MyBatis-Plus

一、简介

> 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。
>
> 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。

> 官网地址：https://mp.baomidou.com/

MyBatis-Plus

为简化开发而生

快速开始 →

润物无声	效率至上	丰富功能
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。	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。	代码生成、自动分页、逻辑删除、自动填充等功能一应俱全。

特性

- **无侵入**：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
- **损耗小**：启动即会自动注入基本 CRUD，性能基本无损耗，直接面向对象操作
- **强大的 CRUD 操作**：内置通用 Mapper、通用 Service，仅仅通过少量配置即可实现**单表大部分 CRUD 操作**，更有强大的条件构造器，满足各类使用需求
- **支持 Lambda 形式调用**：通过 Lambda 表达式，方便的编写各类查询条件，无需再担心字段写错
- **支持主键自动生成**：支持多达 4 种主键策略（内含分布式唯一 ID 生成器 - Sequence），可自由配置，完美解决主键问题
- **支持 ActiveRecord 模式**：支持 ActiveRecord 形式调用，实体类只需继承 Model 类即可进行强大的 CRUD 操作
- **支持自定义全局通用操作**：支持全局通用方法注入（ Write once, use anywhere ）
- **内置代码生成器**：采用代码或者 Maven 插件可快速生成 Mapper 、 Model 、 Service 、 Controller 层代码，支持模板引擎，更有超多自定义配置等您来使用
- **内置分页插件**：基于 MyBatis 物理分页，开发者无需关心具体操作，配置好插件之后，写分页等同于普通 List 查询
- **分页插件支持多种数据库**：支持 MySQL、MariaDB、Oracle、DB2、H2、HSQL、SQLite、Postgre、SQLServer 等多种数据库
- **内置性能分析插件**：可输出 Sql 语句以及其执行时间，建议开发测试时启用该功能，能快速揪出慢查询
- **内置全局拦截插件**：提供全表 delete 、 update 操作智能分析阻断，也可自定义拦截规则，预防误操作

支持数据库

- MySQL，Oracle，DB2，H2，HSQL，SQLite，PostgreSQL，SQLServer，Phoenix，Gauss，ClickHouse，Sybase，OceanBase，Firebird，Cubrid，Goldilocks，csiidb
- 达梦数据库，虚谷数据库，人大金仓数据库，南大通用(华库)数据库，南大通用数据库，神通数据库，瀚高数据库

二、快速配置

（一）前置技术

> Spring、MyBatis、Maven、SpringBoot

（二）基础配置

数据库创建

```sql
create table tb_student(
    no integer primary key not null auto_increment,
    name varchar(200),
    sex char(1),
    tel varchar(11),
    address varchar(200),
    class varchar(20),
    del_flg char(1) default 1
);


insert into tb_student(name, sex, tel, address, class, del_flg)
values
('Tom', '1', '18203908190', '河南郑州', 'Java1', '1'),
('Jack', '1', '18203908191', '河南郑州', 'Java1',  '1'),
('Peter', '1', '18203908192', '河南郑州', 'Java1', '1'),
('Lily', '0', '18203908193', '河南郑州', 'Java1', '1'),
('Marry', '0', '18203908194', '河南郑州', 'Java1', '1');
commit;
```

实体类

```java
public class Student {
    private String no;
    private String name;
    private String sex;
    private String tel;
    private String address;
    private String clazz;
    private String delFlg;

    // 省略set/get方法
}
```

Maven依赖

```
<!--Junit测试-->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
<!--Spring相关依赖-->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.2.2.RELEASE</version>
</dependency>
<!--SpringJDBC依赖-->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.2.2.RELEASE</version>
</dependency>
<!--Spring测试依赖-->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>5.2.2.RELEASE</version>
</dependency>
<!--阿里数据库连接池-->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.1.23</version>
</dependency>
<!--MySQL数据库驱动-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.21</version>
</dependency>
<!--日志依赖-->
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

MyBatis配置

```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>
    <settings>
        <setting name="logImpl" value="log4j"/>
    </settings>
</configuration>
```

Log4j配置

```properties
# 日志的级别,日志目的地
# 本文中设定的是全局的DEGUB，为的是开发过程中方便查看调试信息
log4j.rootLogger=DEBUG,console

# 配置目的地
log4j.appender.console=org.apache.log4j.ConsoleAppender
# 日志打印的类别，err：错误信息，字体颜色为红色；out：打印信息，
log4j.appender.console.target=System.err

# 配置控制台信息的布局
log4j.appender.console.layout=org.apache.log4j.PatternLayout
#%p：日志级别
#%c：类名
#%m：信息
#%n：换行
#%d：日期{yyyy-MM-dd hh:mm:ss}
log4j.appender.console.layout.ConversionPattern=[%p] %d{yyyy-MM-dd hh:mm:ss} %c %m%n

# 单独针对某个包设置打印级别
log4j.logger.com.soft.dao=DEBUG
```

数据库配置

```properties
mysql.driver=com.mysql.cj.jdbc.Driver
mysql.url=jdbc:mysql://localhost:3306/peter?serverTimezone=GMT
mysql.username=root
mysql.password=root
```

Spring配置

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        https://www.springframework.org/schema/mvc/spring-mvc.xsd">
```

```
    <!--数据源配置-->
    <context:property-placeholder location="classpath*:jdbc.properties"/>
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="driverClassName" value="${mysql.driver}"/>
        <property name="url" value="${mysql.url}"/>
        <property name="username" value="${mysql.username}"/>
        <property name="password" value="${mysql.password}"/>
    </bean>

    <!--MyBatis配置-->
    <bean class="org.mybatis.spring.SqlSessionFactoryBean">
        <!--数据源注入-->
        <property name="dataSource" ref="dataSource"/>
        <!--加载MyBatis配置文件-->
        <property name="configLocation" value="mybatis-config.xml"/>
        <!--映射文件加载-->
        <property name="mapperLocations"
value="classpath*:com/soft/dao/*.xml"/>
    </bean>

    <!--接口扫描-->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.soft.dao"/>
    </bean>
</beans>
```

（三）集成MyBatis-Plus

> 注意：提前配置好实体类和数据库的对应关系。例如：使用 @TableName 关联数据库表名。@TableId 配置 主键。

1、Spring 集成 MP

导入依赖：

```
<!--
    MyBatis Plus依赖：会自动依赖MyBatis和MyBatis-Spring依赖
    引入 MyBatis-Plus 之后请不要再次引入 MyBatis 以及 MyBatis-Spring，以避免因版本差异导致的问题。
-->
<!-- SSM -->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus</artifactId>
    <version>3.5.1</version>
</dependency>
```

修改Spring配置：

> MP集成非常简单，只需要把MyBatis和Spring集成时使用的对象`org.mybatis.spring.SqlSessionFactoryBean`切换为MP中集成的对象即可

```
<!--
```

```
    MyBatis-spring：org.mybatis.spring.SqlSessionFactoryBean
    MyBatisPlus：
com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean
-->
<bean
class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBea
n">
    <!--数据源注入-->
    <property name="dataSource" ref="dataSource"/>
    <!--加载MyBatis配置文件-->
    <property name="configLocation" value="mybatis-config.xml"/>
    <!--映射文件加载-->
    <property name="mapperLocations" value="classpath*:com/soft/dao/*.xml"/>
</bean>

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage"
value="com.baomidou.mybatisplus.samples.quickstart.mapper"/>
</bean>
```

测试：

```
@RunWith(SpringJUnit4ClassRunner.class)
// classpath不能少，标识的是编译后的文件路径
@ContextConfiguration({"classpath:spring-jdbc.xml"})
public class SpringMpTests {

    @Autowired
    StudentMapper studentMapper;

    @Test
    void selectById() {
        // 根据主键查询
        System.out.println(studentMapper.selectById(38));
    }
}
```

2、SpringBoot 集成 MP

导入依赖

```
<!-- SpringBoot -->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.5.1</version>
</dependency>
```

添加MyBatis-Plus扫描

```java
@SpringBootApplication
@MapperScan("com.soft.mapper")
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

测试

> 测试类中导入持久层接口，调用方法

```java
@SpringBootTest
public class SpringBootMpApplicationTests {

    @Autowired
    StudentMapper studentMapper;

    @Test
    void selectById() {
        // 根据主键查询
        System.out.println(studentMapper.selectById(38));
    }
}

```

> MyBatis： 1、创建实体类 2、创建Dao接口 3、创建Dao接口映射文件，编写SQL
>
> MyBatisPlus： 1、创建实体类 2、创建Dao接口，继承BaseMapper
>
> 结论：MyBatisPlus 只需要简单配置就可以实现基本增删改查操作，甚至不需要映射文件

三、配置说明

注解说明

@TableName

描述：表名注解

属性	类型	必须指定	默认值	描述
value	String	否	""	表名
schema	String	否	""	schema
keepGlobalPrefix	boolean	否	false	是否保持使用全局的 tablePrefix 的值(如果设置了全局 tablePrefix 且自行设置了 value 的值)
resultMap	String	否	""	xml 中 resultMap 的 id

属性	类型	必须指定	默认值	描述
autoResultMap	boolean	否	false	是否自动构建 resultMap 并使用(如果设置 resultMap 则不会进行 resultMap 的自动构建并注入)

@TableId

描述：主键注解

属性	类型	必须指定	默认值	描述
value	String	否	""	主键字段名
type	Enum	否	IdType.NONE	主键类型

IdType

值	描述
AUTO	数据库ID自增
NONE	无状态,该类型为未设置主键类型(注解里等于跟随全局,全局里约等于 INPUT)
INPUT	insert前自行set主键值
ASSIGN_ID	分配ID(主键类型为Number(Long和Integer)或String)(since 3.3.0),使用接口`IdentifierGenerator`的方法`nextId`(默认实现类为`DefaultIdentifierGenerator`雪花算法)
ASSIGN_UUID	分配UUID,主键类型为String(since 3.3.0),使用接口`IdentifierGenerator`的方法`nextUUID`(默认default方法)
~~ID_WORKER~~	分布式全局唯一ID 长整型类型(please use `ASSIGN_ID`)
~~UUID~~	32位UUID字符串(please use `ASSIGN_UUID`)
~~ID_WORKER_STR~~	分布式全局唯一ID 字符串类型(please use `ASSIGN_ID`)

@TableField

描述：字段注解(非主键)

属性	类型	必须指定	默认值	描述
value	String	否	""	数据库字段名
el	String	否	""	映射为原生`#{ ... }`逻辑,相当于写在xml 里的`#{ ... }`部分
exist	boolean	否	true	是否为数据库表字段

属性	类型	必须指定	默认值	描述
condition	String	否	""	字段 `where` 实体查询比较条件,有值设置则按设置的值为准,没有则为默认全局的 `%s=#{%s}`
update	String	否	""	字段 `update set` 部分注入, 例如：update="%s+1"：表示更新时会set version=version+1(该属性优先级高于 `el` 属性)
insertStrategy	Enum	N	DEFAULT	举例：NOT_NULL: `insert into table_a(<if test="columnProperty != null">column</if>) values (<if test="columnProperty != null">#{columnProperty}</if>)`
updateStrategy	Enum	N	DEFAULT	举例：IGNORED: `update table_a set column=#{columnProperty}`
whereStrategy	Enum	N	DEFAULT	举例：NOT_EMPTY: `where <if test="columnProperty != null and columnProperty!=''">column=#{columnProperty}</if>`
fill	Enum	否	FieldFill.DEFAULT	字段自动填充策略
select	boolean	否	true	是否进行 select 查询
keepGlobalFormat	boolean	否	false	是否保持使用全局的 format 进行处理
jdbcType	JdbcType	否	JdbcType.UNDEFINED	JDBC类型 (该默认值不代表会按照该值生效)
typeHandler	Class<? extends TypeHandler>	否	UnknownTypeHandler.class	类型处理器 (该默认值不代表会按照该值生效)
numericScale	String	否	""	指定小数点后保留的位数

FieldStrategy

值	描述
IGNORED	忽略判断
NOT_NULL	非NULL判断
NOT_EMPTY	非空判断(只对字符串类型字段,其他类型字段依然为非NULL判断)
DEFAULT	追随全局配置

FieldFill

值	描述
DEFAULT	默认不处理
INSERT	插入时填充字段

值	描述
UPDATE	更新时填充字段
INSERT_UPDATE	插入和更新时填充字段

@Version

描述：乐观锁注解、标记 `@Verison` 在字段上

看到如下错误，配置MP 的拦截器

```
Caused by: org.apache.ibatis.binding.BindingException: Parameter 'MP_OPTLOCK_VERSION_ORIGINAL' not found. Available parameters are [param1, et]
    at org.apache.ibatis.binding.MapperMethod$ParamMap.get(MapperMethod.java:212)
    at org.apache.ibatis.reflection.wrapper.MapWrapper.get(MapWrapper.java:45)
    at org.apache.ibatis.reflection.MetaObject.getValue(MetaObject.java:122)
    at com.baomidou.mybatisplus.core.MybatisParameterHandler.setParameters(MybatisParameterHandler.java:224)
    at org.apache.ibatis.executor.statement.PreparedStatementHandler.parameterize(PreparedStatementHandler.java:94)
    at org.apache.ibatis.executor.statement.RoutingStatementHandler.parameterize(RoutingStatementHandler.java:64)
    at org.apache.ibatis.executor.SimpleExecutor.prepareStatement(SimpleExecutor.java:88)
    at org.apache.ibatis.executor.SimpleExecutor.doUpdate(SimpleExecutor.java:49)
    at org.apache.ibatis.executor.BaseExecutor.update(BaseExecutor.java:117)
    at org.apache.ibatis.executor.CachingExecutor.update(CachingExecutor.java:76)
    at org.apache.ibatis.session.defaults.DefaultSqlSession.update(DefaultSqlSession.java:194)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.mybatis.spring.SqlSessionTemplate$SqlSessionInterceptor.invoke(SqlSessionTemplate.java:427)
    ... 76 more
```

```
package com.soft.config;
```

```java

import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
import
com.baomidou.mybatisplus.extension.plugins.inner.OptimisticLockerInnerInterceptor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * MyBatis 配置
 * 乐观锁配置
 */
@Configuration
public class MyBatisPlusConfiguration {
    @Bean
    public MybatisPlusInterceptor myOptimisticLockerInnerInterceptor(){
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
        // 3.4.0 之前版本
        // interceptor.addInnerInterceptor(new
OptimisticLockerInterceptor());
        // 3.4.0 之后版本
        interceptor.addInnerInterceptor(new
OptimisticLockerInnerInterceptor());
        return interceptor;
    }
}
```

> 乐观锁更新失败时不会报错。更新结果是1的时候表示成功。更新结果为0的时候表示失败。

@TableLogic

描述：表字段逻辑处理注解（逻辑删除）

属性	类型	必须指定	默认值	描述
value	String	否	""	逻辑未删除值
delval	String	否	""	逻辑删除值

@SqlParser

描述：租户注解,支持method上以及mapper接口上

属性	类型	必须指定	默认值	描述
filter	boolean	否	false	true: 表示过滤SQL解析，即不会进入ISqlParser解析链，否则会进解析链并追加例如tenant_id等条件

@KeySequence

描述：序列主键策略 `oracle` 属性：value、resultMap

属性	类型	必须指定	默认值	描述
value	String	否	""	序列名
clazz	Class	否	Long.class	id的类型, 可以指定String.class，这样返回的Sequence值是字符串"1"

@EnumValue

描述：通枚举类注解(注解在枚举字段上)

常用方法

Service CRUD

> 说明:
>
> - 通用 Service CRUD 封装**IService**接口，进一步封装 CRUD， 采用 `get 查询单行` `remove 删除`，`list 查询集合` `page 分页` 前缀命名方式区分 `Mapper` 层避免混淆，
> - 泛型 `T` 为任意实体对象
> - 建议如果存在自定义通用 Service 方法的可能，请创建自己的 `IBaseService` 继承 `Mybatis-Plus` 提供的基类
> - 对象 `Wrapper` 为 条件构造器
>
> 使用:
>
> 1. 业务层接口集成 `IService` 接口，方便接口注入的时候也能调用 MP 的方法
> 2. 实现类集成 `ServiceImpl` 类，ServiceImpl 帮助实现了 IService 的方法

添加

```
// 插入一条记录（选择字段，策略插入）
boolean save(T entity);
// 插入（批量）
boolean saveBatch(Collection<T> entityList);
// 插入（批量）
boolean saveBatch(Collection<T> entityList, int batchSize);
```

类型	参数名	描述
T	entity	实体对象
`Collection<T>`	entityList	实体对象集合
int	batchSize	插入批次数量

删除

```
// 根据 entity 条件，删除记录
boolean remove(Wrapper<T> queryWrapper);
// 根据 ID 删除
boolean removeById(Serializable id);
// 根据 columnMap 条件，删除记录
boolean removeByMap(Map<String, Object> columnMap);
// 删除（根据ID 批量删除）
boolean removeByIds(Collection<? extends Serializable> idList);
```

类型	参数名	描述
`Wrapper<T>`	queryWrapper	实体包装类 QueryWrapper
Serializable	id	主键ID
`Map<String, Object>`	columnMap	表字段 map 对象
`Collection<? extends Serializable>`	idList	主键ID列表

修改

```
// 根据 UpdateWrapper 条件，更新记录 需要设置sqlset
boolean update(Wrapper<T> updateWrapper);
// 根据 whereEntity 条件，更新记录
boolean update(T entity, Wrapper<T> updateWrapper);
// 根据 ID 选择修改
boolean updateById(T entity);
// 根据ID 批量更新
boolean updateBatchById(Collection<T> entityList);
// 根据ID 批量更新
boolean updateBatchById(Collection<T> entityList, int batchSize);
```

类型	参数名	描述
`Wrapper<T>`	updateWrapper	实体对象封装操作类 UpdateWrapper
T	entity	实体对象
`Collection<T>`	entityList	实体对象集合
int	batchSize	更新批次数量

查询

Get：单条记录查询

```java
// 根据 ID 查询
T getById(Serializable id);
// 根据 Wrapper，查询一条记录。结果集，如果是多个会抛出异常，随机取一条加上限制条件 wrapper.last("LIMIT 1")
T getOne(Wrapper<T> queryWrapper);
// 根据 Wrapper，查询一条记录
T getOne(Wrapper<T> queryWrapper, boolean throwEx);
// 根据 Wrapper，查询一条记录
Map<String, Object> getMap(Wrapper<T> queryWrapper);
// 根据 Wrapper，查询一条记录
<V> V getObj(Wrapper<T> queryWrapper, Function<? super Object, V> mapper);
```

类型	参数名	描述
Serializable	id	主键ID
Wrapper	queryWrapper	实体对象封装操作类 QueryWrapper
boolean	throwEx	有多个 result 是否抛出异常
T	entity	实体对象
Function<? super Object, V>	mapper	转换函数

List：多条记录查询

```java
// 查询所有
List<T> list();
// 查询列表
List<T> list(Wrapper<T> queryWrapper);
// 查询（根据ID 批量查询）
Collection<T> listByIds(Collection<? extends Serializable> idList);
// 查询（根据 columnMap 条件）
Collection<T> listByMap(Map<String, Object> columnMap);
// 查询所有列表
List<Map<String, Object>> listMaps();
// 查询列表
List<Map<String, Object>> listMaps(Wrapper<T> queryWrapper);
// 查询全部记录
List<Object> listObjs();
// 查询全部记录
<V> List<V> listObjs(Function<? super Object, V> mapper);
// 根据 Wrapper 条件，查询全部记录
List<Object> listObjs(Wrapper<T> queryWrapper);
// 根据 Wrapper 条件，查询全部记录
<V> List<V> listObjs(Wrapper<T> queryWrapper, Function<? super Object, V> mapper);
```

类型	参数名	描述
Wrapper	queryWrapper	实体对象封装操作类 QueryWrapper
Collection<? extends Serializable>	idList	主键ID列表

类型	参数名	描述
Map<?String, Object>	columnMap	表字段 map 对象
Function<? super Object, V>	mapper	转换函数

Page：分页查询

```
// 无条件分页查询
IPage<T> page(IPage<T> page);
// 条件分页查询
IPage<T> page(IPage<T> page, Wrapper<T> queryWrapper);
// 无条件分页查询
IPage<Map<String, Object>> pageMaps(IPage<T> page);
// 条件分页查询
IPage<Map<String, Object>> pageMaps(IPage<T> page, Wrapper<T> queryWrapper);
```

类型	参数名	描述
IPage	page	翻页对象
Wrapper	queryWrapper	实体对象封装操作类 QueryWrapper

Count：查询记录数

```
// 查询总记录数
int count();
// 根据 Wrapper 条件，查询总记录数
int count(Wrapper<T> queryWrapper);
```

Chain：链式操作

```
// 链式查询 普通
QueryChainWrapper<T> query();
// 链式查询 lambda 式。注意：不支持 Kotlin
LambdaQueryChainWrapper<T> lambdaQuery();

// 示例：
query().eq("column", value).one();
lambdaQuery().eq(Entity::getId, value).list();
```

```
// 链式更改 普通
UpdateChainWrapper<T> update();
// 链式更改 lambda 式。注意：不支持 Kotlin
LambdaUpdateChainWrapper<T> lambdaUpdate();

// 示例：
update().eq("column", value).remove();
lambdaUpdate().eq(Entity::getId, value).update(entity);
```

Mapper CRUD

> 说明:
>
> - 通用 CRUD 封装BaseMapper接口，为 `Mybatis-Plus` 启动时自动解析实体表关系映射转换为 `Mybatis` 内部对象注入容器
> - 泛型 `T` 为任意实体对象
> - 参数 `Serializable` 为任意类型主键 `Mybatis-Plus` 不推荐使用复合主键约定每一张表都有自己的唯一 `id` 主键
> - 对象 `Wrapper` 为 条件构造器
>
> 使用:
>
> - 持久层接口继承`BaseMapper`接口

添加

```
// 插入一条记录
int insert(T entity);
```

类型	参数名	描述
T	entity	实体对象

删除

```
// 根据 entity 条件，删除记录
int delete(@Param(Constants.WRAPPER) Wrapper<T> wrapper);
// 根据 columnMap 条件，删除记录
int deleteByMap(@Param(Constants.COLUMN_MAP) Map<String, Object> columnMap);

// 在实体类中配置id（@TableId）
// 删除（根据ID 批量删除）
int deleteBatchIds(@Param(Constants.COLLECTION) Collection<? extends Serializable> idList);
// 根据 ID 删除
int deleteById(Serializable id);
```

类型	参数名	描述
Wrapper	wrapper	实体对象封装操作类（可以为 null）
Collection<? extends Serializable>	idList	主键ID列表(不能为 null 以及 empty)
Serializable	id	主键ID
Map<String, Object>	columnMap	表字段 map 对象

修改

```java
// 根据 whereEntity 条件，更新记录
int update(@Param(Constants.ENTITY) T entity, @Param(Constants.WRAPPER) Wrapper<T> updateWrapper);
// 根据 ID 修改
int updateById(@Param(Constants.ENTITY) T entity);
```

类型	参数名	描述
T	entity	实体对象 (set 条件值,可为 null)
Wrapper	updateWrapper	实体对象封装操作类（可以为 null,里面的 entity 用于生成 where 语句）

> 删除和修改的入参Wrapper推荐使用UpdateWrapper<>

查询

```java
// 根据 ID 查询
T selectById(Serializable id);
// 根据 entity 条件，查询一条记录
T selectOne(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);

// 查询（根据ID 批量查询）
List<T> selectBatchIds(@Param(Constants.COLLECTION) Collection<? extends Serializable> idList);
// 根据 entity 条件，查询全部记录
List<T> selectList(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
// 查询（根据 columnMap 条件）
List<T> selectByMap(@Param(Constants.COLUMN_MAP) Map<String, Object> columnMap);
// 根据 Wrapper 条件，查询全部记录
List<Map<String, Object>> selectMaps(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
// 根据 Wrapper 条件，查询全部记录。注意： 只返回第一个字段的值
List<Object> selectObjs(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);

// 根据 entity 条件，查询全部记录（并翻页），需要额外配置分页插件
IPage<T> selectPage(IPage<T> page, @Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
// 根据 Wrapper 条件，查询全部记录（并翻页），需要额外配置分页插件
IPage<Map<String, Object>> selectMapsPage(IPage<T> page, @Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
// 根据 Wrapper 条件，查询总记录数
Integer selectCount(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

类型	参数名	描述
Serializable	id	主键ID
Wrapper	queryWrapper	实体对象封装操作类（可以为 null）
Collection<? extends Serializable>	idList	主键ID列表(不能为 null 以及 empty)

类型	参数名	描述
Map<String, Object>	columnMap	表字段 map 对象
IPage	page	分页查询条件（可以为 RowBounds.DEFAULT

QueryWrapper 常用方法

```
// 精确匹配：字段等于值
queryWrapper.eq(String column, Object value);
// 模糊匹配询：字段包含值
queryWrapper.like(String column, Object value);
// 模糊匹配：字段以值结尾
queryWrapper.likeLeft(String column, Object value);
// 模糊匹配：字段以值开头
queryWrapper.likeRight(String column, Object value);
// 范围匹配
queryWrapper.in(String column, Object ... vlues);
queryWrapper.in(String column, Collection<?> coll);
// 大于
queryWrapper.gt(String column, Object value);
// 小于
queryWrapper.lt(String column, Object value);
// 范围匹配
queryWrapper.between(String column, Object value1,  Object value2);
```

分页插件

springBoot 配置

```
@Configuration
public class MyBatisPlusConfiguration {
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor() {
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
        // MySQL 分页
        interceptor.addInnerInterceptor(new
PaginationInnerInterceptor(DbType.MYSQL));
        return interceptor;
    }

}
```

spring配置

```
<!-- 整合 MyBatis -->
<bean id="sqlSessionFactoryBean"
class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBea
n">
    <!-- 引入数据源 -->
    <property name="dataSource" ref="dataSource"/>
    <!-- 管理映射文件，接口和映射文件在同一个包下时，这里可以省略 -->
```

```
    <property name="mapperLocations"
value="classpath:com/soft/mapper/*.xml"/>
    <!-- 配置其他的参数 -->
    <!-- 加载原有的 MyBatis 配置文件 -->
    <property name="configLocation" value="classpath:mybatis-config.xml"/>
    <!-- 配置插件 -->
    <property name="plugins">
        <array>
            <ref bean="mybatisPlusInterceptor"/>
        </array>
    </property>
</bean>

<!-- MP 插件 -->
<bean id="mybatisPlusInterceptor"
class="com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor">
    <property name="interceptors">
        <list>
            <!-- 配置分页插件 -->
            <bean
class="com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInter
ceptor">
                <property name="dbType" value="MYSQL"/>
            </bean>
        </list>
    </property>
</bean>
```

持久层接口

```
public interface MPMapper extends BaseMapper<Student> {
}
```

业务层接口和实现类

```
public interface MPService {
    Page page(Page page, Wrapper wrapper);
}
```

```
@Service
public class MPServiceImpl implements MPService {

    @Resource
    MPMapper mpMapper;

    @Override
    public Page page(Page page, Wrapper wrapper) {
        return mpMapper.selectPage(page, wrapper);
    }
}
```

控制层

```java
package com.soft.controller;

import com.baomidou.mybatisplus.core.conditions.Wrapper;
import com.baomidou.mybatisplus.core.conditions.query.Query;
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.baomidou.mybatisplus.core.conditions.update.UpdateWrapper;
import com.baomidou.mybatisplus.extension.incrementer.OracleKeyGenerator;
import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
import com.soft.entity.Student;
import com.soft.service.MPService;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import javax.annotation.Resource;
import java.util.List;

@Controller
@RequestMapping("/mp")
public class MPController {
    @Resource
    MPService mpService;

    @RequestMapping("/page")
    @ResponseBody
    public Page<Student> page(){
        // 方法1
        Page<Student> page = Page.of(2, 5);
        // 方法2
        Page page = new Page(1, 5);

        QueryWrapper<Student> wrapper = new QueryWrapper<>();
        wrapper.orderByAsc("no");

        Page result = mpService.page(page, wrapper);

        return result;
    }
}
```

Sequence主键

spring配置

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd
```

```
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd
        http://www.springframework.org/schema/mvc
        https://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/tx
        https://www.springframework.org/schema/tx/spring-tx.xsd">


    <bean id="sqlSessionFactory"
class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBea
n">
        <!--其他配置省略-->
        <!--引入全局配置-->
        <property name="globalConfig" ref="globalConfig"/>
    </bean>

    <!--全局的DB配置-->
    <bean id="globalConfig"
class="com.baomidou.mybatisplus.core.config.GlobalConfig">
        <property name="dbConfig" ref="dbConfig"/>
    </bean>
    <bean id="dbConfig"
class="com.baomidou.mybatisplus.core.config.GlobalConfig.DbConfig">
        <property name="keyGenerator" ref="keyGenerator"/>
    </bean>
    <!--主键生成策略，使用的是Oracle的-->
    <bean id="keyGenerator"
class="com.baomidou.mybatisplus.extension.incrementer.OracleKeyGenerator"/>
</beans>
```

实体类配置

```
package com.soft.entity;

import com.baomidou.mybatisplus.annotation.*;
import org.apache.ibatis.type.JdbcType;

@TableName(value="tb_student")
// 配置主键的序列名称，value的值是序列的名称
@KeySequence("SEQ_STUDENT")
public class Student extends BaseEntity {
    // 配置主键
    @TableId(value="no", type=IdType.INPUT)
    private String no;
    private String name;

    // 省略set/get方法
}
```

逻辑删除

```
<!-- 整合 MyBatis -->
<!--<bean id="sqlSessionFactoryBean"
class="org.mybatis.spring.SqlSessionFactoryBean">-->
<bean id="sqlSessionFactoryBean"
class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean">
    <!-- 全局配置 -->
    <property name="globalConfig">
        <!-- 全局配置，配置逻辑删除 -->
        <bean class="com.baomidou.mybatisplus.core.config.GlobalConfig">
            <property name="dbConfig" ref="dbConfig"/>
        </bean>
    </property>
</bean>

<!-- 创建 dbConfig 对象 -->
<bean id="dbConfig"
class="com.baomidou.mybatisplus.core.config.GlobalConfig$DbConfig">
    <!-- 逻辑删除的字段 -->
    <property name="logicDeleteField" value="delFlg"/>
    <!-- 删除标识 -->
    <property name="logicDeleteValue" value="0"/>
    <!-- 不删除标识 -->
    <property name="logicNotDeleteValue" value="1"/>
</bean>
```