Mybatis-Plus

1.快速入门

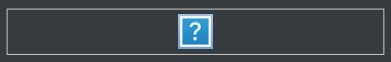
简介

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

特性

1. **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
2. **损耗小：**启动时会自动注入基本CRUD，性能基本无损耗，直接面向对象操作
3. **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
4. **支持Lambda形式调用：**通过Lambda表达式，方便的编写各类查询条件，无需担心字段写错
5. **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
6. **支持ActiveRecord模式：**支持ActiveRecord形式调用，实体类只需要继承Model类即可进行强大的CRUD操作
7. **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)
8. **内置代码生成器：**采用代码或者Maven插件可快速生成Mapper、Controller层代码，支持模板引擎，更有超多自定义配置
9. **内置分业插件：**给予MySql、MariaDB、Oracle、DB2、H2、HSQL、SQLite、Postgre、SQLServer等多种数据库
10. **内置性能分析插件：**可输出SQL语句及其执行时间，建议开发测试时启用该功能，能快速找出满查询
11. **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

框架结构



代码托管

Gitee

Github

快速开始

创建一张User表，结构如下

id	name	age	email
1	Jone	18	test1@baomidou.com
2	Jack	20	test2@baomidou.com
3	Tom	28	test3@baomidou.com
4	Sandy	21	test4@baomidou.com
5	Billie	24	test5@baomidou.com

对应的sql脚本如下：

```
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
  `id` bigint NOT NULL COMMENT '主键ID',
  `name` varchar(30) DEFAULT NULL COMMENT '姓名',
  `age` int DEFAULT NULL COMMENT '年龄',
  `email` varchar(50) DEFAULT NULL COMMENT '邮箱',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;

BEGIN;
```

```
INSERT INTO `user` (`id`, `name`, `age`, `email`) VALUES (1, 'Jone', 18,
'test1@baomidou.com');
INSERT INTO `user` (`id`, `name`, `age`, `email`) VALUES (2, 'Jack', 20,
'test2@baomidou.com');
INSERT INTO `user` (`id`, `name`, `age`, `email`) VALUES (3, 'Tom', 28,
'test3@baomidou.com');
INSERT INTO `user` (`id`, `name`, `age`, `email`) VALUES (4, 'Sandy',
21, 'test4@baomidou.com');
INSERT INTO `user` (`id`, `name`, `age`, `email`) VALUES (5, 'Billie',
24, 'test5@baomidou.com');
COMMIT;
```

初始化工作

使用IDEA的 Spring Initializer 快速初始化一个 Spring Boot 工程

添加依赖

```
        <!--lombok 简化实体类-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <!--mybatis-plus 依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.5.1</version>
        </dependency>
        <!--mybatis 依赖-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
```

数据库配置

```
# mysql数据库连接。spring boot2.1及以上，内置jdbc8驱动
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/mybatis-plus?
useUnicode=true&characterEncoding=utf8&useSSL=false&allowPublicKeyRetrie
val=true&serverTimezone=GMT%2B8
    username: root
    password: root1234

# mybatis日志,在控制台查看mysql语句输出
mybatis-plus:
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
```

编码测试

1. 主启动类

```
package com.maxt.mybatisplus;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;

/**
 * @Author Maxt
 * @Date 2022/3/19 下午7:31
```

```
 * @Version 1.0
 * @Description 用户类
 */

@SpringBootApplication
//添加mapper注解，扫描Mapper文件夹
@MapperScan("com.maxt.mybatisplus.mapper")
public class MybatisPlusApplication {

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

}
```

2. 实体类

```
package com.maxt.mybatisplus.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * @Author Maxt
 * @Date 2022/3/19 下午7:31
 * @Version 1.0
 * @Description 用户类
 */
//set、get方法
@Data
//所有参数构造函数
@AllArgsConstructor
//无参构造函数
@NoArgsConstructor
```

```
public class User {
    private Long id;
    private String name;
    private Integer age;
    private String email;
}
```

3. Mapper类

```
package com.maxt.mybatisplus.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.maxt.mybatisplus.entity.User;
import org.springframework.stereotype.Repository;

/**
 * @Author Maxt
 * @Date 2022/3/19 下午7:41
 * @Version 1.0
 * @Description
 */
@Repository
public interface UserMapper extends BaseMapper<User> {
}
```

4. 测试类

```
package com.maxt.mybatisplus.test;

import com.maxt.mybatisplus.entity.User;
import com.maxt.mybatisplus.mapper.UserMapper;
import org.assertj.core.api.Assert;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
```

```java
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.context.annotation.ComponentScan;

import java.util.List;

/**
 * @Author Maxt
 * @Date 2022/3/19 下午7:43
 * @Version 1.0
 * @Description  用户测试类
 */
@SpringBootTest
public class UserTest {
    @Autowired
    UserMapper userMapper;

    /**
     * 查询所有
     */
    @Test
    public void testSelectAll(){
        System.out.println("selectAll method test");
        List<User> users = userMapper.selectList(null);
        users.forEach(System.out::println);
    }

    /**
     * 插入操作
     * 插入时根据主键策略来判断时候需要插入主键值,
     * 默认为ASSIGN_ID(分布式ID, 根据雪花算法生成)
     */
    @Test
    public void testInsert(){
        User user = new User();
        user.setName("Lucy");
        user.setAge(20);
```

```
        user.setEmail("1234@163.com");
        //insert为插入数据影响的数据行数
        int insert = userMapper.insert(user);
        System.out.println(insert);
    }

    /**
     * 根据id更新操作
     */
    @Test
    public void testUpdate(){
        User user = new User();
        user.setId(1505230152404123650L);
        user.setName("LucyUpdate");
        user.setAge(20);
        user.setEmail("1234@163.com");
        int count = userMapper.updateById(user);
        System.out.println(count);
    }

    /**
     * 根据id删除操作
     */
    @Test
    public void testDelete(){
        User user = new User();
        user.setId(1505230152404123650L);
        int count = userMapper.deleteById(user);
        System.out.println(count);
    }
}
```

注解

@TableName

描述：表名注解，标识实体类对应的表

使用位置：实体类

```
package com.maxt.mybatisplus.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * @Author Maxt
 * @Date 2022/3/19 下午7:31
 * @Version 1.0
 * @Description 用户类
 */
//set、get方法
@Data
//所有参数构造函数
@AllArgsConstructor
//无参构造函数
@NoArgsConstructor
@TableName("user")
public class User {
    private Long id;
    private String name;
    private Integer age;
    private String email;
}
```

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(当全局tablePrefix生效时)
resultMap	String	否	""	xml中resultMap的id(用于满足特定类型的实体类对象绑定)
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的自动构建与注入)
excludeProperty	String[]	否	{}	需要排除的属性名@since3.3.1

关于**autoResultMap**的说明

MP会自动构建一个resultMap并注入到MyBatis里(一般用不上)，注意以下内容

因为MP底层是MyBatis，所以MP只是帮助注入了常用CRUD到MyBatis里，注入之前是动态的(根据Entity字段以及注解变化而变化)，注入之后是静态的(等于XML配置中的内容)

对于typeHandler属性，MyBatis只支持写在2个地方

1. 定义在resultMap里，作用于查询结果的封装
2. 定义在insert和update语句的#{property}中的property后面(例如：#{property,typeHandler=xxx.xxx.xxx})，并且只作用于当前设置值

除了以上两种直接指定typeHandler的形式，MyBatis有一个全局扫描自定义typeHandler包的配置，原理是根据property类型去找对应的typeHandler并使用

@TableId

描述：主键注解

使用位置：实体类主键字段

```
package com.maxt.mybatisplus.entity;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * @Author Maxt
 * @Date 2022/3/19 下午7:31
 * @Version 1.0
 * @Description 用户类
 */
//set、get方法
@Data
//所有参数构造函数
@AllArgsConstructor
//无参构造函数
@NoArgsConstructor
@TableName("user")
public class User {
    @TableId(type = IdType.ASSIGN_ID)
    private Long id;
    private String name;
    private Integer age;
    private String email;
}
```

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	指定主键类型

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)，使用借口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(推荐使用ASSIGN_ID)
~~UUID~~	32位UUID字符串(推荐使用ASSIGN_UUID)
~~ID_WORKER_STR~~	分布式全局唯一ID字符串类型(推荐使用ASSIGN_ID)

@TableField

描述：字段注解(非主键)

```
package com.maxt.mybatisplus.entity;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableField;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import lombok.AllArgsConstructor;
import lombok.Data;
```

```java
import lombok.NoArgsConstructor;

/**
 * @Author Maxt
 * @Date 2022/3/19 下午7:31
 * @Version 1.0
 * @Description 用户类
 */
//set、get方法
@Data
//所有参数构造函数
@AllArgsConstructor
//无参构造函数
@NoArgsConstructor
@TableName("user")
public class User {
    @TableId(type = IdType.ASSIGN_ID)
    private Long id;
    @TableField("name")
    private String name;
    private Integer age;
    private String email;
}
```

属性	类型	必须指定	默认值	描述
value	String	否	""	数据库字段名
exist	String	否	""	是否为数据库字段
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，当在version字段上注解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	否	FieldStrategy.DEFAULT	举例：NOT_NULL insert into table_a(column) values (#{columnProperty})
updateStrategy	Enum	否	FieldStrategy.DEFAULT	举例：IGNORED update table_a set column = #{columnProperty}
whereStrategy	Enum	否	FieldStrategy.DEFAULT	举例：NOT_EMPTY where column=#{columnProperty}
fill	Enum	否	FieldStrategy.DEFAULT	字段自动填充策略
select	boolean	否	true	是否进行select查询
keepGlobalFormat	boolean	否	false	是否使用全局的format进行处理
jdbcType	JdbcType	否	JdbcType.UNDEFINED	JDBC类型(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(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

s

```java
package com.baomidou.mybatisplus.annotation;

/**
 * SQL 比较条件常量定义类
 *
 * @author hubin
 * @since 2018-01-05
 */
public class SqlCondition {
    /**
     * 等于
     */
    public static final String EQUAL = "%s=#{%s}";
    /**
     * 不等于
     */
    public static final String NOT_EQUAL = "%s&lt;&gt;#{%s}";
    /**
     * % 两边 %
     */
    public static final String LIKE = "%s LIKE CONCAT('%%',#{%s},'%%')";

    /**
     * % 两边 % [oracle使用]
     */
    public static final String ORACLE_LIKE = "%s LIKE
CONCAT(CONCAT('%%',#{%s}),'%%')";
    /**
     * % 左
     */
    public static final String LIKE_LEFT = "%s LIKE CONCAT('%%',#{%s})";
    /**
     * 右 %
     */
```

```
    public static final String LIKE_RIGHT = "%s LIKE CONCAT(#
{%s},'%%')";
}
```

"jdbcType"、"typeHandler"、"numericScale"的说明

numericScale：只生效于update的sql

jdbcType和typeHandler如果不配合@TableName#autoResultMap = true一起使用，也只生效于update的sql

typeHandler：如果字段类型和set进去的类型为equals关系，则只需要让typeHandler让Mybatis加载到即可，不需要使用注解

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，其他字段类型依然为非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
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@Version在字段上

@EnumValue

描述：普通枚举类注解(注解在枚举字段上)

@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

@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"

@InterceptorIgnore

描述：作用与xxxMapper.java方法之上，各属性代表对应的插件，各属性不给值则默认为false，设置为true忽略拦截

属性	类型	默认值	描述
tenantLine	String	""	行级租户
dynamicTableName	String	""	动态表名
blockAttack	String	""	攻击SQL阻断解析器，防止全表更新与删除
illegalSql	String	""	垃圾SQL拦截

MybatisPlusInterceptor插件：核心插件，目前代理了Executor#query和Executor#update和StatementHandler#prepare方法

属性

private List interceptors = new ArrayList<>();

InnerInterceptor

提供的插件都将基于此接口来实现功能

目前已有的功能

1. 自动分页：PaginationInnerInterceptor
2. 多租户：TenantLineInnerInterceptor
3. 动态表名：DynamicTableNameInnerInterceptor
4. 乐观锁：OptimisticLockerInnerInterceptor
5. sql性能规范：IllegalSQLInnerInterceptor
6. 防止全表更新与删除：BlockAttackInnerInterceptor

> 注意：
>
> 使用多个功能需要注意顺序关系，建议使用如下顺序
>
> 1. 多租户，动态表名
> 2. 分页，乐观锁
> 3. sql性能规范，防止全表更新与删除

> 总结：对sql进行单次改造的优先放入，不对sql进行改造的最后放入

@OrderBy

描述：内置SQL默认指定排序，优先级低于wrapper

属性	类型	必须指定	默认值	描述
isDesc	boolean	否	""	是否倒序查询
sort	short	否	Short.MAX_VALUE	数字越小越靠前

2.核心功能

代码生成器(新)

> 注意：
>
> 使用版本：mybatis-plus-generator 3.5.1 及其以上版本，对历史不兼容

引入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.4</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.maxt</groupId>
    <artifactId>mybatis-plus</artifactId>
    <version>0.0.1-SNAPSHOT</version>
```

```xml
    <name>mybatis-plus</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <!--lombok 简化实体类-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <!--mybatis-plus 依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.5.1</version>
        </dependency>
        <!--mybatis-plus 测试依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter-test</artifactId>
            <version>3.5.1</version>
        </dependency>
        <!--代码生成器 3.5.1⬆-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-generator</artifactId>
            <version>3.5.1</version>
        </dependency>
        <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-freemarker</artifactId>
        </dependency>
        <!--mybatis 依赖-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

编码测试

```
package com.maxt.mybatisplus.test;

import com.baomidou.mybatisplus.annotation.FieldFill;
import com.baomidou.mybatisplus.generator.FastAutoGenerator;
import com.baomidou.mybatisplus.generator.config.OutputFile;
import com.baomidou.mybatisplus.generator.engine.BeetlTemplateEngine;
import
com.baomidou.mybatisplus.generator.engine.FreemarkerTemplateEngine;
import com.baomidou.mybatisplus.generator.fill.Column;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

/**
 * @Author Maxt
 * @Date 2022/3/20 下午3:23
 * @Version 1.0
 * @Description  代码快速生成与代码交互式生成测试
 */
@SpringBootTest
public class FastAutoGeneratorTest {

    /**
     * 快速生成
     */
    @Test
    public void testFastGenerator(){
        FastAutoGenerator.create("jdbc:mysql://localhost:3306/mybatis-
plus", "root", "root1234")
                .globalConfig(builder -> {
                            //设置作者
```

```
                    builder.author("Maxt")
                            //开启swagger模式
                            .enableSwagger()
                            //覆盖已生成文件
                            .fileOverride()
                            //指定输出目录
                            .outputDir("generator");
                })
                .packageConfig(builder -> {
                            //设置父包名

builder.parent("com.maxt.mybatisplus.samples.generator")
                             //设置父包模块名
                             .moduleName("system")
                             //设置mapperXml生成路径

.pathInfo(Collections.singletonMap(OutputFile.mapperXml, "generator"));
                })
                .strategyConfig(builder -> {
                            //设置需要生成对表名
                    builder.addInclude("user");
                            //设置过滤表前缀
                            //.addTablePrefix("t_", "c_");
                })
                //使用Freemarker引擎模板，默认的是Velocity引擎模板
                .templateEngine(new FreemarkerTemplateEngine())
                .execute();
    }

    /**
     * 交互式生成
     */
    @Test
    public void testScannerGenerator(){
        FastAutoGenerator.create("jdbc:mysql://localhost:3306/mybatis-
plus", "root", "root1234")
```

```
                .globalConfig((scanner, builder) ->
                        builder.author(scanner.apply("请输入作者名称? "))
                        .fileOverride()
                )
                .packageConfig((scanner, builder) ->
                        builder.parent(scanner.apply("请输入包名? "))
                )
                .strategyConfig((scanner, builder) ->
                        builder.addInclude(getTables(scanner.apply("请输
入表名，多个表用英文逗号分割，所有表输入all")))

.controllerBuilder().enableRestStyle().enableHyphenStyle()
                        .entityBuilder().enableLombok().addTableFills(
                                new Column("create_time",
FieldFill.INSERT)
                        ).build()
                )
                //模板引擎配置，默认Velocity 可选模板引擎Beetl或Freemarker
                //.templateEngine(new BeetlTemplateEngine())
                //.templateEngine(new FreemarkerTemplateEngine())
                .execute();
    }

    private List<String> getTables(String tables) {
        return "all".equals(tables) ? Collections.emptyList() :
Arrays.asList(tables.split(","));
    }
}
```

代码生成器(旧)

> 注意：
>
> 适用版本：mybatis-plus- generator 3.5.1 以下版本

AutoGenerator是MyBatis-Plus的代码生成器，通过AutoGenerator可以快速生成Entity、Mapper、Mapper XML、Service、Controller等各个模块等代码，极大提升了开发效率

引入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.4</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.maxt</groupId>
    <artifactId>mybatis-plus</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>mybatis-plus</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <!--lombok 简化实体类-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
```

```
        </dependency>
        <!--mybatis-plus 依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.5.0</version>
        </dependency>
        <!--mybatis-plus 测试依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter-test</artifactId>
            <version>3.5.0</version>
        </dependency>
        <!--代码生成器 3.5.0⬇️-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-generator</artifactId>
            <version>3.5.0</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-freemarker</artifactId>
        </dependency>
        <!--mybatis 依赖-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
```

```
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

编码测试

```
package com.maxt.mybatisplus.test;

import com.baomidou.mybatisplus.generator.AutoGenerator;
import com.baomidou.mybatisplus.generator.config.*;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

/**
 * @Author Maxt
 * @Date 2022/3/20 下午4:13
 * @Version 1.0
 * @Description
 */
@SpringBootTest
```

```
public class CodeGeneratorTest {

    @Test
    public void testCodeGenerator(){
        //代码生成器
        new AutoGenerator(
                new
DataSourceConfig.Builder("jdbc:mysql://localhost:3306/mybatis-plus",
"root", "root1234")
                        .build())
                //全局配置
        .global(new GlobalConfig.Builder()
                //作者
                .author("Maxt")
                //输出位置
                .outputDir("generator")
                //打开文件
                .openDir(false)
                //开启swagger模式
                .enableSwagger()
                .build())
        //包配置
        .packageInfo(new PackageConfig.Builder()
                //父模块名称
                .moduleName("system")
                //父包名
                .parent("com.maxt.mybatisplus")
                .build()
        )
        //自定义配置
        //.injection(new InjectionConfig())
        //模板引擎
        .template(new TemplateConfig.Builder()
                //指定自定义模板路径，会根据使用的模板引擎自动识别
                .entity("")
                .controller("")
```

```
                .service("", "")
                .mapper("")
                .build()
        )
        //策略配置
        .strategy(new StrategyConfig.Builder()
                //设置需要生成对表名
                .addInclude("")
                .addFieldPrefix("")
                //设置过滤表前缀
                .addTablePrefix("")
                .build()
        ).execute();
    }
}
```

CRUD接口

Service CRUD接口

1. 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
2. 泛型 T 为任意实体对象
3. 如果存在自定义通用Service方法，请创建自己的IBaseService继承Mybatis-plus提供的基类
4. 对象Wrapper为条件构造器

Save

```
    //插入一条就来（选择字段，策略插入）
    default boolean save(T entity) {
        return SqlHelper.retBool(this.getBaseMapper().insert(entity));
    }
    //插入（批量）
    @Transactional(
        rollbackFor = {Exception.class}
    )
    default boolean saveBatch(Collection<T> entityList) {
        return this.saveBatch(entityList, 1000);
    }
    //插入（批量） batchSize（插入批次数量）
    boolean saveBatch(Collection<T> entityList, int batchSize);
```

SaveOrUpdate

```
    //TableId注解存在更新记录，否则插入一条记录
    boolean saveOrUpdate(T entity);
    //批量修改插入
    @Transactional(
        rollbackFor = {Exception.class}
    )
    default boolean saveOrUpdateBatch(Collection<T> entityList) {
        return this.saveOrUpdateBatch(entityList, 1000);
    }
    //批量修改插入  batchSize 插入批次数量
    boolean saveOrUpdateBatch(Collection<T> entityList, int batchSize);
```

Remove

```
    //根据id删除
    default boolean removeById(Serializable id) {
        return SqlHelper.retBool(this.getBaseMapper().deleteById(id));
    }
```

```
    default boolean removeById(T entity) {
        return
SqlHelper.retBool(this.getBaseMapper().deleteById(entity));
    }
    //根据columnMap条件删除
    default boolean removeByMap(Map<String, Object> columnMap) {
        Assert.notEmpty(columnMap, "error: columnMap must not be empty",
new Object[0]);
        return
SqlHelper.retBool(this.getBaseMapper().deleteByMap(columnMap));
    }
    //根据entity条件删除
    default boolean remove(Wrapper<T> queryWrapper) {
        return
SqlHelper.retBool(this.getBaseMapper().delete(queryWrapper));
    }
    //根据id集合进行批量删除
    default boolean removeByIds(Collection<?> list) {
        return CollectionUtils.isEmpty(list) ? false :
SqlHelper.retBool(this.getBaseMapper().deleteBatchIds(list));
    }

    @Transactional(
        rollbackFor = {Exception.class}
    )
    default boolean removeByIds(Collection<?> list, boolean useFill) {
        if (CollectionUtils.isEmpty(list)) {
            return false;
        } else {
            return useFill ? this.removeBatchByIds(list, true) :
SqlHelper.retBool(this.getBaseMapper().deleteBatchIds(list));
        }
    }

    @Transactional(
        rollbackFor = {Exception.class}
```

```
    )
    default boolean removeBatchByIds(Collection<?> list) {
        return this.removeBatchByIds(list, 1000);
    }

    @Transactional(
        rollbackFor = {Exception.class}
    )
    default boolean removeBatchByIds(Collection<?> list, boolean
useFill) {
        return this.removeBatchByIds(list, 1000, useFill);
    }
```

Update

```
    //根据id选择修改
    default boolean updateById(T entity) {
        return
SqlHelper.retBool(this.getBaseMapper().updateById(entity));
    }
    //根据updateWrapper条件，更新记录，需要设置sqlset
    default boolean update(Wrapper<T> updateWrapper) {
        return this.update((Object)null, updateWrapper);
    }

    default boolean update(T entity, Wrapper<T> updateWrapper) {
        return SqlHelper.retBool(this.getBaseMapper().update(entity,
updateWrapper));
    }

    @Transactional(
        rollbackFor = {Exception.class}
    )
    default boolean updateBatchById(Collection<T> entityList) {
        return this.updateBatchById(entityList, 1000);
    }
```

```
    //根据id批量更新  batchSize 更新批次数量
    boolean updateBatchById(Collection<T> entityList, int batchSize);
```

Get

```
    //根据id查询
    default T getById(Serializable id) {
        return this.getBaseMapper().selectById(id);
    }
    //根据Wrapper，查询一条记录。结果集，如果是多个会抛出异常，可加上限制条件防止抛出异常 wrapper.last("LIMIT 1")
    default T getOne(Wrapper<T> queryWrapper) {
        return this.getOne(queryWrapper, true);
    }

    T getOne(Wrapper<T> queryWrapper, boolean throwEx);
    //根据wrapper查询一组记录
    Map<String, Object> getMap(Wrapper<T> queryWrapper);
    //根据wrapper查询一条记录  Function<? super Object, V> 转换函数
    <V> V getObj(Wrapper<T> queryWrapper, Function<? super Object, V> mapper);
```

List

```
// 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);
```

Page

```
// 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);
```

Count

```
// 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

query

```
// 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();
```

update

```
// 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接口

1. 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
2. 泛型 `T` 为任意实体对象
3. 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
4. 对象 Wrapper 为 条件构造器

Insert

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

Delete

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

Update

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

Select

```
// 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）
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）
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);
```

mapper 层选装件

> 选装件位于com.baomidou.mybatisplus.extension.injector.methods包下，需要配合Sql注入器使用
>
> 案例：https://gitee.com/baomidou/mybatis-plus-samples/tree/master/mybatis-plus-sample-sql-injector

AlwaysUpdateSomeColumnById

```
int alwaysUpdateSomeColumnById(T entity);
```

insertBatchSomeColumn

```
int insertBatchSomeColumn(List<T> entityList);
```

logicDeleteByIdWithFill

```
int logicDeleteByIdWithFill(T entity);
```

ActiveRecord模式

1. 实体类只需要继承Model类即可进行强大的CRUD操作
2. 需要项目中以注入对应实体的BaseMapper

操作步骤

```
//1.继承Model
class User extends Model<User>{
    // fields...
}
//2.调用CURD方法
User user = new User();
user.insert();
user.selectAll();
user.updateById();
user.deleteById();
```

SimpleQuery工具类

1. 对selectList查询后的结果用Stream流进行了一些封装，使其可以返回一些指定结果，简洁了api的调用
2. 需要项目中以注入对应实体的BaseMapper
3. 对于下方参数peeks，其类型为Consumer...，可一直往后叠加操作例如：List ids = SimpleQuery.list(Wrappers.lambdaQuery(), Entity::getId, System.out::println, user -> userNames.add(user.getName()));

keyMap

```
// 查询表内记录，封装返回为Map<属性,实体>  SFunction<E, A> 实体中属性的getter，用于封装后map中作为key的条件  Consumer<E>... 可叠加的后续操作
Map<A, E> keyMap(LambdaQueryWrapper<E> wrapper, SFunction<E, A> sFunction, Consumer<E>... peeks);
// 查询表内记录，封装返回为Map<属性,实体>，考虑了并行流的情况
Map<A, E> keyMap(LambdaQueryWrapper<E> wrapper, SFunction<E, A> sFunction, boolean isParallel, Consumer<E>... peeks);
```

map

```
// 查询表内记录，封装返回为Map<属性,属性>  SFunction<E, A> keyFunc 封装后map中作为key的条件 SFunction<E, P> 封装后map中作为value的条件
Map<A, P> map(LambdaQueryWrapper<E> wrapper, SFunction<E, A> keyFunc, SFunction<E, P> valueFunc, Consumer<E>... peeks);
// 查询表内记录，封装返回为Map<属性,属性>，考虑了并行流的情况
Map<A, P> map(LambdaQueryWrapper<E> wrapper, SFunction<E, A> keyFunc, SFunction<E, P> valueFunc, boolean isParallel, Consumer<E>... peeks);
```

group

```
// 查询表内记录，封装返回为Map<属性,List<实体>>
Map<K, List<T>> group(LambdaQueryWrapper<T> wrapper, SFunction<T, A> sFunction, Consumer<T>... peeks);
// 查询表内记录，封装返回为Map<属性,List<实体>>，考虑了并行流的情况
Map<K, List<T>> group(LambdaQueryWrapper<T> wrapper, SFunction<T, K> sFunction, boolean isParallel, Consumer<T>... peeks);
// 查询表内记录，封装返回为Map<属性,分组后对集合进行的下游收集器>
M group(LambdaQueryWrapper<T> wrapper, SFunction<T, K> sFunction, Collector<? super T, A, D> downstream, Consumer<T>... peeks);
// 查询表内记录，封装返回为Map<属性,分组后对集合进行的下游收集器>，考虑了并行流的情况
M group(LambdaQueryWrapper<T> wrapper, SFunction<T, K> sFunction, Collector<? super T, A, D> downstream, boolean isParallel, Consumer<T>... peeks);
```

类型	参数名	描述
T	entity	实体对象
K	attribute	实体属性类型，也是map中key的类型
D	-	下游收集器返回类型，也是map中value的类型
A	-	下游操作中间类型
M	-	最终结束返回的Map<K, D>
LambdaQueryWrapper	wrapper	支持lambda的条件构造器
SFunction<E, A>	sFunction	分组依据，封装后map中作为key的条件
Collector<T, D, A>	downstream	下游收集器
boolean	isParallel	为true时底层使用并行流执行
Consumer...	peeks	可叠加的后续操作

list

```
// 查询表内记录，封装返回为List<属性>
List<A> list(LambdaQueryWrapper<E> wrapper, SFunction<E, A> sFunction,
Consumer<E>... peeks);
// 查询表内记录，封装返回为List<属性>，考虑了并行流的情况
List<A> list(LambdaQueryWrapper<E> wrapper, SFunction<E, A> sFunction,
boolean isParallel, Consumer<E>... peeks);
```

条件构造器

> 说明：
>
> 1. 以下出现的第一个入参boolean condition表示该条件是否加入最后生成的sql中，例如：query.like(StringUtils.isNotBlank(name), Entity::getName, name).eq(age != null && age >= 0, Entity::getAge, age)
> 2. 以下代码块内的多个方法均为从上往下补全个别boolean类型的入参，默认为true
> 3. 以下出现的泛型Param均为Wrapper的子类实例(均具有AbstractWrapper的所有

方法)
4. 以下方法在入参中出现的R为泛型，在普通wrapper中是String，在LambdaWrapper中是函数(例如：Entity::getId，Entity为实体类，getId为字段id的getMethod)
5. 以下方法入参中 R column均表示数据库字段，当 R 具体类型为String时则为数据库字段名(字段名是数据库关键字的自己用转义符包裹)而不是实体类数据字段名！！！另当 R 具体类型为SFunction时项目runtime不支持eclipse自家的编译器
6. 以下举例均为使用普通wrapper，入参为Map和List的均以json形式表现。使用中如果入参的Map或者List为空，则不会加入最后生成的sql中

> 警告：
>
> 不支持以及不赞成在RPC调用中把Wrapper进行传输
>
> 1. Wrapper很重
> 2. 传输wrapper可以类比为你的controller用Map接受值(开发一时爽，维护火葬场)
> 3. 正确的RPC调用姿势是写一个DTO进行传输，被调用方再根据DTO执行相应的操作

AbstractWrapper

> 说明：
>
> QueryWrapper(LambdaQueryWrapper)和UpdateWrapper(LambdaUpdateWrapper)的父类用于生成sql的where条件，entity属性也用于生成sql的where条件
>
> 注意：entity生成的where条件与使用各个api生成的where条件没有任何关联关系

allEq

```
//params: key为数据库字段名，value为字段值
allEq(Map<R, V> params)
//null2IsNull 为true则map的value为null时调用isNull方法，为false时则忽略value
allEq(Map<R, V> params, boolean null2IsNull)
allEq(boolean condition, Map<R, V> params, boolean null2IsNull)

//eg
allEq({id:1,name:"老王",age:null}) -----> id = 1 and name = "老王" and
age is null
allEq({id:1,name:"老王",age:null}, false) ----> id = 1 and name = "老王"
```

查询方式	说明
setSqlSelect	设置 SELECT 查询字段
where	WHERE 语句，拼接 + WHERE 条件
and	AND 语句，拼接 + AND 字段=值
andNew	AND 语句，拼接 + AND (字段=值)
or	OR 语句，拼接 + OR 字段=值
orNew	OR 语句，拼接 + OR (字段=值)
eq	等于=
allEq	基于 map 内容等于=
ne	不等于<>
gt	大于>
ge	大于等于>=
lt	小于<
le	小于等于<=
like	模糊查询 LIKE
notLike	模糊查询 NOT LIKE
in	IN 查询

notIn	NOT IN 查询
isNull	NULL 值查询
isNotNull	IS NOT NULL
groupBy	分组 GROUP BY
having	HAVING 关键词
orderBy	排序 ORDER BY
orderAsc	ASC 排序 ORDER BY
orderDesc	DESC 排序 ORDER BY
exists	EXISTS 条件语句
notExists	NOT EXISTS 条件语句
between	BETWEEN 条件语句
notBetween	NOT BETWEEN 条件语句
addFilter	自由拼接 SQL
last	拼接在最后，例如：last(“LIMIT 1”)

主键策略

> 提示：
>
> 主键生成策略必须使用INPUT
>
> 支持父类定义@KeySequence子类继承使用
>
> 内置支持：
>
> DB2KeyGenerator
>
> H2KeyGenerator
>
> KingbaseKeyGenerator

> OracleKeyGenerator
>
> PostgreKeyGenerator
>
> 如果内置支持不满足要求，可实现IkeyGenerator接口来进行扩展

ep

```
@KeySequence(value = "SEQ_ORACLE_STRING_KEY", clazz = String.class)
public class YourEntity {
    @TableId(value = "ID_STR", type = IdType.INPUT)
    private String idStr;
}
```

Spring Boot配置主键策略

方式一：使用配置类

```
@Bean
public IKeyGenerator keyGenerator() {
    return new H2KeyGenerator();
}
```

方式二：通过MybatisPlusPropertiesCustomizer自定义

```
@Bean
public MybatisPlusPropertiesCustomizer plusPropertiesCustomizer(){
    List<IKeyGenerator> objects = new ArrayList<>();
    objects.add(new H2KeyGenerator());
    return properties ->
properties.getGlobalConfig().getDbConfig().setKeyGenerators(objects);
}
```

自定义ID生成器

> 自3.3.0开始，默认使用雪花算法+UUID(不含中划线)

方法	主键生成策略	主键类型	说明
nextId	ASSIGN_ID	Long, Integer, String	支持自动转换为String类型，但数值类型不支持自动转换，需精准匹配，例如返回Long，实体主键就不支持定义为Integer
nextUUID	ASSIGN_UUID	String	默认不含中划线的UUID生成

Spring Boot自定义主键生成

方式一：声明为Bean供Spring扫描注入

```
@Component
public class CustomerIdGenerator implements IdentifierGenerator {
    @Override
    public Number nextId(Object entity) {
        //可以将当前传入的class全类名来作为bizKey，或者提取参数来生成bizKey进行分布式Id调用生成
        String bizKey = entity.getClass().getName();
        //根据bizKey调用分布式ID生成
        long id = 0L;
        return id;
    }
}
```

方式二：使用配置类

```
@Bean
public IdentifierGenerator identifierGenerator(){
    return new CustomerIdGenerator();
}
```

方式三：通过MybatisPlusPropertiesCustomizer自定义

```
@Bean
public MybatisPlusPropertiesCustomizer plusPropertiesCustomizer(){
    return properties ->
properties.getGlobalConfig().setIdentifierGenerator(new
CustomerIdGenerator());
}
```

3.扩展

逻辑删除

说明

> 只对自动注入的sql起效
>
> 插入：不作限制
>
> 查找：追加where条件过滤掉已删除数据，且使用wrapper.entity生成的where条件会忽略该字段
>
> 更新：追加where条件防止更新到已删除数据，且使用wrapper.entity生成的where条件会忽略该字段
>
> 删除：转变为更新
>
> 例如：
>
> 删除：update user set deleted = 1 where id = 1 and deleted = 0;
>
> 查找：select id, name, age, deleted from user where deleted = 0;
>
> 字段类型支持说明：
>
> 支持所有数据类型(推荐使用Integer，Boolean，LocalDateTime)

> 如果数据库字段使用datetime，逻辑未删除值和已删除值支持配置为字符串null，另一个值支持配置为函数来获取 如now()
>
> 附录：
>
> 逻辑删除是为了方便数据恢复和保护数据本身价值等等等一种方案，但实际就是删除
>
> 如果需要频繁查出来看就不应使用逻辑删除，而是以一个状态去表示

编码测试

1. 数据库表添加deleted字段

```
alter table `user` add column `deleted` boolean default false
```

2. 配置application.yml(如果项目中逻辑删除默认值与逻辑未删除默认值与MP中的默认值相同，可不用配置application.xml)

```
mybatis-plus:
  global-config:
    db-config:
      # 全局逻辑删除的实体字段名(since 3.3.0，配置后可以忽略不配置@TableLogic注解)
      logic-delete-field: flag
      # 逻辑已删除值(默认为1)
      logic-delete-value: 1
      # 逻辑未删除值(默认为0)
      logic-not-delete-value: 0
```

3. 实体类字段上加上@TableLogic注解

```
@TableLogic
private Integer deleted;
```

4. 测试

```
/**
 * 逻辑删除测试
 */
@Test
public void testLogicDelete(){
    int result = userMapper.deleteById(1L);
    System.out.println(result);
}
```

通用枚举

解决了繁琐的配置，让mybatis优雅的使用枚举属性

> 自3.3.0开始，如果无需使用原生枚举，可配置默认枚举来省略扫描通用枚举配置
>
> 升级说明：
>
> 3.3.0以下版本改变了原生默认行为，升级时请将默认枚举设置为EnumOrdinalTypeHandler
>
> 影响用户：
>
> 实体中使用原生枚举
>
> 其他说明：
>
> 配置枚举包扫描的时候能提前注册使用注解枚举的缓存

声明通用枚举属性

方式一：使用@EnumValue注解枚举属性

```
package com.maxt.mybatisplus.enums;

import lombok.Getter;
```

```
    @Getter
    public enum GradeEnum {
        PRIMARY(1, "小学"),
        SECONDORY(2, "中学"),
        HIGH(3, "高中");

        GradeEnum(int code, String desc) {
            this.code = code;
            this.desc = desc;
        }
        @EnumValue
        private int code;
        private String desc;
    }
```

方式二：枚举属性，实现IEnum接口

```
    package com.maxt.mybatisplus.enums;

    import com.baomidou.mybatisplus.annotation.IEnum;

    public enum AgeEnum implements IEnum<Integer> {
        ONE(1,"一岁"),
        TWO(2, "二岁"),
        THREE(3, "三岁");

        private int value;
        private String desc;

        AgeEnum(int value, String desc) {
            this.value = value;
            this.desc = desc;
        }

        @Override
        public Integer getValue() {
```

```
            return this.value;
        }
    }
```

实体属性使用枚举类型

```
package com.maxt.mybatisplus.entity;

import com.maxt.mybatisplus.enums.AgeEnum;
import com.maxt.mybatisplus.enums.GradeEnum;
import lombok.Data;

/**
 * @Author Maxt
 * @Date 2022/3/21 上午11:22
 * @Version 1.0
 * @Description
 */
@Data
public class UserWithEnum {
    /**
     * 名字
     * 数据库字段：name
     */
    private String name;
    /**
     * 年龄，IEnum接口的枚举处理
     * 数据库字段：age
     */
    private AgeEnum age;
    /**
     * 年级，原生枚举(带{@link
com.baomidou.mybatisplus.annotation.EnumValue})
     * 数据库字段：grade
     */
    private GradeEnum grade;
```

```
}
```

配置扫描通用枚举

1. 配置application.yml

```
mybatis-plus:
  type-enums-package: com.maxt.mybatisplus.enums
```

2. 自定义配置类MybatisPlusAutoConfiguration

```
@Configuration
public class MyBatisPlusAutoConfiguration {
    @Bean
    public MybatisPlusPropertiesCustomizer
mybatisPlusPropertiesCustomizer(){
        MybatisConfiguration configuration = new MybatisConfiguration();

 configuration.setDefaultEnumTypeHandler(MybatisEnumTypeHandler.class);
        return properties ->
properties.setConfiguration(configuration).getGlobalConfig().setBanner(f
alse);
    }
}
```

如何序列化枚举值为数据库存储值

Jackson

重写toString方法

springboot

```
@Bean
public Jackson2ObjectMapperBuilderCustomizer customizer(){
    return builder ->
builder.featuresToEnable(SerializationFeature.WRITE_ENUMS_USING_TO_STRIN
G);
}
```

jackson

```
ObjectMapper objectMapper = new ObjectMapper();
objectMapper.configure(SerializationFeature.WRITE_ENUMS_USING_TO_STRING,
true);
```

以上两种方式任选其一，然后在枚举中复写toString方法即可

注解处理

```
package com.maxt.mybatisplus.enums;

import com.baomidou.mybatisplus.annotation.EnumValue;
import lombok.Getter;

@Getter
public enum GradeEnum {
    PRIMARY(1, "小学"),
    SECONDORY(2, "中学"),
    HIGH(3, "高中");

    GradeEnum(int code, String desc) {
        this.code = code;
        this.desc = desc;
    }
    @EnumValue
    //标记响应json值，需要添加jackson依赖
    @JsonValue
    private int code;
```

```
    private String desc;
}
```

FastJson

重写toString方法

全局处理方式

```
FastJsonConfig config = new FastJsonConfig();
config.setSerializerFeatures(SerializerFeature.WriteEnumUsingToString);
```

局部处理方式

```
@JSONField(serialzeFeatures= SerializerFeature.WriteEnumUsingToString)
private GradeEnum grade;
```

以上两种方式任选其一，然后在枚举中复写toString方法即可

字段类型处理器

类型处理器用于JavaType与JdbcType之间的转换，用于PreparedStatement设置参数值和从ResultSet或CallableStatement中取出一个值，以下是mybatis-plus内置常用类型处理器如何通过TableField注解快速注入到mybatis容器中

```
@Data
public class OtherInfo {
    /**
     * 性别
     */
    private String sex;
    /**
     * 居住城市
     */
    private String city;
}
```

```java
@Data
@Accessors(chain = true)
@TableName(value = "user", autoResultMap = true)
public class UserWithJson {
    private Long id;

    /**
     * 注意：必须开启映射注解
     * @TableName(autoResultMap=true)
     * 以下两种类型处理器，二选一 也可以同时存在
     * 注意：选择对应的JSON解析依赖包
     */
    @TableField(typeHandler = JacksonTypeHandler.class)
    //@TableField(typeHandler = FastjsonTypeHandler.class)
    private OtherInfo otherInfo;
}
```

该注解对应XML中写法为

```xml
<result column="other_info" jdbcType="VARCHAR" property="otherInfo"
typeHandler="com.baomidou.mybatisplus.extension.handlers.JacksonTypeHand
ler"></result>
```

自动填充功能

> 原理：
>
> 实现元对象处理器接口：
> com.baomidou.mybatisplus.core.handlers.MetaObjectHandler
>
> 注解填充字段@TableField(fill=FieldFill.INSERT)

1. 数据库表添加create_time与update_time字段

```
ALTER TABLE `user` ADD COLUMN `create_time` DATETIME NOT NULL DEFAULT
'2022-03-21 14:30:00'
ALTER TABLE `user` ADD COLUMN `update_time` DATETIME DEFAULT NULL
```

2. 编码测试

```
package com.maxt.mybatisplus.entity;

import com.baomidou.mybatisplus.annotation.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

/**
 * @Author Maxt
 * @Date 2022/3/19 下午7:31
 * @Version 1.0
 * @Description 用户类
 */
//set、get方法
@Data
//所有参数构造函数
@AllArgsConstructor
//无参构造函数
@NoArgsConstructor
@TableName("user")
public class User {
    @TableId(type = IdType.ASSIGN_ID)
    private Long id;
    @TableField("name")
    private String name;
    private Integer age;
    private String email;
    @TableLogic
```

```
    private Integer deleted;
    @TableField(fill = FieldFill.INSERT)
    private LocalDateTime createTime;
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private LocalDateTime updateTime;
}
```

```
package com.maxt.mybatisplus.config;

import com.baomidou.mybatisplus.core.handlers.MetaObjectHandler;
import lombok.extern.slf4j.Slf4j;
import org.apache.ibatis.reflection.MetaObject;
import org.springframework.stereotype.Component;

import java.time.LocalDateTime;

/**
 * @Author Maxt
 * @Date 2022/3/21 下午2:52
 * @Version 1.0
 * @Description
 */
@Slf4j
@Component
public class MyMetaObjectHandler implements MetaObjectHandler {
    @Override
    public void insertFill(MetaObject metaObject) {
        log.info("start insert fill...");
        //起始版本 3.3.0(推荐使用)
        this.strictInsertFill(metaObject, "createTime",
LocalDateTime.class, LocalDateTime.now());
        this.strictInsertFill(metaObject, "updateTime",
LocalDateTime.class, LocalDateTime.now());
        //或者
        //起始版本 3.3.3(推荐)
```

```
        //this.strictInsertFill(metaObject, "createTime", () ->
LocalDateTime.now(), LocalDateTime.class);
        //或者
        //3.3.0 该方法有bug
        //this.fillStrategy(metaObject, "createTime",
LocalDateTime.now());
    }

    @Override
    public void updateFill(MetaObject metaObject) {
        log.info("start update fill...");
        //起始版本 3.3.0(推荐使用)
        this.strictInsertFill(metaObject, "updateTime",
LocalDateTime.class, LocalDateTime.now());
        //或者
        //起始版本 3.3.3(推荐)
        //this.strictInsertFill(metaObject, "updateTime", () ->
LocalDateTime.now(), LocalDateTime.class);
        //或者
        //3.3.0 该方法有bug
        //this.fillStrategy(metaObject, "updateTime",
LocalDateTime.now());
    }
}
```

```
    /**
     * 插入操作
     * 插入时根据主键策略来判断时候需要插入主键值,
     * 默认为ASSIGN_ID(分布式ID, 根据雪花算法生成)
     */
    @Test
    public void testInsert(){
        User user = new User();
        user.setName("Lucy");
        user.setAge(20);
        user.setEmail("1234@163.com");
```

```
        //insert为插入数据影响的数据行数
        int insert = userMapper.insert(user);
        System.out.println(insert);
    }

    /**
     * 根据id更新操作
     */
    @Test
    public void testUpdate(){
        User user = new User();
        user.setId(1L);
        user.setName("LucyUpdateTime");
        user.setAge(20);
        user.setEmail("1234@163.com");
        int count = userMapper.updateById(user);
        System.out.println(count);
    }
```

> 注意事项：
>
> 1. 填充原理是直接给entity的属性设置值
> 2. 注解则是指定该属性在对应情况下必有值，如果无值则入库会是null
> 3. MetaObjectHandler提供的默认方法的策略均为：如果属性有值则不覆盖，如果填充值为null则不填充
> 4. 字段必须声明TableField注解，属性fill选择对应策略，该声明告知Mybatis-Plus需要预留注入SQL字段
> 5. 填充处理器MyMetaObjectHandler在Spring Boot中需要声明@Componet或@Bean注入
> 6. 要想根据注解FieldFill.xxx和字段名以及字段类型来区分必须使用父类的strictInsertFill或者strictUpdateFill
> 7. 不需要根据任何来区分可以使用父类的fiiStrategy方法
> 8. update(T t, Wrapper updateWrapper)时t不能为空，否则自动填充失败

```
public enum FieldFill {
```

```
        /**
         * 默认不处理
         */
        DEFAULT,
        /**
         * 插入填充字段
         */
        INSERT,
        /**
         * 更新填充字段
         */
        UPDATE,
        /**
         * 插入和更新填充字段
         */
        INSERT_UPDATE
    }
```

SQL注入器

> 注入器配置
>
> 全局配置sqlInjector用于注入ISqlInjector接口的子类，实现自定义方法注入

SQL自动注入器接口ISqlInjector

```
public interface ISqlInjector {

    /**
     * <p>
     * 检查SQL是否注入(已经注入过不再注入)
     * </p>
     *
     * @param builderAssistant mapper 信息
     * @param mapperClass      mapper 接口的 class 对象
     */
    void inspectInject(MapperBuilderAssistant builderAssistant, Class<?>
mapperClass);
}
```

自定义自己的通用方法可以实现接口ISqlInjector也可以继承抽象类AbstractSqlInjector注入通用方法sql语句，然后继承BaseMapper添加自定义方法，全局配置sqlInjector注入MP会自动将类所有方法注入到mybatis容器中

编程测试

```
package com.maxt.mybatisplus.config.sql.injector;

import com.baomidou.mybatisplus.core.injector.AbstractMethod;
import com.baomidou.mybatisplus.core.injector.DefaultSqlInjector;
import com.baomidou.mybatisplus.core.metadata.TableInfo;

import java.util.List;

/**
 * @Author Maxt
 * @Date 2022/3/21 下午4:50
 * @Version 1.0
 * @Description
 */
public class MyLogicSqlInjector extends DefaultSqlInjector {
```

```
    /**
     * 如果只需要增加方法，保留MP自带方法
     * 可以使用super.getMethodList()再add
     * @return
     */
    @Override
    public List<AbstractMethod> getMethodList(Class<?> mapperClass,
TableInfo tableInfo) {
        List<AbstractMethod> methodList =
super.getMethodList(mapperClass, tableInfo);
        methodList.add(new DeleteAll());
        methodList.add(new MyInsertAll());
        methodList.add(new MysqlInsertAllBatch());
        return methodList;
    }
}
```

```
package com.maxt.mybatisplus.config.sql.injector;

import com.baomidou.mybatisplus.core.injector.AbstractMethod;
import com.baomidou.mybatisplus.core.metadata.TableInfo;
import org.apache.ibatis.mapping.MappedStatement;
import org.apache.ibatis.mapping.SqlSource;

/**
 * @Author Maxt
 * @Date 2022/3/21 下午4:55
 * @Version 1.0
 * @Description
 */
public class DeleteAll extends AbstractMethod {
    @Override
    public MappedStatement injectMappedStatement(Class<?> mapperClass,
Class<?> modelClass, TableInfo tableInfo) {
        /* 执行SQL，动态SQL参考类 SqlMethod*/
```

```
        String sql = "delete from " + tableInfo.getTableName();
        /*mapper接口方法名一致*/
        String method = "deleteAll";
        SqlSource sqlSource =
languageDriver.createSqlSource(configuration, sql, modelClass);
        return this.addDeleteMappedStatement(mapperClass, method,
sqlSource);
    }
}

package com.maxt.mybatisplus.config.sql.injector;

import com.baomidou.mybatisplus.core.injector.AbstractMethod;
import com.baomidou.mybatisplus.core.metadata.TableInfo;
import lombok.extern.slf4j.Slf4j;
import org.apache.ibatis.executor.keygen.NoKeyGenerator;
import org.apache.ibatis.mapping.MappedStatement;
import org.apache.ibatis.mapping.SqlSource;

/**
 * @Author Maxt
 * @Date 2022/3/21 下午5:03
 * @Version 1.0
 * @Description
 */
@Slf4j
public class MyInsertAll extends AbstractMethod {
    @Override
    public MappedStatement injectMappedStatement(Class<?> mapperClass,
Class<?> modelClass, TableInfo tableInfo) {
        String sql = "insert into %s %s values %s";
        StringBuffer fieldSql = new StringBuffer();
        fieldSql.append(tableInfo.getKeyColumn()).append(",");
        StringBuffer valueSql = new StringBuffer();
        valueSql.append("#
{").append(tableInfo.getKeyProperty()).append("},");
```

```
        tableInfo.getFieldList().forEach(x ->{
            fieldSql.append(x.getColumn()).append(",");
            valueSql.append("#{").append(x.getProperty()).append("},");
        });
        fieldSql.delete(fieldSql.length()-1, fieldSql.length());
        fieldSql.insert(0, "(");
        fieldSql.append(")");
        valueSql.insert(0, "(");
        valueSql.delete(valueSql.length()-1, valueSql.length());
        valueSql.append(")");
        log.info("fieldSql:"+fieldSql.toString());
        log.info("valueSql:"+valueSql.toString());
        SqlSource sqlSource =
languageDriver.createSqlSource(configuration, String.format(sql,
tableInfo.getTableName(), fieldSql.toString(), valueSql.toString()),
modelClass);
        return this.addInsertMappedStatement(mapperClass, modelClass,
"myInsertAll", sqlSource, new NoKeyGenerator(), null, null);
    }
}


package com.maxt.mybatisplus.config.sql.injector;

import com.baomidou.mybatisplus.core.injector.AbstractMethod;
import com.baomidou.mybatisplus.core.metadata.TableInfo;
import org.apache.ibatis.executor.keygen.NoKeyGenerator;
import org.apache.ibatis.mapping.MappedStatement;
import org.apache.ibatis.mapping.SqlSource;

/**
 * @Author Maxt
 * @Date 2022/3/21 下午5:20
 * @Version 1.0
 * @Description
 */
```

```
public class MysqlInsertAllBatch extends AbstractMethod {
    @Override
    public MappedStatement injectMappedStatement(Class<?> mapperClass,
Class<?> modelClass, TableInfo tableInfo) {
        final String sql = "<script>insert into %s %s values
%s</script>";
        final String fieldSql = prepareFieldSql(tableInfo);
        final String valueSql =
prepareValuesSqlForMysqlBatch(tableInfo);
        final String sqlResult = String.format(sql,
tableInfo.getTableName(), fieldSql, valueSql);
        SqlSource sqlSource =
languageDriver.createSqlSource(configuration, sqlResult, modelClass);
        return this.addInsertMappedStatement(mapperClass, modelClass,
"mysqlInsertAllBatch", sqlSource, new NoKeyGenerator(), null, null);
    }

    private String prepareFieldSql(TableInfo tableInfo) {
        StringBuilder fieldSql = new StringBuilder();
        fieldSql.append(tableInfo.getKeyColumn()).append(",");
        tableInfo.getFieldList().forEach(x -> {
            fieldSql.append(x.getColumn()).append(",");
        });
        fieldSql.delete(fieldSql.length() - 1, fieldSql.length());
        fieldSql.insert(0, "(");
        fieldSql.append(")");
        return fieldSql.toString();
    }

    private String prepareValuesSqlForMysqlBatch(TableInfo tableInfo) {
        final StringBuilder valueSql = new StringBuilder();
        valueSql.append("<foreach collection=\"list\" item=\"item\"
index=\"index\" open=\"(\" separator=\"),(\" close=\")\">");
        valueSql.append("#
{item.").append(tableInfo.getKeyProperty()).append("},");
```

```
        tableInfo.getFieldList().forEach(x -> valueSql.append("#
{item.").append(x.getProperty()).append("},"));
        valueSql.delete(valueSql.length() - 1, valueSql.length());
        valueSql.append("</foreach>");
        return valueSql.toString();
    }
}
```

```
package com.maxt.mybatisplus.config;

import
com.baomidou.mybatisplus.autoconfigure.MybatisPlusPropertiesCustomizer;
import com.baomidou.mybatisplus.core.MybatisConfiguration;
import com.baomidou.mybatisplus.core.handlers.MybatisEnumTypeHandler;
import com.maxt.mybatisplus.config.sql.injector.MyLogicSqlInjector;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * @Author Maxt
 * @Date 2022/3/21 上午11:37
 * @Version 1.0
 * @Description
 */
@Configuration
public class MyBatisPlusAutoConfiguration {
    @Bean
    public MyLogicSqlInjector myLogicSqlInjector(){
        return new MyLogicSqlInjector();
    }
}
```

```
package com.maxt.mybatisplus.test;
```

```java
import com.maxt.mybatisplus.entity.User;
import com.maxt.mybatisplus.mapper.MyBaseMapper;
import com.maxt.mybatisplus.mapper.UserMapper;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.ArrayList;
import java.util.List;

/**
 * @Author Maxt
 * @Date 2022/3/21 下午5:40
 * @Version 1.0
 * @Description
 */
@SpringBootTest
public class MySqlInjectorTest {

    @Autowired
    UserMapper userMapper;

    /**
     * 删除所有
     */
    @Test
    public void testDeleteAll(){
        userMapper.deleteAll();
    }

    /**
     * 批量插入所有
     */
    @Test
    public void testInsertAllBatch(){
```

```
        List<User> list = new ArrayList<>();
        User user1 = new User();
        user1.setId(100L);
        user1.setName("aa");
        user1.setAge(20);
        user1.setEmail("11@qq.com");
        list.add(user1);
        User user2 = new User();
        user1.setId(101L);
        user1.setName("bb");
        user1.setAge(20);
        user1.setEmail("22@qq.com");
        list.add(user2);
        int count = userMapper.mysqlInsertAllBatch(list);
        System.out.println(count);
    }

    @Test
    void myInsertAll() {
        long id = 1008888L;
        User user = new User();
        user.setId(id);
        user.setName("aa");
        user.setAge(20);
        user.setEmail("11@qq.com");
        int count = userMapper.myInsertAll(user);
        System.out.println(count);
    }
}
```

```
package com.maxt.mybatisplus.entity;

import com.baomidou.mybatisplus.annotation.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```java
import lombok.experimental.Accessors;

import java.time.LocalDateTime;

/**
 * @Author Maxt
 * @Date 2022/3/19 下午7:31
 * @Version 1.0
 * @Description 用户类
 */
//set、get方法
@Data
//所有参数构造函数
@AllArgsConstructor
//无参构造函数
@NoArgsConstructor
@TableName("user")
@Accessors(chain = true)
public class User {
    @TableId(type = IdType.ASSIGN_ID)
    private Long id;
    @TableField("name")
    private String name;
    private Integer age;
    private String email;
    @TableLogic
    private Integer deleted;
    @TableField(fill = FieldFill.INSERT)
    private LocalDateTime createTime;
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private LocalDateTime updateTime;
}
```

执行SQL分析打印

> 该功能依赖p6spy组件，完美的输出打印SQL及执行时长 3.1.0以上版本

1. 添加maven依赖

```
<!--执行sql分析打印-->
<dependency>
    <groupId>p6spy</groupId>
    <artifactId>p6spy</artifactId>
    <version>3.9.1</version>
</dependency>
```

2. 配置application.yml

```
spring:
  datasource:
    #driver-class-name: com.mysql.cj.jdbc.Driver
    driver-class-name: com.p6spy.engine.spy.P6SpyDriver
    url: jdbc:p6spy:mysql://localhost:3306/mybatis-plus?
useUnicode=true&characterEncoding=utf8&useSSL=false&allowPublicKeyRetrie
val=true&serverTimezone=GMT%2B8
    #url: jdbc:mysql://localhost:3306/mybatis-plus?
useUnicode=true&characterEncoding=utf8&useSSL=false&allowPublicKeyRetrie
val=true&serverTimezone=GMT%2B8
    username: root
    password: root1234
```

3. 配置spy.properties

```
#3.2.1以上使用
modulelist=com.baomidou.mybatisplus.extension.p6spy.MybatisPlusLogFactor
y,com.p6spy.engine.outage.P6OutageFactory
#3.2.1以下使用或者不配置
```

```
#modulelist=com.p6spy.engine.logging.P6LogFactory,com.p6spy.engine.outag
e.P6OutageFactory
# 自定义日志打印
logMessageFormat=com.baomidou.mybatisplus.extension.p6spy.P6SpyLogger
#日志输出到控制台
appender=com.baomidou.mybatisplus.extension.p6spy.StdoutLogger
# 使用日志系统记录 sql
#appender=com.p6spy.engine.spy.appender.Slf4JLogger
# 设置 p6spy driver 代理
deregisterdrivers=true
# 取消JDBC URL前缀
useprefix=true
# 配置记录 Log 例外,可去掉的结果集有
error,info,batch,debug,statement,commit,rollback,result,resultset.
excludecategories=info,debug,result,commit,resultset
# 日期格式
dateformat=yyyy-MM-dd HH:mm:ss
# 实际驱动可多个
#driverlist=org.h2.Driver
# 是否开启慢SQL记录
outagedetection=true
# 慢SQL记录标准 2 秒
outagedetectioninterval=2
```

> 注意：
>
> 1. Driver-class-name为p6spy提供的驱动类
> 2. url前缀为jdbc:p6sy跟着冒号为对应数据库连接地址
> 3. 打印出sql为null，在excludecategories增加commit
> 4. 批量操作不打印sql，去除excludecategories的batch
> 5. 批量重复打印问题请使用MybatisPlusLogFactory(3.2.1新增)
> 6. 该插件有性能损耗，不建议生产环境使用

插件

插件主体

> 注意
>
> 版本要求：3.4.0版本以上

MybatisPlusInterceptor

该插件是核心插件，目前代理了Executor#query和Executor#update和StatementHandler#prepare方法

属性

```
private List<InnerInterceptor> interceptors = new ArrayList<>();
```

InnerInterceptor

提供的插件都将基于此接口来实现功能，目前已有的功能

1. 自动分页：PaginationInnerInterceptor
2. 多租户：TenantLineInnerInterceptor
3. 动态表名：DynamicTableNameInnerInterceptor
4. 乐观锁：OptimisticLockerInnerInterceptor
5. sql性能规范：IllegalSQLInnerInterceptor
6. 防止全表更新与删除：BlockAttackInnerInterceptor

> 注意：
>
> 使用多个功能需要注意顺序关系，建议使用如下顺序
>
> 1. 多租户，动态表名
> 2. 分页，乐观锁
> 3. sql性能规范，防止全表更新与删除

> 总结：对sql进行单次改造的优先放入，不对sql进行改造的最后放入

分页插件 PaginationInnerInterceptor

Spring Boot

```
package com.maxt.mybatisplus.config;

import com.baomidou.mybatisplus.annotation.DbType;
import
com.baomidou.mybatisplus.autoconfigure.MybatisPlusPropertiesCustomizer;
import com.baomidou.mybatisplus.core.MybatisConfiguration;
import com.baomidou.mybatisplus.core.handlers.MybatisEnumTypeHandler;
import
com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
import
com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerIntercep
tor;
import com.maxt.mybatisplus.config.sql.injector.MyLogicSqlInjector;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * @Author Maxt
 * @Date 2022/3/21 上午11:37
 * @Version 1.0
 * @Description
 */
@Configuration
public class MyBatisPlusAutoConfiguration {

    /**
     * 新的分页插件，一缓和二缓遵循mybatis的规则，需要设置
MybatisConfiguration#useDeprecatedExecutor = false
     * 避免缓存出现问题，(该属性会在旧插件移除后一同移除)
     * @return
```

```
     */
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor(){
        MybatisPlusInterceptor interceptor = new
MybatisPlusInterceptor();
        interceptor.addInnerInterceptor(new
PaginationInnerInterceptor(DbType.MYSQL));
        return interceptor;
    }
}
```

```
    /**
     * 测试分页
     */
    @Test
    public void testSelectByPage(){
        Page<User> page = new Page(1, 3);
        Page<User> userPage = userMapper.selectPage(page, null);
        ////总页数
        long pages = userPage.getPages();
        //当前页
        long current = userPage.getCurrent();
        //查询数据集合
        List<User> records = userPage.getRecords();
        //总记录数
        long total = userPage.getTotal();
        //下一页
        boolean hasNext = userPage.hasNext();
        //上一页
        boolean hasPrevious = userPage.hasPrevious();
        System.out.println(pages);
        System.out.println(current);
        System.out.println(records);
        System.out.println(total);
        System.out.println(hasNext);
        System.out.println(hasPrevious);
```

```
    }
```

乐观锁插件 OptimisticLockerInnerInterceptor

当要更新一条记录的时候，希望这条记录没有被被人更新

> 乐观锁实现方式：
>
> 1. 取出记录时，获取当前version
> 2. 更新时，带上这个version
> 3. 执行更新时，set version = new Version where version = old Version
> 4. 如果version不对，就更新失败

1. 配置插件

```
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor(){
        MybatisPlusInterceptor interceptor = new
MybatisPlusInterceptor();
        interceptor.addInnerInterceptor(new
PaginationInnerInterceptor(DbType.MYSQL));
        //乐观锁
        interceptor.addInnerInterceptor(new
OptimisticLockerInnerInterceptor());
        return interceptor;
    }
```

2. 在实体类字段上加上@Version

```
@Version
private Integer version;
```

> 说明：

1. 支持的数据类型只有：int，Integer，long，Long，Date，TimeStamp，LocalDateTime
2. 整数类型下new Version = old Version + 1
3. new Version会回写到entity中
4. 仅支持updateById(id)与update(entity, wrapper)方法
5. 在update(entity, wrapper)方法下，wrapper不能复用

多租户插件 TenantLineInnerInterceptor

```
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor(){
        MybatisPlusInterceptor interceptor = new
MybatisPlusInterceptor();
        //多租户
        interceptor.addInnerInterceptor(new
TenantLineInnerInterceptor(new TenantLineHandler() {
            @Override
            public Expression getTenantId() {
                return new LongValue();
            }
            //这是default方法，默认返回false表示所有表都需要拼多租户条件
            @Override
            public boolean ignoreTable(String tableName) {
                return !"user".equalsIgnoreCase(tableName);
            }
        }));
        //如果使用了分页插件，必须先add租户插件在add分页插件
        interceptor.addInnerInterceptor(new
PaginationInnerInterceptor(DbType.MYSQL));
        //乐观锁
        interceptor.addInnerInterceptor(new
OptimisticLockerInnerInterceptor());
        return interceptor;
    }
```

```java
public interface TenantLineHandler {

    /**
     * 获取租户 ID 值表达式，只支持单个 ID 值
     * <p>
     *
     * @return 租户 ID 值表达式
     */
    Expression getTenantId();

    /**
     * 获取租户字段名
     * <p>
     * 默认字段名叫: tenant_id
     *
     * @return 租户字段名
     */
    default String getTenantIdColumn() {
        return "tenant_id";
    }

    /**
     * 根据表名判断是否忽略拼接多租户条件
     * <p>
     * 默认都要进行解析并拼接多租户条件
     *
     * @param tableName 表名
     * @return 是否忽略, true:表示忽略, false:需要解析并拼接多租户条件
     */
    default boolean ignoreTable(String tableName) {
        return false;
    }
}
```

> 说明：

> 多租户 != 权限过滤，不要乱用，租户之间时完全隔离的
>
> 启用多租户后所有执行的method的sql都会进行处理
>
> 自写的sql请按规范书写(sql涉及到多个表的每个表都要给别名，特别是inner join的要写标准的inner join)

防止全表更新与删除插件 BlockAttackInnerInterceptor

> 针对update和delete语句
>
> 作用：阻止恶意的全表更新删除

配置BlockAttackInnerInterceptor拦截器

```
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor(){
        MybatisPlusInterceptor interceptor = new
MybatisPlusInterceptor();
        //多租户
        interceptor.addInnerInterceptor(new
TenantLineInnerInterceptor(new TenantLineHandler() {
            @Override
            public Expression getTenantId() {
                return new LongValue();
            }
            //这是default方法，默认返回false表示所有表都需要拼多租户条件
            @Override
            public boolean ignoreTable(String tableName) {
                return !"user".equalsIgnoreCase(tableName);
            }
        }));
        //如果使用了分页插件，必须先add租户插件在add分页插件
        interceptor.addInnerInterceptor(new
PaginationInnerInterceptor(DbType.MYSQL));
```

```
        //乐观锁
        interceptor.addInnerInterceptor(new
OptimisticLockerInnerInterceptor());
        interceptor.addInnerInterceptor(new
BlockAttackInnerInterceptor());
        return interceptor;
    }
```

```
package com.maxt.mybatisplus.test;

import
com.baomidou.mybatisplus.core.conditions.update.LambdaUpdateWrapper;
import com.maxt.mybatisplus.entity.User;
import com.maxt.mybatisplus.mapper.UserMapper;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

//全表更新
@SpringBootTest
public class QueryWrapperTest {

    @Autowired
    private UserMapper userMapper;

    /**
     + SQL：UPDATE user  SET name=?,email=?;
     */
    @Test
    public void testAllUpdate() {
        User user = new User();
        user.setId(999L);
        user.setName("custom_name");
        user.setEmail("xxx@mail.com");
```

```
        //
com.baomidou.mybatisplus.core.exceptions.MybatisPlusException:
Prohibition of table update operation
        userMapper.update(user, null);
    }

    /**
     + SQL: UPDATE user  SET name=?, email=? WHERE id = ?;
     */
    @Test
    public void testSomeUpdate() {
        LambdaUpdateWrapper<User> wrapper = new LambdaUpdateWrapper<>();
        wrapper.eq(User::getId, 1);
        User user = new User();
        user.setId(10L);
        user.setName("custom_name");
        user.setEmail("xxx@mail.com");
        userMapper.update(user, wrapper);
    }
}
```

动态表名插件 DynamicTableNameInnerInterceptor

> 注意事项:
>
> 原理为解析替换设定表名处理器的返回表名，表名建议可以定义复杂一些避免误替换
>
> 例如：真实表名为user设定为mp_dt_userc处理器替换为user_2019等

```
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor(){
        MybatisPlusInterceptor interceptor = new
MybatisPlusInterceptor();
        //多租户
```

```
        /*interceptor.addInnerInterceptor(new
TenantLineInnerInterceptor(new TenantLineHandler() {
            @Override
            public Expression getTenantId() {
                return new LongValue();
            }
            //这是default方法，默认返回false表示所有表都需要拼多租户条件
            @Override
            public boolean ignoreTable(String tableName) {
                return !"user".equalsIgnoreCase(tableName);
            }
        }));*/
        //如果使用了分页插件，必须先add租户插件在add分页插件
        interceptor.addInnerInterceptor(new
PaginationInnerInterceptor(DbType.MYSQL));
        //乐观锁
        interceptor.addInnerInterceptor(new
OptimisticLockerInnerInterceptor());
        interceptor.addInnerInterceptor(new
BlockAttackInnerInterceptor());
        DynamicTableNameInnerInterceptor
dynamicTableNameInnerInterceptor = new
DynamicTableNameInnerInterceptor();
        dynamicTableNameInnerInterceptor.setTableNameHandler((sql,
tableName) -> {
            // 获取参数方法
            Map<String, Object> paramMap =
RequestDataHelper.getRequestData();
            paramMap.forEach((k, v) -> System.err.println(k + "----" +
v));

            String year = "_2018";
            int random = new Random().nextInt(10);
            if (random % 2 == 1) {
                year = "_2019";
            }
```

```
            return tableName + year;
        });

 interceptor.addInnerInterceptor(dynamicTableNameInnerInterceptor);
        return interceptor;
    }
```

```
package com.maxt.mybatisplus.config;

import com.baomidou.mybatisplus.core.toolkit.CollectionUtils;

import java.util.Map;

/**
 * @Author Maxt
 * @Date 2022/3/21 下午11:07
 * @Version 1.0
 * @Description  请求参数传递辅助类
 */
public class RequestDataHelper {
    /**
     * 请求参数存取
     */
    private static final ThreadLocal<Map<String, Object>> REQUEST_DATA =
new ThreadLocal<>();

    /**
     * 设置请求参数
     *
     * @param requestData 请求参数 MAP 对象
     */
    public static void setRequestData(Map<String, Object> requestData) {
        REQUEST_DATA.set(requestData);
    }

    /**
```

```
     * 获取请求参数
     *
     * @param param 请求参数
     * @return 请求参数 MAP 对象
     */
    public static <T> T getRequestData(String param) {
        Map<String, Object> dataMap = getRequestData();
        if (CollectionUtils.isNotEmpty(dataMap)) {
            return (T) dataMap.get(param);
        }
        return null;
    }

    /**
     * 获取请求参数
     *
     * @return 请求参数 MAP 对象
     */
    public static Map<String, Object> getRequestData() {
        return REQUEST_DATA.get();
    }
}
```

```
package com.maxt.mybatisplus.test;

import com.maxt.mybatisplus.config.RequestDataHelper;
import com.maxt.mybatisplus.entity.User;
import com.maxt.mybatisplus.mapper.UserMapper;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.HashMap;

/**
 * @Author Maxt
```

```
 * @Date 2022/3/21 下午11:09
 * @Version 1.0
 * @Description
 */
@SpringBootTest
class DynamicTableNameTest {
    @Autowired
    private UserMapper userMapper;

    @Test
    void test() {
        RequestDataHelper.setRequestData(new HashMap<String, Object>()
{{
            put("id", 123);
            put("hello", "tomcat");
            put("name", "汤姆凯特");
        }});
        // 观察打印 SQL 目前随机访问 user_2018  user_2019 表
        for (int i = 0; i < 6; i++) {
            User user = userMapper.selectById(1);
            System.err.println(user.getName());
        }
    }
}
```