# MyBatis学习笔记

# 第一章

## Sql映射文件

写sql语句的，mybatis会执行这些sql

### 指定约束文件

|  |
| --- |
| <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"> |

mybatis-3-mapper.dtd是约束文件的名称，扩展名是dtd的

### 约束文件的作用

限制，检查在当前文件中出现的标签，属性必须符合mybatis的要求

Mapper是当前文件的根标签，必须的

Namespace：叫做命名空间，唯一值的，可以是自定义的字符串，要求你使用dao接口的全限定名称

4.在当前文件中，可以使用特定的标签，表示数据库的特定操作。

<select>:表示执行查询，select语句

<update>:表示更新数据库的操作， 就是在<update>标签中 写的是update sql语句

<insert>:表示插入， 放的是insert语句

<delete>:表示删除， 执行的delete语句

### MyBatis的主配置文件

|  |
| --- |
| <?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>  <!--环境配置，数据库的连接信息  default:必须和某个environment的id值一样  告诉mybatis使用哪个数据库的链接信息，也就是访问哪个数据库    -->  <environments default="mydev">  <!--environment：一个数据库的配置，环境  id：唯一值，自定义，表示环境的名称      -->  <environment id="mydev">  <!--  transactionManager:mybatis的事务类型  type：JDBC（表示使用jdbc中的Connection对象的commit，rollback做事务处理）  -->  <transactionManager type="JDBC"/>  <!--  dataSource：表示数据源，连接数据库的   type：表示数据源的类型，POOLED表示使用连接池的  -->  <dataSource type="POOLED">  <!--  driver：数据库的驱动类名  user、  username、password是固定的，不能自定义。  -->    <property name="driver" value="com.mysql.jdbc.Driver"/>  <!--连接数据库的url字符串-->  <property name="url" value="jdbc:mysql://localhost:3306/springdb"/>  <!--username:访问数据库的名称-->  <property name="username" value="root"/>  <property name="password" value="123456"/>  </dataSource>  </environment>  </environments>  <mappers>  <mapper resource="org/mybatis/example/BlogMapper.xml"/>  </mappers> </configuration> <!--  mybatis的主配置文件：主要配置了数据库的配置信息，sql映射文件的位置 1.约束文件  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  mybatis-3-config.dtd：约束文件的名称  2.configuration   --> |

### 测试程序

|  |
| --- |
| public class MyApp {  public static void main(String[] args) throws IOException {  //访问mubatis读取student数据  //1.定义一个mybatis主配置文件的名称  String config="mybatis.xml";  //2.读取这个config表示的文件  InputStream in= Resources.*getResourceAsStream*(config);  //3.创建了SqlSessionFactoryBuilder对象  SqlSessionFactoryBuilder builder=new SqlSessionFactoryBuilder();  //4.创建SqlSessionFactory对象  SqlSessionFactory factory=builder.build(in);  //5.【重要】 获取SqlSession对象，从SqlSessionFactory中获取SqlSession  SqlSession sqlSession=factory.openSession();  //6.【重要】指定要执行的sql语句的标识。sql映射文件中的namespace+"."  String sqlId="com.lyy.dao.StudentDao"+"."+"selectStudents";  //7.执行sql语句，通过sqlId找到语句  List<Student> studentList=sqlSession.selectList(sqlId);  //8.输出结果  studentList.forEach(stu -> System.*out*.println(stu));  //关闭SqlSession对象  sqlSession.close();  } } |

### 开启日志

|  |
| --- |
| <**settings**> <**setting name="logImpl" value="STDOUT\_LOGGING"** /> </**settings**> |

# ：主要类的介绍

Resuorces：mybatis中的一个类，负责读取主配置文件

InputStream in=Resources.getResourceAsStream(“mybatis.xml”);

SqlSessionFactoryBuilder:创建SqlSessionFactory对象，

SqlSessionFactoryBuilder builder=new SqlSessionFactoryBuilder();

//创建SqlSessionFactory对象

SqlSessionFactory factory=builder.build(in)

SqlSessionFactory:重量级对象，程序创建这个对象时耗时比较长，使用资源比较多，在整个的项目中，有一个就够用了。

SqlSessionFactory是一个接口，接口的实现类，DefaultSqlSessionFactory

SqlSessionFactory作用：为了获取SqlSession对象。

SqlSession sqlSession=factory.OpenSession();

openSession（）方法的说明：

1. openSession（）：无参数的，获取是非自动提交事务的SqlSession对象
2. openSession（boolean ）：openSession（true）是获取自动提交事务的SqlSession。
3. openSession（false）：非自动提交事务的SqlSession对象

SqlSession：SqlSession接口，定义了操作数据逇方法：例如：selectone（），selectList（），

Insert（），update（），delete（），commit（），rollback（）；

SqlSession接口的实现类：DefaultSqlSession（），

使用要求：不是线程安全的，需要在方法的内部使用，在执行sql语句之前，使用SqlSession获取，在执行完sql语句后，需要关闭它，执行sqlSession.close()。这样能保证他的使用线程是安全的

## MyBatis的动态代理

|  |
| --- |
| public class TestMybatis {  @Test  public void testSelectStudents(){  StudentDao dao=new StudentDaoImpl();  List<Student> studentList= dao.selectStudents();  */\*\*  \*List<Student> studentList= dao.selectStudents();  \* 1.dao对象，类型是StudentDao，全限定名称：com.lyy.dao.StudentDao  \* 全限定名称：namespace是一样的  \* 2：方法名称，selectStudents：这个方法名称就是mapper文件中的id值  \* 3.通过dao中方法的返回值也可以确定Mybatis要调用的SqlSession的方法  \* 吐过返回值是list，调用的是SqlSession.selectList()方法  \* 吐过返回值是int，或是非list，看mapper文件中的标签 是<insert><update>就会调用SqlSession的insert方法，update等方法  \*mybatis的动态代理：mybatis根据dao的方法调用，获取执行sql语句的信息  \* mbatis根据你的dao接口，创建出一个dao接口的实现类，并创建这个类的对象  \* 完成SqlSession调用方法，访问数据库  \*  \*/* for (Student stu:studentList){  System.*out*.println(stu);  }  }  @Test  public void TestInsert(){  StudentDao dao=new StudentDaoImpl();  Student student=new Student();  student.setAge(18);  student.setName("lyy");  student.setId(1008);  student.setEmail("ly320@163.com");  int i = dao.insertStudent(student);  System.*out*.println(i);  } } |

使用动态的代理的方法，直接使用SqlSession.getMapper(dao接口)

getMapper能够获取dao接口对应的实现类对象

|  |
| --- |
| SqlSession sqlSession= MyBatisUtils.*getSqlSession*(); StudentDao dao=sqlSession.getMapper(StudentDao.class); //调用dao的方法，执行数据库的操作 List<Student> students=dao.selectStudents(); |

## 深入理解参数

### 传入参数

从java代码中把数据传入到mapper文件的sql语句中

### paramterType

paramType：接口中方法形参的类型，类型的完全限定名或者别名。这个属性是可选的，因为mybatis可以推断出具体传入语句的参数，默认值为未设置（unset）。接口中方法的参数从java代码传入到mapper文件的sql语句

int或java.lang,Integer

hashmap或java.util.HashMap

list或java.util.ArrayList

student或ocm,lyy.domain,Student

<select> <insert> ,<update>,<delete>都可以使用parameterType指定类型

#### 一个简单类型的参数的获取

*/\*\*  
 \* 一个简单类型的参数：  
 \* 简单类型：mybatis把java的基本数据类型和String都叫做简单数据类型  
 \* 在mapper文件获取简单类型的一个参数的值，使用#{任意字符}  
 \*/*public Student selectStudentById(Integer id);

select id,name,email,age from student where id=#{id}

#### 多个参数--@Param

当dao接口方法多个参数，需要通过名称使用参数。在方法形参前面加入@Param（“自定义参数名”），mapper文件使用#{自定义参数名}.

|  |
| --- |
| */\*\*  \* 多个参数：命名参数，在形参定义的1前面，加入@Param("自定义参数名称")  \*/* List<Student> selectMulitParam(@Param("myname") String name, @Param("myage")Integer age);  <!--多个参数，使用@Param自定义名称--> <select id="selectMulitParam" resultType="com.lyy.domain.Student">  select id,name,email,age from student where name =#{myname} or age=#{myage} </select> |

#### 多个对象—使用对象

使用java对象传递参数，java的属性值就是sql需要的参数值。每一个属性就是一个参数。语法格式：#{property，javaType=java中数据类型名，jdbcType=数据类型名称}

JavaType，jdbcType的类型MyBatis可以检测出来，一般不需要设置。常用格式#{property}

Mybatis-3.5.1.pdf

|  |
| --- |
| <!--多个参数，使用java对象的属性值，作为参数的实际值 使用对象的语法：#{属性名，javaType=类型名称，jdbcType=数据库类型}很少用 javaType：指java中的属性数据类型 jdbcType：在数据库中的数据类型 例如：#{paramName，javaType=java.lang.string,jdbcType=VARCHAR} 我们使用的是简化方式：#{属性名} ，javaType，jdbcType的值mybatis反射能够获取。不要提供  -->  <!-- <select id="selectMulitObject" resultType="com.lyy.domain.Student">  select id,name,email,age from student where  name=#{paramName，javaType=java.lang.String,jdbcType=VARCHAR}  or age=#{paramAge，javaType=java.lang.Integer,jdbcType=INTEGER}  </select>-->  <select id="selectMulitObject" resultType="com.lyy.domain.Student">  select id,name,email,age from student where  name=#{paramName} or age=#{paramAge}  </select> |

#### 多个参数—按位置

参数位置从0开始引用参数语法#{arg位置}，第一个参数是#{arg0}，第二个是#{arg1}

|  |
| --- |
| <!--多个简单参数的传入：简单参数，按位置传入--> <select id="selectMulitPosition" resultType="com.lyy.domain.Student">  select id,name,email,age from student where name =#{arg0} or age={#arg1} </select> |

#### 多个参数—使用Map

Map集合可以存储多个值，使用Map向mapper文件一次传入多个参数。Map集合使用String的key，Object类型的值存储参数。Mapper文件使用#{key}引用参数值

例如：Map<String,Object> data=new HashMap<String,Object>();

Data.put(“myname”,”李莉”)；

Data.put(“myage”,20);

|  |
| --- |
| <!--多个参数，使用Map,使用语法：#{map的key}--> <select id="selectMulitByMap" resultType="com.lyy.domain.Student">  select id,name,email,age from student where  name=#{myname} or age=#{myage} </select> |

#### 比较占位符#和$

##### #：占位符

告诉mybatis使用实际的参数值代替。并使用PrepareStatement对象执行sql语句，#（…）代替sql语句的“？”。这样做更安全，通常也是首选做法

##### $：字符串替换

告诉mybatis使用$包含的“字符串”替换所在位置。使用Statement把sql

语句和${}的内容链接起来。主要用在替换表名，列名，不同列排序等操作

Select id,name,email,age from student where id=#{studentId}

#的结果：Select id,name,email,age from student where id=？

Select id,name,email,age from student where id=${studentId}

$的结果：Select id,name,email,age from student where id=1001

String sql=“select id，name，email，age from student where id=”+“1001”；

使用的Statement对象执行sql，效率比PrepareedStatement低

$：可以替换表名或者列名，你能确定数据是安全的，可以使用$

#和$区别

1.#使用？在sql语句中做占位的，使用PrepareedStatement执行sql语句，效率高

2.#能够避免sql注入，更安全。

3.$不使用占位符，是字符串连接方式，使用Statement对象执行sql，效率低

4.$有sql注入的风险，缺乏安全性

5.$:可以替换表名或者列名

#### 动态代理的步骤

什么是动态代理：

Mybatis帮你创建dao接口的实现类，在实现类中调用SqlSession的方法执行sql语句

使用动态代理：

1.获取SqlSession对象，SqlSessionFactory.OpenSession（）

2.使用getMapper方法获取某个接口的对象，sqlSessionMapper（接口.class）

3.使用dao接口的方法，调用方法就执行mapper文件中的sql语句

使用动态代理方式的要求：

1. dao接口和mapper文件放在一起，同一个目录
2. dao接口和mapper文件名称一致
3. mapper文件中的namespace的值是dao接口的全限定名称
4. mapper文件中的<select>,<insert>,<update>,<delete>等的id是接口中方法名称
5. dao中不要使用重载方法，不要使用同名的，不同参数的方法

# 封装mybatis输出结果

## ResultType

ResultType：执行sql语句得到ResultSet转换的类型，使用类型的完全限定名或别名。注意如果返回的是集合，那应该设置为集合包含的类型，而不是集本身。ResultType和resultMap，不能同时使用

## Mybatis的输出结果

1. resultType结果类型：指sql语句执行完毕之后，数据转为的java对象，java类型是任意的  
   resultType把ResultSet指定列值付给同名的属性
2. 处理方式：

1.mybatis执行sql语句，然后mybatis调用类的无参数构造方法，创建对象

2.mybatis把ResultSet指定列值付给同名的属性

## 定义自定义类型的别名

1. 在mybatis主配置文件中定义，使用<typeAlias>定义别名
2. 可以在resultType中使用自定义别名

|  |
| --- |
| 1. <!--定义别名--> <typeAliases>  <!--可以指定一个类型一个自定义别名  type:自定义类型的全限定名称  alias：自定义别名（短小，容易记忆的）  -->  <typeAlias type="com.lyy.domain.Student" alias="stu"/>  <typeAlias type="com.lyy.vo.ViewStudent" alias="vstu"/>   <!--第二种方式 <package> name是包名，这个包中的所有类，类名就是别名（类名不区分大小写） --> <package name="com.lyy.domain"/> <package name="com.lyy.vo"/> </typeAliases> |

## Map

Sql的查询结果作为Map的key和value。推荐使用Map<Object,Object>

注意：Map作为接口返回值，sql语句的查询结果最多只能有一条记录。一条记录是错误

|  |
| --- |
| <!--返回Map 1)列名是map的key，列值是map的value 2）返回map的时候，最多犯规一行记录。多余一行记录是错误的。 --> <select id="selectMapById" resultType="java.util.HashMap">  select id,name from student where id=#{stuid} </select> |

## ResultMap

ResultMap可以自定义sql的结果和java对象属性的映射关系。更灵活的把列值赋给指定属性

常用在列名和java、对象属性名不一样的情况

使用方法：

先定义resultMap，指定类名和属性的对应

在<select>中把resultMap替换为resultMap

|  |
| --- |
| <!--使用resultMap 1)先定义一个resultMap 2）在select标签，使用resultMap来引用1定义的 --> <!--定义resultMap的标签 id：自定义名称，表示你定义的这个resultMap type：java类型的全限定名称 --> <resultMap id="studentMap" type="com.lyy.domain.Student">  <!--列名和java属性的关系-->  <!--主键列：使用id标签  column:列名  property:java类型的属性名  -->  <!--非主键列，使用result-->  <id column="id" property="id"/>  <result column="name" property="name"/>  <result column="email" property="email"/>  <result column="age" property="age"/>   </resultMap> <select id="selectAllStudents" resultType="studentMap">  select id,name,email,age from student </select> |

ResultMap和resultType不能一起用，二选其一

当属性名和列名不一样时，可以有以下两种方式实现

|  |
| --- |
| <!--当属性名和列表不一样  第一种方式：resultMap-->  <!--使用resultMap  1)先定义一个resultMap  2）在select标签，使用resultMap来引用1定义的  -->  <!--定义resultMap的标签  id：自定义名称，表示你定义的这个resultMap  type：java类型的全限定名称  --> <!-- <resultMap id="studentMap" type="com.lyy.domain.Student">  &lt;!&ndash;列名和java属性的关系&ndash;&gt;  &lt;!&ndash;主键列：使用id标签  column:列名  property:java类型的属性名  &ndash;&gt;  &lt;!&ndash;非主键列，使用result&ndash;&gt;  <id column="id" property="id"/>  <result column="name" property="name"/>  <result column="email" property="email"/>  <result column="age" property="age"/>   </resultMap>  <select id="selectAllStudents" resultType="studentMap">  select id,name,email,age from student  </select>--> <!-- /\*  列名和属性名不一样时：第二种方式  resultType的默认原则是，同名的列赋值给同名的属性，使用列别名(java对象的属性名)  \*/-->  <select id="selectDiffcolProperty" resultType="com.lyy.domain.Mystudent">  select id as stuid,name as stuname,email as stuemail,age as stuage from student  </select> |

## 模糊like

模糊查询的实现有两种方式，一是java代码中给查询数据加上“%”；二是在mapper文件中拼接%李%

第一种方式：

|  |
| --- |
| <!--第一种like，java代码指定like的内容--> <select id="selectOneLike" resultType="com.lyy.domain.Student">  select id,name,email,age from student where name like #{name} </select>  public void selectOneLike(){  SqlSession sqlSession=MyBatisUtils.*getSqlSession*();  StudentDao dao = sqlSession.getMapper(StudentDao.class);  //准备好like的内容  String name="%李%";  List<Student> lists = dao.selectOneLike(name);  for (Student student:lists){  System.*out*.println("查询到的学生信息="+student);  } } |

第二种方式：

|  |
| --- |
| <!--第二种方式：在mappern文件中拼接like的内容--> <select id="selectLikeTwo" resultType="com.lyy.domain.Student">  select id,name,email,age from student where name like "%" #{name} "%" </select> |

# 动态sql

## 动态sql

Sql的内容是变化的，可以根据条件获取到不同的sql语句。主要是where部分发生变化

动态sql的实现：使用的是mybatis提供的标签，<if>,<where>,<foreach>

## 1.<if>

<if>是判断条件，

语法<if test=”判断java对象的属性值”>

部分sql语句

</if>

|  |
| --- |
| <!--if-->  <select id="selectStudentIf" resultType="com.lyy.domain.Student">  select id,name,age email from student where <!-- if  <if:test="使用/\*\*/参数java对象的属性值作为1判断条件，语法：属性=xxx值"  -->   <if test="name !=null and name !='' ">  name =#{name}  </if>  <if test="age > 0">  and age>#{age}  </if>  </select> |

## 2.where

<where>用来包含多个<if>的，当多个if有一个成立的话，<where>会自动增加一个where关键字，并去掉if中多余的and，or等。

|  |
| --- |
| where：<where> <if><if>...</where> --> <select id="selectStudentWhere" resultType="com.lyy.domain.Student">  select id,name,email,age from student <where>  <if test="name !=null and name !='' ">  name =#{name}  </if>  <if test="age > 0">  and age>#{age}  </if> </where> </select> |

## <foreach>

### Foreach添加list集合

<foreach/>标签用于实现对于数组与集合的遍历。对其使用，需要注意：

Collection表示要遍历的集合类型，list，array等。

Open，close，separator为遍历内容的SQL拼接

|  |
| --- |
| <!--foreach使用1，List<Integer> collection:表示接口中的方法参数的类型，如果是数组使用array，如果是list使用list item：自定义的，表示数组或结合成员的变量 open：循环开始时的字符 close：循环结束时的数组 separator:集合成员之间的分隔符 --> <select id="selectForeachOne" resultType="com.lyy.domain.Student">  select \* from student where id in  <foreach collection="list" item="myid" open="(" close=")" separator=",">  #{myid}  </foreach> </select>  @Test public void testForEach(){  SqlSession sqlSession=MyBatisUtils.*getSqlSession*();  StudentDao dao=sqlSession.getMapper(StudentDao.class);  List<Integer> list=new ArrayList<>();  list.add(1001);  list.add(1002);  list.add(1003);  List<Student> students=dao.selectForeachOne(list);  for (Student stu:students){  System.*out*.println("forOne=="+stu);  } } |

### <foreach>添加对象

|  |
| --- |
| <foreach collection="list" item="stu" open="(" close=")" separator=",">  #{stu.id} </foreach>  </select>  @Test public void testForEachTwo(){  SqlSession sqlSession=MyBatisUtils.*getSqlSession*();  StudentDao dao=sqlSession.getMapper(StudentDao.class);  List<Student> students=new ArrayList<Student>();  Student s1 = new Student();  s1.setId(1001);  students.add(s1);  s1=new Student();  s1.setId(1005);  students.add(s1);  List<Student> studentList = dao.selectForeachTwo(students);  for (Student stu:students){  System.*out*.println("forTwo=="+stu);  } } |

## 代码片段

<sql/>标签用于定义SQL片断，以便其他SQL标签服用。而其他标签使用该SQL片断，需要使用<include/>子标签。该<sql/>标签可以定义SQL语句中的任何部分，所以<include/>子标签可以放在动态SQL的任何位置

Sql代码片段，就是复用一些语法

步骤

先定义<sql id=”自定义名称唯一”>sql语句，表名，字段等</sql>

再使用，<include refid=”id的值”/>

|  |
| --- |
| <sql id="studentSql">  select id,name,age,email from student </sql>  <include refid="studentSql"></include> |

# 属性文件配置

|  |
| --- |
| <environments default="mydev">  <environment id="mydev">  <!--  transactionManager:mybatis提交事务，回滚事务的方式  type：事务的处理的类型  1）JDBC:表示mybatis底层是调用JDBC中的Connection对象的，commit，rollback  2）MANAGER：把mybatis的事务处理委托给其他的容器（一个服务器软件，一个框架（spring））     -->  <transactionManager type="JDBC"/>  <!--  dataSource：表示数据源1，java体系中，规定了实现了javax.sql.DataSource接口的都是数据源  数据源表示Connection对象的  1)POOLED：使用连接池，mybatis会创建pooledDataSource类  2）UPOOLED：不使用连接池，在每一次执行sql语句的时候，先创建连接，执行sql，再关闭连接  mybatis会创建unPooledDataSource，管理Connection对象的使用  3）JNDI：java命名和目录服务（windows的注册表）  -->  <dataSource type="POOLED">  <property name="driver" value="com.mysql.jdbc.Driver"/>  <property name="url" value="jdbc:mysql://localhost:3306/springdb"/>  <property name="username" value="root"/>  <property name="password" value="123456"/>  </dataSource>  </environment> </environments> <mappers>  <mapper resource="com/lyy/dao/StudentDao.xml"/> </mappers> |

1.数据库的属性配置文件：数据库连接信息放到一个单独的文件中。和mybatis主配置文件分开。

目的·是便于修改，保存，处理多个数据库的信息。

1. 在resources目录中定义一个属性配置文件中，xxx.properties，例如 jdbc.properties  
   在属性配置文件中，定义数据，格式是key=value key：一般使用，做多级目录的
2. 在mybatis的主配置文件，使用<property>指定文件的位置，在需要使用值大的地方，${key}

|  |
| --- |
| 1. jdbc.driver=com.mysql.jdbc.Driver jdbc.url=jdbc:mysql://localhost:3306/springdb jdbc.user=root jdbc.passwd=123456   <dataSource type="POOLED">  <!--指定properties文件的位置，从类路径根开始找文件--> <properties resource="jdbc.properties">  </properties>  <property name="driver" value="${jdbc.driver}"/>  <property name="url" value="${jdbc.url}"/>  <property name="username" value="${jdbc.user}"/>  <property name="password" value="${jdbc.passwd}"/> |

1. Mapper映射到表中的多个mapper文件

|  |
| --- |
| 1. <!--sql mapper（sql映射文件）的位置--> <mappers>  <!--第一种方式：指定多个文件mapper文件-->  <!--<mapper resource="com.lyy.dao.OrderDao.xml"/>-->  <mapper resource="com/lyy/dao/StudentDao.xml"/>  <!--第二种方式：使用包名  name:xml(mapper文件)所在的包名,这个包中的xml文件一次都能加载给mybatis  使用package的要求：  1.mapper文件名称需要和接口名称一样，区分大小写的一样  2.mapper文件和dao需要在同一目录  -->  <package name="com.lyy.dao"/> </mappers> |

# 扩展

## pageHelper

pageHelper是做数据分页的