Mybatis 快速入门

> 官网: MyBatis 中文网

需求

对用户信息的增删改查操作。
1、查询所有用户信息（查询所有）
2、根据用户 ID 来查询用户信息；(查询单条数据)
2、根据用户名称来模糊查询用户信息列表；(根据条件查询)
3、添加用户
4、删除用户
5、修改用户

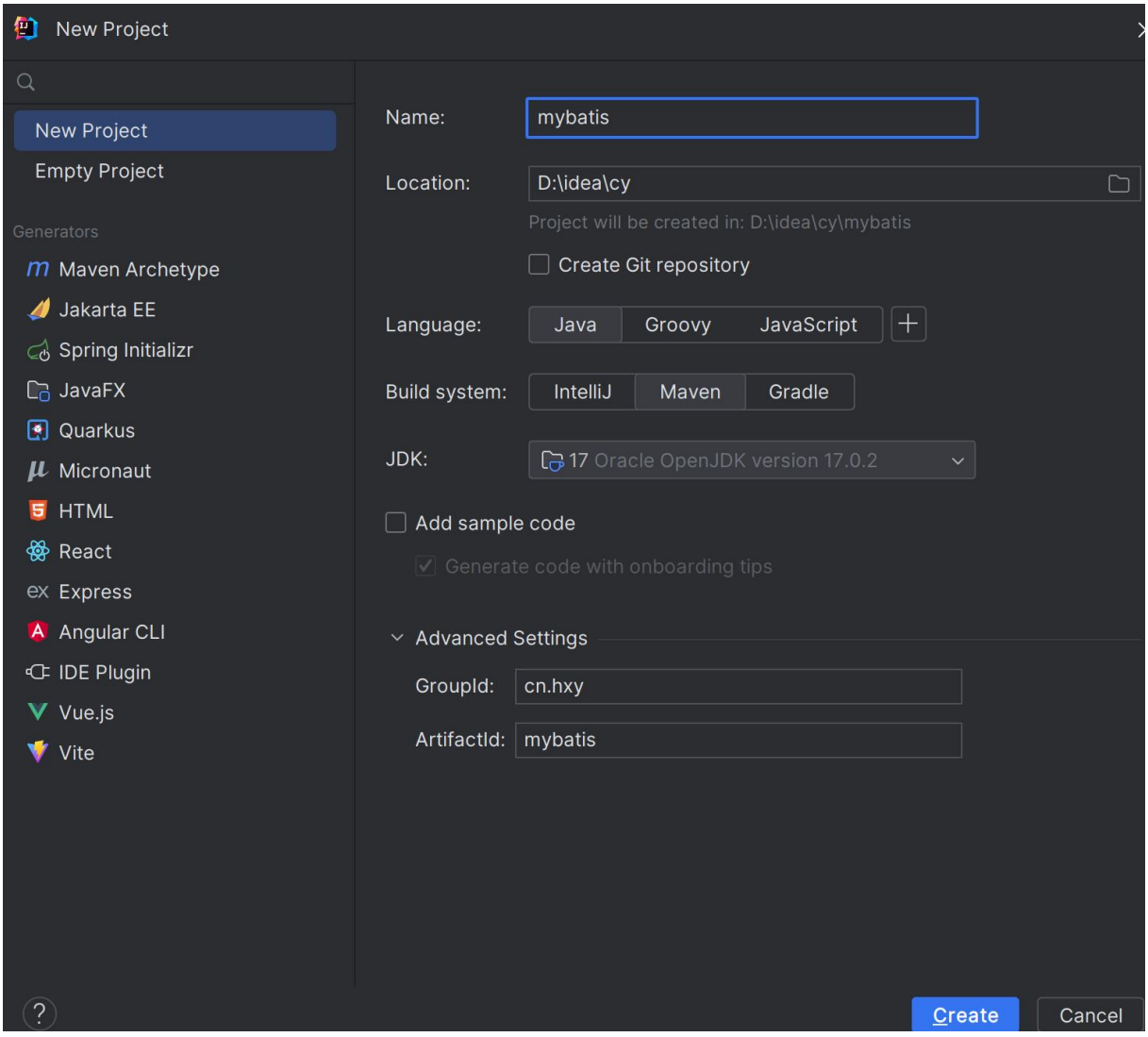
快速入门

> 1. 创建 user 表，添加数据
> 2. 创建模块，导入坐标
> 3. 编写 MyBatis 核心配置文件-- >替换连接信息解决硬编码问题
> 4. 编写 SQL 映射文件-->统一-管理 sql语句，解决硬编码问题
> 5. 编码：
>   a. 定义 POJ0 类
>   b. 加载核心配置文件， 获取 SqlSessionFactory 对象
>   c. 获取 SqlSession 对象，执行 SQL 语句
>   d. 释放资源

1. 创建 user 表,添加数据

```
CREATE TABLE `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(255) DEFAULT NULL,
  `birthday` date DEFAULT NULL,
   `sex` varchar(2) DEFAULT NULL,
  `home_address` varchar(255) DEFAULT NULL,
 PRIMARY KEY (`id`)
);
```

2. 创建对应的 maven 项目，导入依赖



maven: 项目管理工具/框架，帮助我们管理项目，管理依赖等

- 在以前开发，我们要用到一些 jar 包，每次都要手动的去网上下载 jar 包（拷贝到项目）添加到我们的项目依赖中
- 帮我们自动的将需要的 jar 包添加到项目依赖中
- 怎么添加： 通过在 pom.xml 文件中写入对应 jar 包的依赖

```
    <dependencies>
        <!--mybatis 依赖-->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.5.5</version>
        </dependency>

        <!--mysql 驱动-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <!--5.1.46-->
            <version>5.0.4</version>
        </dependency>

        <!--单元测试依赖-->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13</version>
            <scope>test</scope>
        </dependency>

        <!--添加 slf4j 日志 api-->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.7.30</version>
        </dependency>

        <!--添加 logback-classic 依赖-->
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.2.3</version>
        </dependency>

        <!--添加 logback-core 依赖-->
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-core</artifactId>
```

导入 logback.xml 到 resource 目录下

配置打印日志

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <!--    console 表示当前日志信息是可以输出到控制台的-->
    <appender name="Console"
class="ch.qos.logback.core.ConsoleAppender">
        <encoder>

<pattern>[%level]  %cyan([%thread]) %boldGreen(%logger{15}) -
 %msg %n</pattern>
        </encoder>
    </appender>
    <logger name="com.Carter_x" level="DEBUG" additivity="false">
        <appender-ref ref="Console"/>
    </logger>
</configuration>
```

3. 编写核心配置文件

在 resource 目录下创建核心配置文件 mybatis-config.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>
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="${driver}"/>
        <property name="url" value="${url}"/>
        <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
      </dataSource>
    </environment>
  </environments>
  <-- 加载 sql 映射文件-->
  <mappers>
    <mapper resource="org/mybatis/example/BlogMapper.xml"/>
  </mappers>
</configuration>
```

```
<dataSource type="POOLED">
                <property name="driver"
value="com.mysql.jdbc.Driver"/>
                <property name="url"
value="jdbc:mysql:///mybatis?useSSL=false"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
```

4. 编写 sql 映射文件 UserMapper.xml

创建对应的 User 实体类

```
<?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="testMy">
    <select id="selectAll" resultType="cn.hxy.pojo.User">
        select * from users
    </select>
</mapper>
```

注意别忘了修改 config.xml 中的映射文件地址

对象关系映射　java 类中的对象　=　数据库表对应的关系

5. 编码

1. 定义 pojo 类（实体类 entity/domain）
2. 加载核心配置文件，获取 SqlSessionFactory 对象
3. 获取 SqlSession 对象，执行 SQL 语句
4. 释放资源

1. **定义 pojo 实体类**

```
package cn.hxy.entity;

import java.util.Date;

public class User {
    private Integer id;
    private String username;
    private Date birthday;
    private String sex;
    private String address;

    public User() {
    }

    public User(Integer id, String username, Date birthday,
String sex, String address) {
        this.id = id;
        this.username = username;
        this.birthday = birthday;
        this.sex = sex;
        this.address = address;
    }

    public Integer getId() {
        return id;
    }

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

    public String getUsername() {
        return username;
    }

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

    public Date getBirthday() {
        return birthday;
    }
```

2. 创建 main 类加载核心配置文件

从官网拷贝获取 SqlSessionFactory 的代码

```
    public static void main(String[] args) throws IOException {
          //1. 加载核心配置文件，获取 SqlSessionFactory
          String resource = "mybatis-config.xml";
          InputStream inputStream =
Resources.getResourceAsStream(resource);
          SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);

          // 2.获取 SqlSession 对象
          SqlSession sqlSession = sqlSessionFactory.openSession();

          //3.执行 sql 语句
          List<User> users =
sqlSession.selectList("testMy.selectAll");
          System.out.println(users);

          //4. 释放资源
          sqlSession.close();
      }
```

1. 定义了一个包路径，并且创建了对应的 pojo 实体类
2. 设置了 sqlSessionFactory
3. 创建 sqlSession 执行 sql 语句
4. 改写 sql 映射文件中的 sql
5. 打印查询到所有数据，并且关闭 sqlSession

Mapper 代理开发

- 解决原生方式的硬编码
- 简化后期执行 sql

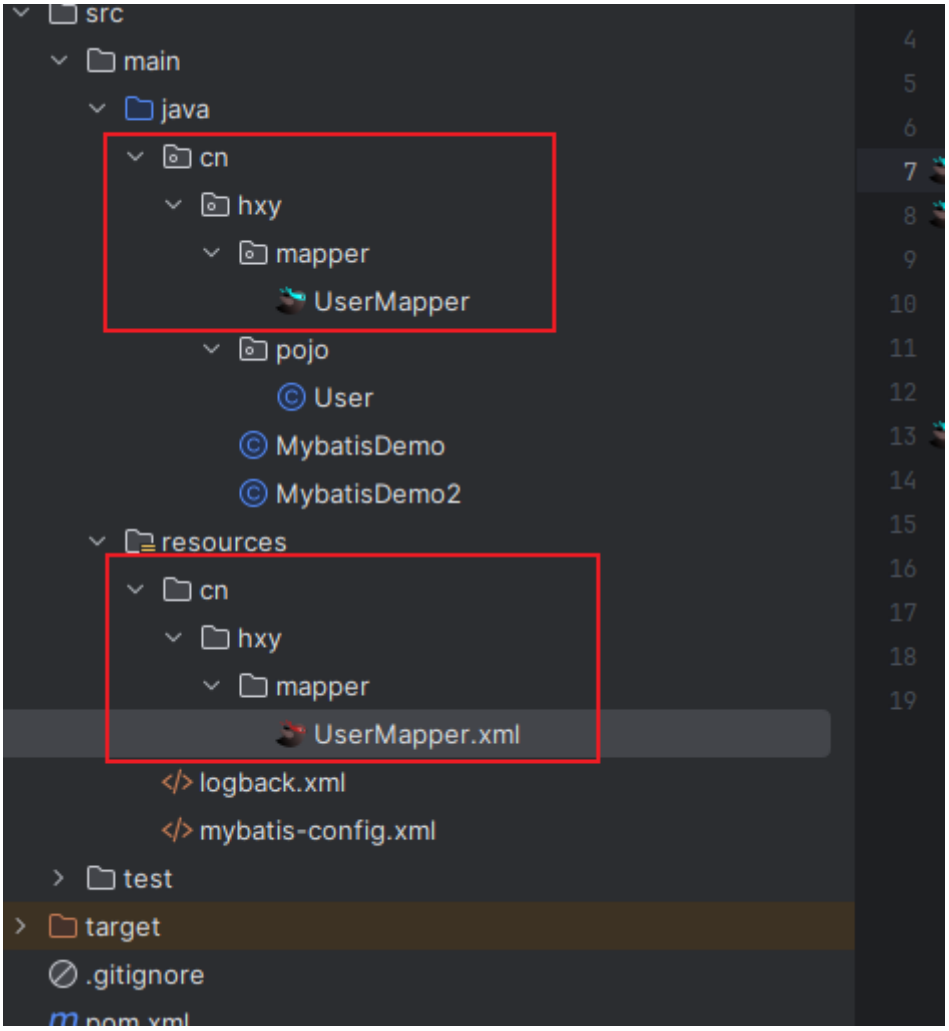
步骤:

1. 定义与 SQL 映射文件同名的 Mapper 接口，并且将 Mapper 接口和 SQL 映射文件放置在同一目录下
2. 设置 SQL 映射文件的 namespace 属性为 Mapper 接口全限定名
3. 在 Mapper 接口中定义方法，方法名就是 SQL 映射文件中 sql|语句的 id,并保持参数类型和返回值类型一致
4. 编码
    a. 通过 SqlSession 的 getMapper 方法获取 Mapper 接口的代理对象
    b. 调用对应方法完成 sqI 的执行

1. 创建 UserMapper 接口

注意将对应的 sql 映射文件和该 UserMapper 借口放在同一目录下

创建对应的层级目录



2. 设置 SQL 映射文件的 namespace 属性为 Mapper 接口全限定名

```
<?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="cn.hxy.mapper.UserMapper">
    <select id="selectAll" resultType="cn.hxy.pojo.User">
        select * from users
    </select>
</mapper>
```

3. 在 UserMapper 接口中定义方法

```
public interface UserMapper {
    List<User> selectAll();
}
```

插件：mybatisX

注意: 修改核心配置文件中的 mapper 路径

4. 编码

1. 通过 SqlSession 的 getMapper 方法获取 Mapper 接口的代理对象
2. 调用对应方法完成 sqI 的执行

```
 public static void main(String[] args) throws IOException {
        //加载核心配置文件，获取 SqlSessionFactory
        String resource = "mybatis-config.xml";
        InputStream inputStream =
Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);

        // 获取 SqlSession 对象
        SqlSession sqlSession = sqlSessionFactory.openSession();

        //执行 sql 语句
        //获取 UserMapper 接口的代理对象
        UserMapper mapper =
sqlSession.getMapper(UserMapper.class);
        List<User> users = mapper.selectAll();
        System.out.println(users);
        //释放资源
        sqlSession.close();
    }
```

5. 代理开发扫描包加载映射文件

```
    <package name="cn.hxy.mapper"/>
```

核心配置文件 mybatis-config.xml

- 环境配置
- 事务配置
- 数据源配置
- 别名配置

别名配置

```
      <typeAliases>
            <package name="cn.hxy.pojo"/>
        </typeAliases>
```

注意:配置有前后顺序,否则会报错

配置文件完成增删改查

1. 查询所有

- sql 语句怎么写
- 要不要参数
- 返回结果是什么
- test 进行测试

数据库字段和实体类字段名不一致怎么处理

1. 起别名(但是如果字段很多表很多就很复杂)

```
    <select id="selectAll" resultType="User">
            select id,username,birthday,sex,home_address as
    homeAddress from users
        </select>
```

2. 抽取 sql 片段(确定不灵活,不同的 sql 语句需要查询不同的字段)

```
     <sql id="userSql">
            id,username,birthday,sex,home_address homeAddress
        </sql>
        <select id="selectAll" resultType="User">
            select
                <include refid="userSql" />
                from users
        </select>
```

上面两种方式都有一定的缺点,所以我们可以使用 resultMap

```
    <resultMap id="userResultMap" type="user">
           <result column="home_address" property="homeAddress"/>
       </resultMap>

       <select id="selectAll" resultMap="userResultMap">
           select * from users
       </select>
```

2. 查询单条数据

- sql 语句怎么写
- 要不要参数
- 返回结果是什么
- test 进行测试

定义 mapper 接口中的查询方法

```
    User selectOne(Integer id);
```

定义 sql 映射文件中查询一条数据的 statement

注意

- #{}的作用: 占位符,可以防止 sql 注入,在构建 sql 语句时使用' ? '替代对应的查询条件参数

  select * from users where id = ?
- 传入参数: 可以忽略

```
    <select id="selectOne" resultType="cn.hxy.pojo.User">
           select * from users where id=#{id}
       </select>
```

测试类进行测试

```
    @Test
    public void testSelectAOne() throws IOException {
        //模拟前端点击某条数据，传入该条数据的 id 为 2
        Integer id = 2;

        String resource = "mybatis-config.xml";
        InputStream inputStream =
Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
        SqlSession sqlSession = sqlSessionFactory.openSession();
        UserMapper mapper =
sqlSession.getMapper(UserMapper.class);
        //调用 mapper 接口中的 selectOne 方法，根据传入 id 查询一条数据
        User users = mapper.selectOne(id);
        System.out.println(users);
        sqlSession.close();
    }
```

3. 条件查询

定义接口方法

```
 //通过注解的方式
 List<User> selectByCondition(@Param("username") String username,
@Param("homeAddress") String homeAddress);

 //通过对象的方式
 //通过 map 的方式
 List<User> selectByCondition(Map map);
```

定义 sql 语句

```
 <select id="selectByCondition" resultMap="userResultMap">
        select * from users
        where username like #{username}
        and home_address like #{homeAddress}
    </select>
```

定义测试类测试

传入单个参数

```
@Test
    public void testSelectByCondition() throws IOException {
        String username = "张";
        String homeAddress = "成都";
        username = "%" + username + "%";
        homeAddress = "%" + homeAddress + "%";
        String resource = "mybatis-config.xml";
        InputStream inputStream =
Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
        SqlSession sqlSession = sqlSessionFactory.openSession();
        UserMapper mapper =
sqlSession.getMapper(UserMapper.class);
        List<User> users = mapper.selectByCondition(username,
homeAddress);
        System.out.println(users);
        sqlSession.close();
    }
```

传入 map

```
    @Test
        public void testSelectByCondition() throws IOException {
            String username = "张";
            String homeAddress = "成都";
            username = "%" + username + "%";
            homeAddress = "%" + homeAddress + "%";

            HashMap map = new HashMap();
            map.put("username", username);
            map.put("homeAddress", homeAddress);


            String resource = "mybatis-config.xml";
            InputStream inputStream =
    Resources.getResourceAsStream(resource);
            SqlSessionFactory sqlSessionFactory = new
    SqlSessionFactoryBuilder().build(inputStream);
            SqlSession sqlSession = sqlSessionFactory.openSession();
            UserMapper mapper =
    sqlSession.getMapper(UserMapper.class);
            List<User> users = mapper.selectByCondition(map);
            System.out.println(users);
            sqlSession.close();
        }
```

4. 动态条件查询

> 在上面的条件查询中,存在一个问题,如果用户没有传入条件参数,获取传入了某一个几个条件参数,该怎么写 sql 语句,我们需要让映射文件中的 sql 语句根据条件参数是否传入生成

动态 sql 语句官网: [动态 SQL_MyBatis 中文网]

```
    <select id="selectByCondition" resultMap="userResultMap">
        select * from users
        <where>
            <if test="username!=null and username!=''">
                username like #{username}
            </if>
            <if test="homeAddress!=null and homeAddress!=''">
                and home_address like #{homeAddress}
            </if>
        </where>

    </select>
```

5. 添加

注意提交事务

```
    sqlSession.commit();
```

添加返回主键 id

6. 修改/动态修改

7. 删除/批量删除