入门程序

1. 需求

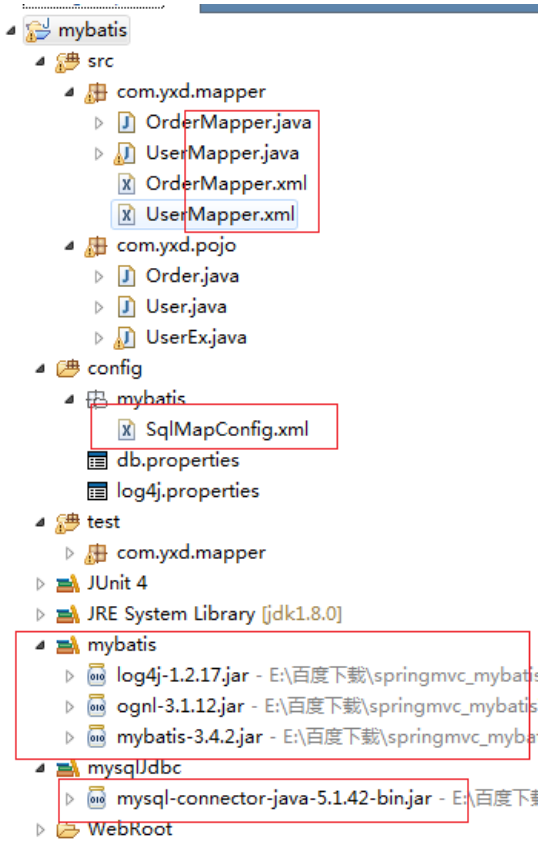
- 根据用户id（主键）查询用户信息。
- 根据用户名称模糊查询用户信息。
- 添加用户。
- 删除用户。
- 更新用户。

2. 环境

- java环境：jdk1.8
- myEclipse
- mysql：5.7
- mybatis运行环境（jar包）：3.4.2
  - lib中的依赖包
  - mybatis-3.4.2.jar
- mysql驱动包

3. 工程结构

1. config中添加log4j.properties配置文件
2. config中创建mybatis文件夹，放置SqlMapConfig.xml配置文件
3. src中创建com.yxd.mapper放置mapper接口和mapper的xml配置文件。
4. test用来做测试



3. log4j.properties

- 内容

```
# 注意：设置成debug级别的，发布后设置成error,stdout为未来日志的目的地名称
log4j.rootLogger=DEBUG, stdout
lo
# Console控制台设置
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
# %l:添加该属性，可以在日志中看出该条日志在哪里发生的 （一般不用）
log4j.appender.stdout.layout.ConversionPattern= %d{yyyy MMM dd HH:mm:ss}  %p [%t] - %m%n
```

5. SqlMapConfig.xml

- 配置mybatis的运行环境，数据源、事务等。

```
<?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>
    <!-- 和spring整合后environments配置将被废除 -->
    <environments default="development">
        <environment id="development">
            <!-- 使用jdbc事务管理器，事务控制由mybatis -->
            <transactionManager type="JDBC" />
            <!-- 数据库连接池，由mybatis管理 -->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver" />
                <property name="url" value="jdbc:mysql://127.0.0.1:3306/mybatis?useUnicode=true&characterEncoding=utf-8" />
                <property name="username" value="root" />
                <property name="password" value="shadowolf1995" />
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <!-- 映射文件的位置 -->
        <mapper resource="sqlmap/User.xml" />
    </mappers>
</configuration>
```

6. 根据用户id（主键）查询用户信息

1. 创建po类

```
import java.util.Date;

public class User {

    // 属性名和数据库表字段对应
    private int id;
    private  String username;
    private String sex;
    private Date birthday;
    private String address;

    public int getId() {
        return id;
    }
```

```
    public void setId(int id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    public Date getBirthday() {
        return birthday;
    }

    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

 @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", sex='" + sex + '\'' +
                ", birthday=" + birthday +
                ", address='" + address + '\'' +
                '}';
    }
}
```

2. 映射文件（User.xml）

- 映射文件命名
  - User.xml（原始ibatis命名方式），mapper代理开发映射文件名称叫XXXMapper.xml，比如：UserMapper.xml、ItemsMapper.xml
- 在映射文件中配置sql语句

```
<?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">
<!-- namespace命名空间，作用就是对sql进行分类化管理，理解sql隔离
 注意：使用mapper代理方法开发，namespace有特殊重要作用
 -->
<mapper namespace="test">
    <!-- 在配置文件中配置很多sql语句 -->
    <!-- 通过select执行数据库的查询
     id：标识映射文件中的sql，称为statement的id
```

```
     将sql语句封装到mappedStatement对象中，所以id称为statement的id
     parameterType：指定输入参数的类型，这里指定int型
     #{}表示一个占位符
     #{id}：其中的id表示接收输入的参数，参数名称就是id，如果输入参数是简单类型，
            #{}中的参数名可以任意，可以是value或其他名称
     resultType：指定sql输出结果所映射成的java对象类型，select指定resultType表示将单条记录映射成的java对象。
     -->
    <select id="findUserById" parameterType="int" resultType="com.swxc.mybatis.po.User">
        SELECT * FROM USER WHERE id = #{id}
    </select>
</mapper>
```

- 在SqlMapConfig.xml中加载映射文件
  - 在SqlMapConfig.xml中加载User.xml

```
<mappers>
    <!-- 加载映射文件 -->
    <mapper resource="sqlmap/User.xml" />
</mappers>
```

- 程序编写

```
import com.swxc.mybatis.po.User;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
import org.junit.Test;

import java.io.IOException;
import java.io.InputStream;


public class MybatisFirst {

    // 根据id查询用户信息，得到一条记录的结果
    @Test
    public void findUserByIdTest() throws IOException {

        // mybatis配置文件
        String resource = "SqlMapConfig.xml";
        // 得到配置文件流
        InputStream inputStream = Resources.getResourceAsStream(resource);

        // 创建会话工厂，传入mybatis配置文件信息
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

        // 通过工厂得到SqlSession
        SqlSession sqlSession = sqlSessionFactory.openSession();

        // 通过SqlSession操作数据库
        // 第一个参数：映射文件中的statement的id，等于=namespace+statement的id
        // 第二个参数：指定和映射文件中所匹配的paramenterType类型的参数
        // sqlSession.selectOne结果是与映射文件中所匹配的resultType类型的对象
        User user = sqlSession.selectOne("test.findUserById", 1);

        System.out.println(user);

        // 释放资源
        sqlSession.close();

    }

}
```

7.根据用户名称模糊查询用户信息

1. 映射文件

- 使用User.xml，添加根据用户名称模糊查询用户信息的sql语句。

```
<!-- 根据用户名称模糊查询用户信息，可能返回多条
 resultType：指定单条记录所映射的java对象类型
 ${}：表示拼接sql串，将接收到的参数内容不加任何修饰拼接到sql中
 使用${}拼接sql，引起sql注入
 ${username}：接收输入参数的内容，如果传入类型是简单类型，${}中只能使用value
 -->
<select id="findUserByName" parameterType="java.lang.String" resultType="com.swxc.mybatis.po.User">
    SELECT * FROM USER WHERE username LIKE '%${value}%'
</select>
```

2. 程序代码

```
// 根据用户名称模糊查询用户列表
@Test
public void findUserByNameTest() throws IOException {
    // mybatis配置文件
    String resource = "SqlMapConfig.xml";
    // 得到配置文件流
    InputStream inputStream = Resources.getResourceAsStream(resource);

    // 创建会话工厂，传入mybatis配置文件信息
    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

    // 通过工厂得到SqlSession
    SqlSession sqlSession = sqlSessionFactory.openSession();

    List list = sqlSession.selectList("test.findUserByName", "五");
    System.out.println(list);
    sqlSession.close();
}
```

8. 添加用户

1. 映射文件

- 在User.xml中配置添加用户的Statement

```
<!-- 添加用户
 parameterType：指定输入参数类型是pojo（包括用户信息）
 #{}中指定pojo的属性名，接收到pojo对象的属性值，mybatis通过OGNL获取对象的属性值
 -->
<insert id="insertUser" parameterType="com.swxc.mybatis.po.User">
    INSERT INTO user(id, username, birthday, sex, address) values(#{id},#{username},#{birthday},#{sex},#{address})
</insert>
```

2. 程序代码

```
// 插入用户
@Test
public void insertUser() throws IOException {
    String resource = "SqlMapConfig.xml";
    InputStream inputStream = Resources.getResourceAsStream(resource);
    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 插入用户对象
```

```
    User user = new User();
    user.setUsername("关小军");
    user.setBirthday(new Date());
    user.setSex("1");
    user.setAddress("河南郑州");
    sqlSession.insert("test.insertUser", user);
    // 提交事务
    sqlSession.commit();
    sqlSession.close();
}
```

3. 自增主键返回

- mysql自增主键，执行insert提交之前自动生成一个自增主键。
- 通过mysql函数获取到刚插入记录的自增主键。
  - LAST*INSERT*ID()
  - 是在insert之后调用次函数
- 修改User.xml中insert部分

```
<insert id="insertUser" parameterType="com.swxc.mybatis.po.User">
    <!--
    将插入数据的主键返回，返回到user对象中
    SELECT LAST_INSERT_ID()：得到刚insert进去记录的主键值，只适用于自增主键
    keyProperty：将查询到的主键值设置到parameterType指定的对象的哪个属性
    order：SELECT LAST_INSERT_ID()执行顺序，相对于insert语句来说它的执行顺序
    resultType：指定SELECT LAST_INSERT_ID()结果的类型
     -->
    <selectKey keyProperty="id" order="AFTER" resultType="java.lang.Integer">
        SELECT LAST_INSERT_ID()
    </selectKey>
    INSERT INTO user(username, birthday, sex, address) values(#{username},#{birthday},#{sex},#{address})
</insert>
```

4. 非自增主键的返回（使用uuid()）

- 使用mysql的uuid()函数生成主键，需要修改表中id字段的类型为string，长度设置成35位。
- 执行思路：先通过uuid()查询到主键，将主键输入到sql语句中。
- 执行uuid()语句顺序相对于insert语句之前。
- 修改User.xml中insert部分

```
<!-- 使用mysql的uuid()方法生成主键
 执行过程：
 首先通过uuid()得到主键，将主键设置到user对象的id属性中
 其次在insert执行时，从user对象中去除id属性值
 -->
<selectKey keyProperty="id" order="BEFORE" resultType="java.lang.String">
    SELECT uuid()
</selectKey>
INSERT INTO user(id, username, birthday, sex, address) values(#{id},#{username},#{birthday},#{sex},#{address})
```

9. 删除用户

1. 映射文件

```
<!-- 删除用户
 根据id删除用户，需要输入id值
 -->
<delete id="deleteUser" parameterType="java.lang.Integer">
    DELETE FROM user WHERE id = #{id}
</delete>
```

2. 程序代码

```
// 删除用户
@Test
public void deleteUserTest() throws IOException {
    String resource = "SqlMapConfig.xml";
    InputStream inputStream = Resources.getResourceAsStream(resource);
    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
    SqlSession sqlSession = sqlSessionFactory.openSession();
    sqlSession.delete("test.deleteUser", 7);
    sqlSession.commit();
    sqlSession.close();
}
```

10. 更新用户

1. 映射文件

```
<!-- 更新用户
 分析:
 需要传入用户的id
 需要传入用户的更新信息
 parameterType指定User对象，包括id和更新信息，注意：id必须存在
 #{id}：从输入User对象中获取id属性值
 -->
<update id="" parameterType="com.swxc.mybatis.po.User">
    UPDATE user SET username = #{username}, birthday = #{birthday}, sex = #{sex}, address = #{address} WHERE id = #{id}
</update>
```

2. 程序代码

```
// 更新用户
@Test
public void updateUserTest() throws IOException {
    String resouorce = "SqlMapConfig.xml";
    InputStream inputStream = Resources.getResourceAsStream(resouorce);
    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
    SqlSession sqlSession = sqlSessionFactory.openSession();
    User user = new User();
    user.setId(6);
    user.setUsername("小王");
    user.setBirthday(new Date());
    user.setSex("2");
    user.setAddress("北京");
    sqlSession.update("test.updateUser", user);
    sqlSession.commit();
    sqlSession.close();
    }
```

11. 总结

1. parameterType

- 在映射文件中通过parameterType指定输入参数的类型。

2. resultType

- 在映射文件中通过resultType指定输出结果的类型。

3. #{}和${}

- #{}表示一个占位符号，#{}接收输入参数，类型可以是简单类型、pojo、hashmap。

    - 如果接收的是简单类型，#{}中可以写成value或其他名称。
    - #{}接收pojo对象值，通过OGNL读取对象中的属性值，通过属性.属性.属性...获取属性值。
- ${}表示一个拼接符号，会引起sql注入，所以不建议使用${}。
    - ${}接收输入参数，类型可以是简单类型、pojo、hashmap。
    - 如果接收的是简单类型，#{}中只能写成value。
    - ${}接收pojo对象值，通过OGNL读取对象中的属性值，通过属性.属性.属性...获取属性值。

4. selectOne和selectList

- selectOne表示查询出一条记录进行映射。
- selectList表示查询出一个列表（多条记录）进行映射。
- 如果使用selectOne可以实现，那么使用selectList也可以实现（list中只有一条记录）。

12. mybatis和hibernate本质区别和应用场景

- hibernate
    - 特点：是一个标准的ORM框架（对象关系映射），入门门槛较高，不需要编写sql，sql语句自动生成。对sql语句进行优化修改比较困难。
    - 应用场景：适用于需求变化不多的中小型项目，比如：后台管理系统、erp、orm、oa。
- mybatis
    - 特点：专注sql本身，需要程序员自己编写sql语句，sql修改、优化比较方便。mybatis是一个不完全的ORM框架，虽然程序员自己写sql，mybatis也可以实现映射（输入映射、输出映射）。
    - 应用场景：适用于需求变化较多的项目，比如：互联网项目。