MyBatis

第四讲
MyBatis动态SQL

知识准备

- 映射器接口中方法的参数情况
  - 当根据单个条件查询时，可以直接以该条件为参数
  - 当根据多个条件查询时，可以将JavaBean作为参数
  - 当根据多个条件查询且多个条件不属于某一个JavaBean时，可以Map类型作为参数，且通过Map中的key值来映射XML中SQL使用的参数的名字

知识准备

- 映射器接口中方法的参数情况
  - 如果要使用多个参数，必须使用@Param注解指定参数名

```
public List<User> findUserByNameAndPassword(
                @Param("name")String name,
                @Param("password")String password);
```

知识准备

- 映射器接口中方法的参数情况
  - 映射文件中可以引用@Param注解中指定的参数名称

```
<select id="findUserByNameAndPassword"
        resultMap="userMap">
    select * from user where
    user_name = #{name}
    and password = #{password}
</select>
```

MyBatis动态SQL

- MyBatis 的一个强大的特性之一就是它的动态 SQL 能力。

  也就是可以根据不同的条件拼接SQL语句。以下是动态SQL在XML中支持的几种元素

  - if
  - choose (when, otherwise)
  - trim (where, set)
  - foreach
  - bind

目录

if

- if元素通常用于where语句中，通过判断参数值来决定是否使用某个查询条件
- 假设现在有一个需求：实现一个用户管理高级查询功能，根据用户输入的条件去检索用户信息

if

- 创建实体类User和映射器接口

```
//实体类
public class User {
    private Integer id;
    private String userName;
    private String password;
    private String phone;
    private String email;
    //省略构造方法
    //省略getter、setter方法
}
```

```
//映射器接口
public interface UserMapper {

  public User findUserByPhoneOrEmail
          (Map<String, String> map);
}
```

if

- 不使用动态SQL时，实现上述需求的映射文件如下：

```
<mapper namespace="com.mybatis.mapper.UserMapper">
    <select id="findUserByPhoneOrEmail"
            resultType="com.mybatis.mapper.User">
        select id, user_name userName,
            password, phone, email
        from user
        where phone = #{phone} and email = #{email}
    </select>
</mapper>
```

if

- 调用该方法时，传入的Map类型的实际参数中，必须存在phone和email这两个key值

```
Map<String, String> user = new HashMap<>();
user.put("phone", "123456789");
user.put("email", "test@163.com");
userMapper.findUserByPhoneOrEmail(user);
```

if

- 不使用动态SQL时，只有同时输入phone和email两个条件时，才能查出正确结果，此时需要if元素来解决这个问题

```
<select id="findUserByPhoneOrEmail" resultType="User">
     select id, user_name userName, password, phone, email
     from user where 1 = 1
     <if test="phone != null and !phone.equals('')">
         and phone = #{phone}
     </if>
     <if test="email != null and !email.equals('')">
         and email = #{email}
     </if>
 </select>
```

if

- test属性的值为符合ONGL（Object-Graph Navigation Language对象图导航语言）要求的判断表达式，表达式中能够出现方法调用，但是结果只能为true或者false
- MyBatis常用的ONGL表达式中可以使用的操作符如下：

Java常用操作符	+、-、*、/、%、==、!=、&&、||、!、点、[ ]
自己特有的操作符	or、and、eq、neq、lt、lte、gt、gte、not

if

- 如果不想让where条件中出现“1=1”这种表达式时，就需要使用where元素
- where元素的作用：如果该元素中有内容，就在生成SQL语句时加上where条件，如果该元素的内容以AND或者OR开头，就去除这两个单词

if

- 上述例子改用where元素实现如下：

```
<select id="findUserByPhoneOrEmail" resultType="User">
    select id, user_name userName, password, phone, email
    from user
    <where>
        <if test="phone != null and !phone.equals('')">
            and phone = #{phone}
        </if>
        <if test="email != null and !email.equals('')">
            and email = #{email}
        </if>
    </where>
 </select>
```

if

- 练习1：使用if元素实现动态列更新，即更新用户信息时，只更

```
<update id="updateUserByIdSelective">
      update user set
      <if test="userName != null and !userName.equals('')">
          user_name = #{userName},
      </if>
      ……
      <if test="email != null and !email.equals('')">
          email = #{email},
      </if>
      id = #{id}
      where id = #{id}
 </update>
```

if

- 实现动态列更新时，可以使用set元素
- set元素作用：当该元素有内容时，生成SQL语句时就加上set语句；当set元素的内容以逗号结尾时，去掉逗号

```
<update id="updateUserByIdSelective">
    update user
    <set>
       <if test="userName != null and !userName.equals('')">
       user_name = #{userName},
       </if>
       <if test="password != null and !password.equals('')">
       password = #{password},
       </if>
       <if test="phone != null and !phone.equals('')">
       phone = #{phone},
       </if>
       <if test="email != null and !email.equals('')">
       email = #{email},
       </if>
    </set>
    where id = #{id}
</update>
```

if

- 练习2：插入用户信息时，如果邮箱为空，就使用数据库中设置的默认值，而不要传入空值

```
<insert id="insertUser">
     insert into user(user_name, password, phone
     <if test="email != null and !email.equals('')">
     ,email
     </if>
     ) values(#{userName}, #{password}, #{phone}
     <if test="email != null and !email.equals('')">
     ,#{email}
     </if>)
 </insert>
```

目录

choose

- choose元素中包含when和otherwise两个子元素
  - 一个choose元素中至少有一个when子元素
  - 一个choose元素中可以包含0个或1个otherwise子元素
- 使用choose元素可以实现类似if...else...的逻辑

choose

```
<select id="findUserByIdOrName"
        resultType="com.mybatis.entity.User">
    select id, user_name userName, password, phone, email
    from user where 1 = 1
    <choose>
        <when test="id != null and id != ''">
        and id = #{id}
        </when>
        <when test="userName != null and userName != ''">
        and user_name = #{userName}
        </when>
        <otherwise>
        and 1 = 2
        </otherwise>
    </choose>
 </select>
```

目录

trim

- trim元素可以实现where和set元素的功能
- trim元素对应的where功能的实现

```
<trim prefix="where" prefixOverrides="AND |OR ">……</trim>
```

- trim元素对应的set功能的实现

```
<trim prefix="set" suffixOverrides=",">……</trim>
```

trim

- trim元素的属性，都在trim元素包含内容时起作用：
  - prefix：给内容增加该属性指定的前缀
  - prefixOverrides：把内容中匹配的前缀字符串去掉
  - suffix：给内容增加该属性指定的后缀
  - suffixOverrides：把内容中匹配的后缀字符串去掉

目录

foreach

- 思考：如何根据传入的用户id集合查询出所有符合条件的用户？
- foreach可以对数组、Map或实现了Iterable接口（如：List、Set）的对象进行遍历
- 映射接口中添加如下方法

```
public List<User> findUserByIdList(List<Integer> idList);
```

foreach

- 映射文件如下：

```
<select id="findUserByIdList"
        resultType="com.mybatis.entity.User">
    select id, user_name userName, password, phone, email
    from user
    where id in
    <foreach collection="list" open="(" close=")"
             separator="," item="id" index="i">
        #{id}
    </foreach>
 </select>
```

foreach

- foreach各个属性的含义

属性	含义
collection	必填，值为要迭代循环的属性名
item	变量名，值为从迭代对象中取出的每一个值
index	索引的属性名，在遍历集合、数组时为当前索引值，当遍历Map时，该值为Map的key值
open	整个循环内容开头的字符串
close	整个循环内容结尾的字符串
separator	每次循环的分隔符

foreach

- collection属性的取值有多种情况
  - 当参数为List集合时，collection取值为list
  - 当参数为数组时，collection取值为array
  - 当参数为Map类型时，collection取值默认情况下为_parameter，也可以使用@Param注解指定名字，collection取值为该注解指定的名字

foreach

- foreach还能实现批量插入。目前支持批量插入的数据库有DB2、SQL Server 2008及以上版本、MySQL等
- 在映射接口中添加批量插入的方法

```
public int insertList(List<User> users);
```

foreach

- 使用foreach实现批量插入功能

```
<insert id="insertList">
     insert into user(user_name, password, phone, email)
     values
     <foreach collection="list" item="user" separator=",">
     (#{user.userName}, #{user.password},
     #{user.phone}, #{user.email})
     </foreach>
 </insert>
```

foreach

- MySQL数据库还支持批量插入后，返回所有记录的主键的操作，MyBatis要想实现此功能，需要修改XML映射文件

```
<insert id="insertList" useGeneratedKeys="true" keyProperty="id">
     insert into user(user_name, password, phone, email)
     values
     <foreach collection="list" item="user" separator=",">
     (#{user.userName}, #{user.password},
     #{user.phone}, #{user.email})
     </foreach>
 </insert>
```

foreach

- foreach实现动态更新，当参数是Map类型时，foreach元素的index属性值对应的不是索引值，而是Map中的key
- 实现通过指定的列名和对应的值去更新数据，在映射接口中添加如下方法

```
public int updateByMap(Map<String, Object> map);
```

foreach

- 这里key作为列名，对应的值作为该列的值，通过foreach将需要更新的字段拼接在SQL语句中

```
<update id="updateByMap">
    update user set
    <foreach collection="_parameter" item="value"
        index="key" separator=",">
        ${key} = #{value}
    </foreach>
    where id = #{id}
</update>
```

foreach

- 上述方法在调用时，Map类型的实参中的key值要与表中字段名对应
- 也可以将映射器接口中方法的参数添加@Param注解

```
public int updateByMap(@Param("map")Map<String,
    Object> map);
```

foreach

- 映射文件修改如下：

```
<update id="updateByMap">
    update user set
    <foreach collection="map" item="value"
        index="key" separator=",">
        ${key} = #{value}
    </foreach>
    where id = #{map.id}
</update>
```

扩展：bind用法

- bind可以使用OGNL表达式创建一个变量并将其绑定到上下文中。如：在模糊查询中可以使用

```
public List<User> findUserLike(
                        @Param("name")String name);
```

扩展：bind用法

- bind可以使用OGNL表达式创建一个变量并将其绑定到上下文中。如：在模糊查询中可以使用

```
<select id="findUserLike" resultMap="userMap">
    <bind name="userName" value="'%' + name + '%'"/>
    select * from user
        where user_name like #{userName}
</select>
```

本章小结

- 映射接口中参数问题
- 动态SQL在XML中支持的几种元素
  - if
  - choose (when, otherwise)
  - trim (where, set)
  - foreach
  - bind

THANK YOU