实验一：MyBatis 入门程序

一、 **实验目的**：MyBatis 是当前主流的 Java 持久层框架之一，它与 Hibernate 一样，也是一种 ORM 框架。因其性能优异，且具有高度的灵活性、可优化性和易于维护等特点，所以受到了广大互联网企业的青睐，是目前大型互联网项目的首选框架。

二、 **预习要求**：

1、了解 MyBatis 的基础知识

2、 熟悉 MyBatis 的工作原理

3、 掌握 MyBatis 入门程序的编写

三、 **实验内容**：

1、MyBatis 入门程序——查询用户

2、MyBatis 入门程序——添加用户

3、MyBatis 入门程序——更新用户

4、MyBatis 入门程序——删除用户

四、 **实验方法和步骤**：

创建 pojo

pojo 类作为 mybatis 进行 sql 映射使用，po 类通常与数据库表对应，

数据库 user 表如下图：

常规 DDL

```
CREATE TABLE `user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(32) NOT NULL COMMENT '用户名称',
  `birthday` date DEFAULT NULL COMMENT '生日',
  `sex` char(1) DEFAULT NULL COMMENT '性别',
  `address` varchar(256) DEFAULT NULL COMMENT '地址',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=27 DEFAULT CHARSET=utf8;
```

User.java 如下：

```
public class User {
    private int id;
    private String username;// 用户姓名
    private String sex;// 性别
    private Date birthday;// 生日
    private String address;// 地址

get/set……
toString 方法
```

sql 映射文件

创建 sql 映射文件 User.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">
<!-- namespace：命名空间，用于隔离sql -->
<mapper namespace="test">
</mapper>
```

创建配置文件

创建 MyBatis 核心配置文件 SqlMapConfig.xml，如下：

```
<?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配置将废除 -->
    <environments default="development">
        <environment id="development">
            <!-- 使用jdbc事务管理 -->
            <transactionManager type="JDBC" />
            <!-- 数据库连接池 -->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"
/>
                <property name="url"
```

```
                value="jdbc:mysql:///test?characterEncoding=utf-8"
/>
            <property name="username" value="root" />
            <property name="password" value="123" />
        </dataSource>
    </environment>
</environments>

<mappers>
    <mapper resource="User.xml" />
</mappers>
</configuration>
```

SqlMapConfig.xml 是 mybatis 核心配置文件，配置文件内容为数据源、事务管理。

创建 log4j.properties 如下：

```
# Global logging configuration
log4j.rootLogger=DEBUG, stdout
# Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

实现根据 id 查询用户

使用的 sql:
SELECT * FROM `user` WHERE id = 1

映射文件：

在 user.xml 中添加 select 标签，编写 sql：

```
<?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，用于隔离 sql-->
<mapper namespace="test">

    <!-- id:statement 的 id 或者叫做 sql 的 id-->
    <!-- parameterType:声明输入参数的类型 -->
    <!-- resultType:声明输出结果的类型，应该填写 pojo 的全路径 -->
    <!-- #{}：输入参数的占位符，相当于 jdbc 的？ -->
```

```
    <select id="queryUserById" parameterType="int"
        resultType="com.haust.mybatis.pojo.User">
        SELECT * FROM `user` WHERE id  = #{id}
    </select>

</mapper>
```

测试程序：

测试程序步骤：

1. 创建 SqlSessionFactoryBuilder 对象
2. 加载 SqlMapConfig.xml 配置文件
3. 创建 SqlSessionFactory 对象
4. 创建 SqlSession 对象
5. 执行 SqlSession 对象执行查询，获取结果 User
6. 打印结果
7. 释放资源

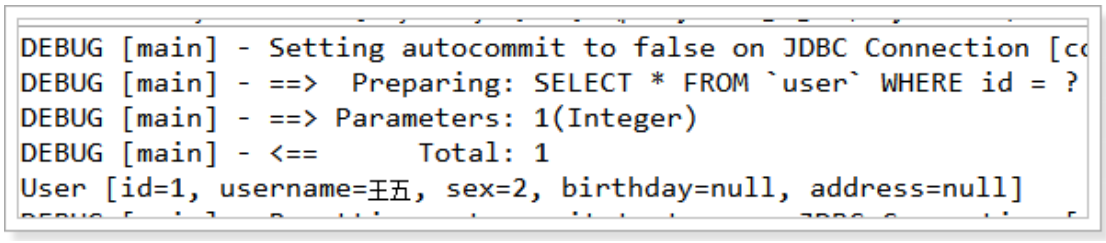
创建 MybatisTest，编写测试程序如下：

```
public class MybatisTest {
    private SqlSessionFactory sqlSessionFactory = null;

    @Before
    public void init() throws Exception {
        // 1. 创建 SqlSessionFactoryBuilder 对象
        SqlSessionFactoryBuilder sqlSessionFactoryBuilder = new
SqlSessionFactoryBuilder();

        // 2. 加载 SqlMapConfig.xml 配置文件
        InputStream inputStream =
Resources.getResourceAsStream("SqlMapConfig.xml");

        // 3. 创建 SqlSessionFactory 对象
        this.sqlSessionFactory =
sqlSessionFactoryBuilder.build(inputStream);
    }

    @Test
    public void testQueryUserById() throws Exception {
        // 4. 创建 SqlSession 对象
        SqlSession sqlSession = sqlSessionFactory.openSession();

        // 5. 执行 SqlSession 对象执行查询，获取结果 User
```

```
        // 第一个参数是 User.xml 的 statement 的 id，第二个参数是执行 sql 需要的参数：
        Object user = sqlSession.selectOne("test.queryUserById", 1);

        // 6. 打印结果
        System.out.println(user);

        // 7. 释放资源
        sqlSession.close();
    }
}
```

效果

测试结果如下图



实现根据用户名模糊查询用户

查询 sql：
SELECT * FROM `user` WHERE username LIKE '%王%'

方法一

映射文件

在 User.xml 配置文件中添加如下内容：

```
    <!-- 如果返回多个结果，mybatis 会自动把返回的结果放在 list 容器中 -->
    <!-- resultType 的配置和返回一个结果的配置一样 -->
    <select id="queryUserByUsername1" parameterType="string"
        resultType="com.haust.mybatis.pojo.User">
        SELECT * FROM `user` WHERE username LIKE #{username}
    </select>
```

测试程序

MybatisTest 中添加测试方法如下：

```
    @Test
    public void testQueryUserByUsername1() throws Exception {
        // 4. 创建 SqlSession 对象
        SqlSession sqlSession = sqlSessionFactory.openSession();

        // 5. 执行 SqlSession 对象执行查询，获取结果 User
        // 查询多条数据使用 selectList 方法
        List<User> list =
sqlSession.selectList("test.queryUserByUsername1", "%王%");

        // 6. 打印结果
        for (User user : list) {
            System.out.println(user);
        }

        // 7. 释放资源
        sqlSession.close();
    }
```

效果

测试效果如下图：

```
<terminated> MybatisTest.testQueryUserByUsername1 [JUnit] D:\pack\jdk1.7.0_72_x86\bin\javaw.exe (2016年7月26
DEBUG [main] - ==>  Preparing: SELECT * FROM `user` WHERE username LIKE ?
DEBUG [main] - ==> Parameters: %王%(String)
DEBUG [main] - <==      Total: 2
User [id=1, username=王五, sex=2, birthday=null, address=null]
User [id=26, username=王五, sex=null, birthday=null, address=null]
```

方法二

映射文件：

在 User.xml 配置文件中添加如下内容：

```
    <!-- 如果传入的参数是简单数据类型，${}里面必须写 value -->
    <select id="queryUserByUsername2" parameterType="string"
        resultType="com.haust.mybatis.pojo.User">
```

```
        SELECT * FROM `user` WHERE username LIKE '%${value}%'
    </select>
```

测试程序：

MybatisTest 中添加测试方法如下：

```
@Test
public void testQueryUserByUsername2() throws Exception {
    // 4. 创建 SqlSession 对象
    SqlSession sqlSession = sqlSessionFactory.openSession();

    // 5. 执行 SqlSession 对象执行查询，获取结果 User
    // 查询多条数据使用 selectList 方法
    List<Object> list = sqlSession.selectList("queryUserByUsername2",
"王");

    // 6. 打印结果
    for (Object user : list) {
        System.out.println(user);
    }

    // 7. 释放资源
    sqlSession.close();
}
```

效果

测试结果如下图：

```
DEBUG [main] - ==>  Preparing: SELECT * FROM `user` WHERE username LIKE '%王%'
DEBUG [main] - ==> Parameters:
DEBUG [main] - <==      Total: 2
User [id=1, username=王五, sex=2, birthday=null, address=null]
User [id=26, username=王五, sex=null, birthday=null, address=null]
```

实现添加用户

使用的 sql：

INSERT INTO `user` (username,birthday,sex,address) VALUES

('黄忠','2016-07-26','1','三国')

映射文件：

在 User.xml 配置文件中添加如下内容：

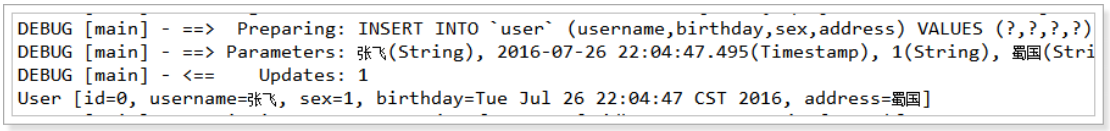
```
    <!-- 保存用户 -->
    <insert id="saveUser"
parameterType="com.haust.mybatis.pojo.User">
        INSERT INTO `user`
        (username,birthday,sex,address) VALUES
        (#{username},#{birthday},#{sex},#{address})
    </insert>
```

测试程序

MybatisTest 中添加测试方法如下：

```
@Test
public void testSaveUser() {
    // 4. 创建 SqlSession 对象
    SqlSession sqlSession = sqlSessionFactory.openSession();

    // 5. 执行 SqlSession 对象执行保存
    // 创建需要保存的 User
    User user = new User();
    user.setUsername("张飞");
    user.setSex("1");
    user.setBirthday(new Date());
    user.setAddress("蜀国");

    sqlSession.insert("test.saveUser", user);
    System.out.println(user);

    // 需要进行事务提交
    sqlSession.commit();

    // 7. 释放资源
    sqlSession.close();
}
```

效果



修改用户

根据用户 id 修改用户名

使用的 sql：
UPDATE `user` SET username = '赵云' WHERE id = 26

映射文件

在 User.xml 配置文件中添加如下内容：

```
<!-- 更新用户 -->
<update id="updateUserById"
parameterType="com.haust.mybatis.pojo.User">
    UPDATE `user` SET
    username = #{username} WHERE id = #{id}
</update>
```

测试程序

MybatisTest 中添加测试方法如下：

```
@Test
public void testUpdateUserById() {
    // 4. 创建 SqlSession 对象
    SqlSession sqlSession = sqlSessionFactory.openSession();

    // 5. 执行 SqlSession 对象执行更新
    // 创建需要更新的 User
    User user = new User();
    user.setId(26);
    user.setUsername("关羽");
```

```
    user.setSex("1");
    user.setBirthday(new Date());
    user.setAddress("蜀国");

    sqlSession.update("test.updateUserById", user);

    // 需要进行事务提交
    sqlSession.commit();

    // 7. 释放资源
    sqlSession.close();
}
```

效果

测试效果如下图：

```
DEBUG [main] - ==>  Preparing: UPDATE `user` SET username = ? WHERE id = ?
DEBUG [main] - ==> Parameters: 关羽(String), 26(Integer)
DEBUG [main] - <==    Updates: 1
```

删除用户

根据用户 id 删除用户

使用的 sql
DELETE FROM `user` WHERE id = 47

映射文件：

在 User.xml 配置文件中添加如下内容：

```
    <!-- 删除用户 -->
    <delete id="deleteUserById" parameterType="int">
        delete from user where
        id=#{id}
    </delete>
```

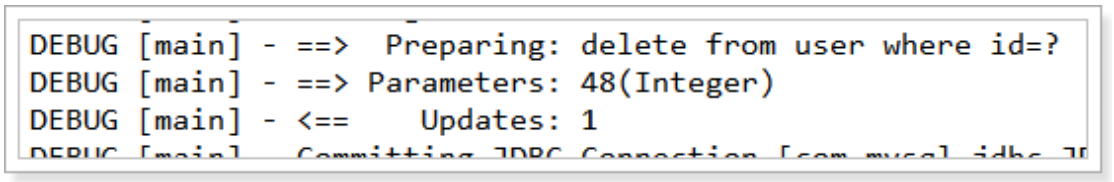
测试程序：

MybatisTest 中添加测试方法如下：

```
@Test
public void testDeleteUserById() {
    // 4. 创建 SqlSession 对象
    SqlSession sqlSession = sqlSessionFactory.openSession();

    // 5. 执行 SqlSession 对象执行删除
    sqlSession.delete("test.deleteUserById", 48);

    // 需要进行事务提交
    sqlSession.commit();

    // 7. 释放资源
    sqlSession.close();
}
```

效果

测试效果如下图：



五、 **思考题：**

无

六、 实验作业要求：

（一） 实验目的：

（二） 实验内容：

（三） 实验结果：可以是运行结果截图或其他形式的结果展示

（四） 问题及解决：实验中遇到的问题及解决方法。

回答思考题提出的问题。