mybatis第一天 mybatis基础知识

教学思路：

整个教学思路以订单商品的项目作为驱动。

mybatis第一天：基础知识

mybatis是一个java持久层框架，java中操作关系型 数据库用的是jdbc，mybatis是对jdbc的一个封装。

1、从一个jdbc程序开始，通过jdbc 程序找到使用原生态的jdbc开发程序，存在哪些问题？？通过学习mybatis，mybatis是如何解决这些问题。

2、mybatis的架构（重点）

3、mybatis的入门程序（重点）

实现 用户的查询、添加、修改、删除。

4、mybatis开发dao的两种方法。（重点）

原始的dao开发方式（dao接口和dao实现都需要编写）

mapper代理方式（只需要写dao接口）

5、输入映射类型和输出映射类型。

6、动态sql

第二天：高级知识

高级映射查询（一对一、一对多、多对多）（重点）

查询缓存

延迟加载

mybatis和spring整合（重点）

mybatis逆向工程 。

# 开发环境

jdk1.7.0\_72

eclipse：eclipse-3.7-indigo

mysql：mysql5.1

## 创建数据库



先导入sql\_table.sql，再导入sql\_data.sql(记录系统的初始化数据)

通常需要提供初始化数据的数据库脚本。



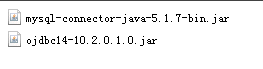
# jdbc编程中问题

企业开发中，根据项目大小、特点进行技术选型 ，jdbc操作数据库时效率是很高的，jdbc也是技术选型的参考。

## jdbc程序

参考教案

需要数据库的驱动包：



上边是mysql的驱动，下边是oracle的驱动。

## jdbc问题总结

1、数据库连接频繁的创建和关闭，缺点浪费数据库的资源，影响操作效率

设想：使用数据库连接池

2、sql语句是硬编码，如果需求变更需要修改sql，就需要修改java代码，需要重新编译，系统不易维护。

设想：将sql语句 统一配置在文件中，修改sql不需要修改java代码。

3、通过preparedStatement向占位符设置参数，存在硬编码（ 参数位置，参数）问题。系统不易维护。

设想：将sql中的占位符及对应的参数类型配置在配置文件中，能够自动输入 映射。

4、遍历查询结果集存在硬编码（列名）。

设想：自动进行sql查询结果向java对象的映射（输出映射）。

# mybatis架构(重点)

## mybatis介绍

MyBatis 本是[apache](http://baike.baidu.com/view/28283.htm)的一个开源项目[iBatis](http://baike.baidu.com/view/628102.htm), 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis，实质上Mybatis对ibatis进行一些改进。 目前mybatis在github上托管。git（分布式版本控制，当前比较流程）

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。

## mybatis架构

SqlMapConfig.xml（mybatis全局配置文件，名称不固定）

运行环境（数据源、事务）

mapper.xml(配置sql语句), mapper.xml.....

SqlSessionFactory（会话工厂）

作用：创建SqlSession

SqlSession（向面用户的接口，数据库操作方法）

作用：操作数据库

Executor（数据库操作的执行器）

Executor是一个接口，有两个实现 （默认执行器和缓存执行器）

mysql

MappedStatement（mybatis的封装对象）

封装：sql语句(占位符)

输入 映射

java简单类型

pojo类型

hashmap

输出 映射

java简单类型

pojo类型

hashmap

# mybatis入门程序

## 需求

实现用户查询：

根据用户id（主键）查询用户信息（单条记录）

根据用户名称模糊查询用户信息（多条记录）

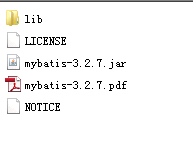
用户添加

用户删除

用户修改

## 导入jar包

从mybatis管网下载（地址：https://github.com/mybatis/mybatis-3/releases）



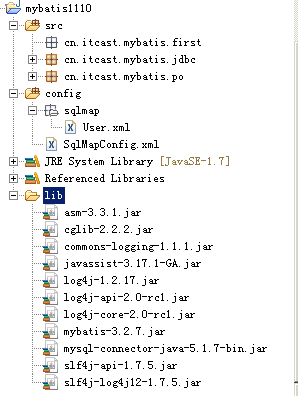
mybatis-3.2.7.pdf---操作手册

mybatis-3.2.7.jar--核心 jar包

依赖的jar包



## 工程结构



## log4j.properties（公用文件）

# Global logging configuration，建议开发环境中要用debug

log4j.rootLogger=DEBUG, stdout

# Console output...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

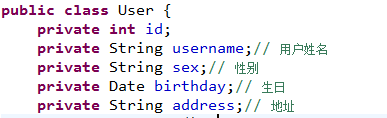
## SqlMapConfig.xml(公用文件)

通过SqlMapConfig.xml加载mybatis运行环境。



## 根据id查询用户

### pojo（User.java）



### User.xml（重点）

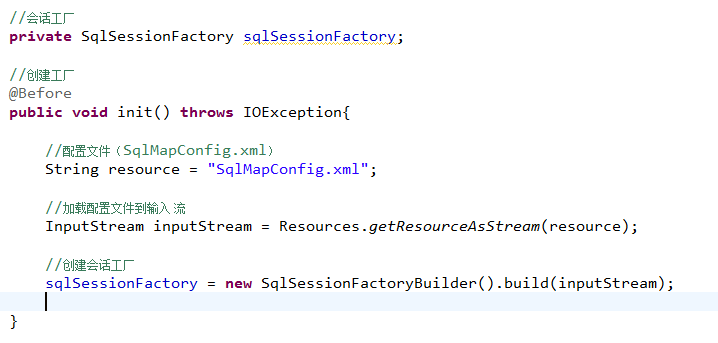
建议命名规则：表名+mapper.xml

早期ibatis命名规则：表名.xml



### 编码

创建SqlSessionFactory:

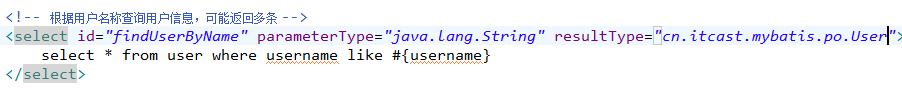




## 根据用户名称模糊查询用户信息

根据用户名称模糊查询用户信息可能返回多条记录。

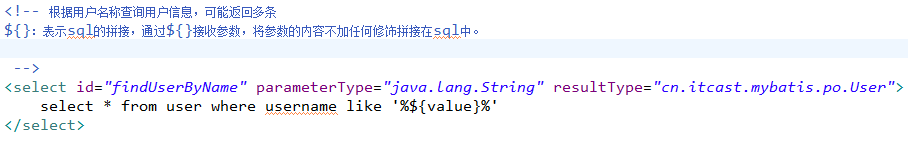
### User.xml



### 编码



### 使用${}接收参数



## mybatis开发过程小结

1、编写SqlMapConfig.xml

2、编写mapper.xml

定义了statement

3、编程通过配置文件创建SqlSessionFactory

4、通过SqlSessionFactory获取SqlSession

5、通过SqlSession操作数据库

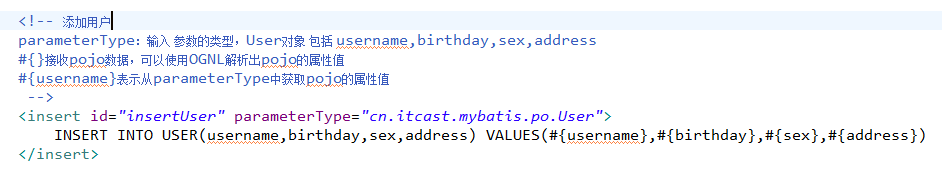
如果执行添加、更新、删除需要调用SqlSession.commit()

6、SqlSesion使用完成要关闭

## 用户添加

向用户表插入一条记录。

### User.xml



### 编码



### 主键返回

需求：user对象插入到数据库后，新记录的主键要通过user对象返回，通过user获取主键值。

解决思路：

通过LAST\_INSERT\_ID()获取刚插入记录的自增主键值，在insert语句执行后，执行select LAST\_INSERT\_ID()就可以获取自增主键。

User.xml修改：

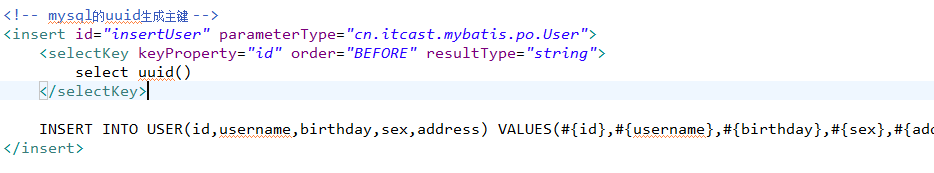


使用mysql的uuid机制生成主键：

使用uuid生成主键的好处是不考虑数据库移植后主键冲突问题。

实现思路：

先查询uuid得到主键，将主键设置到user对象中，将user对象插入数据库。

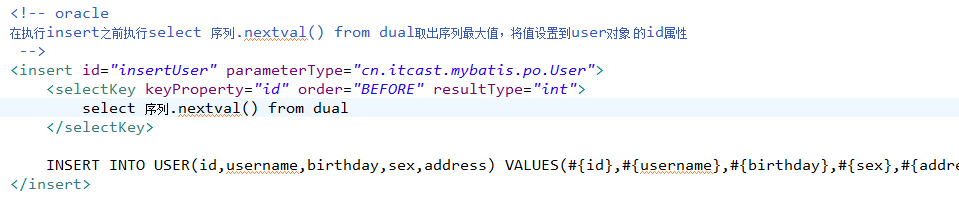


实现 oracle数据库主键返回，如何做？？

oracle没有自增主键机制，使用序列完成主键生成。

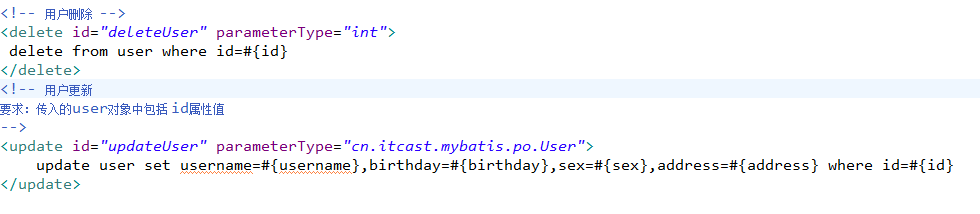
实现思路：

先查询序列得到主键，将主键设置到user对象中，将user对象插入数据库。



## 用户删除和更新

### User.xml



### 编码

// 测试根据id删除用户(得到单条记录)

@Test

**public** **void** testDeleteUser() {

// 通过sqlSessionFactory创建sqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

// 通过sqlSession操作数据库

**try** {

sqlSession.delete("test.deleteUser", 35);

// 需要提交事务

sqlSession.commit();

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

// 关闭sqlSession

sqlSession.close();

}

}

// 测试根据id更新用户(得到单条记录)

@Test

**public** **void** testUpdateUser() {

// 通过sqlSessionFactory创建sqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

// 通过sqlSession操作数据库

// 创建更新数据对象，要求必须包括 id

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

user.setId(35);

user.setUsername("燕青");

user.setAddress("河南郑州");

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

user.setSex("1");

**try** {

sqlSession.update("test.updateUser", user);

// 需要提交事务

sqlSession.commit();

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

// 关闭sqlSession

sqlSession.close();

}

System.*out*.println("用户的id=" + user.getId());

}

## Mybatis解决jdbc编程的问题

1. 数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库链接池可解决此问题。

解决：在SqlMapConfig.xml中配置数据链接池，使用连接池管理数据库链接。

1. Sql语句写在代码中造成代码不易维护，实际应用sql变化的可能较大，sql变动需要改变java代码。

解决：将Sql语句配置在XXXXmapper.xml文件中与java代码分离。

1. 向sql语句传参数麻烦，因为sql语句的where条件不一定，可能多也可能少，占位符需要和参数一一对应。

解决：Mybatis自动将java对象映射至sql语句，通过statement中的parameterType定义输入参数的类型。

1. 对结果集解析麻烦，sql变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成pojo对象解析比较方便。

解决：Mybatis自动将sql执行结果映射至java对象，通过statement中的resultType定义输出结果的类型。

## mybatis与hibernate重要区别

企业开发进行技术选型 ，考虑mybatis与hibernate适用场景。

mybatis：入门简单，程序员容易上手开发，节省开发成本 。mybatis需要程序员自己编写sql语句，是一个不完全 的ORM框架，对sql修改和优化非常容易实现 。

mybatis适合开发需求变更频繁的系统，比如：互联网项目。

hibernate：入门门槛高，如果用hibernate写出高性能的程序不容易实现。hibernate不用写sql语句，是一个非常标准ORM（对象关系映射）框架。

hibernate适合需求固定，对象数据模型稳定，中小型项目，比如：企业OA系统。

总之，企业在技术选型时根据项目实际情况，以降低成本和提高系统的可维护性为出发点进行技术选型。

## 总结

### SqlMapConfig.xml

是mybatis全局配置文件，只有一个，名称不固定的，主要mapper.xml，mapper.xml中配置 sql语句

### mapper.xml

mapper.xml是以statement为单位进行配置。（把一个sql称为一个statement），satatement中配置 sql语句、parameterType输入参数类型（完成输入映射）、resultType输出结果类型（完成输出映射）。

还提供了parameterMap配置输入参数类型（过期了，不推荐使用了）

还提供resultMap配置输出结果类型（完成输出映射），明天重点讲通过resultMap完成复杂数据类型的映射（一对多，多对多映射）

### #{}

表示一个占位符，向占位符输入参数，mybatis自动进行java类型和jdbc类型的转换。

程序员不需要考虑参数的类型，比如：传入字符串，mybatis最终拼接好的sql就是参数两边加单引号。

#{}接收pojo数据，可以使用OGNL解析出pojo的属性值

### ${}

表示sql的拼接，通过${}接收参数，将参数的内容不加任何修饰拼接在sql中。

${}也可以接收pojo数据，可以使用OGNL解析出pojo的属性值

缺点：不能防止sql注入。

### selectOne

用于查询单条记录，不能用于查询多条记录，否则异常：

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

### selectList

用于查询多条记录，可以用于查询单条记录的。

# mybatis开发dao的方法

## SqlSession作用范围

是使用局部变量、成员变量。。。。？？？

### SqlSessionFactoryBuilder

SqlSessionFactoryBuilder是以工具类方式来使用，需要创建sqlSessionFactory就new一个SqlSessionFactoryBuilder。

### sqlSessionFactory

正常开发时，以单例方式管理sqlSessionFactory，整个系统运行过程中sqlSessionFactory只有一个实例，将来和spring整合后由spring以单例方式管理sqlSessionFactory。

### SqlSession

sqlSession是一个面向用户（程序员）的接口，程序员调用sqlSession的接口方法进行操作数据库。

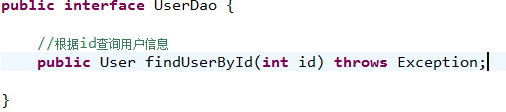
sqlSession能否以单例 方式使用？？

由于sqlSession是线程不安全，所以sqlSession最佳应用范围在方法体内，在方法体内定义局部变量使用sqlSession。

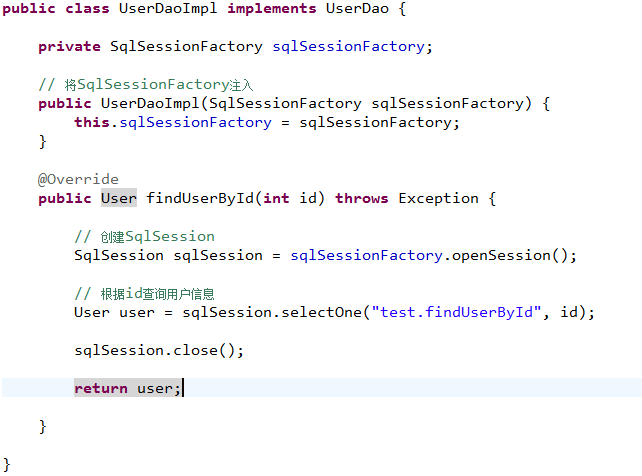
## 原始dao开发方式

程序员需要写dao接口和dao 的实现 类

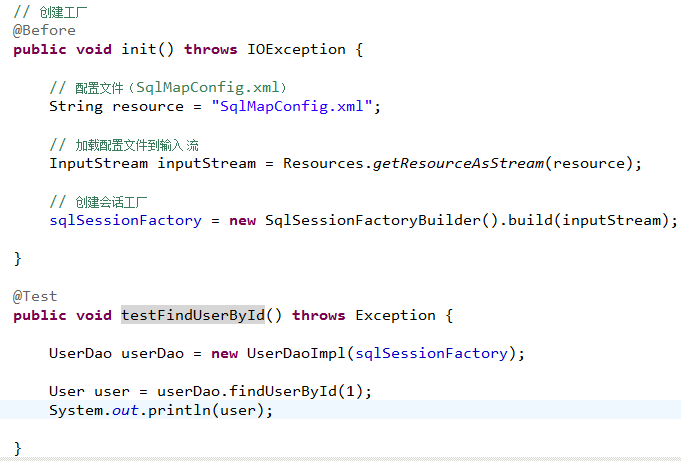
### dao接口



### dao接口实现



### 测试代码



## mapper代理的方式

程序员只需要写dao接口，dao接口实现对象由mybatis自动生成代理对象。

本身dao在三层架构中就是一个通用的接口。

### 上边原始dao开发方式的问题

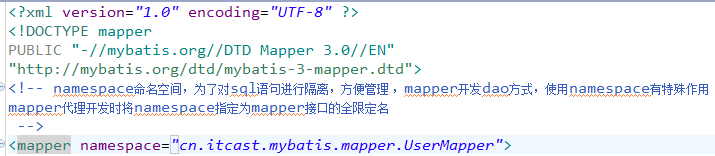
1、dao的实现类中存在重复代码，整个mybatis操作的过程代码模板重复（先创建sqlsession、调用sqlsession的方法、关闭sqlsession）

2、dao的实现 类中存在硬编码，调用sqlsession方法时将statement的id硬编码。

### mapper开发规范

要想让mybatis自动创建dao接口实现类的代理对象，必须遵循一些规则：

1、mapper.xml中namespace指定为mapper接口的全限定名



此步骤目的：通过mapper.xml和mapper.java进行关联。

2、mapper.xml中statement的id就是mapper.java中方法名

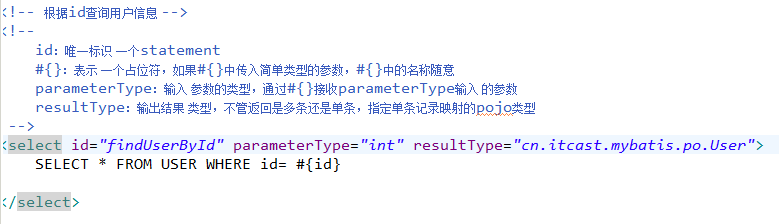
3、mapper.xml中statement的parameterType和mapper.java中方法输入参数类型一致

4、mapper.xml中statement的resultType和mapper.java中方法返回值类型一致.

### mapper.xml（映射文件）

mapper映射文件的命名方式建议：表名Mapper.xml

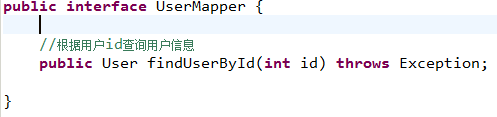
namespace指定为mapper接口的全限定名



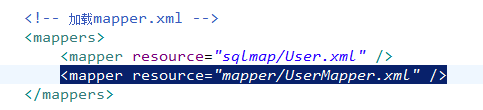
### mapper接口

mybatis提出了mapper接口，相当 于dao 接口。

mapper接口的命名方式建议：表名Mapper



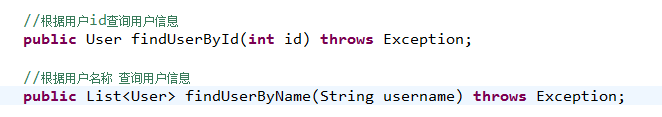
### 将mapper.xml在SqlMapConfig.xml中加载



### mapper接口返回单个对象和集合对象

不管查询记录是单条还是多条，在 statement中resultType定义一致，都是单条记录映射的pojo类型。

mapper接口方法返回值，如果是返回的单个对象，返回值类型是pojo类型，生成的代理对象内部通过selectOne获取记录，如果返回值类型是集合对象，生成的代理对象内部通过selectList获取记录。



### 问题

#### 返回值的问题

如果方法调用的statement，返回是多条记录，而mapper.java方法的返回值为pojo，此时代理对象通过selectOne调用，由于返回多条记录，所以报错：

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

#### 输入参数的问题

使用mapper代理的方式开发，mapper接口方法输入 参数只有一个，可扩展性是否很差？？

可扩展性没有问题，因为dao层就是通用的，可以通过扩展pojo（定义pojo包装类型）将不同的参数（可以是pojo也可以是其他的简单类型）传入进去。

# sqlMapConfig.xml

SqlMapConfig.xml中配置的内容和顺序如下：

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属性定义

可以把一些通用的属性值配置在属性文件中，加载到mybatis运行环境内。

比如：创建db.properties配置数据库连接参数。



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。

**优先级由高到低为: parameterType > resource或url加载的属性>properties元素体内定义的属性**

建议使用properties，不要在properties中定义属性，只引用定义的properties文件中属性，并且properties文件中定义的key要有一些特殊的规则。目的就是为了避免以上三种传递属性出现同key时的加载优先级问题，避免因为这种代码而情况引起的错误率。

## settings全局参数配置

mybatis运行时可以调整一些全局参数（相当于软件的运行参数），参考：<mybatis-settings.xlsx>

根据使用需求进行参数配置。

注意：小心配置，配置参数会影响mybatis的执行。

ibatis的全局配置参数中包括很多的性能参数（最大线程数，最大待时间。。。），通过调整这些性能参数使ibatis达到高性能的运行，mybatis没有这些性能参数，由mybatis自动调节。

## typeAliases（类型别名）(常用)

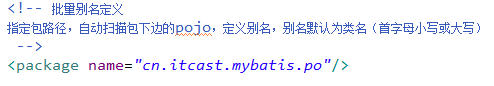
可以将parameterType、resultType中指定的类型 通过别名引用。

### mybaits提供了很多别名

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

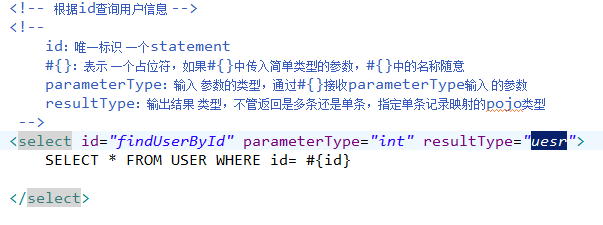
### 自定义别名





### 使用别名

在parameterType、resultType中使用别名：



## typeHandlers（类型处理器）

类型处理器将java类型和jdbc类型进行映射。

mybatis默认提供很多类型处理器，一般情况下够用了。

Mybatis支持类型处理器：

|  |  |  |
| --- | --- | --- |
| **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（映射器）



# 输入和输出映射

通过parameterType完成输入映射，通过resultType和resultMap完成输出映射。

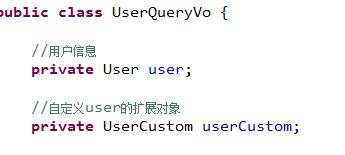
## parameterType传递pojo包装对象

可以定义pojo包装类型扩展mapper接口输入参数的内容。

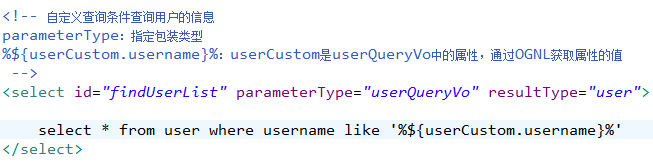
需求：

自定义查询条件查询用户信息，需要向statement输入查询条件，查询条件可以有user信息、商品信息。。。。

### 包装类型



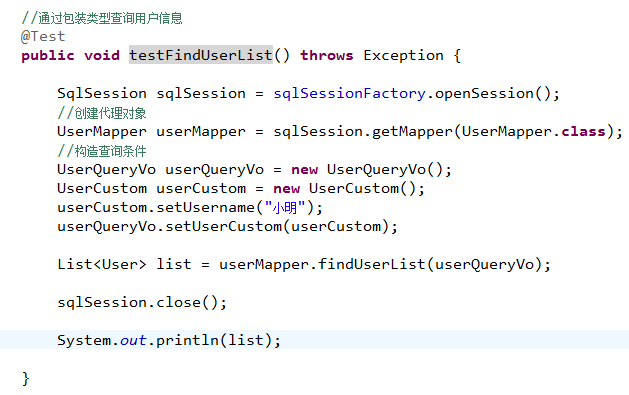
### mapper.xml



### mapper.java



### 测试



### 异常

如果parameterType中指定属性错误，异常，找不到getter方法（无法获取到属性值）：

org.apache.ibatis.exceptions.PersistenceException:

### Error querying database. Cause: org.apache.ibatis.reflection.ReflectionException: There is no getter for property named 'userCusto' in 'class cn.itcast.mybatis.po.UserQueryVo'

### Cause: org.apache.ibatis.reflection.ReflectionException: There is no getter for property named 'userCusto' in 'class cn.itcast.mybatis.po.UserQueryVo'

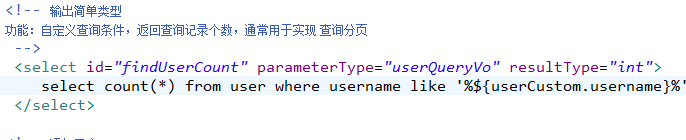
注意：如果将来和spring整合后，不是通过调用getter方法来获取属性值，通过反射强读取pojo的属性值。（强读取）

## resultType

指定输出结果的类型（pojo、简单类型、hashmap..），将sql查询结果映射为java对象 。

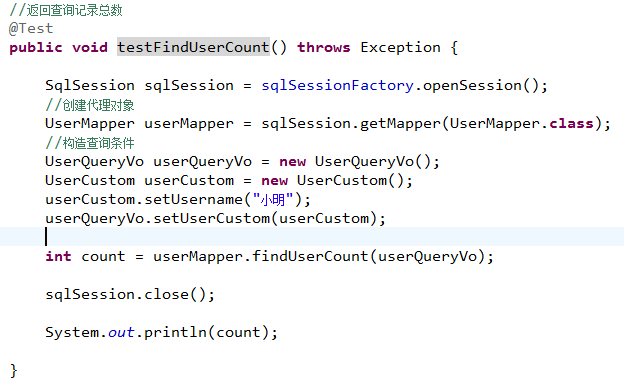
### 返回简单类型

mapper.xml



mapper.java





注意：

如果查询记录结果集为一条记录且一列再使用返回简单类型。

## resultMap(入门)

resultType ：指定输出结果的类型（pojo、简单类型、hashmap..），将sql查询结果映射为java对象 。

使用resultType注意：sql查询的列名要和resultType指定pojo的属性名相同，指定相同 属性方可映射成功，如果sql查询的列名要和resultType指定pojo的属性名全部不相同，list中无法创建pojo对象的（都是null）。

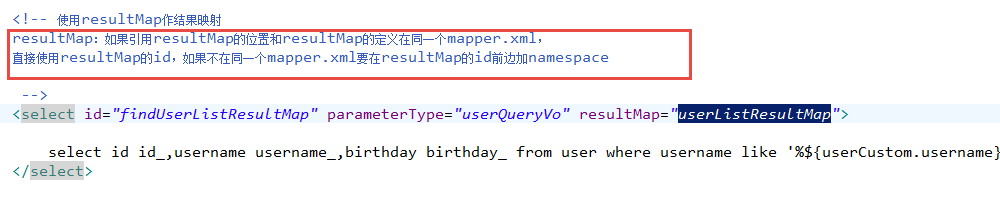
resultMap：将sql查询结果映射为java对象。

如果sql查询列名和最终要映射的pojo的属性名不一致，使用resultMap将列名和pojo的属性名做一个对应关系 （列名和属性名映射配置）

### resultMap配置



### 使用resultMap



### mapper.java



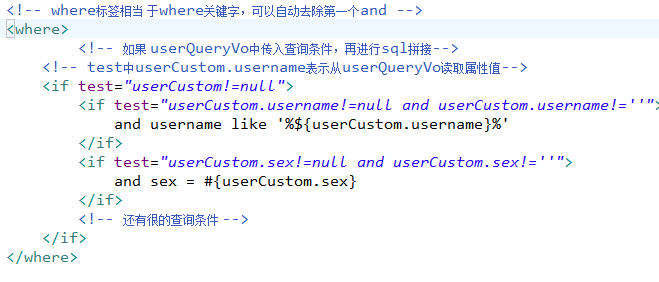
# 动态sql(重点)

mybatis重点是对sql的灵活解析和处理。

## 需求

将自定义查询条件查询用户列表和查询用户列表总记录数改为动态sql

## if和where



## sql片段

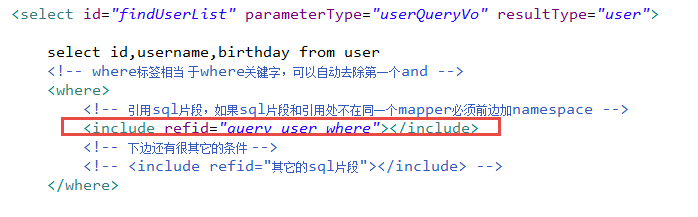
通过sql片段可以将通用的sql语句抽取出来，单独定义，在其它的statement中可以引用sql片段。

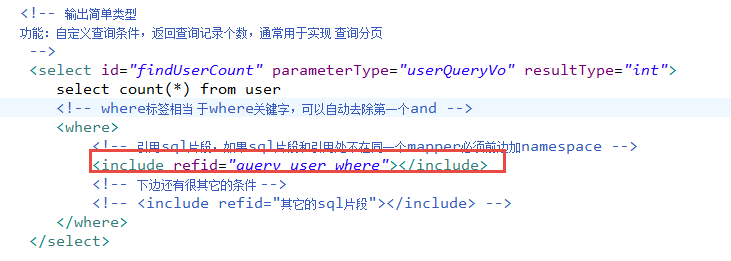
通用的sql语句，常用：where条件、查询列

### sql片段的定义



### 引用sql片段





## foreach

在statement通过foreach遍历parameterType中的集合类型。

需求：

根据多个用户id查询用户信息。

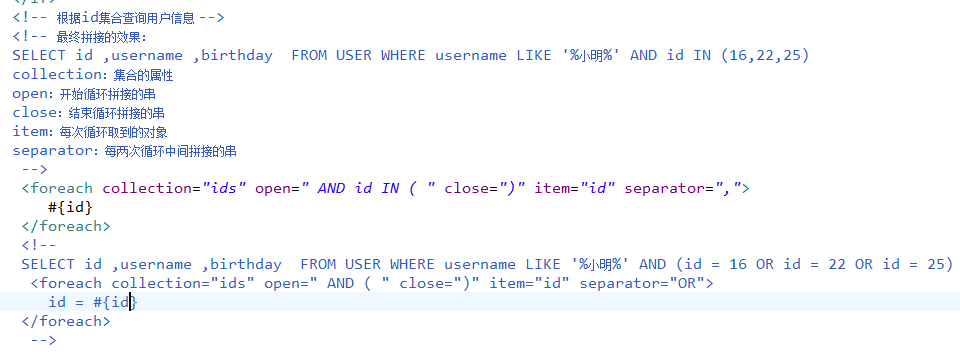
### 在userQueryVo中定义list<Integer> ids

在userQueryvo中定义list<Integer> ids存储多个id

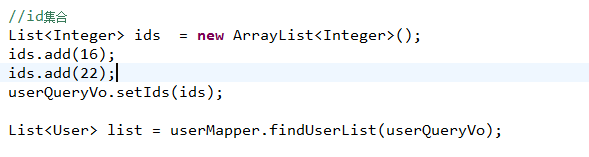


### 修改（动态sql）sql片段的where语句

使用foreach遍历list：



### 测试代码



mybatis第二天 高级映射 查询缓存 和spring整合

# 复习

mybatis是什么？

mybatis是一个持久层框架，是apache下的开源项目，前身是itbatis，是一个不完全的ORM框架，mybatis提供输入和输出的映射，需要程序员自己写sql语句，mybatis重点对 sql语句的灵活操作。

适合用于：需求变化频繁， 数据模型不固定的项目，例如：互联网项目。

mybatis架构？

SqlMapConfig.xml（名称不固定），配置内容：数据源、事务、properties、typeAliases、settings、mapper配置。

SqlSessionFactory--会话工厂，作用是创建SqlSession，实际开发中以单例模式管理 SqlSessionFactory。

SqlSession--会话，是一个面向用户（程序员）的接口，使用mapper代理方法开发是不需要程序员直接调用sqlSession的方法。是线程不安全，最佳适用场合方法体内。

mybatis开发dao的方法：

1、原始dao开发方法，需要程序员编写dao接口和实现类，此方法在当前企业中还有使用，因为ibatis使用的就是原始dao开发方法。

2、mapper代理方法，程序员只需要写mapper接口（相当于dao接口），mybatis自动根据mapper接口和mapper接口对应的statement自动生成代理对象（接口实现类对象）。

开发需要遵循规则：

1）mapper.xml中namespace是mapper接口的全限定名

2）mapper.xml中statement的id为mapper接口方法名

3）mapper.xml中statement的输入映射类型（parameterType）和mapper接口方法输入参数类型一致

4）mapper.xml中statement的输出映射类型（resultType）和mapper接口方法返回结果类型一致

resultType和resultMap都可以完成输出映射：

resultType映射要求sql查询的列名和输出映射pojo类型的属性名一致

resultMap映射时对sql查询的列名和输出映射pojo类型的属性名作一个对应关系。

动态sql：

#{}和${}完成输入参数的属性值获取，通过OGNL获取parameterType指定pojo的属性名。

#{}：占位符号，好处防止sql注入

${}：sql拼接符号

if

where

foreach

# 课程计划

1、使用resultMap完成高级映射(重点)

学习商品订单数据模型（一对一、一对多、多对多）

resultMap实现一对一、一对多、多对多

延迟加载

2、查询缓存(重点)

一级缓存

二级缓存

3、mbatis和spring整合(重点)

4、mybatis逆向工程(常用)

# 商品订单数据模型



学会在企业中如何去分析陌生表的数据模型：

1、学习单表记录了什么东西（去学习理解需求）

2、学习单表重要字段的意义（优先学习不能为空的字段）

3、学习表与表之间的关系（一对一、一对多、多对多）

通过表的外键分析表之间的关系

注意：分析表与表之间的关系时是要建立在具体 的业务意义基础之上

user

orders

user\_id：用户id

user\_id

orderdetail

order\_id：订单id

item\_id：商品id

order\_id

items

item\_id

一对一

一对多

一对多

一对一

一对一

一对多

orders--> items

通过orderdetail

建立 一对多

items -->orders

通过orderdetail

建立 一对多

总之：多对多

用户和商品是多对多

用户表user：

记录了购买商品的用户

订单表orders：

记录了用户所创建的订单信息

订单明细表orderdetail：

记录了用户创建订单的详细信息

商品信息表items：

记录了商家提供的商品信息

分析表与表之间的关系：

用户user和订单orders：

user---->orders：一个用户可以创建多个订单 一对多

orders-->user：一个订单只能由一个用户创建 一对一

订单orders和订单明细orderdetail：

orders-->orderdetail：一个订单可以包括多个订单明细 一对多

orderdetail-->orders：一个订单明细只属于一个订单 一对一

订单明细orderdetail和商品信息items：

orderdetail-->items：一个订单明细对应一个商品信息 一对一

items--> orderdetail：一个商品对应多个订单明细 一对多

# 一对一查询

## 需求

查询订单信息关联查询用户信息

## sql语句

查询语句：

先确定主查询表：订单信息表

再确定关联查询表：用户信息

通过orders关联查询用户使用user\_id一个外键，只能关联查询出一条用户记录就可以使用内连接

SELECT

orders.\*,

user.username,

user.sex

FROM

orders,

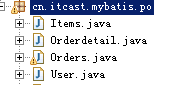
USER

WHERE orders.user\_id = user.id

## 使用resultType实现

### 创建po类

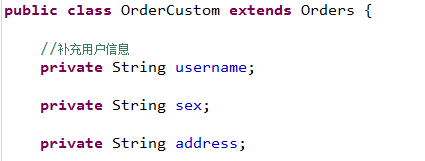
基础的单表的 po类：



### 一对一查询映射的pojo

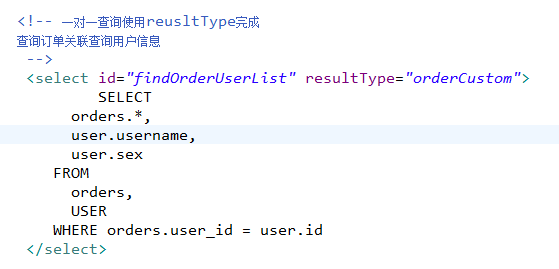
创建pojo包括 订单信息和用户信息，resultType才可以完成映射。

创建OrderCustom作为自定义pojo，继承sql查询列多的po类。

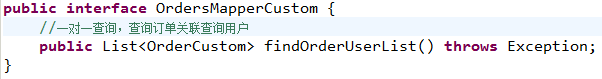


### mapper.xml

定义mapper.xml文件，



### mapper.java

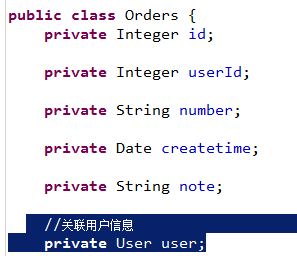


## 使用resultMap实现一对一

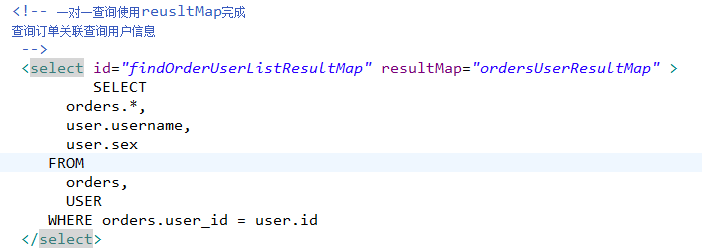
### resultMap映射思路

resultMap提供一对一关联查询的映射和一对多关联查询映射，一对一映射思路：将关联查询的信息映射到pojo中，如下：

在Orders类中创建一个User属性，将关联查询的信息映射到User属性中。



### mapper.xml



### resultMap定义



### mapper.java



## 小结

resultType：要自定义pojo 保证sql查询列和pojo的属性对应，这种方法相对较简单，所以应用广泛。

resultMap：使用association完成一对一映射需要配置一个resultMap，过程有点复杂，**如果要实现延迟加载就只能用resultMap实现** ，如果为了方便对关联信息进行解析，也可以用association将关联信息映射到pojo中方便解析。

# 一对多查询

## 需求

查询所有订单信息及订单下的订单明细信息。

## sql语句

主查询表：订单表

关联查询表：订单明细

SELECT

orders.\*,

user.username,

user.sex ,

orderdetail.id orderdetail\_id,

orderdetail.items\_num,

orderdetail.items\_id

FROM

orders,

USER,

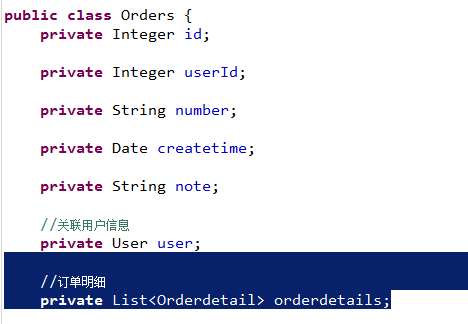
orderdetail

WHERE orders.user\_id = user.id AND orders.id = orderdetail.orders\_id

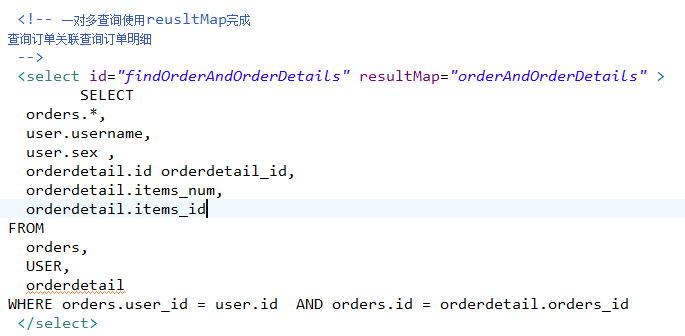
## resultMap进行一对多映射思路

resultMap 提供collection完成关联信息映射到集合对象中。

在orders类中创建集合属性：



## mapper.xml



## resultMap定义



## mapper.java



# 一对多查询(复杂)

## 需求

查询所有用户信息，关联查询订单及订单明细信息及商品信息，订单明细信息中关联查询商品信息

## sql

主查询表：用户信息

关联查询：订单、订单明细，商品信息

SELECT

orders.\*,

user.username,

user.sex ,

orderdetail.id orderdetail\_id,

orderdetail.items\_num,

orderdetail.items\_id,

items.name items\_name,

items.detail items\_detail

FROM

orders,

USER,

orderdetail,

items

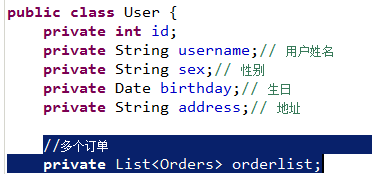
WHERE orders.user\_id = user.id AND orders.id = orderdetail.orders\_id AND items.id = orderdetail.items\_id

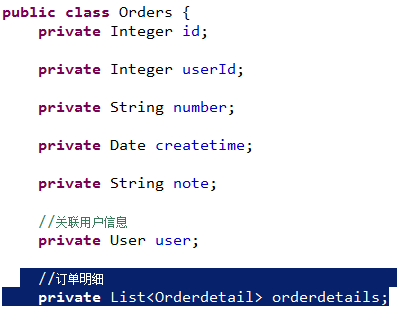
## pojo定义

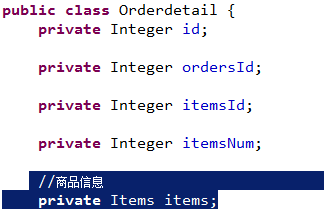
在user.java中创建映射的属性：集合 List<Orders> orderlist

在Orders中创建映射的属性：集合List<Orderdetail> orderdetails

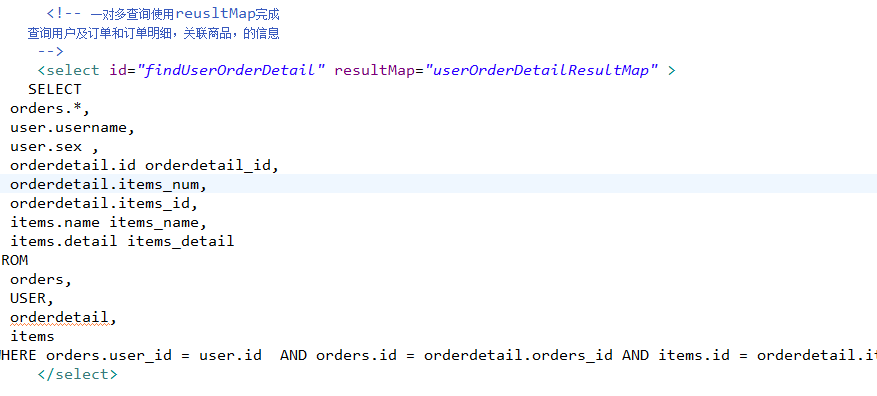
在Orderdetail中创建商品属性：pojo Items items







## mapper.xml



## resultMap



## mapper.java



# 多对多查询（自己实现 自学）

一对多是多对多的特例。

需求1：

查询显示字段：用户账号、用户名称、用户性别、商品名称、商品价格(最常见)

企业开发中常见明细列表，用户购买商品明细列表，

使用resultType将上边查询列映射到pojo输出。

需求2：

查询显示字段：用户账号、用户名称、购买商品数量、商品明细（鼠标移上显示明细）

使用resultMap将用户购买的商品明细列表映射到user对象中。

# 延迟加载

**延迟加载和hibernate中的懒加载是一个意思，作用也是一样的**

## 使用延迟加载意义

**延迟加载相当于hibernate中的懒加载**

在进行数据查询时，**为了提高数据库查询性能**，尽量使用单表查询，因为单表查询比多表关联查询速度要快。

如果查询单表就可以满足需求，一开始先查询单表，当需要关联信息时，再关联查询，当需要关联信息再查询这个叫延迟加载。

mybatis中resultMap提供延迟加载功能，通过resultMap配置延迟加载。

## 配置mybatis支持延迟加载

|  |  |  |  |
| --- | --- | --- | --- |
| 设置项 | 描述 | 允许值 | 默认值 |
| lazyLoadingEnabled | 全局性设置懒加载。如果设为‘false’，则所有相关联的都会被初始化加载。 | true | false | false |
| aggressiveLazyLoading | 当设置为‘true’的时候，懒加载的对象可能被任何懒属性全部加载。否则，每个属性都**按需加载**。 | true | false | true |

<!-- 全局配置参数 -->

<settings>

<!-- 延迟加载总开关 -->

<setting name=*"lazyLoadingEnabled"* value=*"true"* />

<!-- 设置按需加载 -->

<setting name=*"aggressiveLazyLoading"* value=*"false"* />

</settings>

## 延迟加载实现

### 实现思路

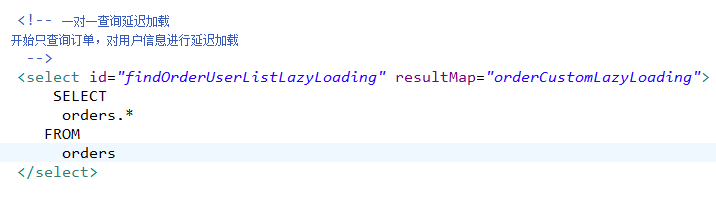
需求：

查询订单及用户的信息，一对一查询。

刚开始只查询订单信息

当需要用户时调用 Orders类中的getUser()方法执行延迟加载 ，向数据库发出sql。

### mapper.xml



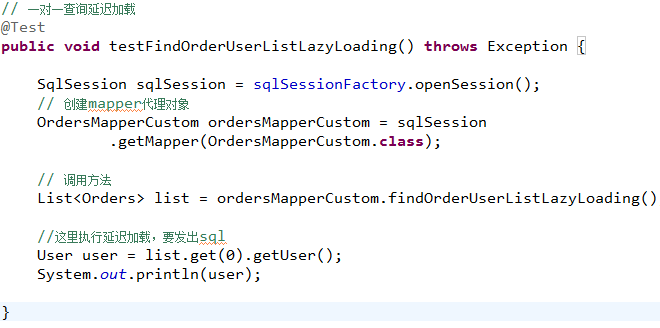
### resultMap



### mapper.java



### 测试代码



## 一对多延迟加载（自学）

一对多延迟加载的方法同一对一延迟加载，在collection标签中配置select内容。

本部分内容自学。

# resultType、resultMap、延迟加载使用场景总结

延迟加载：

延迟加载实现的方法多种多样，在只查询单表就可以满足需求，为了提高数据库查询性能使用延迟加载，再查询关联信息。（**有时可以不一定非要使用mybatis的延迟加载实现**）

mybatis提供延迟加载的功能用于service层。

**resultType**：（没有特殊要求建议使用resultType）

作用：

将查询结果按照sql列名pojo属性名一致性映射到pojo中。

场合：

常见一些明细记录的展示，将关联查询信息全部展示在页面时，此时可直接使用resultType将每一条记录映射到pojo中，在前端页面遍历list（list中是pojo）即可。

**resultMap**：

使用association和collection完成一对一和一对多高级映射。

**association**：

作用：

将关联查询信息映射到一个pojo类中。

场合：

为了**方便获取关联信息**可以使用association将关联订单映射为pojo，比如：查询订单及关联用户信息。

**collection**：

作用：

将关联查询信息映射到一个list集合中。

场合：

为了**方便获取关联信息**可以使用collection将关联信息映射到list集合中，比如：查询用户权限范围模块和功能，可使用collection将模块和功能列表映射到list中。

# 查询缓存

## 缓存的意义

将用户经常查询的数据放在缓存（内存）中，用户去查询数据就不用从磁盘上(关系型数据库数据文件)查询，从缓存中查询，从而提高查询效率，解决了高并发系统的性能问题。

用户、程序

**系统持久层(mybatis)**

**缓存区域(数据库取出数据缓存**)

数据库

缓存数据

系统业务层(spring)

缓存区域（业务数据进行缓存）

系统控制层,表现层(springmvc)

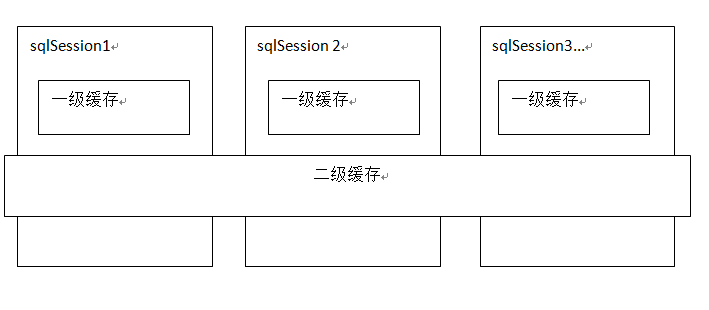
缓存区域(响应的jsp页面进行缓存)

缓存数据

缓存数据

## mybatis持久层缓存

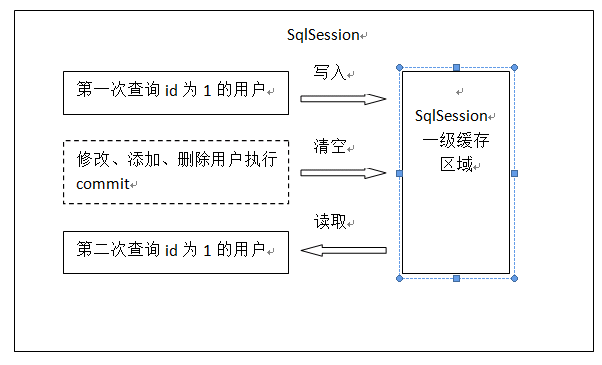
mybatis提供一级缓存和二级缓存



mybatis一级缓存是一个SqlSession级别，sqlsession只能访问自己的一级缓存的数据，二级缓存是跨sqlSession，是mapper级别的缓存，对于mapper级别的缓存不同的sqlsession是可以共享的。

## 一级缓存

### 原理



第一次发出一个查询sql，sql查询结果写入sqlsession的一级缓存中，缓存使用的数据结构是一个

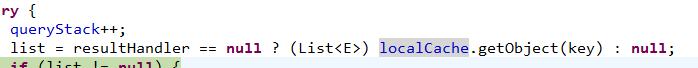
map<key,value>

key：hashcode+sql+sql输入参数+输出参数（sql的唯一标识）

value：用户信息

同一个sqlsession再次发出相同的sql，就从缓存中取不走数据库。如果两次中间出现commit操作（修改、添加、删除），本sqlsession中的一级缓存区域全部清空，下次再去缓存中查询不到所以要从数据库查询，从数据库查询到再写入缓存。

每次查询都先从缓存中查询：



如果缓存中查询到则将缓存数据直接返回。

如果缓存中查询不到就从数据库查询：



### 一级缓存配置

mybatis默认支持一级缓存不需要配置。

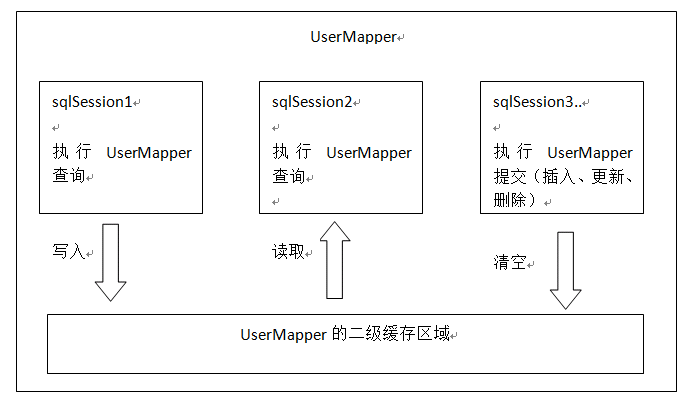
**注意：mybatis和spring整合后进行mapper代理开发，不支持一级缓存，mybatis和spring整合，spring按照mapper的模板去生成mapper代理对象，模板中在最后统一关闭sqlsession。**

### 一级缓存测试



## 二级缓存

### 原理



二级缓存的范围是mapper级别（mapper同一个命名空间），mapper以命名空间为单位创建缓存数据结构，结构是map<key、value>。

每次查询先看是否开启二级缓存，如果开启从二级缓存的数据结构中取缓存数据，



如果从二级缓存没有取到，再从一级缓存中找，如果一级缓存也没有，从数据库查询。

### mybatis二级缓存配置

在核心配置文件SqlMapConfig.xml中加入

<setting name=*"cacheEnabled"* value=*"true"*/>

|  |  |  |  |
| --- | --- | --- | --- |
|  | 描述 | 允许值 | 默认值 |
| cacheEnabled | 对在此配置文件下的所有cache 进行全局性开/关设置。 | true false | true |

要在你的Mapper映射文件中添加一行：  <cache /> ，表示**此mapper**开启二级缓存。

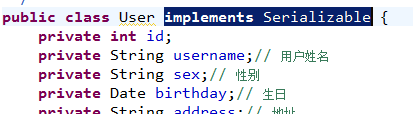
### 查询结果映射的pojo序列化

mybatis二级缓存需要将查询结果映射的pojo实现 java.io.serializable接口，如果不实现则抛出异常：

org.apache.ibatis.cache.CacheException: Error serializing object. Cause: java.io.NotSerializableException: cn.itcast.mybatis.po.User

二级缓存可以将内存的数据写到磁盘，存在对象的序列化和反序列化，所以要实现java.io.serializable接口。

如果结果映射的pojo中还包括了pojo，都要实现java.io.serializable接口。



### 二级缓存禁用

对于变化频率较高的sql，需要禁用二级缓存：

在statement中设置useCache=false可以禁用当前select语句的二级缓存，即每次查询都会发出sql去查询，默认情况是true，即该sql使用二级缓存。

<select id="findOrderListResultMap" resultMap="ordersUserMap" useCache="false">

### 刷新缓存

如果sqlsession操作commit操作，对二级缓存进行刷新（全局清空）。

设置statement的flushCache是否刷新缓存，默认值是true。

### 测试代码



### mybatis的cache参数（了解）

mybatis的cache参数只适用于mybatis维护缓存。

flushInterval（刷新间隔）可以被设置为任意的正整数，而且它们代表一个合理的毫秒形式的时间段。默认情况是不设置，也就是没有刷新间隔，缓存仅仅调用语句时刷新。

size（引用数目）可以被设置为任意正整数，要记住你缓存的对象数目和你运行环境的可用内存资源数目。默认值是1024。

readOnly（只读）属性可以被设置为true或false。只读的缓存会给所有调用者返回缓存对象的相同实例。因此这些对象不能被修改。这提供了很重要的性能优势。可读写的缓存会返回缓存对象的拷贝（通过序列化）。这会慢一些，但是安全，因此默认是false。

如下例子：

<cache  eviction="FIFO"  flushInterval="60000"  size="512"  readOnly="true"/>

这个更高级的配置创建了一个 FIFO 缓存,并每隔 60 秒刷新,存数结果对象或列表的 512 个引用,而且返回的对象被认为是只读的,因此在不同线程中的调用者之间修改它们会导致冲突。可用的收回策略有, 默认的是 LRU:

1. LRU – 最近最少使用的:移除最长时间不被使用的对象。
2. FIFO – 先进先出:按对象进入缓存的顺序来移除它们。
3. SOFT – 软引用:移除基于垃圾回收器状态和软引用规则的对象。
4. WEAK – 弱引用:更积极地移除基于垃圾收集器状态和弱引用规则的对象。

## mybatis和ehcache缓存框架整合

mybatis二级缓存通过ehcache维护缓存数据。

### 分布缓存

将缓存数据数据进行分布式管理。

系统1

springmvc

mybatis

系统2

springmvc

mybatis

用户请求

缓存框架（远程服务器集群）

redis、memcached、ehcache

存取缓存数据

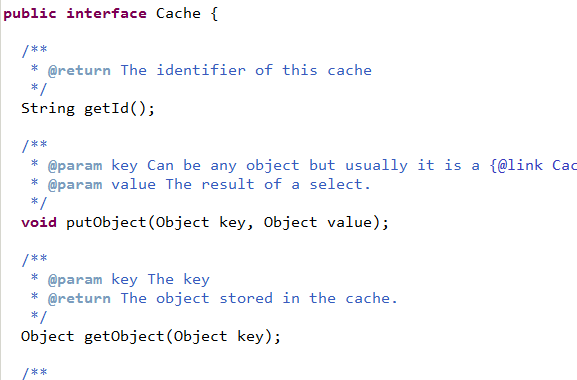
存取缓存数据

### mybatis和ehcache思路

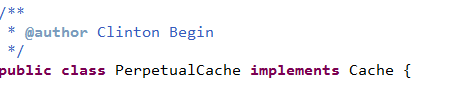
因为mybatis无法完成分布式缓存，所以有必要让mybatis和其他缓存框架进行整合，实现支持分布式缓存管理

通过mybatis和ehcache框架进行整合，就可以把缓存数据的管理托管给ehcache。

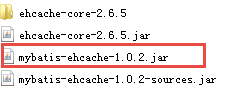
在mybatis中提供一个cache接口，只要实现cache接口就可以把缓存数据灵活的管理起来。



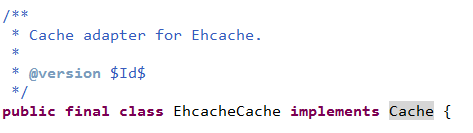
mybatis中默认实现：



### 下载和ehcache整合的jar包



ehcache对cache接口的实现类：

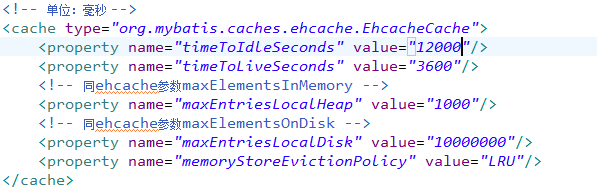


### 配置ehcache.xml



### 整合测试

在mapper.xml添加ehcache配置：



## 二级缓存的应用场景

对查询频率高，变化频率低的数据建议使用二级缓存。

对于访问多的查询请求且用户对查询结果实时性要求不高，此时可采用mybatis二级缓存技术降低数据库访问量，提高访问速度，业务场景比如：耗时较高的统计分析sql、电话账单查询sql等。

实现方法如下：通过设置刷新间隔时间，由mybatis每隔一段时间自动清空缓存，根据数据变化频率设置缓存刷新间隔flushInterval，比如设置为30分钟、60分钟、24小时等，根据需求而定。

## mybatis局限性

mybatis二级缓存对细粒度的数据级别的缓存实现不好，比如如下需求：对商品信息进行缓存，由于商品信息查询访问量大，但是要求用户每次都能查询最新的商品信息，此时如果使用mybatis的二级缓存就无法实现当一个商品变化时只刷新该商品的缓存信息而不刷新其它商品的信息，因为mybaits的二级缓存区域以mapper为单位划分，当一个商品信息变化会将所有商品信息的缓存数据全部清空。解决此类问题需要在业务层根据需求对数据有针对性缓存。

# mybatis和spring整合

## mybaits和spring整合的思路

1、让spring管理SqlSessionFactory（管理的目的是为了生成会话，生成会话的目的是为了产生mapper的动态代理对象）

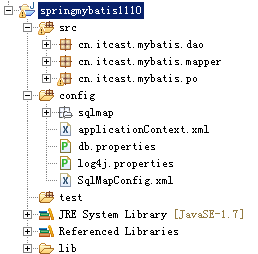
2、让spring管理mapper对象和dao（原始开发方式，需要管理的是dao）。

使用spring和mybatis整合开发mapper代理及原始dao接口。

自动开启事务，自动关闭 sqlsession.

3、让spring管理数据源( 数据库连接池)

## 创建整合工程



## 加入jar包

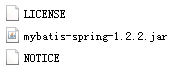
1、mybatis3.2.7本身的jar包

2、数据库驱动包

3、spring3.2.0

4、spring和mybatis整合包

从mybatis的官方下载spring和mybatis整合包



## log4j.properties

## SqlMapconfig.xml

mybatis配置文件：别名、settings，数据源不在这里配置

## applicationContext.xml

1、数据源（dbcp连接池）

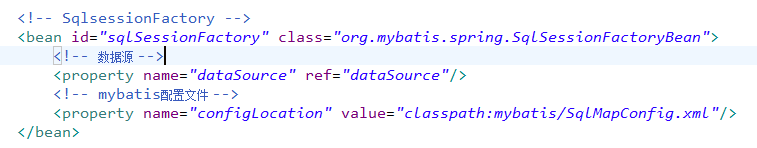
2、SqlSessionFactory

3、mapper或dao

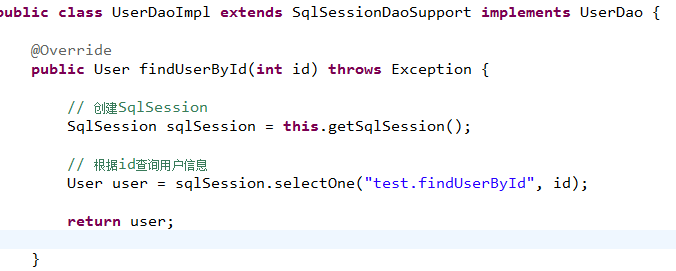
## 整合开发原始dao接口

### 配置SqlSessionFactory

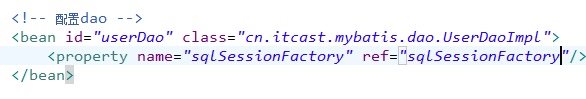
在 applicationContext.xml配置SqlSessionFactory



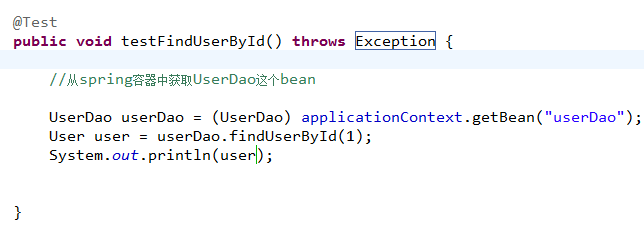
### 开发dao



### 配置 dao



### 测试dao接口

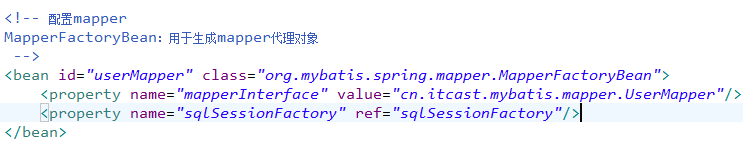


## 整合开发mapper代理方法

### 开发mapper.xml和mapper.java



### 使用MapperFactoryBean



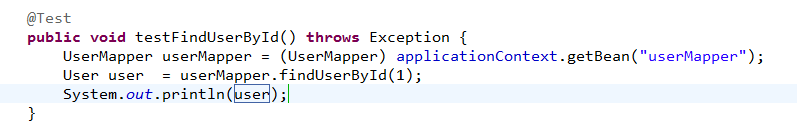
使用此方法对于每个mapper都需要配置，比较繁琐。

### 使用MapperScannerConfigurer（扫描mapper）



使用扫描器自动扫描mapper，生成代理对象，比较方便。

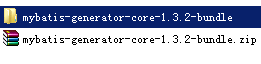
### 测试mapper接口



# mybatis逆向工程 （MyBatis Generator）

## 什么是mybatis的逆向工程

mybatis官方为了提高开发效率，提高自动对**单表**生成sql，包括 ：mapper.xml、mapper.java、表名.java(po类)



在企业开发中通常是在设计阶段对表进行设计 、创建。

在开发阶段根据表结构创建对应的po类。

**mybatis逆向工程的方向**：**由数据库表--->java代码**

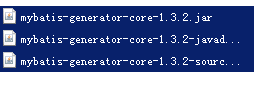
## 逆向工程 使用配置

运行逆向工程 方法：



本教程使用通过java程序运行逆向工程 。

逆向工程 运行所需要的jar包：



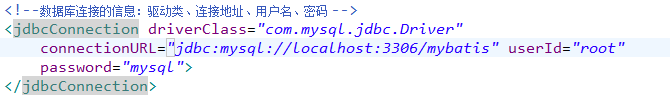
数据库驱动包

日志包

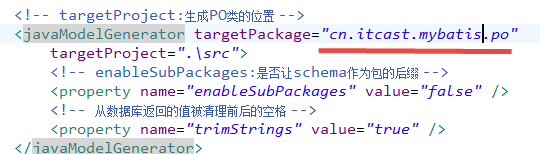
### xml配置

需要使用用配置的地方：

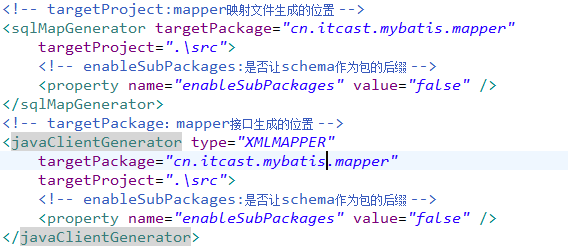
1、连接数据库的地址和驱动



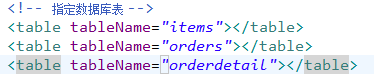
2、需要配置po类的包路径



3、需要配置mapper包的路径

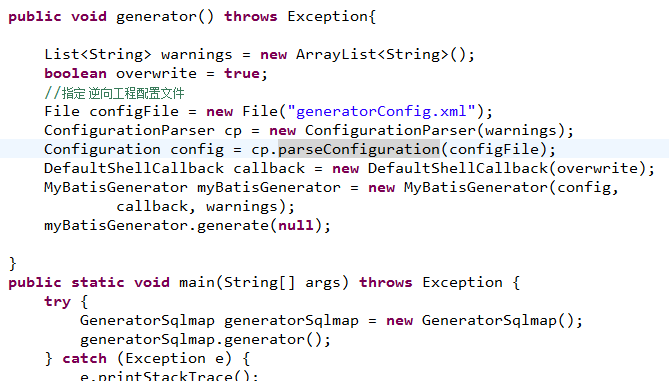


4、指定数据库表



### java程序

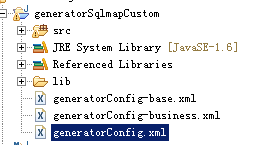
通过java程序生成mapper类、po类。。



## 使用逆向工程生成的代码

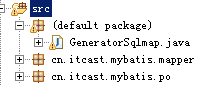
### 第一步：配置generatorConfig.xml

参考：

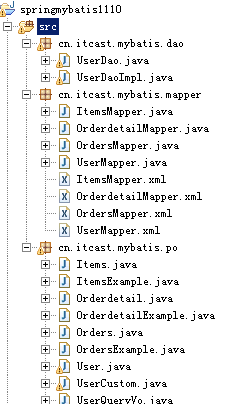


### 第二步：配置执行java程序

执行java程序后，所生成的代码已经生成到工程中。



### 第三步：将生成的代码拷贝到工程中



## 测试生成的代码

参考：

