**MyBatis Study Notes**

**@author Count Monte Cristo**

**@create D20150831**

**@version 2016年04月08日 13:24:15**

**~~D20150915~~ ~~D20151025~~ ~~D20151102~~ ~~D20151221~~ D20160317**

**@description 记录MyBatis的重要知识**

**@reference 传智播客MyBatis学习视频**

**@motto VICTORY LOVES PREPARATOIN**

**@criteria 凯旋基诺**

**@copyright CMC all rights reserved (c)**

**<!-- 基础部分 -🡪**

1. **MyBatis简介**
   1. @reference <http://mybatis.github.io/>
2. **MyBatis课程简介**
   1. **基础知识（第一天）**
      1. 对原生态JDBC程序问题总结
      2. **MyBatis框架原理（掌握）**
      3. MyBatis入门程序
         1. 用户的CRUD
      4. **MyBatis开发Dao的两种方法**
         1. 原始Dao开发方法（掌握）
            1. 程序需要编写Dao接口和Dao的实现类
         2. MyBatis的Mapper接口（掌握）
      5. MyBatis配置文件SqlMapConfig.xml
      6. **MyBatis核心（掌握）**
         1. 输入映射
         2. 输出映射
      7. **MyBatis的动态SQL（掌握）**
   2. **高级知识（第二天）（掌握）**
      1. 订单商品数据模型分析
      2. **高级映射**
         1. OneToOne
         2. OneToMany && ManyToOne
         3. ManyToMany
      3. **MyBatis延迟加载**
      4. **MyBatis查询缓存（一级缓存、二级缓存）**
      5. **MyBatis和Spring的整合**
      6. MyBatis的逆向工程
3. **对原生态JDBC程序中问题的总结**
   1. **环境**
      1. Java环境：jdk1.7
      2. Eclipse：Luna
      3. MySQL：5.1
   2. **PreparedStatement**
      1. 是数据库预编译的Statement（数据库语句）
      2. 把编译的结果存放在数据库中的缓存中，下次如果Statement相同的话就可以利用缓存
   3. **问题**
      1. **数据库连接，使用的时候就创建，不使用就立即释放**
         1. 缺点
            1. 对数据库造成频繁的打开和关闭，浪费数据库资源，降低性能
         2. 解决方案
            1. 可以使用数据库连接池解决该问题
      2. **将SQL语句硬编码（不利于系统的维护）到Java代码中**
         1. 缺点
            1. 如果SQL语句修改，重新编译Java代码
         2. 解决方案
            1. 将SQL写在配置文件中，即使SQL变化也不用再重新编译Java代码，只是需要的时候进行加载配置文件
      3. **向PreparedStatement中设置参数，对占位符位置和设置的参数，硬编码到了Java代码中**
         1. 缺点
            1. 不利于系统维护
         2. 解决方案
            1. 把占位符的位置信息和参数写在配置文件中
      4. **从ResultSet中遍历结果集数据时，存在硬编码**
         1. 解决方案
            1. 将查询的结果集自动映射到Java对象
4. **MyBatis框架原理**
   1. **MyBatis是一个持久层框架，Apache的顶级项目（托管到GitHub）**
      1. 让程序员把主要的精力放在SQL逻辑上，通过MyBatis提供的映射方式，自由灵活地生成（半自动化，大部分需要程序员写SQL）满足需求的SQL

SqlMapConfig.xml（是MyBatis的全局配置文件，名称不固定）配置了数据源、事务等MyBatis运行环境、配置映射文件（配置SQL语句）：mapper.xml

SqlSessionFactory（会话工厂），根据配置文件创建工厂

作用：创建SqlSession

MySQL

mappedStatement（底层封装对象）

作用：对数据库操作进行存储封装 ，包括SQL语句、输入参数、输出结果类型

SqlSession（会话），是一个接口，面向用户（程序员）的接口

作用：操作数据库（发出SQL，CRUD）

Executor（执行器），是一个接口（基本执行器、缓存执行器）

作用：SqlSession内部通过执行器操作数据库

输入参数类型

Java简单类型

HashMap

POJO自定义

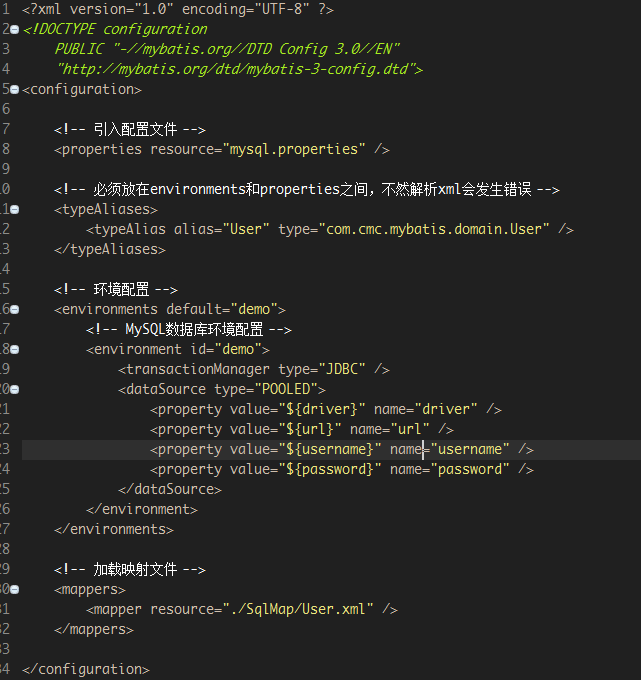
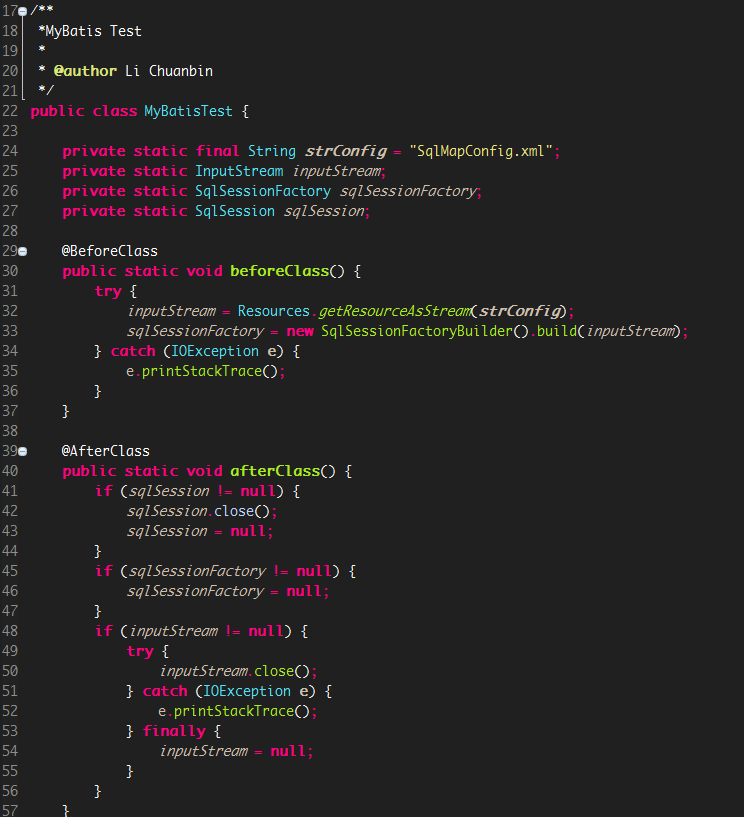
输出结果类型

Java简单类型

HashMap

POJO自定义

* + 1. MyBatis可以将向PreparedStatement中输入的参数自动进行输入映射（输入映射），将查询结果灵活映射成Java对象（输出映射）
  1. **MyBatis框架**
     1. 结构图

1. **入门程序**
   1. @reference MyBatis\_Demo\_HelloWorld2
   2. **环境配置**
      1. 添加MyBatis的相关jar包
         1. @reference <http://mvnrepository.com/>
         2. 添加其他需要的jar包
   3. **程序解析**
      1. **全局配置文件**
      2. **查询配置文件**
      3. **实体**
         1. 普通JavaBean
      4. **测试类**

****

* 1. **知识总结**
     1. **parameterType**
        1. 在映射文件中通过parmeterType指定输入参数的类型
     2. **resultType**
        1. 在映射文件中指定返回结果数据的类型
     3. **#{}和${}**
        1. #{}表示一个占位符，输入的参数类型可以是简单数据类型、POJO、HashMap
           1. 如果是简单数据类型，则#{}中可以写成value和其他名称
           2. 接受POJO对象的值，使用OGNL读取对象中的属性值，如果User中嵌套User，则可以使用OGNL的#{属性.属性...}的方式获取嵌套对象属性的属性值
        2. ${}表示一个拼接符号，但是避免不了SQL注入，不建议使用
           1. 接受的参数类型可以是简单数据类型、POJO、HashMap
           2. 当接受简单数据类型的时候，${}中的参数名只能写value
     4. **selectOne和selectList**
        1. selectOne表示查询出一条记录，如果使用selectOne可以实现查询结果，selectList也可以实现，只是此时List就一条数据而已
        2. selectList表示查询出多条数据，如果使用selectOne，且此时数据多条，则会报错

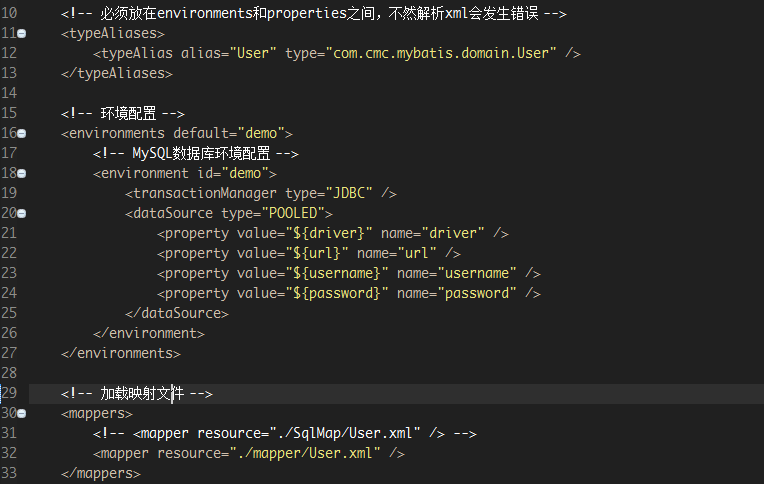
1. **获取主键**
   1. **获取自增主键的值**
      1. **MySQL自增主键**
         1. 执行insert提交之后自动生成主键（一般是自增id主键）
         2. 通过MySQL函数获取到刚刚插入记录的自增主键（last\_insert\_id()）
         3. 调用时间
            1. 在insert语句之后调用
   2. **获取非自增主键的值**
      1. **情况**
         1. 使用MySQL的uuid()生成主键，此时需要修改id的字段类型为String类型，（数据库）长度设置为35
      2. **执行思路**
         1. 先通过uuid()查询到主键，将主键输入到SQL语句中
         2. 执行uuid()的顺序先于insert语句
2. **MyBatis的CRUD**
   1. @reference MyBatis\_Demo\_HelloWorld2
   2. 具体细节参考实例注释
3. **MyBatis和Hibernate本质区别和应用场景**
   1. **MyBatis**
      1. **本质**
         1. 专注于SQL本身，需要程序员自己编写SQL语句，SQL修改、优化比较方便
         2. 是一个不完全的ORM框架，虽然需要程序员自己写SQL，但是MyBatis也可以实现映射（输入映射和输出映射）
      2. **应用场景**
         1. 适用于需求变化较多的项目，比如：互联网项目
   2. **Hibernate**
      1. **本质**
         1. 是一个标准的ORM框架（对象关系映射）。入门门槛较高，SQL语句自动生成，不需要程序员手动写（当然也可以自定义），对SQL语句进行了优化，修改比较困难
      2. **应用场景**
         1. 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等
   3. **扩展**
      1. 企业进行技术选型：低成本、高回报作为技术选型的原则，根据项目组的技术力量进行选择
4. **MyBatis开发DAO的方法**
   1. @reference MyBatis\_Demo\_HelloWorld2
   2. **SqlSession的使用范围**
      1. **SqlSessionFactoryBuilder**
         1. 通过SqlSessionFactoryBuilder创建会话工厂SqlSessionFactory
         2. 将SqlSessionFactoryBuilder当成一个工具类使用即可，不需要使用单例管理
         3. 在需要创建SqlSessionFactory的时候，只需要new一次该工具类即可
         4. 工具类中的“工具”（方法）最好写成静态方法（@version 有待经过合理的深究）
      2. **SqlSessionFactory**
         1. 通过SqlSessionFactory创建SqlSession，使用单例模式管理SqlSessionFactory（工厂一旦创建就一直使用一个实例）
      3. **SqlSession**
         1. 面向用户（程序员）的接口，提供用于操作数据库的方法（有很多）
         2. 线程不安全，在SqlSession实现类中除了有接口中的方法（操作数据库的方法）还有数据域属性同时有多个用户修改变量或者方法，导致产生不可预料的结果
         3. SqlSession最佳应用场合在方法体内（避免线程不安全，因为此时是局部变量，多个线程不能同时访问该变量）
   3. **具体开发DAO的方法**
      1. **原始Dao开发方法**
         1. **程序员需要写Dao接口和Dao类的实现**
         2. **思路**
            1. 整合Spring后，需要向Dao实现中注入SqlSessionFactory，在方法体内通过SqlSessionFactory创建SqlSession
         3. **实例**
         4. **原始DAO开发存在的问题**
            1. DAO接口的实现类中的方法中存在大量的模板方法（重复的方法，如sqlSession.commit()），增加了工作量
            2. 调用SqlSession方法的时候进行了Statement的id硬编码
            3. 由于SqlSession方法使用了泛型，调用SqlSession方法传入变量的时候即使变量类型传入错误，在编译阶段也不会报错，不利于程序员开发
      2. **Mapper代理方法（MyBatis特有的，建议使用）（写mapper.xml和mapper.java）**
         1. **程序员需要写Mapper接口，相当于Dao的接口**
         2. **思路（Mapper代理的开发规范）**
            1. 需要写Mapper接口
            2. 需要写mapper.xml映射文件
            3. MyBatis自动生成Mapper接口实现类的代理对象（前提是符合其开发规范）
            4. 规范（4个规则，其实就是利用反射机制，实现具体的功能）

在mapper.xml中的namespace要等于mapper接口（mapper.java）的地址（包+类）

mapper.java接口中的方法名和mapper.xml中的Statement的id一致

mapper.java接口中方法的输入参数类型和Statement中的parameterType指定的类型一致

mapper.java接口中返回值的类型和Statement中的resultType的类型一致

* + - 1. **实例**
         1. **mapper接口（UserMapper）**
         2. **mapper.xml(user.xml)**
         3. **全局配置文件**
         4. **测试方法**
      2. **注意**
         1. **代理对象内部调用selectOne和selectList的机制**

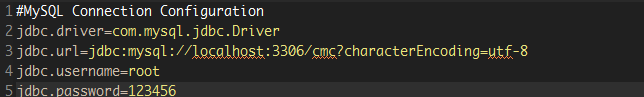
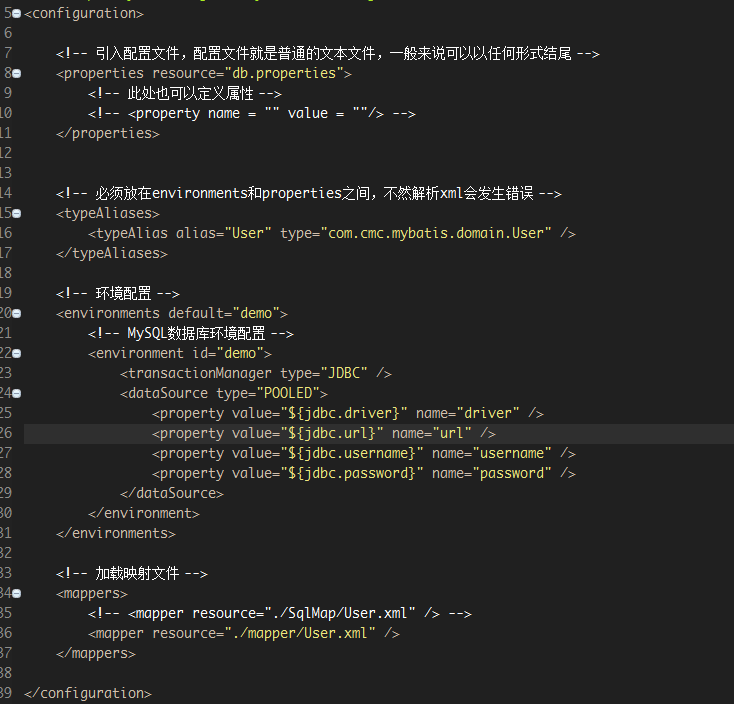
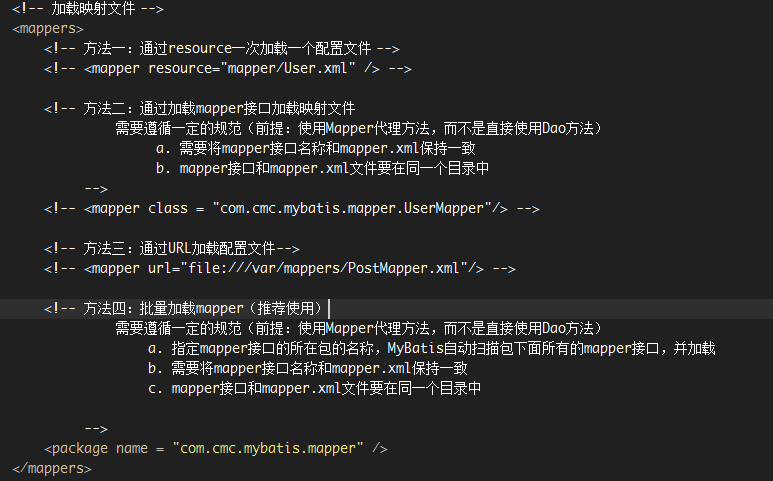
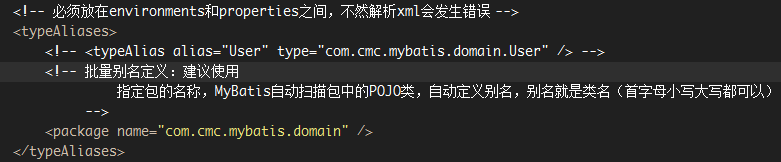
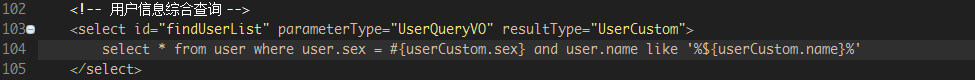
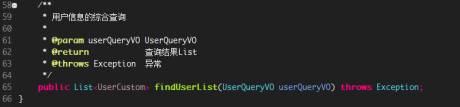
如果mapper方法返回的单个POJO对象，则代理对象内部会通过selectOne查询数据库

如果mapper方法返回的多个POJO对象，则代理对象内部会通过selectList查询数据库

* + - * 1. **mapper接口的方法参数只能有一个**

此时可以使用包装类型的POJO来满足不同的业务需求

持久层中方法的参数可以使用包装类型（map等），service方法中建议不要使用包装类型（不利于业务层的扩展）

1. **MyBatis全局配置文件解析**
   1. **主要内容**
      1. properties（属性）
      2. settings（全局配置参数）
      3. typeAliases（类型别名）
      4. typeHandles（类型处理器）
      5. objectFactory（对象工厂）
      6. plugins（插件）
      7. environments（环境集合属性）
         1. environment（环境子属性对象）
            1. transaction（事务管理）
            2. dataSource（数据源）
      8. mappers（映射器）
   2. @reference MyBatis\_Demo\_HelloWorld2
   3. **properties属性**
      1. 将一些特定属性的配置放在单独的properties文件中，只需要在MyBatis全局配置文件（以下值SqlMapConfig.xml文件）中加载即可，不需要杂糅在SqlMapConfig.xml文件中，造成相关属性在SqlMapConfig.xml文件中的硬编码
   4. **MyBatis加载属性顺序**
      1. 先加载<properties>标签体内的属性
      2. 然后读取<properties>标签中resource或者URL加载的属性，会覆盖已经读取的同名属性
      3. 最后读取parameterType传递的属性，会覆盖已经读取的同名属性
      4. 总结
         1. 优先级：parameterType > resource或者URL > 标签体内的属性
      5. 建议
         1. 不要再<properties>标签中不要提那家任何的属性值，只把相关属性定义在properties对应的文件中，而且在<properties>标签中的属性要有一定的特殊性，如：xxx.xxx，而不要使用xxx
      6. 实例
         1. db.properties
   5. **settings全局参数配置**
      1. **作用**
         1. @reference [http://mybatis.github.io/mybatis-3/zh/configuration.html#settings](http://mybatis.github.io/mybatis-3/zh/configuration.html" \l "settings)
         2. MyBatis框架运行的时候可以调整的一些运行参数，比如开启二级缓存，开启延迟加载
         3. 注意
            1. 如果参数设置不对，会影响到MyBatis框架的运行
   6. **mappers（映射配置）**
   7. **typeHandlers（类型处理器）**
      1. MyBatis中通过typeHandler完成JDBC和Java类型的转换
      2. 通常情况下，MyBatis所提供的类型处理器已经满足了平时的编程使用，不需要程序员进行自定义
   8. **typeAliases（别名） 重点**
      1. 针对parameterType和resultType指定的类型定义一些别名，方便开发，因为一些名称过长
      2. 例如：int默认是java.lang.Integer的别名
      3. 实例
   9. **输入映射**
      1. 通过parameterType制定输入参数的类型，类型可以是简单类型、HashMap、POJO类型
         1. 传递POJO的包装对象
            1. 需求：完成用户信息的综合查询，需要传入复杂的查询条件，例如：用户信息、商品信息等
            2. **解决方案：建议使用自定义包装类型的POJO，把复杂的查询条件包装进去，有利于系统的维护（系统架构师必知）**
      2. 实例
         1. Entity
         2. mapper.xml
            1. 在UserMapper.xml中定义用户信息的综合查询（查询比较复杂，一般都要进行关联查询，如：高级查询）
            2. 实例
         3. mapper.java
            1. 实例
         4. 测试方法
            1. 实例
   10. **输出映射**
       1. **resultType**
          1. @reference MyBatis\_Demo\_HelloWorld2
          2. 输出**POJO对象、POJO列表、简单数据**类型
          3. 使用resultType进行输出映射的时候，只有查询出来的列名和POJO中的属性名称一致，该列才可以映射成功
          4. 如果查询出来的列名和POJO中的属性名称全部不一样，则不会查询出POJO对象
          5. 只要查询出来的列名和POJO中的属性有一个一致，就会创建POJO对象，对象中一致的属性才能取到值，其他的都为默认值
          6. 注意
             1. 不管输出的是POJO单个对象还是POJO的对象列表，在mapper.xml中resultType指定的类型都是POJO，但是mapper.java中的返回类型就分别是POJO和List<POJO>，生成的动态代理对象是根据mapper方法的返回值的类型来确定具体调用selectOne（返回单个对象）还是selectList（返回集合对象）方法
       2. **resultMap（查询名称和属性名称不同的时候使用）**
          1. @reference MyBatis\_Demo\_HelloWorld2
          2. 完成高级以及复杂的输出结果映射
          3. 使用方法
             1. 如果查询出来的列名和POJO的属性名称不一致，则可以通过定义一个resultMap对列名和属性名称之间做一个映射

定义resultMap

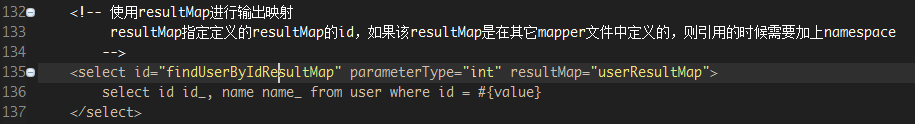
使用resultMap作为Statement的输出映射类型

* + - 1. 实例
         1. 使用User完成映射

select id id\_, name name\_ from user where id = #{value}

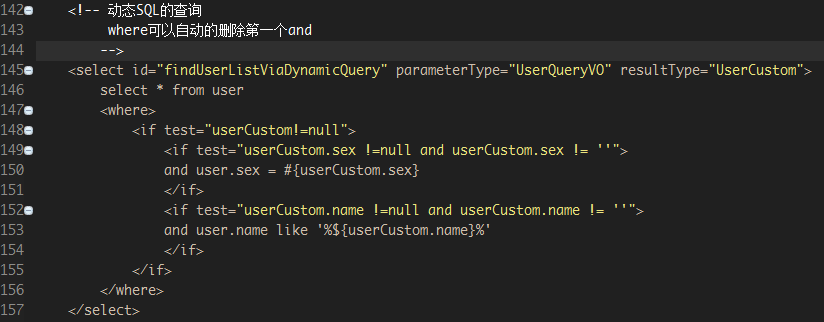
User类中属性名和查询语句中的属性名称不一致

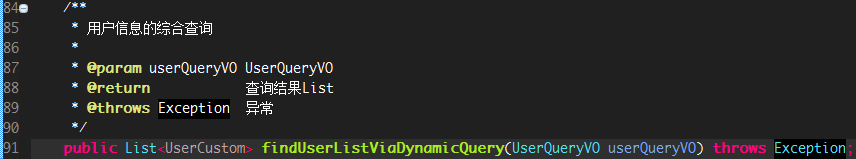
定义resultMap

使用resultMap作为Statement的输出映射类型

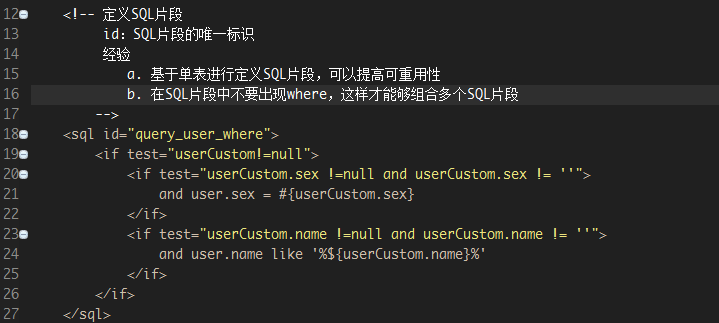
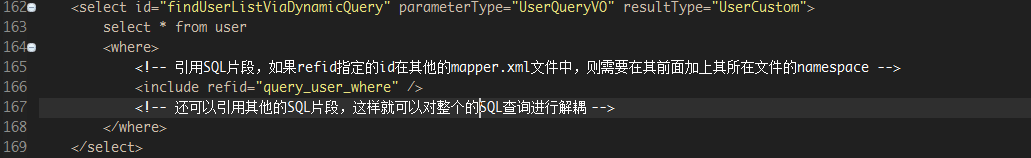
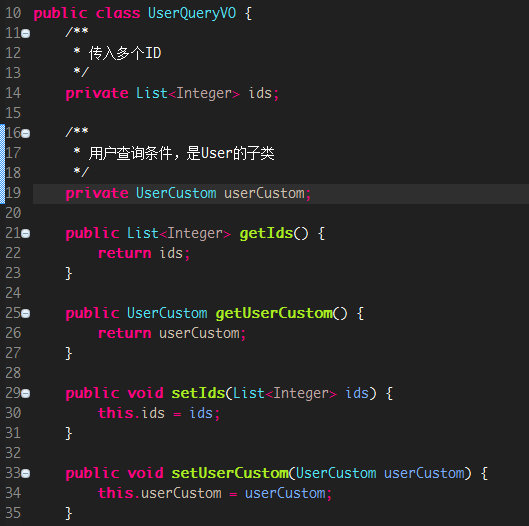
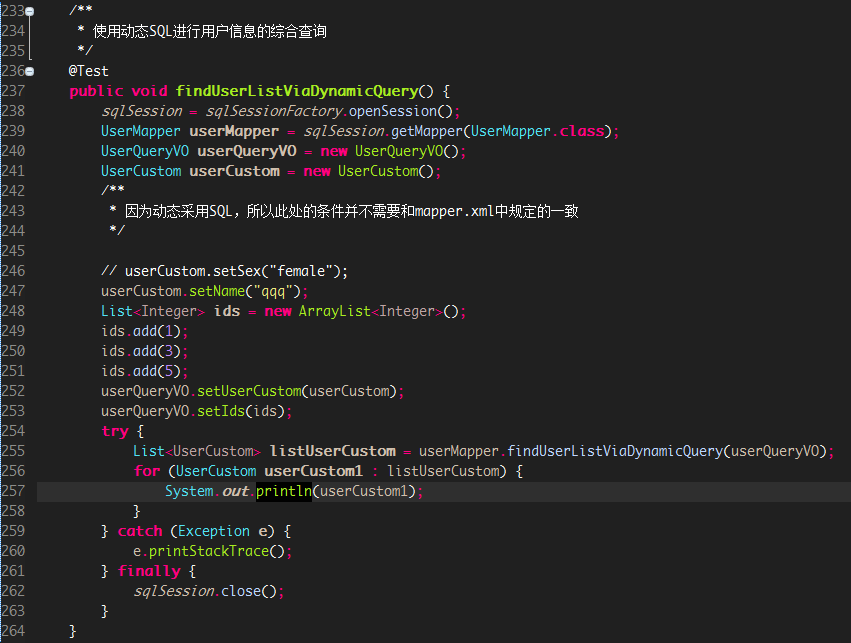
测试方法

* 1. **动态SQL（掌握）**
     1. **含义**
        1. MyBatis核心：通过表达式进行判断，对SQL进行灵活拼接、组装等操作
     2. **需求**
        1. **用户信息综合查询**和**用户信息查询列表总数**这两个Statement的定义使用动态SQL
        2. 具体步骤
           1. 对查询条件进行判断，如果输入的条件不为空，才进行表达式的拼接
           2. 实例

mapper.xml

mapper.java

测试方法

* 1. **SQL片段**
     1. 优点
        1. 分解复杂的SQL语句，方便程序的
     2. 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片段，其他的statement中就可以引用该公共的SQL片段
     3. 具体实现
        1. 定义SQL片段
        2. 引用SQL片段
  2. **<foreach>标签**
     1. **用途**
        1. 向SQL传递数组或者是List，MyBatis使用<foreach>标签进行解析
     2. **需求**
        1. 在用户查询列表和查询总数的Statement中增加多个id的输入查询
        2. SQL语句如下
           1. select \* from user where （id=1 or id=3 or id=5）或者select \* from user where id in(1, 3, 5)
     3. **具体实现**
        1. 在输入参数类型中要添加List<Integer>来存储多个ID
        2. 修改mapper.xml
           1. user where （id=1 or id=3 or id=5）在查询条件中，需要修改SQL片段
           2. 测试方法

**<!-- 高级部分 -🡪**

1. **高级映射**
   1. **对商品订单的数据模型进行分析**
      1. **数据模型分析思路**
         1. 每张表记录的数据内容
            1. 分模块对每张表的记录的内容进行熟悉，相当于学习系统需求（需求就是功能）的过程
         2. 每张表重要的字段信息
            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的时候一定要建立在某个业务意义基础之上，也就是说要从实际的业务出发，不能想当然
            2. 先分析数据库级别关联的表格所对应业务之间的关系
      2. **具体模型分析**
         1. **表格分析**
            1. **视图**

n

orders\_id(PK)

items\_id(PK)

订单明细表：orderdetail

user\_id(PK)

订单表：orders

* + - * 1. **具体信息**

1

商品表：items

用户表：user

**用户表：user**

记录了购买商品的用户信息

**订单表：orders**

记录了用户购买商品的订单

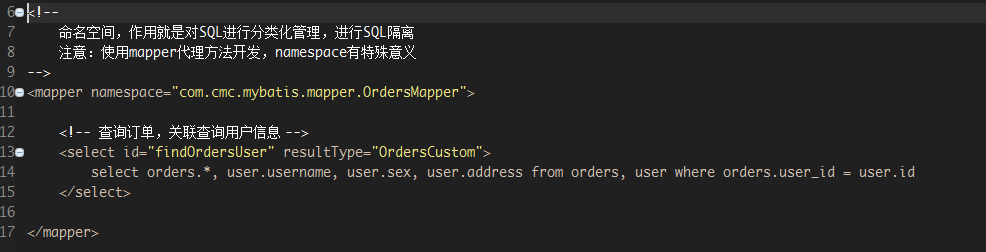
**订单明细表：orderdetail**

记录了订单的详细信息，即订单购买商品的信息

**商品表：items**

记录了用户购买商品的信息

* + - 1. **表与表之间的业务关系**
         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订，一对一（有问题，有待处理）
         3. orders – orderdetail：一个订单可以包含多个订单明细，因为一个订单可以购买多个商品，一对多关系
         4. 其他同理
  1. **查询**
     1. **OneToOne查询**
        1. **需求**
           1. 查询订单查询，关联查询创建订单的用户信息
        2. **具体实现**
           1. **resultType**

**mapper.xml**

**mapper.java**

**测试**

* + - * 1. **resultMap**

@reference MyBatis\_Demo

**SQL语句**

同resultType

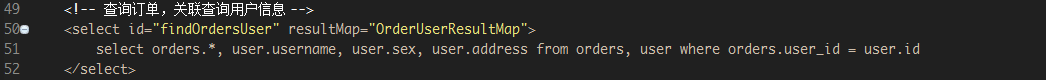
**思路**

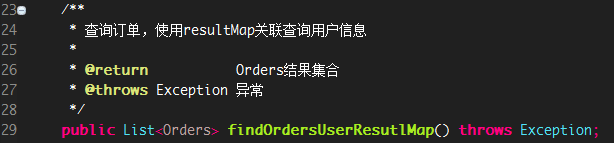
****使用resultMap将查询结果中的订单信息映射到Orders对象中，在Orders类中添加User属性，将关联查询出来的用户信息映射到Orders对象的user属性中

需要Orders类中添加user属性

**mapper.xml**

定义resultMap

statement

**mapper.java**

**测试**

* + - 1. **resultType和resultMap实现一对一查询总结**
         1. 使用resultType实现较为简单，如果POJO中没有包含查询出来的列名，只需要增加列名对象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. 需要单独定义resultMap，实现比较复杂，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到POJO的属性中
         3. 其他区别

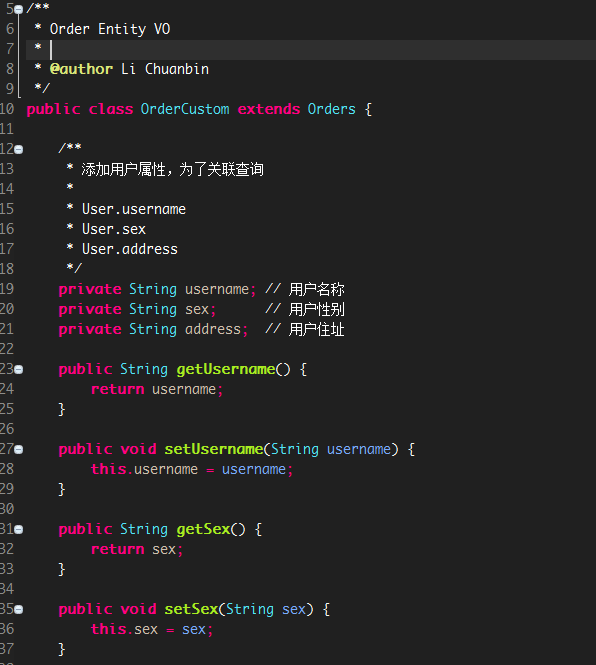
resultMap可以实现延迟加载，而resultType无法实现延迟加载

* + - 1. **如何写SQL语句**
         1. **确定查询的主表（orders表）**
         2. **确定查询的关联表（user表）**
         3. **使用外连接还是内连接**

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，所以可以使用内连接

* + - * 1. **具体SQL**

select orders.\*, user.username, user.sex, user.address from orders, user where orders.user\_id = user.id（内连接，有外键）

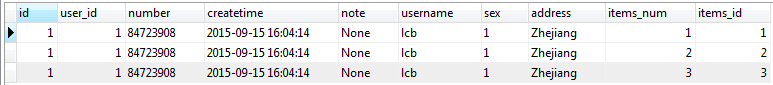
* + - 1. **创建POJO**
         1. 将SQL查询的结果集映射到POJO中，POJO中必须要包括所有的查询属性
         2. 原始的domain不能够映射全部字段，需要创建新的POJO
         3. 创建一个POJO，继承包括查询字段较多的domain类
    1. **OneToMany查询**
       1. **需求**
          1. 查询订单及订单明细
          2. 要求

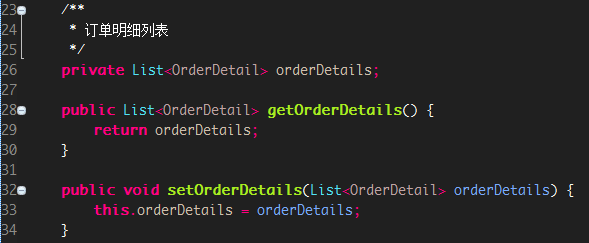
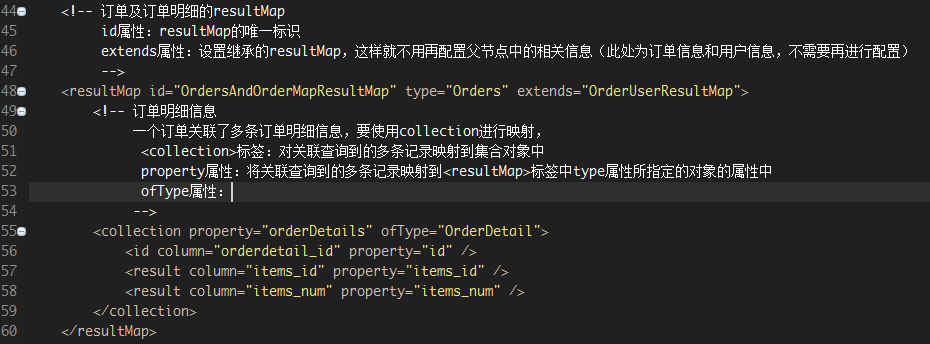
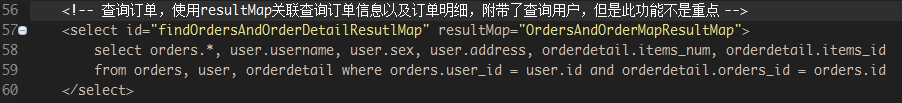
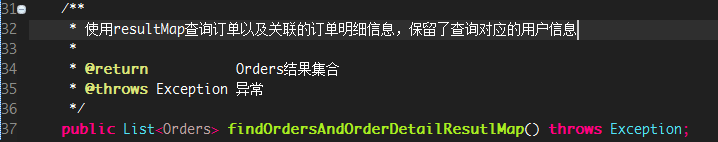
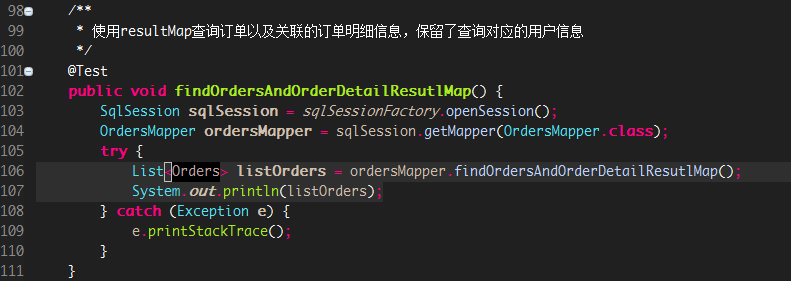
对orders映射不能出现重复记录（冗余的数据总归不好，要想办法消除掉）

* + - * 1. 具体做法

在Orders类中添加List<OrderDetail> orderDetails属性，最终会将订单信息映射到Orders的对象中，订单所对应的订单明细映射到Orders对象中的orderDetailis属性中，者此时orders表格中查询出来的记录就是一条（根据此数据库目前的数据）

* + - 1. **SQL语句**
         1. 确定主查询表：订单表
         2. 确定关联查询表：订单详情表
         3. 具体语句
         4. select orders.\*, user.username, user.sex, user.address, orderdetail.id orderdetail\_id, orderdetail.items\_num, orderdetail.items\_id from orders, user, orderdetail where orders.user\_id = user.id and orderdetail.orders\_id = orders.id
         5. 查询结果

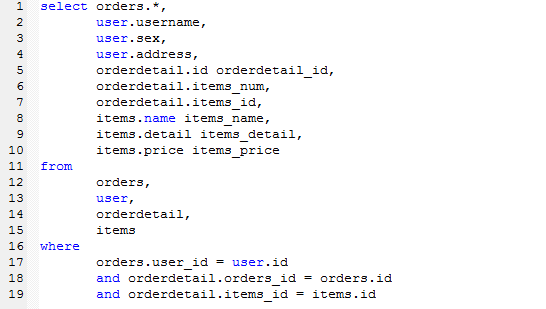
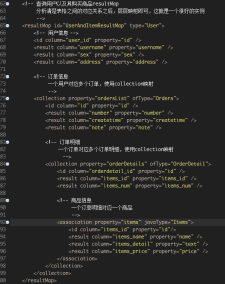


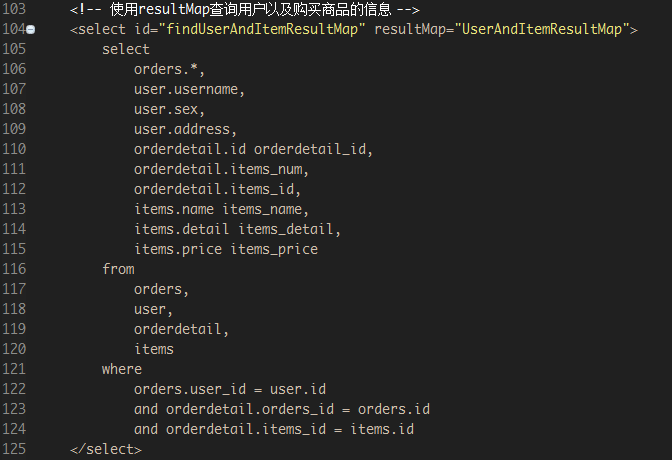
* + - 1. **在Orders类中添加订单明细属性**
      2. **mapper.xml**
         1. 定义resultMap
         2. Statement
      3. **mapper.java**
      4. **测试**
      5. **总结**
         1. MyBatis使用collection（resultMap中）对关联查询的多条记录映射到一个list集合属性中
         2. 若使用resultType，则需要开发者自己处理，使用双重循环遍历去掉重复记录，将订单明细放在OrderDetail对象中
    1. **ManyToMany查询**
       1. **需求**
          1. 查询用户以及用户购买商品信息
       2. **分析**
          1. 查询主表

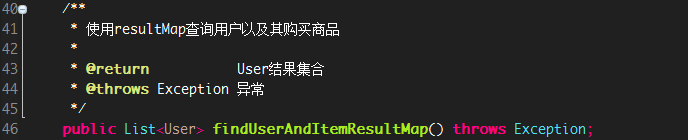
用户表（user）

* + - * 1. 关联表

商品表（items）、订单表（orders）、订单明细表（orderdetail）（但是用户表和商品表之间没有直接关联，是通过订单表和订单明细表）

* + - 1. **SQL语句**
      2. **映射思路**
         1. 将用户信息映射到user中
         2. 在User类中添加订单列表属性List<Orders> orderList，将用户创建的订单映射到orderList属性中
         3. 在Orders中添加订单明细列表属性List<OrderDetail> orderDetails，将订单明细映射到orderDetails属性
         4. 在OrderDetail中添加List<Item> item属性，将订单明细所对应的商品映射到items属性中
         5. 规律：根据关联管理（FK）一级一级地进行映射
      3. **mapper.xml**
         1. 定义resultMap
         2. Statement



* + - 1. **mapper.java**
      2. **测试**
      3. **总结**
         1. 将查询用户购买商品信息（注意此时查询的是商品信息，附带的是对应该商品的用户信息，而不是查询用户信息，附带商品信息）明细清单，包含用户名、用户地址、购买商品名称、购买商品时间、购买商品数量
         2. 针对上面的需求，将查询到的记录映射到扩展的POJO中，很简单的实现明细清单的功能
      4. **小结**
         1. 一对多是多对多的特例，如果有下面的需求：查询用户购买的商品信息，用户和商品的关系是多对多关系
         2. 需求一

查询字段

用户账号、用户名称、用户性别、商品名称、商品价格（最常见，企业开发中常见的明细列表，用户够阿米商品明细列表）

解决方案

使用resultTye将上边查询列映射到POJO（包含了上面列对应的属性）输出

* + - * 1. 需求二

查询字段

用户账号，用户名称、购买商品数量、购买商品明细（鼠标移撒上显示明细）

解决方案

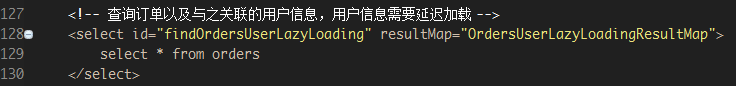
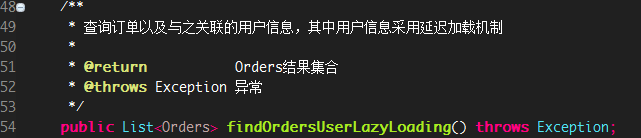
使用resultMap将用户购买的商品明细列表映射到User对象中

* + - 1. **resultType小结**
         1. 作用

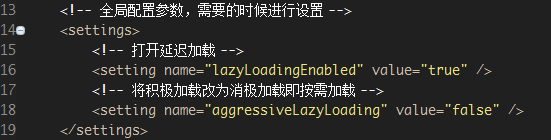
将查询结果按照SQL列名映射到和POJO对象同名的属性中

* + - * 1. 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的时候可以直接使用resultType将每一条记录映射到POJO中，在前端页面遍历POJO的集合（一般是List）即可

* + - 1. **ManyToMany**
         1. 使用resultMap是针对查询结果映射有特殊要求的功能，没有特殊功能（如把关联的结果映射为集合）的话可以直接使用resultType自定义POJO类进行实现，虽然这样部分记录的字段可能是有冗余的，但是并不影响实际运行，因为遵循了实际情况
    1. **resultType和resultMap总结**
       1. 有待整理
  1. **延迟加载（按需加载查询，减轻数据库的压力）**
     1. **背景**
        1. resultMap可以实现高级映射（使用association、collection实现OneToOne以及OneToMany映射）的延迟加载，因为association、collection具备延迟加载功能
     2. **需求**
        1. 查询订单并且关联查询用户信息，如果查询到订单信息即可满足要求，当我们需要查询用户信息的时候再查询用户信息，把对用户信息的按需查询就是延迟加载
     3. **概念**
        1. 先从单表查询，需要的时候进行关联表的查询，因为查询单表要比关联查询多张表格速度快很多，所以这样大大提高了数据库的性能，所以延迟加载能够提升系统的部分性能，进而提升整体系统的性能
     4. **使用association实现延迟加载**
        1. **需求**
           1. 先查询Orders的信息，需要的时候再查询User的信息
        2. **mapper.xml**
           1. **resultMap**
           2. **statement**
        3. **mapper.java**
        4. **测试**

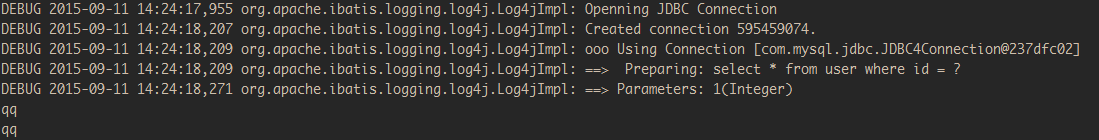
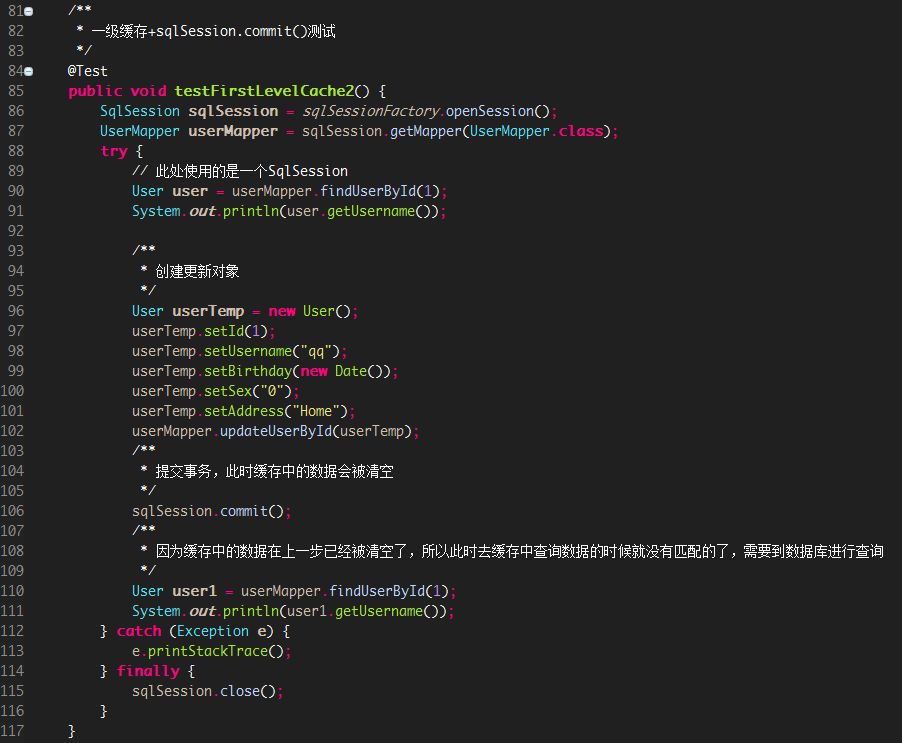
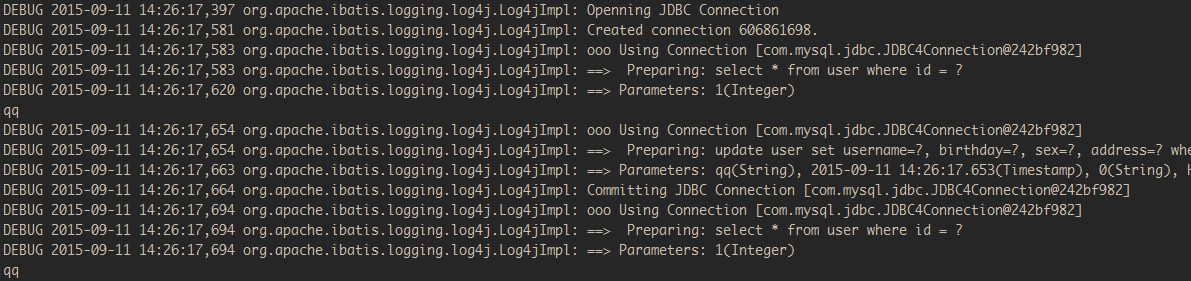
****

* + - 1. **全局配置文件中开启延迟加载**
    1. **延迟加载思考**
       1. 不使用MyBatis提供的association和collection中的延迟加载功能，如何实现延迟加载？
       2. 实现方法
          1. 定义两个mapper（代理）

查询订单列表

根据用户id查询用户信息

* + - 1. 实现思路
         1. 先去查询第一个mapper方法，获取订单信息列表
         2. 在程序中（service）按需调用第二个mapper方法去查询用户信息（参数是第一个mapper查询出来的id）
      2. 以上就是延迟加载的实质，没有框架也能够实现
      3. 总之，使用延迟加载方法，先去查询简单的SQL（不一定非要单表，但是最好单表），然后按需查询与之关联的其他表

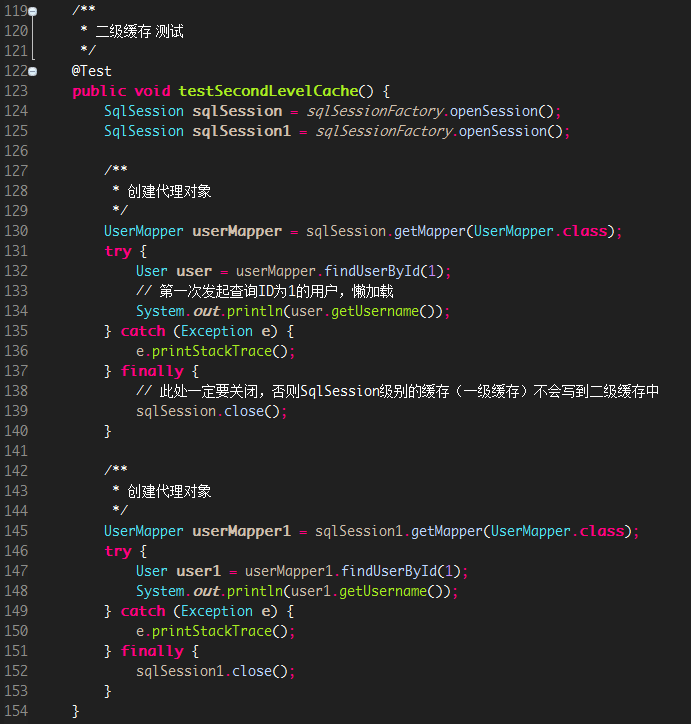
1. **查询缓存**
   1. 用途
      1. 用于减轻数据库压力，提高数据库性能
   2. MyBatis提供一级缓存和二级缓存功能
   3. **一级缓存（SqlSession级别的缓存）**
      1. 在操作数据库的时候需要构造SqlSession对象，在对象中有一个数据结构（HashMap）用于存储缓存数据，不同的SqlSession对象之间的缓存数据区域是互相不影响的，即SqlSession是一级缓存，其数据只在自己的缓存区域中，和其他SqlSession不共用
      2. 测试
         1. MyBatis默认支持一级缓存
         2. 测试方法（没有进行commit()）
            1. 具体方法
            2. 结果
         3. 测试方法（进行了commit()）
            1. 具体方法
            2. 结果
      3. 注意
         1. 可以debug调试一下查看具体的调用过程，实际上对应于每个操作都有一个标识符的，首先要比较标识符，一样的话就代表缓存中存在此类查询，否则没有，具体可以参考debug过程；
         2. SQLSession对象关闭之后，其缓存就会自动清空，此时有相同的查询也没有对应的缓存。
   4. **二级缓存（了解MyBatis二级缓存的使用场景）**
      1. 概念
         1. mapper（namespace）级别的缓存，多个SqlSession去操作同一个Mapper的SQL语句，其数据会缓存在二级缓存中，即同一个namespace的mapper是二级缓存，这样多个SqlSession之间共用数据
      2. 二级缓存实例
         1. 开启二级缓存
            1. 需要同时在MyBatis的全局配置文件和mapper.xml文件中进行配置

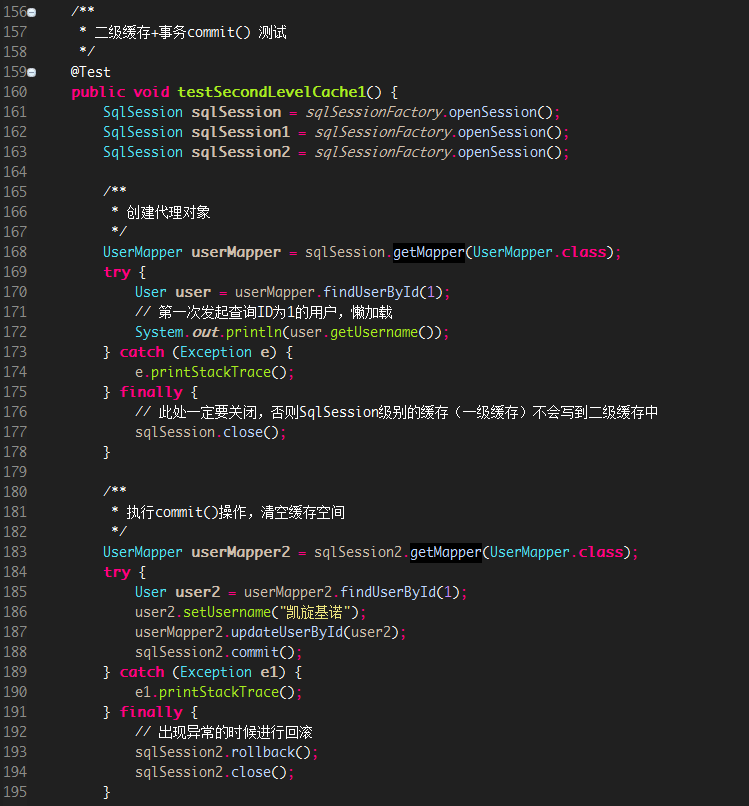
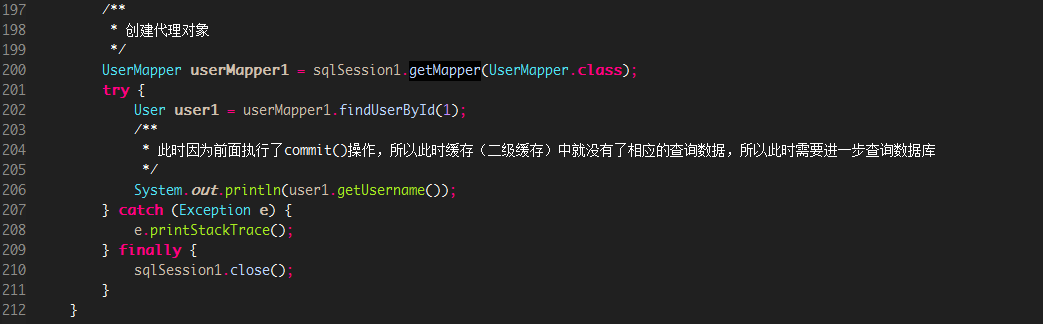
全局配置文件

mapper.xml

* + - 1. 调用POJO类实现序列化接口
         1. 目的

因为数据不一定存储在内存中，也有可能存储在硬盘上，所以为了将缓存数据取出执行反序列化操作，需要实现序列化接口，存储介质多种多样，需要实现序列化，这样无论存放在哪里以后直接反序列化操作就可以取回数据

* + 1. 测试



* + 1. 注意
       1. 如果MyBatis打开了二级缓存，则查询结果不仅仅会存放在默认打开的一级缓存中，二级会存在二级缓存中
       2. 多个SqlSession可以共享一个二级缓存（mapper定义的namespace）
       3. 每一个namespace的mapper有一个二级缓存区域（如果两个mapper的namespace相同，则共用一个二级缓存区域）
       4. 同一级缓存一样，如果mapper（namespace）中调用了commit()方法，则会清空该二级缓存
  1. **使用缓存的作用**
     1. 如果缓存中有满足查询条件的查询结果，则就不需要去查询数据库而可以直接查询缓存（先检查缓存中有没有对应的数据，有则使用，无则查询数据库），这样速度快、减轻数据库压力
  2. **注意**
     1. 第一次发起查询的时候，澶寻结果会自动保存在缓存中，第二次查询（同样的查询）的时候如果缓存中有则会直接使用缓存中的数据，如果没有的话会直接查询数据库。此时“没有”出现的有两种情况，一是查询的语句是第一次查询，而是SqlSession对象调用了commit()操作（可能是执行了插入、更新、删除操作），这样的话就会清空（flush）SqlSession中的缓存（一级缓存），此时就不能直接从一级缓存中取数据了，这样做的目的是让缓存中存储的信息是最新的，避免脏读
     2. MyBatis支持事务机制，而且默认是打开的，所以SQL的插入、更新、删除操需要进行commit操作（先flush缓存再同步到数据库中），失败的话就rollback（遵循事务的原子性）
     3. sqlSession如果不close()的话是不会写到二级缓存当中的，其数据只会保存在一级缓存当中
     4. 禁用二级缓存
        1. statement中使用useCache=”false”可以禁用当前select语句的二级缓存，若不进行设置默认情况下SQL使用二级缓存
        2. 示例
        3. 场景
           1. 对每次查询都需要最新数据的statement最好设置禁用二级缓存
     5. 刷新二级缓存（清空二级缓存）
        1. 在与mapper的同一namespace中，如果有其他CUD操作数据后没有执行刷新（commit操作）则会出现脏读的情况。
        2. 解决方案
           1. 设置statement中的属性flushCache的值为true即可。
        3. 注意
           1. 默认情况下statement的flushCache属性的值为true；
           2. 使用缓存的时候如果手动修改了数据库与缓存对应的数据的值会出现脏读。
     6. MyBatis Cache参数
        1. flushInterval
           1. 含义

刷新间隔

* + - 1. ReadOnly
         1. Faf
    1. MyBatis整合EhCache
       1. 分布式缓存
          1. 为了提高系统的并发、性能，一般对系统进行分布式部署（集群部署方式）。
          2. 场景

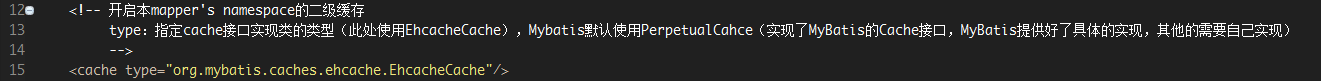
在分布式系统当中，当用户访问其中一台服务器，此时数据只保存到该服务器上（不使用分布式缓存），此时要使用分布式缓存把用户信息同步所有的服务器中，进行集中管理。

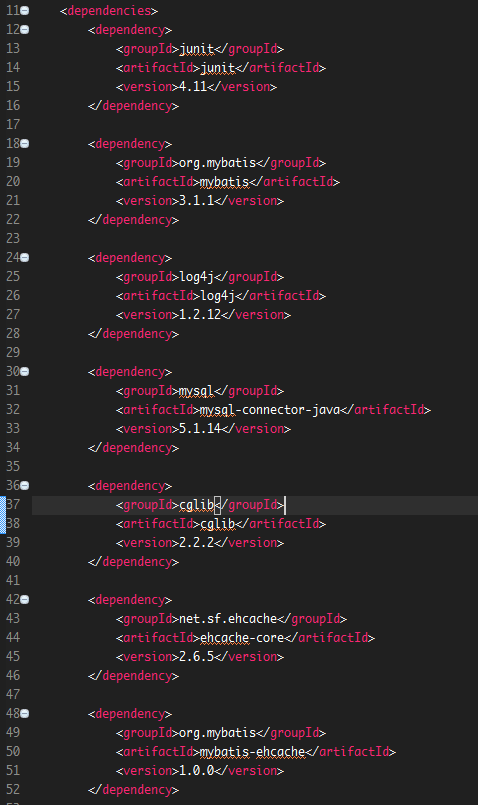
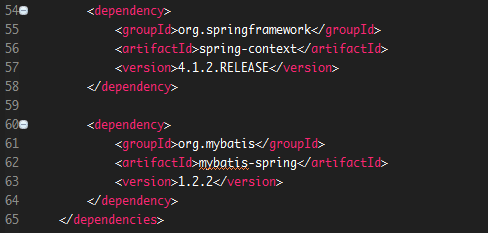
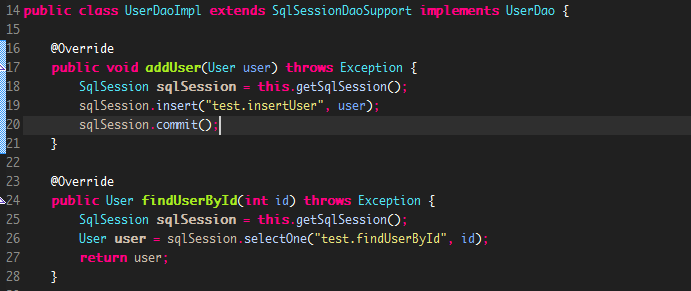
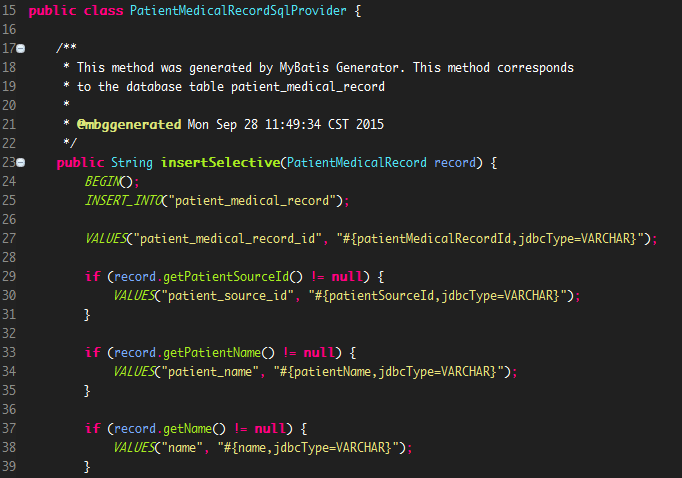
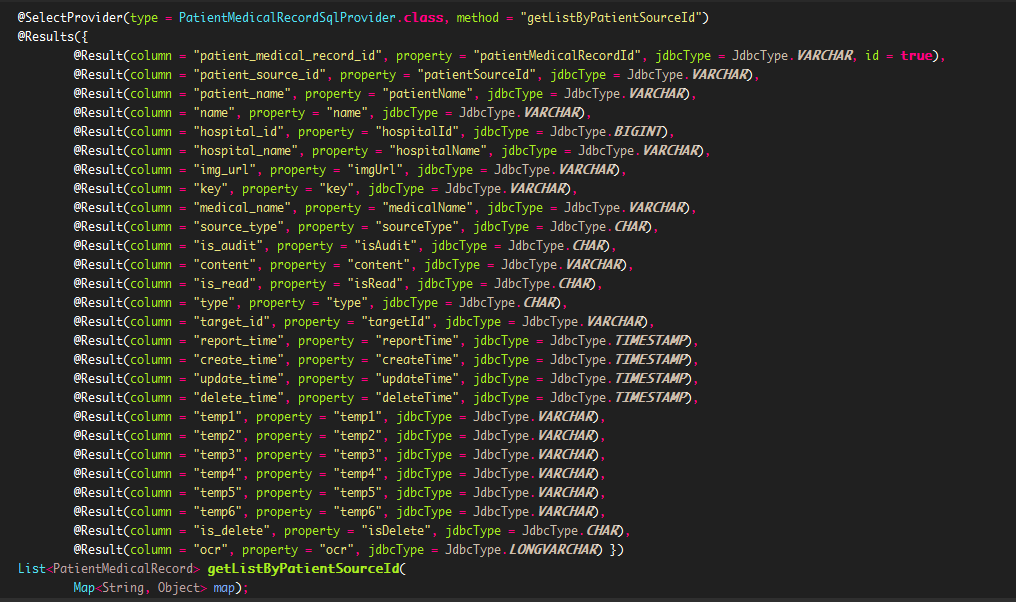
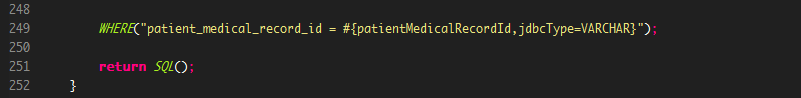
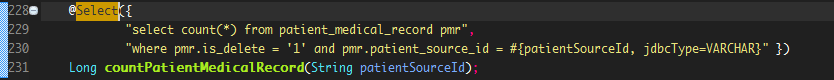
* + - * 1. 常见分布式缓存框架

redis

memcached

ehcache

* + - 1. EhCache
         1. 一种分布式缓存框架
      2. 注意
         1. MyBatis不支持分布式缓存，所以需要和分布式缓存框架整合；
         2. MyBatis默认实现的Cache类是PerpetualCache。
      3. 整合方法
         1. MyBatis提供了一个Cache接口，如果要实现自己的缓存，实现Cache接口即可
         2. 加入对应MyBatis的EhCache开发包（Maven）
      4. 实例（已经在pom文件中配置好了相关的开发支持包）
    1. 二级缓存局限性
       1. MyBatis二级缓存对细粒度的数据级别的缓存实现不足。细粒度缓存，如查询最新的商品信息，其他商品信息不动。

1. **MyBaits和Spring整合（掌握）** @version D20151025
   1. **整合具体内容**
      1. 需要Spring通过单例方式(@question ？)管理SqlSessionFactory；
      2. Spring和MyBatis整合生成代理对象，使用SQLSessio0nFactory创建SQLSession对象；
      3. 持久层的Mapper都需要Spring进行管理。
   2. **整合环境**
      1. Maven pom.xml
   3. **和Spring整合之后原始Dao开发**
      1. 通过Spring注入sqlSessionFactory
      2. SqlSessionDaoSupport中含有通过sqlSessionFactory设置sqlSession的方法，所以可以直接getSqlSession()
   4. **和Spring整合Mapper开发**
2. **逆向工程**
3. **annotation版本** @version D20151130
   1. **select**
      1. 使用provider
      2. 不使用provider
      3. 注意
         1. provider中可以使用MyBatis提供的关键字进行sql组合，也可以直接使用StringBuffer进行append（此时因为是人工组合的，所以要注意sql语句段之间要有空格，否则sql语法就错了。而如果是MyBatis负责组合的话就没有“空格”这个问题了），本质：组合sql字符串，也就是说方法的返回值都是String类型；
         2. 不使用provider的时候可以使用@Select方法进行字符串的拼接（字符数组拼接，此时因为是MyBatis自动组合sql字符串，所以不需要考虑“空格”问题），**建议使用provider，这样更加符合“单一职责原则”；**
         3. @Param注解，如果方法里面的参数在**两个以及两个以下**则不需要使用该注解，可以直接通过#{参数名称}或者#{参数位置，例如1（第一个参数与）}，**三个以及三个以上**则需要@Param注解参数的名称，**建议都使用@Param进行参数的注解**。且MyBatis会把方法的参数（无论是否使用@Param注解）的值存放在Map<String, Object>中，可以在Provider直接通过Map<String, Object>对象进行引用相关参数；
         4. Provider的方法中只能接收一个参数，如果mapper方法里面有多个参数则使用@Param标注，然后在provider中通过参数Map<String, Object> map进行引用，**不能接收多个bean作为参数（注意：一般面向对象编程参数的类型也一定和返回类型一致，所以就不存在参数为多个bean的情况）**
         5. result映射关系
            1. 上面的映射关系是比较全的了，但是上面完全**可以省略javaType和jdbcType属性**
         6. 在进行CRUD操作的时候，如果传递的是一个实体对象，则SQL完全可以直接引用实体对象的属性
         7. **select在数据库没有对应记录的时候，如果返回的是对象，则结果是null；如果返回的是List<?>，则结果是[]（内容为空，而不是null，亦即空指针，原理：这个需要看MyBatis的源码）**
         8. Provider中方法参数映射问题
            1. 视图
            2. 补充

单个未加@Param注解的参数不可以直接使用Map<String, Object>，直接把参数映射就行。如String id、Student

* 1. **insert**
     1. 同select
  2. **update**
     1. 同select
  3. **delete**
     1. 同select