Mybatis

1. mybatis\_day01
   1. 对原生态jdbc程序中问题总结
      1. 环境

java环境：jdk 1.8

mysql：5.7

eclipse：luna

* + 1. 创建mysql数据库
    2. jdbc程序
       1. 使用jdbc查询mysql数据库中用户表的记录
       2. 创建java工程，导入jar包
    3. 问题总结
       1. 数据库连接，使用时就创建，不使用就立即释放，对数据库进行频繁连接开启和关闭，造成数据库资源的浪费，影响数据库的性能
          1. 解决方案：使用数据库的连接池来管理数据库的连接
       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
          1. 解决：使用配置文件，配置sql语句、
       3. 向preparedStatement中设置参数，对占位符位置和参数值，硬编码在java代码中
          1. 解决：将sql语句，占位符，和参数值，都配置在sql语句中
       4. 从resultSet中遍历结果集数据，存在硬编码，将获取表字段硬编码，不利于系统维护
          1. 能不能将查询结果集，自动映射成java对象
  1. mybatis的框架
     1. mybatis是什么？
        1. mybatis是一个持久层框架，是apache下的顶级项目
        2. 托管到googlecode下，再后来托管到gitHub下（https://github.com/mybatis/mybatis-3/releases）
        3. 让我们程序猿将主要精力放在sql上，通过mybatis提供的映射方式，自由灵活生成（半自动化，大部分需要程序猿编写sql）满足需要的sql语句
        4. mybatis可以将我们向preparedStatement输入的参数自动映射，将查询结果集灵活的映射出java对象
     2. mybatis框架

sqlMapConfig.xml（mybatis的全局配置文件，名称不固定）配置了数据源，事务等运行环境，配置映射文件，mapper.xml

sqlSeesionFactory（会话工厂），根据配置文件创建工厂作用：创建sqlSession，

mysql

sqlsession（会话）

作用：操作数据库（sql的crud操作）

Executor（执行器）也是一个接口（基本执行器，缓存执行器）作用：sqlsession是通过执行器来操作数据库的

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、输入结果类型

输入参数类型

java简单类

hashmap

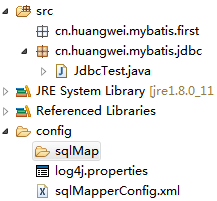
pojo自定义类型

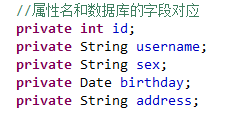
输出结果类型

java简单类

hashmap

pojo自定义类型

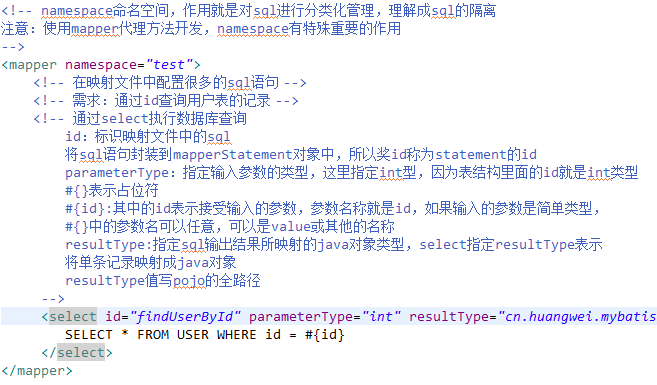
* 1. 入门程序
     1. 需求
        1. 根据用户id查询用户信息
        2. 根据用户名称模糊查询用户信息
        3. 添加用户
        4. 删除用户
        5. 更新用户
     2. mybatis的运行环境
        1. [https://github.com/mybatis/mybatis-3/releases 版本3.2.7](https://github.com/mybatis/mybatis-3/releases%20%20版本3.2.7)
        2. lib下：依赖包
        3. mybatis-3.2.7jar：核心包
        4. mysql驱动包
     3. log4j.properties文件
     4. 
     5. sqlMapConfig.xml
        1. 配置mybatis的运行环境，数据源，事务等
        2. 
     6. 根据用户id（主键）查询用户信息
        1. 创建pojo类



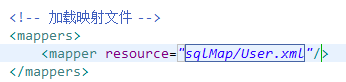
* + - 1. 创建映射文件
         1. 映射文件的命名

User.xml, mapper代理开发映射文件名称叫xxxMapper.xml,比如UserMapper.xml, ItemsMapper.xml

* + - * 1. 在映射文件中配置sql语句



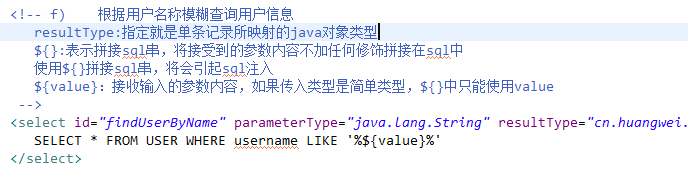
* + - * 1. 在sqlMapConfig中加载映射文件

加载user.xml

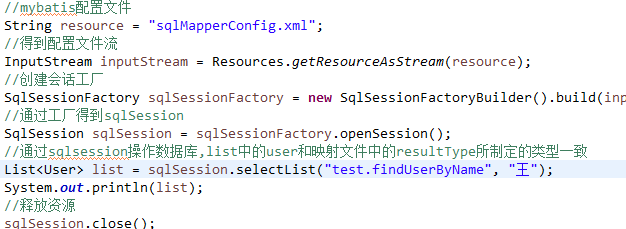
* + - * 1. 程序编写

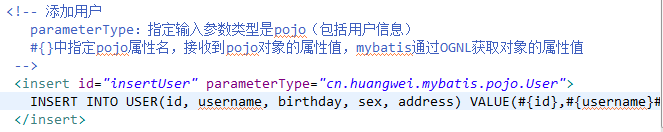
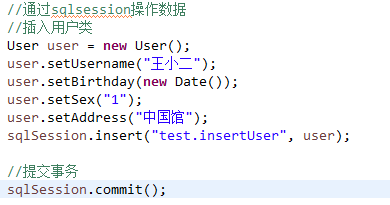


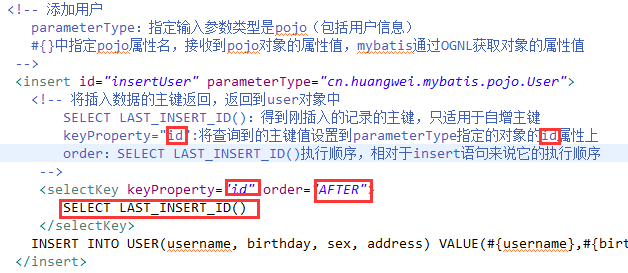
* + 1. 根据用户名称模糊查询用户信息
       1. 映射文件
          1. 使用User.xml，添加根据用户名称模糊查询用户信息的sql语句



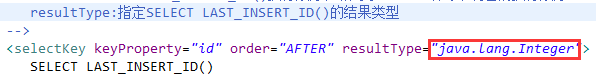
* + - * 1. java代码



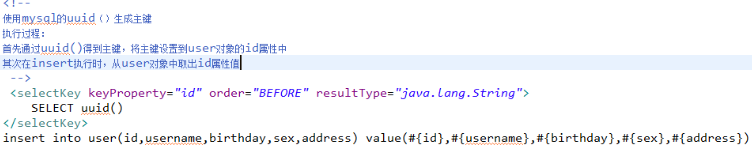
* + 1. 添加用户
       1. 映射文件
          1. User.xml中配置添加用户的sql（statement）
          2. 
       2. 程序代码
          1. 
       3. 自增主键的返回
          1. mysql的自增主键，执行insert提交之前，自动生成一个自增主键，通过mysql函数能够获取到刚插入的自增主键：函数LAST\_INSERT\_ID，实在insert之后调用此函数

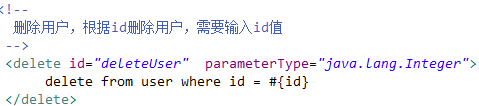
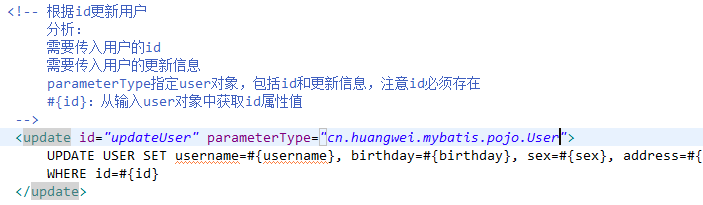
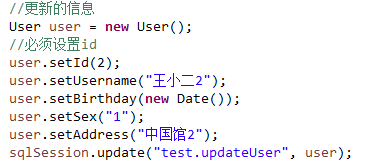
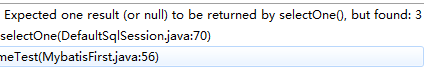
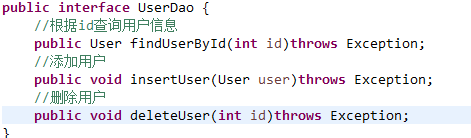


异常：

解决：

* + - * 1. 非自增主键返回（使用uuid）



* + 1. 删除用户
       1. 映射文件
          1. 
       2. 程序代码
          1. 
    2. 更新用户
       1. 映射文件
          1. 
       2. 程序代码
          1. 
    3. 小结
       1. parameterType
          1. 在映射文件中通过parameterType指定输入参数的类型
       2. resultType
          1. 在映射文件中通过resultType指定输出结果的类型
       3. #{}和${}
          1. #{}表示一个占位符，#{}接受输入参数，类型可以简单类型，pojo，hashMap，如果接受简单类型，#{}可以写成value或者其他名称，#{}接受pojo对象值，通过OGNL读取对象的属性值，通过属性.属性.属性..的方式获取对象属性值
          2. ${}表示一个拼接符号，会引起sql拼接注入，不建议使用，${}接受输入参数，类型可以是简单类型，pojo，hashMap，如果接受类型是简单类型，${}中只能写成value
       4. selectOne和selectList
          1. 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）
          2. selectList表示查询出一个列表（多条记录）进行映射，如果使用selectList查询多条数据，不能使用selectOne、
          3. 报错：
    4. mybatis和hibernate本质区别和应用场景
       1. hibernate：入门门槛比较高，是一个标准的orm框架，不需要程序猿来写sql，想要对sql语句进行优化，修改比较困难
          1. 应用场景：适用于需求变化不多的中小型项目，比如后台管理系统，erp，orm，oa。。。
       2. mybatis：专注sql本身，需要程序猿自己编写sql语句，sql修改，优化比较方便，mybatis是一个不完全的orm框架，虽然程序猿自己写sql，但是它也能实现映射（输入映射，输出映射）
          1. 应用场景：适用于需求变化较多的项目，比如：互联网项目，
       3. 企业在进行技术选型时，以低成本，高回报，作为技术选型的原则，根据项目组技术力量进行选择
  1. mybatis开发dao的方法
     1. sqlSession使用范围
        1. sqlsessionFactoryBuilder
           1. 通过sqlsessionFactoryBuilder创建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的时候，只需要new一个sqlSessionFactorybuilder即可
        2. sqlsessionFactory
           1. 通过sqlsessionFactory创建sqlsession，通过单例模式来管理sqlsessionFactory（工厂一旦创建就一直使用一个实例），将来和spring整合之后，使用单例模式管理sqlSessionFactory
        3. sqlsession
           1. 是一个面向用户（程序猿）的接口
           2. 提供了好多操作数据库的方法，如selectOne（返回单个对象）、selectList（返回list集合）
           3. sqlsession是线程不安全的，在sqlsession实现类中，除了有接口的方法（操作数据库的方法），还有数据域属性
           4. **sqlsession最佳应用场合在方法体内，定义成局部变量**
     2. 原始dao开发方法(程序猿需要写dao接口和dao实现类)
        1. 思路
           1. 程序猿需要写dao接口和dao实现类
           2. 需要向dao实现类中注入sqlsessionFactory，在方法体内通过sqlsessionFactory创建sqlsession
        2. dao接口
           1. 
        3. dao接口实现类

|  |
| --- |
| //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 {  // TODO Auto-generated method stub  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 {  // TODO Auto-generated method stub  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 {  // TODO Auto-generated method stub  SqlSession sqlSession = sqlSessionFactory.openSession();  //执行删除操作  sqlSession.delete("test.deleteUser", 1);  //提交事务  sqlSession.commit();  //释放资源  sqlSession.close();  }  **测试代码** |

* + - 1. 总结原始到的开发问题
         1. dao接口实现类方法中大量存在模板方法，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，不利于程序猿的开发
    1. mapper代理方法(程序员只需要写mapper接口（相当于dao接口）)
       1. 思路
          1. 程序猿编写mapper接口（相当于dao接口）要遵循开发规范，mybatis可以自动生成mapper接口实现类的代理对象
          2. 程序猿还需要编写mapper.xml映射文件
          3. 开发规范

在mapper.xml中namespace等于，mapper接口地址

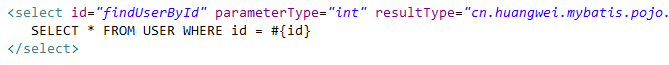


xxmapper.java接口中的方法名和mapper.xml中statement的id一致

xxmapper.java接口中方法的输入参数类型和mapper.xml中statement的parameterType指定的类型一致

xxmapper.java接口中方法的返回值类型和mapper.xml中statement的resultType指定的类型一致

方法定义





总结

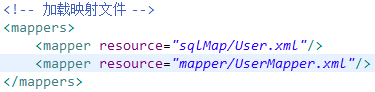
以上开发规范主要是对下边的代码进行统一生成

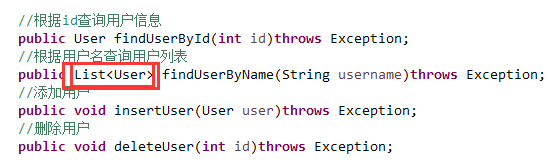




等

* + - 1. xxmapper.java
         1. 
      2. mapper.xml
         1. 
      3. 测试
         1. 
         2. 错误：

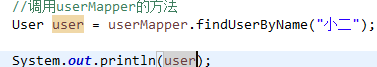


* + - 1. 代理对象内部调用selectList或者selectOne
         1. 

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



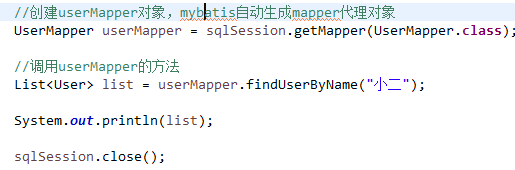


出现错误



正确做法





* + - 1. 一些问题的总结
         1. 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，是否不利于扩展维护

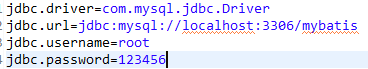
系统框架中，dao层代码是被业务层共用的，即使mapper接口只有一个参数，可以使用包装类型的pojo满足不同的业务方法的需求

注意：持久层中方法参数可以使用包装类型，map，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)

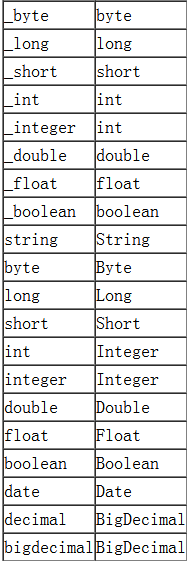
* 1. SqlMapConfig.xml
     1. mybatis的全局配置文件：SqlMapConfig.xml，配置内容如下
        1. properties（属性）
        2. setting（全局配置参数）
        3. typeAliases（类型别名）
        4. typeHandlers（类型处理器）
        5. objectFactory（对象工厂）
        6. plugins（插件）
        7. environments（环境集合属性对象）
           1. environment（环境子属性对象）

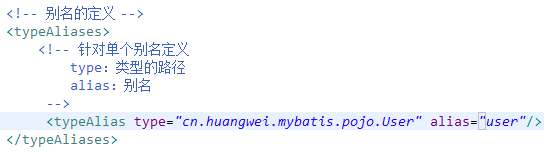
transactionManager（事务管理）

dataSource（数据源）

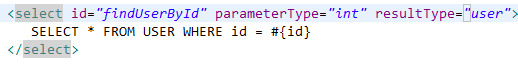
* + - 1. mappers（映射器）
    1. properties属性
       1. 需求
       2. 将数据库连接参数单独配置在db.properties中，只需要在sqlmapConfig.xml中加载db.properties的属性值。在sqlmapConfig.xml中就不需要对数据库的连接参数进行硬编码了
       3. 将数据库连接参数只配置在db.properties中，原因：方便对参数进行统一管理，其他xml可以引用该db.properties的属性值
       4. 
       5. 在sqlmapConfig.xml加载属性文件



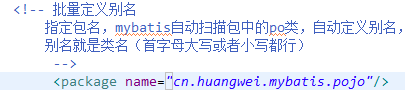
* + - 1. 注意：mybatis将按照下面的顺序来加载属性
         1. 在properties元素体内定义的属性首先被读取
         2. 然后读取properties元素中resource或者url加载的属性，它会覆盖已读取的同名属性
         3. 最后读取parameterType传递的属性，他会覆盖它会覆盖已读取的同名属性
         4. 因此，通过parameterType传递的属性具有最高优先级，resource或url属性次之，最低优先级是properties元素体内定义的属性
         5. 建议:不要在properties元素体内添加任何属性值，只将属性值定义在properties文件中，在properties文件中定义的属性名称 要有一定的特殊性，如xxx.xxx.xxx
    1. settings 全局参数配置
       1. mybatis框架运行时可以调整一些运行参数
          1. 比如：开启二级缓存，开启延迟加载
       2. 全局参数将会影响mybatis的运行行为
    2. typeAliases（别名）重点
       1. 需求
          1. 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
          2. 如果在指定类型时，输入类型全路径，不方便开发，可以定义一些别名，在mapper.xml中通过别名定义，方便开发
       2. mybatis默认支持别名
          1. 
       3. 自定义别名pojo
          1. 单个别名的定义



* + - * 1. 引用别名



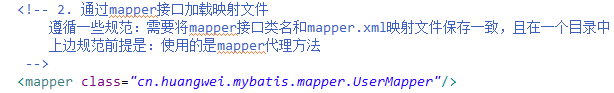
* + - * 1. 批量定义别名



* + 1. typeHandlers（类型处理器）
       1. mybatis中通过typeHandlers完成jdbc类型和java类型的装换
       2. 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
    2. mapper（映射器）
       1. 单个加载映射器
          1. 通过resource加载单个映射文件

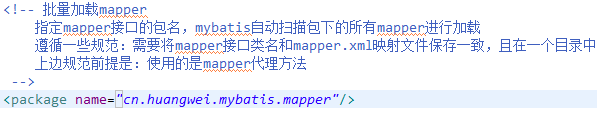
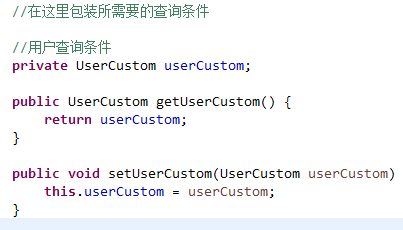
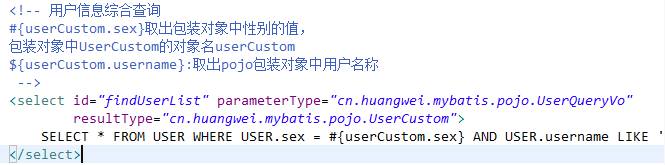


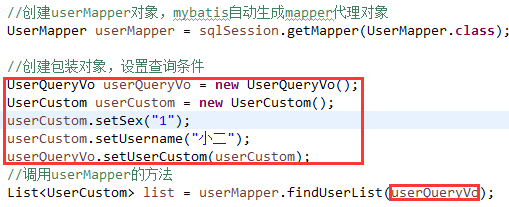
* + - * 1. 通过mapper接口加载映射文件



* + - * 1. 按照上边的规范，将mapper.xml和mapper.java放在一个目录，且同名



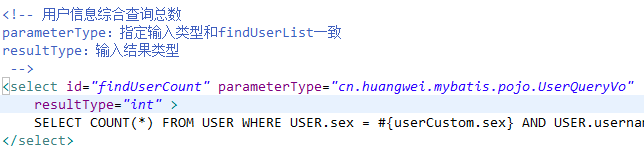
* + - 1. 批量加载映射器（推荐使用）
         1. 
  1. 输入映射
     1. 通过parameterType指定输入参数的类型，类型可以是简单的类型，hashmap，pojo的包装类型等
     2. 传递pojo的包装对象
        1. 需求
           1. 完成用户信息的综合查询，需要传入查询条件（可能包括用户信息，其他信息比如商品，订单的信息）
        2. 定义包装类型pojo
           1. 针对上边需求，建议使用自定义的包装类型的pojo，在包装类型的pojo中将复杂的查询条件包装进去
           2. 
        3. mapper.xml
           1. 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）
           2. 
        4. mapper.java
           1. 
           2. 测试代码



* 1. 输出映射
     1. resultType
        1. 注意
           1. **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。**
           2. 如果查询出来的列名和pojo的属性全部不一致，没有创建pojo对象
           3. 只有查询出来的列名和pojo的属性有一个一致，就会创建pojo对象
        2. 输出简单类型
           1. 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



* + - * 1. 小结

查询出来的结果集只有一行且一列，可以使用简单类型进行输出映射

* + - 1. 输出pojo对象和pojo列表
         1. 不管是输出的pojo对象还是一个列表（list中包括pojo），在mapper.xml中resultType指定的类型是一样的，在mapper.java中指定的方法返回值类型不一样

输出单个pojo对象，方法返回值是单个对象类型



输出pojo对象list，方法返回值是List<Pojo>



* + - * 1. 生成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还是selectList。
    1. resultMap
       1. mybatis中使用resultMap完成高级输出结果映射
       2. resultMap的使用方法

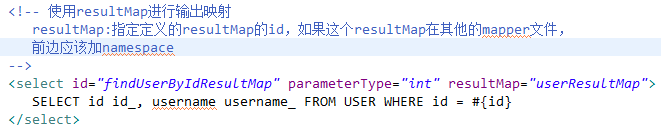
如果查询出来的列名和pojo的属性名不一致，通过定义一个resultMap对列名和pojo属性名之间作一个映射关系

* + - * 1. 定义resultMap
        2. 使用resultMap作为statement的输出映射类型
      1. 将下边的sql使用User类完成映射
         1. 
         2. User类中属性名和上边的查询列名不一致

定义resultMap



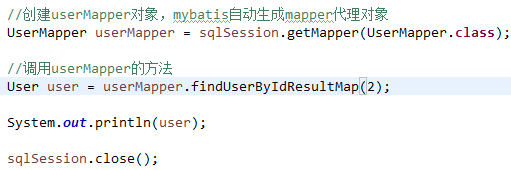
使用resultMap作为statement的输出映射类型



mapper.xml

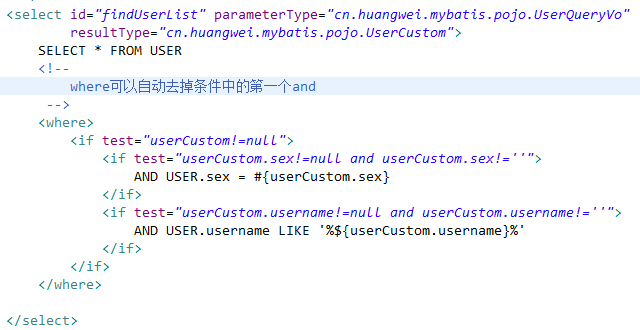
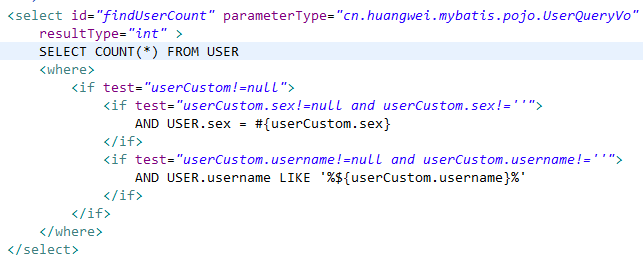


测试



小结

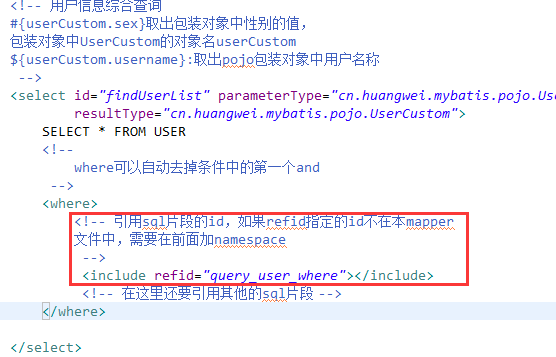
使用resultType进行输出映射时，只有查询出来的列名和pojo的属性名一致，该列才可以映射成功，如果查询出来的 列名和pojo的属性名不一致，通过定义一个resultMap对列名和pojo属性之间做一个映射关系

* 1. 动态sql
     1. 什么是动态sql
        1. mybatis的核心 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
     2. 需求
        1. 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
        2. 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的拼接
     3. mapper.xml
        1. 
        2. 
     4. 测试代码
        1. list
           1. 
           2. 
           3. 
        2. count
           1. 
           2. 
  2. sql片段
     1. 需求
        1. 将上边实现的动态sql判断代码块抽取出来，组成一个sql的片段，其他的statement中就可以引用sql片段
     2. 操作
        1. mapper.xml
           1. 定义sql片段

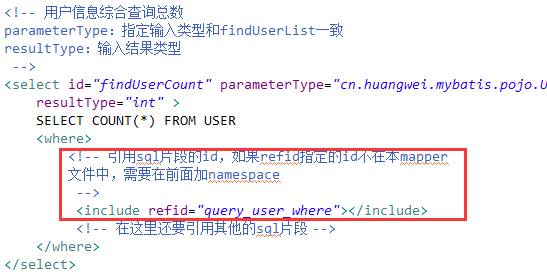


* + - * 1. 引用sql片段

list



count



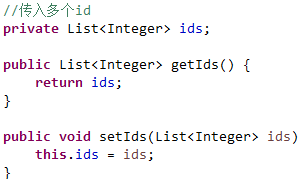
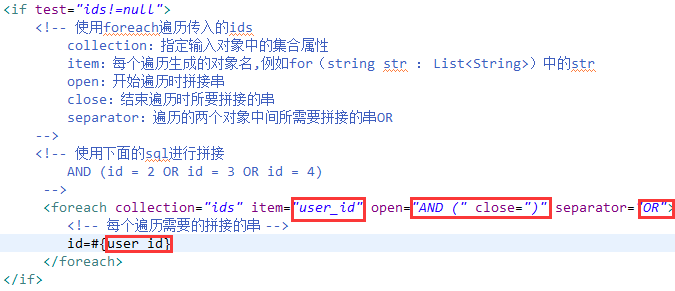
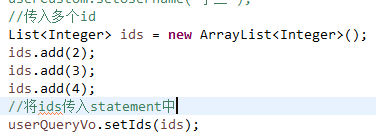
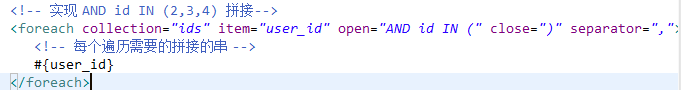
* + - * 1. 测试代码



* 1. foreach
     1. 向sql传递数组或者list，mybatis使用foreach解析
     2. 需求
        1. 用户查询列表和查询总数的statement中增加多个id输入查询
           1. sql语句如下

SELECT \* FROM USER WHERE id = 2 OR id = 3 OR id = 4

SELECT \* FROM USER WHERE id IN(2,3,4)

* + 1. 在输入参数包装类型UserQueryVo中添加List<Integer> ids传入多个id
       1. 
    2. 修改mapper.xml
       1. WHERE id = 2 OR id = 3 OR id = 4，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
          1. 
    3. 测试代码
       1. 
       2. 
       3. 第二种
          1. 

1. mybatis\_day02
   1. 回顾
      1. mybatis是什么？
         1. mybatis是一个持久层框架，mybatis是一个不完全的orm框架，sql语句需要程序猿自己去编写，但是mybatis也有映射（输入参数映射，输出结果映射）
         2. mybatis入门门槛不高，学习成本低，让程序员把精力放在sql语句上面，对sql语句优化非常方便，适用于需求变化较多的项目，比如互联网项目
      2. mybatis的执行过程
         1. 配置mybatis的配置文件，SqlMapConfig.xml（名称不固定）
         2. 通过配置文件，加载mybatis的运行环境，创建SqlSessionFactory会话工厂
            1. SqlSessionFactory在实际使用时按照单例方式
         3. 通过SqlSessionFactory创建SqlSession
            1. SqlSession是一个面向用户的接口（提供操作数据库的方法），实现对象是线程不安全的，建议SqlSession应用场合在方法体内
         4. 调用SqlSession的方法去操作数据
            1. 如果需要提交事务，需要执行sqlSession的commit方法
         5. 释放资源，关闭sqlsession
      3. mybatis开发dao的方法
         1. 原始dao的方法
            1. 需要程序猿编写mapper接口（就是dao接口）
            2. 需要在dao实现类中注入一个sqlSessionFactory工厂
         2. mapper代理开发方法（建议使用）
            1. 只需要程序猿编写mapper接口（就是dao接口），程序猿在编写mapper.xml(映射文件)和mapper.java需要遵循一个开发规范

mapper.xml中namespace就是mapper.java的类全路径

mapper.xml中的id和mapper.java里面的方法名一致

mapper.xml中的parameterType指定参数类型和mapper.java的方法输入参数类型一致

mapper.xml中statement的resultType指定输出结果的类型和mapper.java的方法返回值类型一致

* + 1. 输入映射和输出映射
       1. 输入映射
          1. parameterType:指定输入参数类型可以简单类型，pojo，hashmap
          2. 对于综合查询，建议parameterType使用包装的pojo，有利于系统扩展
       2. 输出映射：
          1. resultType：

查询到的列名和resultType指定的pojo的属性名一致，才能映射成功

* + - * 1. resultMap

可以通过resultMap完成一些高级映射

如果查询到的列名 和映射的pojo的属性名不一致时，通过resultMap设置列名和属性名之间的对应关系（映射关系）,可以完成映射

高级映射

将关联查询的列映射到一个pojo的属性中（一对一）

将关联查询的列映射到一个List<pojo>中（一对多）

* + 1. 动态sql
       1. if判断（掌握）
       2. where
       3. foreach
       4. sql片段（掌握）
  1. 订单商品数据模型
     1. 数据模型分析思路
        1. 分模块对每张表记录的内容进行熟悉，相当于你学习系统需求的过程
        2. 每张表重要的字段设置
           1. 非空字段、外键字段
        3. 数据库级别表与表之间的关系
           1. 外键关系
        4. 表与表之间的业务关系
           1. 在分析表与表之间的业务关系时，一定要建立在某个业务意义基础上去分析
     2. 数据模型分析

订单表orders

number：订单号

user\_id：外键，用户id

订单明细表orderDetail

orders\_id：外键，订单id

items\_id:外键，商品id

商品表items

用户表user

id：自增主键

user\_id外键

orders\_id

items\_id

一对一

一对多

一对多

一对一

一对一

一对多

orders---》items

一对多+一对一=一对多

items--》orders

一对多+一对一=一对多

总之：items和orders是多对多关系

user---》item

一对多+一对多+一对一=一对多

items—》user

一对多+一对一+一对一=一对多

总之：items和user是多对多关系

用户表user：

记录了购买商品的用户信息

订单表order：

记录了用户所创建的订单

订单明细表：orderDetail

记录了订单的详细信息即购买商品的信息

商品表items：

记录了商品的信息

* + 1. 表与表之间的业务关系
       1. 在分析表与表之间的业务关系时，需要件里在某个业务意义基础上去分析数据库级别之间有关系的表之间的业务关系
       2. user和orders
          1. user---》orders：一个用户可以创建多个订单，一对多
          2. orders---》user：一个订单只能被一个用户创建，一对一
       3. orders和orderDetail
          1. orders-🡪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
          2. orderDetail—》orders：一个订单明细只能包括在一个订单中，一对一
       4. orderDetail和items
          1. orderDetail---》items：一个明细对应一个商品，一对一
          2. items---》orderDetail：一个商品可以包括在多个明细中，一对多
       5. 数据库级别间没有关系的表
          1. orders和items

orders和items之间通过orderDetail建立关系

* 1. 一对一查询
     1. 需求
        1. 查询订单信息，关联查询创建订单的用户信息
     2. resultType
        1. sql语句

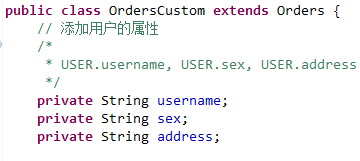
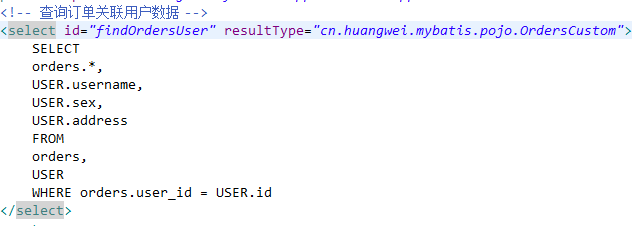
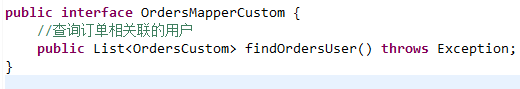
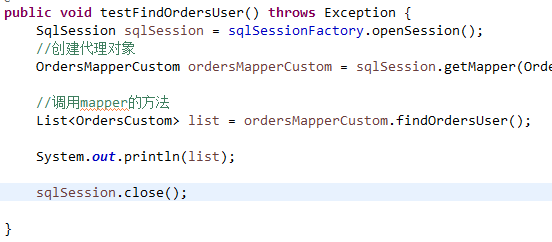
确定查询的主表：订单表

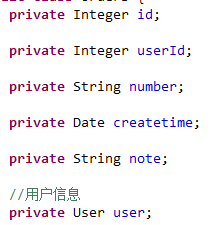
确定查询的关联表：用户表

关联查询使用内连接还是外链接？

由于orders表中有一个外键（user\_id），通过外键关联查询用户表只能查询出一条记录，可以使用内连接

|  |
| --- |
| SELECT  orders.\*,  USER.username,  USER.sex,  USER.address  FROM  orders,  USER  WHERE orders.user\_id = USER.id |

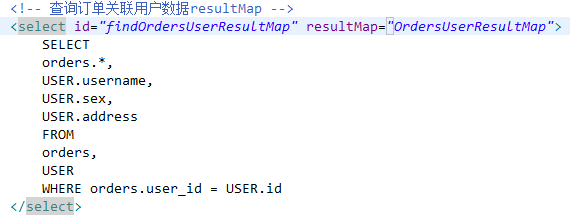
* + - 1. 创建pojo
         1. 将上边sql查询的结果集映射到pojo’中，但是pojo中必须要包括所有的查询列名
         2. 原始的orders.java不能映射所有的字段，需要创建新的pojo
         3. 创建pojo继承包括我们查询字段较多的类
         4. 
      2. mapper.xml
         1. 
      3. mapper.java
         1. 
      4. 测试
         1. 
    1. resultMap
       1. sql语句
          1. 同resultType
       2. 使用resultMap进行映射的思路
          1. 使用resultMap将查询结果中的订单信息映射到Orders对象中，在orders类中添加user属性，将关联查询出来的用户信息映射到orders对象中的user属性
          2. 需要在orders类中添加user属性

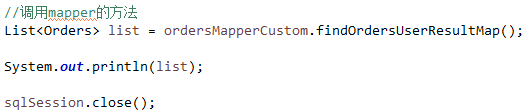


* + - 1. mapper.xml
         1. 定义resultMap



statement



* + - 1. mapper.java
         1. 
      2. 测试
         1. 
    1. resultType和resultMap实现一对一查询小结
       1. 实现一对一查询
          1. resultType：使用resultType实现较为简单，如果pojo中没有包括查询出来的类名，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
          2. resultMap:需要单独定义resultMap，实现有点麻烦，如果对查询结果有特殊要求，使用resultMap可以完成将关联查询映射pojo的属性中。resultMap可以实现延迟加载，resultType无法实现延迟加载
  1. 一对多查询
     1. 需求

查询订单及订单明细的信息

* + 1. sql语句

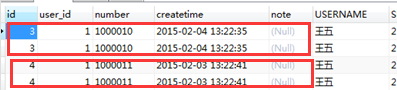
确定主查询表：订单表

确定关联表：订单明细表

|  |
| --- |
| SELECT  orders.\*,  USER.USERNAME,  USER.SEX,  USER.ADDRESS,  orderdetail.id orderdetail\_id,  orderdetail.items\_id,  orderdetail.items\_num,  orderdetail.orders\_id  FROM  orders,  USER,  orderdetail  WHERE orders.user\_id = USER.id AND orderdetail.orders\_id=orders.id |

* + 1. 分析

使用resultType将上边的查询结果映射到pojo中，订单的信息就是重复的

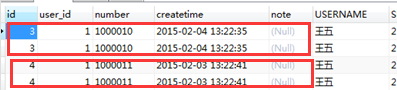


要求：

对orders映射不能出现重复记录。

在orders.java类中添加List<orderDetail> orderDetails属性

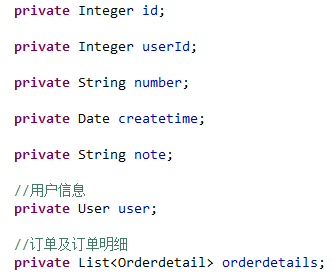
最终会将订单信息映射到orders中，订单所对应的订单明细映射到orders中的orderDetails属性中。



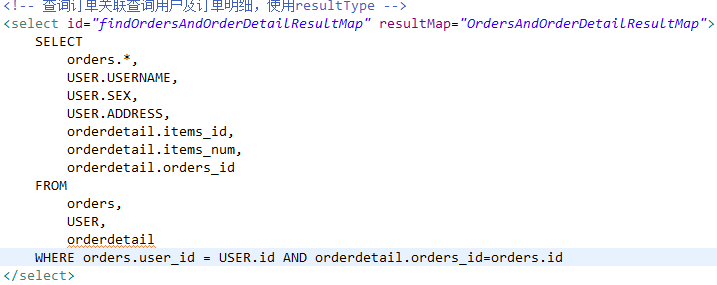
映射成orders的记录数为两条（orders信息不重复）

每个orders中的orderDetails属性存储了该订单所对应的订单明细

* + 1. 在orders中添加list订单明细属性



* + 1. mapper.xml



|  |
| --- |
| <resultMap type=*"cn.huangwei.mybatis.pojo.Orders"* id=*"OrdersAndOrderDetailResultMap"* extends=*"OrdersUserResultMap"*>  <!-- 订单信息 -->  <!-- 用户信息 -->  <!-- 上面两个信息已经继承 -->  <!-- 订单明细  一个订单关联查询出了多条明细，要使用collection进行映射  collection：对关联查询到多条记录映射到集合对象中  property：将关联查询多条记录映射到cn.huangwei.mybatis.pojo.Orders的那个属性  ofType:指定映射到orders类中集合属性中的pojo的类型  -->  <collection property=*"orderdetails"* ofType=*"cn.huangwei.mybatis.pojo.Orderdetail"*>  <!--  id:订单明细唯一标识  property：要将订单明细的唯一标识映射到cn.huangwei.mybatis.pojo.Orderdetail的那个属性  -->  <id column=*"orderdetail\_id"* property=*"id"*/>  <result column=*"items\_id"* property=*"itemsId"*/>  <result column=*"items\_num"* property=*"itemsNum"*/>  <result column=*"orders\_id"* property=*"ordersId"*/>  </collection>    </resultMap> |

* + 1. mapper.java



* + 1. 小结

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：

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，将订单明细存储在orderDetails中

* 1. 多对多查询
     1. 需求

查询用户及用户购买商品信息

* + 1. 分析（sql语句）

查询主表：用户表

关联表：由于用户和商品没有直接关联，通过订单和订单明细进行惯量，所以关联表有orders，orderDetail，items

|  |
| --- |
| SELECT  orders.\*,  USER.username,  USER.sex,  USER.address,  orderdetail.id orderdetail\_id,  orderdetail.items\_id,  orderdetail.items\_num,  orderdetail.orders\_id,  items.name items\_name,  items.detail items\_detail,  items.price items\_price  FROM  orders,  USER,  orderdetail,  items  WHERE orders.user\_id = USER.id AND orderdetail.orders\_id = orders.id  AND orderdetail.items\_id = items.id |

* + 1. 映射思路

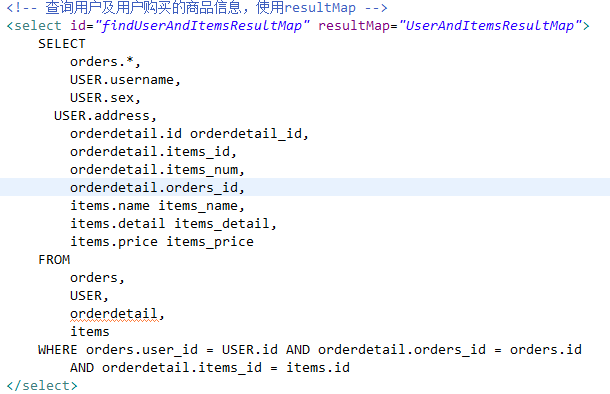
将用户信息映射到user中

在user类中添加订单列表的属性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> orderdetails，将订单的明细映射到orderDetails

在orderDetail中添加items属性，将订单明细所对应的商品映射到items

* + 1. mapper.xml



* + 1. resultmap的定义

|  |
| --- |
| <!-- 查询用户及用户购买的商品信息 -->  <resultMap type=*"cn.huangwei.mybatis.pojo.User"* id=*"UserAndItemsResultMap"*>  <!-- 用户信息 -->  <id column=*"user\_id"* property=*"id"*/>  <result column=*"username"* property=*"username"*/>  <result column=*"sex"* property=*"sex"*/>  <result column=*"address"* property=*"address"*/>    <!-- 订单信息 -->  <collection property=*"ordersList"* ofType=*"cn.huangwei.mybatis.pojo.Orders"*>  <id column=*"id"* property=*"id"*/>  <result column=*"user\_id"* property=*"userId"*/>  <result column=*"number"* property=*"number"*/>  <result column=*"createtime"* property=*"createtime"*/>  <result column=*"note"* property=*"note"*/>    <!-- 订单明细  一个订单包括多个明细  -->  <collection property=*"orderdetails"* ofType=*"cn.huangwei.mybatis.pojo.Orderdetail"*>  <id column=*"orderdetail\_id"* property=*"id"*/>  <result column=*"items\_id"* property=*"itemsId"*/>  <result column=*"items\_num"* property=*"itemsNum"*/>  <result column=*"orders\_id"* property=*"ordersId"*/>    <!-- 商品信息  一个明细对应一个商品  -->  <association property=*"items"* javaType=*"cn.huangwei.mybatis.pojo.Items"*>  <id column=*"items\_id"* property=*"id"*/>  <result column=*"items\_name"* property=*"name"*/>  <result column=*"items\_detail"* property=*"detail"*/>  <result column=*"items\_price"* property=*"price"*/>  </association>    </collection>    </collection>    </resultMap> |

* + 1. mapper.java



* + 1. 多对多查询总结

将查询用户购买的商品信息明细清单，（用户名，用户地址，购买商品名称，购买商品时间，购买商品数量）

针对上边需求就是用resultType将查询到的记录映射到一个扩展的pojo中，很简单实现明细清单的功能

一对多是多对多的特例，如下需求：

查询用户购买的商品信息，用户和商品的关系是多对多的关系

需求1：查询字段：用户账号、用户名称、用户性别、商品名称、商品价格

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：

查询字段：用户账号、用户名称、购买商品数量、商品明细（鼠标以上显示明细）

使用resultMap将用户购买的商品明细列表映射到user对象

总结：使用resultMap是针对那些对查询结果映射有特殊要求的功能。

* 1. resultMap总结
     1. 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，比如用户购买商品明细，将关联查询信息全部展示在页面时，此时可直接使用resultType将每一天记录映射到pojo中，在前端页面遍历list（list中是pojo）即可。

* + 1. resultMap

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

association:

作用：将关联查询信息映射到一个pojo对象中

场合：为了方便查询关联信息可以使用association将关联订单信息映射为用户对象的pojo属性中，比如查询订单及关联用户信息

使用resultType无法将查询结果映射到pojo对象的pojo属性，根据对结果集查询遍历的需要选择使用resultType和resultMap。

collection：

作用：将关联查询信息映射到一个list集合中

场合：

为了方便查询遍历关联信息可以使用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及模块下的菜单，可使用collection将模块映射到模块list中，将菜单列表映射到模块对象的菜单list属性中，这样的作用也是方便对查询结果集进行遍历查询。

如果使用resultType无法将查询结果映射到list集合中

* 1. 延迟加载
     1. 什么是延迟加载

resultMap可以实现高级映射（association。collection实现一对一和一对多的映射）association。collection具备延迟加载功能。

需求：如果查询订单并且关联用户信息，先查询订单信息即可满足要求，当我们需要查询用户信息时，在查询用户信息，把对用户信息的需求，按需查询就是延迟加载

延迟加载：先从单表查询，需要时再从关联表去关联查询，大大提高数据库性能，因为查询单表要比关联查询多张表速度要快

* + 1. 使用association实现延时加载
       1. 需求

查询订单并且关联查询用户信息

* + - 1. mapper.xml

需要定义两个maper的方法对应的statement

1. 查询订单信息

select \* from orders

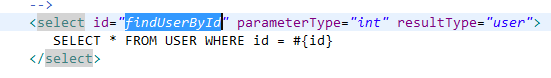
在查询订单的statement中使用association去延迟加载（执行）下边的statement（关联查询用户信息）

|  |
| --- |
| <!-- 查询订单关联用户，用户信息需要延迟加载 -->  <select id=*"findOrdersUserLazyLoading"* resultMap=*"OrdersUserLazyLoading"*>  SELECT \*　FROM orders  </select> |

1. 在关联查询用户信息

通过上边查询到的订单信息中user\_id去关联查询用户信息

使用userMapper.xml中的findUserById



上面先去执行*findOrdersUserLazyLoading，当需要去查询用户的时候再去执行findUserByid，通过resultMap的定义将延时加载执行配置起来*

* + - 1. 延迟加载resultMap

(association)

|  |
| --- |
| <!-- 延迟加载的resultMap -->  <resultMap type=*"cn.huangwei.mybatis.pojo.Orders"* id=*"OrdersUserLazyLoading"*>  <!-- 对订单信息进行映射配置 -->  <id column=*"id"* property=*"id"*/>  <result column=*"user\_id"* property=*"userId"*/>  <result column=*"number"* property=*"number"*/>  <result column=*"createtime"* property=*"createtime"*/>  <result column=*"note"* property=*"note"*/>  <!-- 实现对用户信息进行延迟加载  select：指定延迟加载需要执行的statement的id（是根据user\_id查询用户信息的statement）  要使用userMapper.xml中的findUserById完成根据用户id（user——id）进行用户信息的查询，  如果findUserById不在本mapper.xml中，需要在findUserById前面加上userMapper.xml的命名空间  colomn：订单信息中关联用户信息查询的列，就是orders.user\_id=user.id中的user\_id  关联查询的sql理解为：  SELECT orders.\*,  (SELECT username FROM USER WHERE  orders.user\_id = USER.id  )username,  (SELECT sex FROM USER WHERE  orders.user\_id = USER.id  )sex  FROM orders  -->  <association property=*"user"* javaType=*"cn.huangwei.mybatis.pojo.User"*  select=*"cn.huangwei.mybatis.mapper.UserMapper.findUserById"* column=*"user\_id"*>  <!-- 实现用户信息进行延时加载 -->    </association>  </resultMap> |

* + - 1. mapper.java



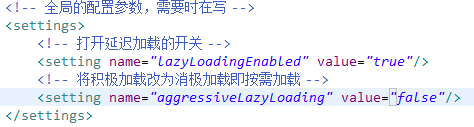
* + - 1. 测试
         1. 思路：

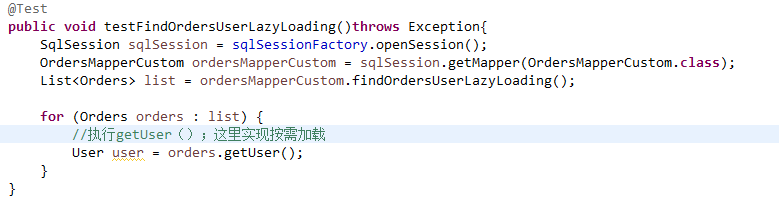
1. 执行上面mapper方法（findOrdersUserLazyLoading），内部调用ordersMapperCustom.xml中的findOrdersUserLazyLoading只查询order信息（单表）
2. 在程序中去遍历上一步骤查询出的List《orders》，当我们调用orders中的getUser方法时，开始进行延迟加载
3. 延迟加载，去调用userMapper.xml中findUserById这个方法获取用户信息
   * + - 1. 延迟加载配置

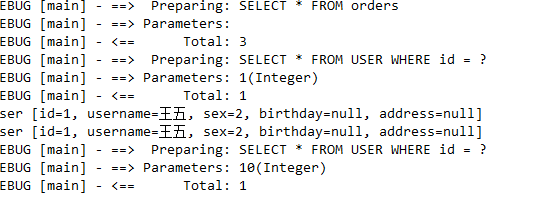
mybatis默认没有开启延迟加载，需要在sqlmapconfig.xml中setting参数配置

|  |  |  |  |
| --- | --- | --- | --- |
| 设直线 | 描述 | 允许值 | 默认值 |
| lazyLoadingEnabled | 全局设置懒加载，如果设为false，则所有相关联的都会被初始化加载 | True | False |
| AggressiveLazyLoading | 当设置为true是，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 |

在sqlmapconfig.xml中写



* + - * 1. 



* + - 1. 延迟加载思考
         1. 不使用mybatis提供的association及collection中的延迟加载，如何实现延迟加载？
         2. 实现方法：

定义两个mapper方法：

查询订单列表

根据用户id查询用户信息

* + - * 1. 实现思路

先去查询第一个mapper方法，获取订单信息列表

在程序中（service）按需去调用第二个mapper方法去查询用户信息

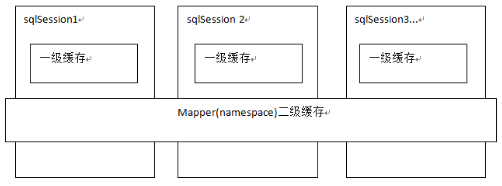
* + - * 1. 总之：

使用延时加载方法，先去查询简单的sql（最好单表，也可以关联查询），再去按需要加载关联查询的其他信息

* 1. 查询缓存
     1. 什么是查询缓存

mybatis提供查询缓存，用于减轻数据压力，提高数据库性能

mybatis提供一级缓存和二级缓存



一级缓存是sqlsession级别的缓存

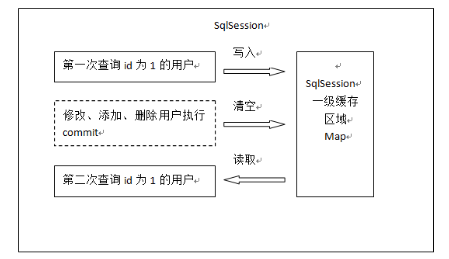
在操作数据库时，需要构造sqlsession对象，在对象中有一个数据结构（hashmap）用于存储缓存数据，不同的sqlsession之间的缓存数据区域（hashmap）是互相不影响的

二级缓存是mapper级别的缓存，多个sqlsession去操作同一个mapper的sql语句，多个sqlsession去操作数据库得到数据会存在耳机缓存区域，多个sqlsession可以共用二级缓存，二级缓存是跨sqlsession的。

为什么要用缓存？

如果缓存中有数据就不用从数据库中获取，提高系统的性能

* + 1. 一级缓存
       1. 一级缓存的工作原理



第一次发起查询用户id为1的用户信息，先去找缓存中是否有id为1的用户信息，如果没有，从数据库查询用户信息。

得到用户信息后，将用户信息存到一级缓存中。

如果sqlsession去执行commit操作（执行插入，删除，更新），清空sqlsession的一级缓存，这样做的目的为了让一级缓存中存储的是最新的信息，避免脏读

第二次发起查询用户id为1的用户信息，先去找缓存中是否有id为1的用户信息，缓存中有，直接从缓存中获取用户信息。

* + - 1. 一级缓存的测试

mybatis默认支持一级缓存，不需要再配置文件中去配置。

按照上边一级缓存原理步骤去测试



* + - 1. 一级缓存的应用

正式开发，是将mybatis和spring进行整合开发，事务控制在service中。

一个service方法中包括很多mapper方法调用

service{

//开始执行时，开启事务，创建Sqlsession对象

//第一次调用mapper方法findUserById（1）

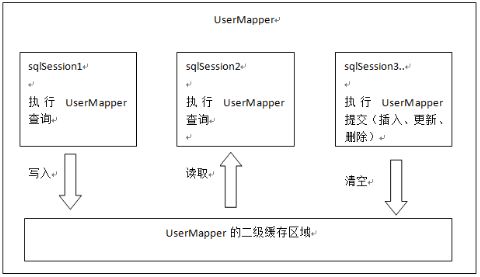
//第二次调用mapper的方法findUserById（1），从一级缓存中取数据

//方法结束，sqlSession关闭

}

如果是执行两次service调用查询相同的用户信息，不走一级缓存，因为session方法结束，sqlSession方法结束，sqlSession就关闭，一级缓存就清空

* + 1. 二级缓存
       1. 原理



首先开启mybatis的二级缓存

sqlSession1去查询用户id为1的用户信息，查询到用户信息会将查询数据存储到二级缓存中。

sqlsession2去查询用户id为1的用户信息，去缓存中找是否存在数据，如果存在直接从缓存中取出数据

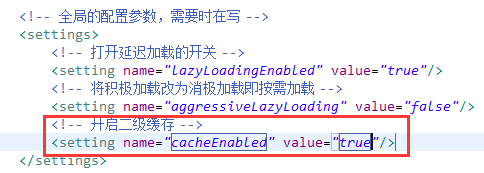
如果sqlsession3执行相同的mapper，并且commit之后，对应的二级缓存就会被清空

二级缓存和一级缓存的区别，二级缓存的范围更大，多个sqlSession可以共享一个UserMapper的二级缓存区域。各自的mapper都有各自的二级缓存区域。每一个namespace的mapper有一个二级缓存区域，如果userMapper和orderMapper共用一个namespace，那么他们共用一个二级缓存区域v

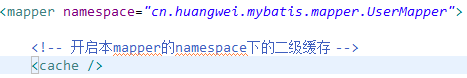
* + - 1. 开启二级缓存

mybatis的二级缓存是mapper范围级别，除了在sqlmapConfig设置二级缓存的总开关，还要在具体的mapper.xml里面开启二级缓存

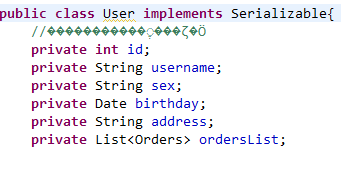
sqlmapConfig.xml文件



在userMapper.xml中开启二级缓存

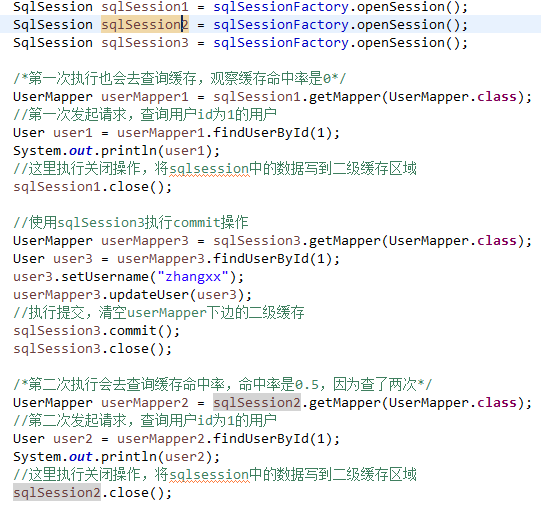


* + - 1. 调用pojo类实现序列化接口



为了将缓存数据取出执行反序列化操作，因为二级缓存数据存储介质多种多样，在内存里面不一样

* + - 1. 测试方法



* + - 1. userCache配置禁用二级缓存

在statement中设置usercache=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” >

总结：针对每次查询都需要最新的数据sql，要设置成userCache=false，禁用二级缓存

* + - 1. 刷新缓存（就是清空）

在mapper的用一个namespace中，如果有其他insert，update、delete操作数据后需要刷新缓存，如果不执行刷新缓存会出现脏读

设置statement配置中的flushCache=”true”属性，默认情况下为true即刷新缓存，如果改成false则不会刷新，使用缓存时如果手动修改数据库表中的查询数据会出现脏读

如下：

<insert id=”insertUser” parameterType=”cn.itcast.mybatis.po.User” flushCache=”true”>

总结：一般下执行完commit操作都需要刷新缓存，避免脏读

* + 1. mybatis整合ehcache

ehcache是一个分布式缓存

* + - 1. 分布缓存

我们系统为了提高系统并发性能，一般对系统进行分布式部署（集群部署方式）

系统工程

springmvc

mybatis

服务器1

系统工程

springmvc

mybatis

服务器2

对缓存数据进行集中管理，使用分布式缓存框架

redis、memache、ehcache

不使用分布缓存，缓存的数据在各个服务器单独存储，会导致各个服务器之间不共享，例如要各个服务器里面登录好多次。

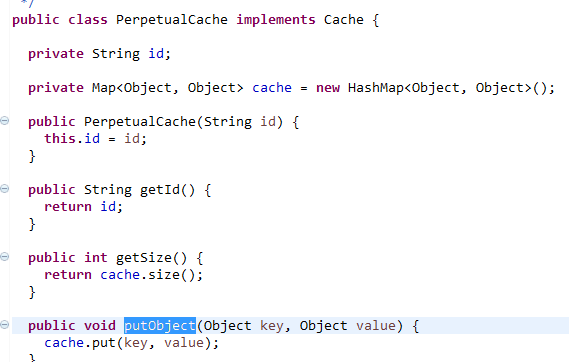
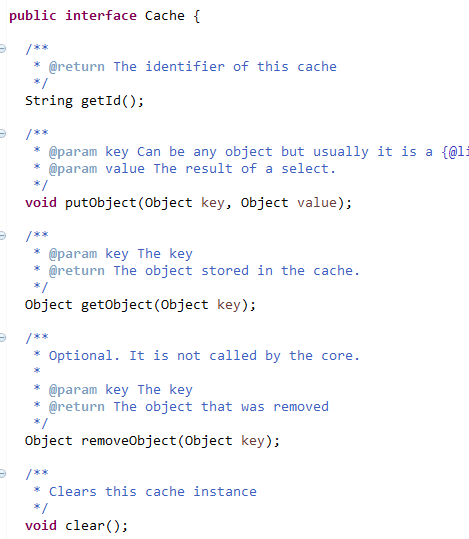
mybatis无法实现分布式缓存，需要和其他的分布式缓存框架进行整合

* + - 1. 整合方法（掌握）

mybatis提供了一个cache接口，如果要实现自己的缓存逻辑，实现cache接口开发即可

mybatis和ehcache整合，mybatis和ehcache整合包中提供了一个cache接口的是实现类

左下是mybatis的cache接口 右下是mybatis的cache接口实现类



* + - 1. 加入ehcache的包



* + - 1. 整合ehcache

配置mapper中的cache的type为ehcache的全路径



* + - 1. 入ehcache的配置文件

在classpath下加入ehcache的配置文件

|  |
| --- |
| <ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:noNamespaceSchemaLocation="../config/ehcache.xsd">  <diskStore path="F:\develop\ehcache" />  <defaultCache  maxElementsInMemory="1000"  maxElementsOnDisk="10000000"  eternal="false"  overflowToDisk="false"  timeToIdleSeconds="120"  timeToLiveSeconds="120"  diskExpiryThreadIntervalSeconds="120"  memoryStoreEvictionPolicy="LRU">  </defaultCache>  </ehcache> |

* + 1. 二级缓存的应用场景

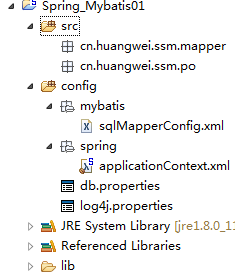
对于访问多的查询请求且用户对查询结果的实时性要求不高，此时可采用mybatis二级缓存技术降低数据库的访问量，提高访问速度，业务场景：耗时较高的统计分析sql，电话账单查询sql等

实现方法：通过设置刷新间隔时间，由mybatis每个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30分钟，60分钟，24小时等，根据需求而定

* + 1. 二级缓存的局限性

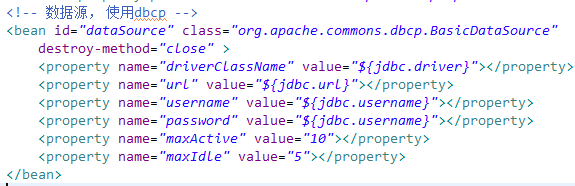
mybatis二级缓存对细粒度的数据级别的缓存实现不好，比如如下需求：对商品信息进行缓存，由于商品信息查询访问量大，但是要求用户每次都能查询最新的商品信息，此时如果使用mybatis的二级缓存就无法实心当一个商品变化时只需要刷新该商品的缓存信息而不刷新其他商品的信息，因为mybatis的二级缓存是mapper为单位划分的，当一个商品信息变化会将所有的商品信息的缓存数据全部清空，解决此问题需要在业务层根据需求对数据有针对性的缓存

**6-15暂时跳过**

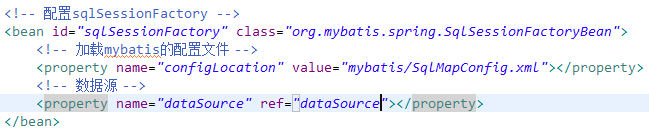
* 1. spring和mybatis的整合
     1. 整合思路
        1. 需要spring通过单例的方式管理SqlSessionFactory，spring和mybatis的整合生成代理对象，使用sqlSessionFactory创建sqlsession（spring和mybatis整合自动完成的），持久层的mapper都需要有spring进行管理
     2. 整合的环境搭建
        1. 创建新的java工程（接近实际开发的工程结构）
        2. jar包
           1. mybatis 3.2.7的jar包
           2. spring4.2.4的jar包、
           3. mybatis和spring整合包：早期ibatis和spring整合是由spring官方提供，mybatis和spring整合是由mybatis提供
        3. 
     3. sqlSessionFactory的配置
        1. 配置applicationContext.xml
           1. 配置数据库配置文件



* + - * 1. 配置数据源

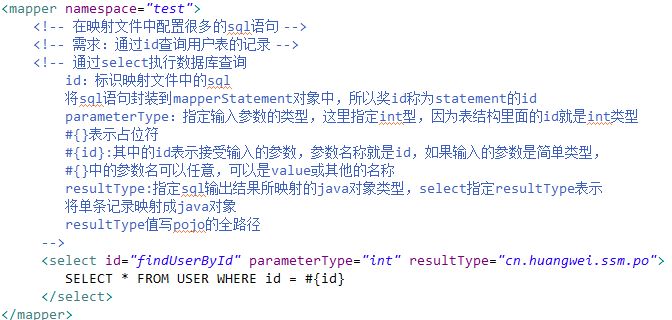


* + - * 1. 配置sqlSessionFactory在applicationContext.xml里面



* + - * 1. 原始dao开发（和spring整合后）

**user**.xml



**注意：sqlMapConfig.xml中加载user.xml**



dao

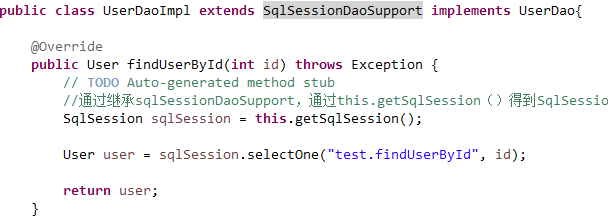
interface Userdao



dao接口实现类

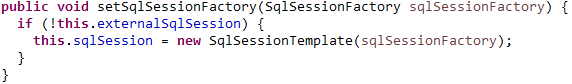
dao接口实现类需要注入sqlSessionFactory，通过spring进行注入，这里用spring声明配置方式，配置dao的bean

让userDaoImpl实现类继承SqlSessionDaoSupport

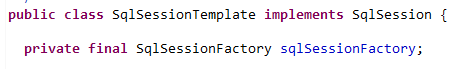


配置dao

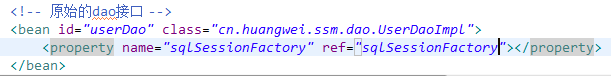
在applicationContext.xml中配置dao



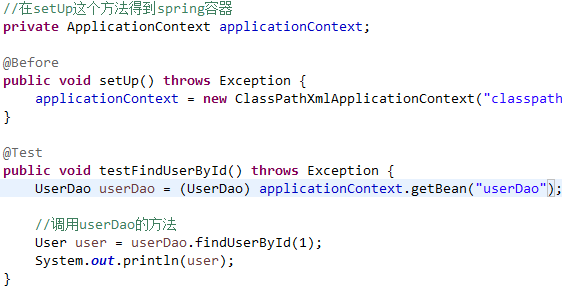




正因为有上面的，所以name才是sqlSessionFactory



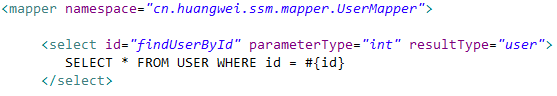
测试



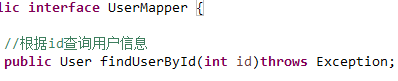
mapper代理开发



mapper.xml



mapper.java



在applicationContext里面配置mapperFactoryBean

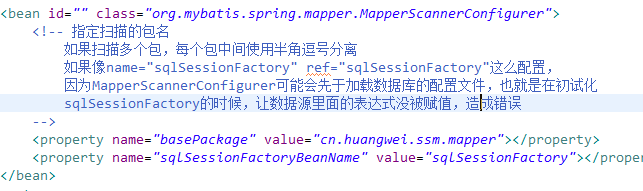






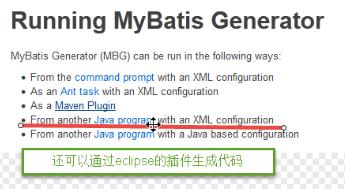
此方法需要对每个mapper进行配置，麻烦

**解决：通过MapperScannerConfigurer对mapper进行扫描（推荐使用）**



测试代码



* 1. 逆向工程
     1. 什么是逆向工程
        1. mybatis需要程序猿自己编写sql语句，mybatis官方提供逆向工程，可以针对单表自动生成mybatis执行所需要的代码（mapper.java, mapper.xml, po）
        2. 企业的实际开发中，常用的逆向工程方式：
           1. 由于数据库的表生成java代码
     2. 下载逆向工程
        1. 
     3. 使用方法
        1. 
        2. 建议使用java程序与进行逆向工程，不依赖开发工具
           1. 生成代码的xml配置文件

|  |
| --- |
| <?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=*"testTables"* targetRuntime=*"Mybatis3"*>  <commentGenerator>  <!-- 是否去除自动生成的注释 true ||false -->  <property name=*"suppressAllComments"* value=*"true"*/>  </commentGenerator>    <!-- mysql:数据库连接的信息：驱动类，链接地址，用户名，密码 -->  <jdbcConnection driverClass=*"com.mysql.jdbc.Driver"*  connectionURL=*"jdbc:mysql://localhost:3306/mybatis\_02"* userId=*"root"*  password=*"123456"*>  </jdbcConnection>    <!-- oracle -->  <!-- <jdbcConnection driverClass="oracle.jdbc.OracleDriver"  connectionURL="jdbc:oracle:thin:@127.0.0.1:1521:yycg"  userId="yycg"  password="yycg">  </jdbcConnection> -->    <!-- 默认false，把jdbc decimal 和numeric 类型解析为Integer，为true是把jdbc decimal  和numeric 类型解析为java.math.BigDecimal-->  <javaTypeResolver>  <property name=*"forceBigDecimal"* value=*"false"*/>  </javaTypeResolver>    <!-- targetProject:生成PO类的位置 -->  <javaModelGenerator targetPackage=*"****cn.huangwei.ssm.po****"*  targetProject=*".\src"*>  <!-- enableSubPackages:是否 让schema作为包的后缀-->  <property name=*"enableSubPackages"* value=*"false"*/>  <!-- 从数据库返回的值被清理前后的空格 -->  <property name=*"trimStrings"* value=*"true"*/>  </javaModelGenerator>    <!-- targetProject：mapper映射文件生成的位置 -->  <sqlMapGenerator targetPackage=*"****cn.huangwei.ssm.mapper****"*  targetProject=*".\src"*>  <!-- enableSubPackages：是否让schema作为包的后缀 -->  <property name=*"enableSubPackages"* value=*"false"* />  </sqlMapGenerator>    <!-- targetPackage:mapper接口生成的位置 -->  <javaClientGenerator targetPackage=*"****cn.huangwei.ssm.mapper****"*  type=*"XMLMAPPER"*  targetProject=*".\src"*>  <!-- enableSubPackages:是否让schema作为包的后缀 -->  <property name=*"enableSubPackages"* value=*"false"*/>  </javaClientGenerator>    <!-- 指定数据库表 mysql-->  **<table tableName=*"items"*></table>**  **<table tableName=*"orders"*></table>**  **<table tableName=*"orderdetail"*></table>**  **<table tableName=*"user"*></table>**    <!-- oracle -->  <!-- <table schema="" tableName="sys\_user" ></table>  <table schema="" tableName="sys\_role" ></table>  <table schema="" tableName="sys\_permission" ></table>  <table schema="" tableName="sys\_user\_role" ></table>  <table schema="" tableName="sys\_role\_permission" ></table> -->    <!-- 有些表的字段需要指定java类型 -->  <!-- <table tableName="" schema="">  <columnOverride column="" javaType=""/>  </table> -->  </context>  </generatorConfiguration> |

* + - 1. 执行生成程序
         1. 需要将生成工程中所生成的代码拷贝到自己的工程中
         2. 测试itemsMapper中的方法