第7章 动态SQL

湖南科技大学
计算机科学与工程学院

学习目标/Target

- 掌握MyBatis中动态SQL元素的使用
- 掌握MyBatis的条件查询操作
- 掌握MyBatis的更新操作
- 掌握MyBatis的复杂查询操作

章节概述/ Summary

在实际项目的开发中，开发人员在使用JDBC或其他持久层框架进行开发时，经常需要根据不同的条件拼接SQL语句，拼接SQL语句时还要确保不能遗漏必要的空格、标点符号等，这种编程方式给开发人员带来了非常大的不便，而MyBatis提供的SQL语句动态组装功能，恰能很好地解决这一问题。本章将对MyBatis框架的动态SQL进行详细讲解。

目录/Contents

7.1 动态SQL中的元素

7.1 动态SQL中的元素

- 掌握动态SQL中的常用元素，能够说出MyBatis动态SQL的常用元素有哪些

7.1 动态SQL中的元素

使用动态SQL的好处

动态SQL是MyBatis的强大特性之一，MyBatis采用了功能强大的基于OGNL（Object Graph Navigation Language）的表达式来完成动态SQL。在MyBatis的映射文件中，开发人员可通过动态SQL元素灵活组装SQL语句，这在很大程度上避免了单一SQL语句的反复堆砌，提高了SQL语句的复用性。

7.1 动态SQL中的元素

动态SQL常用元素

元素	说明
<if>	判断语句，用于单条件判断
<choose>（<when>、<otherwise>）	相当于Java中的switch...case...default语句，用于多条件判断
<where>	简化SQL语句中where的条件判断
<trim>	可以灵活地去除多余的关键字
<set>	用于SQL语句的动态更新
<foreach>	循环语句，常用于in语句等列举条件中

7.2 条件查询操作

7.2.1 <if>元素

- 掌握Mybatis中的<if>元素，能够使用<if>编写动态SQL语句

7.2.1 <if>元素

<if>元素的应用

在MyBatis中，<if>元素是最常用的判断元素，它类似于Java中的if语句，主要用于实现某些简单的条件判断。在实际应用中，我们可能会通过某个条件查询某个数据。例如，要查找某个客户的信息，可以通过姓名或者年龄来查找客户，也可以不填写年龄直接通过姓名来查找客户，还可以都不填写而查询出所有客户，此时姓名和年龄就是非必须条件。类似于这种情况，在MyBatis中就可以通过<if>元素来实现。

7.2.1 <if>元素

通过一个具体的案例演示单条件判断下<if>元素的使用，案例具体实现步骤如下。

STEP 01

数据库准备：在名称为mybatis的数据库中，创建一个t_customer表，并插入几条测试数据。

```
USE mybatis;
CREATE TABLE t_customer (
    id int(32) PRIMARY KEY AUTO_INCREMENT,
    username varchar(50),
    jobs varchar(50),
    phone varchar(16));
INSERT INTO t_customer VALUES ('1', 'joy', 'teacher', '13733333333');
INSERT INTO t_customer VALUES ('2', 'jack', 'teacher', '13522222222');
INSERT INTO t_customer VALUES ('3', 'tom', 'worker', '15111111111');
```

7.2.1 <if>元素

STEP 02

POJO类准备：创建持久化类Customer，在类中声明id、username、jobs和phone属性，及属性对应的getter/setter方法。

```
public class Customer {
    private Integer id;   private String username; // 主键ID、客户名称
    private String jobs;  private String phone;      // 职业、电话
    // 省略getter/setter方法
    @Override
    public String toString() {
        return "Customer [id=" + id + ", username=" + username + ",
jobs=" + jobs + ", phone=" + phone + "]"; }
}
```

7.2.1 <if>元素

STEP 03

创建映射文件：创建映射文件CustomerMapper.xml，在映射文件中，根据客户姓名和年龄组合条件查询客户信息，使用<if>元素编写该组合条件的动态SQL。

```
<!– 该xml文件中只列出了if元素的动态SQL-->
<if test="username !=null and username !=  ‘’   “>
      and username like concat('%',#{username}, '%')
</if>
<if test="jobs !=null and jobs !=  ‘’   “>
    and jobs= #{jobs}
</if>
```

7.2.1 <if>元素

STEP 04

修改核心配置文件：在配置文件mybatis-config.xml中，引入CustomerMapper.xml映射文件，将CustomerMapper.xml映射文件加载到程序中。

```
<mapper
resource="com/itheima/mapper/CustomerMapper.xml">
</mapper>
```

7.2.1 <if>元素

STEP 05 创建获取SqlSession对象的工具类。

```
public class MybatisUtils {
        private static SqlSessionFactory sqlSessionFactory = null;
        // 初始化SqlSessionFactory对象
        static {
                try {
                        // 使用MyBatis提供的Resources类加载MyBatis的配置文件
                        Reader reader = Resources.getResourceAsReader("mybatis-config.xml");
                        // 构建SqlSessionFactory工厂
                        sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
        // 获取SqlSession对象的静态方法
        public static SqlSession getSession() {
                return sqlSessionFactory.openSession();
        }
}
```

7.2.1 <if>元素

STEP 06

编写测试类：在测试类MyBatisTest中，编写测试方法findCustomerByNameAndJobsTest()，该方法用于根据客户姓名和职业组合条件查询客户信息列表。

```
public class MyBatisTest {
  @Test
   public void findCustomerByNameAndJobsTest(){
      SqlSession session = MyBatisUtils.getSession();Customer customer = new Customer();
      customer.setUsername( “jack");
      customer.setJobs("teacher");
      List<Customer> customers =
session.selectList("com.itheima.mapper.CustomerMapper.findCustomerByNameAndJobs",customer);
       for (Customer customer2 : customers) {
           System.out.println(customer2);
        }
      session.close();
}
}
```

7.2.2 <choose>、<when>、<otherwise>元素

- 掌握 MyBatis 中的 <choose>、<when>、<otherwise>元素，能够将这三个元素组合使用

7.2.2 <choose>、<when>、<otherwise>元素

<choose><when>otherwise>使用场景

在使用<if>元素时，只要test属性中的表达式为true，就会执行元素中的条件语句，但是在实际应用中，有时只需要从多个选项中选择一个去执行。

例如下面的场景：“当客户名称不为空，则只根据客户名称进行客户筛选；当客户名称为空，而客户职业不为空，则只根据客户职业进行客户筛选。当客户名称和客户职业都为空，则要求查询出所有电话不为空的客户信息。”

针对上面情况，使用<if>元素进行处理是不合适的。MyBatis提供了<choose>、<when>、<otherwise>元素进行处理，这三个元素往往组合在一起使用，作用相当于Java语言中的if...else if...else。

7.2.2 <choose>、<when>、<otherwise>元素

使用<choose>、<when>、<otherwise>元素组合演示上面场景的情况。

STEP 01

在映射文件CustomerMapper.xml中，添加使用<choose>、<when>、<otherwise>元素执行上述情况的动态SQL。

```
<!-- 只展示三个组合元素的部分-->
<choose>
    <when test="username !=null and username !=''">
      and username like concat('%',#{username}, '%')
    </when>
    <when test="jobs !=null and jobs !=''">
      and jobs= #{jobs}
    </when>
    <otherwise>and phone is not null</otherwise>
</choose>
```

7.2.2 <choose>、<when>、<otherwise>元素

STEP 02

在测试类MyBatisTest中，编写测试方法findCustomerByNameOrJobsTest()，该方法用于根据客户姓名或职业查询客户信息列表。

```
public void findCustomerByNameOrJobsTest() {
    SqlSession session=MyBatisUtils.getSession();
    Customer customer=new Customer();
    customer.setUsername("tom");customer.setJobs("teacher");
    List<Customer> customers=session.selectList("com.itheima.mapper"
+ ".CustomerMapper.findCustomerByNameOrJobs",customer);
       for (Customer customer2 : customers) {
                System.out.println(customer2);}
session.close();
}
```

7.2.3 <where>、<trim>元素

- 掌握MyBatis中的<where>、<trim>元素，能够使用<where>、<trim>元素“拼接”动态SQL

7.2.3 <where>、<trim>元素

<where><trim>使用场景

在映射文件中，编写的SQL后面加入了“where 1=1”的条件的话，既保证了where后面的条件成立，又避免了where后面第一个词是and或者or之类的关键字。

例如下面这条Mybatis拼接出的SQL语句是不正确的。

```
select * from t_customer where and username like concat('%',?,
'%') and jobs = #{jobs}
```

上述SQL语句中，where后直接跟的是and，这在运行时会报SQL语法错误，针对这种情况，可以使用MyBatis提供的<where>元素和<trim>元素进行处理。

7.2.2 <where>、<trim>元素

<where>元素

```
<select id="findCustomerByNameAndJobs"
        parameterType="com.itheima.pojo.Customer"
        resultType="com.itheima.pojo.Customer">
    select * from t_customer
    <where>
        <if test="username !=null and username !=''">
            and username like concat('%',#{username}, '%')</if>
        <if test="jobs !=null and jobs !=''">
            and jobs= #{jobs}</if>
    </where></select>
```

7.2.2 <where>、<trim>元素

<where>元素

上述代码配置中，<where>元素会自动判断由组合条件拼装的SQL语句，只有<where>元素内的某一个或多个条件成立时，才会在拼接SQL中加入where关键字，否则将不会添加；即使where之后的内容有多余的“AND”或“OR”，<where>元素也会自动将他们去除。

7.2.2 <where>、<trim>元素

<trim>元素

<trim>元素用于删除多余的关键字，它可以直接实现<where>元素的功能。<trim>元素包含4个属性。

属性	说明
prefix	指定给SQL语句增加的前缀
prefixOverrides	指定SQL语句中要去掉的前缀字符串
suffix	指定给SQL语句增加的后缀
suffixOverrides	指定SQL语句中要去掉的后缀字符串

7.2.2 <where>、<trim>元素

<trim>元素

```
<select id="findCustomerByNameAndJobs"
        parameterType="com.itheima.pojo.Customer"
        resultType="com.itheima.pojo.Customer">
  select * from t_customer
  <trim prefix="where" prefixOverrides="and" >
    <if test="username !=null and username !=''">
      and username like concat('%',#{username}, '%')</if>
    <if test="jobs !=null and jobs !=''">
      and jobs= #{jobs}</if></trim>
</select>
```

7.2.2 <where>、<trim>元素

<trim>元素

上述配置代码中，<trim>元素的作用是去除一些多余的前缀字符串，它的prefix属性代表的是语句的前缀（where），而prefixOverrides属性代表的是需要去除的前缀字符串（SQL中的“AND”或“OR”）。

7.3 更新操作

7.3 更新操作

- 掌握Mybatis的更新操作，能够在代码中进行使用<set>元素进行更新

7.3 更新操作

<set>元素使用场景

在Hibernate框架中，如果想要更新某一个对象，就需要发送所有的字段给持久化对象，然而在实际应用中，大多数情况下都是更新某一个或几个字段。如果更新的每一条数据都要将其所有的属性都更新一遍，那么执行效率是非常差的。为了解决更新数据的效率问题，MyBatis提供了<set>元素。<set>元素主要用于更新操作，它可以在动态SQL语句前输出一个SET关键字，并将SQL语句中最后一个多余的逗号去除。<set>元素与<if>元素结合可以只更新需要更新的字段。

7.3 更新操作

通过一个案例演示如何使用<set>元素更新数据库的信息，案例具体步骤如下。

STEP 01

在映射文件CustomerMapper.xml中，添加使用<set>元素执行更新操作的动态SQL。

```
<update id="updateCustomerBySet”
parameterType="com.itheima.pojo.Customer">update t_customer
    <set>
        <if test="username !=null and username !=''">
            username=#{username},</if>
        <if test="jobs !=null and jobs !=''">  jobs=#{jobs},</if>
        <if test="phone !=null and phone !=''">phone=#{phone},</if>
    </set> where id=#{id}
</update>
```

7.3 更新操作

STEP 02

编写测试方法updateCustomerBySetTest()。

```
public void updateCustomerBySetTest() {
    SqlSession sqlSession = MyBatisUtils.getSession();
    Customer customer = new Customer();        customer.setId(3);
    customer.setPhone("13311111234");
    int rows = sqlSession.update("com.itheima.mapper"
            + ".CustomerMapper.updateCustomerBySet", customer);
    if(rows > 0) {System.out.println("您成功修改了"+rows+"条数据！");
    } else { System.out.println("执行修改操作失败！！！");
    }sqlSession.commit();sqlSession.close();
}
```

<set>元素字段非空

在映射文件中使用<set>元素和<if>元素组合进行update语句动态SQL组装时，如果<set>元素内包含的内容都为空，则会出现SQL语法错误。因此，在使用<set>元素进行字段信息更新时，要确保传入的更新字段不能都为空。

使用<trim>元素更新

除了使用<set>元素外，还可以通过<trim>元素来实现更新操作。其中， <trim>元素的prefix属性指定要添加的<trim>元素所包含内容的前缀为set，suffixOverrides属性指定去除的<trim>元素所包含内容的后缀为逗号 。

7.4 复杂查询操作

7.4.1 <foreach>元素中的属性

- 掌握<foreach>元素，能够熟练使用<foreach>元素以及它的属性

7.4.1 <foreach>元素中的属性

<foreach>元素的属性

属性	说明
item	表示集合中每一个元素进行迭代时的别名。该属性为必选。
index	在List和数组中，index是元素的序号，在Map中，index是元素的key。该属性可选。
open	表示foreach语句代码的开始符号，一般和close=“)”合用。常用在in条件语句中。该属性可选。
separator	表示元素之间的分隔符，例如，在条件语句中，separator=“,”会自动在元素中间用“,”隔开，避免手动输入逗号导致SQL错误，错误示例如in(1,2,)。该属性可选。
close	表示foreach语句代码的关闭符号，一般和open="("合用。常用在in条件语句中。该属性可选。
collection	用于指定遍历参数的类型。注意，该属性必须指定，不同情况下，该属性的值是不一样的。

7.4.1 <foreach>元素中的属性

<collection>属性的取值

在遍历参数时，<collection>属性的值是必须指定的。不同情况下，该属性的取值也是不一样的，主要有以下三种情况：List类型、数值类型、Map类型。

7.4.1 <foreach>元素中的属性

List类型

若入参为单参数且参数类型是一个List，collection属性值为list。

7.4.1 <foreach>元素中的属性

数组类型

若入参为单参数且参数类型是一个数组，collection属性值为array。

7.4.1 <foreach>元素中的属性

Map类型

若传入参数为多参数，就需要把参数封装为一个Map进行处理，collection属性值为Map。若传入参数为多参数，就需要把参数封装为一个Map进行处理，collection属性值为Map。

7.4.2 <foreach>元素迭代数组

- 掌握<foreach>元素对数组的迭代，能够使用<foreach>元素对数组进行迭代

7.4.2 <foreach>元素迭代数组

<foeeach>实现入参为数组类型的遍历

例如，要从数据表t_customer中查询出id为1、2、3的客户信息，就可以利用数组作为参数，存储id的属性值1、2、3，并通过<foreach>元素迭代数组完成客户信息的批量查询操作。

7.4.2 <foreach>元素迭代数组

<foreach>元素迭代数组的实现具体如下。

STEP 01

在映射文件CustomerMapper.xml中，添加使用<foreach>元素迭代数组执行批量查询操作的动态SQL。

```
<select id="findByArray" parameterType="java.util.Arrays"
        resultType="com.itheima.pojo.Customer">select * from t_customer where
id in
    <foreach item="id" index="index" collection="array"
             open="(" separator="," close=")">     #{id}
    </foreach>
select>
```

7.4.2 <foreach>元素迭代数组

STEP 02

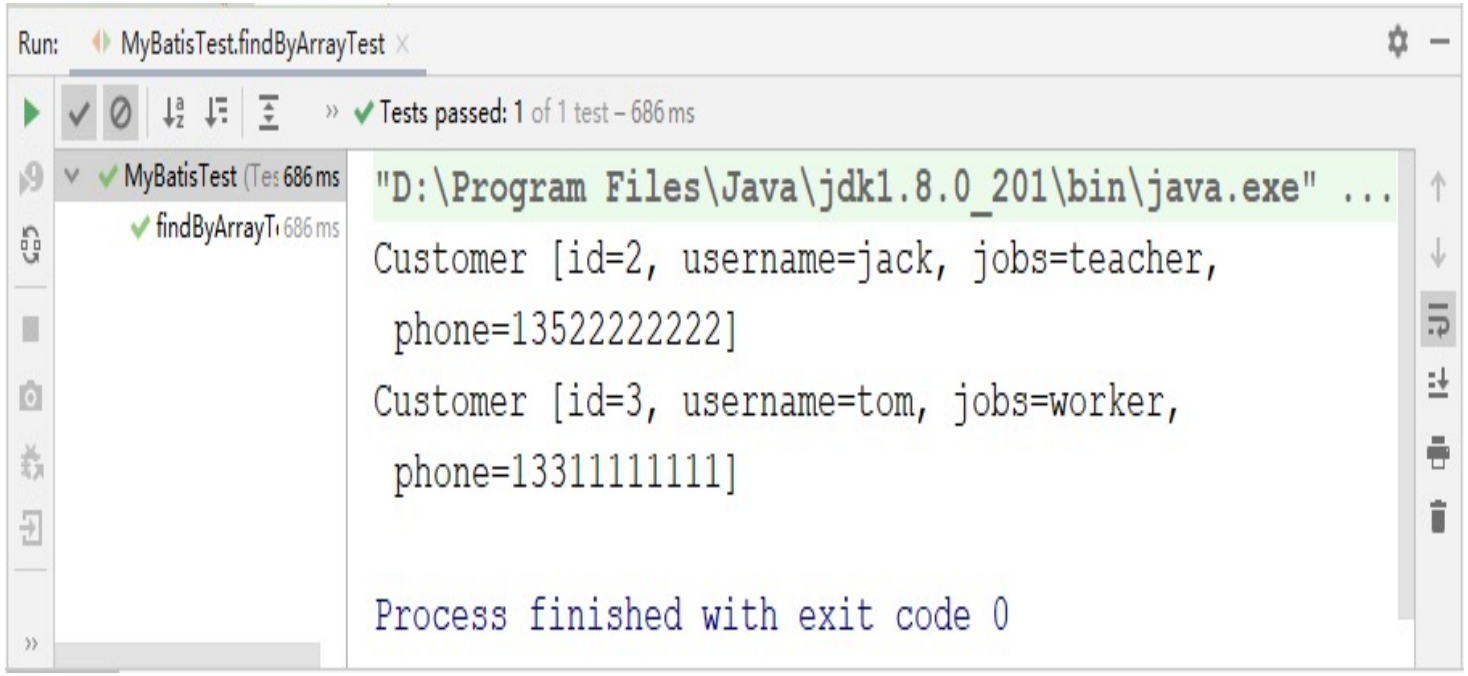
在测试类MyBatisTest中，编写测试方法findByArrayTest()方法，实现客户信息的批量查询。

```
public void findByArrayTest() {
    SqlSession session = MyBatisUtils.getSession(); // 获取SqlSession
    Integer[] roleIds = {2,3}; // 创建数组，封装查询id
    // 执行SqlSession的查询方法，返回结果集
    List<Customer> customers = session.selectList("com.itheima.mapper"
            + ".CustomerMapper.findByArray", roleIds);
    for (Customer customer : customers) {System.out.println(customer);}
    session.close();
}
```

7.4.2 <foreach>元素迭代数组

STEP 03

执行MyBatisTest测试类的findByArrayTest()方法，控制台会输出结果。



7.4.3 <foreach>元素迭代List

- 掌握<foreach>元素对List的迭代，能够使用<foreach>元素对List进行迭代

7.4.3 <foreach>元素迭代List

<foreach>元素迭代List的实现步骤具体如下。

STEP 01

在映射文件CustomerMapper.xml中，添加使用<foreach>元素迭代List集合执行批量查询操作的动态SQL。

```
<select id="findByList" parameterType="java.util.Arrays"
        resultType="com.itheima.pojo.Customer">
    select * from t_customer where id in
    <foreach item="id" index="index" collection="list"
             open="(" separator="," close=")">
        #{id}
    </foreach>
</select>
```

7.4.3 <foreach>元素迭代List

STEP 02

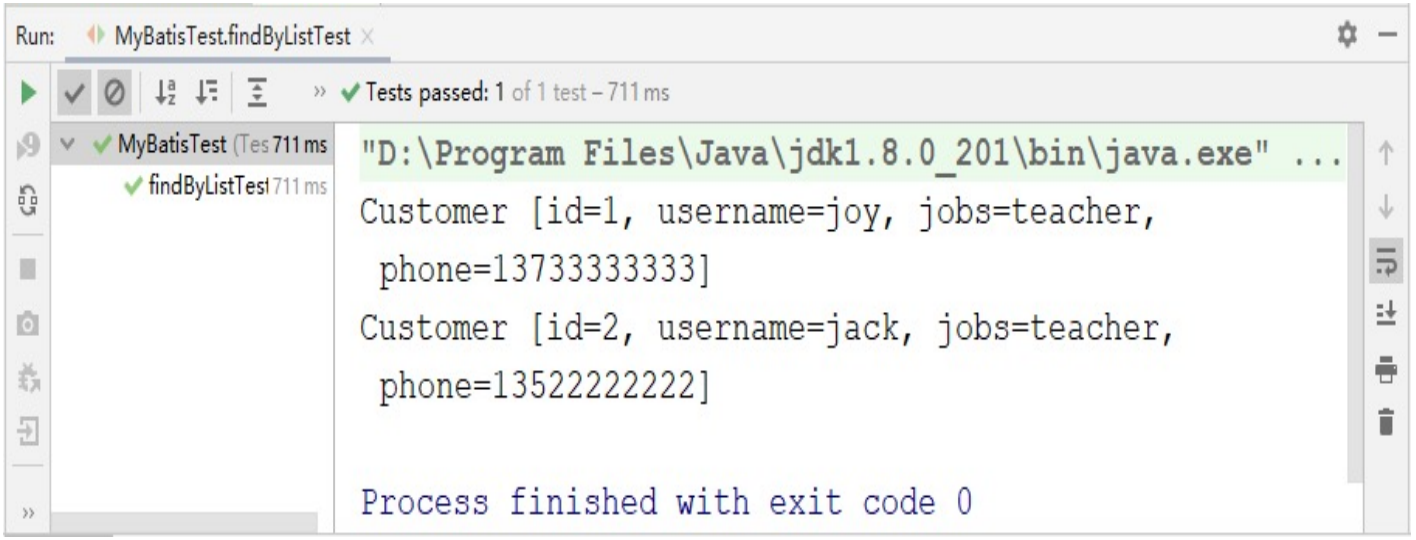
在测试类MyBatisTest中，编写测试方法findByListTest()，用于批量查询客户信息。

```
public void findByListTest() {
    SqlSession session = MyBatisUtils.getSession();
    List<Integer> ids=new ArrayList<Integer>();
    ids.add(1);  ids.add(2);
    List<Customer> customers =
session.selectList("com.itheima.mapper.CustomerMapper.findByList", ids);
    for (Customer customer : customers) {System.out.println(customer);
    }
session.close();
}
```

7.4.3 <foreach>元素迭代List

STEP 03

执行MyBatisTest测试类的findByListTest()方法，控制台会输出结果。



7.4.4 <foreach>元素迭代Map

- 掌握<foreach>元素对Map集合的迭代，能够使用<foreach>元素对Map进行迭代

7.4.4 <foreach>元素迭代Map

下面通过一个案例演示如何使用<foreach>元素迭代Map集合，实现多参数入参查询操作，案例具体实现步骤如下。

STEP 01

在映射文件CustomerMapper.xml中，添加使用<foreach>元素迭代Map集合执行批量查询操作的动态SQL。

```
<select id="findByMap" parameterType="java.util.Map"
        resultType="com.itheima.pojo.Customer">
    select * from t_customer where jobs=#{jobs} and id in
    <foreach item="roleMap" index="index" collection="id" open="("
             separator="," close=")"> #{roleMap}
    </foreach>
</select>
```

7.4.4 <foreach>元素迭代Map

STEP 02

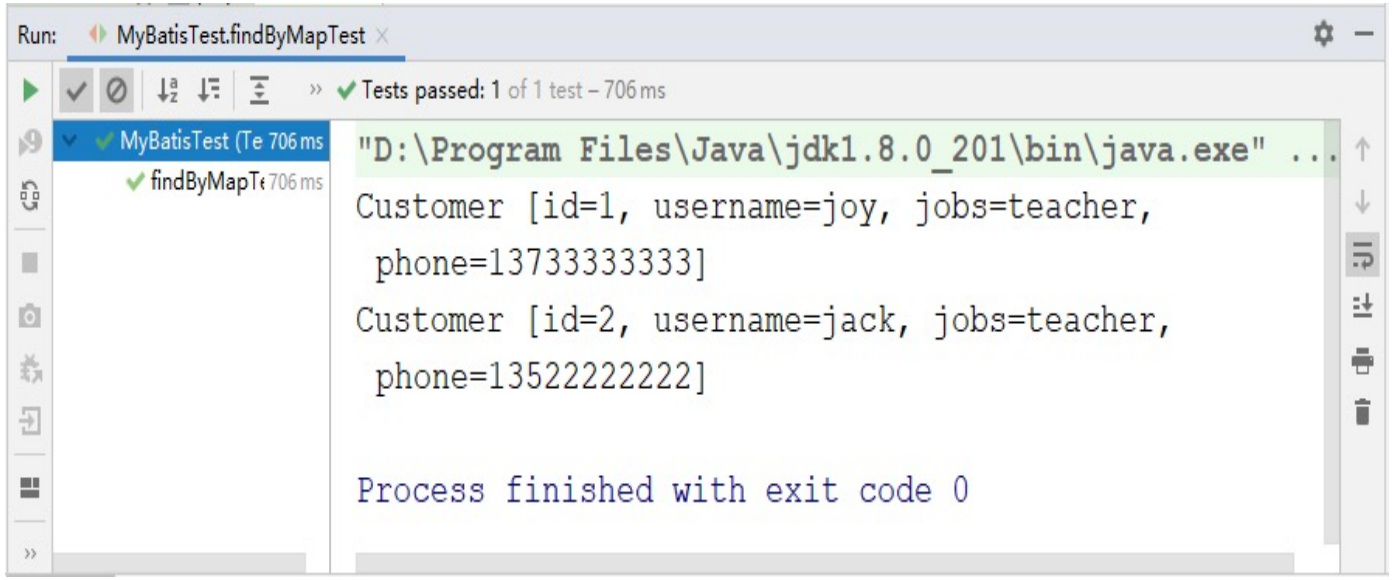
在测试类MyBatisTest中，编写测试方法findByMapTest()，用于批量查询客户信息。

```
public void findByMapTest() {
        SqlSession session = MyBatisUtils.getSession();
        List<Integer> ids=new ArrayList<Integer>();
        ids.add(1); ids.add(2); ids.add(3);
        Map<String,Object> conditionMap = new HashMap<String, Object>();
        conditionMap.put(  “id",ids); conditionMap.put("jobs","teacher");
        List<Customer> customers = session.selectList("com.itheima.mapper"
            + ".CustomerMapper.findByMap", conditionMap);
        for (Customer customer : customers) { System.out.println(customer);}
       session.close();
 }
```

7.4.4 <foreach>元素迭代Map

STEP 03

执行MyBatisTest测试类的findByMapTest()方法，控制台会输出结果。



7.5 案例：学生信息查询系统

7.5 案例：学生信息查询系统

- 完成一个学生信息查询系统，能够实现多条件查询、单条件查询

7.5 案例：学生信息查询系统

学生信息查询系统的功能

本章对MyBatis的动态SQL进行了详细讲解，包括使用动态SQL进行条件查询、更新以及复杂查询操作。本案例要求利用本章所学知识完成一个学生信息查询系统，该系统要求实现2个以下功能。

7.5 案例：学生信息查询系统

多条件查询

当用户输入的学生姓名不为空，则只根据学生姓名进行学生信息的查询；

当用户输入的学生姓名为空，而学生专业不为空，则只根据学生专业进行学生的查询；

7.5 案例：学生信息查询系统

单条件查询

查询出所有id值小于5的学生的信息；

7.5 案例：学生信息查询系统

多条件查询

STEP 01

项目搭建：创建一个名称为mybatis-demo03的项目，项目的具体搭建过程请参考1.3节。

7.5 案例：学生信息查询系统

STEP 02

数据准备：在名称为mybatis的数据库中，创建一个dm_student表，并插入几条测试数据。

```
USE mybatis;
CREATE TABLE dm_student(
    id int(32) PRIMARY KEY AUTO_INCREMENT,
    name varchar(50),
    major varchar(50),
    sno varchar(16) );
# 插入7条数据，其他省略
INSERT INTO dm_student VALUES ('1', '张三', '数学', '10001');
```

7.5 案例：学生信息查询系统

STEP 03

POJO类准备：创建持久化类Student，在类中声明id、name、major和sno属性，以及属性对应的getter/setter方法。

```
public class Student {// 定义变量主键id，姓名name，专业major，学号sno
    private Integer id;  private String name;
    private String major;
    private String sno;
    // 省略getter/setter方法
    @Override
    public String toString() {
        return "Student{" + "id=" + id + ", name=" + '" + name + ",
major=" + major + ", sno=" + sno + '}';}}
```

7.5 案例：学生信息查询系统

STEP 04

创建映射文件：创建映射文件StudentMapper.xml，编写根据学生姓名和专业组合成的条件查询学生信息的动态SQL。

```
<mapper namespace="com.itheima.mapper.StudentMapper">
  <select id= “findStudentByNameAndMajor”
parameterType= “com.itheima.pojo.Student” resultType="com.itheima.pojo.Student">
        select * from dm_student where 1=1        <choose>
           <when test="name !=null and name !=''">
              and name like concat('%',#{name}, '%')</when>
           <when test="major !=null and major !=''"> and major= #{major}</when>
       <otherwise> and sno is not null</otherwise> </choose>
   </select>
</mapper>
```

7.5 案例：学生信息查询系统

STEP 05

修改mybatis-config.xml核心配置文件：在mybatis-config.xml映射文件的<mappers>元素下添加StudentMapper.xml映射文件路径的配置，用于将StudentMapper.xml映射文件加载到程序中。具体配置代码如下。

```
<mapper resource="com/itheima/mapper/StudentMapper.xml"/>
```

7.5 案例：学生信息查询系统

STEP 06

编写MyBatisUtils工具类：创建MyBatisUtils工具类，该类用于封装读取配置文件信息的代码。

```
public class MyBatisUtils {
    private static SqlSessionFactory sqlSessionFactory = null;
    static {
        try {
        Reader reader = Resources.getResourceAsReader( “mybatis-config.xml” );
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
                } catch (Exception e) {e.printStackTrace();}          }
     public static SqlSession getSession() {
         return sqlSessionFactory.openSession();}
}
```

7.5 案例：学生信息查询系统

STEP 07

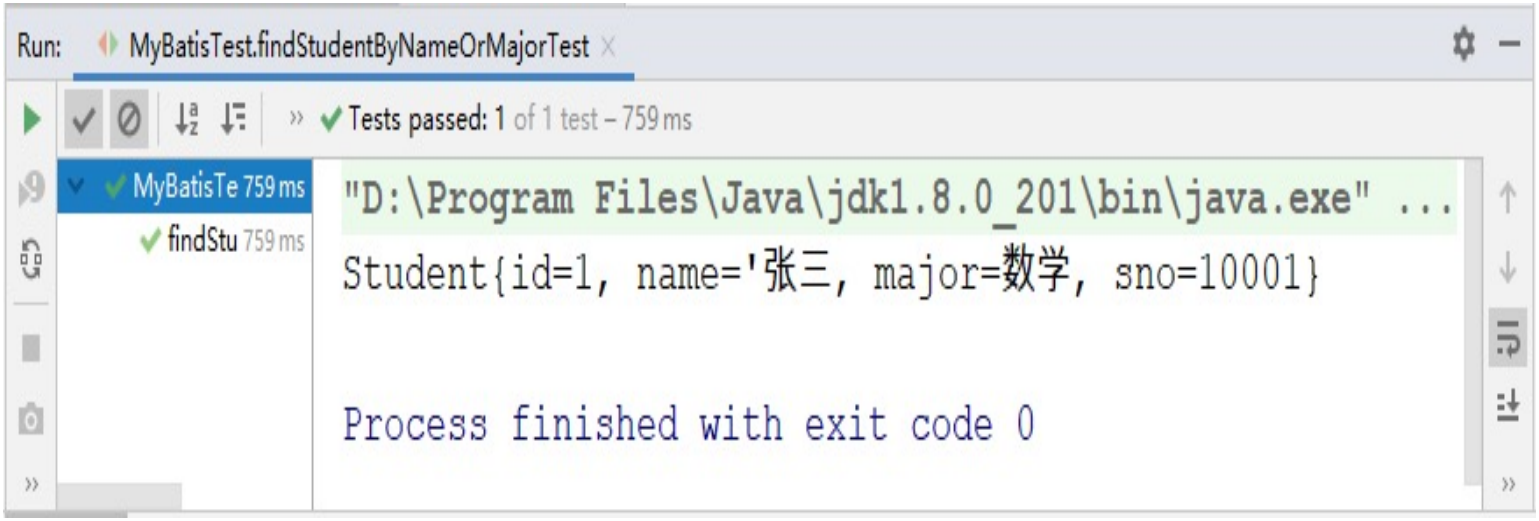
编写测试方法：在测试类MyBatisTest中，编写测试方法findStudentByNameOrMajorTest()，该方法用于根据学生姓名或专业查询学生信息。

```
public void findStudentByNameOrMajorTest() {
    SqlSession session=MyBatisUtils.getSession();
    Student student=new Student();
    student.setName("张三");
    student.setMajor("英语");
    List<Student> students = session.selectList("com.itheima.mapper"
            + ".StudentMapper.findStudentByNameAndMajor",student);
    for (Student student2 : students) {System.out.println(student2);}
    session.close();
}
```

7.5 案例：学生信息查询系统

STEP 08

查看运行结果：执行测试类MyBatisTest的findStudentByNameOrMajorTest()方法，控制台会输出结果。



7.5 案例：学生信息查询系统

多条件查询案例结果分析

由输出结果分析可知，在查询学生信息时，虽然同时传入了姓名和专业两个查询条件，但MyBatis所生成的SQL只是动态组装了学生姓名条件进行查询。如果将案例代码中的student.setName(“张三”)删除或者注释掉，使SQL只按专业进行查询。再次执行findStudentByNameOrMajorTest ()方法，MyBatis生成的SQL会组装学生职业进行条件查询，同样查询出了学生信息。如果学生姓名和专业都为空， MyBatis的SQL便会组装<otherwise>元素中的SQL片段进行条件查询。

7.5 案例：学生信息查询系统

单条件查询

STEP 01

修改映射文件：在映射文件StudentMapper.xml中的<mapper>元素下，编写查询所有id值小于5的学生信息的动态SQL。

```
<select id="findByList" parameterType="java.util.List"
        resultType="com.itheima.pojo.Student">
                        select * from dm_student where id in
    <foreach item="id" index="index" collection="list"
             open="(" separator="," close=")">
        #{id}
    </foreach>
</select>
```

7.5 案例：学生信息查询系统

STEP 02

编写测试方法 ：在测试类MyBatisTest中，编写测试方法findByListTest()。

```
public void findByListTest() {
    SqlSession session = MyBatisUtils.getSession();
    List<Integer> ids=new ArrayList<Integer>();
    for(int i =1;i<5;i++){
            ids.add(i);}
    List<Student> students = session.selectList("com.itheima.mapper"
        + ".StudentMapper.findByList", ids);
    for (Student student : students) { System.out.println(student);}
    session.close();
}
```

本章小结

本章主要讲解了动态SQL相关知识。首先讲解了动态SQL中的元素；其次讲解了条件查询操作，包括<if>元素、<choose>元素、<when>元素、<otherwise>元素、<where>元素和<trim>元素的使用；然后讲解了更新操作；最后讲解了复杂查询操作。通过本章的学习，读者可以了解常用动态SQL元素的主要作用，并能够掌握这些元素在实际开发中的应用。在MyBatis框架中，这些动态SQL元素十分重要，熟练的掌握它们能够极大地提高开发效率。