1、动态sql的概念

```
--动态sql：
    就是说sql语句是动态变化的，映射文件中的sql语句根据条件让mybatis获取不同的sql语句
    主要是where的部分进行动态化
--实现动态sql
    通过使用mybatis中的标签来实现sql语句的动态化，标签有<if>、<where>、<foreach>
```

2、标签，是判断条件的

```
语法如下：
    <if test="判断java对象的属性值">
        待拼接的sql语句
    </if>
在idea-maven-mybatis工程中添加新模块进行测试，模块项目名为：d-maven-dynamic-sql-
mybatis04
<!--注意标签<if>的动态sql语句中接口方法中的参数一定要是Java对象-->
```

UserDao接口方法如下：

```
package com.studymyself.dao;

import com.studymyself.entity.User;

import java.util.List;

//接口操作t_user表
public interface UserDao {

    //通过if动态sql语句查询数据,参数必须是Java对象
    public List<User> selectByIf(User user);

}
```

sql映射文件如下：

```
<?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="com.studymyself.dao.UserDao">

    <select id="selectByIf" resultType="User">
        select id,loginName,loginPwd,realName from t_user
        where
        <!--上面表示主语句-->
        <!--if表示这个标签中传递过来的Java对象的id字段值大于0
            就将if标签中的sql语句块拼接到主语句中去-->
```

```
        <if test="id > 0">
             id=#{id}
        </if>
        <!--传过来的java对象中的realName值不为空或不是空字符串
            就将if标签中的sql语句块拼接到主语句中去-->
        <if test="loginName!= null or loginName !='' ">
            or realName = #{realName}
        </if>
    </select>

</mapper>
```

测试程序如下

```
  @Test
    public void testSelectByIf(){

        //获取SqlSession对象
        SqlSession sqlSession = MyBatisUtil.getSqlSession();
        //通过SqlSession对象中的方法获取实现类对象
        UserDao userDao = sqlSession.getMapper(UserDao.class);
        User user = new User();
        user.setId(1);
        user.setRealName("钟荣杰");
        List<User> users = userDao.selectByIf(user);
        for (User user1:
             users) {
            System.out.println(user1);
        }


    }
```

从上面的语句中我们可以知道有这样的情况：

第一个if标签中条件不成立，但第二个条件成立将sql语句块添加到主语句时会拼接成错误的sql语句块：

```
select id,loginName,loginPwd,realName from t_user where or realName = ?
```

避免这种情况，我们可以在where后添加一个 1=1 的条件，然后第一个if标签中的语句块前面都加连接关键字or and

```
 <select id="selectByIf" resultType="User">
        select id,loginName,loginPwd,realName from t_user
        where 1=1
        <if test="id > 0">
            or id=#{id}
        </if>
        <if test="realName!= null or realName !='' ">
            or realName = #{realName}
        </if>
    </select>
```

上面在where后边添加恒等条件的方式可以用标签解决

3、标签

<where>标签用来代替sql主语句中where关键的位置，用来包含多个if标签，当多个if只要有一个成立，该标签就字自动生成一个where关键字，还会将第一个拼接到sql主语句的if标签的sql语句块的or、and连接关键字去掉。

在dao接口中添加方法，如下

```
package com.studymyself.dao;

import com.studymyself.entity.User;

import java.util.List;

//接口操作t_user表
public interface UserDao {

    //通过if动态sql语句查询数据,参数必须是Java对象
    public List<User> selectByIf(User user);

    //通过if和where动态sql语句查询数据,参数必须是Java对象
    public List<User> selectByWhere(User user);
}
```

sql映射文件如下

```
<?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="com.studymyself.dao.UserDao">


    <select id="selectByIf" resultType="User">
        select id,loginName,loginPwd,realName from t_user
        where
        <!--上面表示主语句-->
        <!--if表示这个标签中传递过来的Java对象的id字段值大于0
            就将if标签中的sql语句块拼接到主语句末端中去-->
        <if test="id > 0">
             id=#{id}
        </if>
        <!--传过来的java对象中的realName值不为空或不是空字符串
            就将if标签中的sql语句块拼接到主语句末端中去-->
        <if test="realName!= null or realName !='' ">
            or realName = #{realName}
        </if>
    </select>

    <select id="selectByWhere" resultType="User">
        select id,loginName,loginPwd,realName from t_user
        <where>
            <if test="id > 0">
                or id=#{id}
            </if>
```

```
            <if test="realName!= null or realName !='' ">
                or realName = #{realName}
            </if>
        </where>

    </select>

</mapper>
```

测试程序如下

```
@Test
    public void testSelectByWhere(){

        //获取SqlSession对象
        SqlSession sqlSession = MyBatisUtil.getSqlSession();
        //通过SqlSession对象中的方法获取实现类对象
        UserDao userDao = sqlSession.getMapper(UserDao.class);
        User user = new User();
        user.setId(1);
        user.setRealName("钟荣杰");
        List<User> users = userDao.selectByWhere(user);
        for (User user1:
                users) {
            System.out.println(user1);
        }

    }
```

上面测试程序的执行日志如下

```
Opening JDBC Connection
Created connection 2114444063.
Setting autocommit to false on JDBC Connection
[com.mysql.jdbc.JDBC4Connection@7e07db1f]
==>  Preparing: select id,loginName,loginPwd,realName from t_user WHERE id=? or
realName = ?
==> Parameters: 1(Integer), 钟荣杰(String)
<==    Columns: id, loginName, loginPwd, realName
<==        Row: 1, xxpiaozhiyan, yanyan, 朴智妍
<==        Row: 3, 16020520009, 123456789, 钟荣杰
<==      Total: 2
User{id=1, loginName='xxpiaozhiyan', loginPwd='yanyan', realName='朴智妍'}
User{id=3, loginName='16020520009', loginPwd='123456789', realName='钟荣杰'}
```

从日志可以看到，条件where是个大写的生成的，还把or去掉了，两个if语句不成立的话就是一个无条件的查询语句。

4、标签

当我们要执行的是带有in条件的sql语句时，in中的值是多个的，

--当接口方法中传的参数是一个Map集合，我们可以用平常使用的方法把需要传参用#{}来代替，其中括号中的属性名是集合的key值，创建map集合，给集合添加数据，然后调用接口方法把map集合传进去。

--当接口中的参数是List集合时，我们创建List集合，给集合添加数据，然后调用接口方法把List集合传进去，由于List集合不是键值对存储的，传进去后没法通过特定属性名用#{}取值。

上面的第二种方式在用<foreach>标签后可以将List集合中的值存放到in条件中，原理如下面代码所示：

```
 //下面代码演示<foreach>对传入的List集合处理的内部机理

        //这里新建List集合
        List<Integer> list = new ArrayList<>();
        //添加数据
        list.add(1);
        list.add(4);
        list.add(7);

        //假设映射文件中sql主语句，需要添加in后面的条件
        String sql = "select id,loginName,loginPwd,realName from t_user where id
in";

        //需要的条件是这样的：(1,7,4)
        //创建一个字符串拼接类StringBuffer
        StringBuffer buffer = new StringBuffer();
        //添加字符(
        buffer.append("(");
        //遍历list集合，并且追加到buffer中，中间记住加，
        for (Integer i:
             list) {
            buffer.append(i).append(",");
        }
        //除掉倒数第二个逗号
        buffer.deleteCharAt(buffer.length()-1);
        //追加最后一个)
        buffer.append(")");
        //与主语句拼接
        sql = sql + buffer;
        System.out.println(sql);//select id,loginName,loginPwd,realName from
t_user where id in(1,4,7)
```

```
<foreach>就是用来循环java中数组和list集合的。主要用在in语句中，使用规范如下
 <foreach ollection="list" item="id" open="(" close=")" separator=",">
        #{id}
</foreach>
 collection:表示接口中方法的参数类型，数组参数填array，集合参数填list
 item：这个是自定义的变量，表示数组或集合成员的元素
 open：循环开始时的字符，如上面代码中的"("
 close:循环结束时的字符，如上面代码中的")"
 separator:集合或数组中元素拼接时之间的分隔符。如逗号

 <!--有时候我们不一定要把所有的参数都写，如下-->
    <!--当我们把foreach标签放到()中，open和close就不用写了-->
     <select id="selectByForeach" resultType="User">
        select id,loginName,loginPwd,realName from t_user where id in(
        <foreach ollection="list" item="id" separator=",">
             #{id}
```

```
        </foreach>
         )

    </select>

    <!--有时候我们不一定要把所有的参数都写，如下-->
    <!--当我们把foreach标签放到()中，逗号放到#{}后面，open和close以及separator就不用写了
        但是需要在foreach标签后添加一个不影响查询条件的值,因为拼接后sql语句的in条件多出一个,
    结果： select id,loginName,loginPwd,realName from t_user where id in(?,?,?,0)
        -->
     <select id="selectByForeach" resultType="User">
        select id,loginName,loginPwd,realName from t_user where id in(
        <foreach ollection="list" item="id">
             #{id},
        </foreach>
         0
         )

    </select>

  <!--有时候list集合中的参数不一定是简单类型，当是java对象时，如下操作拿到对象中字段的值进行拼接-->
    <select id="selectByForeach" resultType="User">
        select id,loginName,loginPwd,realName from t_user where id in
       <foreach ollection="list" item="user" open="(" close=")" separator=",">
        #{user.id}
       </foreach>
    </select>
```

使用foreach具体方法如下

dao接口方法

```
package com.studymyself.dao;

import com.studymyself.entity.User;

import java.util.List;

//接口操作t_user表
public interface UserDao {

    //通过if动态sql语句查询数据,参数必须是Java对象
    public List<User> selectByIf(User user);

    //通过if和where动态sql语句查询数据,参数必须是Java对象
    public List<User> selectByWhere(User user);

    //执行查询有in条件的sql语句
    public List<User> selectByForeach(List<Integer> integers);
}
```

mapper文件如下，是在in后面添加foreach标签

```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="com.studymyself.dao.UserDao">


    <select id="selectByIf" resultType="User">
        select id,loginName,loginPwd,realName from t_user
        where
        <!--上面表示主语句-->
        <!--if表示这个标签中传递过来的Java对象的id字段值大于0
            就将if标签中的sql语句块拼接到主语句末端中去-->
        <if test="id > 0">
             id=#{id}
        </if>
        <!--传过来的java对象中的realName值不为空或不是空字符串
            就将if标签中的sql语句块拼接到主语句末端中去-->
            <if test="realName!= null or realName !='' ">
            or realName = #{realName}
        </if>
    </select>

    <select id="selectByWhere" resultType="User">
        select id,loginName,loginPwd,realName from t_user
        <where>
            <if test="id > 0">
                or id=#{id}
            </if>
            <if test="realName!= null and realName !='' ">
                or realName = #{realName}
            </if>
        </where>

    </select>

    <select id="selectByForeach" resultType="User">
        select id,loginName,loginPwd,realName from t_user where id in
        <foreach collection="list" item="id" open="(" close=")" separator=",">
            #{id}
        </foreach>
    </select>

</mapper>
```

测试程序

```java
    @Test
     public void testSelectByForeach(){

        //获取SqlSession对象
        SqlSession sqlSession = MyBatisUtil.getSqlSession();
        //通过SqlSession对象中的方法获取实现类对象
        UserDao userDao = sqlSession.getMapper(UserDao.class);
        List<Integer> list = new ArrayList<>();
```

```
        //添加数据
        list.add(1);
        list.add(4);
        list.add(7);
    List<User> users = userDao.selectByForeach(list);
        for (User user:
             users) {
            System.out.println(user);
        }

    }
```

日志

```
Opening JDBC Connection
Mon Feb 01 02:22:32 CST 2021 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+ requirements SSL connection must be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verifyServerCertificate property is set to 'false'. You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.
Created connection 762476028.
Setting autocommit to false on JDBC Connection
[com.mysql.jdbc.JDBC4Connection@2d7275fc]
==>  Preparing: select id,loginName,loginPwd,realName from t_user where id in ( ? , ? , ? )
==> Parameters: 1(Integer), 4(Integer), 7(Integer)
<==    Columns: id, loginName, loginPwd, realName
<==        Row: 1, xxpiaozhiyan, yanyan, 朴智妍
<==        Row: 4, lulujintaiyan, zxcvbnm, 金泰妍
<==        Row: 7, 6666666@mail.com, 8888888, 中国大使馆
<==      Total: 3
User{id=1, loginName='xxpiaozhiyan', loginPwd='yanyan', realName='朴智妍'}
User{id=4, loginName='lulujintaiyan', loginPwd='zxcvbnm', realName='金泰妍'}
User{id=7, loginName='6666666@mail.com', loginPwd='8888888', realName='中国大使馆'}
```

5、代码片段

```
--sql代码片段就是复用映射文件中一些sql语句块，映射文件中的一些sql语句块需要重复使用时，可以使用该方法
使用步骤：
    1、先使用<sql>标签定义复用的sql语句
    <sql id="自定义的唯一名称">
    sql语句
    </sql>
    2、在需要该sql语句的位置使用<include>标签表示该位置就是定义的sql语句
    <include refid="前面定义的代码片段的id" />
    --在mapper文件中具体使用如下：
```

```
<!--定义sql片段-->
<sql id="selectUserSql">
select id,loginName,loginPwd,realName from t_user
</sql>

<select id="getAll" resultType="User">
<!--使用sql片段-->
    <include refid="selectUserSql"/>
</select>
```