mybatis 第一天 mybatis的基础知识

课程安排：

mybatis和springmvc通过订单商品 案例驱动

第一天：基础知识（重点，内容量多）

对原生态jdbc程序（单独使用jdbc开发）问题总结

mybatis框架原理 （掌握）

mybatis入门程序

用户的增、删、改、查

mybatis开发dao两种方法：

* 原始dao开发方法（程序需要编写dao接口和dao实现类）（掌握）
* mybaits的mapper接口（相当于dao接口）代理开发方法（掌握）

mybatis配置文件SqlMapConfig.xml

mybatis核心：

mybatis输入映射（掌握）

mybatis输出映射（掌握）

mybatis的动态sql（掌握）

第二天：高级知识

订单商品数据模型分析

高级结果集映射（一对一、一对多、多对多）

mybatis延迟加载

mybatis查询缓存（一级缓存、二级缓存）

mybaits和spring进行整合（掌握）

mybatis逆向工程

# 对原生态jdbc程序中问题总结

## 环境

java环境：jdk1.7.0\_72

eclipse：indigo

mysql：5.1

## 创建mysql数据

导入下边的脚本：



sql\_table.sql：记录表结构

sql\_data.sql：记录测试数据，在实际企业开发中，**最后提供一个初始化数据脚本**



## jdbc程序

使用jdbc查询mysql数据库中用户表的记录。

创建java工程，加入jar包：

数据库驱动包（mysql5.1）



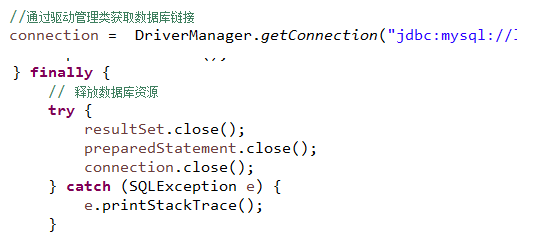
上边的是mysql驱动。

下边的是oracle的驱动。

程序代码：



## 问题总结



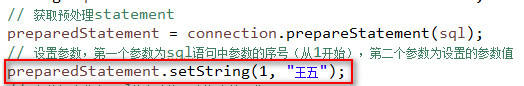
1、数据库连接，使用时就创建，不使用立即释放，导致对数据库进行频繁连接的开启和关闭，造成数据库资源浪费，影响数据库性能。

设想：使用数据库连接池管理数据库连接。



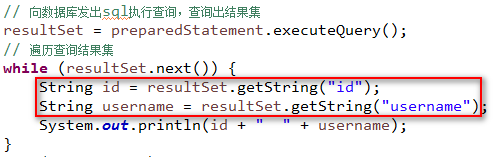
2、将sql语句硬编码到java代码中，如果sql 语句修改，需要重新编译java代码，不利于系统维护。

设想：将sql语句配置在xml配置文件中，即使sql变化，不需要对java代码进行重新编译。



3、向preparedStatement中设置参数，对占位符号位置和设置参数值，硬编码在java代码中，不利于系统维护。

设想：将sql语句及占位符号和参数全部配置在xml中。



4、从resutSet中遍历结果集数据时，存在硬编码，将获取表的字段进行硬编码，，不利于系统维护。

设想：将查询的结果集，自动映射成java对象。

# mybatis框架

## mybatis是什么？

mybatis是一个持久层的框架，是apache下的顶级项目。

mybatis托管到goolecode下，再后来托管到github下(https://github.com/mybatis/mybatis-3/releases)。

mybatis让程序将主要精力放在sql上，通过mybatis提供的映射方式，自由灵活生成（半自动化，大部分需要程序员编写sql）满足需要sql语句。

mybatis可以将向 preparedStatement中的输入参数自动进行输入映射，将查询结果集灵活映射成java对象。（输出映射）

## mybatis框架

SqlMapConfig.xml（是mybatis的全局配置文件，名称不固定的）

配置了数据源、事务等mybatis运行环境

配置映射文件（配置sql语句）

mapper.xml（映射文件）、mapper.xml、mapper.xml.....

SqlSessionFactory（会话工厂），根据配置文件创建工厂

作用：创建SqlSession

msyql

SqlSession（会话），是一个接口，面向用户（程序员）的接口

作用：操作数据库（发出sql增、删、改、查）

Executor（执行器），是一个接口（基本执行器、缓存执行器）

作用：SqlSession内部通过执行器操作数据库

mapped statement（底层封装对象）

作用：对操作数据库存储封装，包括 sql语句，输入参数、输出结果类型

输入参数类型

java简单类型

hashmap

pojo自定义

输出结果 类型

java简单类型

hashmap

pojo自定义

# 入门程序

## 需求

根据用户id（主键）查询用户信息

根据用户名称模糊查询用户信息

添加用户

删除 用户

更新用户

## 环境

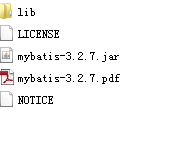
java环境：jdk1.7.0\_72

eclipse：indigo

mysql：5.1

mybatis运行环境（jar包）：

从https://github.com/mybatis/mybatis-3/releases下载，3.2.7版本



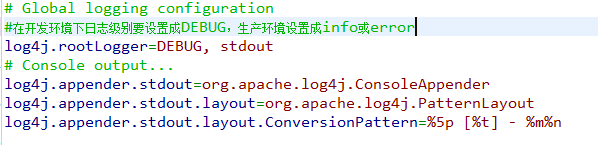
lib下：依赖包

mybatis-3.2.7.jar：核心 包

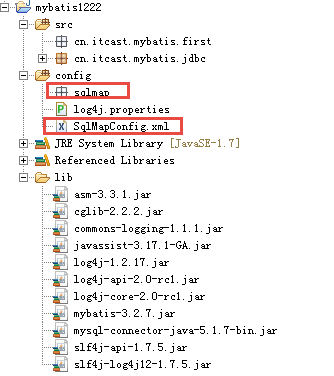
mybatis-3.2.7.pdf，操作指南

加入mysql的驱动包

## log4j.properties



## 工程结构



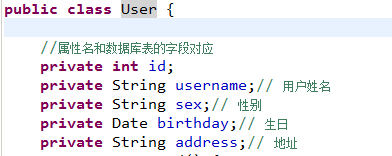
## SqlMapConfig.xml

配置mybatis的运行环境，数据源、事务等。



## 根据用户id（主键）查询用户信息

### 创建po类

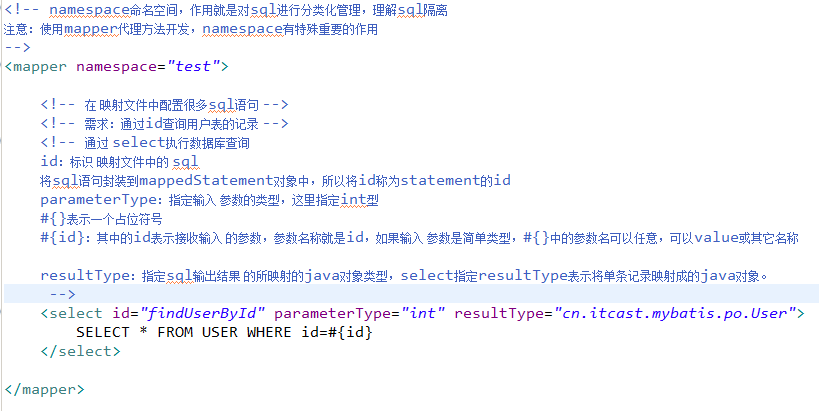


### 映射文件

映射文件命名：

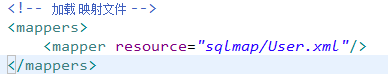
User.xml（原始ibatis命名），mapper代理开发映射文件名称叫XXXMapper.xml，比如：UserMapper.xml、ItemsMapper.xml

在映射文件中配置sql语句。



### 在SqlMapConfig.xml加载映射文件

在sqlMapConfig.xml中加载User.xml:



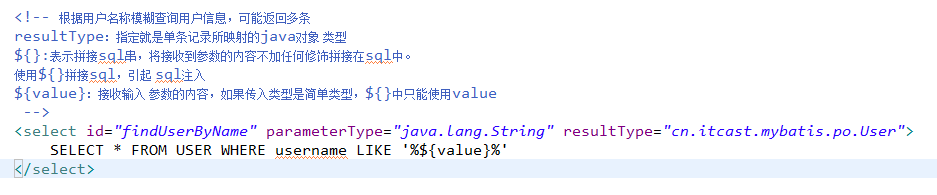
### 程序编写



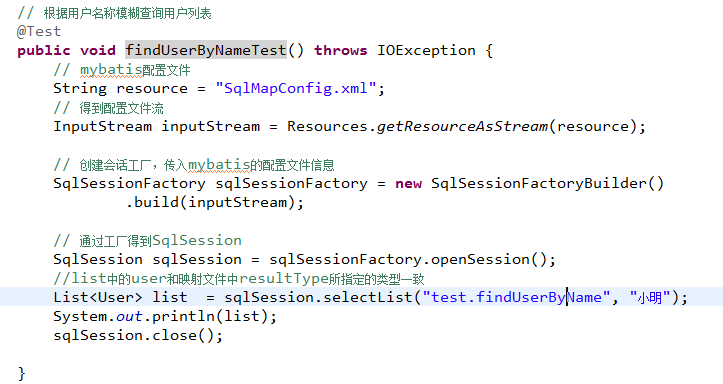
## 根据用户名称模糊查询用户信息

### 映射文件

使用User.xml，添加根据用户名称模糊查询用户信息的sql语句。



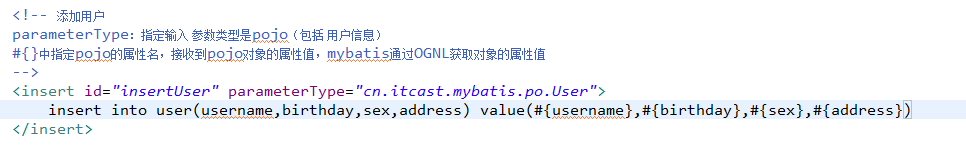
### 程序代码



## 添加用户

### 映射文件

在 User.xml中配置添加用户的Statement



### 程序代码



### 自增主键返回

mysql自增主键，执行insert提交之前自动生成一个自增主键。

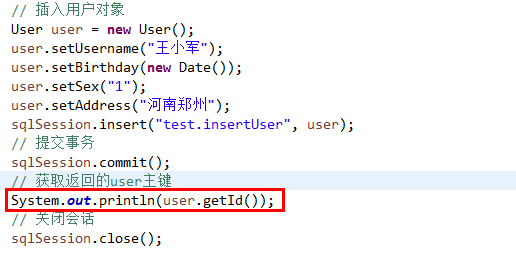
通过mysql函数获取到刚插入记录的自增主键：

LAST\_INSERT\_ID()

执行LAST\_INSERT\_ID()语句顺序相对于insert语句**之后**执行。

修改insertUser定义：





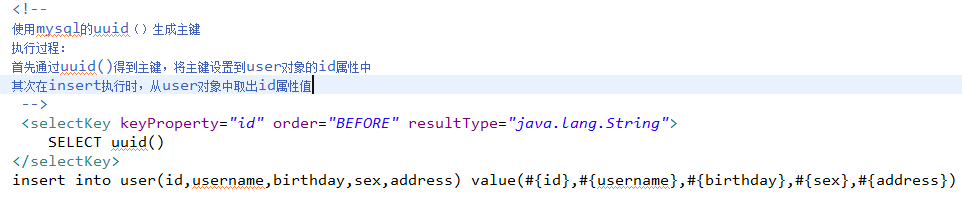
### 非自增主键返回(使用uuid())

使用mysql的uuid()函数生成主键，需要修改表中id字段类型为string，长度设置成35位。

执行思路：

先通过uuid()查询到主键，将主键输入到sql语句中。

执行uuid()语句顺序相对于insert语句**之前**执行。



通过oracle的序列生成主键：

<selectKey keyProperty="id" order="BEFORE" resultType="java.lang.String">

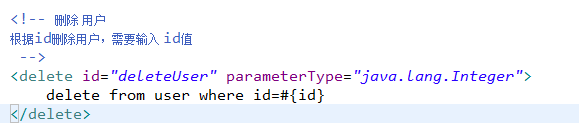
SELECT 序列名.nextval()

</selectKey>

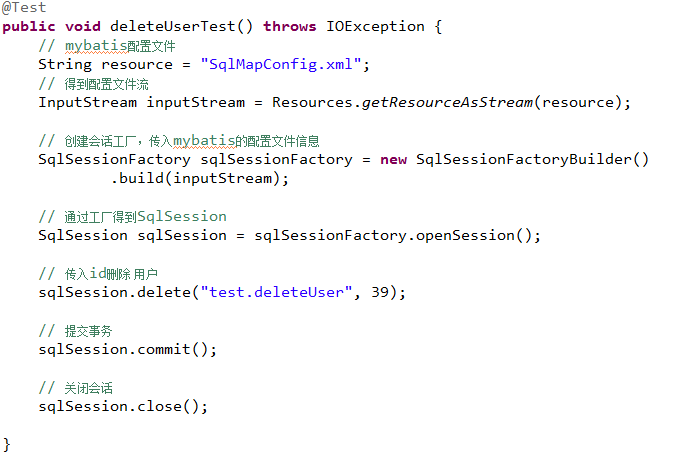
insert into user(id,username,birthday,sex,address) value(#{id},#{username},#{birthday},#{sex},#{address})

## 删除用户

### 映射文件

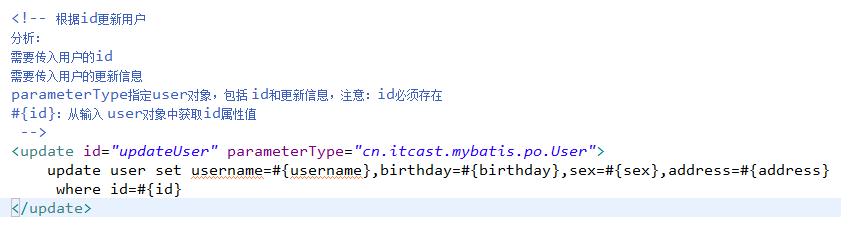


### 代码：



## 更新用户

### 映射文件



### 代码



## 总结

### parameterType

在映射文件中通过parameterType指定输入 参数的类型。

### resultType

在映射文件中通过resultType指定输出结果的类型。

### #{}和${}

#{} 表示一个占位符号，#{}接收输入参数，类型可以是简单类型，pojo、hashmap。

如果接收简单类型，#{}中可以写成value或其它名称。

#{}接收pojo对象值，通过OGNL读取对象中的属性值，通过属性.属性.属性...的方式获取对象属性值。

${} 表示一个拼接符号，会引用sql注入，所以不建议使用${}。

${}接收输入参数，类型可以是简单类型，pojo、hashmap。

如果接收简单类型，${}中只能写成value。

${}接收pojo对象值，通过OGNL读取对象中的属性值，通过属性.属性.属性...的方式获取对象属性值。

### selectOne和selectList

selectOne表示查询出一条记录进行映射。如果使用selectOne可以实现使用selectList也可以实现（list中只有一个对象）。

selectList表示查询出一个列表（多条记录）进行映射。如果使用selectList查询多条记录，不能使用selectOne。

如果使用selectOne报错：

org.apache.ibatis.exceptions.TooManyResultsException: Expected one result (or null) to be returned by selectOne(), but found: 4

## mybatis和hibernate本质区别和应用场景

hibernate：是一个标准ORM框架（对象关系映射）。入门门槛较高的，不需要程序写sql，sql语句自动生成了。

对sql语句进行优化、修改比较困难的。

应用场景：

适用与需求变化不多的中小型项目，比如：后台管理系统，erp、orm、oa。。

mybatis：专注是sql本身，需要程序员自己编写sql语句，sql修改、优化比较方便。mybatis是一个不完全 的ORM框架，虽然程序员自己写sql，mybatis 也可以实现映射（输入映射、输出映射）。

应用场景：

适用与需求变化较多的项目，比如：互联网项目。

企业进行技术选型，以低成本 高回报作为技术选型的原则，根据项目组的技术力量进行选择。

# mybatis开发dao的方法

## SqlSession使用范围

### SqlSessionFactoryBuilder

通过SqlSessionFactoryBuilder创建会话工厂SqlSessionFactory

将SqlSessionFactoryBuilder**当成一个工具类使用即可**，不需要使用单例管理SqlSessionFactoryBuilder。

在需要创建SqlSessionFactory时候，只需要new一次SqlSessionFactoryBuilder即可。

### SqlSessionFactory

通过SqlSessionFactory创建SqlSession，使用单例模式管理sqlSessionFactory（工厂一旦创建，使用一个实例）。

将来mybatis和spring整合后，使用单例模式管理sqlSessionFactory。

### SqlSession

SqlSession是一个面向用户（程序员）的接口。

SqlSession中提供了很多操作数据库的方法：如：selectOne(返回单个对象)、selectList（返回单个或多个对象）、。

**SqlSession是线程不安全的**，在SqlSesion实现类中除了有接口中的方法（操作数据库的方法）还有数据域属性。

**SqlSession最佳应用场合在方法体内，定义成局部变量使用。**

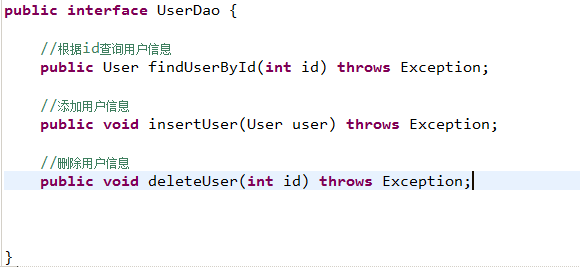
## 原始dao开发方法（程序员需要写dao接口和dao实现类）

### 思路

程序员需要写dao接口和dao实现类。

需要向dao实现类中注入SqlSessionFactory，在方法体内通过SqlSessionFactory创建SqlSession

### dao接口



### dao接口实现类

**public** **class** UserDaoImpl **implements** UserDao {

// 需要向dao实现类中注入SqlSessionFactory

// 这里通过构造方法注入

**private** SqlSessionFactory sqlSessionFactory;

**public** UserDaoImpl(SqlSessionFactory sqlSessionFactory) {

**this**.sqlSessionFactory = sqlSessionFactory;

}

@Override

**public** User findUserById(**int** id) **throws** Exception {

SqlSession sqlSession = sqlSessionFactory.openSession();

User user = sqlSession.selectOne("test.findUserById", id);

// 释放资源

sqlSession.close();

**return** user;

}

@Override

**public** **void** insertUser(User user) **throws** Exception {

SqlSession sqlSession = sqlSessionFactory.openSession();

//执行插入操作

sqlSession.insert("test.insertUser", user);

// 提交事务

sqlSession.commit();

// 释放资源

sqlSession.close();

}

@Override

**public** **void** deleteUser(**int** id) **throws** Exception {

SqlSession sqlSession = sqlSessionFactory.openSession();

//执行插入操作

sqlSession.delete("test.deleteUser", id);

// 提交事务

sqlSession.commit();

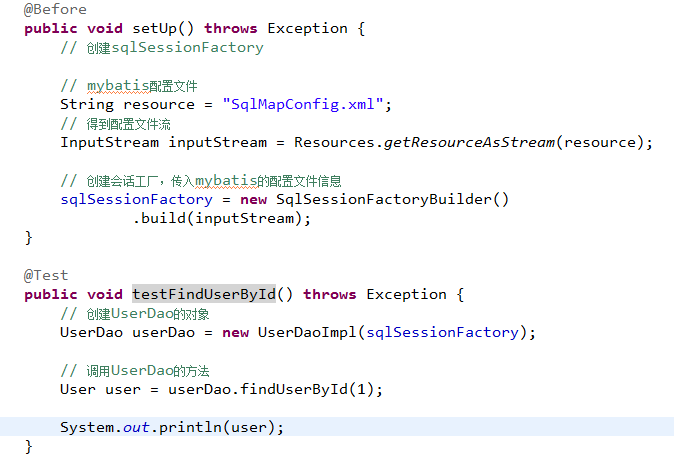
// 释放资源

sqlSession.close();

}

}

### 测试代码：



### 总结原始 dao开发问题

1、dao接口实现类方法中存在大量模板方法，设想能否将这些代码提取出来，大大减轻程序员的工作量。

2、调用sqlsession方法时将statement的id硬编码了

3、调用sqlsession方法时传入的变量，由于sqlsession方法使用泛型，即使变量类型传入错误，在编译阶段也不报错，不利于程序员开发。

## mapper代理方法（程序员只需要mapper接口（相当 于dao接口））

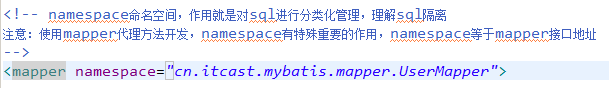
### 思路（mapper代理开发规范）

程序员需要编写mapper.xml映射文件

程序员编写mapper接口需要遵循一些开发规范，mybatis可以自动生成mapper接口实现类代理对象。

开发规范：

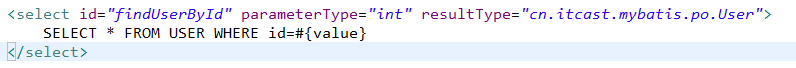
1、**在mapper.xml中namespace等于mapper接口地址**



**2、mapper.java接口中的方法名和mapper.xml中statement的id一致**

**3、mapper.java接口中的方法输入参数类型和mapper.xml中statement的parameterType指定的类型一致。**

**4、mapper.java接口中的方法返回值类型和mapper.xml中statement的resultType指定的类型一致。**





总结：

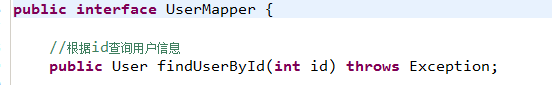
以上开发规范主要是对下边的代码进行统一生成：

User user = sqlSession.selectOne("test.findUserById", id);

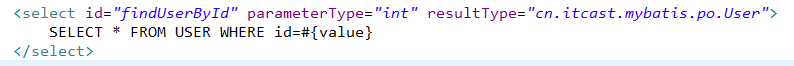
sqlSession.insert("test.insertUser", user);

。。。。

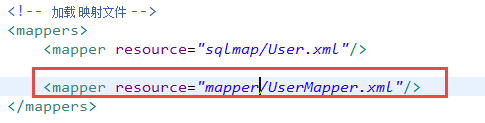
### mapper.java



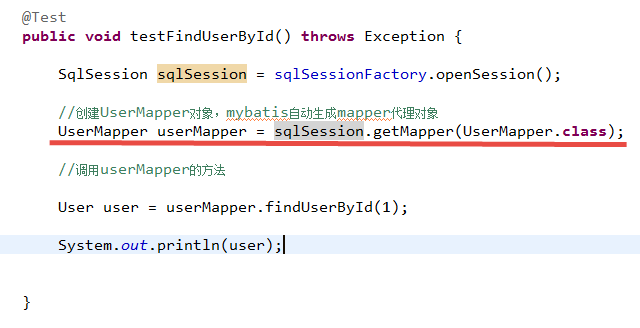
### mapper.xml



### 在SqlMapConfig.xml中加载mapper.xml



### 测试



### 一些问题总结

#### 代理对象内部调用selectOne或selectList

* 如果mapper方法返回单个pojo对象（非集合对象），代理对象内部通过selectOne查询数据库。
* 如果mapper方法返回集合对象，代理对象内部通过selectList查询数据库。

#### mapper接口方法参数只能有一个是否影响系统 开发

mapper接口方法参数只能有一个，系统是否不利于扩展维护。

**系统 框架中，dao层的代码是被业务层公用的。**

**即使mapper接口只有一个参数，可以使用包装类型的pojo满足不同的业务方法的需求。**

注意：持久层方法的参数可以用包装类型、map等。但是service方法中建议不要使用包装类型（不利于业务层的可扩展）。

# SqlMapConfig.xml

mybatis的全局配置文件SqlMapConfig.xml，配置内容如下：

properties（属性）

settings（全局配置参数）

typeAliases（类型别名）

typeHandlers（类型处理器）

objectFactory（对象工厂）

plugins（插件）

environments（环境集合属性对象）

environment（环境子属性对象）

transactionManager（事务管理）

dataSource（数据源）

mappers（映射器）

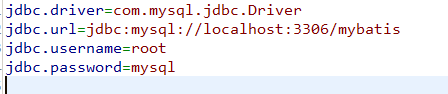
## properties属性

需求：

将数据库连接参数单独配置在db.properties中，只需要在SqlMapConfig.xml中加载db.properties的属性值。

在SqlMapConfig.xml中就不需要对数据库连接参数硬编码。

将数据库连接参数值配置在db.properties中，原因：方便对参数进行统一管理，其它xml方便引用该db.properties。



在sqlMapConfig.xml加载属性文件：



properties特性：

注意： MyBatis 将按照下面的顺序来加载属性：

* 在 properties 元素体内定义的属性首先被读取。
* 然后会读取properties 元素中resource或 url 加载的属性，它会覆盖已读取的同名属性。
* 最后读取parameterType传递的属性，它会覆盖已读取的同名属性。

建议：

不要在properties元素体内添加任何属性值，只将属性值定义在properties文件中。

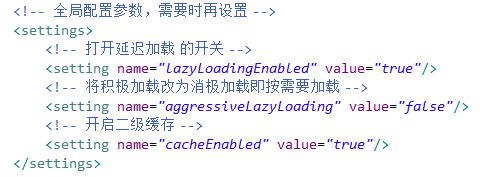


在properties文件中定义属性名最好有一定的特殊性，如：XXXXX.XXXXX.XXXX。避免属性值被其他地方定义的同名属性覆盖。

## settings全局参数配置

mybatis框架在运行时可以调整一些运行参数。

比如：开启二级缓存、开启延迟加载。。



全局参数将会影响mybatis的运行行为。

详细参见“学习资料/mybatis-settings.xlsx”文件







## typeAliases（别名）重点

### 需求

在mapper.xml中，定义很多的statement，statement需要parameterType指定输入参数的类型、需要resultType指定输出结果的映射类型。

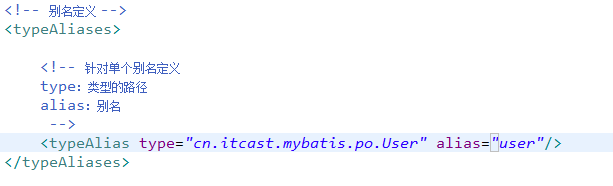
如果在指定类型时输入类型全路径，不方便进行开发，可以针对parameterType或resultType指定的类型定义一些别名，在mapper.xml中通过别名定义，方便开发。

### mybatis默认支持别名

|  |  |
| --- | --- |
| 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

#### 单个别名定义



引用别名：



#### 批量定义别名（常用）



## typeHandlers（类型处理器）

mybatis中通过typeHandlers完成jdbc类型和java类型的转换。

通常情况下，mybatis提供的类型处理器满足日常需要，不需要自定义.

mybatis默认支持的类型处理器：

|  |  |  |
| --- | --- | --- |
| 类型处理器 | **Java**类型 | **JDBC**类型 |
| BooleanTypeHandler | Boolean，boolean | 任何兼容的布尔值 |
| ByteTypeHandler | Byte，byte | 任何兼容的数字或字节类型 |
| ShortTypeHandler | Short，short | 任何兼容的数字或短整型 |
| IntegerTypeHandler | Integer，int | 任何兼容的数字和整型 |
| LongTypeHandler | Long，long | 任何兼容的数字或长整型 |
| FloatTypeHandler | Float，float | 任何兼容的数字或单精度浮点型 |
| DoubleTypeHandler | Double，double | 任何兼容的数字或双精度浮点型 |
| BigDecimalTypeHandler | BigDecimal | 任何兼容的数字或十进制小数类型 |
| StringTypeHandler | String | CHAR和VARCHAR类型 |
| ClobTypeHandler | String | CLOB和LONGVARCHAR类型 |
| NStringTypeHandler | String | NVARCHAR和NCHAR类型 |
| NClobTypeHandler | String | NCLOB类型 |
| ByteArrayTypeHandler | byte[] | 任何兼容的字节流类型 |
| BlobTypeHandler | byte[] | BLOB和LONGVARBINARY类型 |
| DateTypeHandler | Date（java.util） | TIMESTAMP类型 |
| DateOnlyTypeHandler | Date（java.util） | DATE类型 |
| TimeOnlyTypeHandler | Date（java.util） | TIME类型 |
| SqlTimestampTypeHandler | Timestamp（java.sql） | TIMESTAMP类型 |
| SqlDateTypeHandler | Date（java.sql） | DATE类型 |
| SqlTimeTypeHandler | Time（java.sql） | TIME类型 |
| ObjectTypeHandler | 任意 | 其他或未指定类型 |
| EnumTypeHandler | Enumeration类型 | VARCHAR-任何兼容的字符串类型，作为代码存储（而不是索引）。 |

## mappers（映射配置）

### 通过resource加载单个映射文件



### 通过mapper接口加载单个mapper

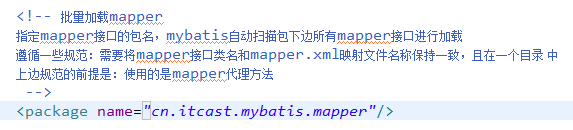




按照上边的规范，将mapper.java和mapper.xml放在一个目录 ，且同名。



### 批量加载mapper(推荐使用)



# 输入映射

通过parameterType指定输入参数的类型，类型可以是简单类型、hashmap、pojo的包装类型

。

## 传递pojo的包装对象

### 需求

完成用户信息的综合查询，需要传入查询条件很复杂（可能包括用户信息、其它信息，比如商品、订单的）

### 定义包装类型pojo

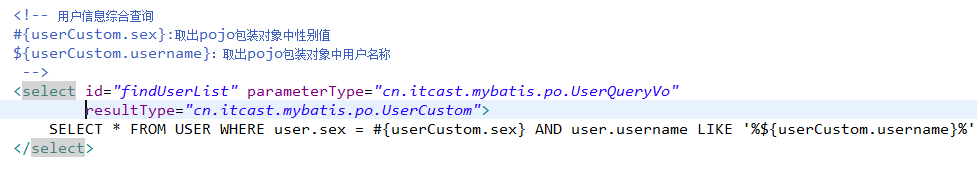
针对上边需求，建议使用自定义的包装类型的pojo。

在包装类型的pojo中将复杂的查询条件包装进去。



### mapper.xml

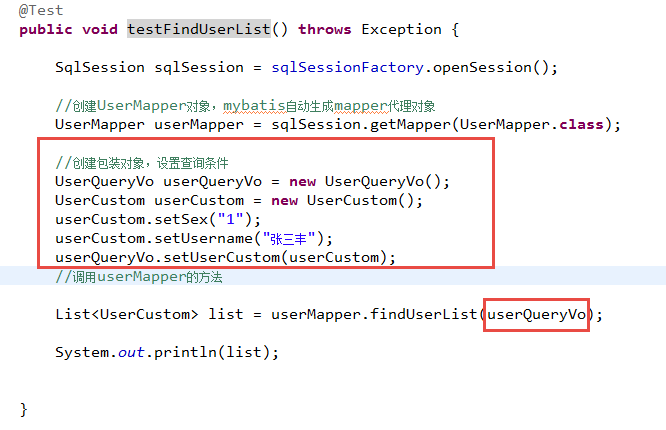
在UserMapper.xml中定义用户信息综合查询（查询条件复杂，通过高级查询进行复杂关联查询）。



### mapper.java



### 测试代码



## 传递hashmap

Sql映射文件定义如下：

<!-- 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：

**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resultType进行输出映射，只有查询出来的列名和pojo中的属性名一致，该列才可以映射成功。**

如果查询出来的列名和pojo中的属性名都不一致，则不创建pojo对象。

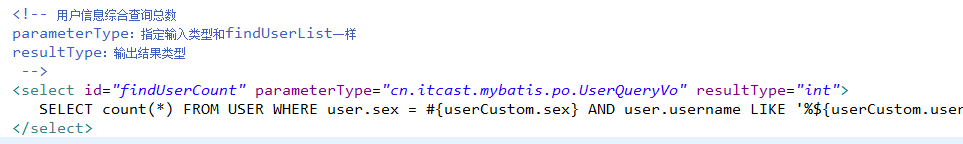
只要查询出来的列名和pojo中的属性有一个一致，则会创建pojo对象。

### 输出简单类型

#### 需求

用户信息的综合查询列表总数，通过查询总数和上边用户综合查询列表才可以实现分页。

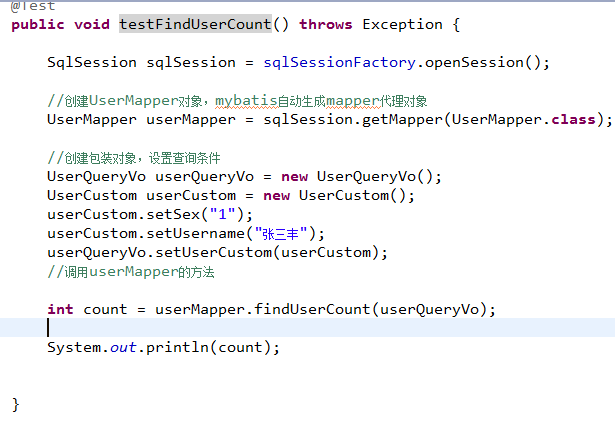
#### mapper.xml



#### mapper.java



#### 测试代码



#### 小结

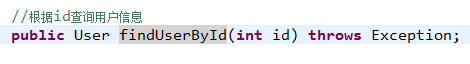
查询出来的结果集只有**一行一列**，可以使用简单类型进行输出映射。

### 输出pojo对象和pojo列表

不管是输出的pojo单个对象还是一个列表（list），在mapper.xml中**resultType指定的类型是一样的。**

在mapper.java指定的方法返回值类型不一样：

1、输出单个pojo对象，方法返回值是单个对象类型



2、输出pojo对象list，方法返回值是List<Pojo>



生成的动态代理对象中是根据mapper方法的返回值类型确定是调用selectOne(返回单个对象调用)还是selectList （返回集合对象调用 ）.

## resultMap

mybatis中使用resultMap完成高级输出结果映射。

### resultMap使用方法

如果查询出来的**列名和pojo的属性名不一致**，通过定义一个resultMap对列名和pojo属性名之间作一个映射关系。

1、定义resultMap

2、使用resultMap作为statement的输出映射类型

### 将下边的sql使用User完成映射

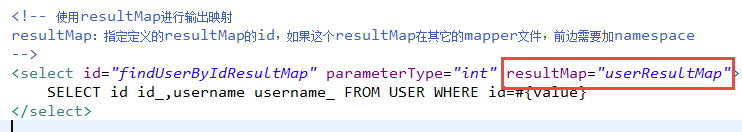
SELECT id id\_,username username\_ FROM USER WHERE id=#{value}

User类中属性名和上边查询列名不一致。

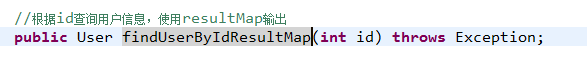
#### 定义reusltMap



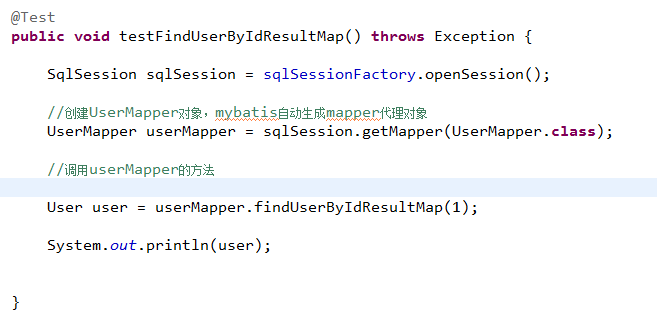
#### 使用resultMap作为statement的输出映射类型



#### mapper.java



#### 测试



## 小结

**使用resultType进行输出映射，只有查询出来的列名和pojo中的属性名一致，该列才可以映射成功。**

**如果查询出来的列名和pojo的属性名不一致，通过定义一个resultMap对列名和pojo属性名之间作一个映射关系。**

# 动态sql

## 什么是动态sql

mybatis核心 对sql语句进行灵活操作，通过表达式进行判断，对sql进行灵活拼接、组装。

## 需求

用户信息综合查询列表和用户信息查询列表总数这两个statement的定义使用动态sql。

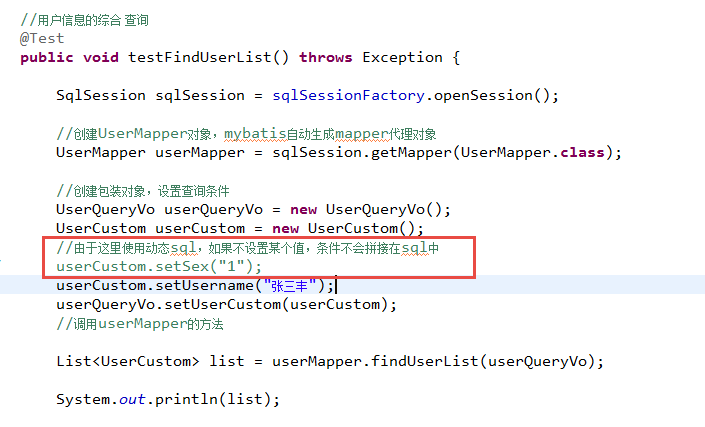
对查询条件进行判断，如果输入参数不为空才进行查询条件拼接。

## mapper.xml





## 测试代码





## sql片段

### 需求

**将上边实现的动态sql判断代码块抽取出来，组成一个sql片段。其它的statement中就可以引用sql片段。**

**方便程序员进行开发。**

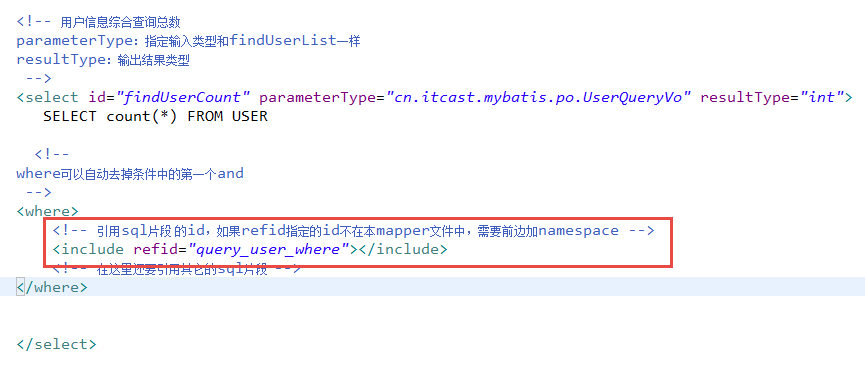
### 定义sql片段



### 引用sql片段

在mapper.xml中定义的statement中引用sql片段：





## foreach

向sql传递数组或List，mybatis使用foreach解析

### 需求

在用户查询列表和查询总数的statement中增加多个id输入查询。

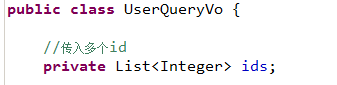
sql语句如下：

两种方法：

SELECT \* FROM USER WHERE id=1 OR id=10 OR id=16

SELECT \* FROM USER WHERE id IN(1,10,16)

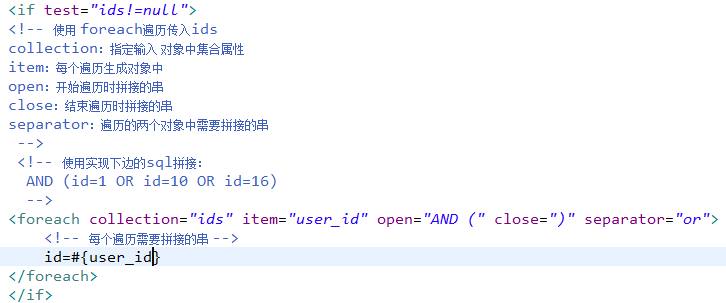
### 在输入参数类型中添加List<Integer> ids传入多个id



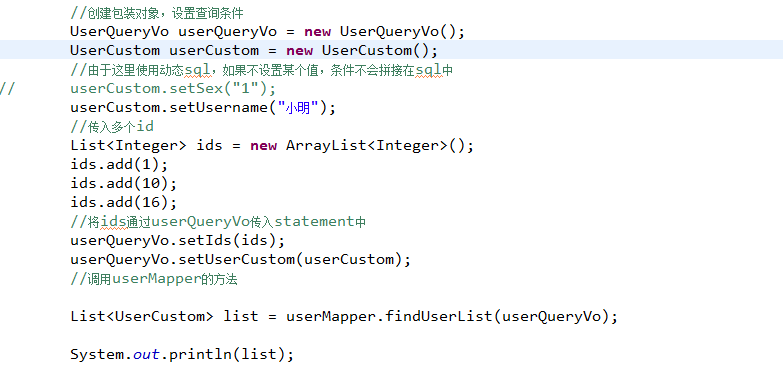
### 修改mapper.xml

WHERE id=1 OR id=10 OR id=16

在查询条件中，查询条件定义成一个sql片段，需要修改sql片段。



### 测试代码



### 另外一个sql的实现：

