# 1关联查询

## 案例：用户和定单

|  |
| --- |
|  |

|  |
| --- |
| * **user和orders：**   User 与orders：一个用户可以创建多个订单，一对多  Orders 与 user：多个订单只由一个用户创建，多对一   * **orders和orderdetail：**   Orders 与 orderdetail：一个订单可以包括 多个订单明细，因为一个订单可以购买多个商品，每个商品的购买信息在orderdetail记录，一对多关系  orderdetail 与orders：多个订单明细包括在一个订单中, 多对一   * **orderdetail和items：**   Orderdetail 与 items：多个订单明细只对应一个商品信息，多对一  Items 与 orderdetail:一个商品可以包括在多个订单明细 ，一对多 |

## 需求：

**根据商品ID查找定单信息，包括用户名和地址**

|  |
| --- |
| #查找id为10的所有定单  SELECT  orders.id, orders.number,orders.createtime,orders.note,`user`.username,`user`.address  FROM  orders ,`user`  WHERE  orders.user\_id = `user`.id AND `user`.id = 10; |

## 1.2 一对一 resultType实现

复杂查询时，单表对应的po类已不能满足输出结果集的映射。所以要根据需求建立一个**扩展类来作为resultType的类型**。

|  |
| --- |
| #查找某个定单id的信息，包括用户名字和地址  SELECT o.\*,u.username,u.address FROM orders o,user u  WHERE o.user\_id = u.id AND o.id = 3 |

### 第一步：写个定单的扩展类

|  |
| --- |
|  |

### 第二步：声明定单接口

|  |
| --- |
|  |

### 第三步：声明定单配置文件

|  |
| --- |
|  |

### 第四步：加载映射文件

|  |
| --- |
|  |

### 第五步：测试

|  |
| --- |
|  |

### 一对一 resultMap实现

**掌握association的使用**

|  |
| --- |
| OrdersMapper.java添加一个方法 |
|  |
| OrdersMapper.xml |
|  |
| 测试 |
|  |

### 总结

**resultType**：使用resultType实现较为简单，如果pojo中没有包括查询出来的列名，需要增加列名对应的属性，即可完成映射。**如果没有查询结果的特殊要求建议使用resultType。**

**resultMap**：需要单独定义resultMap，实现有点麻烦，如果对查询结果有特殊的要求，使用resultMap可以完成将关联查询映射pojo的对象属性中。

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

## 1.3 一对多

### 需求：

根据定单ID查找定单信息、用户信息和定单明细信息

|  |
| --- |
| Select  orders.id,  orders.user\_id,  orders.number,  orders.createtime,  orders.note,  user.username,  user.address,  orderdetail.id detail\_id,  orderdetail.items\_id,  orderdetail.items\_num  from  orders,user,orderdetail  where  orders.user\_id = user.id  and orders.id = orderdetail.orders\_id  and orders.id = #{?}; |
| SELECT  o.\*,  u.username,  u.address,  od.id detail\_id,  od.items\_id,  od.items\_num  FROM  orders o,  user u,  orderdetail od  WHERE  o.user\_id = u.id  AND o.id = od.orders\_id  AND o.id = 3 |

目标：掌握collection的使用

### 第一步：在Orders中添加定单明细

|  |
| --- |
|  |

### 第二步：Mapper接口

|  |
| --- |
|  |

### 第三步：OrderMapper.xml

resultMap中有个extends属性，可以继承【自行研究】

|  |
| --- |
|  |

### 第四步：测试

|  |
| --- |
|  |

### 总结

mybatis使用resultMap的collection对关联查询的多条记录映射到一个list集合属性中。

**使用resultType实现：**

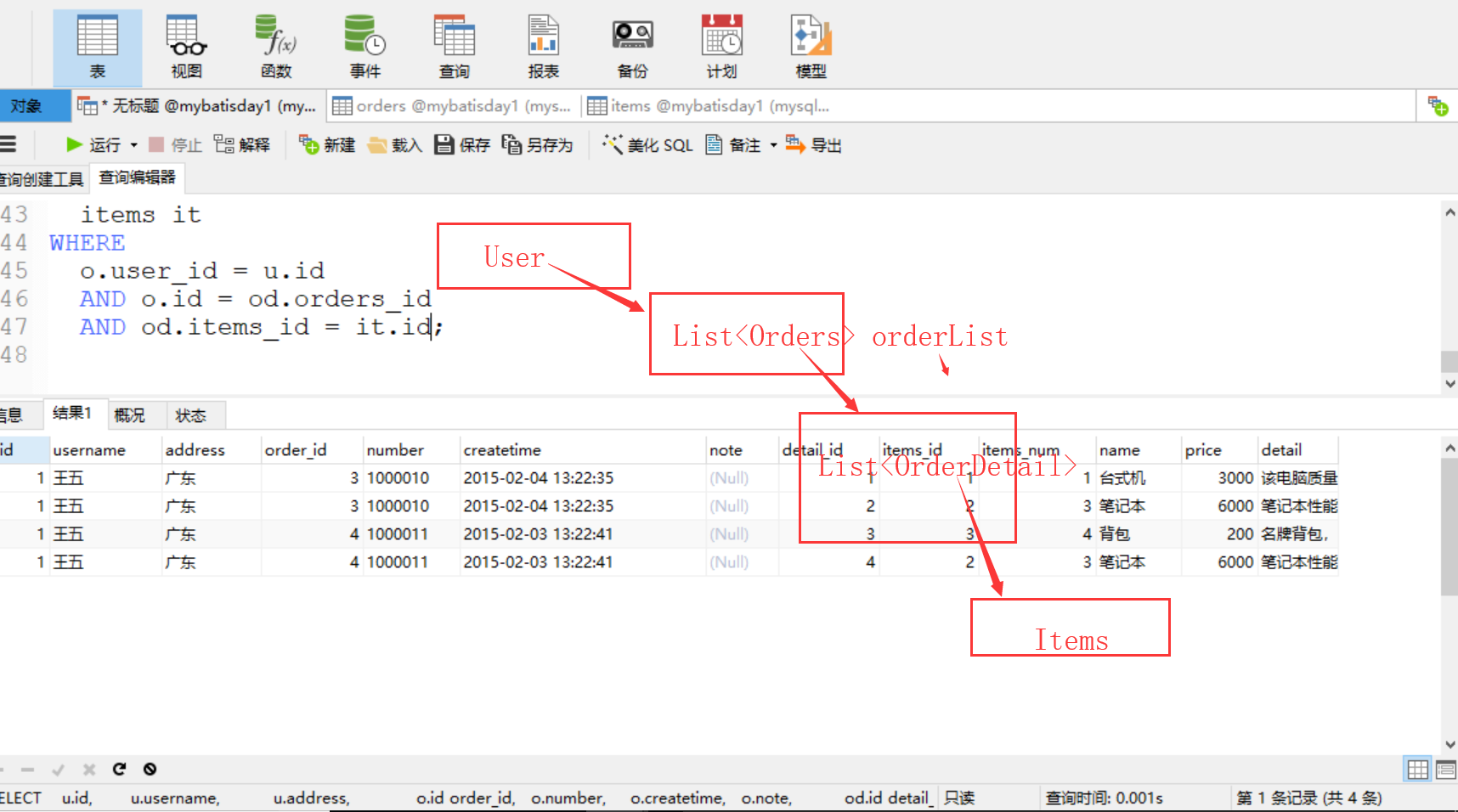
需要对结果集进行二次处理。

将订单明细映射到orders中的orderdetails中，需要自己处理，使用双重循环遍历，去掉重复记录，将订单明细放在orderdetails中。

## 1.4 多对多

### 需求

查询用户信息及用户购买的商品信息，**要求将关联信息映射到主pojo的pojo属性中**



### Sql

|  |
| --- |
| Select  user.id,  user.username,  user.address,  orders.id orders\_id,  orders.user\_id,  orders.number,  orders.createtime,  orders.note,  orderdetail.id detail\_id,  orderdetail.items\_id,  orderdetail.items\_num,  items.name items\_name,  items.detail items\_detail  FROM  USER,orders,orderdetail,items  WHERE user.`id` = orders.`user\_id`  AND orders.`id` = orderdetail.`orders\_id`  AND orderdetail.`items\_id` = items.`id` |
| SELECT  u.id,  u.username,  u.address,  o.id order\_id,  o.number,  o.createtime,  o.note,  od.id detail\_id,  od.items\_id,  od.items\_num,  it.name,  it.price,  it.detail  FROM  user u,  orders o,  orderdetail od,  items it  WHERE  o.user\_id = u.id  AND o.id = od.orders\_id  AND od.items\_id = it.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，将用户创建的订单映射到orderslist
* 在Orders中添加订单明细列表属性List<Orderdetail> detailList，将订单的明细映射到detailList
* 在Orderdetail中添加Items属性，将订单明细所对应的商品映射到Items

### 第一步：UserMapper.java

|  |
| --- |
|  |

### 第二步：User/Orders/Orderdetail.java

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

### 第三步：UserMapper.xml

|  |
| --- |
|  |

### 第四步：测试

|  |
| --- |
|  |
| 打印效果 |

### 总结

resultType：将查询结果按照sql列名pojo属性名一致性映射到pojo中。

resultMap：使用association和collection完成一对一和一对多高级映射（对结果有特殊的映射要求）。

association：将关联查询信息映射到一个pojo对象中。

collection：将关联查询信息映射到一个list集合中。

# 延时加载

## 2.1 延迟加载

延迟加载又叫懒加载，也叫按需加载。也就是说先加载主信息，在需要的时候，再去加载从信息。

在mybatis中，resultMap标签 的association标签和collection标签具有延迟加载的功能。

## 2.1 案例：

### Mapper.java

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

### Mapper.xml

|  |
| --- |
| UserMappler.xml |
|  |
| OrdersMapper.xml |
|  |

### 测试

|  |
| --- |
|  |

### 配置懒加载

|  |
| --- |
|  |

# 3查询缓存

## 3.1 Mybatis的缓存理解

Mybatis的缓存，包括一级缓存和二级缓存，一级缓存是默认使用的。二级缓存需要手动开启。

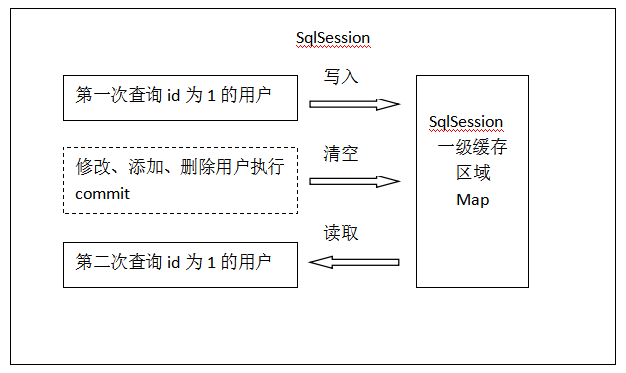
一级缓存指的就是sqlsession，在sqlsession中有一个数据区域，是map结构，这个区域就是一级缓存区域。一级缓存中的key是由sql语句、条件、statement等信息组成一个唯一值。一级缓存中的value，就是查询出的结果对象。

二级缓存指的就是同一个namespace下的mapper，二级缓存中，也有一个map结构，这个区域就是一级缓存区域。一级缓存中的key是由sql语句、条件、statement等信息组成一个唯一值。一级缓存中的value，就是查询出的结果对象。

|  |
| --- |
|  |

## 3.2 一级缓存

### 原理：



### 测试1

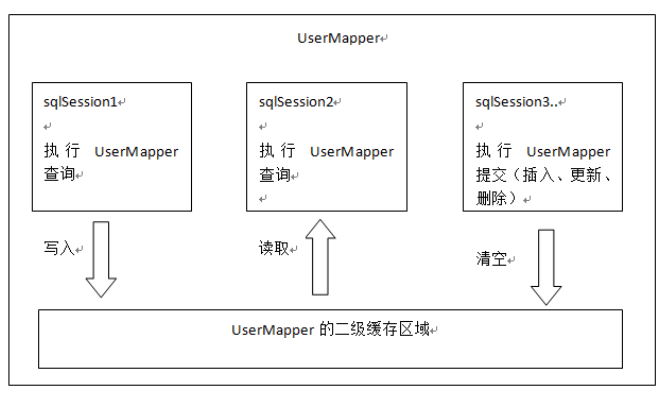
|  |
| --- |
|  |

### 测试2

|  |
| --- |
|  |

## 3.3 二级缓存

### 原理：



### 使用：

#### 开启二级缓存总开关

|  |
| --- |
|  |

#### UserMapper中配置二级缓存

|  |
| --- |
|  |

#### User系列化

|  |
| --- |
|  |

#### 测试

|  |
| --- |
|  |

#### 禁用指定方法二级缓存

|  |
| --- |
|  |

#### 刷新缓存

|  |
| --- |
|  |

## 3.4 整合ehcache

Mybatis本身是一个持久层框架，它不是专门的缓存框架，所以它对缓存的实现不够好，不能支持分布式。

Ehcache是一个分布式的缓存框架。

#### 什么是分布式

系统为了提高性能，通常会对系统采用分布式部署（集群部署方式）

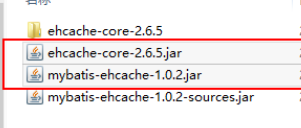
|  |
| --- |
|  |

#### 整合思路

Cache是一个接口，它的默认实现是mybatis的PerpetualCache。如果想整合mybatis的二级缓存，那么实现Cache接口即可。

|  |
| --- |
|  |

#### 添加jar包



#### 设置映射文件中cache标签

的type值为ehcache的实现类

|  |
| --- |
|  |

#### 在src下添加ehcache的配置文件

|  |
| --- |
|  |
| * maxElementsInMemory :设置基于内存的缓存中可存放的对象最大数目 * eternal:设置对象是否为永久的,true表示永不过期,此时将忽略 * timeToIdleSeconds 和 timeToLiveSeconds属性; 默认值是false * timeToIdleSeconds:设置对象空闲最长时间,以秒为单位, 超过这个时间,对象过期。当对象过期时,EHCache会把它从缓存中清除。如果此值为0,表示对象可以无限期地处于空闲状态。 * timeToLiveSeconds:设置对象生存最长时间,超过这个时间,对象过期。如果此值为0,表示对象可以无限期地存在于缓存中. 该属性值必须大于或等于 timeToIdleSeconds 属性值 * overflowToDisk:设置基于内在的缓存中的对象数目达到上限后,是否把溢出的对象写到基于硬盘的缓存中 * diskPersistent 当jvm结束时是否持久化对象 true false 默认是false * diskExpiryThreadIntervalSeconds 指定专门用于清除过期对象的监听线程的轮询时间   memoryStoreEvictionPolicy - 当内存缓存达到最大，有新的element加入的时候， 移除缓存中element的策略。**默认是LRU（最近最少使用），可选的有LFU（最不常使用）和FIFO（先进先出）** |

#### 测试：用上面二级缓存例子即可

#### 二级缓存应用场景

使用场景：**对于访问响应速度要求高，但是实时性不高的查询，可以采用二级缓存技术**。

注意：在使用二级缓存的时候，要设置一下刷新间隔（cache标签中有一个flashInterval属性）来定时刷新二级缓存，这个刷新间隔根据具体需求来设置，比如设置30分钟、60分钟等，单位为毫秒。

#### 局限性

**Mybatis二级缓存对细粒度的数据，缓存实现不好。**

场景：

对商品信息进行缓存，由于商品信息查询访问量大，但是要求用户每次查询都是最新的商品信息，此时如果使用二级缓存，就无法实现当一个商品发生变化只刷新该商品的缓存信息而不刷新其他商品缓存信息，因为二级缓存是mapper级别的，当一个商品的信息发送更新，所有的商品信息缓存数据都会清空。

解决此类问题，需要在业务层根据需要对数据有针对性的缓存。

比如可以对经常变化的 数据操作单独放到另一个namespace的mapper中。

# mybaties整合spring

SSH:struts2+spring+hibernate

SSM:SpringMVC + Spring + MyBatis

Spring 3.2

Mybaties 3.2.7

## 4.1 创建工程导包

### 导入mybaties包

|  |
| --- |
| mybaties核心包 |
| mybatis依赖包 |

### 导入mysql数据库驱动

|  |
| --- |
|  |

### 数据库dbcp连接池

|  |
| --- |
|  |

### 导入spring+mvc包

|  |
| --- |
|  |

### Mybatis-spring整合包

|  |
| --- |
|  |

## 4.2 配置mybatis的核心配置文件

核心配置文件、创建User模型、映射文件，

|  |
| --- |
|  |
|  |

## 4.3 spring的数据源

|  |
| --- |
| <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-3.2.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-3.2.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-3.2.xsd "*>      <!-- 1.配置数据库，dbcp数据库连接池 -->  <bean id=*"datasourse"* class=*"org.apache.commons.dbcp.BasicDataSource"*>  <property name=*"driverClassName"* value=*"xxx"*/>  <property name=*"url"* value=*"xxx"*/>  <property name=*"username"* value=*"xxx*/>  <property name=*"password"* value=*"xxx"*/>  <!-- 最大连接 -->  <property name=*"maxActive"* value=*"10"*/>  <!--最大空闲数 -->  <property name=*"maxIdle"* value=*"5"*/>  </bean>  </beans> |

## 4.4 spring配置SqlSessionFactory

|  |
| --- |
| <!-- 3.配置SqlSessionFactory -->  <bean id=*"sqlSessionFactoryBean"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <property name=*"configLocation"* value=*"SqlMapConfig.xml"*></property>  <property name=*"dataSource"* ref=*"dataSourse"*></property>  </bean> |

## 4.5 编写个UserDaoImpl,接口省略

|  |
| --- |
|  |

## 4.6 spring中配置daobean

|  |
| --- |
|  |

## 4.7测试

|  |
| --- |
|  |

## 4.8 换成Mapper接口整合dao

### 创建Mapper映射文件

|  |
| --- |
| UserMapper |
|  |
| UserMapper.xml |
|  |

### 核心配置文件加载映射文件

|  |
| --- |
|  |

### Spring配置MapperFactoryBean

使用工厂Bean生成userMapper对象

|  |
| --- |
| <!-- 4.配置 -->  <bean id = *"userMapper"* class=*"org.mybatis.spring.mapper.MapperFactoryBean"*>  <property name=*"mapperInterface"* value=*"com.gyf.backoffice.mapper.UserMapper"*/>  <property name=*"sqlSessionFactory"* ref=*"sqlSessionFactoryBean"*/>  </bean> |

### 测试

|  |
| --- |
|  |

### 用MapperScannerConfigurer批量扫描创建代理对象

【上面的代码麻烦，每一个mappler就创建一个工厂bean】

|  |
| --- |
| <!-- mapper代理开发方式之批量mapper配置 ,bean的名字默认为mapper接口类名的首字母小写  注意：  1.jdk1.8 用这种方式，bean不能创建成功 ，改成jdk1.7的即可  2.或者spring我换成spring3.2.9或以上就OK了  -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"com.gyf.backoffice.mapper"*/>  <property name=*"sqlSessionFactoryBeanName"* value=*"sqlSessionFactoryBean"*/>  </bean> |

测试与上一个测试一样

# 逆向工程

## 简介

**简单点说，就是通过数据库中的单表，自动生成java代码。**

Mybatis官方提供了逆向工程

可以针对单表自动生成mybatis代码（mapper.java\mapper.xml\po类）

企业开发中，逆向工程是个很常用的工具。

## 下载逆向工程

<https://github.com/mybatis/generator/releases/tag/mybatis-generator-1.3.2>

## 使用方法

1. 创建简单的java项目
2. 导入jar包,创建generator配置文件；
3. 使用java类来执行逆向工程；
4. 把生成的代码拷贝到项目中。
5. 在正式项目中使用逆向工程生成的代码

### 第一步：创建generator配置文件

在classpath下，创建generator.xml配置文件：（文件内容可以从逆向工程的jar包中docs目录下的index.html中找到相关代码）

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE generatorConfiguration  PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">  <generatorConfiguration>    <context id=*"mysqlTable"* targetRuntime=*"MyBatis3"*>  <!-- 1.数据连接参数 -->  <jdbcConnection driverClass=*"com.mysql.jdbc.Driver"*  connectionURL=*"jdbc:mysql://localhost:3306/mybatis"*  userId=*"root"*  password=*"123456"*>  </jdbcConnection>    <!-- 2.默认false，把JDBC DECIMAL 和 NUMERIC 类型解析为 Integer，为 true时把JDBC DECIMAL  和 NUMERIC 类型解析为java.math.BigDecimal -->  <javaTypeResolver >  <property name=*"forceBigDecimals"* value=*"false"* />  </javaTypeResolver>  <!-- 3.生成模型的位置 -->  <javaModelGenerator targetPackage=*"com.gyf.backoffice.domain"* targetProject=*".\src"*>  <!-- enableSubPackages:是否让schema作为包的后缀 -->  <property name=*"enableSubPackages"* value=*"true"* />  <property name=*"trimStrings"* value=*"true"* />  </javaModelGenerator>  <!-- 4.targetProject:mapper映射文件生成的位置 -->  <sqlMapGenerator targetPackage=*"com.gyf.backoffice.mapper"* targetProject=*".\src"*>  <!-- enableSubPackages:是否让schema作为包的后缀 -->  <property name=*"enableSubPackages"* value=*"true"* />  </sqlMapGenerator>  <!-- 5. targetPackage：mapper接口生成的位置 -->  <javaClientGenerator type=*"XMLMAPPER"* targetPackage=*"com.gyf.backoffice.mapper"*  targetProject=*".\src"*>  <property name=*"enableSubPackages"* value=*"true"* />  </javaClientGenerator>  <!-- 6.要生成的表 -->  <table tableName=*"items"*/>  <table tableName=*"orderdetail"*/>  <table tableName=*"orders"*/>  <table tableName=*"user"*/>  </context>  </generatorConfiguration> |

### 第二步：使用java类来执行逆向工程

需要导入mysql的驱动包和mybatis的逆向工程包

|  |
| --- |
| public class Generator {  public static void main(String[] args) throws Exception{  List<String> warnings = new ArrayList<String>();  boolean overwrite = true;  File configFile = new File("config/generator.xml");  ConfigurationParser cp = new ConfigurationParser(warnings);  Configuration config = cp.parseConfiguration(configFile);  DefaultShellCallback callback = new DefaultShellCallback(overwrite);  MyBatisGenerator myBatisGenerator = new MyBatisGenerator(config,  callback, warnings);  myBatisGenerator.generate(null);  }  } |

### 第三步：把生成的代码拷贝到项目中

如果正式项目中已经有po类所在的包了，那么就只需要拷贝po类到指定包下就可以。

如果正式项目中没有po包，那么就把逆向工程中整个po类的包拷贝过去。

Mapper.xml和mapper.java的拷贝与po类一样。

|  |
| --- |
|  |

### 第四步：测试

逆向工程提供了很多查询方法，可以不用写sql,这个根hibernate有点类似

|  |
| --- |
|  |