# 1 MyBatis简介

## 1.1 MyBatis

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。

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。

对jdbc的封装框架有哪些：Hibernate,dbutils,jdbcTemplate[spring]，mybatis

原理：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。

## 1.2 jdbc程序代码

|  |
| --- |
|  |

**存在的问题**

* 数据库连接频繁开启和关闭，会严重影响数据库的性能。
* 代码中存在硬编码，分别是数据库部分的硬编码和SQL执行部分的硬编码。

## 1.3 MyBatis的框架核心

|  |
| --- |
| 1. mybatis配置文件，包括**Mybatis全局配置文件和Mybatis映射文件**，其中全局配置文件配置了数据源、事务等信息；映射文件配置了SQL执行相关的 信息。 2. mybatis通过读取配置文件信息（全局配置文件和映射文件），构造出**SqlSessionFactory，**即会话工厂。 3. 通过SqlSessionFactory，可以创建**SqlSession**即会话。Mybatis是通过SqlSession来操作数据库的。 4. SqlSession本身不能直接操作数据库，它是通过底层的**Executor**执行器接口来操作数据库的。Executor接口有两个实现类，一个是普通执行器，一个是**缓存执行器（默认）**。 5. Executor执行器要处理的SQL信息是封装到一个底层对象**MappedStatement**中。该对象包括：SQL语句、输入参数映射信息、输出结果集映射信息。其中输入参数和输出结果的映射类型包括**HashMap集合对象、POJO对象类型**。 |

# Mybatis入门

## 2.1 环境准备

### 下载MyBatis

mybaits的代码由github.com管理，下载地址：<https://github.com/mybatis/mybatis-3/releases>

### 创建项目导包

导入下面的包

|  |
| --- |
|  |
|  |
| Mybatis，mysql，junit，log4j |

### 添加log4j.properties

主要目的：让mybatis也能显示sql语句

Mybatis使用的日志包是log4j的，所以需要添加log4j.properties。

在classpath下创建log4j.properties如下：【文件内容可以从mybatis-3.2.7.pdf中拷贝】

# Global logging configuration

log4j.rootLogger=DEBUG, stdout

# Console output...

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

# log4j.appender.stdout.layout=org.apache.log4j.TTCCLayout

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

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

日志级别在开发阶段设置成DEBUG，在生产阶段设置成INFO或者ERROR。

## 2.2 开发步骤

|  |
| --- |
| 1. 创建PO（model）类，根据需求创建； 2. 创建全局配置文件SqlMapConfig.xml； 3. 编写映射文件； 4. 加载映射文件，在SqlMapConfig.xml中进行加载； 5. 编写测试程序，即编写Java代码，连接并操作数据库。   思路：   1. 读取配置文件； 2. 通过SqlSessionFactoryBuilder创建SqlSessionFactory会话工厂。 3. 通过SqlSessionFactory创建SqlSession。 4. 调用SqlSession的操作数据库方法。 5. 关闭SqlSession。 |

### 创建PO类

|  |
| --- |
|  |

### 创建SqlMapConfig.xml

建议使用：mybatis-config.xml

在classpath(src)下，创建SqlMapConfig.xml文件【SqlMapConfig.xml（文件头可以从mybatis-3.2.7.pdf文档的2.1.2小节中拷贝）】

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>  <!-- 配置mybatis的环境信息 -->  <environments default=*"development"*>  <environment id=*"development"*>  <!-- 配置JDBC事务控制，由mybatis进行管理 -->  <transactionManager type=*"JDBC"*></transactionManager>  <!-- 配置数据源，采用dbcp连接池 -->  <dataSource type=*"POOLED"*>  <property name=*"driver"* value=*"com.mysql.jdbc.Driver"*/>  <property name=*"url"* value=*"jdbc:mysql://localhost:3306/mybatis?useUnicode=true&amp;characterEncoding=utf8"*/>  <property name=*"username"* value=*"root"*/>  <property name=*"password"* value=*"123456"*/>  </dataSource>  </environment>  </environments>  </configuration> |
|  |

### 映射文件

命名为：UserMapper.xml

在classpath下，创建sqlmap文件夹。在sqlmap目录下，创建User.xml映射文件。

**【Mybatis的映射文件头（可以从mybatis-3.2.7.pdf文件中拷贝）】**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!--  namespace：命名空间，它的作用就是对SQL进行分类化管理，可以理解为SQL隔离  mapper元素的namespace本来是可以随便写的， 但是官方文档建议我们写成当前配置文件的包路径（用“.”隔开）  注意：使用mapper代理开发时，namespace有特殊且重要的作用  -->  <mapper namespace=*"test"*>  <!--  [id]：statement的id，要求在命名空间内唯一  [parameterType]：入参的java类型  [resultType]：查询出的单条结果集对应的java类型  [#{}]： 表示一个占位符?  [#{id}]：表示该占位符待接收参数的名称为id。注意：如果参数为简单类型时，#{}里面的参数名称可以是任意定义  -->  <!--  注意：  1. 只要是查询语句，就一定有返回的结果集，那么就必须配置resultType指定使用什么类来封装结果集中的行！  2.resultType="User"并不能决定当前的查询语句，就会把查出的所有结果都存入一个User对象中！  3.当前查询语句所查出的结果，究竟会被封装到List中，还是一个User对象中，由调用的SqlSession的方法决定  如果调用的是 selectList，则把结果存入List中  如果调用的是 selectOne，则把结果存入一个User中  -->  <select id=*"findUserById"* parameterType=*"int"* resultType=*"com.gyf.domain.User"*>  SELECT \* FROM USER WHERE id = #{id}  </select>  </mapper> |

### 配置文件加载映射文件

|  |
| --- |
|  |
| <!-- 注意，路径中使用的是"/"隔开目录，而不是"." --> |

### 测试类

|  |
| --- |
|  |
| public void testSave() throws Exception {  // 加载mybatis主配置文件  // 从App.class字节码所在路径开始寻找mybatis-config.xml  InputStream in = App.class.getResourceAsStream("mybatis-config.xml");  // 创建SqlSessionFactory，SqlSessionFactory是专门用来生成 SqlSession的  SqlSessionFactory sf = new SqlSessionFactoryBuilder().build(in);  // 获取一个会话，这里的会话，是从连接数据库开始， 到关闭了数据库连接为止。 每一个SqlSession对象中都封装了一个Connection对象  // 注意，在获取会话的同时，事务已经开启了，这一点和hibernatw不一样！  SqlSession s = sf.openSession();    // 以下的aa.save，表示  /\*  aa用来查找某一个具体的子配置文件， 要与XXXMapper.xml中的mapper元素的namespace一致  save要与子配置文件中的语句的id一致  \*/  User user = new User();  user.setName("张无极");  user.setBirthday(new Date());  user.setMoney(333d);    s.insert("com.woniuxy.a\_hello.UserMapper.save", user);  // 也可以不加命名空间  s.insert("save", user);  // 提交事务，虽然事务是自动开启的，但是提交，还必须手动提交。  s.commit();  // 关闭会话，表面上是关闭mybatis的会话，实际上底层是关闭了数据库连接！  s.close();  } |

## 2.3 更多案例讲解

### 模糊查询用户信息

|  |
| --- |
| <!--  [${}]：表示拼接SQL字符串  [${value}]：表示要拼接的是简单类型参数。  注意：  1、如果参数为简单类型时，${}里面的参数名称必须为value  2、${}会引起SQL注入，一般情况下不推荐使用。但是有些场景必须使用${}，比如order by ${colname}  -->  <select id=*"findUserByName"* parameterType=*"String"* resultType=*"com.gyf.domain.User"*>  SELECT \* FROM USER WHERE username like '%${value}%'  </select> |
|  |

### 插入用户信息

|  |
| --- |
|  |
|  |

### 删除用户

|  |
| --- |
|  |
|  |
| Session.selete(“deleteUser”,27)，也可以运行。所以说，命名空间可以不加，建议加 |

### 更新用户

|  |
| --- |
|  |
|  |

### 主键返回之MySQL自增主键

意思是：模型没有id值，通过这种方法，可以将id值返回，赋值给模型id

* 先执行sql，再执行selectKey，将最后插入的主键值返回

思路：

* MySQL自增主键，是指在insert之前MySQL会自动生成一个自增的主键。
* 我们可以通过MySQL的函数获取到刚插入的自增主键:

LAST\_INSERT\_ID()

* 这个函数是在insert语句之后去调用。

|  |
| --- |
| <insert id=*"insertUser"* parameterType=*"com.gyf.domain.User"*>  <!--  [selectKey标签]：通过select查询来生成主键  [keyProperty]：指定存放生成主键的属性  [resultType]：生成主键所对应的Java类型  [order]：指定该查询主键SQL语句的执行顺序，相对于insert语句  [last\_insert\_id]：MySQL的函数，要配合insert语句一起使用 -->  <selectKey keyProperty=*"id"* resultType=*"int"* order=*"AFTER"*>  SELECT LAST\_INSERT\_ID()  </selectKey>  <!-- 如果主键的值是通过MySQL自增机制生成的，那么我们此处不需要再显示的给ID赋值 -->  INSERT INTO USER (username,sex,birthday,address)  VALUES(#{username},#{sex},#{birthday},#{address})  </insert> |

### 主键返回之MySQL自增UUID

* 先生成uuid，在执行sql，然后返回主键值

|  |
| --- |
| <insert id=*"insertUser"* parameterType=*"com.gyf.domain.User"*>  <selectKey keyProperty=*"id"* resultType=*"String"* order=*"BEFORE"*>  SELECT UUID()  </selectKey>  INSERT INTO USER (username,sex,birthday,address)  VALUES(#{username},#{sex},#{birthday},#{address})  </insert> |
| <insert id=*"save"* parameterType=*"User"*>  <selectKey keyProperty=*"id"* resultType=*"java.lang.String"* order=*"BEFORE"*>  SELECT REPLACE(UUID(),'-','@')  </selectKey>  insert into user2 values(#{id}, #{name}, #{birthday}, #{money})  </insert> |
| 会先执行上面的语句，把查到的id值保存。下面的#{id},会取上面的id值，不再是user的 |

### ORCLE主键

SELECT user\_seq.nextval() FROM dual

## 2.4 细节

### parameterType和resultType

* parameterType指定输入参数的java类型，可以填写别名或Java类的全限定名。
* resultType指定输出结果的java类型，可以填写别名或Java类的全限定名。

|  |
| --- |
| parameterType：  <!--  以下insert语句，确实是需要参数的，是需要写出parameterType="com.woniuxy.b\_parametertype.User"  但是，mybatis允许我们省略 parameterType， 此时必须保证，s.insert()所传入的对象，拥有#{}中所描述的属性。  再测试，我们传入一个不是User类型的对象，只要这个而对象拥有#{}中所描述的属性即可。 经过测试，这样是可以的！！  再测试，我们传入一个Map，保证这个map的键值与#{}所描述的属性一致。 经过测试，这样也是可以的！！  个人建议把parameterType明显地写出来！ 以提高程序可读性！！  -->  <insert id="save" parameterType="User">  insert into user values(null, #{name}, #{birthday}, #{money})  </insert> |

### **#{} 和 ${}** 的区别

|  |
| --- |
| #{}：相当于预处理中的占位符？。  #{}里面的参数表示接收java输入参数的名称。  #{}可以接受HashMap、POJO类型的参数。  当接受简单类型的参数时，#{}里面可以是value，也可以是其他。  #{}可以防止SQL注入。  ${}：相当于拼接SQL串，对传入的值不做任何解释的原样输出。  ${}会引起SQL注入，所以要谨慎使用。  ${}可以接受HashMap、POJO类型的参数。  当接受简单类型（基本类型+String）的参数时，${}里面只能是value。 |
| #{} 和 ${} 的区别  1. #{} 会在参数前后添加引号  ${} 不会在参数前后添加引号  2. ${} 底层使用的是jdbc中的Statement，没有使用占位符（？）  #{} 底层使用的是jdbc中的PreparedStatement，有使用占位符（？）  3. ${}有sql注入漏洞的危险，#{}没有！！    什么时候使用${} : 填装 表名 和 列名时 ，应该使用${}  什么时候使用#{} : 填装 表中的值的时候，应该使用#{} |

### selectOne和selectList

* selectOne：只能查询0或1条记录，大于1条记录的话，会报错：
* selectList：可以查询0或N条记录

### 模型的属性名，与表中的列名不一致

解决方法：

|  |
| --- |
| 方法1：  <select id="findAll" resultType="User">  select  id as id2,  name as name2,  birthday as birthday2,  money as money2  from user  </select> |
| 方法2：推荐方式！ 因为这种方式的重用性更高！！  <select id="findAll2" resultMap="rm" >  Select \* from user  </select>  <resultMap type="User" id="rm">  <id property="id2" column="id"></id>  <result property="name2" column="name"/>  <result property="birthday2" column="birthday"/>  <result property="money2" column="money"/>  </resultMap> |

## 2.5 MyBatis的Dao编写 【一般不用，有更多好方式】

### dao

|  |
| --- |
|  |
|  |

### 测试

|  |
| --- |
|  |

## 2.6 MyBatis的Dao编写【mapper代理方式实现】

Mapper代理的开发方式，程序员只需要编写mapper接口（相当于dao接口）即可。Mybatis会自动的为mapper接口生成动态代理实现类。

**不过要实现mapper代理的开发方式，需要遵循一些开发规范。**

### 开发规范

|  |
| --- |
| 1. mapper接口的全限定名要和mapper映射文件的namespace的值相同。 2. mapper接口的方法名称要和mapper映射文件中的statement的id相同； 3. mapper接口的方法参数只能有一个，且类型要和mapper映射文件中statement的parameterType的值保持一致。 4. mapper接口的返回值类型要和mapper映射文件中statement的resultType值或resultMap中的type值保持一致；   通过规范式的开发mapper接口，可以解决原始dao开发当中存在的问题：   * 模板代码已经去掉； * 剩下去不掉的操作数据库的代码，其实就是一行代码。这行代码中硬编码的部分，通过第一和第二个规范就可以解决。 |
| 使用Mybatis自动生成dao层， 有以下几个必须：  1. 必须创建一个接口，该接口的名字必须与子配置文件UserMapper.xml同名（但是没有xml）  2. 以前我们说UserMapper.xml子配置文件中的mapper元素的namespace是可以随便写的，  但是建议叫做UserMapper的包路径  现在，namespace就必须写成UserMapper的包路径了！ 也就是当前接口的包路径。  3. 该接口中的方法名，必须与UserMapper.xml子配置文件中的语句的id相同  该接口中的方法的参数，必须与UserMapper.xml子配置文件中的语句的parameterType相同  该接口中的方法的返回值，必须与UserMapper.xml子配置文件中的语句的resultType相匹配（不是相同）  4. 该接口中的任何方法，禁止使用方法重载！！ 你懂得！！ |

### 编写步骤

### 第一步：【】

编写UserMapper接口和定义mapper映射文件UserMapper.xml（**注意namespace的值**），放到mapper包中。

|  |
| --- |
|  |

### 第二步：【添加映射配置文件】

|  |
| --- |
|  |

### 第三步：测试

|  |
| --- |
|  |

## 2.7 抽取MybatisUtils

|  |
| --- |
| public class MybatisUtils {  private static SqlSessionFactory sf;    static {  InputStream in = MybatisUtils.class.getResourceAsStream("mybatis-config.xml");  sf = new SqlSessionFactoryBuilder().build(in);  }  public static SqlSession getSqlSession() {  return sf.openSession(true);  }  } |

# 3 全局配置文件其它配置

## 3.1 properties数据库文件配置

### 在src下配置个db.properties文件

|  |
| --- |
|  |

### 修改全局的配置文件

|  |
| --- |
|  |

## 3.2 setting设置

|  |
| --- |
|  |

### 具体配置详解

|  |
| --- |
|  |

## 3.3 typeAliases

别名是使用是为了在映射文件中，更方便的去指定参数和结果集的类型，不再用写很长的一段全限定名。

### mybatis支持的别名

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

上面这些都称作简单类型。

### 自定义别名

|  |
| --- |
|  |
| <!--  针对于com.cheney.pojo包下的类，在mybatis子配置文件中都可以使用简单类名（没有包名） 别名为类名，首字母大小写都可以-->  <typeAliases>  <package name="com.cheney.pojo"/>  </typeAliases> |

## mappers

### <mapper resource=’’/>

使用相对于类路径的资源

如：<mapper resource="com/cheney/pojo/UserMapper.xml"/>

### <mapper url=’’/> 【不用】

使用完全限定路径

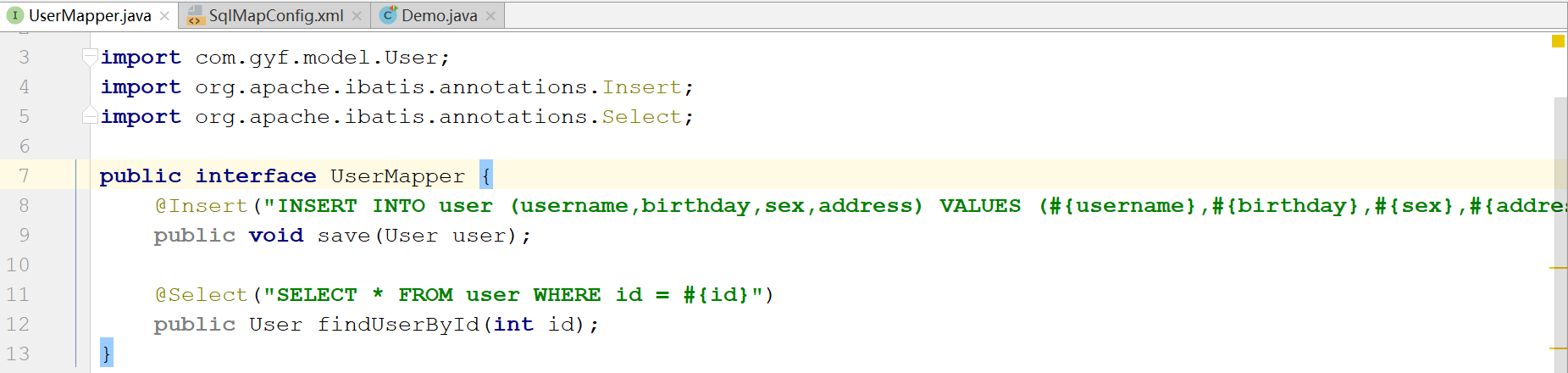
如：<mapper url="file:///D:\workspace\_spingmvc\mybatis\_01\config\sqlmap\User.xml" />

### <mapper class=’’/>

使用mapper接口的全限定名

如：<mapper class="cn.gyf.mybatis.mapper.UserMapper"/>

也可使用注解开发，把xml文件删除



**注意：此种方法要求mapper接口和mapper映射文件要名称相同，且放到同一个目录下**；

### <package name=’’/>（推荐）

注册指定包下的所有映射文件

如：<package name="com.cheney.mapper"/>

**注意：此种方法要求mapper接口和mapper映射文件要名称相同，且放到同一个目录下**；

# Mybatis的映射文件

## 4.1 输入映射ParameterType

指定输入参数的java类型，可以使用别名或者类的全限定名。它可以接收**简单类型,POJO对象、HashMap**。

### 传递简单类型

根据用户ID查询用户信息。

|  |
| --- |
|  |

### 传递POJO对象

|  |
| --- |
|  |

### 传递POJO包装对象

开发中通过pojo传递查询条件 ，查询条件是综合的查询条件，不仅包括用户查询条件还包括其它的查询条件（比如将用户购买商品信息也作为查询条件），这时可以使用包装对象传递输入参数。

**需求**

综合查询用户信息，需要传入查询条件复杂，比如（用户信息、订单信息、商品信息）。

* vo ：键值对对象，相当于key、value
* po : persist object 持久化对象
* pojo : 简单的java对象
* entity : 实体

|  |
| --- |
| 定义POJO包装类 |
|  |

|  |
| --- |
| 修改UserMapper.java |
|  |

|  |
| --- |
| 修改UsrMappler.xml |
|  |

|  |
| --- |
| 测试 |
|  |

### 传递Map对象

|  |
| --- |
| 修改UserMapper.java |
|  |
| 修改UserMapper.xml |
|  |
| 测试 |
|  |

## 4.2 输出映射 resultType/resultMap

### resultType

使用resultType进行结果映射时，查询的列名和映射的pojo属性名完全一致，该列才能映射成功。

如果查询的列名和映射的pojo属性名全部不一致，则不会创建pojo对象；

如果查询的列名和映射的pojo属性名有一个一致，就会创建pojo对象。但是不一致的属性不会封装。

#### 输出简单类型

当输出结果只有一列时，可以使用ResultType指定简单类型作为输出结果类型。

**案例：输出一个count(\*)**

|  |
| --- |
| **UserMapper.java** |
|  |
| **UserMapper.xml** |
|  |
| 测试 |
|  |

#### 输出POJO单个对象

|  |
| --- |
|  |
|  |

#### 输出POJO列表

|  |
| --- |
|  |
|  |

**总结：**

输出单个pojo对象和pojo列表时，mapper映射文件中的resultType的类型是一样的，mapper接口的方法返回值不同。

同样的mapper映射文件，返回单个对象和对象列表时，mapper接口在生成动态代理的时候，会根据返回值的类型，决定调用selectOne方法还是selectList方法。

### resultMap

如果查询出来的列名和属性名不一致，通过定义一个**resultMap将列名和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

|  |
| --- |
| UserMapper.java |
|  |
| UserMapper.xml |
|  |
| 测试 |
|  |

# 动态SQL

## 5.1 if和where

* If标签：作为判断入参来使用的，如果符合条件，则把if标签体内的SQL拼接上。

**注意：用if进行判断是否为空时，不仅要判断null，也要判断空字符串‘’；**

* Where标签：

1. 如果where里面有内容，则会拼接where
2. 会去掉条件中的第一个and符号。

|  |
| --- |
|  |
|  |

## 5.2 choose、when、otherwise

* 相当于java中的switch，从前到后，只要有一个符合，就跳出。

|  |
| --- |
|  |

## 5.3 set、trim

* Set标签：一般用于update语句时使用

1. 如果set里面有内容，则会拼接set
2. 会去掉最后一个 “，”符号

* trim标签：使用起来灵活

1. prefix，suffix ： 前后加上任意字符
2. prefixOverrides，suffixOverrides ：前后覆盖的字符

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

## 5.4 SQL片断

Mybatis提供了SQL片段的功能，可以提高SQL的**可重用性**。

|  |
| --- |
|  |

## 5.5 foreach 遍历

### 参数为包装类，集合是其属性

**案例：查询指定id的用户**

【SELECT \* FROM `user` where id in (31,32,33); 】

|  |
| --- |
| UserQueryVO.java |
|  |
| UserMapper.xml |
|  |
| 测试： |
|  |
|  |
| 注意：${} 和 #{} 都可以。Sql语句都能查询出来 |

### 参数为集合

如果参数是集合的话，parameterType可以写全名【java.util.List】，也可以写别名【list】

遍历或者判断的时候，都用list变量

|  |
| --- |
|  |

### 参数为数组

如果参数是数组的话，parameterType写数组xxx[] ,也可以写java.util.List 或写list

遍历或者判断的时候，都用array变量

|  |
| --- |
|  |

# mybatis与hibernate的区别【面试题】

## Mybatis技术特点：

好处：

1. 通过直接编写SQL语句，可以直接对SQL进行性能的优化；
2. 学习门槛低，学习成本低。只要有SQL基础，就可以学习mybatis，而且很容易上手；
3. 由于直接编写SQL语句，所以灵活多变，代码维护性更好。

缺点：

1. 不能支持数据库无关性，即数据库发生变更，要写多套代码进行支持，移植性不好。
   1. Mysql:limit
   2. Oracle:rownum
2. 需要编写结果映射。

## Hibernate技术特点：

好处：

1. 标准的orm框架，程序员不需要编写SQL语句。（Object Relational Mapping）对象关系映射
2. 具有良好的数据库无关性，即数据库发生变化的话，代码无需再次编写。
   1. 以后,mysql数据迁移到oracle，只需要改方言配置

缺点:

1. 学习门槛高，需要对数据关系模型有良好的基础，而且在设置OR映射的时候，需要考虑好性能和对象模型的权衡。
2. 程序员不能自主的去进行SQL性能优化。

## Mybatis应用场景：

需求多变的互联网项目，例如电商项目。

## Hibernate应用**场**景：

需求明确、业务固定的项目，例如OA项目、ERP项目等。

## Jdbc、hibernate、mybatis的比较

从运行速度上来比较：

jdbc > mybatis > hibernate

从开发速度上来比较：

hibernate > mybatis > jdbc

从流程程度上来比较：

mybatis > hibernate > jdbc

mybatis为什么会比hibernate流行呢？ 因为现在的应用，大部分都要应对海量用户，要处理高并发。

为了提高应用程序的查询性能，在持久层而言，势必要对sql语句进行优化，而hibernate最大的缺点就是无法直接编写sql语句，而调优sql语句，必然少不了直接编辑sql语句， 这一点hibernate无法满足！ 所以才使用mybaits框架代替了hibernate，毕竟在使用mybatis的时候，所有的sql语句，都必须手写，mybatis框架不会自动生成sql语句。

注意，这并不意味着hibernate就过时了，hibernate对于那些访问量不高的应用还是首选！

# 7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   * **orders和items：多对多** |

## 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 （扩展类）

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

|  |
| --- |
|  |
|  |

### resultMap实现（模型里有模型）

**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 （模型里有集合）

**目标：掌握collection的使用**

### 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 |

### 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：**

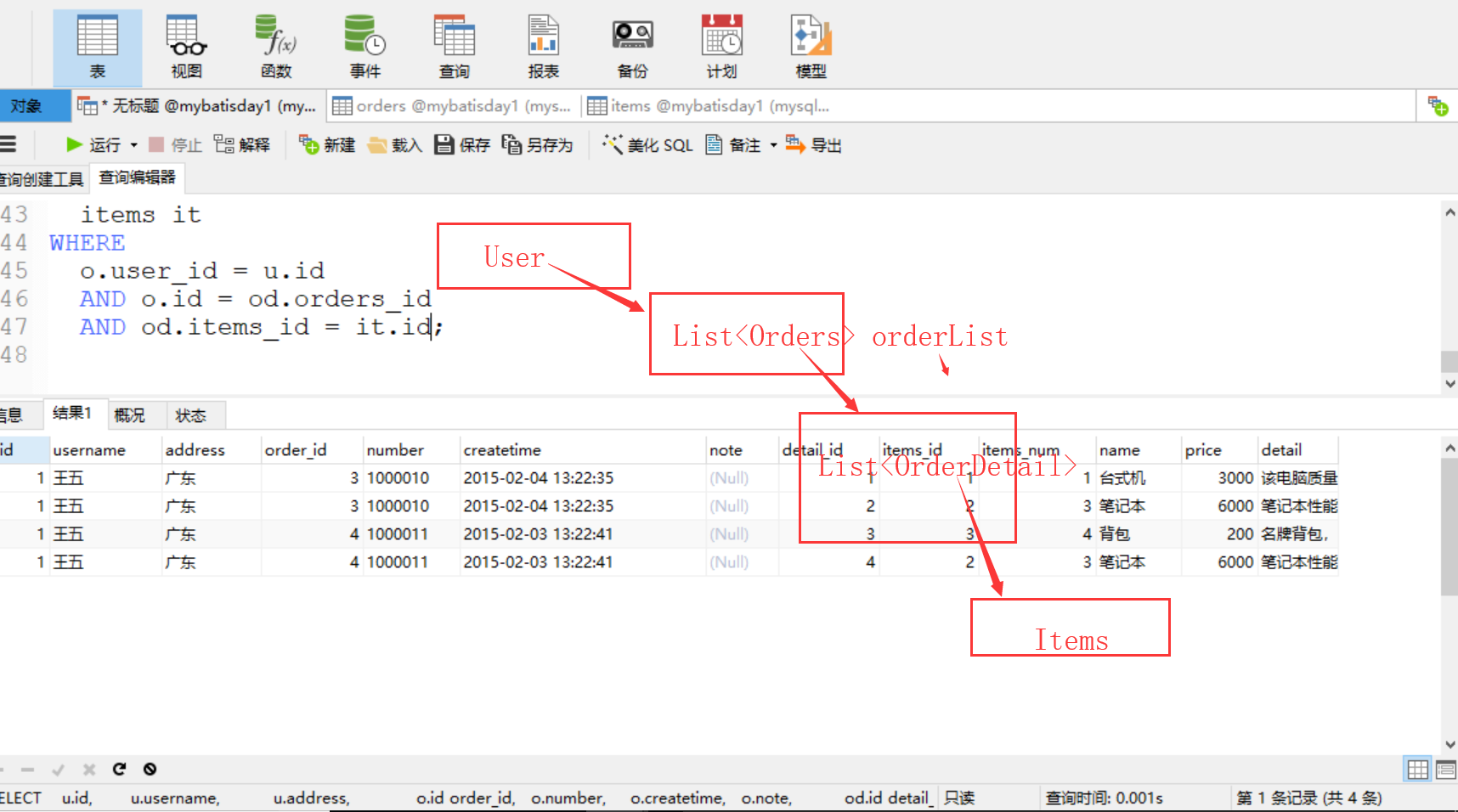
需要对结果集进行二次处理。

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。

# 8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。
* 在主sql语句中不查子属性，而是在association和collection标签中配置select和colum。这样，调用get方法才去执行select里面的方法，而colum则是方法参数对应的字段。

## 2.1 案例：

### Mapper.java

|  |
| --- |
|  |
|  |

### Mapper.xml

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

|  |
| --- |
|  |
| 注意：测试时应该使用dept.getDname()来检测延迟加载是否配置成功，因为如果直接输出dept对象的话（哪怕dept的toString中并没有访问emps），总是直接加载员工表的数据! |

### 测试

|  |
| --- |
|  |

### 配置懒加载

|  |
| --- |
|  |
| <settings>  <setting name="lazyLoadingEnabled" value="true" />  <setting name="aggressiveLazyLoading" value="false" />  </settings> |

# 9查询缓存

## 9.1 Mybatis的缓存理解

* 什么是缓存：缓存就是一片内存。（map）
* 缓存的作用：让数据更接近于使用者。从而提高程序的性能。
* 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。

|  |
| --- |
|  |

## 9.2 一级缓存

### 原理：

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。

* mybatis的一级缓存，是SqlSession级别的缓存。也就是说，只要把SqlSession关闭了，那么一级缓存也就释放了！简单来说，一级缓存不能跨会话！！
* mybatis的一级缓存其实是SqlSession内部封装的一个HashMap。

键值的组成：

statementId + rowBounds + 传递给JDBC的SQL + 传递给JDBC的参数值

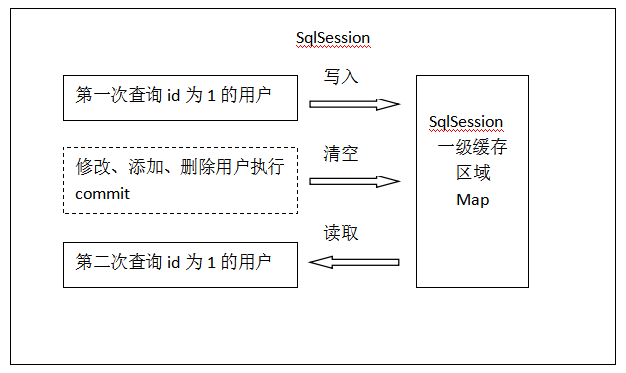
语句id rowBounds对象 sql语句 sql语句的参数

以mapper.find(11);为例，该查询使用的键值是：

find + null + “select \* from user where id = ?” + 11

rowBounds类似分页，查出来再分。一般不使用

* 执行了close、 clearCache 、update、 delete 、insert就会清空一级缓存



### 测试1

|  |
| --- |
|  |

### 测试2

|  |
| --- |
|  |

## 9.3 二级缓存

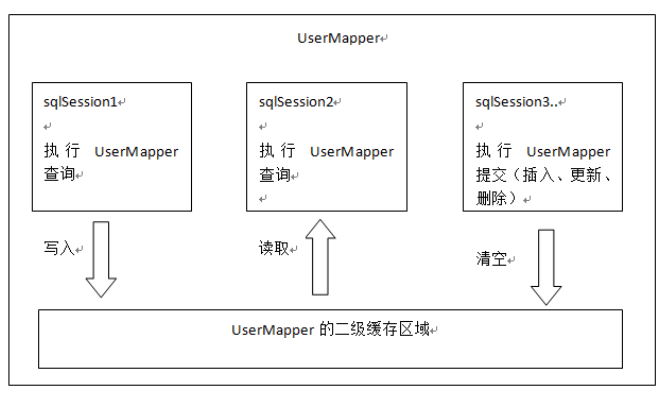
### 原理：

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，这个区域就是二级缓存区域。

* mybatis的二级缓存，是SqlSessionFactory级别的缓存。也就是说，只要把SqlSessionFactory关闭了，那么二级缓存也就释放了！所以二级缓存可以跨会话，毕竟SqlSessionFactory就是用来创建不同的SqlSession，就是说，SqlSessionFactory的生命周期中，有多个SqlSession的生命周期！SqlSessionFactory往往与整个应用同生共死，就是在web应用启动的时候，SqlSessionFactory就被创建了，直到web应用被卸载的时候，SqlSessionFactory才被关闭！
* mybatis的二级缓存其实是SqlSessionFactory内部封装的一个HashMap。二级缓存的键值组成与一级缓存的键值组成是一样的！
* mybaits的二级缓存，默认是关闭着的，不会其效果的，而mybatis的一级缓存是开启的（也无法关闭）
* 开启二级缓存

|  |
| --- |
| 1) 在主配置中  <settings>  <setting name="cacheEnabled" value="true"/>  </settings>  2) 在子配置中 (即哪个模型要使用二级缓存)  <cache /> 默认使用mybatis自带的perpetualCache |

* 既然二级缓存可以跨会话，那么必然对提升系统性能有更大的帮助，那为什么默认是关闭着的呢？？因为二级缓存并不是用的越多就越好。 针对于经常被查询的操作，可以使用二级缓存，针对于经常修改的操作不适合用二级缓存。
* 注意：只有在session关闭后，才会把当前一级缓存中的数据保存到二级缓存中。同一个namespace下的任意session执行了close 、clearCache 、update、 delete、 insert就会清空二级缓存



### 使用：

#### 开启二级缓存总开关

|  |
| --- |
|  |

#### UserMapper中配置二级缓存

在哪个模型中使用二级缓存，type不指定，则默认使用自带的perpetualCache

|  |
| --- |
|  |

#### User序列化

|  |
| --- |
|  |

#### 测试

|  |
| --- |
|  |

### 指定方法禁用二级缓存

|  |
| --- |
|  |

### 刷新缓存

|  |
| --- |
|  |

## 9.4 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）

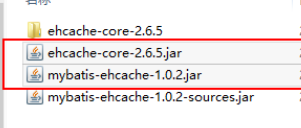
|  |
| --- |
|  |

#### 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

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）** |

#### 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。

# 10 mybaties整合spring

SSH:struts2+spring+hibernate

SSM:SpringMVC + Spring + MyBatis

Spring 3.2

Mybaties 3.2.7

## 10.1 传统整合

### 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

|  |
| --- |
|  |

### 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，

|  |
| --- |
|  |
|  |

### 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> |

### spring配置SqlSessionFactory

|  |
| --- |
| <!-- 2.配置SqlSessionFactory -->  <bean id=*"sqlSessionFactoryBean"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <property name=*"configLocation"* value=*"classpath:SqlMapConfig.xml"*></property>  <property name=*"dataSource"* ref=*"dataSourse"*></property>  </bean> |

### 编写个UserDaoImpl,接口省略

|  |
| --- |
|  |

### spring中配置daobean（传统配置）

传统配置dao

|  |
| --- |
|  |

### 测试

|  |
| --- |
|  |

## 10.2 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（工厂配置）

由spring使用工厂Bean创建userMapper对象

|  |
| --- |
| <!-- 4. *由spring使用工厂Bean生成userMapper对象*-->  <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

|  |
| --- |
|  |

## 10.3 用MapperScannerConfigurer创建代理对象（批量配置）

【上面的代码麻烦，每一个mapper就创建一个工厂bean】

可以使用MapperScannerConfigurer，批量扫描创建代理对象。

批量创建mapper的bean对象，内部会扫描指定包下的mapper，创建对象

|  |
| --- |
| <!-- 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> |

## 10.4 配置dao的三种方式

* 第一种：传统配置

缺点：dao需要继承SqlSessionDaoSupport,

|  |
| --- |
|  |

* 第二种：由spring使用工厂Bean创建userMapper对象

缺点：配置时，需要写多个mapper接口

|  |
| --- |
| <!-- 4. *由spring使用工厂Bean生成userMapper对象*-->  <bean id = *"userMapper"* class=*"org.mybatis.spring.mapper.MapperFactoryBean"*>  <property name=*"mapperInterface"* value=*"com.gyf.backoffice.mapper.UserMapper"*/>  <property name=*"sqlSessionFactory"* ref=*"sqlSessionFactoryBean"*/>  </bean> |

* 第三种：用MapperScannerConfigurer批量扫描创建代理对象

|  |
| --- |
| <!-- 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> |

# 11 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；

Mybatis，mysql，junit，log4j，mybatis-generator-core (专门用于逆向工厂的jar包)

1. 使用java类来执行逆向工程；
2. 把生成的代码拷贝到项目中。
3. 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=*"root"*>  </jdbcConnection>    <!-- 2.java类型解析  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" domainObjectName="Items"* />  <table tableName=*"orderdetail"*/>  <table tableName=*"orders"*/>  <table tableName=*"user"*/>  </context>  </generatorConfiguration> |

|  |
| --- |
| 考入mybatis逆向工程的配置文件，考入到类路径下即可（src/main/resources或src/test/resources） |
| <?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=*"ctx1"*>  <!-- 配置pojo的序列化 -->  <plugin type=*"org.mybatis.generator.plugins.SerializablePlugin"* />    <commentGenerator>  <!-- 是否去除自动生成的注释 true：是 ： false:否 -->  <property name=*"suppressAllComments"* value=*"true"* />  </commentGenerator>  <!--数据库连接的信息：驱动类、连接地址、用户名、密码,这里配置的是mysql的，当然也可以配置oracle等数据库 @@@@@ -->  <jdbcConnection driverClass=*"com.mysql.jdbc.Driver"*  connectionURL=*"jdbc:mysql://localhost:3306/demo"* userId=*"root"*  password=*"root"*>  </jdbcConnection>  <!-- 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>  <!-- targetProject:生成PO类的位置 @@@@@ -->  <javaModelGenerator targetPackage=*"com.cheney.pojo"*  targetProject=*".\src\main\java"*>  <!-- 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=*"com.cheney.dao"*  targetProject=*".\src\main\java"*>  <!-- 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 type=*"XMLMAPPER"*  targetPackage=*"com.cheney.dao"* targetProject=*".\src\main\java"*>  <!-- enableSubPackages:是否让schema作为包的后缀 -->  <property name=*"enableSubPackages"* value=*"false"* />  </javaClientGenerator>  <!-- 指定数据库表 @@@@@-->  <table tableName=*"dept"*></table>  <table tableName=*"emp"*></table>  </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("src/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);  }  } |

|  |
| --- |
| 导入执行逆向工厂的应用：建议把该类存放到test目录下 |
| package com.cheney.test;  import java.io.File;  import java.util.ArrayList;  import java.util.List;  import org.mybatis.generator.api.MyBatisGenerator;  import org.mybatis.generator.config.Configuration;  import org.mybatis.generator.config.xml.ConfigurationParser;  import org.mybatis.generator.internal.DefaultShellCallback;  public class GeneratorSqlmap {    public void generator() throws Exception {  List<String> warnings = new ArrayList<String>();  boolean overwrite = true;  // 指定 逆向工程配置文件  File configFile = new File("src/test/resources/generatorConfig.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);    System.out.println("over");  }    public static void main(String[] args) throws Exception {  try {  GeneratorSqlmap t = new GeneratorSqlmap();  t.generator();  } catch (Exception e) {  e.printStackTrace();  }  }  } |
| 1. 切记，不要运行多次，一次就好！ 2. 注意，mybatis的逆向工程不能像hibernate那样，逆向生成一对一、一对多、多对多的配置。   就算使用了mybatis的逆向工程，也仍然需要自己手动编写mybatis中的一对一、一对多、多对多的配置。 |

### 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

|  |
| --- |
|  |

## 方法详解

* 查条数

long countByExample(UserExample example) ：按条件查，结果的条数

相当于 select count(\*)

同样，数所有，传入null

* 删

1. int deleteByExample(UserExample example) ：按条件删
2. int deleteByPrimaryKey(Integer id) ：按主键删

* 增

1. int insert(User record) ：静态的的，插入所有字段
2. int insertSelective(User record) ：动态sql语句

* 查

1. List<User> selectByExample(UserExample example) ：按照条件查找

如果查所有：则传入null

1. User selectByPrimaryKey(Integer id) ：按照主键查找

* 改

1. int updateByExampleSelective(@Param("record") User record, @Param("example") UserExample example) ：按照条件更新，动态sql
2. int updateByExample(@Param("record") User record, @Param("example") UserExample example) ：按照条件更新，全字段
3. int updateByPrimaryKeySelective(User record) ：按照主键更新，动态sql
4. int updateByPrimaryKey(User record) ：按照主键更新，全字段

|  |
| --- |
|  |