# 10. MyBatis-Plus使用

## 本节目标

1. 学习使用MyBatis-Plus完成基础的增删改查
2. 学习使用MyBatis-Plus的条件构造器以及自定义SQL完成一些复杂的查询

# 1. MyBatis-Plus介绍

MyBatis-Plus(简称 MP) 是一个 MyBatis 的增强工具, 在 MyBatis 的基础上只做增强不做改变, 为简化开发. 提高效率而生

**特性:**

- 润物无声: 只做增强不做改变，引入它不会对现有工程产生影响，如丝般顺滑.
- 效率至上: 只需简单配置，即可快速进行单表 CRUD 操作，从而节省大量时间.
- 丰富功能: 代码生成、自动分页、逻辑删除、自动填充、拦截器等功能一应俱全.
- 广泛认可: 连续 5 年获得开源中国年度最佳开源项目殊荣，Github 累计 16K Star.

**支持数据库:**

PostgreSQL, MySQL, MariaDB, Oracle, SQL Server, OceanBase, H2, DB2...

(任何能使用 MyBatis 进行增删改查，并且支持标准 SQL 的数据库应该都在 MyBatis-Plus 的支持范围内)

**官网地址**: MyBatis-Plus 🚀 为简化开发而生

**愿景**

我们的愿景是成为 MyBatis 最好的搭档，就像 **魂斗罗** 中的 1P、2P，基友搭配，效率翻倍。


# 2. 快速上手

Mybatis-Plus操作数据库的步骤：

1. 准备工作(数据准备, 项目准备, 引入依赖, 配置数据库连接信息)
2. 编码(数据库表对应的实体类, 以及数据操作的Mapper文件)
3. 测试

## 2.1 准备工作

### 2.1.1 数据准备

创建用户表, 并创建对应的实体类User

> 继续使用MyBatis学习阶段的数据库表就可以

```sql
-- 创建数据库
DROP DATABASE IF EXISTS mybatis_test;

CREATE DATABASE mybatis_test DEFAULT CHARACTER SET utf8mb4;

-- 使用数据数据
USE mybatis_test;

-- 创建表[用户表]
DROP TABLE IF EXISTS user_info;
```

```sql
CREATE TABLE `user_info` (
        `id` INT ( 11 ) NOT NULL AUTO_INCREMENT,
        `username` VARCHAR ( 127 ) NOT NULL,
        `password` VARCHAR ( 127 ) NOT NULL,
        `age` TINYINT ( 4 ) NOT NULL,
        `gender` TINYINT ( 4 ) DEFAULT '0' COMMENT '1-男 2-女 0-默认',
        `phone` VARCHAR ( 15 ) DEFAULT NULL,
        `delete_flag` TINYINT ( 4 ) DEFAULT 0 COMMENT '0-正常, 1-删除',
        `create_time` DATETIME DEFAULT now(),
        `update_time` DATETIME DEFAULT now(),
        PRIMARY KEY ( `id` )
) ENGINE = INNODB DEFAULT CHARSET = utf8mb4;

-- 添加用户信息
INSERT INTO mybatis_test.user_info( username, `password`, age, gender, phone )
VALUES ( 'admin', 'admin', 18, 1, '18612340001' );
INSERT INTO mybatis_test.user_info( username, `password`, age, gender, phone )
VALUES ( 'zhangsan', 'zhangsan', 18, 1, '18612340002' );
INSERT INTO mybatis_test.user_info( username, `password`, age, gender, phone )
VALUES ( 'lisi', 'lisi', 18, 1, '18612340003' );
INSERT INTO mybatis_test.user_info( username, `password`, age, gender, phone )
VALUES ( 'wangwu', 'wangwu', 18, 1, '18612340004' );
```

### 2.1.2 项目准备

1. 创建springboot工程
2. 添加MyBatis-Plus和MySQL依赖, 配置数据库连接信息

**Spring Boot2**

```xml
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.5.7</version>
</dependency>
```

**Spring Boot3**

```xml
<dependency>
  <groupId>com.baomidou</groupId>
  <artifactId>mybatis-plus-spring-boot3-starter</artifactId>
  <version>3.5.5</version>
</dependency>
```

**MySQL**

```
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
```

**配置数据库**

application.yml文件, 配置内容如下:

```
# 数据库连接配置
spring:
  datasource:
    url: jdbc:mysql://127.0.0.1:3306/mybatis_test?characterEncoding=utf8&useSSL=false
    username: root
    password: root
    driver-class-name: com.mysql.cj.jdbc.Driver
```

application.properties文件, 配置内容如下:

```
#驱动类名称
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#数据库连接的url
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/mybatis_test?characterEncoding=utf8&useSSL=false
#连接数据库的用户名
spring.datasource.username=root
#连接数据库的密码
spring.datasource.password=root
```

## 2.2 编码

创建实体类 UserInfo

> 实体类的属性名与表中的字段名一一对应

```
import lombok.Data;
```

```java
import java.util.Date;

@Data
public class UserInfo {
    private Integer id;
    private String username;
    private String password;
    private Integer age;
    private Integer gender;
    private String phone;
    private Integer deleteFlag;
    private Date createTime;
    private Date updateTime;
}
```

编写Mapper接口类

> MybatisPlus提供了一个基础的 `BaseMapper` 接口，已经实现了单表的CRUD, 我们自定义的Mapper只需要继承这个BaseMapper, 就无需自己实现单表CRUD了


```java
@Mapper
public interface UserInfoMapper extends BaseMapper<UserInfo> {
}
```

> 也可以在启动类上添加 `@MapperScan`, 扫描Mapper文件夹, 二选一即可.

## 2.3 CRUD单元测试

在创建出来的SpringBoot工程中，在src下的test目录下，已经自动帮我们创建好了测试类，我们可以直接使用这个测试类来进行测试.

编写几个单元测试, 测试基本的CRUD功能

```
@SpringBootTest
class MybatisPlusDemoApplicationTests {

    @Autowired
    private UserInfoMapper userInfoMapper;

    @Test
    void testInsert() {
        UserInfo user = new UserInfo();
        user.setUsername("bite");
        user.setPassword("123456");
        user.setAge(11);
        user.setGender(0);
        user.setPhone("18610001234");
        userInfoMapper.insert(user);
    }

    @Test
    void testSelectById() {
        UserInfo user = userInfoMapper.selectById(1L);
        System.out.println("user:  " + user);
    }

    @Test
    void testSelectByIds() {
        List<UserInfo> users = userInfoMapper.selectBatchIds(List.of(1L, 2L, 3L,
4L));
        users.forEach(System.out::println);
    }

    @Test
    void testUpdateById() {
        UserInfo user = new UserInfo();
        user.setId(1);
        user.setPassword("4444444");
        userInfoMapper.updateById(user);
    }
```

```
37
38      @Test
39      void testDelete() {
40          userInfoMapper.deleteById(5L);
41      }
42
43  }
```

运行结果如下:

```
UserInfo(id=1, username=admin, password=4444444, age=18, gender=1, phone=18612340001, deleteFlag=0, createTime=Thu Jul 18 17:34:13
 CST 2024, updateTime=Thu Jul 18 17:34:13 CST 2024)
UserInfo(id=2, username=zhangsan, password=zhangsan, age=18, gender=1, phone=18612340002, deleteFlag=0, createTime=Thu Jul 18
 17:34:13 CST 2024, updateTime=Thu Jul 18 17:34:13 CST 2024)
UserInfo(id=3, username=lisi, password=lisi, age=18, gender=1, phone=18612340003, deleteFlag=0, createTime=Thu Jul 18 17:34:13 CST
 2024, updateTime=Thu Jul 18 17:34:13 CST 2024)
UserInfo(id=4, username=wangwu, password=wangwu, age=18, gender=1, phone=18612340004, deleteFlag=0, createTime=Thu Jul 18 17:34:13
 CST 2024, updateTime=Thu Jul 18 17:34:13 CST 2024)
user:  UserInfo(id=1, username=admin, password=4444444, age=18, gender=1, phone=18612340001, deleteFlag=0, createTime=Thu Jul 18
 17:34:13 CST 2024, updateTime=Thu Jul 18 17:34:13 CST 2024)
2024-07-19T11:22:33.977+08:00  INFO 22824 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown
 initiated...
2024-07-19T11:22:33.985+08:00  INFO 22824 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown
 completed.
```

数据的CRUD操作, 全部执行成功了

# 3. MyBatis-Plus复杂操作

## 3.1 常见注解

在上面的程序中, MyBatis是如何知道, 我们要操作的是哪张表, 表里有哪些字段呢?

我们来看下咱们Mapper的代码

```
1  @Mapper
2  public interface UserInfoMapper extends BaseMapper<UserInfo> {
3
4  }
```

UserInfoMapper 在继承 `BaseMapper` 时, 指定了一个泛型, 这个UserInfo就是与数据库表相对应的实体类.

MyBatis-Plus会根据这个实体类来推断表的信息.

默认情况下:

1. 表名: 实体类的驼峰表示法转换成蛇形表示法(下划线分割), 作为表名. 比如UserInfo -> user_info
2. 字段: 根据实体类的属性名 转换为蛇形表示法作为字段名. 比如deleteFlag -> delete_flag

3. 主键: 默认为id

那如果实体类和数据库不是按照上述规则定义的呢? MyBatis-Plus也给我们提供了一下注解, 让我们标识表的信息.

### 3.1.1 @TableName

修改实体类名UserInfo为 Userinfo, 重新执行测试方法testSelectById

运行结果:

```
### SQL: SELECT id,username,password,age,gender,phone,delete_flag,create_time,update_time FROM userinfo WHERE id=?
### Cause: java.sql.SQLSyntaxErrorException: Table 'mybatis_test.userinfo' doesn't exist
; bad SQL grammar []
```

从日志可以看到, 默认查找的表名为userinfo.

我们可以通过 `@TableName` 来标识实体类对应的表

```java
@Data
@TableName("user_info")
public class Userinfo {
    private Integer id;
    private String username;
    private String password;
    private Integer age;
    private Integer gender;
    private String phone;
    private Integer deleteFlag;
    private Date createTime;
    private Date updateTime;
}
```

再次运行程序, 程序运行结果正常.

### 3.1.2 @TableField

修改属性名 deleteFlag 为 deleteflag, 重新执行测试方法testSelectById

运行结果:

```
### SQL: SELECT id,username,password,age,gender,phone,deleteflag,create_time,update_time FROM user_info WHERE id=?
### Cause: java.sql.SQLSyntaxErrorException: Unknown column 'deleteflag' in 'field list'
; bad SQL grammar []
```

从日志可以看到, 根据属性名转换后的字段名为: deleteflag.

我们可以通过 `@TableField` 来标识 对应的字段名

```
@Data
@TableName("user_info")
public class Userinfo {
    private Integer id;
    private String username;
    private String password;
    private Integer age;
    private Integer gender;
    private String phone;
    @TableField("delete_flag")
    private Integer deleteflag;
    private Date createTime;
    private Date updateTime;
}
```

再次运行程序, 程序运行结果正常.

### 3.1.3 @TableId

修改属性名 id 为 userId, 重新执行测试方法testSelectById

运行结果:

```
org.apache.ibatis.binding.BindingException: Invalid bound statement (not found): com.bite.mybatis.plus.mapper.UserInfoMapper
.selectById

    at org.apache.ibatis.binding.MapperMethod$SqlCommand.<init>(MapperMethod.java:229)
    at com.baomidou.mybatisplus.core.override.MybatisMapperMethod.<init>(MybatisMapperMethod.java:50)
    at com.baomidou.mybatisplus.core.override.MybatisMapperProxy.lambda$cachedInvoker$0(MybatisMapperProxy.java:111)
    at java.base/java.util.concurrent.ConcurrentHashMap.computeIfAbsent(ConcurrentHashMap.java:1708)
```

我们可以通过 `@TableId` 来 指定对应的主键

```
@Data
@TableName("user_info")
public class Userinfo {
    @TableId("id")
    private Integer userId;
    private String username;
    private String password;
    private Integer age;
    private Integer gender;
    private String phone;
    @TableField("delete_flag")
```

```
12      private Integer deleteflag;
13      private Date createTime;
14      private Date updateTime;
15  }
```

> 如果属性名和字段名不一致, 需要在 `@TableId` 指明对应的字段名
>
> 属性名和字段一致的情况下, 直接加 `@TableId` 注解就可以.

再次运行程序, 程序运行结果正常.

## 3.2 打印日志

为了方便学习, 我们可以借助日志, 查看Mybatis-Plus执行的SQL语句, 参数和执行结果

Mybatis-Plus配置日志如下:

```yaml
1 mybatis-plus:
2   configuration: # 配置打印 MyBatis日志
3     log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
```

## 3.3 条件构造器

入门程序里的使用, 都是简单的CRUD, 在实际的应用场景中, 我们还需要使用更复杂的操作, MyBatis-Plus 也给我们提供了相应的支持.

MyBatis-Plus 提供了一套强大的条件构造器(Wrapper), 用于构建复杂的数据库查询条件. Wrapper 类允许开发者以链式调用的方式构造查询条件, 无需编写繁琐的 SQL 语句, 从而提高开发效率并减少 SQL 注入的风险.

以下是主要的 Wrapper 类及其功能:

- **AbstractWrapper**：这是一个抽象基类, 提供了所有 Wrapper 类共有的方法和属性. 详细参考官网介绍: 条件构造器
- **QueryWrapper**：用于构造查询条件, 在AbstractWrapper的基础上拓展了一个select方法, 允许指定查询字段.
- **UpdateWrapper:** 用于构造更新条件, 可以在更新数据时指定条件.
- **LambdaQueryWrapper**：基于 Lambda 表达式的查询条件构造器, 它通过 Lambda 表达式来引用实体类的属性，从而避免了硬编码字段名.
- **LambdaUpdateWrapper:** 基于 Lambda 表达式的更新条件构造器, 它允许你使用 Lambda 表达式来指定更新字段和条件，同样避免了硬编码字段名的问题.

接下来通过代码来看下如何使用.

### 3.3.1 QueryWrapper

QueryWrapper并不只用于查询语句, 无论是修改, 删除, 查询, 都可以使用QueryWrapper来构建查询条件.

**查询**

完成下述SQL查询

```
SELECT id,username,password,age FROM user_info WHERE age = 18 AND username
"%min%"
```

测试代码

```
@Test
void testQueryWrapper(){
    QueryWrapper<UserInfo> userInfoQueryWrapper = new QueryWrapper<UserInfo>()
            .select("id","username","password","age")
            .eq("age",18)
            .like("username", "min");
    List<UserInfo> userInfos = userInfoMapper.selectList(userInfoQueryWrapper);
    userInfos.forEach(System.out::println);
}
```

> **注意:**
>
> 默认情况下Mybatis-Plus会根据 `@TableFiled` 生成别名, 当指定了QueryWrapper的select属性后就仅仅是属性值而没有了别名. 查询出来的结果会对应不上
>
> 解决办法:
>
> 1. 自己写自定义SQL
> 2. 实体类名和字段名保持一致
> 3. 不指定QueryWrapper的select字段
> 4. 使用LambdaQueryWrapper实现
>
> 其中1和4下面讲解

**更新**

完成下述SQL查询

```
UPDATE user_info SET delete_flag=? WHERE age < 20
```

测试代码

```
@Test
void testUpdateByQueryWrapper(){
    QueryWrapper<UserInfo> userInfoQueryWrapper = new QueryWrapper<UserInfo>()
          .lt("age", 20);
    UserInfo userInfo = new UserInfo();
    userInfo.setDeleteFlag(1);
    userInfoMapper.update(userInfo, userInfoQueryWrapper);
}
```

- `lt` : "less than" 的缩写，表示小于.
- `le` : "less than or equal to"的缩写, 表示小于等于
- `ge` : "greater than or equal to" 的缩写, 表示大于等于.
- `gt` : "greater than" 的缩写, 表示大于.
- `eq` : "equals" 的缩写, 表示等于.
- `ne` : "not equals" 的缩写, 表示不等于.

**删除**

完成下述SQL查询

```
DELETE FROM user_info WHERE age = 18
```

测试代码

```
@Test
void testDeleteByQueryWrapper(){
    QueryWrapper<UserInfo> userInfoQueryWrapper = new QueryWrapper<UserInfo>()
          .eq("age",18);
    userInfoMapper.delete(userInfoQueryWrapper);
}
```

## 3.3.2 UpdateWrapper

对于更新, 我们也可以直接使用 UpdateWrapper, 在不创建实体对象的情况下, 直接设置更新字段和条件.

**基础更新:**

完成下述SQL查询

```
UPDATE user_info SET delete_flag=0, age=5 WHERE id IN (1,2,3)
```

测试代码

```
@Test
void testUpdateByUpdateWrapper(){
    UpdateWrapper<UserInfo> updateWrapper = new UpdateWrapper<UserInfo>()
            .set("delete_flag",0)
            .set("age", 5)
            .in("id", List.of(1,2,3));
    userInfoMapper.update(updateWrapper);
}
```

**基于SQL更新:**

完成下述SQL查询

```
UPDATE user_info SET age = age+10 WHERE id IN (1,2,3)
```

测试代码

```
@Test
void testUpdateBySQLUpdateWrapper(){
    UpdateWrapper<UserInfo> updateWrapper = new UpdateWrapper<UserInfo>()
            .setSql("age = age+10")
            .in("id", List.of(1,2,3));
    userInfoMapper.update(updateWrapper);
}
```

### 3.3.3 LambdaQueryWrapper

QueryWrapper 和 UpdateWrapper存在一个问题, 就是需要写死字段名, 如果字段名发生变更, 可能会因为测试不到位酿成事故.

MyBatis-Plus 给我们提供了一种基于Lambda表达式的条件构造器, 它通过 Lambda 表达式来引用实体类的属性，从而避免了硬编码字段名, 也提高了代码的可读性和可维护性.

- LambdaQueryWrapper
- LambdaUpdateWrapper

分别对应上述的QueryWrapper和UpdateWrapper

接下来我们看下具体使用.

```
@Test
void testLambdaQueryWrapper(){
    QueryWrapper<UserInfo> queryWrapper = new QueryWrapper<UserInfo>();
    queryWrapper.lambda()
          .select(UserInfo::getUsername, UserInfo::getPassword,UserInfo::getAge)
          .eq(UserInfo::getUserId, 1);
    userInfoMapper.selectList(queryWrapper).forEach(System.out::println);
}
```

### 3.3.4 LambdaUpdateWrapper

- LambdaUpdateWrapper用法和 LambdaQueryWrapper相似

```
@Test
void testLambdUpdateByUpdateWrapper(){
    UpdateWrapper<UserInfo> updateWrapper = new UpdateWrapper<UserInfo>();
    updateWrapper.lambda()
          .set(UserInfo::getDeleteFlag, 0)
          .set(UserInfo::getAge, 5)
          .in(UserInfo::getUserId, List.of(1,2,3));
    userInfoMapper.update(updateWrapper);
}
```

## 3.4 自定义SQL

在实际的开发中, MyBatis-Plus提供的操作不能满足我们的实际需求, MyBatis-Plus 也提供了自定义SQL的功能, 我们可以利用Wrapper构造查询条件, 再结合Mapper编写SQL

> 📌 为了使用这一功能， `mybatis-plus` 版本不低于 `3.0.7`

## 代码示例1

完成下述SQL查询

```
select id,username,password,age FROM user_info WHERE username = "admin"
```

Mapper:

```
@Mapper
public interface UserInfoMapper extends BaseMapper<UserInfo> {
    @Select("select id,username,password,age FROM user_info ${ew.customSqlSegment}")
    List<UserInfo> queryUserByCustom(@Param(Constants.WRAPPER) Wrapper<UserInfo> wrapper);
}
```

测试代码

```
@Test
void testQueryUserByCustom(){
   QueryWrapper<UserInfo> queryWrapper = new QueryWrapper<UserInfo>()
         .eq("username","admin");
   userInfoMapper.queryUserByCustom(queryWrapper).forEach(System.out::println);
}
```

**注意事项:**

- **参数命名**：在自定义 SQL 时, 传递 Wrapper 对象作为参数时, 参数名必须为 `ew`，或者使用注解 `@Param(Constants.WRAPPER)` 明确指定参数为 Wrapper 对象.
- **使用 `${ew.customSqlSegment}`**：在 SQL 语句中，使用 `${ew.customSqlSegment}` 来引用 Wrapper 对象生成的 SQL 片段.
- **不支持基于 entity 的 where 语句**：自定义 SQL 时，Wrapper 对象不会基于实体类自动生成 where 子句，你需要手动编写完整的 SQL 语句.

## 代码示例2

MyBatis-Plus 在 MyBatis 的基础上只做增强不做改变, 所以也支持XML的实现方式

上述功能也可以使用XML的方式完成

1. 配置mapper路径

```yaml
mybatis-plus:
  mapper-locations: "classpath*:/mapper/**.xml" # Mapper.xml
```

2. 定义方法

```java
@Mapper
public interface UserInfoMapper extends BaseMapper<UserInfo> {
    List<UserInfo> queryUserByCustom2(@Param(Constants.WRAPPER)
Wrapper<UserInfo> wrapper);
}
```

3. 编写XML

```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.bite.mybatis.plus.mapper.UserInfoMapper">

    <select id="queryUserByCustom2">
        select id,username,password,age FROM user_info ${ew.customSqlSegment}
    </select>
</mapper>
```

4. 测试

```java
@Test
void testQueryUserByCustom2(){
   QueryWrapper<UserInfo> queryWrapper = new QueryWrapper<UserInfo>()
         .eq("username","admin");

userInfoMapper.queryUserByCustom2(queryWrapper).forEach(System.out::println);
}
```

## 代码示例3

完成下述SQL查询

```
UPDATE user_info SET age = age+10 WHERE id IN (1,2,3)
```

Mapper:

```
@Update("UPDATE user_info SET age = age+ #{addAge}  ${ew.customSqlSegment}")
void updateUserByCustom(@Param("addAge") int addAge, @Param("ew")
Wrapper<UserInfo> wrapper);
```

测试代码

```
@Test
void updateUserByCustom(){
    QueryWrapper<UserInfo> queryWrapper = new QueryWrapper<UserInfo>()
          .in("id",List.of(1,2,3));
    userInfoMapper.updateUserByCustom(10, queryWrapper);
}
```

## 4. 总结

1. MyBatis-Plus 是 MyBatis 的增强工具, 在 MyBatis 的基础上只做增强不做改变, 可以用更少的代码实现数据库表的CRUD, 让我们的开发变得更加简单.
2. MyBatis-Plus 支持自定义SQL, 版本不低于3.0.7, 传递 Wrapper 对象作为参数时, 参数名必须为 ew, 在 SQL 语句中，使用 ${ew.customSqlSegment} 来引用 Wrapper 对象生成的 SQL 片段