基于简单mybatis的CRUD操作

一、创建UserDao接口

```
import com.itheima.domain.QueryVo;
import com.itheima.domain.User;

import java.util.List;

/**
 * 用户的持久层接口
 */
public interface IUserDao {

    /**
     * 查询所有用户
     * @return
     */
    List<User> findAll();

    /**
     * 保存用户
     * @param user
     */
    void saveUser(User user);

    /**
     * 更新用户
     * @param user
     */
    void updateUser(User user);

    /**
     * 根据Id删除用户
     * @param userId
     */
    void deleteUser(Integer userId);

    /**
     * 根据id查询用户信息
     * @param userId
     * @return
     */
    User findById(Integer userId);

    /**
     * 根据名称模糊查询用户信息
     * @param username
     * @return
     */
    List<User> findByName(String username);
```

```
    /**
     * 查询总用户数
     * @return
     */
    int findTotal();

    /**
     * 根据queryVo中的条件查询用户
     * @param vo
     * @return
     */
    List<User> findUserByVo(QueryVo vo);
}
```

二、创建User类和QueryVo 类

1、注意 User implements Serializable

2、没配置情况下：user类的属性名必须和数据库列名一样

```
import java.io.Serializable;
import java.util.Date;

public class User implements Serializable {

    private Integer userId;
    private String userName;
    private String userAddress;
    private String userSex;
    private Date userBirthday;

    public Integer getUserId() {
        return userId;
    }

    public void setUserId(Integer userId) {
        this.userId = userId;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getUserAddress() {
        return userAddress;
    }

    public void setUserAddress(String userAddress) {
        this.userAddress = userAddress;
    }
```

```
    public String getUserSex() {
        return userSex;
    }

    public void setUserSex(String userSex) {
        this.userSex = userSex;
    }

    public Date getUserBirthday() {
        return userBirthday;
    }

    public void setUserBirthday(Date userBirthday) {
        this.userBirthday = userBirthday;
    }

    @Override
    public String toString() {
        return "User{" +
                "userId=" + userId +
                ", userName='" + userName + '\'' +
                ", userAddress='" + userAddress + '\'' +
                ", userSex='" + userSex + '\'' +
                ", userBirthday=" + userBirthday +
                '}';
    }
}
```

QueryVo 类：

```
public class QueryVo {

    private User user;

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }
}
```

三、配置UserDao.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.itheima.dao.IUserDao">

    <!-- 配置 查询结果的列名和实体类的属性名的对应关系 -->
    <resultMap id="userMap" type="uSeR">
        <!-- 主键字段的对应 -->
        <id property="userId" column="id"></id>
```

```
        <!--非主键字段的对应-->
        <result property="userName" column="username"></result>
        <result property="userAddress" column="address"></result>
        <result property="userSex" column="sex"></result>
        <result property="userBirthday" column="birthday"></result>
    </resultMap>


    <!-- 查询所有 -->
    <select id="findAll" resultMap="userMap">
        <!--select id as userId,username as userName,address as userAddress,sex as userSex,birthday
as userBirthday from user;-->
        select * from user;
    </select>

    <!-- 保存用户 -->
    <insert id="saveUser" parameterType="user">
        <!-- 配置插入操作后，获取插入数据的id -->
        <selectKey keyProperty="userId" keyColumn="id" resultType="int" order="AFTER">
            select last_insert_id();
        </selectKey>
        insert into user(username,address,sex,birthday)values(#{userName},#{userAddress},#
{userSex},#{userBirthday});
    </insert>

    <!-- 更新用户 -->
    <update id="updateUser" parameterType="USER">
        update user set username=#{userName},address=#{userAddress},sex=#{userAex},birthday=#
{userBirthday} where id=#{userId}
    </update>

    <!-- 删除用户-->
    <delete id="deleteUser" parameterType="java.lang.Integer">
        delete from user where id = #{uid}
    </delete>

    <!-- 根据id查询用户 -->
    <select id="findById" parameterType="INT" resultMap="userMap">
        select * from user where id = #{uid}
    </select>

    <!-- 根据名称模糊查询 -->
    <select id="findByName" parameterType="string" resultMap="userMap">
          select * from user where username like #{name}
        <!-- select * from user where username like '%${value}%'-->
    </select>

    <!-- 获取用户的总记录条数 -->
    <select id="findTotal" resultType="int">
        select count(id) from user;
    </select>

    <!-- 根据queryVo的条件查询用户 -->
    <select id="findUserByVo" parameterType="com.itheima.domain.QueryVo"
resultMap="userMap">
```

```
        select * from user where username like #{user.username}
    </select>
</mapper>
```

四、配置SqlMapconfig.xml

```
<?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>
    <!-- 配置properties
        可以在标签内部配置连接数据库的信息。也可以通过属性引用外部配置文件信息
        resource属性： 常用的
            用于指定配置文件的位置，是按照类路径的写法来写，并且必须存在于类路径下。
        url属性：
            是要求按照Url的写法来写地址
            URL：Uniform Resource Locator 统一资源定位符。它是可以唯一标识一个资源的位置。
            它的写法：
                http://localhost:8080/mybatisserver/demo1Servlet
                协议      主机     端口       URI

            URI:Uniform Resource Identifier 统一资源标识符。它是在应用中可以唯一定位一个资源的。
    -->
    <properties
url="file:///D:/IdeaProjects/day02_eesy_01mybatisCRUD/src/main/resources/jdbcConfig.properties">

        <!-- <property name="driver" value="com.mysql.jdbc.Driver"></property>
        <property name="url" value="jdbc:mysql://localhost:3306/eesy_mybatis"></property>
        <property name="username" value="root"></property>
        <property name="password" value="1234"></property>-->
    </properties>

    <!--使用typeAliases配置别名，它只能配置domain中类的别名 -->
    <typeAliases>
        <!--typeAlias用于配置别名。type属性指定的是实体类全限定类名。alias属性指定别名，当指定了别名
就再区分大小写
        <typeAlias type="com.itheima.domain.User" alias="user"></typeAlias>-->

        <!-- 用于指定要配置别名的包，当指定之后，该包下的实体类都会注册别名，并且类名就是别名，不再
区分大小写-->
        <package name="com.itheima.domain"></package>
    </typeAliases>

    <!--配置环境-->
    <environments default="mysql">
        <!-- 配置mysql的环境-->
        <environment id="mysql">
            <!-- 配置事务 -->
            <transactionManager type="JDBC"></transactionManager>

            <!--配置连接池-->
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}"></property>
```

```
            <property name="url" value="${jdbc.url}"></property>
            <property name="username" value="${jdbc.username}"></property>
            <property name="password" value="${jdbc.password}"></property>
        </dataSource>
    </environment>
</environments>
<!-- 配置映射文件的位置 -->
<mappers>
    <!--<mapper resource="com/itheima/dao/IUserDao.xml"></mapper>-->
    <!-- package标签是用于指定dao接口所在的包,当指定了之后就不需要在写mapper以及resource或者
class了 -->
    <package name="com.itheima.dao"></package>
</mappers>
</configuration>
```

五、测试

1、注意inti()和destory()的使用，简化代码

```
/**
 * 测试mybatis的crud操作
 */
public class MybatisTest {

    private InputStream in;
    private SqlSession sqlSession;
    private IUserDao userDao;

    @Before//用于在测试方法执行之前执行
    public void init()throws Exception{
        //1.读取配置文件，生成字节输入流
        in = Resources.getResourceAsStream("SqlMapConfig.xml");
        //2.获取SqlSessionFactory
        SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(in);
        //3.获取SqlSession对象
        sqlSession = factory.openSession();
        //4.获取dao的代理对象
        userDao = sqlSession.getMapper(IUserDao.class);
    }

    @After//用于在测试方法执行之后执行
    public void destroy()throws Exception{
        //提交事务
        sqlSession.commit();
        //6.释放资源
        sqlSession.close();
        in.close();
    }


    /**
     * 测试查询所有
```

```
     */
    @Test
    public void testFindAll(){
        //5.执行查询所有方法
        List<User> users = userDao.findAll();
        for(User user : users){
            System.out.println(user);
        }

    }

    /**
     * 测试保存操作
     */
    @Test
    public void testSave(){
        User user = new User();
        user.setUserName("modify User property");
        user.setUserAddress("北京市顺义区");
        user.setUserSex("男");
        user.setUserBirthday(new Date());
        System.out.println("保存操作之前："+user);
        //5.执行保存方法
        userDao.saveUser(user);

        System.out.println("保存操作之后："+user);
    }


/**
     * 测试更新操作
     */
    @Test
    public void testUpdate(){
        User user = new User();
        user.setUserId(50);
        user.setUserName("mybastis update user");
        user.setUserAddress("北京市顺义区");
        user.setUserSex("女");
        user.setUserBirthday(new Date());

        //5.执行保存方法
        userDao.updateUser(user);
    }

    /**
     * 测试删除操作
     */
    @Test
    public void testDelete(){
        //5.执行删除方法
        userDao.deleteUser(48);
    }
```

```
    /**
     * 测试删除操作
     */
    @Test
    public void testFindOne(){
        //5.执行查询一个方法
        User  user = userDao.findById(50);
        System.out.println(user);
    }

        */
    /**
     * 测试模糊查询操作
     */
    @Test
    public void testFindByName(){
        //5.执行查询一个方法
        List<User> users = userDao.findByName("%王%");
//        List<User> users = userDao.findByName("王");
        for(User user : users){
            System.out.println(user);
        }
    }

    /**
     * 测试查询总记录条数
     */
    @Test
    public void testFindTotal(){
        //5.执行查询一个方法
        int count = userDao.findTotal();
        System.out.println(count);
    }


    /**
     * 测试使用QueryVo作为查询条件
     */
    @Test
    public void testFindByVo(){
        QueryVo vo = new QueryVo();
        User user = new User();
        user.setUserName("%王%");
        vo.setUser(user);
        //5.执行查询一个方法
        List<User> users = userDao.findUserByVo(vo);
        for(User u : users){
            System.out.println(u);
        }
    }
}
```