2019

Java EE框架 ---MyBatis

Java EE framework --MyBatis

王磊

计算机工程学院

CONTENTS

一、parameterType传递pojo包装类对象

需求：根据用户名模糊查询用户信息，查询条件放到QueryVo的user属性中

1、编写QueryVo

```
package com.xk.javabean;
import java.io.Serializable;
public class QueryVo implements Serializable{
  private static final long serialVersionUID = 1L;
  private User user;
  public User getUser() {
    return user;
  }
  public void setUser(User user) {
    this.user = user;
  }
}
```

2、相应的Sql语句

```
SELECT * FROM tuserlogin WHERE username LIKE '%张%'
```

一、parameterType传递pojo包装类对象

需求：根据用户名模糊查询用户信息，查询条件放到QueryVo的user属性中

3、编辑user.xml文件

```
    <!-- 使用包装类类型模糊查询用户 -->
    <select id="queryUserByQueryVo" parameterType="com.xk.javabean.QueryVo"
    resultType="com.xk.javabean.User">
        select * from tuserlogin where username like '%${user.username}%'
    </select>
</mapper>
```

一、parameterType传递pojo包装类对象

需求：根据用户名模糊查询用户信息，查询条件放到QueryVo的user属性中

4、编辑UserMapper接口

```
package com.xk.mapper;
import java.util.List;
import com.xk.javabean.QueryVo;
import com.xk.javabean.User;
public interface UserMapper {
        public User getUser(Integer id);
        public List<User> getAllUser();
        public void deleteUser(Integer id);
        public void addUser(User user);
       public List<User> queryUserByQueryVo(QueryVo queryVo);
}
```

一、parameterType传递pojo包装类对象

需求：根据用户名模糊查询用户信息，查询条件放到QueryVo的user属性中

5、在MybatisMapperTest类中添加测试方法

```
@Test
public void testQueryUserByQueryVo() throws IOException {
    SqlSession sqlSession = MybatisUtil.getSqlSession(true);
    UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
    QueryVo queryVo = new QueryVo();
    User user = new User();
    user.setUsername("张");
    queryVo.setUser(user);
    List<User> users = userMapper.queryUserByQueryVo(queryVo);
    for (User u : users) {
        System.out.println(u);
    }
}
```

一、parameterType传递pojo包装类对象

需求：根据用户名模糊查询用户信息，查询条件放到QueryVo的user属性中

6、观察结果

```
DEBUG [main] - Logging initialized using 'class org.apache.ibatis.logging.slf4j.Sl
DEBUG [main] - PooledDataSource forcefully closed/removed all connections.
DEBUG [main] - PooledDataSource forcefully closed/removed all connections.
DEBUG [main] - PooledDataSource forcefully closed/removed all connections.
DEBUG [main] - PooledDataSource forcefully closed/removed all connections.
DEBUG [main] - Opening JDBC Connection
DEBUG [main] - Created connection 1208736537.
DEBUG [main] - ==>  Preparing: select * from tuserlogin where username like '%张%'
DEBUG [main] - ==> Parameters:
DEBUG [main] - <==      Total: 4
User [id=11, username=张恒, password=123456, grade=2, email=zhang@qq.com]
User [id=13, username=张扬菲, password=123456, grade=1, email=zhangyf@126.com]
User [id=21, username=张恒1, password=123456, grade=2, email=zhang@qq.com]
User [id=23, username=张扬菲3, password=123456, grade=1, email=zhangyf@126.com]
```

二、通过标签方法实现动态拼接sql

需求：根据级别和姓名查询用户

查询sql：

SELECT * FROM tuserlogin WHERE grade = 1 AND username LIKE '%张%‘

1、编辑user.xml

```
<!-- 根据多个条件查询用户 -->
<select id="queryByGradeAndName" parameterType="com.xk.javabean.User"
    resultType="com.xk.javabean.User">
    select * from tuserlogin where grade = #{grade} and username like
    '%${username}%'
</select>
```

二、通过标签方法实现动态拼接sql

需求：根据级别和姓名查询用户

查询sql：

SELECT * FROM tuserlogin WHERE grade = 1 AND username LIKE '%张%‘

2、编写Mapper接口

```
public interface UserMapper {
    public User getUser(Integer id);
    public List<User> getAllUser();
    public void deleteUser(Integer id);
    public void addUser(User user);
    public List<User> queryUserByQueryVo(QueryVo queryVo);
    public List<User> queryByGradeAndName(User user);
}
```

二、通过标签方法实现动态拼接sql

需求：根据级别和姓名查询用户

查询sql：

SELECT * FROM tuserlogin WHERE grade = 1 AND username LIKE '%张%‘

3、 MybatisMapperTest类中添加测试方法

```
@Test
public void testqueryByGradeAndName() throws IOException {
    SqlSession sqlSession = MybatisUtil.getSqlSession(true);
    UserMapper userMapper =
    sqlSession.getMapper(UserMapper.class);
    User user = new User();
    user.setGrade("1");
    user.setUsername("张");
    List<User> users = userMapper.queryByGradeAndName(user);
    for (User u : users) {
    System.out.println(u);
    }
}
```

二、通过标签方法实现动态拼接sql

需求：根据级别和姓名查询用户

查询sql：

SELECT * FROM tuserlogin WHERE grade = 1 AND username LIKE '%张%‘

4、观察测试效果

```
DEBUG [main] - ==>  Preparing: select * from tuserlogin where grade = ? and username like '%张%'
DEBUG [main] - ==> Parameters: 1(String)
DEBUG [main] - <==      Total: 2
User [id=13, username=张扬菲, password=123456, grade=1, email=zhangyf@126.com]
User [id=23, username=张扬菲3, password=123456, grade=1, email=zhangyf@126.com]
DEBUG [Finalizer] - PooledDataSource forcefully closed/removed all connections.
```

如果注释掉 user.setGrade("1"); 测试结果如下图

```
DEBUG [main] - ==>  Preparing: select * from tuserlogin where grade = ? and username like '%张%'
DEBUG [main] - ==> Parameters: null
DEBUG [main] - <==      Total: 0
```

很显然测试结果不合理，解决方案，使用动态sql的if标签

二、通过标签方法实现动态拼接sql

需求：根据级别和姓名查询用户

查询sql：

SELECT * FROM tuserlogin WHERE grade = 1 AND username LIKE '%张%‘

改造user.xml

```
<!-- 根据多个条件查询用户 -->
<select id="queryByGradeAndName" parameterType="com.xk.javabean.User"
resultType="com.xk.javabean.User">
        select * from tuserlogin
        <where>
                <if test="grade!=null">
                        and grade = #{grade}
                </if>
                <if test="username!=null and username != ''">
                        and username like '%${username}%'
                </if>
        </where>
</select>
```

二、通过标签方法实现动态拼接sql

需求：根据级别和姓名查询用户
查询sql：
SELECT * FROM tuserlogin WHERE grade = 1 AND username LIKE '%张%‘
改造后，执行效果：

```
DEBUG [main] - ==>  Preparing: select * from tuserlogin WHERE username like '%张%'
DEBUG [main] - ==> Parameters:
DEBUG [main] - <==      Total: 4
User [id=11, username=张恒, password=123456, grade=2, email=zhang@qq.com]
User [id=13, username=张扬菲, password=123456, grade=1, email=zhangyf@126.com]
User [id=21, username=张恒1, password=123456, grade=2, email=zhang@qq.com]
User [id=23, username=张扬菲3, password=123456, grade=1, email=zhangyf@126.com]
```

三、关联查询

需求：创建部门和部门经理表，属于一对一关联。使用Mybati进行关联查询。

表结构如下：

```
-- ----------------------------
-- Table structure for department
-- ----------------------------
DROP TABLE IF EXISTS `department`;
CREATE TABLE `department` (
  `depid` int(11) NOT NULL auto_increment,
  `depname` varchar(20) NOT NULL,
  `depaddress` varchar(20) default NULL,
  `mana_id` int(11) default NULL,
  PRIMARY KEY (`depid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
INSERT INTO `department` VALUES ('1', '人力资源部', '5#13号', '101');
INSERT INTO `department` VALUES ('2', '市场部', '5#14号', '102');
INSERT INTO `department` VALUES ('3', '后勤部', '5#15号', '103');
INSERT INTO `department` VALUES ('4', '技术部', '5#16号', '104');
INSERT INTO `department` VALUES ('5', '外联部', '5#17号', '105');
```

三、关联查询

需求：创建部门和部门经理表，属于一对一关联。使用Mybati进行关联查询。

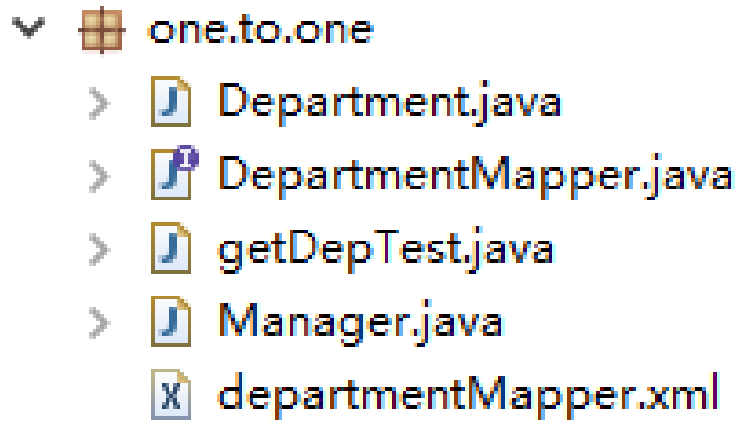
表结构如下：

```
-- ----------------------------
-- Table structure for manager
-- ----------------------------
DROP TABLE IF EXISTS `manager`;
CREATE TABLE `manager` (
  `mana_id` int(11) NOT NULL default '0',
  `mana_name` varchar(20) NOT NULL,
  `mana_sex` varchar(20) default NULL,
  `mana_age` int(11) default NULL,
  PRIMARY KEY  (`mana_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO `manager` VALUES ('101', '张云云', '男', '32');
INSERT INTO `manager` VALUES ('102', '刘晓峰', '男', '44');
INSERT INTO `manager` VALUES ('103', '吴一铭', '男', '28');
INSERT INTO `manager` VALUES ('104', '李建国', '男', '33');
INSERT INTO `manager` VALUES ('105', '王鹏', '男', '38');
```

三、关联查询

创建one.to.one包，结构如下



创建DepartmentMapper接口

```
package one.to.one;

import java.util.List;

public interface DepartmentMapper {

    public List<Department> getAllDepartment();
}
```

三、关联查询

创建Department和Manager类

```java
package one.to.one;
public class Department {
    private Integer depid;
    private String depname;
    private String depaddress;
    private Integer mana_id;
    private Manager manager;
    public Integer getDepid() {
        return depid;
    }
    public void setDepid(Integer depid) {
        this.depid = depid;
    }
    public String getDepname() {
        return depname;
    }
```

```java
package one.to.one;
public class Manager {
    private Integer mana_id;
    private String mana_name;
    private String mana_sex;
    private Integer mana_age;
    private Department department;

    public Department getDepartment() {
        return department;
    }
    public void setDepartment(Department department) {
        this.department = department;
    }
```

三、关联查询

创建departmentMapper.xml

```
<mapper namespace="one.to.one.DepartmentMapper">
    <select id="getAllDepartment" parameterType="Integer" resultMap="departmentMap">
        select * from department d,manager m where d.mana_id=m.mana_id
    </select>
    <resultMap type="one.to.one.Department" id="departmentMap">
        <id column="depid" property="depid"/>
        <result column="depname" property="depname"/>
        <result column="depaddress" property="depaddress"/>
        <association property="manager" javaType="one.to.one.Manager">
            <id column="mana_id" property="mana_id"/>
            <result column="mana_name" property="mana_name"/>
            <result column="mana_sex" property="mana_sex"/>
            <result column="mana_age" property="mana_age"/>
        </association>
    </resultMap>
</mapper>
```

三、关联查询

创建getDepTest测试类

```
package one.to.one;
import java.io.IOException;
import java.util.List;
import org.apache.ibatis.session.SqlSession;
import org.junit.Test;
import com.xk.util.MybatisUtil;
public class getDepTest {
    @Test
    public void testGetDep() throws IOException {

        SqlSession sqlSession = MybatisUtil.getSqlSession(true);

        DepartmentMapper departmentMapper = sqlSession.getMapper(DepartmentMapper.class);
        List<Department> li = departmentMapper.getAllDepartment();
        for (Department d : li) {
            System.out.println(d.getDepid()+" "+d.getDepname()
            +" "+d.getDepaddress()+" "+d.getManager().getMana_name());
        }
        sqlSession.close();
    }
}
```

三、关联查询

观察运行结果

```
DEBUG [main] - Opening JDBC Connection
DEBUG [main] - Created connection 57748372.
DEBUG [main] - ==>  Preparing: select * from department d,manager m where d.mana_id=m.mana_id
DEBUG [main] - ==> Parameters:
DEBUG [main] - <==      Total: 5
1 人力资源部 5#13号 张云云
2 市场部 5#14号 刘晓峰
3 后勤部 5#15号 吴一铭
4 技术部 5#16号 李建国
5 外联部 5#17号 王鹏
```

总结

1、Mybatis jar包

2、Mybatis核心配置文件sqlMapConfig.xml

3、Mybatis Util工具类

4、Mapper动态代理方式

5、标签方法实现动态拼接sql

6、一对一、一对多关联查询