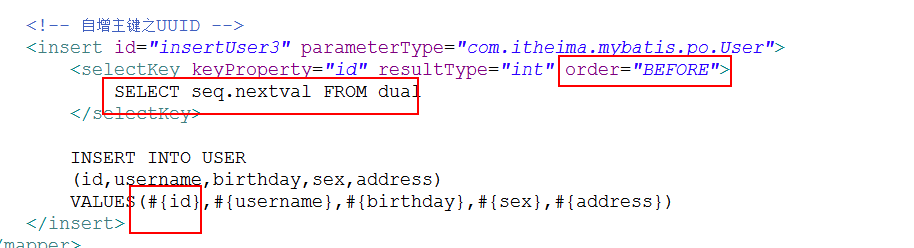
##### 主键返回值UUID

UUID函数是mysql的函数



##### 主键返回值序列

序列也就是sequence，它是Oracle的主键生成策略



### 小结

* #{}和${}

#{}表示占位符?，#{}接收简单类型的参数时，里面的名称可以任意

${}表示拼接符，${}接收简单类型的参数时，里面的名称必须是value

${}里面的值会原样输出，不加解析（如果该参数值是字符串，有不会添加引号）

${}存在sql注入的风险，但是有些场景下必须使用，比如排序后面会动态传入排序的列名

* parameterType和resultType

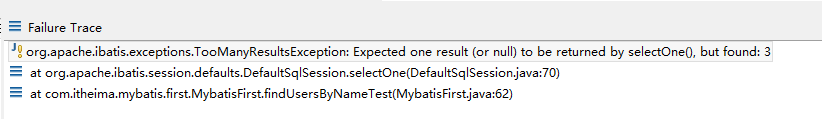
parameterType指定输入参数的java类型，parameterType只有一个，也就是说入参只有一个。

resultType指定输出结果的java类型（是单条记录的java类型）

* selectOne和selectList

selectOne查询单个对象

selectList查询集合对象



# mybatis开发dao的方式

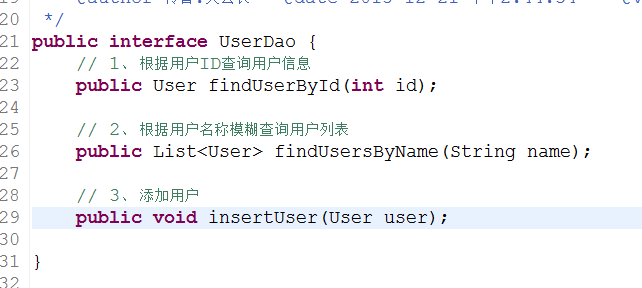
## 需求

1. 根据用户ID查询用户信息
2. 根据用户名称模糊查询用户列表
3. 添加用户

## 原始dao的开发方式

即开发dao接口和dao实现类

### Dao接口



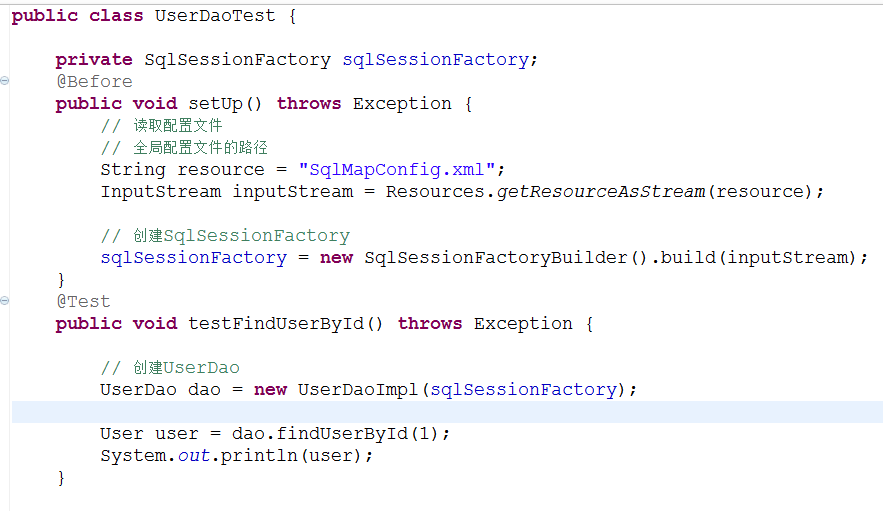
### Dao实现类

SqlSessionFactory，它的生命周期，应该是应用范围，全局范围只有一个工厂，使用单例模式来实现这个功能。与spring集成之后，由spring来对其进行单例管理。

SqlSession，它内部含有一块数据区域，存在线程不安全的问题，所以应该将sqlsession声明到方法内部。

|  |
| --- |
| **public** **class** UserDaoImpl **implements** UserDao {  // 依赖注入  **private** SqlSessionFactory sqlSessionFactory;  **public** UserDaoImpl(SqlSessionFactory sqlSessionFactory) {  **this**.sqlSessionFactory = sqlSessionFactory;  }  @Override  **public** User findUserById(**int** id) **throws** Exception {  // 创建SqlSession  SqlSession sqlSession = sqlSessionFactory.openSession();  // 调用SqlSession的增删改查方法  // 第一个参数：表示statement的唯一标示  User user = sqlSession.selectOne("test.findUserById", id);  System.*out*.println(user);  // 关闭资源  sqlSession.close();  **return** sqlSession.selectOne("test.findUserById", 1);  }  @Override  **public** List<User> findUsersByName(String name) {  // 创建SqlSession  SqlSession sqlSession = sqlSessionFactory.openSession();  // 调用SqlSession的增删改查方法  // 第一个参数：表示statement的唯一标示  List<User> list = sqlSession.selectOne("test.findUsersByName", name);  System.*out*.println(list);  // 关闭资源  sqlSession.close();  **return** list;  }  @Override  **public** **void** insertUser(User user) {  // 创建SqlSession  SqlSession sqlSession = sqlSessionFactory.openSession();  // 调用SqlSession的增删改查方法  // 第一个参数：表示statement的唯一标示  sqlSession.insert("test.insertUser", user);  System.*out*.println(user.getId());  // 提交事务  sqlSession.commit();  // 关闭资源  sqlSession.close();  }  } |

### 测试代码



### 问题思考

1. 有大量的重复的模板代码
2. 存在硬编码

## Mapper代理的开发方式

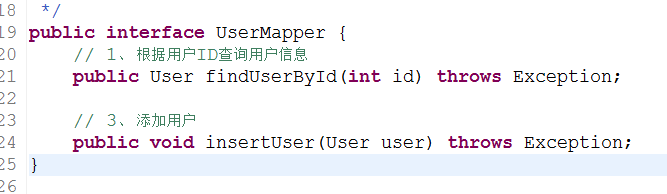
即开发mapper接口（相当于dao接口）

Mapper代理使用的是jdk的代理策略。

### Mapper代理的开发规范

1. mapper接口的全限定名要和mapper映射文件的namespace值一致。
2. mapper接口的方法名称要和mapper映射文件的statement的id一致。
3. mapper接口的方法参数类型要和mapper映射文件的statement的parameterType的值一致，而且它的参数是一个。
4. mapper接口的方法返回值类型要和mapper映射文件的statement的resultType的值一致。

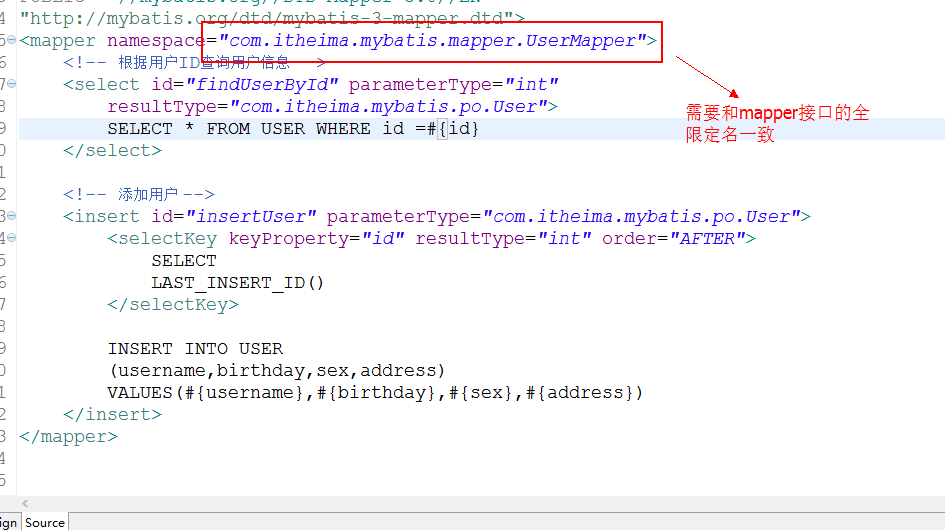
### mapper接口



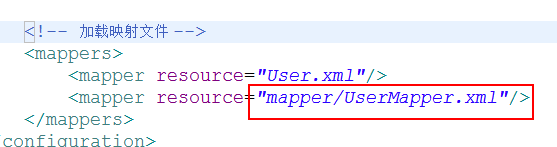
### mapper映射文件

在config下创建mapper目录然后创建UserMapper.xml（这是mybatis的命名规范，当然，也不是必须是这个名称）

sqlSession内部的数据区域本身就是一级缓存，是通过map来存储的。



### 加载映射文件



### 测试代码



# 全局配置文件

## 概览

**SqlMapConfig.xml的配置内容和顺序如下（顺序不能乱）：**

Properties（属性）

Settings（全局参数设置）

typeAliases（类型别名）

typeHandlers（类型处理器）

objectFactory（对象工厂）

plugins（插件）

environments（环境信息集合）

environment（单个环境信息）

transactionManager（事物）

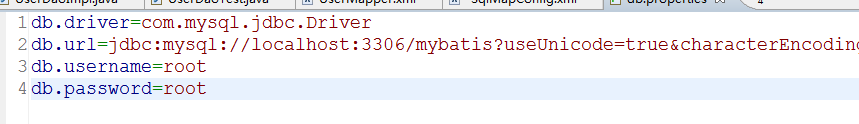
dataSource（数据源）

mappers（映射器）

## 常用配置

### Properties

Db.properties



SqlMapConfig.xml



加载的顺序

1. 先加载properties中property标签声明的属性
2. 再加载properties标签引入的java配置文件中的属性
3. parameterType的值会和properties的属性值发生冲突。

### settings

mybatis全局配置参数，全局参数将会影响mybatis的运行行为。

详细参见“mybatis学习资料/mybatis-settings.xlsx”文件







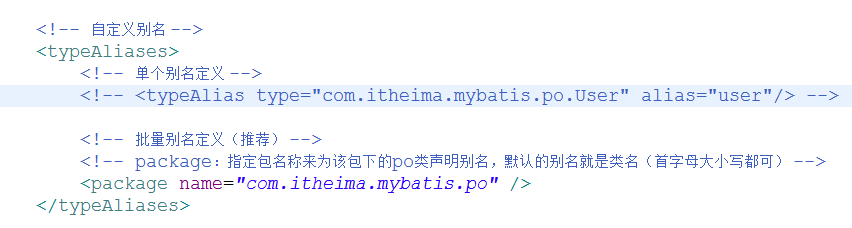
### typeAliases

对po类进行别名的定义

#### mybatis支持的别名

|  |  |
| --- | --- |
| 别名 | 映射的类型 |
| \_byte | byte |
| \_long | long |
| \_short | short |
| \_int | int |
| \_integer | int |
| \_double | double |
| \_float | float |
| \_boolean | boolean |
| string | String |
| byte | Byte |
| long | Long |
| short | Short |
| int | Integer |
| integer | Integer |
| double | Double |
| float | Float |
| boolean | Boolean |
| date | Date |
| decimal | BigDecimal |
| bigdecimal | BigDecimal |

#### 自定义别名



### Mappers

#### <mapper resource=’’/>

使用相对于类路径的资源

如：<mapper resource="sqlmap/User.xml" />

#### <mapper url=’’/>

使用完全限定路径

如：<mapper url="file:///D:\workspace\_spingmvc\mybatis\_01\config\sqlmap\User.xml" />

#### <mapper class=’’/>

使用mapper接口的全限定名

如：<mapper class="cn.itcast.mybatis.mapper.UserMapper"/>

**注意：此种方法要求mapper接口和mapper映射文件要名称相同，且放到同一个目录下**；

#### <package name=’’/>（推荐）

注册指定包下的所有映射文件

如：<package name="cn.itcast.mybatis.mapper"/>

**注意：此种方法要求mapper接口和mapper映射文件要名称相同，且放到同一个目录下**；

# 映射文件

## 输入映射

### 简单类型

参考入门程序之根据用户ID查询用户信息的映射文件

### Pojo类型

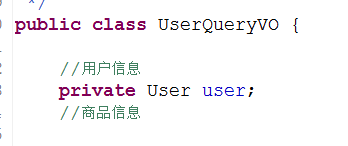
参考入门程序之添加用户的映射文件

### 包装pojo类型

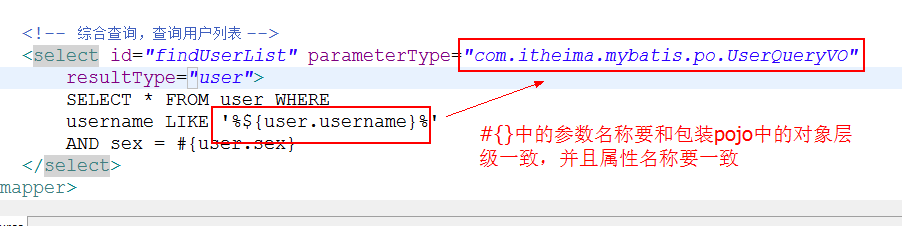
#### 需求

综合查询时，可能会根据用户信息、商品信息、订单信息等作为条件进行查询，用户信息中的查询条件由：用户的名称和性别进行查询

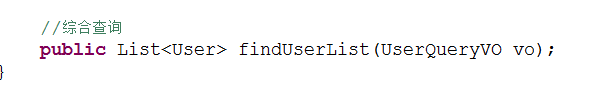
#### 创建包装pojo



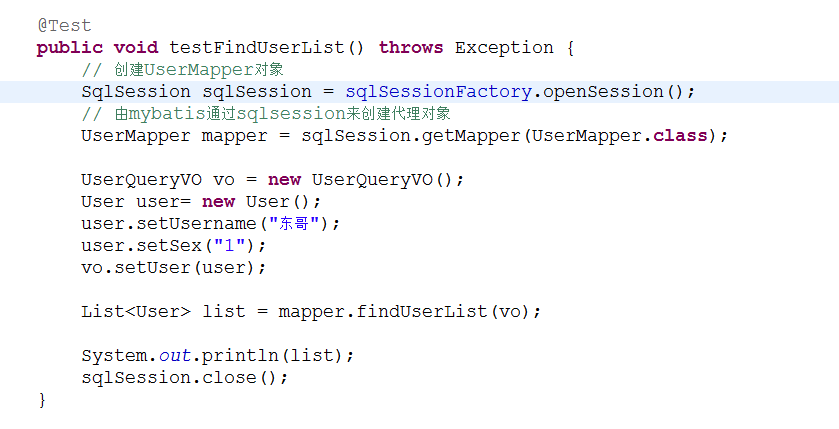
#### 映射文件



#### Mapper接口



#### 测试代码



### Map

同传递POJO对象一样，**map的key相当于pojo的属性**。

#### 映射文件

<!-- 传递hashmap综合查询用户信息 -->

<select id=*"findUserByHashmap"* parameterType=*"hashmap"* resultType=*"user"*>

select \* from user where id=#{id} and username like '%${username}%'

</select>

上边红色标注的是hashmap的key。

#### 测试代码

**Public void** testFindUserByHashmap()**throws** Exception{

//获取session

SqlSession session = sqlSessionFactory.openSession();

//获限mapper接口实例

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

//构造查询条件Hashmap对象

HashMap<String, Object> map = **new** HashMap<String, Object>();

map.put("id", 1);

map.put("username", "管理员");

//传递Hashmap对象查询用户列表

List<User>list = userMapper.findUserByHashmap(map);

//关闭session

session.close();

}

异常测试：

传递的map中的key和sql中解析的key不一致。

测试结果没有报错，只是通过key获取值为空。

## 输出映射

### resultType

#### 使用要求

使用resultType进行结果映射时，需要查询出的列名和映射的对象的属性名一致，才能映射成功。

如果查询的列名和对象的属性名全部不一致，那么映射的对象为空。

如果查询的列名和对象的属性名有一个一致，那么映射的对象不为空，但是只有映射正确那一个属性才有值。

如果查询的sql的列名有别名，那么这个别名就是和属性映射的列名。

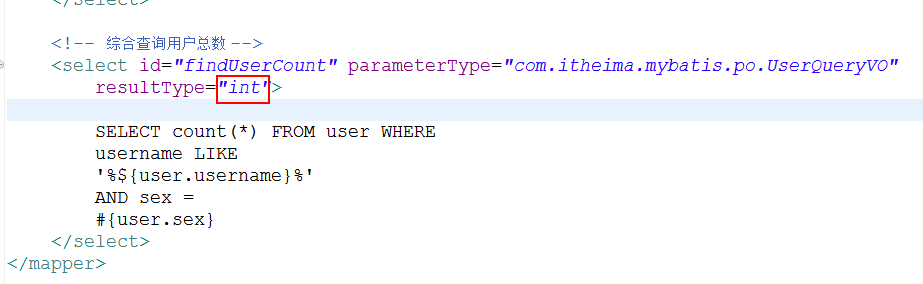
#### 简单类型

注意，对简单类型的结果映射也是有要求的，查询的列必须是一列，才能映射为简单类型。

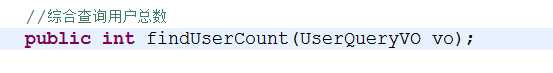
##### 需求

综合查询时，需要根据综合查询的添加查询用户的总数

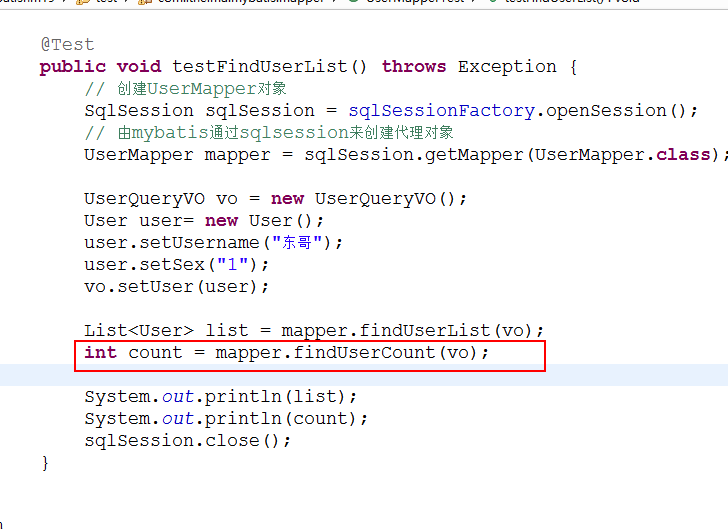
##### 映射文件



##### Mapper接口



##### 测试代码



#### Pojo对象和pojo列表

参考入门程序之根据用户ID查询用户信息和根据用户名称模糊查询用户列表

### resultMap

#### 使用要求

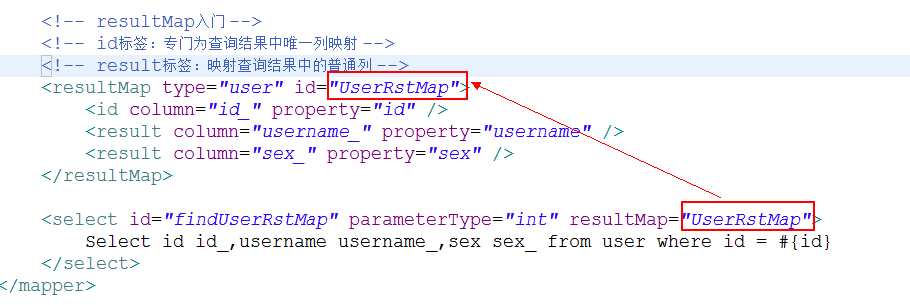
使用resultMap进行结果映射时，不需要查询的列名和映射的属性名必须一致。但是需要声明一个resultMap，来对列名和属性名进行映射。

#### 需求

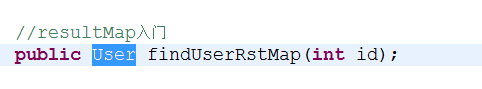
对以下sql查询的结果集进行对象映射

Select id id\_,username username\_,sex sex\_ from user where id = 1;

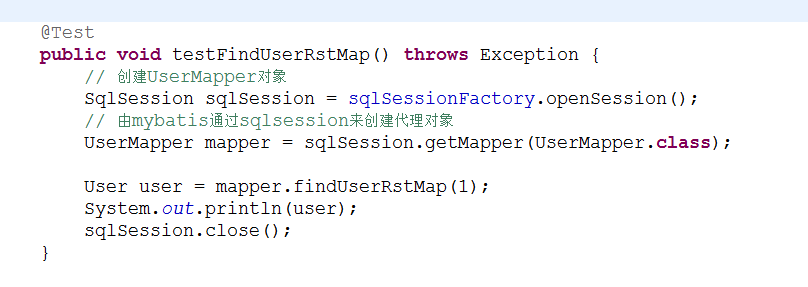
#### 映射文件



#### Mapper接口



#### 测试代码



### 动态sql

在mybatis中，它提供了一些动态sql标签，可以让程序员更快的进行mybatis的开发，这些动态sql可以通过sql的可重用性。。

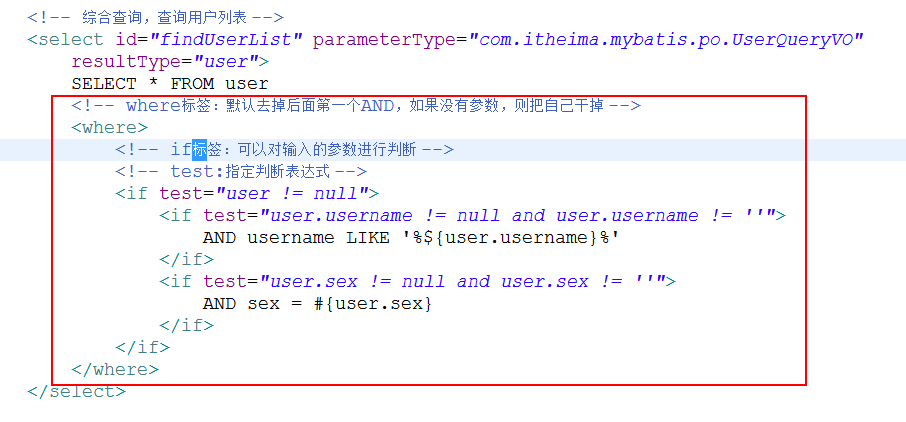
常用的动态sql标签：if标签、where标签、sql片段、foreach标签

#### If标签/where标签

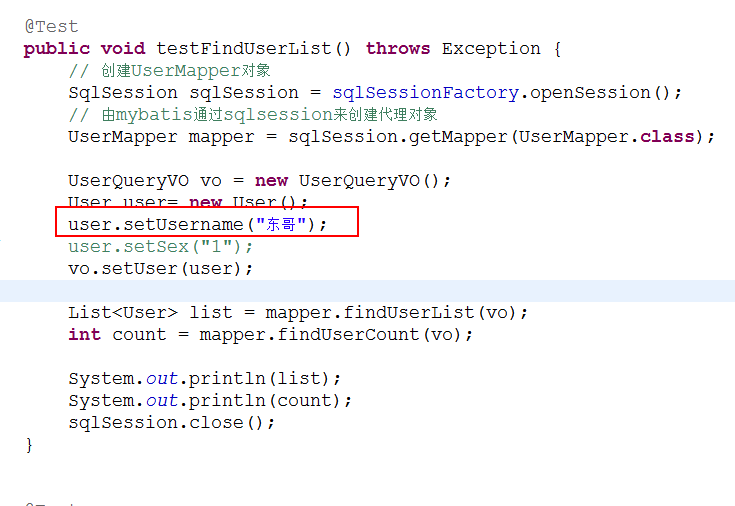
##### 需求

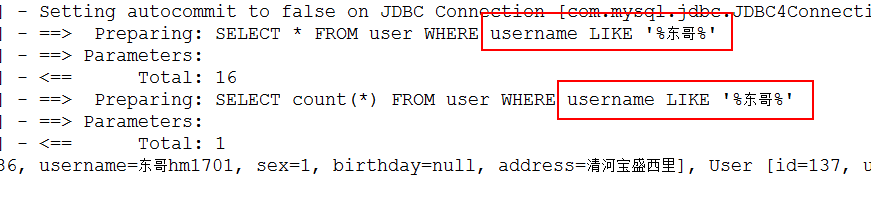
综合查询时，查询条件由用户来输入，用户名称可以为空，需要满足这种情况下的sql编写。

##### 映射文件



##### 测试代码

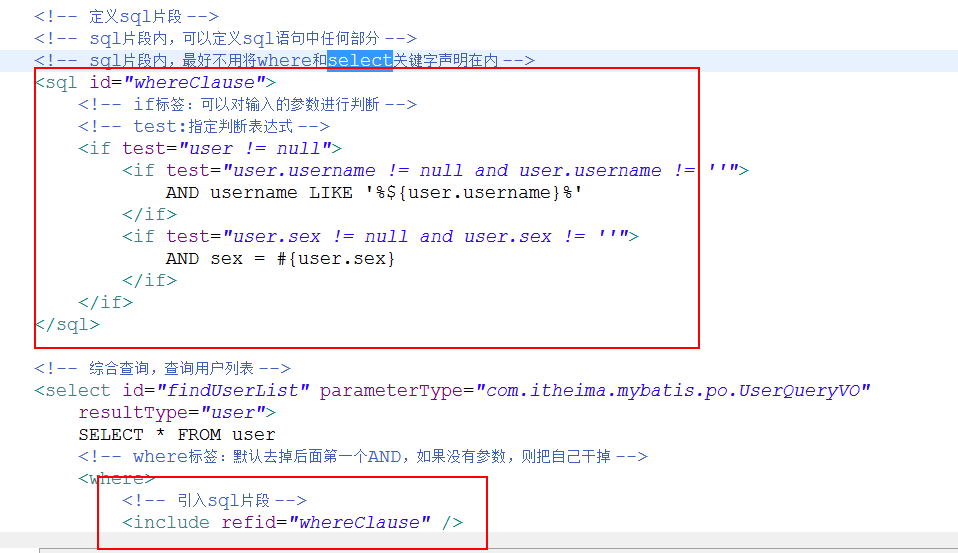




#### Sql片段

Sql片段可以让代码有更高的可重用性

Sql片段需要先定义后使用



#### Foreach标签

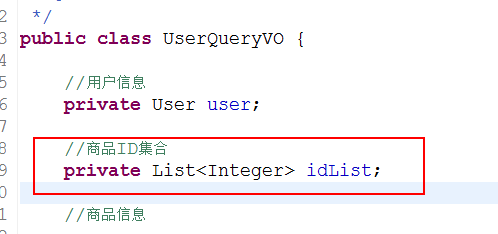
可以循环传入参数值

##### 需求

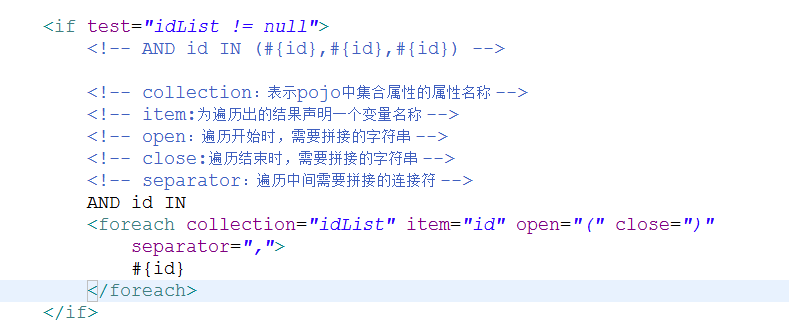
综合查询时，会根据用户ID集合进行查询

SELECT \* FROM USER WHERE id IN (1,2,10)

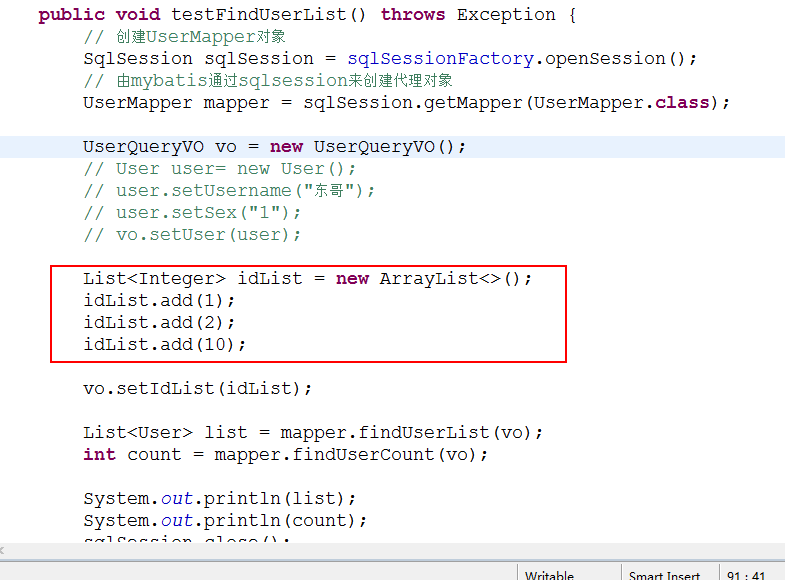
##### 修改包装pojo

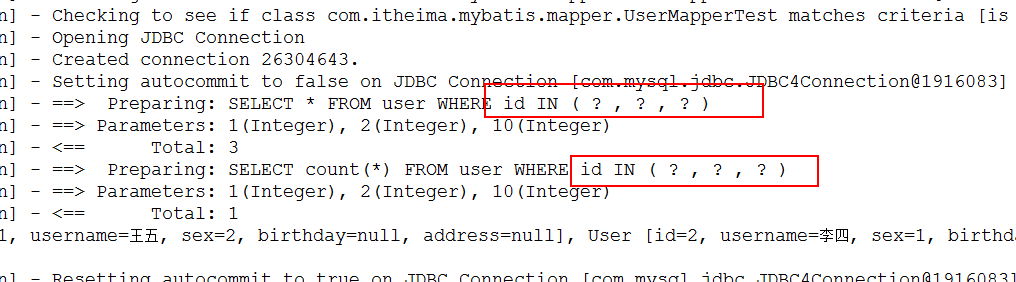


##### 映射文件



##### 测试代码





# mybatis与hibernate的区别及各自应用场景

Mybatis技术特点：

1. 通过直接编写SQL语句，可以直接对SQL进行性能的优化；
2. 学习门槛低，学习成本低。只要有SQL基础，就可以学习mybatis，而且很容易上手；
3. 由于直接编写SQL语句，所以灵活多变，代码维护性更好。
4. 不能支持数据库无关性，即数据库发生变更，要写多套代码进行支持，移植性不好。

Hibernate技术特点：

1. 标准的orm框架，程序员不需要编写SQL语句。
2. 具有良好的数据库无关性，即数据库发生变化的话，代码无需再次编写。
3. 学习门槛高，需要对数据关系模型有良好的基础，而且在设置OR映射的时候，需要考虑好性能和对象模型的权衡。
4. 程序员不能自主的去进行SQL性能优化。

Mybatis应用场景：

需求多变的互联网项目，例如电商项目。

Hibernate应用场景：

需求明确、业务固定的项目，例如OA项目、ERP项目等。