第5章


视频讲解

动态SQL

CHAPTER 5

本章学习目标：

- 了解动态 SQL 的大致原理。
- 掌握常见动态 SQL 元素的使用方法。
- 掌握不同动态 SQL 元素的区别和联系。

在传统 JDBC 的使用过程中，有一个较大的缺点，就是针对每次的数据库访问，都需要重新编写对应的 SQL 语句。在一个具体的软件系统中，数据库的访问都是千变万化、数目繁多的，这就要求开发者编写大量的 SQL 语句。这些 SQL 语句的管理本身就是一个问题，随着语句的增多，代码的可维护性、可阅读性都大大下降，不利于项目的长期维护升级。

开发人员通过研究发现，实际上很多的 SQL 语句虽然最终操作目的不一样，但是在某些地方都有相似之处，例如，针对某个字段或某几个字段查询数据表的某个条目。这些较为类似的 SQL 语句可否想办法将它们归纳到一起？这样或许就能够适当地降低 SQL 代码这部分的复杂度。顺着这个思路，慢慢就出现了动态 SQL 的概念。

5.1 动态 SQL 概述

动态 SQL,顾名思义,就是 SQL 语句不再是完全写死的 String 变量,而是可以根据具体的使用场景进行动态拼接的一种技术方案。在 MyBatis 框架中,提供了较为完善的动态 SQL 技术,开发人员可以根据需要进行灵活的动态 SQL 语句的拼接。这样可以较好地提升 SQL 语句的可维护性和可阅读性,提高 SQL 语句的复用性。

MyBatis 框架的常用动态 SQL 元素如表 5-1 所示。

表 5-1 MyBatis 框架的动态 SQL 元素

属 性	描 述 说 明
if	单条件分支判断语句,相当于 Java 语句中的 if else
choose、when 和 otherwise	多条件分支判断语句,相当于 Java 语句中的 switch case
trim、where、set	辅助元素,用于处理 SQL 语句的拼装问题
foreach	循环语句,相当于 Java 语句中的 for 循环
bind	定义 OGNL 表达式的语句

需要注意的是,动态 SQL 的处理方式是对 SQL 语句本身进行拼接,因此表中相关的元素都是在映射文件中进行使用的。

5.2 动态 SQL 的常用元素

5.2.1 if 元素

if 元素是单条件分支判断语句,它相当于 Java 中的 if else 语句。if 元素一般和 test 属性一起使用,当 if 元素的判断条件成立时,test 属性对应的 SQL 语句生效;当 if 元素的判断条件不成立时,test 属性对应的 SQL 语句不生效。

在实际的使用过程中,if 元素一般用于某个完整 SQL 语句的附加条件的判断。例如,需要查询数据库中的某条数据,可以使用姓名或年龄来进行查询。假如这里的姓名是必须提供的参数,而年龄是可选的参数,那么就可以将年龄这个可选参数对应的 SQL 语句放入 if 元素中。通过这种方式,可以在一个 SQL 语句中,同时包含使用姓名和年龄查询、单独使用姓名查询这两种方式,也就是通过一个 SQL 语句实现了 JDBC 两个 SQL 语句的功能,这正是动态 SQL 机制的优势所在。

下面通过一个完整的案例,来讲解 if 元素在 MyBatis 框架中的具体使用方法。

1. 数据准备

进入 MySQL 的 bin 文件夹,打开 cmd 命令行,输入指令 mysqld --console 开启 MySQL 的服务。打开 HeidiSQL 客户端,连接数据库,新建 student 数据表,并插入以下测试数据,如图 5-1 所示。

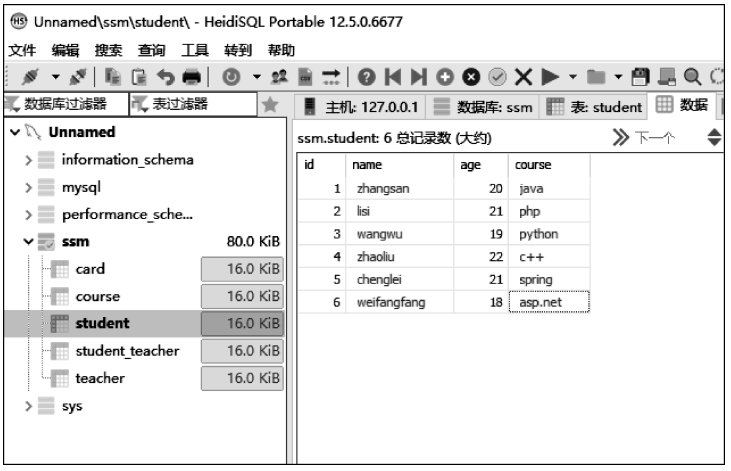

图 5-1　student 数据表中插入测试数据示意图

2. 创建工程

打开 Eclipse 客户端，按照默认参数创建 Dynamic Web Project 即动态 Web 项目。在项目的 main/webapp/WEB-INF/lib 文件夹中，添加 MyBatis 框架所需的各种 jar 包，具体和前面章节案例所讲一致。在 Java 文件夹下创建不同的包，用于放置映射文件的 com.mapper，放置 Java 类的 com.pojo，放置测试代码的 com.test。

3. 创建配置文件

在 Java 文件夹根目录下创建配置文件 MybatisConfig.xml，配置文件中的内容大体和前面章节的内容一致，主要就是完成别名的设置，数据源的地址、用户名和密码等设置，以及映射文件的路径设置。

```
<?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>
  <typeAliases><package name="com.pojo"/></typeAliases>
  <environments default="ssmmysql">
    <environment id="ssmmysql">
      <transactionManager type="JDBC" />
      <dataSource type="POOLED">
      <property name="driver" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/ssm?useSSL=false" />
        <property name="username" value="root" />
        <property name="password" value="" />
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="com/mapper/StudentMapper.xml" />
  </mappers>
</configuration>
```

4. 创建映射文件

在工程的映射文件 Java 包 com.mapper 下,创建项目的映射文件 StudentMapper.xml。

```
<?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">
<mapper namespace="studentMapper">
  <select id="findStudentById" resultType="student">
    select * from student where id = #{id}
  </select>
  <!-- 动态 SQL,if 元素 -->
  <select id="findStudent02" parameterType="String" resultType="student">
  select * from student where id = 2
    <if test = "name!=null and name!=''">
      and name = #{name}
    </if>
  </select>
</mapper>
```

第 06～08 行代码是一个普通的 SQL 查询语句,通过传入一个 int 类型的 id 值,查询数据库后返回一个 com.pojo.Student 类型的 Java 对象。由于配置文件起了别名,这里 Java 类型可以缩写为 student。

第 09～15 行代码是演示 if 元素的使用方法的 select 元素。这里 select 元素的 id 属性对应查询语句的引用标识,parameterType 是查询语句对应的输入参数的数据类型,resultType 是返回的数据类型,也就是 Java 类的名称别名。第 11 行代码 select * from student where id = 2,含义是从 student 表进行查询,这里的 where id=2 是为了方便测试写的 where 子句,主要是为了 if 元素生效后的子句进行拼接。在实际使用过程中,这里的 where 子句也可以根据开发者的意愿替换成其他的 where 条件子句。

第 12～14 行是 if 元素用法的核心代码。if 元素有一个 test 属性,它主要用于条件判断的条件定义。在这里判断条件是 name!=null and name!='',也就是查询的 student 表的 name 字段的参数存在且不为空。当满足这个条件时,if 元素生效,if 元素的内容也就是对应的第 13 行 SQL 语句会和第 11 行 SQL 代码拼接到一起。换言之,当条件 name!=null and name!='' 为真时,对应的 SQL 语句代码是 select * from student where id = 2 and name = #{name},if 元素生效;当条件不为真时,对应的 SQL 语句代码是 select * from student where id = 2,if 元素不生效。

5. 创建测试文件

TestMain.java

```
package com.test;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import org.apache.ibatis.io.Resources;
```

```
08  import org.apache.ibatis.session.SqlSession;
09  import org.apache.ibatis.session.SqlSessionFactory;
10  import org.apache.ibatis.session.SqlSessionFactoryBuilder;
11  import com.pojo.Student;
12  public class TestMain {
13    public static void main(String[] args) {
14      //TODO Auto-generated method stub
15      try {
16        String mybatisConfigFilename = "MybatisConfig.xml";
17        InputStream in = Resources.getResourceAsStream(mybatisConfigFilename);
18        SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(in);
19        SqlSession sqlSession = factory.openSession();
20        //普通查询
21        Student stu = sqlSession.selectOne("studentMapper.findStudentById", 5);
22        System.out.println(stu.toString() + '\n');
23        //动态 SQL,if 元素
24        Student stu0201= sqlSession.selectOne("studentMapper.findStudent02");
25        System.out.println(stu0201.toString() + '\n');
26        Student stu0202= sqlSession.selectOne("studentMapper.findStudent02", "lisi");
27        System.out.println(stu0202.toString() + '\n');
28        sqlSession.close();
29      } catch (IOException e) {
30        //TODO Auto-generated catch block
31        e.printStackTrace();
32      }
33    }
34  }
```

在测试文件中，主体代码是 MyBatis 框架的正常使用，包括相关 Java 包的导入、SqlSession 的创建和关闭等，前面章节已经详细讲解过，这里不再赘述。

第 20～22 行代码对应的是一个 select 元素的普通查询的使用，这里主要用来和 if 元素的使用进行对比。第 23～27 行是对含有 if 元素的查询语句的使用，其中，第 24 行代码里没有传入 String 变量，也就是数据表 student 中 name 字段不存在，此时 if 元素不生效。第 26 行代码这里传入了 String 变量，name 字段不为空，此时 if 元素生效。

执行以上代码，控制台会输出对应的 Student 对象的 String 值，如图 5-2 所示。

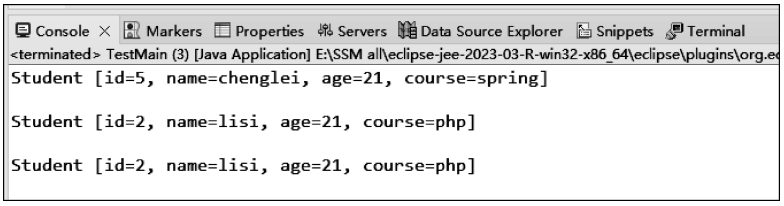

图 5-2　if 元素的代码执行结果示意图

第 24 行和第 26 行代码都是调用的同一条 select 元素，但是一个传入参数一个不传参数，都可以正常执行。本质就是它们底层执行的 SQL 代码不一样，这也就证实了 if 元素加入后，可以实现对 SQL 代码的复用。这里由于数据表中 id＝2 和 name＝lisi 对应的是一条数据，所以有没有 if 元素查询到的结果是一致的，读者也可以自行设计其他的测试数据使得 if 元素使用前后的测试结果不一致。

在这个使用场景里，可以认为 name 字段是查询条件的一个可选参数，当传入 name 字

段时会作为联合查询条件一起查找数据结果。当没有 name 字段时,查询已有的条件,同样可以生效。这就是动态 SQL 里 if 元素的典型使用场景,通过它来完成基本的 SQL 语句的复用。

5.2.2 choose、when、otherwise 元素

if 元素对应的是单条件分支判断,只能处理是或否两种情况。当需要进行多条件分支判断时,就需要用到 choose、when、otherwise 元素了。

和 if 元素类似,choose、when、otherwise 元素也是放在映射文件的 SQL 语句中进行使用,和其他的子句一起完成 SQL 语句的多条件拼接。

同样用一个案例来演示 choose、when、otherwise 元素的使用。大体的项目创建方式和文件组织和 if 元素的一致,这里就直接沿用原项目进行更改。

1. 修改映射文件

在映射文件 StudentMapper.xml 中,加入 choose、when、otherwise 元素的相关定义代码,具体如下。

```
<!-- 动态 SQL,choose,when,otherwise 元素 -->
<select id="findStudent03" parameterType="HashMap" resultType="student">
    select * from student where 1 = 1
    <choose>
        <when test = "name!=null and name!=""">
            and name = #{name}
        </when>
        <when test = "age!=null and age!=""">
            and age = #{age}
        </when>
        <when test = "course!=null and course!=""">
            and course = #{course}
        </when>
        <otherwise>
            and id = 3
        </otherwise>
    </choose>
</select>
```

为了配合 choose、when、otherwise 元素的多条件分支,这里的 select 元素的使用比之前的基础用法复杂。select 元素的属性 parameterType 对应的是 SQL 语句传入的参数的数据类型,这里定义的是 HashMap 哈希表。哈希表的数据存储方式类似于“键值对”,每一个键 key 对应一个取值 value。当需要传入参数的时候,把多个参数按照键值对的方式封装到 HashMap 哈希表里,具体的使用在测试文件中会再继续讲解。

choose、when、otherwise 元素的使用中,最基本的是 choose 元素,when 和 otherwise 是作为 choose 元素的子元素存在的。其中,每一个 when 元素对应一个条件分支,判断的条件用 test 属性来标注。在上面的代码中,一共有三个条件分支,每一个都是分别对不同的字段进行判断,看当前字段是否存在且不为空。如果存在且不为空,则判定条件为真,when 元素的内容,也就是对应的 SQL 子句生效,会加入拼接。以 name 字段为例,用 name!=null and name!=""对它进行判断,如果不为空,则在原 SQL 语句的基础上加入 and name =

#{name}拼接，将 name 字段对应的值作为查询的参数。otherwise 元素则是对 when 元素的补充。当所有的 when 元素判定的条件都不为真时，则 otherwise 元素生效，里面的 SQL 子句加入拼接。

在理解 choose、when、otherwise 元素的时候，可以类比为 Java 中的 switch case 语句。switch 是对条件进行判断，每一个 case 代表一个条件分支，当每一个 case 分支都不满足的时候就执行 default 分支。这里的 swith、case、default 的含义就大致上分别与 choose、when、otherwise 一一对应。

2. 修改测试文件

在测试文件 TestMain.java 中，加入 choose、when、otherwise 元素的相关使用代码，具体如下。

```
//动态 SQL, choose, when, otherwise 元素
//parameterType 需要传入多个参数时
HashMap < String, Object > params = new HashMap < String, Object >();
//params.put("name", "lisi");
params.put("age", 19);
System.out.println(params.toString());
Student stu03 = sqlSession.selectOne("studentMapper.findStudent03", params);
System.out.println(stu03.toString() + '\n');
```

如前所示，映射文件的 select 元素中的参数是 HashMap 哈希表，因此这里测试文件需要先定义一个哈希表的参数 HashMap < String, Object > params。String 对应哈希表的“键”的数据类型是字符串，Object 对应哈希表的“值”的数据类型是 Java 类，由此形成特定的键值对哈希表作为传入的参数。params.put()函数可以往哈希表中放入键值对参数，根据前面映射文件中 choose 元素的分支，这里的参数可以分别对 name、age、course 进行设置（也可以加入任意其他参数，但不会生效）。

根据需要完成三个字段的设置后，就可以调用 select 元素的 SQL 语句进行查询，借助 MyBatis 框架的特性，查询得到的结果是一个 Java 类对象。

执行以上测试代码，控制台会打印输出对应 Java 对象的 String 值，如图 5-3 所示。

图 5-3　choose、when、otherwise 元素的代码执行结果示意图

执行结果演示的是 age＝19 的查询条件，读者也可以根据代码，尝试其他的查询条件，如 name＝lisi 等，执行结果会根据查询条件的不同产生不同的结果。在这个过程中，映射文件内的 SQL 语句是没有变化的，只是查询的参数不同就可以得到不同的查询结果，这就是启动动态 SQL 机制后达到的对 SQL 语句的复用，是 MyBatis 框架的优点体现。

5.2.3　where 元素

根据已经学习的 SQL 语句知识，where 子句是一种常用的 SQL 子句，可以拼接到对应

的 SQL 语句后面。在查询语句中,where 子句可以作为查询条件,如 select * from student where sid = 1。当 where 子句后面的查询条件存在多个时,则需要有对应的连接词进行连接,如 select * from student where sname = lisi and age = 19。

动态 SQL 的本质就是根据编程人员的意愿实现灵活的 SQL 语句的复用和拼接,在这个过程中,单查询条件和多查询条件中多出来的连接词需要更灵活、智能化地处理,这个时候就可以使用 where 元素来完成。

下面是一个完整的 where 元素的使用案例。同样地,这里继续沿用原项目进行更改。

1. 修改映射文件

在映射文件 StudentMapper.xml 中,加入 where 元素的相关定义代码,具体如下。

```
<!-- 动态 SQL,where 元素 -->
<select id="findStudent05" parameterType="HashMap" resultType="student">
select * from student
<where>
<if test = "name!=null and name!=''">
and name = #{name}
</if>
<if test = "course!=null and course!=''">
and course = #{course}
</if>
</where>
</select>
```

在这里,一个较长的 SQL 语句被动态 SQL 元素分隔成了几个不同的子句,其中比较关键的就是 where 元素定义的查询条件的子句。在 where 元素下面是两个 if 元素,分别表示不同的查询条件子句。任意一个 if 元素的查询条件生效时,where 元素会智能地进行判断,如果该子句紧接着 where 元素,则会自动删去 and 单词;如果该子句是接在 where 子句后,则会保留 and 单词。同时 where 元素会自动进行识别,如果有任意子句生效,则会在子句和 select * from student 之间添加 where 单词;如果没有子句生效,则没有 where 单词。

这里的表述稍显烦琐,实际就是 where 元素可以根据具体的 if 元素判断为真、是否有查询条件的子句生效,来动态、智能地进行 SQL 语句的拼接,从而让最后生成的 SQL 语句可以顺利传入数据库执行,防止语法错误。

2. 修改测试文件

在测试文件 TestMain.java 中,加入 where 元素的相关使用代码,具体如下。

```
//动态 SQL,where 元素
HashMap<String, Object> params03 = new HashMap<String, Object>();
params03.put("name", "wangwu");
params03.put("course", "python");
System.out.println(params03.toString());
Student stu05 = sqlSession.selectOne("studentMapper.findStudent04", params03);
System.out.println(stu05.toString() + '\n');
```

这里对 name 和 course 都分别进行了设置,也就是 where 元素里 if 元素的两个判断条件都为真。根据前面对 where 元素使用方式的分析,最终得到的 SQL 语句就是 select * from student where name = #{name} and course = #{course}。读者可以手动将此

SQL 语句输入数据库，会发现执行结果和最终项目代码的执行结果是一致的。

在 Eclipse 里，右击选择项目作为 Java Application 运行，SQL 语句得到正确执行输出对应的查询结果，如图 5-4 所示。

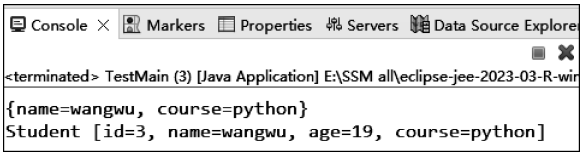

图 5-4　where 元素的代码执行结果示意图

5.2.4　set 元素

在 SQL 语句的语法中，和 where 子句类似的还有其他情况。where 一般是用于查询语句的子句，而在更新语句中需要用到 set 语法。例如，update student set name = lisi where sid = 1，就是将 student 表中 sid 为 1 的数据条目的 name 字段修改为 lisi。当有多个修改字段时，SQL 语句发生变化，如 update student set name = lisi，course = java where sid = 1，同时修改 name 和 course，在这里除了添加修改语句外还增加了一个逗号作为分隔符。

where 元素是针对查询语句的拼接，智能地处理拼接细节。set 元素的用法和作用与此相似，set 元素是针对更新语句的拼接，也可以智能地处理拼接细节，防止 SQL 语法错误。

下面是一个完整的 set 元素的使用案例。同样地，这里继续沿用原项目进行更改。

1. 修改映射文件

在映射文件 StudentMapper.xml 中，加入 set 元素的相关定义代码，具体如下。

```
<!-- 动态 SQL,set 元素 -->
<update id="updateStudent" parameterType="HashMap">
update student
<set>
<if test = "name!=null and name!=''">
name = #{name},
</if>
<if test = "course!=null and course!=''">
course = #{course},
</if>
</set>
where id = #{id}
</update>
```

和刚刚举例说明的 SQL 语句含义类似，这里定义一个更新语句。它根据传入的不同参数，来决定是否对相关字段做修改。set 元素可以动态、智能地往 SQL 语句中添加 set 单词，并智能判断逗号是否保留或删去。

2. 修改测试文件

在测试文件 TestMain.java 中，加入 set 元素的相关使用代码，具体如下。

```
//动态 SQL,set 元素
HashMap<String, Object> params04 = new HashMap<String, Object>();
```

```
params04.put("id", 6);
params04.put("name", "fangjianhua");
params04.put("age", 23);
params04.put("course", "springboot");
System.out.println(params04.toString());
sqlSession.update("studentMapper.updateStudent", params04);
sqlSession.commit(); //不加这一句不会更新
Student stu06 = sqlSession.selectOne("studentMapper.findStudentById", params04.get("id"));
System.out.println(stu06.toString() + '\n');
```

这里对 id 为 6 的数据条目做修改,需要修改的字段包括 name、age 和 course,将它们封装成 HashMap 哈希表键值对参数,传入 SQL 语句。动态 SQL 中的 set 元素和 if 元素生效,会智能地拼接出最终的 SQL 语句 update student set name = fangjianhua,course = springboot where id = 6。读者可以手动将此 SQL 语句输入数据库,会发现执行结果和最终项目代码的执行结果是一致的。

在 MyBatis 中,对 SQL 语句的执行需要调用 SqlSession 对象。查询语句可以直接调用后执行,但是对于更新操作来说,调用后需要手动执行 sqlSession.commit()才会让更新操作生效。这一点需要多加注意。

在 Eclipse 里,右击选择项目作为 Java Application 运行,SQL 语句得到正确执行,输出对应的查询结果,如图 5-5 所示。

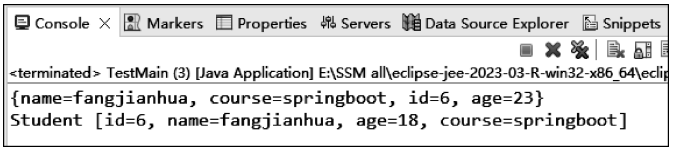

图 5-5 set 元素的代码执行结果示意图

5.2.5 trim 元素

where 元素和 set 元素的使用方法类似,都是智能辅助 SQL 语句的拼装,分别对应查询的 where 子句和更新的 set 子句。除了这两种情况外,SQL 语句的拼装过程中,可能还有其他需要动态、智能调节的情况,这个时候就可以使用更加普适的 trim 元素。

trim 元素是一种辅助拼接 SQL 语句的元素。SQL 语句在代码层面可以看作 String 字符串,当单个 SQL 语句需要和其他子句进行拼接时,可能会产生冗余,或需要删去多余的 String 字符串单词。为了更好地实现拼接,这时候就需要使用 trim 元素来对具体的 SQL 语句进行微调。trim 元素的属性如表 5-2 所示。

表 5-2 trim 元素的属性

属 性	描 述 说 明
prefix	给指定的 SQL 语句增加前缀
prefixOverrides	给指定的 SQL 语句删去前缀
suffix	给指定的 SQL 语句增加后缀
suffixOverrides	给指定的 SQL 语句删去后缀

trim 元素的本质就是对指定的 SQL 语句进行前后缀的删除和添加,因此前面 where 元素和 set 元素实际都是可以使用 trim 元素来实现的,本质上来说,它们都属于 trim 元素

的特殊情况。

下面用一个完整的案例来讲解 trim 元素的具体使用。同样地，这里继续沿用原项目进行更改。

1. 修改映射文件

在映射文件 StudentMapper.xml 中，加入 trim 元素的相关定义代码，具体如下。

```
<!-- 动态 SQL,trim 元素 -->
<select id="findStudent04" parameterType="HashMap" resultType="student">
select * from student
<trim prefix="where" prefixOverrides="and">
<if test = "name!=null and name!=''">
and name = #{name}
</if>
<if test = "course!=null and course!=''">
and course = #{course}
</if>
</trim>
</select>
```

这里的使用场景和前面的 where 元素的类似，都是一个 SQL 查询语句的拼接。当 if 元素判定的条件为真时，where 子句生效，需要添加 where 前缀，并删去 where 和子句之间的 and 前缀。因此在 trim 元素的定义中，将需要添加的前缀设为 prefix="where"，将需要删除的前缀设为 prefixOverrides="and"。

通过分析，这里最终的实现效果和 where 案例中应该是一致的。

2. 修改测试文件

在测试文件 TestMain.java 中，加入 trim 元素的相关使用代码，具体如下。

```
//动态 SQL,trim 元素
HashMap<String, Object> params02 = new HashMap<String, Object>();
params02.put("name", "zhangsan");
params02.put("course", "java");
System.out.println(params02.toString());
Student stu04 = sqlSession.selectOne("studentMapper.findStudent04", params02);
System.out.println(stu04.toString() + '\n');
```

对应的测试代码和 where 元素中的部分也类似，主要就是将查询条件封装成 HashMap 哈希表键值对。通过 trim 元素的使用也可以完成 where 元素中动态、智能拼接 SQL 语句，消除语法错误的功能，最终程序可以成功执行，结果如图 5-6 所示。

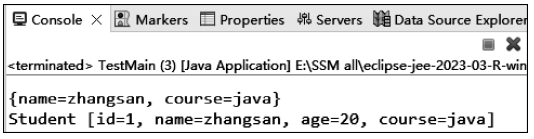

图 5-6　trim 元素的代码执行结果示意图

5.2.6　foreach 元素

foreach 元素是一个和循环相关的动态 SQL 元素。它的作用是对参数的集合进行遍

历,循环调用参数集合中的所有参数。需要强调的是,这个遍历只是将参数集中的参数悉数解析出来拼接到 SQL 语句中,最终执行的是拼接完成后的一个 SQL 语句,而并不是通过循环参数集来多次执行单条 SQL 语句。

在 MyBatis 框架中,foreach 元素支持的参数集的数据类型包括数组、List 链表、Set 集合等。使用 foreach 元素需要对循环的参数集合的解析做好相关定义,包括参数集的下标、元素、间隔符、起止符等,具体的属性如表 5-3 所示。

表 5-3 foreach 元素的属性

属 性	描 述 说 明
collection	需要解析的参数集的参数名称
item	指定循环中的当前元素
index	当前元素在参数集中的位置下标
open、close	包装参数集的起止符号
separator	参数集中各个参数的分隔符

下面是一个完整的 foreach 元素的使用案例。同样地,这里继续沿用原项目进行更改。

1. 修改映射文件

在映射文件 StudentMapper.xml 中,加入 foreach 元素的相关定义代码,具体如下。

```
<!-- 动态 SQL,foreach 元素 -->
< select id="findStudentsByIdList" resultType="student">
select * from student where id in
< foreach item="id" index="index" collection="list" open="(" separator="," close=")">
#{id}
</foreach >
</select >
```

从代码中可以看到,参数集的元素是 id,参数集的变量名是 list,下标是 index,起止符和分隔符分别是“(”、“)”和“,”,实际上这里就可以分析出来参数是一个整型变量的数组 ArrayList。使用了 foreach 元素后,会将参数集合中的 id 值都解析出来,作为一个集合。这里定义的查询语句会通过 where id in 集合来对集合中所有的 id 值进行遍历查询。

2. 修改测试文件

在测试文件 TestMain.java 中,加入 foreach 元素的相关使用代码,具体如下。

```
//动态 SQL,foreach 元素
ArrayList < Integer > list = new ArrayList < Integer >();
list.add(2);
list.add(3);
list.add(6);
System.out.println(list.toString());
List < Student > studentList = sqlSession.selectList("studentMapper.findStudentsByIdList", list);
for (Student student : studentList) {
System.out.println(student.toString());
}
System.out.println('\n');
```

使用 ArrayList < Integer >来定义一个整数类型的数组,作为 id 的集合来存放参数。通

过 list.add()函数将需要查询的 id 值逐一添加到数组中。最后作为查询参数，调用 SQL 语句执行查询。

需要注意的是，由于是参数集合的遍历后执行查询，查询结果可能是多个 Java 对象，因此不能调用 sqlSession.selectOne()，而需要调用它的复数形式 sqlSession.selectList()。对查询结果进行输出的时候，同样需要用循环语句，对复数形式的 Java 对象进行逐一输出打印。最后执行代码，结果如图 5-7 所示。

图 5-7　foreach 元素的代码执行结果示意图

5.2.7　bind 元素

bind 元素的用法相比前面的动态 SQL 元素要简单，它的使用场景相对更加单一。当需要定义一个 OGNL 表达式时，为了防止重复书写表达式，实现类似 Java 变量的先定义后引用的效果，就需要用到 bind 元素。

在很多的 bind 元素的使用场景中，都是通过它来定义一个模糊查询的表达式。

下面通过一个完整的案例来演示其使用方法。同样地，这里继续沿用原项目进行更改。

1. 修改映射文件

在映射文件 StudentMapper.xml 中，加入 bind 元素的相关定义代码，具体如下。

```
<!-- 动态 SQL,bind 元素 -->
<select id="findStudentLike" resultType="student">
<bind name="name_pattern" value="'%'+name+'%'"/>
select * from student where name like #{name_pattern}
</select>
```

这里定义了一个 bind 元素的 OGNL 表达式，名称是 name_pattern。表达式的具体取值，通过 bind 元素的 value 属性来定义，是"'%'+name+'%'"。这样就完成了 bind 元素的定义，在后续要使用的时候，直接在 SQL 语句中引用刚刚定义的 bind 元素的 OGNL 表达式的名称即可。例如，select * from student where name like #{name_pattern}，这里的 #{name_pattern}，MyBatis 框架在执行 SQL 语句的时候，会自动将其解析为刚定义的表达式，也就是"'%'+name+'%'"。

由于这里只是简单演示，代码的使用稍显不合理。但是如果程序中的表达式很多且需要考虑长期维护和代码可读性，启用 bind 元素进行逐一区分，先定义后使用，就显得非常有必要了。

2. 修改测试文件

在测试文件 TestMain.java 中，加入 bind 元素的相关使用代码，具体如下。

```
//动态 SQL,bind 元素,模糊查询
//String nameString = "n";
String nameString = "ng";
List < Student > studentList02 = sqlSession. selectList ( " studentMapper. findStudentLike",
nameString);
for (Student student : studentList02) {
System.out.println(student.toString());
}
System.out.println('\n');
```

这里将 ng 作为参数传入了 mapper 文件中的 SQL 语句,根据 bind 元素的定义,最后得到的 SQL 语句是 select * from student where name like "%ng%"。查询所有姓名中含有"ng"的学生的信息,将所有得到的查询信息打印输出到控制台。最后执行代码,结果如图 5-8 所示。

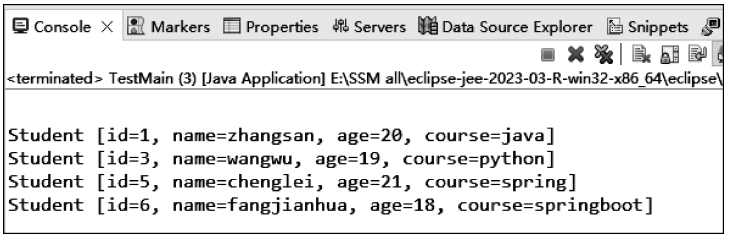

图 5-8 bind 元素的代码执行结果示意图

小结

本章首先介绍了动态 SQL 的应用场景和功能描述,随后详细介绍了动态 SQL 各种元素的相关知识,包括 if、choose、when、otherwise、where、set、trim、foreach、bind 等每种元素的功能介绍和使用方法,并通过使用案例演示了每种元素的具体编程和调试方法。通过本章的学习,读者可以了解动态 SQL 的使用原理,常用的元素和对应的功能,以及在 MyBatis 框架中如何通过编码实现动态 SQL 的应用。

习题

1. (　　)动态元素是多条件分支判断语句,相当于 Java 语句中的 switch case。

   A. choose、when 和 otherwise　　B. trim、where、set

   C. foreach　　D. bind

2. trim 元素的_____属性,可以用于给指定的 SQL 语句删去前缀。

3. 简述动态 SQL 中 bind 元素的作用和使用方法。