mybaits第一天 mybatis基础知识

整个课程是按案例驱动教学，以订单商品案例进行驱动。

mybatis第一天：基础知识（重点）

mybatis介绍

mybatis框架原理(掌握)

mybaits入门程序(掌握)

用户信息进行增、删、改、查

mybatis开发dao层方法：(掌握)

原始dao开发方法（dao接口和实现类需要程序员编写）

mapper代理开发方法（程序员只需要编写接口）

SqlMapConfig.xml（mybatis全局配置文件）(掌握)

mybatis输入映射(掌握)

mybatis输出映射(掌握)

mybatis动态sql(掌握)

第二天：高级知识

mybatis高级映射

mybatis查询缓存

mybatis和spring整合

mybatis逆向工程

# mybatis介绍

mybatis是一个持久层框架，前身是ibatis，mybaits在功能上有了很大升级。

mybatis是apache下的顶级开源项目。

Mybaits现在在github托管，地址：https://github.com/mybatis/mybatis-3/releases

mybatis可以让程序员自己灵活掌握sql语句，需要程序员自己手动编写sql语句。

MyBatis是一个优秀的持久层框架，它对jdbc的操作数据库的过程进行封装，使开发者只需要关注 SQL 本身，而不需要花费精力去处理例如注册驱动、创建connection、创建statement、手动设置参数、结果集检索等jdbc繁杂的过程代码。

Mybatis通过xml或注解的方式将要执行的各种statement（statement、preparedStatemnt、CallableStatement）配置起来，并通过java对象和statement中的sql进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射成java对象并返回。

# 单纯的jdbc程序问题总结

jdbc程序：

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

Connection connection = **null**;

PreparedStatement preparedStatement = **null**;

ResultSet resultSet = **null**;

**try** {

//加载数据库驱动

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

//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", "mysql");

//定义sql语句 ?表示占位符

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

//获取预处理statement

preparedStatement = connection.prepareStatement(sql);

//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");

//向数据库发出sql执行查询，查询出结果集

resultSet = preparedStatement.executeQuery();

//遍历查询结果集

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

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

}

} **catch** (Exception e) {

e.printStackTrace();

}**finally**{

//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();

}

}

}

}

注意点：

PreparedStatement：预编译的Statement，性能高、避免sql注入。

问题：

1. 程序中存在大量的硬编码，系统可维护性差。

2、sql语句硬编码。

解决：将sql语句配置在xml中，这样，sql语句修改了不用去重新编译java代码。

3、数据库连接频繁开关，浪费系统资源。

解决：可以采用数据库连接池。

4、向preparedStatement中设置参数，存在硬编码

解决：将sql语句中的占位符、参数等进行配置，自动将参数设置到preparedStatement中。

5、遍历resultSet时，列名存在硬编码

解决：将resultSet自动映射成List<pojo>(List<User>)

# mybatis框架原理

mybatis配置

SqlMapConfig.xml(mybatis全局配置文件)

XXXmapper.xml、mapper.xml、mapper.xml

sqlSessionFactory（会话工厂）

sqlSession（会话）

Exceutor（执行器 ,底层封装类）

Mapped statement（底层封装类）

mysql

输入参数映射

hashmap

pojo

简单类型

输出映射

hashmap

pojo

简单类型

SqlMapConfig.xml：mybatis全局配置文件，名称不固定的，配置mybatis运行环境（数据源、事务）、mapper映射的配置、全局运行参数。

XXXmapper.xml：mybatis映射文件，配置sql语句，通常命名规则是“表名mapper.xml”，UserMapper.xml，ibatis对映射文件命名规则是“表名.xml”

sqlSessionFactory：会话工厂，创建sqlSession

sqlSession：会话，是一个接口，包括了数据库增、删、改、查，是一个面向程序员的接口。

Exceutor：mybatis提供一个执行器，底层封装类，是一个接口，提供了普通的执行器和缓存执行器。

Exceutor最终要向数据库发起操作请求。

Mapped statement：底层封装类，封装了sql语句、输入 参数类型及输入参数、输出结果类型等。

# 搭建环境

java1.7.0\_72

mysql：5.1

eclipse：indigo

mybatis：mybatis-3.2.7

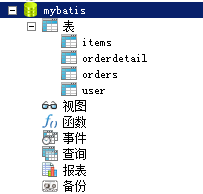
junit：junit-4.9

## 数据库环境

导入以下脚本：

sql\_table.sql：创建表语句、存储过程、触发器

sql\_data.sql：系统初始化数据



## 工程环境

创建java工程

jar：

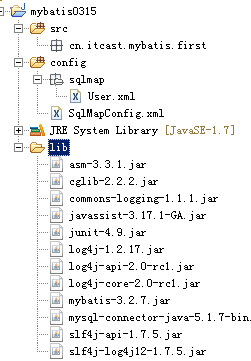
mybatis核心包：mybatis-3.2.7.jar

mybatis依赖包：



mysql驱动包：mysql-connector-java-5.1.7-bin.jar

junit-4.9.jar



# 入门程序

## 需求

根据用户id查询用户信息(只能查询出单个用户)

根据用户名称模糊查询用户信息（可能查询出多条记录）

用户添加

用户删除

用户更新

## 用户id查询用户信息

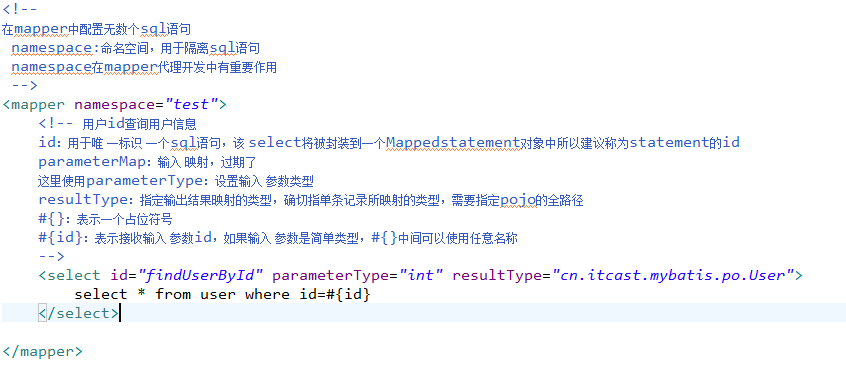
### SqlMapConfig.xml



### po

创建User.java，pojo中的属性名和数据库表的字段名一致。

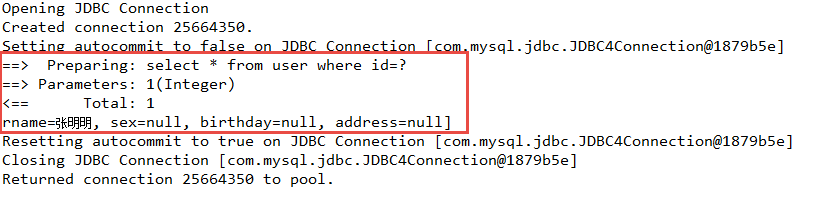
### User.xml



### 编码

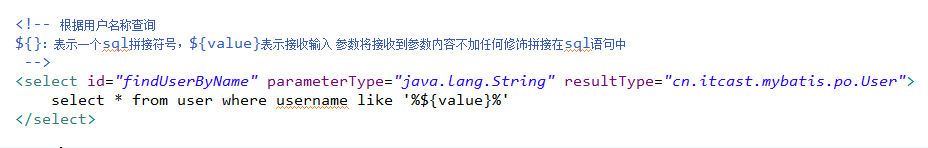


跟踪日志：



## 用户名称查询用户信息

### User.xml



### 编码



## 用户添加、修改、删除

### User.xml



### 编码

参考源代码

## 插入记录获取主键

### 需求

如果存在两张表父子关系，先插入父亲记录，程序员要得到父表新记录的id主键，将父表id作为外键插入到子表中。

### 针对mysql自增主键

实现思路：

在执行insert语句执行之后通过select last\_insert\_id()获取主键，将获取到主键值设置到user对象中。



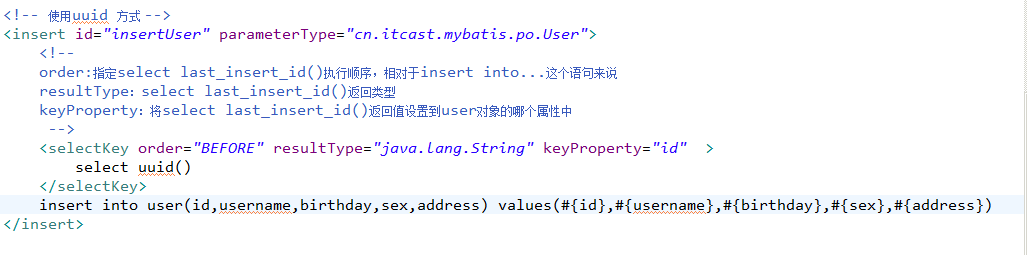
### 针对mysql的uuid

如果从性能上考虑使用uuid作为主键没有mysql自增主键属性高。

如果从数据库可扩展性、可维护性上考虑，mysql自增主键没有uuid好。

实现思路：

在执行insert语句执行之前通过SELECT UUID()生成主键，将主键值设置到user对象中，在insert时从user对象中取到生成的主键。



### 针对oracle的序列生成主键

作为作业，自行实现 。

## mybatis和hibernate本质区别

mybatis入门门槛低，学习成本低，程序员自己编写sql语句，提供sql语句的灵活性。

应用场景：针对需求灵活、变更快、大的项目建议使用mybatis，比如：互联网项目。

hibernate：入门门槛高，不需要程序员编写sql。

应用场景：针对需求固定的中小项目，比如：OA、ERP。

## 小结

mybatis开发思路：

配置全局配置文件，SqlMapConfig.xml

根据需求编写mapper.xml（映射文件）(重点)

在SqlMapconfig.xml中加载映射文件

使用mybatis框架编码实现

parameterType：指定输入映射参数的类型，可以是基本类型、pojo。。。

resultType：指定输出结果映射类型，确切的理解是指定单条记录所映射的类型，可以是pojo、基本类型、hashmap..

#{}：表示一个占位符号，不需要程序员去关注参数内容或类型，mybatis自动进行参数映射，底层使用预编译。

${}：表示一个sql拼接符号，${value}表示接收输入 参数将接收到参数内容不加任何修饰拼接在sql语句中，缺点是无法避免sql注入。底层不使用预编译。

#{}和${}使用OGNL从输入参数中获取参数值，表达式为：属性.属性.属性....

建议使用#{}，有时候必须使用${}，例如：动态拼接表名组成sql语句：

select \* from ${tablename}

再比如将排序字段动态拼接，也必须使用${}：

select \* from user ordre by ${fieldname}

selectOne方法：只查询返回单条记录的sql，如果返回多条记录，使用selectOne抛出异常

能用selectOne就可以使用selectList（List只有一个对象）

selectList方法：将多条记录映射成List<pojo>对象,

# mybatis开发Dao层方法

## sqlSession的使用范围

使用范围：成员变量，方法形参、方法体中局部变量。

### SqlSessionFactoryBuilder

SqlSessionFactoryBuilder用于创建会话工厂SqlSessionFactory，将SqlSessionFactoryBuilder当成 一个**工具类**使用即可。

### SqlSessionFactory

会话工厂，以**单例**方式使用，将会话工厂注入到dao中，在dao中使用会话工厂创建sqlsession。

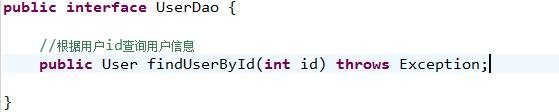
### SqlSession

sqlSession是一个接口，面向程序员的接口，sqlSession实现类是线程不安全，要求sqlSession的使用范围在方法形参或方法体内局部变量。

## 原始dao开发方法

由于历史遗留原因，早期使用的是ibatis，使用ibatis采用原始dao开发方法，需要程序员编写接口和接口实现类。

### dao接口



### dao接口实现 类

需要将SqlSessionFactory注入dao实现类中。



## mapper代理的开发

mybatis提供的开发方法，只需要程序员编写mapper接口(相当于dao接口)，程序员在开发mapper代理接口时需要遵循一些开发规范，mybatis框架才可以生成mapper接口实现代理对象。

### 原始dao开发中的问题

1、每个dao实现方法是相同的代码执行流程（逻辑）

2、selectList和selectOne接收的第二参数是object，如果输入参数类型错误不会在编译阶段报错

3、sqlSession调用方法存在重复，设想能否将mapper.xml中的statement的标签类型和sqlsession的方法关联起来，不用手动编写sqlSession调用方法。

### 开发规范(重点)

如何解决这问题，需要程序员开发时遵循一些开规范：

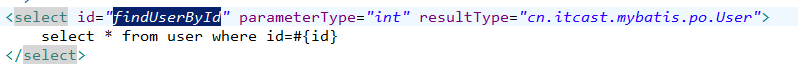
1、要将mapper映射文件和dao接口关联。

mapper映射文件中的namespace等于dao接口全路径



2、要将mapper方法（dao方法）和mapper.xml中statement进行一对一关联。

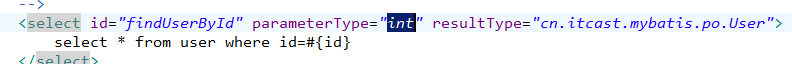
mapper接口方法名和statement的id一致。





3、要将dao方法输入参数类型和statement中的parameterType保持一致。





4、要将dao方法返回值类型和statement中的resultType保持一致。

返回值类型：

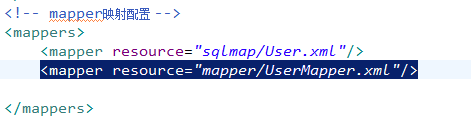
如果dao方法返回List<pojo>，pojo的类型和resultType保持一致，生成的代理对象内部调用selectList。

如果dao方法返回pojo（单个对象），pojo的类型和resultType保持一致，生成的代理对象内部调用selectOne。

### mapper.xml(映射文件)



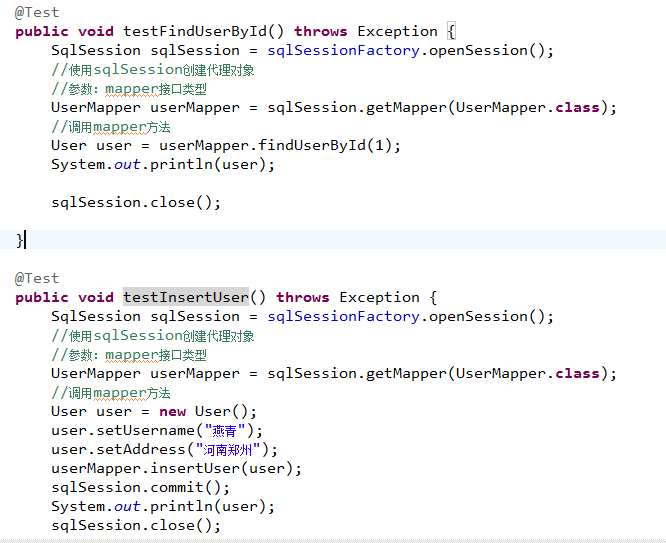
将mapper.xml在SqlMapconfig.xml进行配置，注册mapper。



### mapper.java(java接口)



### 测试



## 小结

实际开发中推荐mapper代理的开发方法。

mapper方法的输入参数的类型和mapper.xml中statement的parameterTyte一致，mapper接口方法只有一个参数，是否影响系统可扩展性？？

mapper属于持久层接口，持久层是给各各service接口服务，所以这里可以将多个参数（如果有多个参数）封装到一个包装对象，其实不影响系统可扩展性。

包装对象：自定义pojo，pojo属性也是pojo，即pojo将各各参数pojo组合起来。

如果mapper方法返回值为pojo单个对象，代理对象内部使用selectOne调用。

如果mapper方法返回值为List集合对象，代理对象内部使用selectList调用

# SqlMapConfig.xml(全局配置文件)

SqlMapConfig.xml中要配置内容如下：

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）

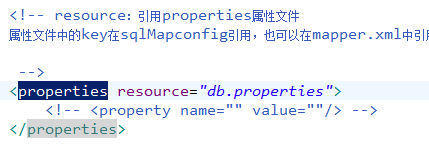
environment（环境子属性对象）

transactionManager（事务管理）

dataSource（数据源）

mappers（映射器）

## properties



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。

因此，通过parameterType传递的属性具有最高优先级，resource或 url 加载的属性次之，最低优先级的是 properties 元素体内定义的属性。

**在实际使用时不需要在properties元素体内定义属性，只需要引用properties文件即可。**

**properties文件中定义的key一定要具有特殊命名规范。**

## setting

mybatis运行参数在需要时进行设置，需要谨慎，如果设置错误影响 mybatis的运行。

## typeAliases

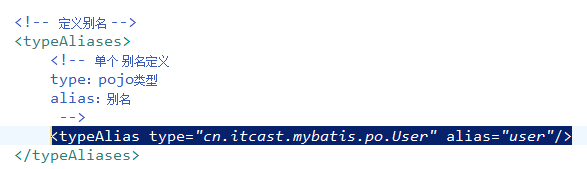
mybatis开发中mapper.xml映射文件开发工具量最大。

可以对mapper.xml中parameterType和resultMap设置别名，方便开发。

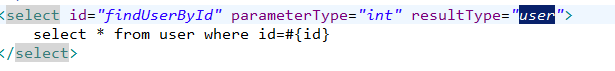
mybatis支持别名：

|  |  |
| --- | --- |
| **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 |
|  |  |

### 单个别名定义



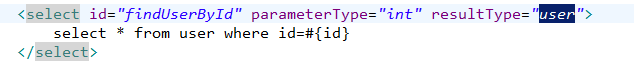
使用别名：



### 批量别名定义

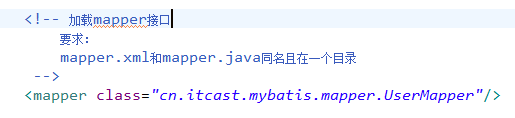


使用别名：



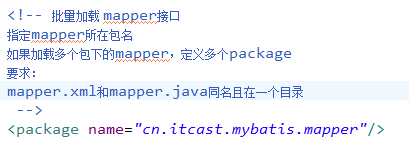
## mapper映射器

### 通过mapper接口加载





### 批量扫描mapper



# 输入映射

mybatis提供parameterType及parameterMap定义输入参数类型，parameterMap已经过期。

parameterType：指定输入参数类型，包括 ：基本类型、pojo、hashmap。

## 输入包装类型对象

### 需求

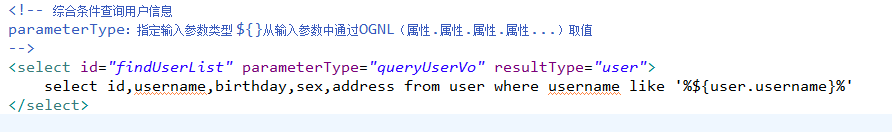
根据综合条件进行用户信息查询，查询条件可能包括：用户信息、商品信息、订单信息

### 定义包装类型

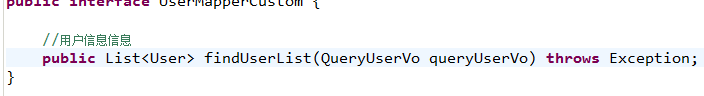
parameterType需要输入包装类型（将用户信息、商品信息、订单信息对象包装进来）



### mapper.xml



### mapper.java



# 输出映射

输出映射：是将sql查询的结果集（可能是多列多行），映射成java对象。

## resultType

resultType：指定将单条记录映射的java对象类型。

要求：sql查询的列名和resultType指定的pojo的属性名一致，方可映射，如果所有的列名和pojo的属性名都不一致，pojo对象是null。

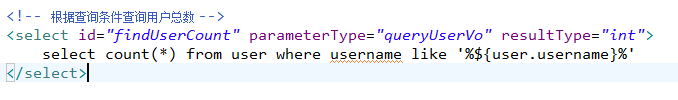
### 返回简单类型

当查询结果集是单行单列可以返回简单类型。

需求：

根据用户查询条件，查询出记录的总数（int）。

mapper.xml



mapper.java



## resultMap

resultMap：如果sql查询的列名和pojo的属性名不一致，可以使用resultMap将列名和pojo的属性名作一个对应关系，就可以映射成功了.

使用resultMap可以完成一些高级映射：

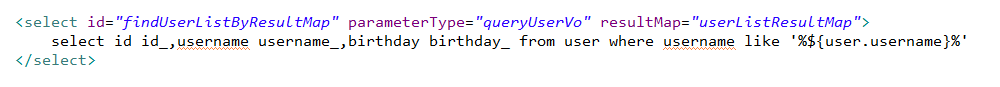
将一些列聚合到一个pojo的对象中，完成一对一查询映射。

将多行多列聚合到一个List<pojo>中，完成一对多查询映射。

### mapper.xml

* 定义resultMap





### mapper.java



# 动态sql

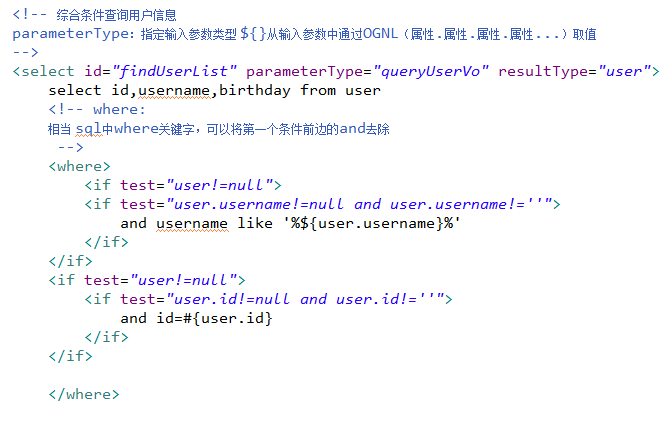
## 需求

根据综合条件进行用户信息查询，根据用户输入的查询条件灵活组成sql语句。

将mapper.xml中重复的sql语句抽取出来，只需要维护一处sql。

## if

解决：根据综合条件进行用户信息查询，根据用户输入的查询条件灵活组成sql语句



## sql片段

解决：将mapper.xml中重复的sql语句抽取出来，只需要维护一处sql

mybatis提供sql片段，将重复的sql语句定义成一个片段，由其它statement进行引用。

### 定义sql片段



### 引用sql片段

